

183

RETI DI CALCOLATORI

LARRY L. PETERSON
BRUCE S. DAVIE

EDIZIONE ITALIANA A CURA
DI MARCELLO DALPASSO

APOGEO

Sommario

Reti di calcolatori

Titolo originale:

Computer Networks: A Systems Approach

Autori:

Larry L. Peterson, Bruce S. Davie

Original English language edition by Morgan Kaufmann Publishers
an Imprint of Elsevier Science
Copyright © 2003 – Elsevier Science

Copyright © 2004 – APOGEO
Via Natale Battaglia, 12 – 20127 Milano (Italy)
Telefono: 02-28910277 (5 linee r.a.) – Telefax: 02-26116334
Email education@apogeonline.com
U.R.L. <http://www.apogeonline.com>

ISBN 88-503-2158-9

Traduzione e revisione: Marcello Dalpasso

Impaginazione elettronica: Grafica Editoriale – Vimercate

Editor: Alberto Kratter Thaler

Copertina e progetto grafico: Enrico Marcandalli

Responsabile di produzione: Vitiano Zaini

Tutti i diritti sono riservati a norma di legge e a norma delle convenzioni internazionali. Nessuna parte di questo libro può essere riprodotta con sistemi elettronici, meccanici o altri, senza l'autorizzazione scritta dell'Editore.

Nomi e marchi citati nel testo sono generalmente depositati o registrati dalle rispettive case produttrici.

<i>Presentazione della terza edizione</i>	<i>xiii</i>
<i>Presentazione della prima edizione</i>	<i>xv</i>
<i>Presentazione dell'edizione italiana</i>	<i>xvii</i>
<i>Prefazione</i>	<i>xix</i>
Capitolo I	<i>I</i>
<i>Principi fondamentali</i>	
Problema Costruire una rete	1
1.1 Applicazioni	2
1.2 Requisiti	3
1.2.1 Connessione	5
1.2.2 Condivisione di risorse efficiente dal punto di vista dei costi	8
1.2.3 Supporto di servizi comuni	12
1.3 Architettura di rete	16
1.3.1 Stratificazione e protocolli	16
1.3.2 L'architettura OSI	22
1.3.3 L'architettura Internet	23
1.4 Implementazione di software di rete	26
1.4.1 Interfaccia per la programmazione di applicazioni (socket)	26
1.4.2 Un esempio di applicazione	29
1.4.3 Problemi nell'implementazione di protocolli	31
1.5 Prestazioni	35
1.5.1 Ampiezza di banda e latenza	35
1.5.2 Prodotto riardo x ampiezza di banda	40

1.5.3 Reti ad alta velocità	41
1.5.4 Prestazioni richieste dalle applicazioni	43
1.6 Riepilogo	44
Problema aperto La rete pervasiva	45
Ulteriori letture	46
Esercizi	48

Capitolo 2

<i>Reti a connessione diretta</i>	55
Problema Connettere fisicamente i calcolatori	55
2.1 Elementi hardware elementari	56
2.1.1 Nodi	56
2.1.2 Linee di collegamento	58
2.2 Codifica (NRZ, NRZI, Manchester, 4B/5B)	64
2.3 Tramatura (framing)	67
2.3.1 Protocolli orientati ai byte (BISYNC, PPP, DDCMP)	69
2.3.2 Protocolli orientati ai bit (HDLC)	70
2.3.3 Framing basato sul clock (SONET)	71
2.4 Rilevazione d'errore	75
2.4.1 Parità bidimensionale	76
2.4.2 Algoritmo di checksum di Internet	77
2.4.3 Verifica di ridondanza ciclica (CRC)	78
2.5 Trasmissione affidabile	83
2.5.1 Stop-and-wait	84
2.5.2 Sliding window	86
2.5.3 Canali logici concorrenti	95
2.6 Ethernet (802.3)	96
2.6.1 Proprietà fisiche	96
2.6.2 Protocollo di accesso	98
2.6.3 L'esperienza di Ethernet	103
2.7 Reti token ring (802.5 e FDDI)	103
2.7.1 Proprietà fisiche	104
2.7.2 Controllo di accesso al mezzo in reti token ring	106
2.7.3 Gestione di una rete token ring	108
2.7.4 Formato del frame	109
2.7.5 FDDI	110
2.8 Wireless (802.11)	114
2.8.1 Proprietà fisiche	114
2.8.2 Evitare le collisioni	115
2.8.3 Sistema di distribuzione	116
2.8.4 Formato del frame	118
2.9 Adattatori di rete	119
2.9.1 Componenti	119
2.9.2 Punto di vista dell'host	120
2.9.3 Collo di bottiglia nella memoria	125
2.10 Riepilogo	127

Problema aperto Fa parte dell'hardware?	128
Ulteriori letture	129
Esercizi	130

Capitolo 3

<i>Commutazione di pacchetto</i>	143
Problema Non tutte le reti sono a connessione diretta	143
3.1 Commutazione e inoltro	144
3.1.1 Datagrammi	146
3.1.2 Comutazione di circuito virtuale	148
3.1.3 Instradamento dalla sorgente (source routing)	154
3.2 Comutatori per LAN e bridge	156
3.2.1 Bridge ad apprendimento (learning bridge)	158
3.2.2 Algoritmo ad albero di copertura (spanning tree)	161
3.2.3 Broadcast e multicast	165
3.2.4 Limiti dei bridge	166
3.3 Commutazione di celle (ATM)	167
3.3.1 Celle	168
3.3.2 Segmentazione e ricostruzione	173
3.3.3 Percorsi virtuali	178
3.3.4 Strati fisici per ATM	178
3.3.5 ATM in una rete locale	179
3.4 Implementazione e prestazioni	184
3.4.1 Porte	185
3.4.2 Matrici di commutazione	189
3.5 Riepilogo	192
Problema aperto Il futuro di ATM	193
Ulteriori letture	194
Esercizi	195

Capitolo 4

<i>Interconnessione di reti</i>	203
Problema Non esiste un'unica rete	203
4.1 Semplice interconnessione di reti (IP)	204
4.1.1 Cos'è una internetwork?	204
4.1.2 Modello di servizio	206
4.1.3 Indirizzi globali	217
4.1.4 Inoltro di datagrammi in IP	219
4.1.5 Traduzione di indirizzi (ARP)	225
4.1.6 Configurazione di host (DHCP)	229
4.1.7 Segnalazione di errori (ICMP)	232
4.1.8 Reti virtuali e tunnel	232
4.2 Instradamento (routing)	235
4.2.1 La rete rappresentata con un grafo	237
4.2.2 Vettore di distanza (RIP)	238
4.2.3 Stato delle linee (OSPF)	245
4.2.4 Metriche	253

4.2.5 Instradamento per host mobili	256
4.3 La rete Internet globale	261
4.3.1 Le sottoreti	261
4.3.2 Instradamento senza classi (CIDR)	266
4.3.3 Instradamento interdominio (BGP)	269
4.3.4 Aree di instradamento	275
4.3.5 La versione 6 di IP (IPv6)	277
4.4 Multicast	288
4.4.1 Multicast a stato delle linee	289
4.4.2 Multicast a vettore di distanza	290
4.4.3 Multicast indipendente dal protocollo (PIM)	293
4.5 Multiprotocol Label Switching (MPLS)	296
4.5.1 Inoltro basato sulla destinazione	297
4.5.2 Instradamento esplicito	302
4.5.3 Reti private virtuali e tunnel	304
4.6 Riepilogo	308
Problema aperto L'installazione di IPv6	309
Ulteriori letture	309
Esercizi	311
 Capitolo 5	
Protocolli di trasporto	325
Problema Far comunicare i processi	325
5.1 Semplice demultiplexing (UDP)	326
5.2 Flusso affidabile di byte (TCP)	329
5.2.1 Problemi end-to-end	329
5.2.2 Formato del segmento	332
5.2.3 Instaurazione e terminazione della connessione	334
5.2.4 Una rivisitazione dell'algoritmo sliding window	338
5.2.5 Simolare la trasmissione	343
5.2.6 Ritrasmissione adattativa	346
5.2.7 Confini tra gruppi di dati (record)	349
5.2.8 Estensioni al protocollo TCP	350
5.2.9 Scelte di progetto alternative	351
5.3 Remote Procedure Call	353
5.3.1 Trasferimento a blocchi (BLAST)	355
5.3.2 Richiesta/Risposta (CHAN)	361
5.3.3 Scavistamento (SELECT)	369
5.3.4 Mettere tutto insieme (SunRPC, DCE)	370
5.4 Prestazioni	377
5.5 Riepilogo	379
Problema aperto Protocolli per applicazioni specifiche	380
Ulteriori letture	381
Esercizi	382

Capitolo 6	
- Controllo della congestione e allocazione di risorse	391
Problema Allocazione di risorse	391
6.1 Problemi nell'allocazione delle risorse	392
6.1.1 Modello della rete	393
6.1.2 Tassonomia	396
6.1.3 Criteri di valutazione	398
6.2 Politiche di gestione delle code	401
6.2.1 FIFO	401
6.2.2 Fair queueing	403
6.3 Controllo di congestione in TCP	407
6.3.1 Aumento additivo/diminuzione moltiplicativa	407
6.3.2 Partenza lenta	410
6.3.3 Ritrasmissione veloce e recupero veloce	414
6.4 Strategie per evitare la congestione	416
6.4.1 DECbit	417
6.4.2 Random early detection (RED)	418
6.4.3 Impedire la congestione alla sorgente	424
6.5 Qualità di servizio	430
6.5.1 Requisiti delle applicazioni	431
6.5.2 Integrated Services (RSVP)	436
6.5.3 Differentiated Services (EF, AF)	445
6.5.4 Qualità di servizio in ATM	449
6.5.5 Controllo di congestione basato su equazioni	452
6.6 Riepilogo	453
Problema aperto All'interno e all'esterno della rete	454
Ulteriori letture	455
Esercizi	456
 Capitolo 7	
Rappresentazione dei dati per la rete	467
Problema Come si trattano i dati?	467
7.1 Formato di presentazione	468
7.1.1 Tassonomia	470
7.1.2 Esempi (XDR, ASN.1, NDR)	473
7.1.3 Linguaggi di marcatura (XML)	477
7.2 Compressione dei dati	480
7.2.1 Algoritmi di compressione senza perdita di informazione	481
7.2.2 Compressione di immagini (JPEG)	483
7.2.3 Compressione video (MPEG)	488
7.2.4 Trasmettere MPEG attraverso la rete	492
7.2.5 Compressione audio (MP3)	496
7.3 Riepilogo	498
Problema aperto Le reti di calcolatori e l'elettronica di consumo	498
Ulteriori letture	499
Esercizi	500

Capitolo 8

<i>Sicurezza delle reti</i>	505
Problema Rendere sicuri i dati	505
8.1 Algoritmi crittografici	506
8.1.1 Requisiti	508
8.1.2 Cifratura a chiave segreta (DES)	509
8.1.3 Cifratura a chiavé pubblica (RSA)	514
8.1.4 Algoritmi di Message Digest (MD5)	516
8.1.5 Implementazione e prestazioni	519
8.2 Strategie di sicurezza	519
8.2.1 Protocolli di autenticazione	519
8.2.2 Protocolli per l'integrità dei messaggi	522
8.2.3 Distribuzione di chiavi pubbliche (X.509)	526
8.3 Esempi di sistemi	528
8.3.1 Pretty Good Privacy (PGP)	529
8.3.2 Secure Shell (SSH)	531
8.3.3 Sicurezza nello strato di trasporto (TLS, SSL, HTTPS)	533
8.3.4 Sicurezza per il protocollo IP (IPSEC)	537
8.4 Firewall	540
8.4.1 Firewall basati su filtri	541
8.4.2 Firewall basati su proxy	542
8.4.3 Limitazioni	544
8.5 Riepilogo	544
Problema aperto Attacchi di tipo "Denial of Service"	545
Ulteriori letture	546
Esercizi	547

Capitolo 9

<i>Applicazioni</i>	553
Problema Ogni applicazione ha bisogno del proprio protocollo	553
9.1 Servizio per i nomi (DNS)	554
9.1.1 Gerarchia di domini	555
9.1.2 Server per i nomi	556
9.1.3 Traduzione dei nomi	559
9.2 Applicazioni tradizionali	562
9.2.1 Posta elettronica (SMTP, MIME, IMAP)	563
9.2.2 World Wide Web (HTTP)	570
9.2.3 Gestione della rete (SNMP)	575
9.3 Applicazioni multimediali	577
9.3.1 Real-time Transport Protocol (RTP)	578
9.3.2 Controllo di sessione e controllo di chiamata (SDP, SIP, H.323)	588
9.4 Reti sovrapposte (overlay networks)	596
9.4.1 Reti sovrapposte per l'instradamento	598
9.4.2 Reti tra pari (peer-to-peer)	605
9.4.3 Reti per la distribuzione di contenuti	613
9.5 Riepilogo	618

Problema aperto Una nuova architettura di rete	618
Ulteriori letture	619
Esercizi	621
<i>Soluzioni di esercizi scelti</i>	627
<i>Glossario</i>	641
<i>Bibliografia</i>	665
<i>Indice analitico</i>	677

Presentazione della terza edizione

David Clark
Massachusetts Institute of Technology

La terza edizione rappresenta un ulteriore importante aggiornamento di questo testo sulle reti, ormai divenuto un classico: il settore continua ad essere in rapida evoluzione e le novità emergono con rapidità impressionante. Questa versione aggiunge la presentazione di molti nuovi ed importanti argomenti, tra cui le reti peer-to-peer, la versione 6 del protocollo IP (IPv6), le reti per la distribuzione dei contenuti e le reti overlay, MPLS e nuove tecniche di commutazione, le tecnologie senza fili (wireless) e per reti mobili, e molti altri ancora. Rispetto alle versioni precedenti, questa è ancora più focalizzata sulle applicazioni, in considerazione della sempre maggiore familiarità di studenti e professionisti con un ampio spettro di applicazioni di rete.

Il libro mantiene fede al suo approccio tradizionale, di fornire le informazioni di cui si ha bisogno per capire il mondo di oggi, ma non perde di vista il suo obiettivo più ampio, di indicare non soltanto i fatti ma anche le motivazioni sottostanti. La filosofia del libro è rimasta la stessa: essere aggiornato ma non soggetto a obsolescenza. Ciò che questo libro vi insegnerà sul mondo attuale delle reti vi darà la possibilità di lavorare nel panorama del futuro: cosa molto importante, dato che non c'è motivo di ritenere che l'evoluzione delle reti rallenterà nel prossimo futuro.

Non è facile ricordare come apparisse il mondo anche soltanto dieci anni fa. A quel tempo Internet non era ancora una realtà commerciale, dieci megabit al secondo era una velocità elevata, non ci preoccupavamo di virus e di messaggi di posta elettronica indesiderati, i nostri calcolatori non avevano protezioni e non ce ne curavamo. Le cose erano più semplici, ma oggi possono essere più stimolanti e fareste meglio a pensare che domani saranno ancora diverse: almeno altrettanto stimolanti, auspicabilmente non meno affidabili e certamente più potenti, più veloci e piene di nuove invenzioni.

A questo punto spero che Larry e Bruce possano riposarsi un po' prima di dover iniziare la prossima revisione del libro. Nel frattempo, usate questo testo per imparare il presente e per prepararvi per il futuro. Divertitevi.

Presentazione della prima edizione

David Clark

Massachusetts Institute of Technology

Il termine "codice a spaghetti" viene universalmente inteso come un'offesa. Ogni valido scienziato dell'informazione incensa i vantaggi della modularità, che porta con sé molti benefici, primo fra tutti l'incredibile vantaggio di non dover comprendere tutte le parti di un problema nello stesso momento per poterlo risolvere. La modularità, quindi, gioca un ruolo importante nella presentazione delle idee contenute in un libro, così come nella scrittura di un programma: se il materiale presente in un libro è organizzato in modo efficace, cioè in modo modulare, il lettore può incominciare dall'inizio e arrivare realmente alla fine.

Il settore dei protocolli di rete è forse unico in questo, nel senso che la "giusta" modularità ci viene fornita nella forma di uno standard internazionale: il modello di riferimento ISO a sette livelli per i protocolli di rete. Questo modello, che presenta un approccio stratificato alla modularità, viene usato quasi universalmente come punto di partenza per la presentazione dell'organizzazione dei protocolli, sia nel caso in cui il progetto in esame è conforme al modello, sia quando devia da esso.

Organizzare un libro di reti attorno al modello a strati sembrerebbe naturale, ma nel far questo si va incontro ad un pericolo, perché il modello OSI non ha avuto molto successo nell'organizzare i concetti chiave delle reti. Alcuni requisiti basilari come l'affidabilità, il controllo di flusso o la sicurezza possono far parte di molti, se non di tutti, i livelli OSI: ciò crea molta confusione nel tentativo di capire il modello di riferimento, al punto da suscitare, a volte, un sentimento di scetticismo. A dire il vero, un libro che sia rigidamente organizzato secondo un modello a strati presenta alcune delle caratteristiche tipiche del codice a spaghetti.

Osservazione che ci porta al presente volume: Peterson e Davie seguono il tradizionale modello stratificato, ma non pretendono che esso sia realmente di aiuto nel comprendere i problemi importanti delle reti di calcolatori. Per questo motivo, gli autori hanno organizzato la presentazione dei concetti fondamentali in modo indipendente dalla stratificazione: leggendo il libro, i lettori impareranno il controllo di flusso, il controllo della congestione, i miglioramenti dell'affidabilità, la rappresentazione dei dati e la sincronizzazione, seguendo un percorso non soggetto alle implicazioni che derivano dall'affrontare questi temi all'interno di uno piuttosto che di un altro strato del modello tradizionale.

Questo è un volume aggiornato, che focalizza l'attenzione sui protocolli più utilizzati al giorno d'oggi, in particolare i protocolli Internet. Peterson e Davie hanno una grande esperienza di Internet e il loro libro tratta non soltanto gli aspetti teorici della progettazione di protocolli, ma anche i fattori che hanno importanza pratica. Sono trattati anche alcuni dei protocolli che stanno assumendo sempre maggiore importanza, in modo che il lettore abbia una prospettiva aggiornata; ancora più importante è il fatto che la presentazione degli aspetti principali discenda dalla natura del problema, piuttosto che dai vincoli del modello di riferimento a strati o dai dettagli dei protocolli odierni. In questo senso i contenuti sono aggiornati ma non soggetti a obsolescenza: la combinazione di esempi attuali tratti dalla realtà e di accurate spiegazioni dei concetti fondamentali rende questo libro unico.

Presentazione dell'edizione italiana

Marcello Dalpasso

Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Padova
marcello.dalpasso@unipa.it

Una delle domande più difficili a cui deve rispondere un curatore editoriale, nel momento in cui viene presa la decisione sulla traduzione di un testo di questo spessore culturale, scientifico e didattico, è: "Perché proprio questo? Quali vantaggi ci sono rispetto ad altri testi dello stesso settore?"

Nel caso del volume di Peterson e Davie, dare una risposta è stato più semplice di quanto accaduto in altre occasioni. Per prima cosa, la loro autorevolezza è indiscussa nel panorama internazionale delle reti di calcolatori e costituisce una miscela vincente tra esperienza accademica e di ricerca scientifica da un lato, e vastissima esperienza professionale in settori di ricerca e sviluppo dall'altro: fra le tante loro attività, entrambi hanno dato apporti significativi allo sviluppo della rete Internet, così come la conosciamo oggi.

L'esperienza didattica di Peterson, in particolare, consente a questo testo di porsi legittimamente l'ambizioso obiettivo di aiutare lo studio di singoli autodidatti così come di studenti universitari di corsi di laurea di primo e di secondo livello, oltre che di master. La trattazione completa e approfondita di ognuno degli aspetti fondamentali delle reti di calcolatori è strutturata in modo da consentire anche una esposizione più panoramica e culturale, lasciando i dettagli ad approfondimenti successivi; proprio questa impostazione consente un agevole utilizzo del testo a supporto di un corso di laurea universitario di primo livello così come di un corso di laurea specialistico, seguendo anche i suggerimenti forniti dagli stessi Autori nella loro prefazione.

Un altro punto di forza di questo testo, soprattutto se confrontato con altri libri sulle reti di calcolatori, è la ricchissima dotazione di esercizi di varie tipologie: si spazia dagli esercizi analitici a quelli di progettazione, dagli esercizi di programmazione a quelli di approfondimento culturale. Lo schema didattico adottato dagli Autori, che prevede l'inserimento nel testo delle soluzioni di alcuni esercizi, selezionati con cura in qualità di "esercizi rappresentativi", costituisce un'interessante novità nel panorama italiano, mentre è una pratica abbastanza consolidata negli Stati Uniti: una sorta di piccola, ma efficace, guida che aiuta lo studente nello svolgimento delle attività complementari allo studio, che raramente in questo settore non prevede una fase di svolgimento di esercizi.

Dal punto di vista scientifico e culturale, poi, il testo si presenta completo e aggiornato fino agli ultimi e innovativi aspetti dell'evoluzione di Internet e, più in generale, del panorama completo delle reti di calcolatori: dalle reti wireless alla nuova versione del protocollo IP, dalle reti peer-to-peer alle reti overlay, fino alle strategie MPLS, non manca davvero nulla.

Per quanto riguarda, in particolare, la presente edizione italiana, la scelta del traduttore è stata quella di mantenere inalterati tutti quei termini per i quali è invalso l'utilizzo del vocabolo inglese anche in Italia, evitando forzature linguistiche difficilmente giustificabili, pur fornendo gli strumenti lessicali per una corretta comprensione. Un esempio fra tutti: in Italia si sente quotidianamente parlare di *router* e disseminare il testo di *intradicatori* lo avrebbe reso quasi illeggibile.

Siamo dunque di fronte ad un testo completo, approfondito e modulare, che certamente soddisferà le esigenze culturali, didattiche e scientifiche di molti, se non di tutti.

Prefazione

Quando, nel 1996, venne pubblicata la prima edizione di questo libro, acquistare prodotti in Internet era una novità e un'azienda che pubblicizzava il proprio nome di dominio veniva considerata "di punta". Oggi il commercio elettronico è una realtà e le società ".com" hanno già attraversato un ciclo economico di forte espansione e di collasso. Un vasto insieme di tecnologie, dai commutatori ottici alle reti senza fili, sta diventando di utilizzo comune e l'unica cosa che sembra essere prevedibile di Internet è il suo costante cambiamento.

Nonostante questi cambiamenti, la domanda che abbiamo posto nella prima edizione è tuttora valida: quali sono i concetti e le tecnologie fondamentali che fanno funzionare Internet? La risposta è che gran parte dell'architettura TCP/IP continua a funzionare proprio come era stata immaginata dai suoi ideatori quasi 30 anni fa, cosa che non significa che l'architettura di Internet non sia interessante, al contrario: comprendere i principi progettuali che sottostanno ad un'architettura che non solo è sopravvissuta, ma che ha contribuito fortemente alla crescita e ai cambiamenti di Internet negli ultimi tre decenni è proprio il modo giusto di iniziare. Come le precedenti edizioni, la terza edizione considera sua pietra miliare la comprensione delle motivazioni che stanno alla base dell'architettura di Internet.

I nostri lettori

Il nostro scopo è che questo libro serva come testo per un corso di reti onnicomprensivo nella laurea specialistica o negli ultimi anni della laurea di primo livello. Siamo anche convinti che il piano editoriale del libro, focalizzato sui concetti chiave, possa essere di interesse per i professionisti dell'industria che si stiano riqualificando su progetti relativi alle reti, così come per i tecnici che si occupano di reti e che vogliono capire "come" funzionano i protocolli con cui lavorano ogni giorno, per avere una panoramica della rete nel suo complesso.

Secondo la nostra esperienza, sia gli studenti sia i professionisti che per la prima volta si avvicinano allo studio delle reti hanno spesso l'impressione che i protocolli di rete siano una specie di editti discesi dall'alto e che il loro compito sia quello di imparare il maggior numero

possibile di acronimi. In realtà, i protocolli non sono altro che gli elementi costitutivi di un sistema complesso, che è stato sviluppato mediante l'applicazione dei principi fondamentali della progettazione ingegneristica. Inoltre, essi vengono costantemente rivisti, estesi e sostituiti sulla base dell'esperienza reale. Con questa consapevolezza, il nostro obiettivo è quello, tramite questo libro, di fare qualcosa di più di una semplice rassegna dei protocolli in uso oggi, spiegando invece i principi fondamentali di una buona progettazione delle reti. Siamo certi che l'apprendimento di questi concetti di base sia il migliore strumento per poter gestire la velocità di cambiamento del settore delle reti.

Modifiche nella terza edizione

Anche se abbiamo posto l'attenzione sui principi fondamentali delle reti, presentiamo questi principi usando esempi tratti dalla rete Internet operativa ai giorni nostri, per cui abbiamo aggiunto parecchio materiale nuovo per seguire le numerose recenti innovazioni nelle reti di calcolatori. Abbiamo, poi, eliminato, riorganizzato e modificato alcune parti del materiale esistente, per tener conto delle modifiche che sono avvenute negli ultimi sette anni.

Forse il cambiamento più significativo che abbiamo notato da quando abbiamo scritto la prima edizione è che oggi quasi tutti i lettori hanno familiarità con le applicazioni di rete, come il World Wide Web e la posta elettronica. Per questo motivo abbiamo aumentato l'attenzione per le applicazioni stesse, fin dal primo capitolo: usiamo le applicazioni per motivare lo studio delle reti e per derivarne un insieme di requisiti che una rete efficiente deve soddisfare per far funzionare le applicazioni presenti e future su scala globale. Ciò nonostante, continuiamo ad utilizzare l'approccio delle edizioni precedenti, orientato alla risoluzione dei problemi, iniziando dall'interconnessione di calcolatori, esaminando gli strati dal basso verso l'alto e terminando con un dettagliato esame delle caratteristiche del livello applicativo. Ci sembra molto importante fornire adeguate motivazioni agli argomenti trattati nel libro, iniziando con le applicazioni e con le loro necessità, ma, allo stesso tempo, siamo convinti che i problemi dei livelli superiori, come quelli dei protocolli di applicazione e di trasporto, possano essere compresi in modo migliore dopo che siano stati evidenziati i problemi di base, che riguardano la connessione dei calcolatori in una rete e la commutazione di pacchetti.

Un'altra importante modifica introdotta in questa edizione riguarda gli esercizi, di cui abbiamo aumentato il numero e la qualità, cercando di identificare quelli particolarmente complessi o che richiedono conoscenze matematiche superiori alla media (tali esercizi sono contrassegnati da un asterisco); inoltre, in ciascun capitolo vengono proposti alcuni esercizi la cui soluzione è presentata nel libro stesso.

Come nella seconda edizione, abbiamo incrementato il numero di argomenti importanti o ne abbiamo esteso la trattazione, mentre altri ancora sono stati aggiornati. Le modifiche più rilevanti in tal senso sono:

- una nuova sezione riguardante il Multiprotocol Label Switching (MPLS), con la trattazione della gestione ingegneristica del traffico e delle reti virtuali private;
- una nuova sezione sulle reti overlay, che comprendono le reti "peer-to-peer" e le reti per la distribuzione di contenuti;
- una trattazione molto più estesa dei protocolli per le applicazioni multimediali, tra cui Session Initiation Protocol (SIP) e Session Description Protocol (SDP);

- una trattazione aggiornata dei meccanismi di controllo della congestione, con l'aggiunta delle conferme selettive per il protocollo TCP, del controllo di congestione basato su equazioni e della notifica esplicita di congestione;
- una trattazione aggiornata dei problemi di sicurezza, che ora parla anche di attacchi di tipo DDoS (Distributed Denial of Service);
- l'inserimento di materiale aggiornato sulla tecnologia wireless, che comprende le tecniche di spread spectrum e i nuovi standard 802.11.

L'impostazione del testo

In un settore soggetto a cambiamenti così repentinamente come quello delle reti di calcolatori, la cosa più importante che un libro può offrire è una visione prospettica, per distinguere ciò che è essenziale da ciò che non lo è, e ciò che è duraturo da ciò che è effimero. Forti della nostra esperienza più che ventennale nel condurre ricerche che hanno portato a nuovi sviluppi nelle tecnologie di rete e hanno contribuito alla produzione commerciale di nuove apparecchiature, nonché nell'insegnare a studenti universitari di ogni livello le ultime novità e tendenze delle reti, abbiamo sviluppato una prospettiva, da noi chiamata approccio sistematico (*system approach*), che costituisce il principio ispiratore di questo libro. Questo approccio ha alcune conseguenze.

- Invece di accettare in modo dogmatico ciò che esiste, noi partiamo dalle fondamenta e ripercorriamo insieme a voi il processo cognitivo che ha prodotto le reti di oggi: questo ci consente di spiegarvi *perché* le reti sono come le vediamo. La nostra esperienza ci ha insegnato che, una volta capiti a fondo i concetti basilari, vi sarà molto semplice assimilare qualsiasi nuovo protocollo con cui dovrete avere a che fare.
- Nonostante il libro sia organizzato secondo il tradizionale modello a strati delle reti, iniziando dai livelli inferiori e procedendo verso l'alto lungo la pila di protocolli, non seguiamo rigidamente l'approccio a livelli. Molti argomenti, di cui il controllo di congestione e la sicurezza sono validi esempi, hanno riflessi su molti livelli della gerarchia, per cui li presentiamo al di fuori del tradizionale modello a strati. In sintesi, siamo convinti che la stratificazione sia ottima come strumento di servizio ma carente come linea guida: molto spesso è maggiormente utile una prospettiva a tutto campo, di tipo end-to-end.
- Invece di illustrare il funzionamento dei protocolli in astratto, utilizziamo i protocolli più importanti tra quelli in uso ai giorni nostri (molti dei quali fanno parte dell'architettura TCP/IP e di Internet) per far vedere come funzionano le reti nella pratica, consentendoci questo di introdurre nella discussione esperienze del mondo reale.
- Anche se ai livelli più bassi le reti sono costituite da elementi hardware che si possono facilmente reperire nei negozi di articoli per computer e da servizi di comunicazione che si possono noleggiare da una compagnia telefonica, è il software che è in grado di fornire servizi innovativi e di adattarsi rapidamente ai cambiamenti. Per questa ragione, poniamo molta enfasi sul modo in cui il software di rete viene realizzato, piuttosto che accontentarci di una descrizione astratta degli algoritmi, presentando anche porzioni di codice estratte da una reale pila di protocolli per farvi vedere come si possano implementare alcuni protocolli ed i relativi algoritmi.
- Le reti sono composte di molti elementi costitutivi e, nonostante sia necessario astrarre dai particolari ininfluenti mentre si risolve un particolare problema, è altresì essenziale

comprendere come tutti questi elementi si compongano per rendere funzionante una rete. Per questo motivo impiegheremo parecchio tempo ad illustrare il comportamento complessivo delle reti e non soltanto quello dei loro componenti presi singolarmente, in modo che si possa capire come funziona una rete nella sua interezza, comprendendo applicazioni e hardware.

- L'approccio sistematico impone di effettuare studi sperimentali, per poi utilizzare i dati raccolti sia per analizzare quantitativamente le diverse opzioni di progetto sia per svolgere una funzione di guida nell'ottimizzazione dell'implementazione: questa enfasi sull'analisi empirica pervade l'intero libro.
- Le reti sono come tutti gli altri sistemi di elaborazione: come i sistemi operativi, le architetture dei processori, i sistemi distribuiti e paralleli, le reti sono grandi e complesse. Per poter gestire questa complessità, i progettisti di sistemi spesso identificano un insieme di regole progettuali, che noi porremo in evidenza a mano a mano che vengono presentate nel libro, corredandole di esempi relativi, ovviamente, alle reti di calcolatori.

Lo schema pedagogico e le caratteristiche del libro

Questa terza edizione continua a presentare alcune interessanti caratteristiche di cui vi invitiamo a sfruttare i vantaggi.

- *Enunciazione di problemi.* All'inizio di ogni capitolo descriviamo un problema che identifica il successivo insieme di argomenti su cui focalizzare l'attenzione nel progetto di una rete. Questa enunciazione presenta e fornisce una motivazione per gli argomenti che saranno approfonditi nel capitolo.
- *Blocchi di testo collaterali.* Nel libro sono presenti blocchi di testo isolati dal flusso principale di lettura, per introdurre speculazioni sull'argomento trattato o un argomento avanzato. In molti casi questi blocchi di testo riportano aneddoti reali sulle reti di calcolatori.
- *Paragrafi evidenziati.* Questi paragrafi riassumono un nucleo importante di informazioni che vogliamo evidenziare all'interno di una discussione, come, ad esempio, un principio di progettazione sistematica di ampia applicabilità.
- *Protocolli reali.* Anche se il libro si concentra sui concetti chiave piuttosto che sulle specifiche di protocolli esistenti, per illustrare la maggior parte delle idee più importanti vengono usati protocolli reali: di conseguenza, il libro può anche essere usato come manuale di riferimento per molti protocolli. Per aiutarvi a reperire le descrizioni dei protocolli, l'intestazione di ciascuna sezione che lo richieda contiene, tra parentesi, i nomi dei protocolli ivi descritti. Ad esempio, la Sezione 5.2, che descrive i principi dei protocolli affidabili da un estremo all'altro, fornisce anche una dettagliata descrizione del protocollo TCP, l'esempio canonico di tali protocolli.
- *Problemi aperti.* Il corpo principale di ciascun capitolo viene concluso con un argomento importante che è ancora in discussione nella comunità scientifica, nel mondo industriale e commerciale, o nella società nel suo complesso. Riteniamo che la discussione di questi temi aiuti a rendere più importanti e interessanti le reti di calcolatori.
- *Ulteriori letture.* Questi elenchi, accuratamente selezionati, compaiono al termine di ciascun capitolo e contengono le pubblicazioni di riferimento sugli argomenti appena presentati. Riteniamo indispensabile che gli studenti di lauree specialistiche leggano con attenzione queste pubblicazioni, per integrare il materiale trattato nel capitolo.

Schema di lettura e di utilizzo in un corso

Il libro è organizzato come segue:

- Il Capitolo 1 presenta i concetti fondamentali utilizzati nel resto del libro. Prendendo spunto da applicazioni di ampia diffusione, delinea ciò che è necessariamente presente nell'architettura di una rete e definisce le metriche per quantificare le prestazioni che solitamente guidano la progettazione di una rete.
- Il Capitolo 2 fornisce una panoramica di tecnologie di rete di basso livello, partendo da Ethernet, passando per token ring, per arrivare alle tecnologie wireless, e descrive anche molti dei problemi che devono essere affrontati da tutti i protocolli del livello di linea di collegamento dei dati (data link), tra cui la codifica, la tramatura (framing) e la rivelazione d'errore.
- Il Capitolo 3 presenta i modelli fondamentali delle reti commutate (a pacchetto o a circuito virtuale) e descrive in dettaglio una delle tecnologie di commutazione predominanti (ATM). Inoltre, delinea i principi della progettazione di commutatori realizzati in hardware.
- Il Capitolo 4 presenta l'interconnessione di reti e descrive i concetti chiave del protocollo IP (Internet Protocol). Questo capitolo affronta un problema centrale: come si possono instradare i pacchetti all'interno di reti quando esse assumono le dimensioni di Internet.
- Il Capitolo 5 si sposta al livello di trasporto, descrivendo in dettaglio sia il protocollo TCP (Transmission Control Protocol) di Internet sia il protocollo RPC (Remote Procedure Call) usato nella realizzazione di applicazioni client/server.
- Il Capitolo 6 parla di controllo di congestione e allocazione di risorse. I problemi affrontati in questo capitolo sono trasversali sia al livello di rete (Capitoli 3 e 4) sia al livello di trasporto (Capitolo 5). In particolare, questo capitolo descrive come funziona il controllo di congestione nel protocollo TCP e presenta i meccanismi usati per fornire qualità di servizio sia in Internet sia in reti ATM.
- Il Capitolo 7 prende in esame i dati inviati attraverso una rete, parlando sia dei problemi di formato dei dati, sia della loro compressione. La discussione della compressione contiene, fra gli altri temi, una spiegazione di come funzionano la compressione video MPEG e la compressione audio MP3.
- Il Capitolo 8 discute di sicurezza delle reti, spaziando da una panoramica sui protocolli di crittografia (DES, RSA, MD5), ai protocolli per i servizi di sicurezza (autenticazione, firma digitale, integrità dei messaggi), fino a sistemi di sicurezza globale (posta elettronica con meccanismi di privacy, IPSEC). Il capitolo parla anche di problemi pratici, come i firewall.
- Il Capitolo 9 descrive un campione rappresentativo di applicazioni di rete e i relativi protocolli utilizzati, tra cui applicazioni tradizionali come la posta elettronica e il Web, applicazioni multimediali come la telefonia IP e il video streaming, e le reti overlay come la condivisione di archivi peer-to-peer e reti di distribuzione di contenuti.

Per un corso di laurea di primo livello, molto probabilmente sarà necessario soffermarsi maggiormente in aula sugli argomenti introduttivi del primo capitolo, tipicamente a spese degli argomenti più avanzati trattati nei Capitoli 6, 7 e 8. Il Capitolo 9 torna a parlare di argomenti molto popolari, come le applicazioni di rete. Al contrario, i docenti di un corso di laurea specialistica dovrebbero essere in grado di affrontare in una lezione o due gli argomenti del primo capitolo (e gli studenti dovrebbero studiare autonomamente quel materiale

in modo più approfondito), recuperando così tempo per trattare in profondità gli ultimi quattro capitoli. Il materiale più basilare, contenuto nei quattro capitoli intermedi (dal Capitolo 2 al Capitolo 5), sarà trattato in tutti i corsi, anche se in un corso di laurea di primo livello si potrebbe decidere di non approfondire le sezioni più avanzate (ad esempio, le Sezioni 2.2, 2.9, 3.4 e 4.4).

Per quei lettori che usino il libro per l'auto-apprendimento, siamo convinti che gli argomenti che abbiamo selezionato coprano il nucleo fondamentale delle reti di calcolatori, per cui raccomandiamo una lettura del libro in sequenza, dall'inizio alla fine. Inoltre, abbiamo inserito nel testo un gran numero di riferimenti, che vi aiuteranno ad identificare materiale aggiuntivo che sia pertinente alle vostre specifiche aree di interesse; infine, troverete le soluzioni di un insieme selezionato di esercizi.

Il libro affronta in modo originale il problema del controllo di congestione, spostando tutti gli argomenti ad esso relativi, nonché quelli connessi all'allocazione di risorse, in un unico capitolo, il Capitolo 6. Abbiamo preso questa decisione perché il problema del controllo di congestione non può essere risolto ad alcun singolo livello e vogliamo che prendiate in considerazione tutte le diverse opzioni di progetto contemporaneamente (coerentemente con la nostra idea, che spesso una trattazione troppo rigida degli strati di una rete renda oscuri alcuni importanti compromessi progettuali). Tuttavia, è possibile affrontare in modo più tradizionale il problema del controllo di congestione, studiando la Sezione 6.2 insieme al Capitolo 3 e la Sezione 6.3 insieme al Capitolo 5.

Esercizi

Abbiamo fatto un ingente sforzo per migliorare gli esercizi sia nella seconda che nella terza edizione. Nella seconda edizione abbiamo consistentemente aumentato il numero di esercizi e, in base ai risultati in aula, sensibilmente migliorato la loro qualità. In questa edizione abbiamo aggiunto altri esercizi ancora, ma abbiamo anche apportato due importanti modifiche.

- Abbiamo contrassegnato con un asterisco, per indicare un particolare livello di difficoltà, quegli esercizi che ci sembrano particolarmente impegnativi o che richiedono conoscenze non fornite dal testo (ad esempio, conoscenze di teoria della probabilità).
- In ciascun capitolo abbiamo aggiunto alcuni esercizi, particolarmente rappresentativi, per i quali vengono fornite soluzioni corrette al termine del libro. Questi esercizi, contrassegnati da un segno di sputa, hanno lo scopo di fornire una traccia che sia di aiuto nella soluzione degli altri esercizi del libro.

Attualmente, ci sono esercizi di tipo diverso.

- Esercizi analitici che chiedono allo studente di eseguire semplici calcoli algebrici che dimostrino la comprensione delle relazioni fondamentali.
- Esercizi di progettazione che chiedono allo studente di proporre e di valutare protocolli per problemi diversi.
- Esercizi di programmazione che chiedono allo studente di scrivere qualche linea di codice per verificare un'idea o per condurre esperimenti con un'applicazione di rete esistente.
- Esercizi di consultazione di materiale di biblioteca che chiedono allo studente di imparare dettagli ulteriori su un particolare argomento.

Inoltre, come descritto più in dettaglio in seguito, sono disponibili sul Web degli esercizi di programmazione basati sui socket.

Materiale supplementare e risorse sul Web

Ulteriore materiale di supporto è disponibile sul sito Web del libro: www.apogeonline.com/libri/02158/allegati/. Vi invitiamo a visitare le pagine dedicate a questo libro con regolarità, dato che aggiungeremo ulteriore materiale di supporto e indicheremo collegamenti a siti dedicati alle reti di calcolatori.

Ringraziamenti

Scrivere questo libro non sarebbe stato possibile senza l'aiuto di molte persone, che vorremmo ringraziare per i loro sforzi profusi nel migliorare il risultato finale. Prima di farlo, però, dobbiamo dire che abbiamo fatto del nostro meglio per correggere gli errori che i revisori ci hanno segnalato e per descrivere accuratamente i protocolli e tutto ciò che i nostri colleghi ci hanno spiegato: gli errori rimasti sono da addebitare soltanto a noi. Se dovete trovarne, vi preghiamo di inviare un messaggio di posta elettronica al nostro editore, Morgan Kaufmann, all'indirizzo netbugs@mfp.com: promettiamo di correggerli nella futura versione di questo libro.

Innanzitutto vogliamo ringraziare coloro che hanno revisionato le versioni preliminari di tutto o di parte del manoscritto. Oltre a coloro che lo hanno fatto per le edizioni precedenti, vogliamo ringraziare Carl Emberger, Isaac Ghansah e Bobby Bhattacharjee per le loro attente revisioni. Grazie anche a Peter Druschel, Limin Wang, Aki Nakao, Dave Oran, George Swallow, Peter Lei e Michael Ramalho per le loro revisioni di alcune sezioni. Vogliamo anche ringraziare tutti coloro che hanno fornito suggerimenti per aiutarci a decidere le modifiche da apportare in questa edizione: Chedley Aouriri, Peter Steenkiste, Esther A. Hughes, Ping-Tsai Chung, Doug Szajda, Mark Andersland, Leo Tam, C. P. Watkins, Brian L. Mark, Miguel A. Labrador, Gene Chase, Harry W. Tyrer, Robert Siegfried, Harlan B. Russell, John R. Black, Robert Y. Ling, Julia Johnson, Karen Collins, Clark Verbrugge, Monjy Rabemanantsoa, Kerry D. LaViolette, William Honig, Kevin Mills, Murat Demirel, J Rufinus, Manton Matthews, Erin W. Fulp, Wayne Daniel, Luiz DaSilva, Don Yates, Raouf Boules, Nick McKeown, Neil T. Spring, Kris Verma, Szuecs Laszlo, Ted Herman, Mark Sternhagen, Zongming Fei, Dulal C. Kar, Mingyan Liu, Ken Surendran, Rakesh Arya, Mario J. Gonzalez, Annie Stanton, Tim Batten e Paul Francis.

Poi, diversi membri del Network System Group a Princeton hanno fornito idee, esempi, correzioni, dati e codice per questo libro. In particolare, vorremmo ringraziare Andy Bavier, Tammo Spalink, Mike Wawrzoniak, Zuki Gottlieb, George Tzanetakis e Chad Mynhier. Inoltre, vogliamo ringraziare nuovamente Defense Advanced Research Projects Agency, National Science Foundation, Intel Corporation e Cisco Systems, Inc. per il loro finanziamento alla nostra ricerca sulle reti di calcolatori negli anni passati.

Infine, vorremmo ringraziare il curatore di questa serie di libri, David Clark, così come le persone di Morgan Kaufmann che ci hanno aiutato nel difficile processo di scrittura del libro. Uno speciale ringraziamento va al curatore della prima versione, Jennifer Mann, al curatore della terza edizione, Rick Adams, al nostro curatore delle fasi di sviluppo, Karyn Johnson, e al nostro direttore di produzione, Simon Crump. È stato un vero piacere lavorare con tutto lo staff di MKP.

Gli Autori

Larry L. Peterson è Professore di *Computer Science* alla Princeton University e ha insegnato, in precedenza, alla University of Arizona. Ha partecipato alla progettazione e alla valutazione di numerosi protocolli di rete e dirige attualmente il progetto PlanetLab. Ha retto l'incarico di *Editor-in-Chief* ("capo curatore editoriale") per *ACM Transactions on Computer Systems* e ha partecipato a comitati di programma per SOSP, SIGCOMM, OSDI e ASPLOS. È *Fellow* di ACM e membro dell'End-to-End Research Group di Internet.

Bruce S. Davie è *Fellow* di Cisco Systems, Inc., dove partecipa allo sviluppo di MPLS e delle tecnologie relative alla qualità di servizio. È autore di numerosi articoli su riviste scientifiche, lavori presentati a conferenze e documenti RFC, nonché coautore di altri due libri nello stesso settore. Partecipa attivamente sia alla Internet Engineering Task Force (IETF) sia all'End-to-End Research Group, oltre ad essere *senior member* dell'associazione IEEE.

Principi fondamentali

Problema

Costruire una rete

Supponete di voler costruire una rete di calcolatori, che abbia la potenzialità di crescere fino a dimensioni planetarie e di supportare applicazioni così diversificate come la teleconferenza, il video-on-demand, il commercio elettronico, l'elaborazione distribuita e le biblioteche digitali. Quali tecnologie, tra quelle disponibili, serviranno come elementi costitutivi e che tipo di architettura software dovrete progettare per integrarli in un servizio di comunicazione efficiente? Rispondere a questa domanda è l'ambizioso obiettivo di questo libro: descrivere gli elementi costitutivi disponibili per costruire reti, per poi mostrare come possano essere usati per strutturare una rete a partire dalle fondamenta.

Prima di poter capire come progettare una rete di calcolatori, dovremmo metterci d'accordo su cosa sia esattamente una rete di calcolatori. In un certo periodo storico, il termine *rete* aveva il significato di un insieme di linee seriali usate per collegare terminali a grandi calcolatori di tipo mainframe. Per alcune persone il termine rete richiama alla mente la rete di telefonia vocale. Per altri, la sola rete che abbia senso è la rete via cavo usata per distribuire i segnali televisivi. La cosa principale che accomuna queste reti è la loro specializzazione, per la gestione di un particolare tipo di dati (pressioni di tasti su una tastiera, voce, video) e la loro tipica connessione a dispositivi con utilizzi specifici (terminali, telefoni e televisori).

Che cosa distingue una rete di calcolatori da questi altri tipi di reti? Probabilmente, la caratteristica più importante di una rete di calcolatori è la sua generalità. Le reti di calcolatori vengono costruite, principalmente, per hardware programmabile senza utilizzi specifici e non sono ottimizzate per un'applicazione particolare, quale potrebbe essere la telefonia o la distribuzione di segnali televisivi. Al contrario, le reti di calcolatori sono in grado di trasportare tipi di dati molto diversi e forniscono supporto ad un ampio e sempre crescente campo di applicazioni. Questo capitolo prende in esame alcune tipiche applicazioni delle reti di calcolatori e discute i requisiti di cui deve essere consapevole un progettista di reti che voglia fornire il supporto a tali applicazioni.

Una volta compresi i requisiti, come procederemo? Per fortuna non stiamo costruendo la prima rete. Prima di noi l'hanno fatto altri, tra i quali vanno menzionati i ricercatori che

appartengono alla comunità scientifica che ha la responsabilità di Internet: useremo il bagaglio di conoscenze generato da Internet come guida per il nostro progetto. Questa esperienza è racchiusa in un'architettura di rete che identifica i componenti hardware e software disponibili e mostra come possano essere utilizzati per formare un sistema di rete completo.

Per iniziare la nostra strada verso la comprensione di come costruire una rete, questo capitolo fa quattro cose. Prima di tutto, fa un'analisi dei requisiti posti per la rete da diverse applicazioni e da diverse comunità di persone (tra cui utilizzatori e operatori della rete). Secondariamente, il capitolo presenta l'idea di un'architettura di rete, che getta le basi per il resto del libro. Successivamente vengono presentati i concetti chiave per l'implementazione delle reti di calcolatori e, infine, vengono identificate le metriche utilizzate per valutare le prestazioni delle reti.

1.1 Applicazioni

La maggior parte delle persone conosce Internet perché ne conosce le applicazioni: il World Wide Web, la posta elettronica, i flussi audio e video, le stanze di chat e la condivisione di file musicali. Il Web, ad esempio, presenta un'interfaccia intuitiva e semplice: gli utenti visualizzano pagine composte da testo e oggetti grafici, selezionano con il mouse oggetti sui quali vogliono saperne di più e compare una nuova pagina. La maggior parte delle persone è anche consapevole che, anche se non si vede, ciascun oggetto selezionabile in una pagina è collegato ad un identificativo della successiva pagina da visualizzare, chiamato URL (Uniform Resource Locator), che associa un nome univoco a ciascuna pagina che può essere visualizzata da un navigatore Web. Ad esempio,

<http://www.mkp.com/pd3e>

è lo URL di una pagina che presenta la versione originale di questo libro presso Morgan Kaufmann. La stringa http indica che per scaricare la pagina si usa il protocollo HTTP (HyperText Transfer Protocol), www.mkp.com è il nome del computer che fornisce la pagina e pd3e identifica univocamente la pagina nel sito della casa editrice.

Ciò di cui la maggior parte degli utenti del Web non è consapevole, però, è che selezionando con il mouse uno di tali URL possono essere scambiati fino a 17 messaggi sulla rete Internet, e questo nell'ipotesi che la pagina stessa sia abbastanza piccola da trovar posto in un singolo messaggio. Questo numero comprende fino a sei messaggi necessari per tradurre il nome del server (www.mkp.com) nel suo indirizzo Internet (213.38.165.180), tre messaggi per allestire una connessione TCP (Transmission Control Protocol) tra il browser e tale server, quattro messaggi per l'invio da parte del browser della richiesta HTTP di tipo "get" e per la risposta da parte del server contenente la pagina richiesta (nonché per la conferma di avvenuta ricezione dei messaggi da parte di entrambi), e quattro messaggi per porre fine alla connessione TCP. Ovviamente, questo non comprende i milioni di messaggi che si scambiano quotidianamente i nodi Internet soltanto per far sapere agli altri nodi della propria esistenza e di essere in grado di fornire pagine Web, di tradurre nomi in indirizzi e di instradare messaggi verso la loro destinazione finale.

Anche se non ancora così utilizzata come la navigazione Web, la diffusione di flussi audio e video (streaming) è un'altra applicazione emergente di Internet. Nonostante un intero file video potrebbe essere completamente scaricato da una macchina remota per essere poi riprodotto sulla macchina locale, in modo simile al processo di scaricamento e visualizzazione

di una pagina Web, ciò comporterebbe di dover attendere lo scaricamento dell'ultimo secondo del file video prima di poter iniziare a vederlo. Per lo streaming video, il mittente e il destinatario sono, rispettivamente, la sorgente e la destinazione del flusso video, cioè la sorgente genera un flusso video (utilizzando, ad esempio, una scheda per la videoregistrazione) e lo invia attraverso Internet all'interno di messaggi, mentre la destinazione visualizza il flusso mentre questo arriva.

Per essere più precisi, non si dovrebbe parlare di applicazione video, ma di dati di tipo video. Un esempio di applicazione video è il video su richiesta (video-on-demand), che legge un film da un archivio e lo trasmette sulla rete. Un altro tipo di applicazione è la videoconferenza, che in realtà è il caso più interessante, perché ha vincoli temporali molto stringenti. Esattamente come quando si usa il telefono, l'interazione tra i partecipanti deve essere sincronizzata; quando una persona gesticola, tale azione deve essere visualizzata all'altro estremo il più velocemente possibile. Un ritardo eccessivo rende il sistema inutilizzabile. Al contrario, anche se trascorrono alcuni secondi dal momento in cui l'utente inizia la videoconferenza al momento in cui viene visualizzata la prima immagine, il servizio viene comunque ritenuto soddisfacente. Inoltre, la videoconferenza, essendo interattiva, implica solitamente che il video fluisce in entrambe le direzioni, mentre un'applicazione di video-on-demand trasmette molto probabilmente il video in un'unica direzione.

L'applicazione vic di Unix è un esempio di uno strumento di videoconferenza molto popolare, di cui la Figura 1.1 mostra il pannello di controllo di una sessione. Si noti che vic è in realtà uno degli strumenti di una suite di applicazioni per la videoconferenza progettati al Lawrence Berkeley Laboratory e a UC Berkeley. Gli altri strumenti sono un'applicazione di lavagna condivisa (wb) che consente agli utenti di scambiarsi schizzi e disegni, uno strumento audiovisivo chiamato vat e un gestore di sessioni (sdr) usato per creare videoconferenze e renderle pubbliche o notificarle agli altri utenti. Tutti questi strumenti operano in ambiente Unix (per questo hanno nomi minuscoli) e sono gratuitamente disponibili su Internet, mentre strumenti analoghi sono disponibili anche per altri sistemi operativi.

Anche se questi sono soltanto due esempi, scaricare pagine dal Web e partecipare ad una videoconferenza dimostra la grande diversità di applicazioni che si possono allestire sulla rete Internet e suggerisce la complessità sottostante al progetto di Internet. Iniziando dalle prime cose e affrontando un problema alla volta, la parte restante di questo libro spiega come costruire una rete che renda possibile un insieme di applicazioni così variegato. Il Capitolo 9 conclude il libro tornando ad esaminare queste due specifiche applicazioni, insieme a parecchie altre che sono divenute popolari nella Internet di oggi.

1.2 Requisiti

Ci siamo appena prefissi un obiettivo ambizioso: capire come costruire da zero una rete di calcolatori. Il nostro approccio per la realizzazione di questo obiettivo sarà quello di iniziare dai principi primi, per poi porci quelle domande che ci porremo in modo naturale se stessimo costruendo una vera rete. Ad ogni passo useremo i protocolli attualmente in uso per illustrare le diverse opzioni di progetto a disposizione, ma non acetteremo queste realizzazioni esistenti come dogmatiche: al contrario, ci chiederemo (e risponderemo) perché le reti vengono progettate in questo modo. Nonostante sia forte la tentazione di cercare soltanto di capire come vengano oggi progettate le reti, è invece importante identificare i concetti basilari e fondamentali, perché le reti cambiano in continuazione con l'evoluzione della tecnologia e vengono introdotte applicazioni sempre nuove. La nostra esperienza ci insegna che, avendo

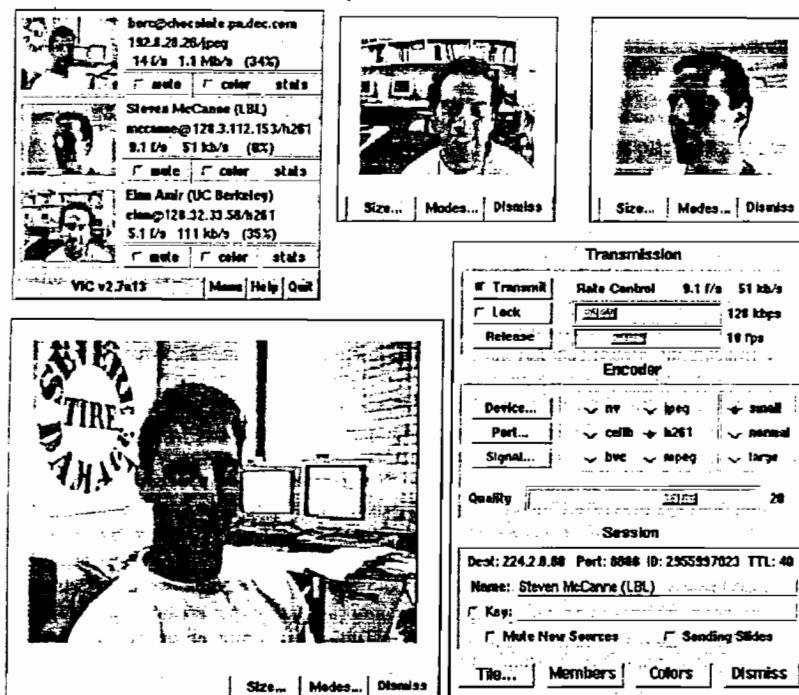


Figura 1.1 La video-applicazione vic.

compreso le idee fondamentali, sarà relativamente semplice comprendere ed assimilare qualsiasi nuovo protocollo che si presenti sulla scena.

Il primo passo consiste nell'identificare l'insieme di vincoli e requisiti che influenzano il progetto della rete. Prima di iniziare, però, è importante capire che ciò che ci aspettiamo da una rete dipende dal nostro punto di vista:

- Un *programmatore di applicazioni* farebbe un elenco dei servizi necessari per le sue applicazioni, come, ad esempio, la garanzia che ciascun messaggio inviato da un'applicazione venga consegnato privo di errori entro un certo periodo di tempo.
- Un *progettista di rete* farebbe un elenco delle proprietà che caratterizzano un progetto efficiente dal punto di vista dei costi, come, ad esempio, il fatto che le risorse di rete vengano utilizzate in modo efficiente e allocate in modo equo fra i vari utenti.
- Un *fornitore di rete* farebbe un elenco delle caratteristiche di un sistema facile da gestire e da amministrare, come, ad esempio, un sistema in cui i guasti possono essere facilmente localizzati e isolati, oppure un sistema di cui è semplice gestire la contabilità di utilizzo.

Questa sezione cerca di estrarre le caratteristiche essenziali di questi diversi punti di vista, per produrre una visione di alto livello delle più importanti proprietà che guidano il progetto di una rete: nel far questo, verranno identificati i punti salienti da affrontare nel resto del libro.

1.2.1 Connessione

Partendo dalle cose ovvie, una rete deve fornire la connessione tra un insieme di calcolatori. A volte è sufficiente costruire una piccola rete che connetta soltanto poche e selezionate macchine: per ragioni di riservatezza e sicurezza, molte reti private (aziendali) hanno l'obiettivo esplicito di limitare l'insieme di macchine connesse alla rete stessa. Al contrario, altre reti (di cui Internet è l'esempio principe) vengono progettate per crescere in modo da consentire loro la potenzialità di connettere tutti i computer del mondo: un sistema che sia progettato per consentire la propria crescita fino ad una dimensione arbitrariamente grande viene detto *scalabile*. Questo libro affronterà il problema della scalabilità usando Internet come modello.

Linee di connessione, nodi e nuvole

Le connessioni di rete avvengono a molti livelli diversi. Al livello inferiore, una rete può essere composta di due o più computer direttamente connessi da qualche mezzo fisico, come un cavo coassiale o una fibra ottica. Tale mezzo fisico viene chiamato *linea di connessione* (link), mentre i computer da esso connessi vengono spesso chiamati *nodi* (a volte un nodo è un elemento hardware più specializzato di un computer, ma per gli scopi di questa trattazione non considereremo questa distinzione). Come evidenziato in Figura 1.2, le connessioni fisiche sono a volte limitate ad una coppia di nodi (e in questo caso la linea di connessione viene detta *punto-punto*), mentre in altri casi uno stesso mezzo fisico può essere condiviso da più di due nodi (tale linea di connessione di dice *ad accesso multiplo*). Il fatto che una certa linea di connessione sia di tipo punto-punto oppure ad accesso multiplo dipende dal modo in cui i nodi vi sono collegati. Inoltre, spesso le linee di connessione ad accesso multiplo hanno una dimensione limitata, sia in termini di distanza geografica che in termini di numero di nodi che vi si possono connettere; un'eccezione in tal senso è costituita da un collegamento satellitare, che può coprire una vasta area geografica.

Se le reti di calcolatori fossero limitate alle situazioni in cui tutti i nodi sono direttamente connessi uno all'altro tramite un comune mezzo fisico, le reti sarebbero assai limitate nel numero di calcolatori che possono essere connessi, oppure il numero di cavi che spunterebbe dal retro di ciascun nodo diventerebbe ben presto ingestibile e molto costoso. Fortunatamente la possibilità di connessione tra due nodi non implica necessariamente una connessione

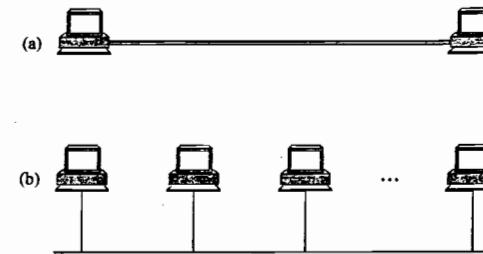


Figura 1.2 Linee di connessione dirette: (a) punto-punto; (b) ad accesso multiplo.

fisica diretta tra di essi: si può ottenere una connessione indiretta mediante la cooperazione di altri nodi. Considerate i due esempi seguenti, che illustrano come si possa connettere un insieme di calcolatori.

La Figura 1.3 mostra un insieme di nodi, ciascuno dei quali è collegato a una o più linee di connessione punto-punto. I nodi che risultano essere collegati ad almeno due linee eseguono delle applicazioni che inoltrano i dati ricevuti da una linea verso un'altra linea: se vengono organizzati in modo sistematico, questi nodi che inoltrano i dati formano una *rete commutata* (*switched network*). Esistono diversi tipi di reti commutate, le più comuni delle quali sono le *reti a commutazione di circuito* (*circuit switched networks*) e le *reti a commutazione di pacchetto* (*packet switched networks*). Le prime vengono utilizzate prevalentemente dal sistema telefonico, mentre le ultime costituiscono la stragrande maggioranza delle reti di calcolatori e su di esse si porrà l'attenzione in questo libro. La caratteristica fondamentale delle reti a commutazione di pacchetto prevede che i nodi di tale rete si scambino i dati a blocchi: immaginate questi blocchi di dati in analogia a segmenti di dati applicativi come un file, un messaggio di posta elettronica o un'immagine. Ciascuno di questi blocchi di dati viene chiamato *pacchetto* o *messaggio*; per il momento useremo questi due termini in modo equivalente, ma nella Sezione 1.2.2 vedremo come ciò non sia sempre vero.

Le reti a commutazione di pacchetto usano solitamente una strategia chiamata *memorizza-e-inoltra* (*store-and-forward*). Come suggerito dal nome, ciascun nodo in una rete store-and-forward riceve un pacchetto completo da una linea di connessione, lo memorizza nella propria memoria interna e quindi lo invia al nodo successivo. Al contrario, in una rete a commutazione di circuito viene per prima cosa allestito un circuito dedicato attraverso una sequenza di linee di connessione, per poi consentire al nodo sorgente l'invio di un flusso di bit al nodo destinazione attraverso tale circuito. Il motivo principale per cui nelle reti di calcolatori viene usata la commutazione di pacchetto invece della commutazione di circuito è l'efficienza, come vedremo fra poco.

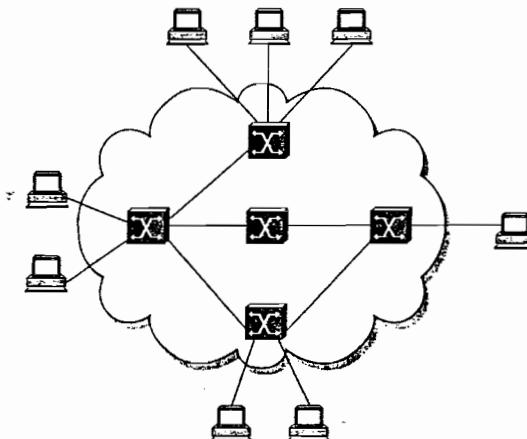


Figura 1.3 Rete commutata.

1.2 Requisiti

La nuvola di Figura 1.3 vuole evidenziare la differenza tra i nodi al suo interno, che *realizzano* la rete e vengono solitamente chiamati *switch* (commutatori), avendo la sola funzione di memorizzare e inoltrare pacchetti, e i nodi all'esterno, che *usano* la rete, vengono chiamati *host* e hanno il compito di interagire con gli utenti e di eseguire programmi applicativi. Notate anche come la nuvola di Figura 1.3 sia uno dei simboli più utilizzati nelle trattazioni di reti di calcolatori: solitamente usiamo una nuvola per indicare un tipo di rete qualsiasi, sia essa una connessione punto-punto o ad accesso multiplo, oppure una rete commutata. Ogni volta che vedete una nuvola in una figura, quindi, potete considerarla una rappresentazione di una rete realizzata con una qualsiasi delle tecnologie presentate in questo libro.

Un secondo modo per connettere indirettamente un insieme di calcolatori è indicato in Figura 1.4, dove alcune reti indipendenti (nuvole) sono tra loro interconnesse per formare una *internetwork*, o più brevemente *internet*. In questo libro adottiamo la convenzione usuale di riferirsi ad una generica interconnessione di reti con il termine *internet* con iniziale minuscola, usando il termine *Internet* con iniziale maiuscola per indicare la attuale rete mondiale operante con architettura TCP/IP. Un nodo che risulti essere connesso a due o più reti viene comunemente detto *router* (instradatore) o *gateway* e svolge all'incirca la stessa funzione di uno switch: inoltra messaggi da una rete ad un'altra. Notate che una internet può essa stessa essere vista come un tipo particolare di rete, per cui una internet può essere composta dall'interconnessione di diverse internet: in questo modo, possiamo costruire ricorsivamente reti arbitrariamente grandi, interconnettendo nuvole per formare una nuvola più grande.

Il semplice fatto di aver connesso l'uno all'altro un insieme di calcolatori non significa che siamo riusciti a fornire un servizio di connessione tra un host e l'altro: dobbiamo aggiungere il requisito tramite il quale la rete consenta a ciascun nodo di indicare quale sia l'altro

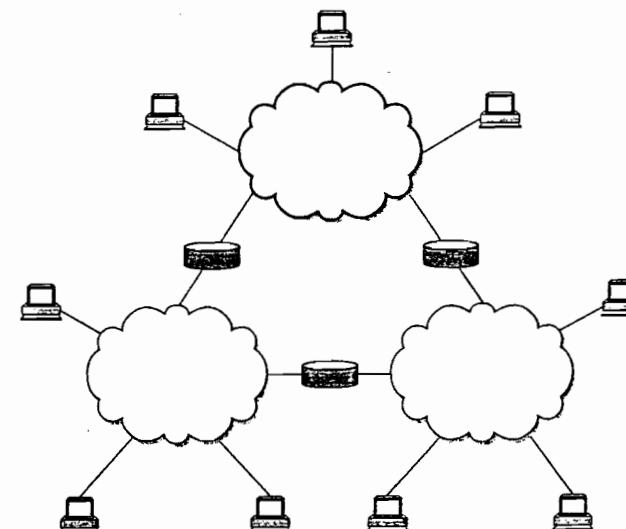


Figura 1.4 Reti interconnesse.

nodo della rete con il quale vuole comunicare. Ciò si realizza assegnando un *indirizzo* a ciascun nodo. Un indirizzo è una stringa di byte che identifica un nodo; ciò significa che la rete può utilizzare l'indirizzo di un nodo per distinguere tale nodo dagli altri nodi connessi alla rete stessa. Quando un nodo sorgente vuole che la rete consegna un messaggio ad un certo nodo destinazione, deve indicare l'indirizzo di tale nodo di destinazione; se il mittente e il destinatario non sono connessi direttamente, gli switch e i router della rete usano tale indirizzo per decidere in che modo inoltrare il messaggio verso la sua destinazione. Questo processo di determinazione sistematica del modo di inoltrare i messaggi verso il proprio nodo di destinazione basandosi sul suo indirizzo viene detto *instradamento (routing)*.

Questa breve presentazione dei problemi di indirizzamento e instradamento è stata condotta nell'ipotesi che il nodo sorgente voglia inviare un messaggio ad un singolo nodo destinazione (*unicast*, diffusione univoca). Anche se questo è lo scenario più comune, è però possibile che il nodo sorgente voglia diffondere un messaggio a tutti i nodi della rete (*broadcast*), oppure ad un sottoinsieme di tali nodi ma non a tutti (*multicast*). Di conseguenza, oltre agli indirizzi dei singoli nodi, la rete deve anche fornire un meccanismo di indirizzamento che consenta la comunicazione broadcast e multicast.

La cosa importante da ricordare riguardo a questa presentazione iniziale è che possiamo definire una *rete* in modo ricorsivo come un'entità formata da due o più nodi connessi da un mezzo fisico oppure da due o più reti connesse da un nodo. In altre parole, una rete può essere vista come un insieme di reti annidate, dove al livello inferiore la rete è realizzata da un supporto fisico. Una delle sfide centrali nella fornitura di connessione all'interno di una rete è la definizione di un indirizzo per ciascun nodo che sia raggiungibile nella rete stessa (con il supporto della trasmissione broadcast e multicast) e la capacità di usare tale indirizzo per instradare i messaggi verso i nodi di destinazione appropriati.

1.2.2 Condivisione di risorse efficiente dal punto di vista dei costi

Come appena detto, questo libro è dedicato alle reti a commutazione di pacchetto. Questa sezione, in particolare, illustra un requisito chiave per le reti di calcolatori, l'efficienza, che ci porta a considerare la commutazione di pacchetto come scelta vincente.

Assegnato un insieme di nodi connessi in modo indiretto da una rete, è possibile, per ciascuna coppia di host, scambiarsi messaggi tramite una sequenza di linee di connessione e di nodi. Ovviamente, è nostra ambizione poter fare di più rispetto ad un'unica coppia di host che comunicano: vogliamo fornire a tutte le coppie di host la possibilità di comunicare. La domanda quindi è: come attuare la condivisione della rete da parte di tutti gli host che vogliono comunicare, in particolar modo se lo vogliono fare tutti contemporaneamente? E, nel caso in cui il problema non sembri abbastanza complesso, come possono diversi host condividere la stessa linea di connessione quando tutti la vogliono utilizzare nello stesso momento?

Per capire come diversi host possano condividere una rete, dobbiamo introdurre il concetto fondamentale di *multiplexing* (multiplazione), che si riferisce alla condivisione tra più utenti di una risorsa di sistema. A livello intuitivo, il multiplexing si può spiegare in analogia con il timesharing (condivisione di tempo) usato nei calcolatori, dove una singola CPU è condivisa da più processi, ciascuno dei quali crede di essere l'unico utente del processore. In modo simile, i dati inviati da più utenti possono usare il multiplexing per condividere la stessa connessione fisica che costituisce una rete.

Per vedere come funziona questo meccanismo, considerate la semplice rete di Figura 1.5, dove i tre host di sinistra (L1-L3) stanno inviando dati ai tre host di destra (R1-R3) condividendo una rete commutata che contiene un solo collegamento fisico (per semplicità, ipotizzate che L1 stia comunicando con R1, e così via). In questa situazione, tre flussi di dati, corrispondenti alle tre coppie di host, vengono multiplati sull'unico collegamento fisico dallo switch 1, per poi essere *demultiplicati* in flussi indipendenti dallo switch 2. Notate che siamo stati intenzionalmente vaghi nel definire cosa significhi precisamente "flusso di dati": per il momento ipotizzate che ciascun host di sinistra abbia un'ingente quantità di dati da inviare alla sua controparte sulla destra.

Per realizzare il multiplexing di più flussi su un unico collegamento fisico esistono diversi metodi; uno dei più comuni è il *multiplexing sincrono a divisione di tempo* (synchronous time-division multiplexing, STDM). L'idea di base di STDM è quella di dividere il tempo in intervalli aventi tutta la stessa durata ed assegnare a ciascun flusso, a rotazione, la possibilità di inviare dati sulla connessione fisica. In altre parole, durante l'intervallo temporale 1 vengono trasmessi i dati del primo flusso, durante l'intervallo temporale 2 vengono trasmessi i dati del secondo flusso, e così via. Questo processo continua finché tutti i flussi hanno avuto un turno a disposizione: a quel punto, è nuovamente il turno del primo flusso, ed il processo viene ripetuto indefinitamente. Un altro metodo è il *multiplexing a divisione di frequenza* (frequency-division multiplexing, FDM), che è basato sull'idea di trasmettere ciascun flusso sulla linea di connessione fisica usando una diversa frequenza, praticamente nello stesso modo in cui i segnali di diversi canali televisivi vengono trasmessi a frequenze diverse sul cavo televisivo.

Nonostante siano semplici da capire, sia STDM che FDM sono duplicemente limitati. Innanzitutto, se uno dei flussi (cioè una coppia di host) non ha dati da inviare, la sua porzione del mezzo fisico, cioè l'intervallo temporale o la frequenza, rimane inutilizzata, anche se uno degli altri flussi ha dei dati da trasmettere. Nelle comunicazioni tra calcolatori, una linea può rimanere inutilizzata per questo motivo per un tempo totale molto lungo: considerate, ad esempio, il tempo che impiegate per leggere una pagina Web (lasciando inattivo il collegamento) confrontato con il tempo speso per recuperare la pagina stessa. Secondariamente, sia STDM che FDM sono limitati a situazioni in cui il massimo numero di flussi è fisso e noto a priori, in quanto è poco pratico ridimensionare l'intervallo di tempo o aggiungere intervalli nel caso di STDM, oppure aggiungere nuove frequenze nel caso di FDM.

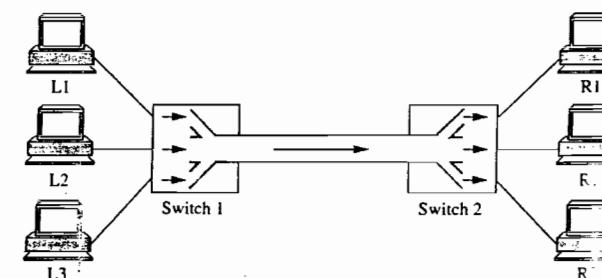


Figura 1.5 Multiplexing di più flussi logici su un unico collegamento fisico.

La modalità di realizzazione del multiplexing che useremo più spesso in questo libro viene chiamata *multiplexing statistico*. Nonostante il nome non sia per nulla di aiuto nella comprensione del concetto, il multiplexing statistico è, in realtà, piuttosto semplice e segue due idee basilari. Come STDM, il multiplexing statistico prevede la condivisione del mezzo fisico nel tempo: dapprima vengono trasmessi sulla linea i dati di un flusso, poi quelli di un altro flusso, e così via. Diversamente da STDM, però, i dati vengono trasmessi da ciascun flusso in base alla richiesta, piuttosto che durante un intervallo di tempo prefissato. In questo modo, se un solo flusso ha dati da trasmettere, li può trasmettere senza aspettare il proprio turno e, quindi, senza vedere trascorrere inutilizzati gli intervalli temporali assegnati agli altri flussi. La commutazione di pacchetto deriva la propria efficienza proprio dal fatto che si evita questo spreco di tempo.

Per come è stato definito fino ad ora, però, il multiplexing statistico non possiede alcun meccanismo per garantire a tutti i flussi di avere, prima o poi, un turno per trasmettere sul mezzo fisico. Una volta che un flusso ha iniziato ad inviare dati, abbiamo bisogno di un modo per interrompere la trasmissione, per dar spazio agli altri flussi. Per soddisfare questa esigenza, il multiplexing statistico definisce un limite superiore per la dimensione del blocco di dati che ciascun flusso può trasmettere ad un dato istante. Questo blocco di dati di dimensioni limitate viene comunemente chiamato *pacchetto*, per distinguerlo dal *messaggio*, arbitrariamente lungo, che un programma applicativo può voler trasmettere. Poiché una rete a commutazione di pacchetto pone un limite alla dimensione massima dei pacchetti stessi, può succedere che un host non riesca ad inviare un intero messaggio all'interno di un pacchetto: la sorgente può avere la necessità di frammentare il messaggio in diversi pacchetti, che il ricevente dovrà assemblare per ottenere il messaggio originario.

In altre parole, ciascun flusso invia una sequenza di pacchetti lungo la connessione fisica, mentre la decisione di quale flusso abbia il diritto di trasmettere viene presa dopo l'invio di ciascun pacchetto. Notate che se vi è un solo flusso che ha dati da trasmettere lo può fare inviando i propri pacchetti in successione, mentre se vi sono più flussi attivi i loro pacchetti viaggiano intercalati lungo la linea di connessione. La Figura 1.6 raffigura uno switch che

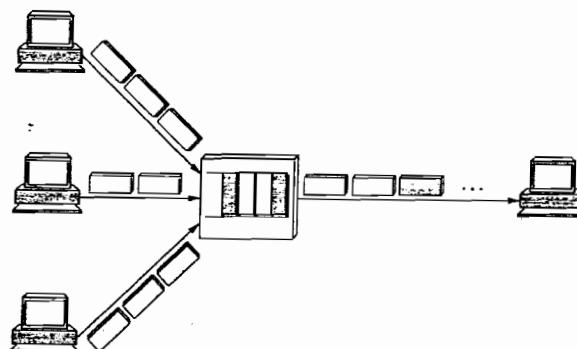


Figura 1.6 Uno switch che effettua il multiplexing di pacchetti provenienti da più sorgenti su una singola linea di connessione condivisa.

1.2 Requisiti

effettua il multiplexing di pacchetti provenienti da più sorgenti su una singola linea di connessione condivisa.

La decisione di quale sia il successivo pacchetto da inviare su una linea condivisa può essere presa in diversi modi. Ad esempio, in una rete composta da switch interconnessi da collegamenti come indicato nella Figura 1.5, la decisione verrebbe presa dallo switch che trasmette i pacchetti sul collegamento (come vedremo in seguito, non tutte le reti a commutazione di pacchetto richiedono la presenza di switch e si possono usare altri metodi per determinare quale sia il successivo pacchetto da inviare). Ciascuno switch in una rete a commutazione di pacchetto prende le proprie decisioni in modo autonomo, prima di inviare ciascun pacchetto. E uno dei problemi che devono essere affrontati da un progettista di rete consiste nel fare in modo che le decisioni siano equi. Per esempio, uno switch potrebbe essere progettato per gestire i pacchetti secondo una politica FIFO (*first-in first-out*, il primo arrivato è il primo ad uscire). Un diverso approccio potrebbe essere quello di gestire i diversi flussi a rotazione, come abbiamo visto succedere in STDM. Ciò potrebbe avere lo scopo di garantire ai vari flussi una particolare frazione dell'ampiezza di banda disponibile sul collegamento, oppure di fare in modo che nessun flusso veda i propri pacchetti ritardati dallo switch per un periodo

SAN, LAN, MAN e WAN

Un modo per classificare le reti sfrutta la loro dimensione. Due esempi ben noti sono le LAN (*local area network*, rete locale) e le WAN (*wide area network*, reti di ampia estensione): le prime hanno un'estensione tipica di meno di un chilometro, mentre le ultime possono estendersi anche su tutto il globo terrestre. Altre reti vengono chiamate MAN (*metropolitan area network*, reti metropolitane) e si estendono tipicamente per alcune decine di chilometri. Il motivo per cui questa classificazione è interessante risiede nel fatto che spesso la dimensione di una rete ha implicazioni sulla tecnologia che può essere impiegata: uno dei fattori chiave è il tempo necessario ai dati per propagarsi da un capo all'altro della rete, e ne parleremo nei capitoli successivi.

Una nota storica interessante riguarda il termine "wide area network", che non fu inizialmente usato per la prima WAN perché non esistevano altri tipi di rete da cui differenziarla. Quando i calcolatori erano incredibilmente rari e costosi, non aveva senso pensare a come connettere tutti i computer di un'area localizzata, dato che vi era un solo calcolatore. Soltanto quando i computer hanno iniziato a proliferare e si resero necessarie le LAN, venne introdotto il termine "WAN" per descrivere le grandi reti che interconnettono calcolatori geograficamente lontani.

Un altro tipo di reti che dobbiamo conoscere sono le SAN (*system area network*, rete di sistema), che sono solitamente confinate all'interno di un'unica stanza e collegano i vari componenti di un grande sistema di calcolo. Ad esempio, HiPPI (High Performance Parallel Interface) e Fiber Channel sono due tecnologie per SAN molto utilizzate per connettere processori a parallelismo massivo con server di memorizzazione dati (per questo motivo, cioè perché spesso connettono i calcolatori a server di memorizzazione dati, le SAN vengono a volte definite *storage area network*). Anche se questo libro non descrive queste reti in dettaglio, è bene saperne qualcosa perché spesso hanno le prestazioni più spinte del momento, ed anche perché diventa sempre più comune la connessione di tali reti con LAN e WAN.

maggiori di una certa quantità di tempo. Se una rete consente ai flussi di richiedere tali prestazioni, si dice che la rete supporta la *qualità di servizio* (quality of service, QoS).

Ancora, notate nella Figura 1.6 che, dovendo lo switch fare il multiplexing di tre flussi di pacchetti entranti verso un collegamento uscente, è possibile che lo switch riceva i pacchetti più velocemente di quanto il collegamento li possa smaltire. In questo caso lo switch è costretto a conservare tali pacchetti nella propria memoria interna: se lo switch ricevesse i pacchetti più velocemente di quanto li possa inviare per un periodo di tempo prolungato, esaurirebbe lo spazio di memoria disponibile ed alcuni pacchetti dovrebbero essere eliminati. Quando uno switch si trova ad operare in queste condizioni, lo si dice *congestionato*.

In sintesi, il multiplexing statistico definisce un modo efficace, dal punto di vista dei costi, per consentire a più utenti (cioè, ad esempio, a più flussi di dati) di condividere risorse di rete (linee di connessione e nodi) con un grado di risoluzione elevato. L'elemento atomico con cui le linee di connessione della rete vengono assegnate ai diversi flussi è il pacchetto ed ogni switch è in grado di gestire l'uso dei collegamenti fisici a cui è connesso sulla base dei singoli pacchetti. Gli obiettivi principali del multiplexing statistico sono la distribuzione equa tra diversi flussi della capacità trasmissiva dei collegamenti fisici e la gestione della congestione, qualora essa sopravvenga.

1.2.3 Supporto di servizi comuni

Anche se la sezione precedente ha delineato i problemi da risolvere per fornire la connessione di un gruppo di host in modo efficiente dal punto di vista dei costi, considerare una rete come uno strumento semplicemente dedicato alla consegna di pacchetti scambiati tra diversi calcolatori è una visione oltremodo semplicistica. Molto più corretto è pensare ad una rete come ad uno strumento che fornisce a più processi applicativi, distribuiti su più calcolatori, un modo per comunicare. In altre parole, il successivo requisito per una rete di calcolatori è che i programmi applicativi in esecuzione sugli host connessi alla rete possano comunicare in modo sensato.

Quando due programmi applicativi hanno bisogno di comunicare fra loro, parecchi fenomeni complessi devono accadere, al di là del semplice invio di un messaggio da un host ad un altro. Un'opzione praticabile per i progettisti di applicazioni sarebbe quella di inserire tutte queste complesse funzionalità all'interno dei loro programmi, ma, poiché molte applicazioni hanno necessità di servizi simili, è molto più razionale realizzare questi servizi una volta soltanto e fare in modo che i progettisti di applicazioni possano costruire i loro programmi utilizzando tali servizi. Un obiettivo ambizioso per un progettista di rete è l'identificazione del corretto insieme di servizi comuni, per nascondere alle applicazioni la complessità della rete senza vincolare inutilmente l'operato del progettista di applicazioni.

Da un punto di vista intuitivo, si può pensare alla rete come fornitrice di *canali logici* attraverso i quali i processi a livello applicativo possono comunicare fra loro; ciascun canale fornisce l'insieme di servizi richiesto da una specifica applicazione. In altre parole, così come usiamo una nuvola per rappresentare in astratto la connessione di un insieme di calcolatori, allo stesso modo possiamo ora pensare ad un canale come ad un elemento astratto che connette un processo ad un altro. La Figura 1.7 mostra una coppia di processi a livello applicativo che comunicano attraverso un canale logico che, a sua volta, è realizzato mediante una nuvola

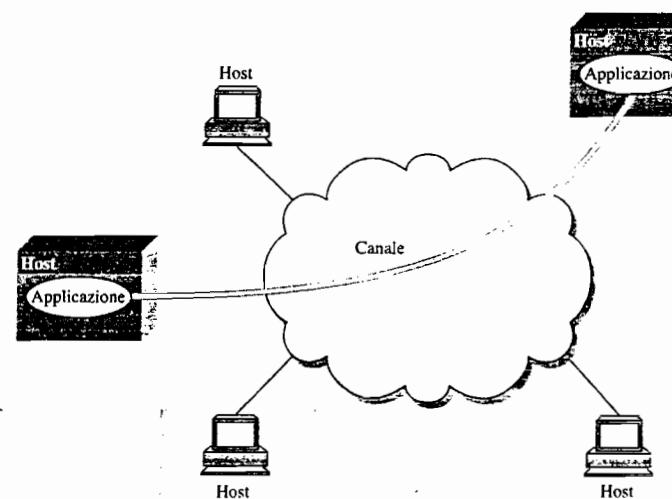


Figura 1.7 Processi che comunicano mediante un canale astratto.

che connette un insieme di host. Possiamo pensare al canale come ad una condutture (*pipe*) che connette due applicazioni, in modo che un'applicazione sorgente possa inserire dati ad un estremo della condutture ed attendersi che i dati vengano trasferiti dalla rete fino all'altro estremo.

L'obiettivo, a questo punto, consiste nell'identificare quali siano le funzionalità che i canali devono fornire ai programmi applicativi. Ad esempio, l'applicazione richiede la garanzia che i messaggi inviati lungo il canale siano arrivati a destinazione, oppure è accettabile che qualche messaggio non arrivi? I messaggi devono necessariamente giungere al processo destinatario nello stesso ordine in cui sono stati spediti, oppure al destinatario non interessa tale ordine? La rete deve garantire che nessun terzo incomodo sia in grado di origliare sul canale, oppure la riservatezza non è importante? In generale, una rete mette a disposizione una varietà di tipi di canale, e ciascuna applicazione può scegliere quello che meglio soddisfa i propri requisiti. La parte restante di questa sezione illustra i ragionamenti che fanno emergere le definizioni dei canali più utili.

Identificare gli schemi di comunicazione comuni

La progettazione di canali astratti richiede, per prima cosa, la comprensione delle esigenze di comunicazione di un insieme rappresentativo di applicazioni, quindi l'estrazione delle loro esigenze comuni ed, infine, l'introduzione nella rete delle funzionalità che soddisfano tali requisiti.

Una delle prime applicazioni supportate da qualsiasi rete è rappresentata da programmi di accesso ai file, come FTP (File Transfer Protocol) o NFS (Network File System). Sebbene si differenzino per molti dettagli (come, ad esempio, il fatto che i file vengono trasferiti attraverso la rete, oppure che soltanto singoli blocchi del file vengono letti/scritti in un certo

istante). la componente di comunicazione per l'accesso a file remoti è caratterizzata da una coppia di processi, uno che richiede che un file venga letto o scritto ed un secondo processo che onora tale richiesta. Il processo che richiede l'accesso al file viene chiamato *client*, mentre il processo che fornisce l'accesso al file viene detto *server*.

La lettura di un file implica che il client invii un breve messaggio di richiesta al server e che il server risponda con un lungo messaggio che contiene i dati presenti nel file. La scrittura funziona in modo opposto: il client invia al server un lungo messaggio contenente i dati che devono essere scritti ed il server risponde con un breve messaggio per confermare che la scrittura sul disco abbia avuto luogo. Una biblioteca digitale, di cui il World Wide Web è un esempio, è un altro processo che funziona in modo simile: un processo client invia una richiesta ed un processo server risponde, fornendo i dati richiesti.

Usando come esempi rappresentativi l'accesso ai file, la biblioteca digitale e le due applicazioni video descritte nell'introduzione (videoconferenza e video-on-demand), potremo decidere di fornire due tipi di canali: canali *richiesta/risposta* (request/reply) e canali *a flusso di messaggi* (message stream). Il canale richiesta/risposta verrebbe usato per il trasferimento di file e per le applicazioni di biblioteca digitale, e garantirebbe che ciascun messaggio spedito da un lato venga ricevuto dall'altro e che venga consegnata una sola copia di ciascun messaggio. Tale canale potrebbe anche garantire la riservatezza e l'integrità dei dati che vi fluiscono, in modo che agenti non autorizzati non possano leggere o modificare i dati che vengono scambiati tra i processi client e server.

Il canale a flusso di messaggi potrebbe essere utilizzato sia per il video-on-demand, sia per le applicazioni di videoconferenza, a patto di essere configurabile per il traffico in una sola o in entrambe le direzioni, nonché per fornire diverse caratteristiche di ritardo. A tale canale potrebbe non essere richiesto di garantire la consegna di tutti i messaggi, poiché un'applicazione video può funzionare in modo adeguato anche se alcuni quadri dell'immagine non vengono ricevuti, mentre dovrebbe garantire che i messaggi vengano consegnati nell'ordine in cui sono stati inviati, per evitare la visualizzazione di quadri fuori sequenza. Come il canale di tipo richiesta/risposta, anche il canale a flusso di messaggi potrebbe garantire la riservatezza e l'integrità dei dati. Infine, il canale a flusso di messaggi potrebbe dover supportare il multicast, in modo che più utenti possano partecipare ad una teleconferenza o visualizzare il filmato.

Nonostante la ricerca del minor numero di canali astratti che possono servire per il maggior numero di applicazioni sia un obiettivo tipico del progettista di reti, cercare di iniziare un progetto con troppo pochi canali può essere pericoloso. Detto molto semplicemente, se il vostro unico strumento è un martello, quasi tutto assomiglia a un chiodo. Ad esempio, se avete a disposizione soltanto i canali a flusso di messaggi e richiesta/risposta, avrete la tentazione di usarli anche per le applicazioni future, anche se nessuno di questi canali ne possiede la semantica corretta. Per questo motivo, probabilmente i progettisti di reti continueranno ad inventare nuovi tipi di canali (e ad aggiungere opzioni ai canali esistenti), almeno finché i progettisti di applicazioni ne inventeranno di nuove.

Noteate anche che, indipendentemente da *quale* funzionalità venga fornita da un certo canale, esiste il problema di *dove* tale funzionalità venga implementata. In molti casi la cosa più semplice è immaginare la connessione tra due host fornita dalla rete sottostante come una semplice *bit pipe* (condutture per bit), con una semantica di comunicazione di alto livello fornita dagli host terminali. Il vantaggio di questo approccio consiste nel rendere più semplici gli switch interni alla rete, che hanno semplicemente il compito di inoltrare pacchetti, ma appesantisce di conseguenza il lavoro degli host terminali, che devono fornire il supporto ai

canali fra i processi e alla loro ricca semantica. L'alternativa richiede di introdurre ulteriori funzionalità negli switch, rendendo possibile l'utilizzo di host terminali "stupidi" (dumb), come un apparecchio telefonico. Nel progetto di reti, affronteremo varie volte questo problema, di come diversi servizi di rete vengano suddivisi tra gli host terminali e i commutatori di pacchetti.

Affidabilità

Come suggerito dagli esempi appena considerati, l'affidabilità della consegna dei messaggi è una delle funzioni più importanti tra quelle fornite da una rete: tuttavia, è difficile decidere come fornire questa affidabilità, senza prima comprendere come una rete possa fallire. La prima cosa da capire è che le reti di calcolatori non agiscono in un mondo perfetto. I computer interrompono bruscamente il loro funzionamento e poi vengono fatti ripartire, le fibre vengono tagliate, le interferenze elettromagnetiche modificano i bit nei dati che vengono trasmessi, gli switch esauriscono il proprio spazio di memorizzazione, e se questi problemi di carattere fisico non fossero sufficienti, il software che gestisce il tutto inoltra, a volte, i pacchetti verso l'ignoto. Per questi motivi, uno dei principali requisiti di una rete è il mascheramento di alcuni tipi di malfunzionamento, in modo da far apparire la rete più affidabile di quanto in realtà essa non sia alle applicazioni che la usano.

Esistono tre classi generiche di malfunzionamenti di cui si devono preoccupare i progettisti di reti. Innanzitutto, mentre un pacchetto viene trasmesso lungo una linea fisica di connessione, *errori di bit* possono essere introdotti nei dati, cioè un valore 1 viene trasformato in un valore 0 o viceversa. A volte vengono modificati singoli bit, ma più spesso accade un errore prolungato (*burst*) nel quale vengono modificati più bit consecutivi. Gli errori di bit tipicamente accadono perché forze esterne, come sorgenti luminose, generatori elettrici o fornì a microonde, interferiscono con la trasmissione dei dati. La buona notizia è che tali errori di bit sono piuttosto rari e colpiscono in media soltanto un bit ogni 10^6 o 10^7 bit in un comune cavo in rame, e un bit ogni 10^{12} o 10^{14} bit in una normale fibra ottica. Come vedremo, esistono tecniche che individuano con elevata probabilità questi errori: dopo averli individuati, a volte è anche possibile correggerli (se sappiamo quali bit sono stati modificati, è sufficiente invertire il valore), mentre in altri casi il danno è così rilevante che è necessario eliminare l'intero pacchetto. In tal caso, è possibile che al mittente venga richiesto di inviare nuovamente il pacchetto.

La seconda classe di malfunzionamenti considera gli errori a livello di pacchetto, anziché di bit: un intero pacchetto viene perduto dalla rete. Un motivo per cui ciò può accadere è che il pacchetto contenga un errore di bit non recuperabile e che, quindi, venga eliminato. Una ragione più probabile, però, è che uno dei nodi che deve gestire il pacchetto (ad esempio, uno switch che lo sta inoltrando da una linea di connessione ad un'altra) sia così sovraccarico da non avere più spazio per memorizzare il pacchetto e, quindi, lo elimini: questo problema è stato chiamato congestione ed è stato introdotto nella Sezione 1.2.2. Meno frequentemente, il software in esecuzione in uno dei nodi che gestisce il pacchetto può compiere un errore: ad esempio, potrebbe erroneamente inoltrare un pacchetto sulla linea di connessione sbagliata, in modo che il pacchetto non possa mai trovare la sua corretta destinazione. Come vedremo, una delle difficoltà principali nella gestione dei pacchetti perduti consiste nel poter distinguere tra un pacchetto veramente perduto e uno che è semplicemente arrivato a destinazione in ritardo.

La terza categoria di malfunzionamenti avviene a livello di nodo e di linea di collegamento: una linea fisica è interrotta, oppure il computer a cui è connessa non funziona. Ciò

può essere causato da problemi software, oppure da un guasto all'alimentazione o da un operatore poco attento. Anche se questi guasti vengono risolti, possono avere un impatto fortemente negativo sulla rete per un tempo prolungato. Ciò nonostante, non devono rendere completamente inutilizzabile la rete. Ad esempio, in una rete a commutazione di pacchetto è a volte possibile aggirare un nodo o una linea guasti. Una delle difficoltà concernenti questa terza classe di malfunzionamenti è la distinzione tra un computer guasto ed uno che è semplicemente lento, oppure, nel caso di una linea di collegamento, tra un mezzo fisico che è stato interrotto e uno che introduce un elevato numero di errori di bit.

L'idea chiave da estrapolare da questa presentazione è che la definizione di canali utili coinvolge sia la comprensione dei requisiti delle applicazioni, sia l'identificazione dei limiti della tecnologia sottostante. L'obiettivo è quello di colmare il divario fra ciò che si attende l'applicazione e ciò che può essere fornito dalla tecnologia sottostante, un divario che viene talvolta chiamato *divario semantico*.

1.3 Architettura di rete

Nel caso non l'abbiate notato, la sezione precedente ha articolato un insieme abbastanza nutrito di requisiti per il progetto di reti: una rete di calcolatori deve fornire una connessione generica, efficace dal punto di vista dei costi, equa e robusta per un grande numero di calcolatori. Se ciò non fosse abbastanza difficile, le reti non rimangono fisse così come sono progettate in un certo istante, ma devono poter evolvere per incorporare modifiche sia nella tecnologia sottostante su cui sono basate sia nelle richieste messe in atto dai programmi applicativi. Progettare una rete per soddisfare questi requisiti non è cosa facile.

Per aiutarci nella gestione di questa complessità, i progettisti di reti hanno sviluppato delle linee guida generali, solitamente chiamate *architetture di rete*, che governano il progetto e la realizzazione di reti. Questa sezione definisce con maggior cura ciò che intendiamo per architettura di rete, presentando le idee basilari, comuni a tutte le architetture di rete, e presenta anche le architetture di rete più diffuse: l'architettura OSI e l'architettura Internet.

1.3.1 Stratificazione e protocolli

All'aumentare della complessità dei sistemi, i progettisti solitamente introducono un nuovo livello di astrazione. Il concetto di astrazione consiste nella definizione di un modello unificante che sia in grado di cogliere alcuni aspetti importanti del sistema, incapsulare tale modello in un oggetto dotato di un'interfaccia che possa essere manipolata dagli altri componenti del sistema e nascondere agli utenti degli oggetti i dettagli di come tali oggetti siano realizzati. La cosa complicata è l'identificazione di astrazioni che forniscano un servizio che si riveli utile in un gran numero di situazioni e che al tempo stesso possa essere realizzato in modo efficiente all'interno del sistema sottostante. Questo è proprio ciò che abbiamo fatto quando, nella sezione precedente, abbiamo introdotto il concetto di canale: abbiamo fornito un'astrazione per i progettisti di applicazioni, che nasconde loro la complessità della rete.

Le astrazioni conducono in modo naturale alla stratificazione, in particolar modo nei sistemi di rete. L'idea generale consiste nel partire dai servizi offerti dagli apparati hardware sottostanti, per poi aggiungere una serie di livelli o strati, ciascuno dei quali fornisce un

servizio di livello più elevato (cioè più astratto); i servizi forniti ai livelli soprastanti sono realizzati utilizzando i servizi forniti dai livelli sottostanti. Traendo le opportune deduzioni dalla discussione dei requisiti che abbiamo delineato nella sezione precedente, ad esempio, potremmo immaginare una rete composta di due livelli di astrazione, inseriti tra i programmi applicativi e l'hardware sottostante, come evidenziato in Figura 1.8. Lo strato immediatamente soprastante all'hardware, in questo caso, fornisce la connessione tra host, astraendo dal fatto che possa esistere una topologia di rete arbitrariamente complessa tra due host qualsiasi. Il livello superiore ad esso si basa sul servizio di comunicazione tra host, così disponibile, e fornisce il supporto per i canali tra processi, nascondendo il fatto che la rete possa, ad esempio, perdere qualche messaggio di quando in quando.

La stratificazione ha due caratteristiche positive. Innanzitutto, scomponete il problema di costruire una rete in sottoproblemi più gestibili: invece di realizzare un componente software monolitico che svolga qualsiasi funzione potremmo mai desiderare, si possono realizzare diversi strati, ciascuno dei quali risolve una parte del problema. Secondariamente, la stratificazione consente una progettazione più modulare: se decidete che volete aggiungere qualche nuovo servizio, può darsi che dobbiate modificare le funzionalità di un solo livello, riutilizzando i servizi forniti dagli altri livelli.

Pensare ad un sistema come ad una sequenza lineare di strati è una semplificazione eccessiva: molte volte ad alcuni livelli del sistema si trovano astrazioni multiple, ciascuna identificata per fornire un diverso servizio agli strati superiori, ma realizzata a partire dalle stesse astrazioni di più basso livello. Per averne un esempio, considerate i due tipi di canale presentati nella Sezione 1.2.3: uno fornisce un servizio di tipo richiesta/risposta, mentre l'altro fornisce un servizio a flusso di messaggi. Questi due canali potrebbero essere offerti in alternativa ad un certo livello di un sistema di rete a strati, come si può vedere in Figura 1.9.



Figura 1.8 Esempio di un sistema di rete a strati.

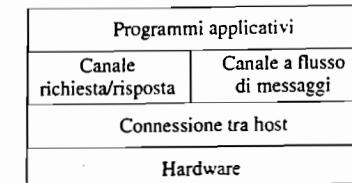


Figura 1.9 Sistema a strati con astrazioni alternative disponibili ad un certo livello.

Usando come base questa discussione della stratificazione, siamo ora pronti per affrontare con maggior precisione il tema delle architetture di rete. Per iniziare, gli oggetti astratti che compongono gli strati di un sistema di rete vengono chiamati *protocolli*: un protocollo, cioè, fornisce un servizio di comunicazione che gli oggetti del livello superiore (come i processi applicativi o i protocolli di più alto livello) usano per scambiarsi messaggi. Ad esempio, potremmo immaginare una rete dotata di un protocollo richiesta/risposta e di un protocollo a flusso di messaggi, corrispondentemente ai canali di tipo richiesta/risposta e a flusso di messaggi discusso in precedenza.

Ciascun protocollo definisce due diverse interfacce. La prima, detta *interfaccia del servizio*, è rivolta agli altri oggetti che, sullo stesso calcolatore, vogliono usare i servizi di comunicazione del protocollo stesso. Questa interfaccia del servizio definisce le operazioni che gli oggetti locali possono compiere sul protocollo: ad esempio, un protocollo di tipo richiesta/risposta potrebbe supportare le operazioni mediante le quali un'applicazione può inviare e ricevere messaggi. La seconda, detta *interfaccia di livello* o *interfaccia peer-to-peer*, è rivolta alla controparte (peer) del protocollo sull'altro calcolatore. Questa seconda interfaccia definisce la forma ed il significato dei messaggi scambiati tra le controparti del protocollo per realizzare il servizio di comunicazione, determinando così il modo in cui, ad esempio, un protocollo di tipo richiesta/risposta comunica con la sua controparte. In altre parole, un protocollo definisce un servizio di comunicazione che rende disponibile localmente, insieme ad un insieme di regole per gestire lo scambio di messaggi che il protocollo attua con la sua controparte per realizzare tale servizio. Questa situazione è evidenziata dalla Figura 1.10.

Tranne che al livello hardware, dove le controparti comunicano direttamente fra loro mediante una linea di connessione, la comunicazione tra entità di pari livello è indiretta: ciascun protocollo comunica con la propria controparte inviando messaggi ad un protocollo di livello inferiore, il quale a sua volta consegna il messaggio alla *propria* controparte. Inoltre, a ciascun livello esistono in generale molti protocolli, ciascuno avendo il compito di fornire un diverso servizio di comunicazione. Per questo motivo rappresentiamo con un *grafo di protocolli* l'insieme di protocolli che costituiscono un'architettura di rete: i nodi del grafo corrispondono ai protocolli, mentre i rami rappresentano una relazione del tipo *dipendente da*. Ad esempio, la Figura 1.11 mostra un grafo di protocolli per l'ipotetico sistema a strati

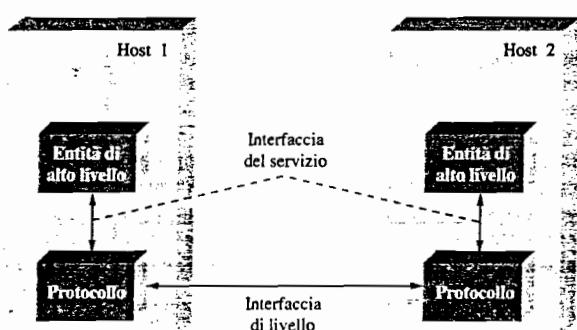


Figura 1.10 Interfaccia del servizio e interfaccia di livello.

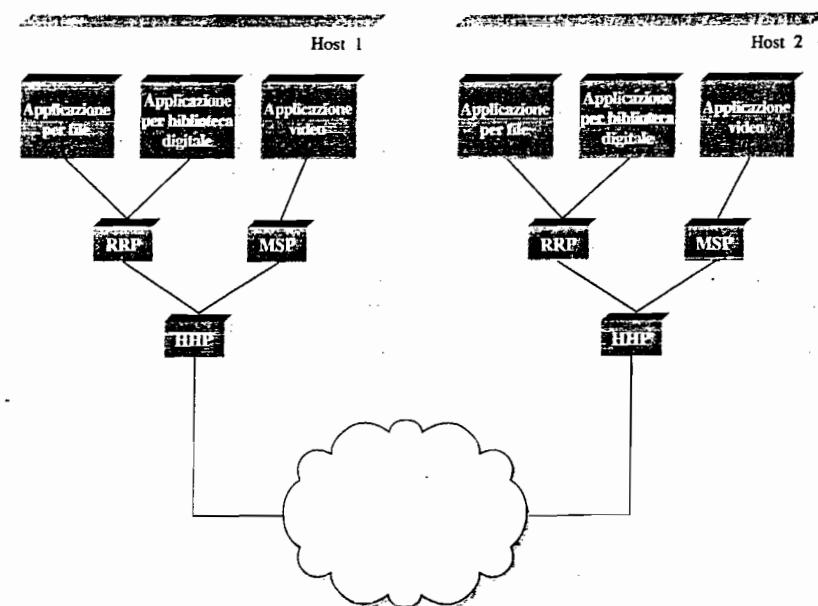


Figura 1.11 Esempio di grafo di protocolli.

di cui abbiamo discusso fino ad ora: i protocolli RRP (*Request/Reply Protocol*, protocollo richiesta/risposta) e MSP (*Message Stream Protocol*, protocollo a flusso di messaggi) realizzano due diversi tipi di canale tra processi e dipendono entrambi dal protocollo HHP (*Host-to-Host Protocol*), che fornisce un servizio di comunicazione tra host.

In questo contesto, supponete che il programma di accesso ai file in esecuzione su host 1 voglia inviare un messaggio alla sua controparte su host 2 usando il servizio di comunicazione offerto dal protocollo RRP, incaricando quindi RRP di inviare il messaggio. Per comunicare con la propria controparte, RRP invoca poi il servizio di HHP, che a sua volta trasmette il messaggio alla propria controparte su host 2. Una volta che il messaggio è giunto al protocollo HHP su host 2, HHP lo inoltra verso l'alto a RRP, il quale a sua volta consegna il messaggio all'applicazione che gestisce i file. In questo caso particolare, si dice che l'applicazione utilizza i servizi della pila di protocolli (*protocol stack*) RRP/HHP.

Si noti che il termine *protocollo* viene utilizzato in due modi diversi: a volte si riferisce alle interfacce astratte, cioè alle operazioni definite dall'interfaccia del servizio e alla forma e significato dei messaggi scambiati tra entità di pari livello, altre volte si riferisce al modulo che realizza concretamente queste due interfacce. Per fare una distinzione tra le interfacce ed il modulo che le realizza, solitamente ci riferiamo alle prime come alla *definizione del protocollo*. Le definizioni dei protocolli sono generalmente espresse con una combinazione di prosa, pseudocodice, diagrammi di transizione di stato, illustrazioni del formato dei pacchetti e altre notazioni astratte. Potrebbe anche accadere che un certo protocollo venga realizzato in

modi diversi da programmatore differenti: ciò è ammissibile, fintanto che ciascuna implementazione è conforme alla definizione del protocollo. L'obiettivo, infatti, è quello di garantire la possibilità dello scambio di messaggi tra due diverse realizzazioni della stessa definizione di protocollo: due o più moduli che siano in grado di fare ciò si dicono *interoperabili*.

Per soddisfare i requisiti di comunicazione di un insieme di applicazioni è possibile immaginare molti diversi protocolli e relativi grafi: fortunatamente esistono enti di standardizzazione, come ISO (International Standards Organization) e IETF (Internet Engineering Task Force), che determinano le strategie per un particolare grafo di protocolli. Questo insieme di regole che stabilisce la forma e il contenuto di un grafo di protocolli viene detto *architettura di rete*. Anche se non rientra tra gli argomenti trattati in questo libro, segnaliamo che gli enti di standardizzazione, quali ISO e IETF, hanno stabilito procedure ben definite per introdurre, valutare e conseguentemente approvare nuovi protocolli nelle loro rispettive architetture. Fra breve descriveremo sinteticamente le architetture definite da ISO e da IETF, ma prima dobbiamo illustrare ancora due argomenti relativi al funzionamento dei grafi di protocolli.

Incapsulamento

Considerate cosa accade in Figura 1.11 quando uno dei programmi applicativi invia un messaggio alla sua controparte, consegnando il messaggio al protocollo RRP. Dal punto di vista di RRP, il messaggio che gli viene consegnato dall'applicazione è una stringa di byte priva di significato. Ad RRP non interessa che questi byte rappresentino un array di numeri interi, un messaggio di posta elettronica, un'immagine digitale o altro: semplicemente, ha ricevuto l'incarico di consegnare il messaggio alla propria controparte. In ogni caso, comunque, RRP deve comunicare alla propria controparte alcune informazioni di controllo, per segnalare in che modo vada gestito il messaggio una volta che esso sia stato ricevuto: ciò viene fatto allegando un'*intestazione* (header) al messaggio. Parlando in generale, un'intestazione è una piccola struttura di dati (da pochi byte a poche dozzine di byte) che viene usata fra le controparti di pari livello per comunicare fra loro. Come suggerito dal nome, le intestazioni vengono solitamente anteposte al messaggio, ma in alcuni casi vengono invece inserite al termine del messaggio stesso (assumendo il ruolo di appendici, *trailer*). La forma esatta dell'intestazione usata da RRP è definita dalla definizione del protocollo stesso. Il resto del messaggio (cioè i dati che vengono trasmessi per incarico dell'applicazione) prendono il nome di *corpo* (body) o *carico utile* (payload) del messaggio. Diciamo, quindi, che i dati dell'applicazione vengono *incapsulati* nel nuovo messaggio creato dal protocollo RRP.

Questo processo di incapsulamento viene ripetuto ad ogni livello del grafo di protocolli; ad esempio, HHP incapsula il messaggio RRP aggiungendo una propria intestazione. Se ipotizziamo ora che HHP invii il messaggio alla propria controparte usando una rete, quando il messaggio arriva al calcolatore destinatario verrà elaborato seguendo un ordine di operazioni inverso: per prima cosa HHP elimina la propria intestazione dal messaggio, lo interpreta (cioè agisce in modo appropriato, in base al contenuto dell'intestazione) e inoltra il corpo del messaggio ad RRP, il quale rimuove l'intestazione che era stata allegata dalla propria controparte, intraprende le azioni conseguenti a quanto indicato in tale intestazione e inoltra il corpo del messaggio al programma applicativo. Il messaggio che viene trasmesso da RRP all'applicazione su host 2 è identico a quello trasmesso dall'applicazione ad RRP su host 1, e l'applicazione non vede le intestazioni che vi sono state allegate per realizzare i servizi di comunicazione di livello inferiore. Questo intero processo è evidenziato dalla Figura 1.12. Notate che in questo esempio i nodi della rete (switch e router) sono in grado di ispezionare l'intestazione HHP che si trova nel messaggio.

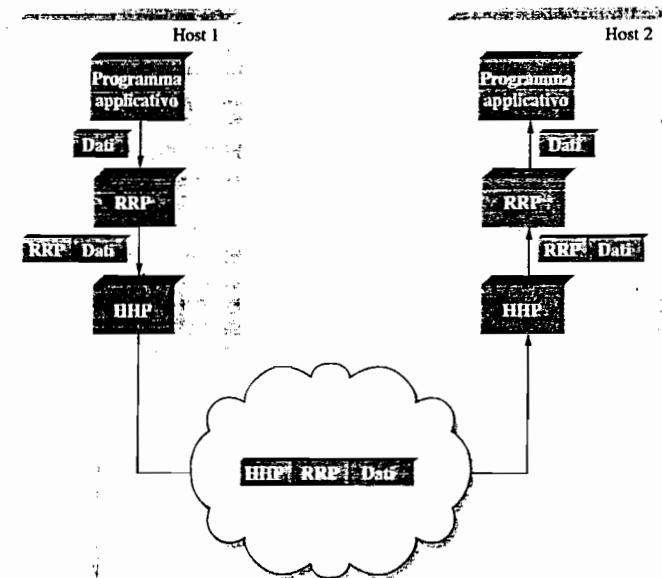


Figura 1.12 I messaggi di alto livello vengono incapsulati in messaggi di basso livello.

Si noti che, dicendo che un protocollo di basso livello non interpreta il messaggio che gli viene trasmesso da un protocollo di livello superiore, intendiamo dire che non è in grado di estrarre alcuna informazione dai dati contenuti nel messaggio stesso. A volte, però, il protocollo di basso livello può applicare alcune semplici trasformazioni ai dati che gli vengono forniti, come, ad esempio, una compressione o una codifica crittografica. In questo caso il protocollo trasforma l'intero corpo del messaggio, che comprende sia i dati originari dell'applicazione, sia tutte le intestazioni indicate a tali dati dai protocolli di livello superiore.

Multiplexing e demultiplexing

Ricordate, dalla Sezione 1.2.2, che un'idea fondamentale della commutazione di pacchetto consiste nell'effettuare il multiplexing di più flussi di dati su un'unica linea fisica di connessione. Questa stessa idea si applica a vari livelli del grafo di protocolli e non soltanto ai nodi di commutazione. In Figura 1.11, ad esempio, possiamo immaginare che il modulo RRP realizzi un canale logico di comunicazione, con i messaggi provenienti da due diverse applicazioni che vengono multiplati sul canale nel calcolatore sorgente e quindi demultiplati nel calcolatore destinazione e consegnati all'applicazione corretta.

Ciò significa, in sostanza, che l'intestazione allegata da RRP ai propri messaggi contiene un identificativo relativo all'applicazione a cui appartengono i messaggi: questo elemento identificativo viene detto *chiave di demultiplexazione* di RRP, o più in breve *chiave di demux*. Nell'host sorgente, RRP inserisce la chiave di demux corretta nella propria intestazione; quando il messaggio viene consegnato al modulo RRP dell'host destinatario, il mo-

dulo elimina l'intestazione, ne ispeziona la chiave di demux e smista il messaggio all'applicazione corretta.

RPP non è l'unico ad utilizzare in questo modo il multiplexing, che è una caratteristica di quasi tutti i protocolli. Ad esempio, anche HPP ha la propria chiave di demux per determinare quali messaggi inoltrare verso l'alto a RPP oppure a MSP; non esiste, però, una posizione comune tra i vari protocolli, nemmeno tra quelli di una singola architettura di rete, riguardo a cosa costituisca effettivamente una chiave di demux. Alcuni protocolli usano un campo di 8 bit (potendo così colloquiare con soli 256 protocolli di livello superiore), mentre altri usano campi di 16 o 32 bit. Ancora, alcuni protocolli usano un unico campo nell'intestazione per realizzare la funzione di chiave di demux, mentre altri usano una coppia di campi. Nel primo caso entrambe le entità terminali della comunicazione usano la stessa chiave di demux, mentre nel secondo caso ciascun agente usa una diversa chiave per identificare il protocollo di livello superiore (o il programma applicativo) a cui il messaggio deve essere consegnato.

1.3.2 L'architettura OSI

ISO è stata una delle prime organizzazioni a definire formalmente un modo comune per connettere calcolatori. La relativa architettura, chiamata architettura OSI (*Open Systems Interconnection*) ed illustrata nella Figura 1.13, definisce una suddivisione delle funzionalità di rete in sette livelli, con uno o più protocolli che realizzano la funzionalità assegnata ad un

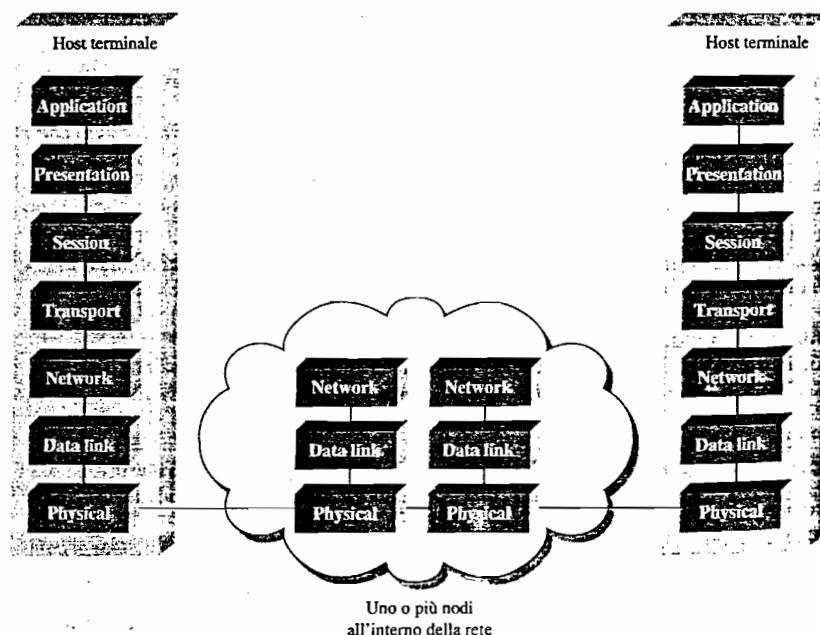


Figura 1.13 L'architettura di rete OSI.

dato livello. In tal senso, lo schema presentato in Figura 1.13 non è di per sé un grafo di protocolli, ma piuttosto un *modello di riferimento* per grafi di protocolli. ISO, di solito congiuntamente ad una seconda organizzazione di standard nota come ITU (International Telecommunications Union)¹, pubblica una serie di definizioni di protocolli basata sull'architettura OSI, serie che a volte viene chiamata "X punto" perché i nomi dei protocolli sono X.25, X.400, X.500 e così via. Esistono diverse reti basate su questi standard, tra cui la rete pubblica X.25 e reti private come Tymnet.

Partendo dal basso e procedendo verso l'alto, lo strato *fisico* (physical) gestisce la trasmissione di flussi non strutturati di bit attraverso una linea di comunicazione. Lo strato di *linea di collegamento* (data link) raccoglie poi un flusso di bit in un aggregato più grande denominato *trama* (frame); questo livello viene tipicamente realizzato dagli adattatori di rete, insieme al software di controllo dei dispositivi (driver) in esecuzione tramite il sistema operativo del nodo: ciò significa che sono i frame, e non i bit, ad essere realmente scambiati tra gli host. Lo strato *di rete* (network) gestisce l'insiradamento fra i nodi in una rete a commutazione di pacchetto; a questo livello il dato atomico scambiato tra i nodi viene tipicamente chiamato *pacchetto* piuttosto che trama, anche se sono fondamentalmente la stessa cosa. I tre livelli inferiori sono implementati in tutti i nodi della rete, compresi gli switch all'interno della rete e gli host connessi lungo la periferia della rete stessa. Lo strato *di trasporto* (transport), poi, realizza ciò che finora abbiamo chiamato case: tra processi: qui, il dato atomico scambiato viene comunemente chiamato *messaggio* piuttosto che pacchetto o trama. Lo strato di trasporto e gli strati superiori sono solitamente in esecuzione soltanto sugli host terminali e non sugli switch e router intermedi.

Sulla definizione dei tre strati superiori c'è meno accordo. Balzando allo strato più alto, il settimo, troviamo lo strato *di applicazione* (application), tra i cui protocolli troviamo, ad esempio, il File Transfer Protocol (FTP), che definisce un protocollo mediante il quale le applicazioni di trasferimento di file possono interoperare. Al di sotto di tale strato, lo strato *di presentazione* (presentation) si occupa del formato dei dati scambiati tra le controparti: ad esempio, si occupa di decidere se un numero intero venga rappresentato con 16, 32 o 64 bit e se il suo bit più significativo venga trasmesso per primo o per ultimo, oppure di come venga codificato un flusso video. Infine, lo strato *di sessione* (session) fornisce uno spazio che viene utilizzato per aggregare i diversi flussi di trasporto che potenzialmente possono venire utilizzati come parti di un'unica applicazione. Ad esempio, può gestire un flusso audio e un flusso video combinati per far funzionare un'applicazione di teleconferenza.

1.3.3 L'architettura Internet

L'architettura Internet, che a volte viene anche chiamata architettura TCP/IP dal nome dei suoi due protocolli principali, è raffigurata in Figura 1.14, con una rappresentazione alternativa in Figura 1.15. L'architettura Internet si è evoluta a partire dall'esperienza di una precedente rete a commutazione di pacchetto che aveva il nome di ARPANET: sia Internet che ARPANET sono state fondate da ARPA (Advanced Research Projects Agency), una delle agenzie del Dipartimento della Difesa degli Stati Uniti d'America che si occupano di finanzi-

¹ Un sottocomitato di ITU dedicato alle telecomunicazioni (ITU-T) ha sostituito un precedente sottocomitato, sempre di ITU, che era conosciuto con il suo nome francese, Comité Consultatif International de Télégraphique et Téléphonique (CCITT).

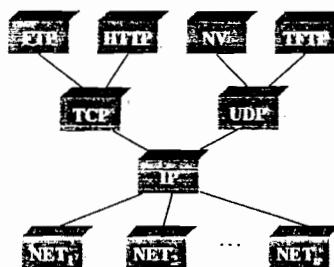


Figura 1.14 Il grafo dei protocolli di Internet.

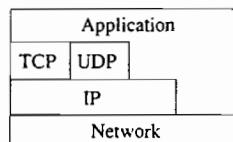


Figura 1.15 Una visione alternativa dell'architettura Internet.

re la ricerca e lo sviluppo. Internet e ARPANET erano già attive al momento della definizione dell'architettura OSI, che è stata fortemente influenzata dall'esperienza maturata nella loro costruzione.

Anche se con qualche sforzo di immaginazione il modello OSI a sette strati può essere applicato ad Internet, di solito si usa un modello a quattro soli strati. Al livello più basso è presente una grande varietà di protocolli di rete, denominati NET_1 , NET_2 e così via; in pratica, questi protocolli sono implementati con una combinazione di hardware (un adattatore di rete) e di software (un componente del sistema operativo per la gestione dell'adattatore, *network device driver*). Ad esempio, in questo strato si trovano i protocolli Ethernet e FDDI (Fiber Distributed Data Interface): tali protocolli, a loro volta, possono in realtà essere costituiti da parecchi sottolivelli, ma l'architettura Internet non fa nessuna ipotesi in merito. Il secondo strato è formato da un unico protocollo, il protocollo IP (Internet Protocol), che fornisce il supporto per l'interconnessione di più tecnologie di rete per formare, dal punto di vista logico, un'unica internetwork. Il terzo strato contiene due protocolli principali, TCP (Transmission Control Protocol) e UDP (User Datagram Protocol), che forniscono due canali logici alternativi per i programmi applicativi: TCP fornisce un canale affidabile a flusso di byte, mentre UDP fornisce un canale non affidabile per la consegna di messaggi (questi messaggi vengono chiamati *datagrammi*, *datagram*, in analogia con i telegrammi). Nella terminologia di Internet, TCP e UDP vengono a volte chiamati protocolli *end-to-end* (perché collegano virtualmente i due estremi della comunicazione in modo diretto), anche se è ugualmente corretto chiamarli protocolli di trasporto.

Al di sopra dello strato di trasporto si trova un insieme di protocolli applicativi, come FTP, TFTP (Trivial File Transport Protocol), Telnet (connessione remota) e SMTP (Simple

1.3 Architettura di rete

Mail Transfer Protocol, il protocollo della posta elettronica), che consentono l'interoperabilità delle applicazioni più diffuse. Per comprendere la differenza che intercorre tra un protocollo dello strato applicativo ed un'applicazione, pensate ai diversi navigatori (browser) che sono disponibili per accedere al servizio World Wide Web (tra i quali, Mosaic, Netscape, Internet Explorer, Lynx, ecc.); un numero altrettanto grande di diversi programmi svolge il ruolo di server Web. Il motivo per il quale potete usare uno qualsiasi di tali programmi applicativi per accedere ad un particolare sito sul Web consiste nella loro conformità allo stesso protocollo al livello applicativo: HTTP (HyperText Transport Protocol). Purtroppo, spesso la confusione è alimentata dal fatto che viene usata la stessa parola sia per identificare un'applicazione sia per identificare il protocollo a livello applicativo da essa utilizzato, come nel caso di FTP.

L'architettura Internet ha tre caratteristiche che è bene sottolineare. Prima di tutto, come è ben raffigurato in Figura 1.15, l'architettura Internet non implica una stratificazione rigida. Un'applicazione può liberamente scavalcare lo strato di trasporto ed usare direttamente IP o una delle reti sottostanti. Ancora maggiore libertà è consentita ai programmati, che possono definire nuove astrazioni di canale o applicazioni che vengano eseguite basandosi su qualsiasi protocollo esistente.

Secondariamente, se osservate attentamente il grafo di protocolli di Figura 1.14, noterete che ha una forma a clessidra: largo in alto, stretto al centro e nuovamente largo alla base. Questa forma rispecchia veramente la filosofia che sta alla base dell'architettura, dove IP ha una funzione chiave, definendo una modalità comune per lo scambio di pacchetti all'interno di una grande varietà di reti. Al di sopra di IP vi possono essere molti protocolli di trasporto, ciascuno per implementare una diversa astrazione di canale da offrire come servizio ai programmi applicativi. In questo modo il problema dello scambio di messaggi tra due host è completamente distinto dal problema di fornire un utile strumento di comunicazione tra processi. Al di sotto di IP, l'architettura consente l'utilizzo di un numero arbitrariamente grande di tecnologie di rete, da Ethernet a FDDI ad ATM, per la realizzazione di singoli collegamenti punto-punto.

Un'ultima caratteristica dell'architettura Internet (o, più precisamente, della cultura di IETF) prevede che per proporre un nuovo protocollo da includere nell'architettura sia necessario fornire sia una definizione del protocollo stesso sia almeno una (e preferibilmente due) implementazioni di riferimento di tale definizione. L'esistenza di una implementazione funzionante è un requisito essenziale per gli standard che vengono adottati da IETF. Questa assunzione culturale della comunità di progettisti aiuta a garantire che i protocolli dell'architettura possano essere implementati in modo efficiente. Forse, il valore che la cultura di Internet ripone nel software funzionante è esemplificato al meglio dalla citazione di una frase scritta sulle magliette che vengono comunemente indossate alle riunioni di IETF:

Rifiutiamo i re, i presidenti e le votazioni. Crediamo nel consenso veritiero e nel codice che funziona.

(Dave Clark)

Di queste tre caratteristiche dell'architettura Internet, la filosofia di progetto a clessidra è sufficientemente importante da giustificare una sua ripetizione. La parte sottile della clessidra rappresenta un insieme minimale e scelto accuratamente di capacità generiche, che consente ad applicazioni di alto livello e a tecnologie di comunicazione di basso livello di coesistere, di condividere potenzialità e di evolvere rapidamente. Tale modello è essenziale perché Internet sia in grado di adattarsi rapidamente alle nuove richieste degli utenti e alle modifiche tecnologiche.

1.4 Implementazione di software di rete

Le architetture di rete e le specifiche dei protocolli sono cose essenziali, ma un buon piano dettagliato non è sufficiente a spiegare il successo fenomenale di Internet: il numero di calcolatori connessi a Internet sta raddoppiando ogni anno dal 1981 e oggi sta raggiungendo i 200 milioni, si stima che il numero di persone che usano Internet sia superiore ai 600 milioni e si ritiene che il numero di bit trasmessi tramite Internet abbia superato nel 2001 il corrispondente valore relativo al sistema di telefonia vocale.

Cosa può giustificare il successo di Internet? Certamente un contributo di molti fattori (tra i quali una buona architettura), ma una cosa che ha reso Internet un successo così travolcente è il fatto che una parte consistente della sua funzionalità sia fornita da software che può essere eseguito da calcolatori di utilizzo generico (*general-purpose computer*). Ciò significa che vi si possono aggiungere rapidamente nuove funzionalità con "un piccolo sforzo di programmazione": come conseguenza, si sono affermate con enorme rapidità nuove applicazioni e sono stati proposti nuovi servizi, come il commercio elettronico, la videoconferenza e la telefonia a commutazione di pacchetto, tanto per nominarne alcuni.

Direttamente correlato a ciò è l'imponente aumento della potenza di calcolo disponibile nei calcolatori più comuni. Nonostante le reti di calcolatori siano sempre state in grado, in linea di principio, di veicolare qualsiasi tipo di informazione, come voce campionata e digitalizzata, immagini digitalizzate e così via, tale potenzialità non era molto interessante quando i calcolatori che dovevano inviare e ricevere questi dati erano troppo lenti per utilizzare queste informazioni. Oggi, praticamente tutti i calcolatori sono in grado di riprodurre voce digitalizzata a velocità reale e flussi video ad una velocità ed una risoluzione sufficiente per alcune applicazioni (ma assolutamente non per tutte); di conseguenza, le reti di oggi hanno iniziato a veicolare contenuti multimediali ed il loro supporto per questo tipo di applicazioni potrà soltanto migliorare con l'aumentare della velocità dell'hardware di elaborazione dei calcolatori.

Il concetto che va astratto da questa discussione è che sapere come realizzare il software di rete svolge un ruolo essenziale nella comprensione delle reti stesse. Con questo obiettivo in mente, questa sezione presenta dapprima alcuni dei problemi relativi alla realizzazione di un programma applicativo che usa la rete, per poi procedere identificando i problemi connessi alla realizzazione dei protocolli in esecuzione all'interno della rete stessa. Per molti aspetti, le applicazioni e i protocolli di rete sono assai simili: il modo in cui un'applicazione usa i servizi della rete è molto simile al modo in cui un protocollo di alto livello usa i servizi dei protocolli di livello inferiore. Come vedremo nel seguito di questa sezione, esistono, comunque, un paio di differenze importanti.

1.4.1 Interfaccia per la programmazione di applicazioni (socket)

Il punto da cui è opportuno iniziare quando si realizza un'applicazione di rete è l'interfaccia che viene resa disponibile dalla rete stessa. Poiché la maggior parte dei protocolli di rete è realizzata da software (e ciò è vero soprattutto per i livelli più alti nella pila di protocolli) e quasi tutte le piattaforme di elaborazione realizzano i protocolli di rete come parte del proprio sistema operativo, quando parliamo di interfaccia "resa disponibile dalla rete" stiamo solitamente parlando dell'interfaccia resa disponibile dal sistema operativo con il proprio sottosistema di rete. Questa interfaccia viene comunemente chiamata API (Application Programming Interface) di rete.

Anche se ciascun sistema operativo ha la libertà di definire la propria API di rete (e la maggior parte lo fa), alcune di queste API si sono, nel tempo, molto diffuse, cioè sono state tradotte e implementate in sistemi operativi diversi da quello di origine. Ciò è quanto è accaduto all'*interfaccia socket*, sviluppata originariamente all'interno della distribuzione di Unix di Berkeley ed ora presente praticamente in tutti i sistemi operativi più diffusi. Il vantaggio del supporto quasi universale, a livello industriale, di una singola API è che le applicazioni che ne fanno uso possono essere facilmente adattate al funzionamento in un diverso sistema operativo. Tuttavia, è importante ricordare che i programmi applicativi solitamente interagiscono con molte parti del sistema operativo, oltre che con il software di rete: ad esempio, leggono e scrivono file, mandano in esecuzione altri processi e visualizzano dati su uno schermo grafico. Il fatto che due sistemi forniscano la stessa API di rete non significa che il loro file system, i loro processi o le loro interfacce grafiche siano compatibili. Nonostante ciò, la comprensione di una API standard ampiamente adottata come i socket di Unix costituisce un valido punto di partenza.

Prima di descrivere l'interfaccia socket, è importante tenere a mente due diversi concetti. Ciascun protocollo fornisce un certo insieme di servizi e la API fornisce una *sintassi* per mezzo della quale quei servizi possono essere utilizzati in un particolare sistema operativo. L'implementazione ha, poi, la responsabilità di creare una corrispondenza tra l'insieme concreto di operazioni e oggetti definito nella API e l'insieme astratto di servizi definiti dal protocollo. Se avete fatto un buon lavoro nella definizione dell'interfaccia, sarà allora possibile usare la sua sintassi per invocare i servizi di molti protocolli diversi: tale generalità è stata certamente uno degli obiettivi dell'interfaccia socket, anche se non è perfetta.

L'astrazione principale dell'interfaccia socket, senza meraviglia, è il *socket* ("presa", nel senso elettrico del termine): potete pensare al socket come al punto in cui il processo di un'applicazione locale si collega alla rete. L'interfaccia definisce le operazioni per creare un socket, per connetterlo alla rete, per inviare e ricevere messaggi attraverso il socket e per chiuderlo. Per semplificare la trattazione, ci limiteremo a mostrare come vengono usati i socket nel protocollo TCP.

Il primo passo consiste nella creazione di un socket, che viene effettuata con l'operazione seguente:

```
int socket(int domain, int type, int protocol)
```

Il motivo per cui questa operazione richiede tre argomenti consiste nel fatto che l'interfaccia socket fu progettata per essere abbastanza generale da poter fornire supporto a qualsiasi insieme di protocolli sottostante. Nello specifico, l'argomento *domain* indica la *famiglia* di protocolli che si sta usando: *PF_INET* indica la famiglia Internet, *PF_UNIX* indica gli strumenti di *pipe* di Unix, mentre *PF_PACKET* indica l'accesso diretto all'interfaccia di rete (scavalcano, cioè, la pila di protocolli TCP/IP). L'argomento *type* indica la semantica di comunicazione: *SOCK_STREAM* viene usato per indicare un flusso di byte, mentre *SOCK_DGRAM* è l'alternativa che indica un servizio orientato ai messaggi, come quello fornito dal protocollo UDP. L'argomento *protocol* identifica lo specifico protocollo che si vuole usare. Nel nostro caso, questo argomento è *UNSPEC* perché la combinazione di *PF_INET* e *SOCK_STREAM* identifica univocamente TCP. Infine, il valore restituito da *socket* è un *handle* (manipolatore) per il socket appena creato, cioè un identificativo mediante il quale ci potremo riferire in futuro al socket, fornendolo come argomento a successive operazioni che riguardino tale socket.

Il passo successivo dipende dal fatto che ci si trovi su un client o su un server. Su una macchina che funge da server, il processo applicativo esegue un'apertura *passiva*: il server dichiara di essere pronto ad accettare connessioni, ma in realtà non ne stabilisce alcuna. Il server fa ciò invocando le seguenti tre operazioni:

```
int bind(int socket, struct sockaddr *address, int addr_len)
int listen(int socket, int backlog)
int accept(int socket, struct sockaddr *address, int *addr_len)
```

L'operazione *bind*, come suggerito dal nome, associa il socket appena creato all'indirizzo *address* specificato, che è l'indirizzo dell'entità *locale* che partecipa alla comunicazione, il server. Notate che, quando viene usato con i protocolli Internet, *address* è una struttura dati che contiene sia l'indirizzo IP del server sia un numero di porta TCP (come vedremo nel Capitolo 5, le porte vengono usate per identificare indirettamente i processi, sono cioè una sorta di chiavi *demux*, così come le abbiamo definite nella Sezione 1.3.1). Il numero di porta è solitamente un numero ben noto, specifico del servizio che viene offerto; ad esempio, i server Web accettano solitamente connessioni sulla porta 80.

L'operazione *listen*, poi, definisce quante connessioni possono rimanere in sospeso sul socket specificato. Infine, l'operazione *accept* porta a termine l'apertura passiva: è un'operazione bloccante che non termina finché un'entità remota non ha stabilito una connessione e, quando restituisce il controllo terminando la propria operazione, restituisce anche un *nuovo* socket che corrisponde alla connessione appena stabilita, con l'argomento *address* che contiene l'indirizzo del partecipante *remoto* alla comunicazione. Si noti che quando *accept* restituisce il controllo dell'esecuzione il socket originario, che era stato fornito come argomento, esiste ancora e continua a corrispondere a quello aperto passivamente: verrà usato per successive invocazioni di *accept*.

Sulla macchina client, il processo applicativo esegue un'apertura *attiva*, cioè dice con chi vuole comunicare, invocando l'unica operazione seguente:

```
int connect(int socket, struct sockaddr *address, int addr_len)
```

Questa operazione non termina finché il protocollo TCP non ha stabilito con successo una connessione e a quel punto l'applicazione può iniziare liberamente ad inviare dati. In questo caso *address* contiene l'indirizzo dell'entità remota coinvolta nella comunicazione; in pratica, il client solitamente specifica soltanto l'indirizzo del partecipante remoto e lascia al sistema il compito di inserire le informazioni relative al partecipante locale. Mentre un server tipicamente rimane in attesa di messaggi su una porta ben nota, al client solitamente non importa quale porta viene usata: semplicemente, il sistema operativo ne seleziona una libera.

Una volta che la connessione è stabilita, i processi applicativi invocano, per inviare e ricevere dati, le due operazioni seguenti:

```
int send(int socket, char *message, int msg_len, int flags)
int recv(int socket, char *buffer, int buf_len, int flags)
```

La prima operazione invia il messaggio *message* tramite il socket specificato, mentre la seconda operazione riceve all'interno del buffer *buffer* un messaggio dal socket specificato. Entrambe le operazioni ricevono un insieme di opzioni *flags* che controllano alcuni dettagli dell'operazione stessa.

1.4.2 Un esempio di applicazione

Mostriamo ora l'implementazione di un semplice programma client/server che usa l'interfaccia socket per inviare messaggi tramite una connessione TCP. Il programma usa anche altre funzioni di utilità di rete del sistema operativo Unix, che introdurremo cammin facendo. La nostra applicazione consente ad un utente di un calcolatore di digitare testo e di inviarlo ad un utente che si trova su un altro calcolatore: è, quindi, una versione semplificata del programma Unix *talk*, che è simile al programma che costituisce il nucleo di una *chat room* del Web.

Client

Iniziamo con la parte client, che riceve come argomento il nome della macchina remota. Per prima cosa, invoca la funzione ausiliaria di Unix, *gethostbyname*, per tradurre tale nome nell'indirizzo IP dell'host remoto. Il passo successivo consiste nella costruzione della struttura dati (*sin*) che contiene l'indirizzo da passare all'interfaccia socket. Notate che questa struttura dati specifica che useremo il socket per una connessione Internet (*AF_INET*). Nel nostro esempio usiamo la porta TCP numero 5432 al posto di un numero di porta a tutti noto: questa porta non è stata assegnata a nessun servizio Internet. L'ultimo passo per stabilire la connessione è l'invocazione di *socket* e *connect*. Terminata l'operazione *connect*, la connessione è stabilita e il programma client entra nel proprio ciclo principale, che legge testo dall'ingresso standard e lo spedisce attraverso il socket.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 5432
#define MAX_LINE 256

int
main(int argc, char *argv[])
{
    FILE *fp;
    struct hostent *hp;
    struct sockaddr_in sin;
    char *host;
    char buf[MAX_LINE];
    int s;
    int len;

    if (argc==2) {
        host = argv[1];
    }
    else {
        fprintf(stderr, "usage: simplex-talk host\n");
        exit(1);
    }
```

```

/* traduce il nome dell'host nell'indirizzo IP della controparte */
hp = gethostbyname(host);
if (!hp) {
    fprintf(stderr, "simplex-talk: unknown host: %s\n", host);
    exit(1);
}

/* costruisce la struttura dati dell'indirizzo */
bzero((char *)&sin, sizeof(sin));
sin.sin_family = AF_INET;
bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
sin.sin_port = htons(SERVER_PORT);

/* apertura attiva */
if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("simplex-talk: socket");
    exit(1);
}
if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
    perror("simplex-talk: connect");
    close(s);
    exit(1);
}
/* ciclo principale: riceve una linea di testo e la invia */
while (fgets(buf, sizeof(buf), stdin)) {
    buf[MAX_LINE-1] = '\0';
    len = strlen(buf) + 1;
    send(s, buf, len, 0);
}

```

Server

Il server è ugualmente semplice; per prima cosa costruisce la struttura dati che rappresenta un indirizzo, inserendovi il numero della propria porta (SERVER_PORT). Evitando di specificare un indirizzo IP, il programma applicativo manifesta l'intenzione di accettare connessioni da qualsiasi indirizzo IP dell'host locale. Successivamente, il server esegue i passi preliminari necessari per un'apertura passiva: crea il socket, lo collega all'indirizzo locale ed impone il numero massimo consentito di connessioni in attesa. Infine, il ciclo principale attende che un host remoto tenti una connessione e, quando lo fa, riceve i caratteri che arrivano tramite la connessione e li visualizza.

```

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

```

```

#define SERVER_PORT 5432
#define MAX_PENDING 5
#define MAX_LINE 256

int
main()
{
    struct sockaddr_in sin;
    char buf[MAX_LINE];
    int len;
    int s, new_s;

    /* costruisce la struttura dati dell'indirizzo */
    bzero((char *)&sin, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons(SERVER_PORT);

    /* imposta l'apertura passiva */
    if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        perror("simplex-talk: socket");
        exit(1);
    }
    if (bind(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
        perror("simplex-talk: bind");
        exit(1);
    }
    listen(s, MAX_PENDING);

    /* attende una connessione, poi riceve testo e lo visualizza */
    while (1) {
        if ((new_s = accept(s, (struct sockaddr *)&sin, &len)) < 0) {
            perror("simplex-talk: accept");
            exit(1);
        }
        while (len = recv(new_s, buf, sizeof(buf), 0))
            fputs(buf, stdout);
        close(new_s);
    }
}

```

1.4.3 Problemi nell'implementazione di protocolli

Come detto all'inizio di questa sezione, il modo in cui i programmi applicativi interagiscono con la rete sottostante è simile al modo in cui un protocollo di alto livello interagisce con un protocollo di livello inferiore. Ad esempio, il protocollo TCP necessita di un'interfaccia per inviare al protocollo IP i messaggi uscenti, e IP deve essere in grado di consegnare i messaggi

in arrivo al protocollo TCP. Questa è precisamente l'interfaccia del servizio, come descritta nella Sezione 1.3.1.

Avendo già una API di rete (ad esempio, i socket), potremmo cedere alla tentazione di usare la medesima interfaccia anche tra ciascuna coppia di protocolli nella pila di protocolli. Anche se questa è certamente un'opzione possibile, in pratica l'interfaccia socket non viene usata in questo modo, perché vi sono inefficienze intrinseche all'interfaccia socket che i realizzatori di protocolli non possono tollerare. I programmati di applicazioni le tollerano perché i socket semplificano il loro compito di programmati e perché tali inefficienze devono essere subite una volta soltanto, ma i realizzatori di protocolli sono spesso ossessionati dalle prestazioni e si devono preoccupare del fatto che i messaggi attraversano parecchi strati di software. La parte rimanente di questa sezione analizza le due differenze principali fra la API di rete e l'interfaccia presente tra due protocolli che si trovano più in basso nel grafo dei protocolli.

Modello a processi

La maggior parte dei sistemi operativi fornisce un'astrazione denominata *processo* o, in alternativa, *thread* (flusso di esecuzione). Ciascun processo viene eseguito in modo largamente indipendente dagli altri processi ed il sistema operativo ha la responsabilità di assicurare che le risorse, come lo spazio di indirizzamento in memoria e i cicli di esecuzione della CPU, vengano assegnate a tutti i processi correnti. Il processo come astrazione rende abbastanza semplice l'esecuzione concorrente di più applicazioni sulla stessa macchina; ad esempio, ciascuna applicazione di un utente viene eseguita in un proprio processo, mentre i vari servizi interni al sistema operativo vengono eseguiti come altri processi. Quando il sistema operativo interrompe l'esecuzione di un processo da parte della CPU e inizia l'esecuzione di un altro processo, questo cambiamento viene detto *cambio di contesto* (context switch).

Progettando il sottosistema di rete, una delle prime domande a cui bisogna rispondere è questa: "Dove sono i processi?" In sostanza esistono due alternative, illustrate in Figura 1.16. Nella prima, che prende il nome di modello *con un processo per ciascun protocollo*, ogni protocollo è realizzato da un processo separato. Ciò significa che, mentre un messaggio viene trasferito verso l'alto o verso il basso nella pila di protocolli, esso viene anche trasferito da un processo/protocollo ad un altro: il processo che implementa il protocollo *i* elabora il messaggio, poi lo trasferisce al protocollo *i - 1*, e così via. Come possa un processo/protocollo trasferire un messaggio al processo/protocollo successivo dipende dalla modalità adottata dal sistema operativo dell'host per la comunicazione interprocesso: solitamente esiste un meccanismo piuttosto semplice per accodare un messaggio diretto ad un processo. Il punto importante, però, è che si rende necessario un cambio di contesto ad ogni livello del grafo di protocolli, e questa è tipicamente un'operazione che richiede tempo.

L'alternativa, che prende il nome di modello *con un processo per ciascun messaggio*, considera ogni protocollo come un segmento di codice statico ed associa i processi ai messaggi: quando dalla rete giunge un messaggio, il sistema operativo mette in esecuzione un processo che si assume la responsabilità del messaggio mentre esso si sposta verso l'alto nel grafo di protocolli. A ciascun livello viene invocata la procedura che implementa il protocollo di tale livello, la qual cosa solitamente produce l'invocazione della procedura del protocollo successivo, e così via. Nel caso di messaggi uscenti, il processo dell'applicazione invoca le procedure necessarie affinché il messaggio venga consegnato. In entrambe le direzioni il grafo di protocolli viene attraversato mediante una serie di invocazioni di procedure.

Nonostante sia a volte più semplice ragionare sul modello con un processo per ciascun protocollo (io realizzo il mio protocollo nel mio processo, e tu realizzi il tuo protocollo nel

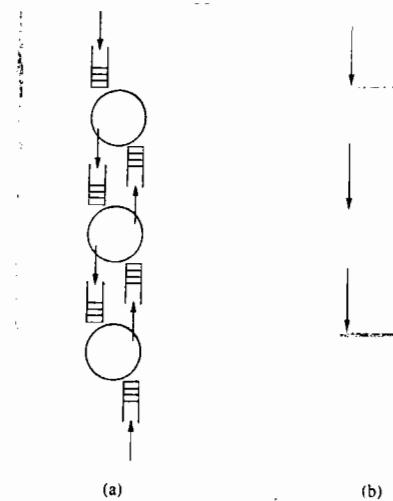


Figura 1.16 Modelli alternativi per i processi: (a) un processo per ciascun protocollo; (b) un processo per ciascun messaggio.

tuo processo), il modello con un processo per ciascun messaggio è, in generale, più efficiente, per un motivo molto semplice: nella maggior parte dei calcolatori, l'invocazione di una procedura è più efficiente di un ordine di grandezza rispetto ad un cambio di contesto. Il primo modello necessita di un costoso cambio di contesto a ciascun livello, mentre il secondo richiede soltanto il costo di un'invocazione di procedura.

Ragioniamo ora sulla relazione che esiste tra l'interfaccia del servizio, come definita precedentemente, ed il modello a processi. Per un messaggio uscente, un protocollo di alto livello invoca un'operazione *send* del protocollo di livello inferiore: dato che il protocollo di alto livello ha in consegna il messaggio quando invoca *send*, questa operazione si può facilmente implementare con un'invocazione di procedura, senza la necessità di un cambio di contesto. Per un messaggio entrante, invece, il protocollo di alto livello invoca l'operazione *receive* del protocollo di livello inferiore e si pone in attesa dell'arrivo di un messaggio, che avverrà in qualche istante nel futuro: questa operazione, in sostanza, provoca un cambio di contesto. In altre parole, il processo che esegue il protocollo di alto livello riceve un messaggio dal processo che esegue il protocollo di livello inferiore. Questo non è un grosso problema se l'applicazione riceve i messaggi direttamente dal sottosistema di rete (ed infatti questa è l'interfaccia corretta per la API di rete, perché i programmi applicativi hanno una visione centrata sui processi), ma ha un impatto significativo sulle prestazioni nel caso in cui tale cambio di contesto avvenga a ciascun livello della pila di protocolli.

Per questa ragione la maggior parte delle implementazioni di protocolli sostituisce l'operazione *receive* con un'operazione *deliver* (consegna): il protocollo di livello inferiore esegue una invocazione verso l'alto per consegnare il messaggio al protocollo di livello superiore. La Figura 1.17 mostra l'interfaccia che ne deriva, in relazione a due protocolli adia-

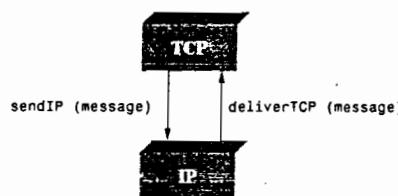


Figura 1.17 Interfaccia tra protocolli adiacenti.

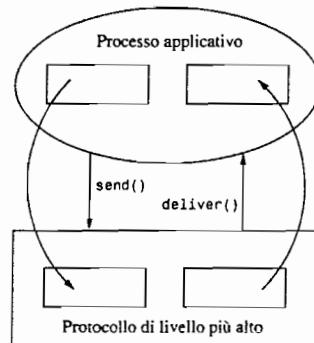


Figura 1.18 Scambio di messaggi entranti/uscenti fra i buffer dell'applicazione e i buffer di rete.

centi, TCP e IP. In generale, i messaggi si spostano verso il basso nel grafo di protocolli mediante una serie di operazioni *send*, e verso l'alto mediante una serie di operazioni *deliver*.

Buffer per i messaggi

Una seconda inefficienza dell'interfaccia socket è dovuta al fatto che il processo applicativo fornisce il buffer che contiene i messaggi uscenti quando invoca *send* e, allo stesso modo, fornisce il buffer in cui viene copiato un messaggio entrante quando invoca l'operazione *receive*. Ciò costringe il protocollo di livello più alto a copiare il messaggio dal buffer dell'applicazione al buffer di rete, e viceversa, come evidenziato in Figura 1.18. Si verifica che la copiatura di dati da un buffer ad un altro è una delle azioni più costose che l'implementazione di un protocollo possa fare, perché, mentre i processori diventano sempre più veloci ad un ritmo incredibile, la memoria non segue lo stesso tasso di sviluppo.

Invece di copiare i dati dei messaggi da un buffer ad un altro ad ogni livello della pila di protocolli, la maggior parte dei sottosistemi di rete definisce un tipo di dati astratto per i messaggi che viene condiviso da tutti i protocolli del grafo di protocolli: questa astrazione non solo consente ai messaggi di transitare verso l'alto e verso il basso nel grafo di protocolli senza copiature, ma di solito fornisce anche modi per compiere senza copiature anche altri tipi di manipolazioni dei messaggi, come aggiungere o togliere intestazioni, frammentare messaggi grandi in un insieme di messaggi più piccoli ed assemblare nuovamente una rac-

olta di piccoli messaggi nel corrispondente messaggio di grandi dimensioni. La forma esatta di questa astrazione per i messaggi differisce da un sistema operativo ad un altro, ma generalmente viene realizzata con una lista concatenata di puntatori a buffer di messaggi, in modo simile a quella mostrata in Figura 1.19. Vi lasciamo come esercizio la definizione di una astrazione generica per messaggi che sia esente da copiature.

1.5 Prestazioni

Fino ad ora abbiamo focalizzato la nostra attenzione sugli aspetti funzionali delle reti, ma, come qualsiasi sistema digitale, le reti di calcolatori hanno anche il requisito di avere buone prestazioni, perché l'efficienza delle elaborazioni distribuite sulla rete dipende spesso in modo diretto dall'efficienza con cui la rete smista i dati. Nonostante il vecchio adagio della programmazione, "Prima fallo funzionare, poi rendilo veloce", sia valido in molti contesti, nelle reti di solito è necessario "progettare per le prestazioni". Per questo motivo, è importante comprendere i diversi fattori che influenzano le prestazioni di una rete.

1.5.1 Ampiezza di banda e latenza

Le prestazioni di una rete si misurano fondamentalmente in due modi: *ampiezza di banda* (bandwidth, spesso chiamata anche *throughput*, quantità di flusso) e *latenza* (latency, chiamata anche *delay*, ritardo). L'ampiezza di banda di una rete è data dal numero di bit che possono essere trasmessi dalla rete in un certo intervallo di tempo. Ad esempio, una rete può avere un'ampiezza di banda di 10 milioni di bit/secondo (Mbps, Megabit per second) e questo significa che è in grado di consegnare 10 milioni di bit ogni secondo. A volte è utile pensare all'ampiezza di banda in termini del tempo necessario per trasmettere un bit: ad esempio, in una rete a 10 Mbps, servono 0.1 microsecondi (μ s) per trasmettere ciascun bit.

Anche se si può parlare dell'ampiezza di banda di una rete come valore complessivo, a volte si vuole essere più precisi, concentrandosi, ad esempio, sull'ampiezza di banda di una singola connessione fisica o di un canale logico tra processi. Al livello fisico l'ampiezza di banda è in costante aumento, senza che si intraveda un limite. Dal punto di vista intuitivo, se pensate ad un intervallo di tempo di un secondo come ad una distanza misurabile con un righello e all'ampiezza di banda come al numero di bit che si possono inserire in

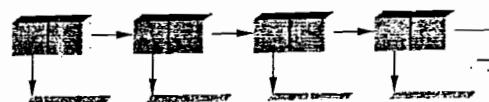


Figura 1.19 Un esempio di struttura dati per messaggi.

Aampiezza di banda e latenza

L'ampiezza di banda e la latenza sono due dei termini usati in modo più confuso quando si parla di reti di calcolatori. Anche se potremmo cercare di dare una definizione precisa di ciascun termine, è importante che sappiate come altre persone li possono usare e che siate consapevoli del fatto che a volte sono addirittura usati come sinonimi. Prima di tutto, in senso letterale l'ampiezza di banda è una misura dell'ampiezza di una banda di frequenza. Ad esempio, un linea telefonica adatta alla trasmissione vocale è in grado di trasmettere segnali in una banda di frequenza compresa tra 300 e 3300 Hz e si dice che ha un'ampiezza di banda di 3000 Hz (= 3300 Hz - 300 Hz). Se trovate l'ampiezza di banda usata in un contesto in cui è espressa in hertz, probabilmente si riferisce al campo di variabilità dei segnali che possono essere utilizzati.

Quando parliamo dell'ampiezza di banda di una linea di comunicazione, solitamente ci riferiamo al numero di bit al secondo che vi si possono trasmettere: ad esempio, possiamo dire che l'ampiezza di banda di una linea Ethernet è 10 Mbps. Tuttavia, è possibile fare un'utile distinzione tra l'ampiezza di banda che è disponibile su una linea di connessione ed il numero di bit al secondo che possono essere realmente trasmessi su quella linea. Per riferirci alle *prestazioni misurate* di un sistema, tendiamo ad utilizzare la parola "throughput". Quindi, a causa di varie inefficienze dell'implementazione, una coppia di nodi collegati da una linea con ampiezza di banda di 10 Mbps potrebbero raggiungere un throughput di soli 2 Mbps. Ciò significa che un'applicazione su uno dei due host potrebbe inviare dati all'altro host alla velocità di 2 Mbps.

Infine, spesso parliamo dei *requisiti* di ampiezza di banda di un'applicazione, che è il numero di bit al secondo che l'applicazione deve trasmettere sulla rete per funzionare in modo accettabile. Per alcune applicazioni ciò potrebbe esprimersi come "il massimo possibile", mentre per altre vi potrebbe essere un numero prefissato (preferibilmente non maggiore dell'ampiezza di banda disponibile sul collegamento); per altre ancora, potrebbe essere un numero che varia nel tempo. Torneremo su questo argomento più avanti, in questa stessa sezione.

tal segmento, allora potete immaginare un bit come un impulso avente una certa ampiezza. Ad esempio, ciascun bit in un collegamento a 1 Mbps ha un'ampiezza di 1 μ s, mentre ciascun bit in un collegamento a 2 Mbps ha un'ampiezza di 0.5 μ s, come si vede in Figura 1.20. Più la tecnologia di trasmissione e ricezione è sofisticata, più stretto può diventare ciascun bit, e, quindi, più elevata può divenire l'ampiezza di banda. Per i canali logici tra processi, l'ampiezza di banda è influenzata anche da altri fattori, tra i quali il numero di volte che il software che realizza il canale deve gestire, ed eventualmente trasformare, ciascun bit di dato.

La seconda metrica delle prestazioni, la latenza, corrisponde al tempo necessario ad un messaggio per attraversare una rete da un capo all'altro (come per l'ampiezza di banda, si può anche parlare di latenza di un singolo collegamento fisico o di un canale tra processi). La latenza viene misurata in termini di tempo: ad esempio, una rete transcontinentale potrebbe avere una latenza di 24 millisecondi (ms), cioè un messaggio impiega 24 ms per viaggiare da un capo all'altro del Nord America. Esistono molte situazioni in cui è più importante conoscere il tempo necessario ad un messaggio per andare e tornare da un capo all'altro della rete,

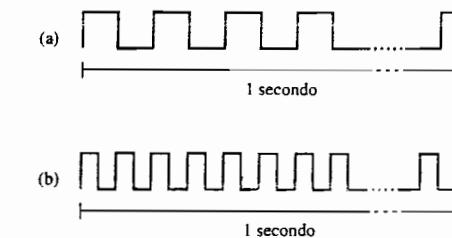


Figura 1.20 I bit trasmessi con una certa ampiezza di banda possono essere visti come impulsi di una determinata ampiezza: (a) bit trasmessi a 1 Mbps (ciascun bit è ampio 1 μ s); (b) bit trasmessi a 2 Mbps (ciascun bit è ampio 0.5 μ s).

piuttosto che la latenza di sola andata: questo tempo viene detto *round-trip time* (RTT, tempo di percorrenza circolare) della rete.

Spesso si considera la latenza come composizione di tre elementi. Innanzitutto, c'è il ritardo di propagazione dovuto alla velocità finita della luce: tale ritardo esiste perché nulla, compresi i bit in una linea di connessione, può viaggiare più veloce della luce. Conoscendo la distanza tra due punti, si può calcolare la latenza dovuta alla velocità finita della luce, anche se bisogna fare attenzione al fatto che la luce viaggia a velocità diverse in mezzi fisici differenti: viaggia a 3.0×10^8 m/s nel vuoto, 2.3×10^8 m/s in un cavo di rame e 2.0×10^8 m/s in una fibra ottica. Poi, bisogna considerare il tempo necessario per trasmettere un dato unitario, che è funzione dell'ampiezza di banda e della dimensione del pacchetto in cui viaggia il dato. Infine, nella rete possono venire introdotti ritardi dovuti a code, perché in generale gli switch devono memorizzare i pacchetti per qualche istante prima di inoltrarli verso una linea in uscita, come visto nella Sezione 1.2.2. Quindi, possiamo definire la latenza complessiva, Latenza, come

$$\begin{aligned} \text{Latenza} &= \text{Propagazione} + \text{Trasmissione} + \text{Accodamento} \\ \text{Propagazione} &= \text{Distanza}/\text{VelocitàDellaLuce} \\ \text{Trasmissione} &= \text{Dimensione}/\text{AmpiezzaDiBanda} \end{aligned}$$

dove Distanza è la lunghezza della linea di connessione attraverso cui viaggiano i dati, VelocitàDellaLuce è la velocità effettiva della luce in tale linea di connessione, Dimensione è la dimensione del pacchetto e AmpiezzaDiBanda è l'ampiezza di banda con cui viene trasmesso il pacchetto. Si noti che se il messaggio contiene un solo bit e consideriamo una singola linea di connessione (invece di un'intera rete), allora i termini Trasmissione e Accodamento sono trascurabili e la latenza corrisponde al solo ritardo di propagazione.

L'ampiezza di banda e la latenza si combinano per definire le caratteristiche di prestazione di una data linea di connessione o di un dato canale. La loro importanza relativa, però, dipende dall'applicazione. Per alcune applicazioni predomina la latenza: ad esempio, un client che invia messaggi di 1 byte ad un server e riceve messaggi di ritorno di 1 byte è vincolato dalla latenza. Ipotizzando che per predisporre la risposta non sia necessaria nessuna elaborazione significativa, l'applicazione avrà prestazioni molto diverse su un canale intercontinentale con un RTT di 100 ms piuttosto che su un canale all'interno di una stanza, con un RTT di 1 ms. Tuttavia, è praticamente privo di importanza il fatto che il canale sia a 1 Mbps piuttosto

che a 100 Mbps, dato che il primo richiede, per la trasmissione di 1 byte, un tempo (Trasmissione) di 8 μ s, mentre il secondo richiede un tempo di 0.08 μ s.

Al contrario, considerate un programma che realizzzi una biblioteca digitale, a cui venga richiesto di recuperare un'immagine di 25 megabyte (MB): maggiore è l'ampiezza di banda disponibile, maggiore sarà la velocità con cui può fornire l'immagine all'utente. In questo caso, l'ampiezza di banda del canale domina le prestazioni. Per rendervene conto, supponete che il canale abbia un'ampiezza di banda di 10 Mbps: ci vorranno 20 secondi per trasmettere l'immagine, rendendo trascurabile il fatto che l'immagine di trovi all'altro capo di un canale con una latenza di 1 ms oppure di 100 ms, perché la differenza tra un tempo di risposta di 20.001 secondi e un tempo di 20.1 secondi non è percepibile dall'utente.

La Figura 1.21 vi dà il senso di come la latenza o l'ampiezza di banda possano dominare le prestazioni in circostanze diverse. Il grafico mostra quanto tempo occorre per trasferire oggetti di varie dimensioni (1 byte, 2KB, 1 MB) attraverso reti con RTT variabili da 1 a 100 ms e velocità delle linee di connessione di 1.5 o 10 Mbps. Sono state usate scale logaritmiche per evidenziare le prestazioni relative. Per un oggetto di 1 byte (diciamo, un carattere premuto su una tastiera), la latenza rimane quasi esattamente uguale al RTT, rendendo indistinguibili una rete a 1.5 Mbps e una a 10 Mbps. Per un oggetto di 2 KB (ad esempio un messaggio di posta elettronica), la velocità della linea di connessione introduce qualche differenza in una rete con RTT di 1 ms, ma non ha quasi alcun effetto in una rete con RTT di 100 ms. Infine, per un oggetto di 1 MB (come un'immagine digitale) il valore di RTT non ha alcuna influenza e

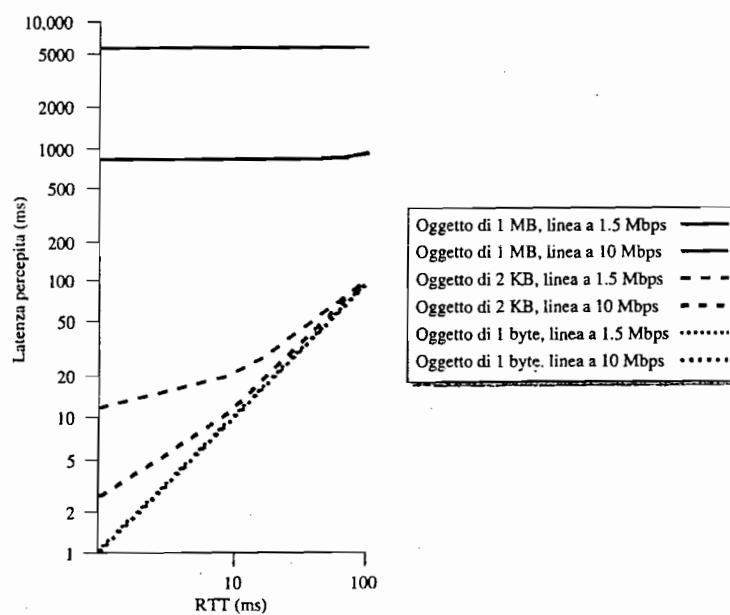


Figura 1.21 Latenza percepita dall'utente (tempo di risposta) in rapporto al round-trip time per oggetti di diverse dimensioni e linee di diversa velocità.

Quanto è grande un mega?

Lavorando con le comuni unità di misura delle reti (MB, Mbps, KB e Kbps), bisogna essere consapevoli dell'esistenza di alcuni trabocchetti. Per prima cosa, bisogna stare molto attenti alla differenza tra bit e byte. In questo libro useremo coerentemente una lettera *b* minuscola per indicare bit e una *B* maiuscola per indicare byte. Secondariamente, bisogna essere sicuri di usare la corretta definizione di mega (M) e chilo (k). *Mega*, ad esempio, può significare 2^{20} oppure 10^6 . Similmente, *chilo* può essere 2^{10} oppure 10^3 . La cosa peggiore è che nell'ambito delle reti usiamo entrambe queste definizioni: ecco spiegato il motivo.

L'ampiezza di banda, che viene spesso espressa in termini di Mbps, è tipicamente determinata dalla velocità del clock che fornisce la cadenza per la trasmissione dei bit: per trasmettere bit a 10 Mbps si usa un clock a 10 MHz. Poiché il *mega* in MHz significa 10^6 hertz, l'unità Mbps viene solitamente definita come 10^6 bit al secondo (analogoamente, Kbps è 10^3 bit al secondo). D'altra parte, quando parliamo di un messaggio che vogliamo trasmettere, spesso esprimiamo la sua dimensione in chilobyte: dato che i messaggi sono memorizzati nella memoria del calcolatore e la memoria viene tipicamente misurata con potenze di due, la lettera *K* in KB ha solitamente il significato di 2^{10} (analogoamente, MB di solito significa 2^{20}). Se mettete insieme le due cose, non è insolito parlare della spedizione di un messaggio di 32 KB lungo un canale a 10 Mbps, che dovrebbe significare che $32 \times 2^{10} \times 8$ bit vengono trasmessi alla velocità di 10×10^6 bit al secondo. In questo libro useremo questa interpretazione, a meno che non sia menzionata esplicitamente una convenzione diversa.

La buona notizia è che molto spesso ci accontentiamo di un calcolo approssimato, nel qual caso è ragionevole ritenere, ad esempio, che un byte contenga 10 bit (per rendere più agevole la conversione fra bit e byte) e che 10^6 sia effettivamente uguale a 2^{20} (per rendere più facile la conversione fra le due definizioni di mega). Notate che la prima di queste approssimazioni introduce un errore del 20%, mentre la seconda introduce un errore del 5% soltanto.

Per aiutarvi nei vostri calcoli a spanne, 100 ms è un valore ragionevole da usare per il round-trip time di una connessione continentale (attraverso gli Stati Uniti d'America), mentre 1 ms è una buona approssimazione del valore di RTT in una rete locale. Nel primo caso, aumentiamo a 100 ms il round-trip time di 48 ms che sarebbe dovuto alla sola velocità della luce in una fibra, perché, come abbiamo già detto, vi sono altre fonti di ritardo, tra cui il tempo di elaborazione all'interno degli switch della rete. Inoltre, state certi che il percorso di una fibra che collega due punti non sarà una linea retta.

le prestazioni sono dominate dalla velocità della linea di connessione per l'intero intervallo di variabilità di RTT.

Notate che in tutto questo libro usiamo i termini *latenza* e *ritardo* in modo generico, cioè per indicare il tempo necessario a svolgere una determinata funzione, come la consegna di un messaggio o il trasferimento di un oggetto. Quando ci riferiamo in particolare al tempo necessario perché un segnale si propaghi da un capo all'altro di una linea di connessione, usiamo invece il termine *ritardo di propagazione*. Infine, renderemo chiaro nel contesto della discussione se stiamo parlando della latenza di sola andata, oppure del round-trip time.

Come considerazione finale, i calcolatori stanno diventando così veloci che quando li connettiamo alle reti è a volte utile ragionare, almeno figurativamente, in termini di *istruzioni per miglio*. Considerate cosa accade quando un calcolatore che sia in grado di eseguire un miliardo di istruzioni al secondo invia un messaggio lungo un canale con un RTT di 100 ms (per semplificare i calcoli, ipotizzate che il messaggio percorra una distanza di 5000 miglia). Se quel calcolatore rimane inattivo per 100 ms in attesa del messaggio di risposta, ha rinunciato alla possibilità di eseguire 100 milioni di istruzioni, cioè 20000 istruzioni per miglio. Sarebbe stato meglio analizzare la rete più in dettaglio, per giustificare questo spreco.

1.5.2 Prodotto ritardo × ampiezza di banda

Riveste una certa utilità considerare il prodotto di queste due metriche, spesso chiamato *prodotto ritardo × ampiezza di banda*. Dal punto di vista intuitivo, se pensiamo ad un canale fra una coppia di processi come ad una condutture (come in Figura 1.22), dove la latenza corrisponde alla lunghezza della condutture e l'ampiezza di banda è in relazione al suo diametro, allora il prodotto ritardo × ampiezza di banda rappresenta il volume interno della condutture, cioè il numero di bit che può contenere. Detto in altro modo, se la latenza (misurata come intervallo di tempo) corrisponde alla lunghezza della condutture, allora è possibile calcolare quanti bit possono trovarsi all'interno della condutture, in base all'ampiezza di ciascun bit (nuovamente misurata come intervallo di tempo). Ad esempio, un canale intercontinentale con una latenza di sola andata di 50 ms ed un'ampiezza di banda di 45 Mbps è in grado di contenere

$$50 \times 10^{-3} \text{ secondi} \times 45 \times 10^6 \text{ bit/secondo} = 2.25 \times 10^6 \text{ bit}$$

cioè circa 280 KB di dati. In altre parole, questo canale (condutture) d'esempio può contenere tanti byte quanti ne poteva contenere un personal computer dei primi anni 80.

Quando si progettano reti ad elevate prestazioni, è importante conoscere il prodotto ritardo × ampiezza di banda perché corrisponde al numero di bit che possono essere inviati dalla sorgente prima che il primo bit arrivi a destinazione. Nel caso in cui il mittente voglia che il destinatario gli segnali che i bit hanno iniziato ad arrivare, poiché bisogna attendere un altro intervallo di tempo uguale alla latenza perché tale segnale si propaghi fino al mittente (e siamo quindi interessati al RTT del canale piuttosto che alla latenza di sola andata), il mittente può inviare una quantità di dati pari al doppio del prodotto ritardo × ampiezza di banda prima di sapere dal destinatario che tutto sta procedendo correttamente. I bit che si trovano nella condutture sono "in viaggio": ciò significa che se il destinatario segnala al mittente di interrompere la trasmissione, potrebbe ancora ricevere una quantità di dati pari al prodotto ritardo × ampiezza di banda prima che il mittente sia in grado di reagire. Nel nostro esempio precedente, tale quantità corrisponde a 5.5×10^6 bit (671 KB). D'altra parte, se il mittente non riempie la condutture (cioè se non invia una quantità di dati pari al prodotto ritardo × ampiezza di banda prima di fermarsi in attesa di un segnale), non sfrutta appieno le potenzialità della rete.

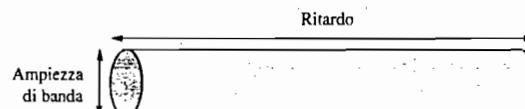


Figura 1.22 La rete vista come una condutture.

Noteate che il più delle volte ci interessano situazioni in cui ciò che conta è il valore di RTT, per cui ci riferiamo semplicemente al prodotto ritardo × ampiezza di banda, senza dire esplicitamente che tale prodotto va moltiplicato per due. Anche in questo caso, è il contesto a chiarire se il "ritardo" nel prodotto ritardo × ampiezza di banda sia la latenza di sola andata oppure il RTT.

1.5.3 Reti ad alta velocità

Le ampiezze di banda disponibili nelle reti di oggi stanno aumentando ad un ritmo vertiginoso e c'è una sconfinata fiducia nel fatto che continuino ad aumentare. Questa situazione porta i progettisti di reti ad iniziare a pensare a cosa accadrà nelle situazioni limite, o, detto in altro modo, quale sia l'impatto sul progetto delle reti di una disponibilità di ampiezza di banda infinita.

Anche se le reti ad alta velocità portano incredibili cambiamenti nell'ampiezza di banda disponibile per le applicazioni, per molti aspetti il loro impatto maggiore sul nostro modo di pensare alle reti viene da ciò che *non* cambia all'aumentare dell'ampiezza di banda: la velocità della luce. Citando Scotty di *Star Trek*, "non si possono cambiare le leggi della fisica". In altre parole, "alta velocità" non significa che la latenza migliora allo stesso ritmo dell'ampiezza di banda: il valore di RTT per una connessione transcontinentale a 1 Gbps è sempre 100 ms, come per una connessione a 1 Mbps.

Per meglio comprendere il significato di un'ampiezza di banda in continuo aumento con una latenza fissa, considerate il tempo richiesto per trasmettere un file di 1 MB attraverso una rete a 1 Mbps in confronto con il tempo richiesto per trasmetterlo attraverso una rete a 1 Gbps, con un RTT di 100 ms in entrambi i casi. Nella prima rete, a 1 Mbps, serve un tempo uguale a 80 volte RTT per trasmettere il file: durante ogni intervallo di tempo di durata RTT viene trasmesso una frazione uguale all'1.125% del file. Lo stesso file, invece, non richiede nemmeno un intero RTT per essere trasmesso lungo la connessione a 1 Gbps, che ha un prodotto ritardo × ampiezza di banda di 12.5 MB.

La Figura 1.23 mostra la differenza tra le due reti. Nella rete a 1 Mbps, il file di 1 MB sembra a tutti gli effetti un flusso di dati che deve essere trasmesso, mentre nella rete a 1 Gbps sembra un singolo pacchetto. Per aiutarvi ad assimilare questo concetto, pensate che un file di 1 MB è per la rete a 1 Gbps ciò che un *pacchetto* di 1 KB è per una rete a 1 Mbps.

Un altro modo per ragionare su questa situazione è che in una rete ad alta velocità durante ciascun intervallo di durata RTT vengono trasmessi molti dati, così tanti che un singolo intervallo di durata RTT diventa una quantità di tempo significativa. Di conseguenza, mentre non stareste tanto a pensare alla differenza fra un trasferimento di file che richiede 101 RTT piuttosto che 100 RTT (una differenza relativa dell'1% soltanto), la differenza tra 1 RTT e 2 RTT diventa improvvisamente importante: un aumento del 100%. In altre parole, le nostre considerazioni sul progetto delle reti iniziano ad essere dominate dalla latenza, piuttosto che dall'ampiezza di banda.

Il modo migliore, forse, per comprendere la relazione che intercorre tra throughput e latenza consiste nel tornare ai principi primi. Il throughput effettivo che si può ottenere in una rete è dato dalla semplice relazione

$$\text{Throughput} = \frac{\text{QuantitàTrasferita}}{\text{TempoDiTrasferimento}}$$

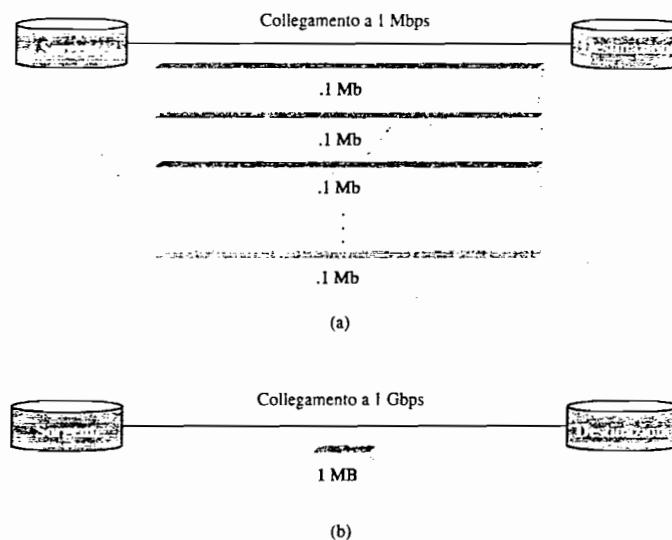


Figura 1.23 Relazione esistente tra ampiezza di banda e latenza. Con un file di 1 MB:
 (a) la connessione a 1 Mbps richiede 80 condutture piene di dati;
 (b) la connessione a 1 Gbps ha solo 1/12 di condutture piene.

dove **TempoDiTrasferimento** non comprende soltanto gli elementi della **Latenza** di sola andata identificati in precedenza in questa sezione, ma anche l'eventuale tempo aggiuntivo speso per richiedere il trasferimento o per renderlo possibile. In generale, questa relazione si rappresenta come

$$\text{TempoDiTrasferimento} = \text{RTT} + 1/\text{AmpiezzaDiBanda} \times \text{QuantitàTrasferita}$$

Usiamo **RTT** in questo calcolo per tener conto del messaggio di richiesta che viene inviato lungo la rete e dei dati che vengono inviati in risposta. Ad esempio, considerate una situazione in cui un utente richiede un file di 1 MB attraverso una rete a 1 Gbps con un round-trip time di 100 ms. Il **TempoDiTrasferimento** comprende sia il tempo di trasmissione per 1 MB ($1 / 1 \text{ Gbps} \times 1 \text{ MB} = 8 \text{ ms}$) sia il valore di RTT di 100 ms, per un tempo totale di 108 ms. Ciò significa che il throughput effettivo sarà

$$1 \text{ MB}/108 \text{ ms} = 74.1 \text{ Mbps}$$

e non 1 Gbps. Trasferire una maggiore quantità di dati aiuta, ovviamente, a migliorare il throughput effettivo: al limite, un trasferimento di dimensioni infinitamente grandi rende il throughput effettivo uguale all'ampiezza di banda della rete. Di converso, dover attendere più di un RTT, ad esempio per ritrasmettere pacchetti perduti, peggiorerà il throughput effettivo per qualsiasi trasferimento di dimensioni finite e sarà massimamente evidente per piccoli trasferimenti.

1.5.4 Prestazioni richieste dalle applicazioni

La discussione in questa sezione ha assunto un punto di vista delle prestazioni incentrato sulla rete: abbiamo cioè parlato nei termini di ciò che una data linea di connessione o canale saranno in grado di fornire. L'ipotesi non esplicitata è che i programmi applicativi abbiano requisiti semplici, che vogliono cioè tanta ampiezza di banda quanta la rete ne può fornire. Ciò è certamente vero per il summenzionato programma per la biblioteca digitale che sta recuperando un'immagine di 25 MB: maggiore è l'ampiezza di banda disponibile, più velocemente il programma potrà fornire l'immagine all'utente.

Tuttavia, alcune applicazioni sono in grado di dichiarare un limite superiore all'ampiezza di banda di cui hanno bisogno, e le applicazioni video sono un esempio tipico. Supponete di voler creare un flusso video con immagini che siano grandi un quarto dell'immagine televisiva normale, cioè con una risoluzione di 352 per 240 pixel. Se ciascun pixel è rappresentato da 24 bit di informazione, come nel caso di colori a 24 bit, allora la dimensione di ciascun frame video è

$$(352 \times 240 \times 24)/8 = 247.5 \text{ KB}$$

Se l'applicazione deve fornire i frame ad una velocità di 30 frame al secondo, potrebbe allora richiedere un throughput di 75 Mbps. La capacità della rete di fornire un'ampiezza di banda maggiore non ha alcun interesse per una tale applicazione, perché essa non ha così tanti dati da trasmettere in un dato intervallo di tempo.

Sfortunatamente, la situazione non è così semplice come suggerito da questo esempio. Poiché la differenza fra due frame consecutivi in un flusso video è spesso minima, è possibile comprimere il flusso di immagini trasmettendo soltanto le differenze tra frame adiacenti. Tale flusso video compresso non fluisce a velocità costante, ma varia nel tempo in dipendenza da fattori quali la quantità di movimento e di dettagli nelle immagini e l'algoritmo di compressione utilizzato. Di conseguenza, è possibile dire quale sia l'ampiezza di banda media richiesta, ma quella istantanea può essere maggiore o minore.

Il punto chiave è l'intervallo di tempo nel quale viene calcolata la media. Supponete che questa applicazione video che stiamo usando come esempio possa essere compressa fino al punto da richiedere, in media, soltanto 2 Mbps. Se trasmette 1 megabit in un intervallo di 1 secondo e 3 megabit nel successivo intervallo di 1 secondo, allora nell'intervallo di 2 secondi sta trasmettendo alla velocità media di 2 Mbps; questo, però, sarebbe un'informazione poco utile per un canale progettato per una velocità non maggiore di 2 megabit in un secondo. Conoscere soltanto l'ampiezza di banda media richiesta da un'applicazione non è sempre sufficiente, ovviamente.

In generale, però, è possibile valutare un limite superiore per la grandezza dei *burst* ("raffiche") di trasmissione di un'applicazione. Un burst si può descrivere con una velocità di picco che viene mantenuta per un certo intervallo di tempo, oppure con il numero di byte che vengono inviati alla velocità di picco prima di tornare alla velocità media o ad una velocità inferiore. Se questa velocità di picco è superiore alla capacità del canale, i dati in eccesso devono essere memorizzati da qualche parte, per essere trasmessi più tardi. Sapendo quale possa essere la dimensione dei burst, il progettista di rete può utilizzare dei buffer di capacità sufficiente per contenere tali burst. Torneremo sull'argomento della descrizione del traffico a burst con maggiore dettaglio nel Capitolo 6.

Così come le necessità di ampiezza di banda di un'applicazione possono essere diverse da "tutto ciò che si può avere", anche la richiesta di ritardo di un'applicazione può essere

formulata in modo più complesso del semplice "il ritardo minore possibile". Nel caso del ritardo, a volte non importa molto che la latenza di sola andata nella rete sia 100 ms oppure 500 ms, mentre è molto importante quanto la latenza varia da pacchetto a pacchetto. La variazione della latenza viene chiamata *jitter*.

Considerate la situazione in cui la sorgente invia un pacchetto ogni 33 ms, come succederebbe nel caso di un'applicazione video che trasmetta 30 frame al secondo. Se i pacchetti arrivano a destinazione con un ritardo reciproco di 33 ms esatti, possiamo dedurre che il ritardo subito da ciascun pacchetto nella rete è esattamente lo stesso. Se, invece, il tempo che intercorre tra l'arrivo a destinazione di due pacchetti (talvolta chiamato *distanza interpacchetto*) è variabile, allora il ritardo subito dalla sequenza di pacchetti è anch'esso variabile e la rete ha introdotto jitter (*nervosismo*) nel flusso di pacchetti, come si vede in Figura 1.24. Una tale variabilità non viene solitamente introdotta in un'unica connessione fisica, ma si può verificare quando i pacchetti subiscono ritardi di accodamento diversi in una rete a commutazione di pacchetto. Questo ritardo di accodamento corrisponde ad una variabilità nel tempo della componente Accodamento della latenza, come definita in precedenza in questa sezione.

Per comprendere l'importanza del jitter, supponete che i pacchetti trasmessi sulla rete contengano frame video e che per visualizzare tali frame sullo schermo il ricevente abbia bisogno di ricevere un nuovo frame ogni 33 ms. Se un frame arriva in anticipo può essere semplicemente memorizzato dal ricevente finché non sia ora di visualizzarlo, ma quando sfortunatamente un frame arriva in ritardo il ricevente non ha il frame che gli serve per aggiornare in tempo lo schermo e la qualità del video ne soffre: il video non è fluido. Notate che non è necessario eliminare il jitter, è sufficiente sapere quanto vale. Infatti, se il ricevente conosce il limite superiore e inferiore della latenza che può caratterizzare un pacchetto, può ritardare il momento in cui inizia la riproduzione del video (cioè in cui visualizza il primo frame) per un tempo sufficientemente lungo da essere certo da avere sempre, in futuro, un frame da visualizzare nel momento in cui gli serve. Il ricevente ritarda il frame, eliminando il problema del jitter, memorizzandolo in un buffer. Torneremo sull'argomento del jitter nel Capitolo 9.

1.6 Riepilogo

Le reti di calcolatori, come Internet, hanno conosciuto nell'ultimo decennio un enorme sviluppo e sono ora in grado di fornire a centinaia di milioni di utenti un gran numero di servizi, tra cui l'accesso a file remoti e a biblioteche digitali e la videoconferenza. Gran parte di questa crescita può essere attribuita alla natura non specifica delle reti di calcolatori e, in particolare, alla possibilità di aggiungere nuove funzionalità alla rete scrivendo semplicemente software da eseguire su computer ampiamente disponibili e ad elevate prestazioni.



Figura 1.24 Jitter introdotto dalla rete.

Detto ciò, il difficile compito di questo libro consiste nel descrivere le reti di calcolatori in modo che, dopo aver terminato la sua lettura e avendo un esercito di programmati ai vostri ordini, dovreste avere la sensazione di poter veramente costruire dal nulla una rete di calcolatori perfettamente funzionante. Questo capitolo getta le fondamenta per la realizzazione di questo obiettivo.

Il primo passo che abbiamo compiuto verso tale obiettivo è stata l'attenta individuazione di ciò che ci aspettiamo precisamente da una rete. Ad esempio, una rete deve innanzitutto fornire in modo efficiente dal punto di vista dei costi la connessione di un insieme di calcolatori: ciò si realizza mediante un'interconnessione nidificata di nodi e linee di collegamento, condividendo questa base hardware mediante l'uso del multiplexing statistico. Tutto questo dà luogo ad una rete a commutazione di pacchetto, sulla quale definiamo successivamente un insieme di servizi di comunicazione tra processi.

Il secondo passo ha visto la definizione di un'architettura a strati che ci servirà come guida per la progettazione, un'architettura che ha come suoi oggetti centrali i protocolli di rete. I protocolli forniscono un servizio di comunicazione a protocolli di livello superiore e definiscono il formato e il significato dei messaggi che vengono scambiati con le controparti in esecuzione su altre macchine. Abbiamo, poi, brevemente presentato due delle architetture più utilizzate: l'architettura OSI e l'architettura Internet. Questo libro segue più fedelmente l'architettura Internet, sia nella sua organizzazione sia come fonte di esempi.

Il terzo passo è stato l'implementazione software di protocolli di rete e di programmi applicativi: entrambi necessitano di un'interfaccia tramite la quale invocare i servizi di altri protocolli del sottosistema di rete. L'interfaccia socket è la più utilizzata di queste interfacce fra i programmi applicativi ed il sottosistema di rete, ma all'interno del sottosistema di rete viene usata un'interfaccia leggermente diversa.

Infine, la rete nel suo complesso deve anche offrire elevate prestazioni, che vengono misurate essenzialmente tramite la latenza e il throughput. Come vedremo nei capitoli successivi, è il prodotto di questi due fattori, chiamato prodotto ritardo × ampiezza di banda, a svolgere spesso un ruolo critico nel progetto di protocolli.

Problema aperto La rete pervasiva

Ci sono pochi dubbi sul fatto che le reti di calcolatori stiano diventando parte integrante della vita quotidiana di un gran numero di persone. Ciò che iniziò più di 20 anni fa come un sistema sperimentale, con il nome di ARPANET, per collegare computer di tipo mainframe tramite linee telefoniche a grande distanza, si è rivelato essere un grande affare, e dove ci sono grandi affari, ci sono anche molti partecipanti in ruoli diversi. In questo caso c'è l'industria dei calcolatori, che è sempre più interessata al supporto di prodotti per le reti a commutazione di pacchetto; ci sono le compagnie telefoniche, che offrono al mercato la possibilità di trasportare non solo la voce, ma dati di qualsiasi tipo; e c'è l'industria della televisione via cavo, che al momento detiene il controllo della parte del mercato relativa all'intrattenimento.

Ipotizzando che l'obiettivo sia una rete pervasiva, che arrivi in tutte le case, il primo problema da risolvere è come realizzare i collegamenti fisici necessari. Anche se è possibile sostenere che la risposta definitiva consiste nel portare una fibra ottica in ogni casa, ad un costo stimato di 1000 dollari per ciascuna casa, con 100 milioni di case soltanto negli Stati Uniti questa idea si concretizza in un investimento di 100 miliardi di dollari. Le alternative su cui si discute maggiormente fanno uso dei cablaggi esistenti per il sistema televisivo oppure

delle coppie di fili in rame usate per fornire il servizio telefonico, ma ciascuno di questi approcci ha i suoi problemi. Ad esempio, i cablaggi televisivi sono oggi asimmetrici: potete trasmettere 150 canali in ogni casa, ma l'ampiezza di banda uscente è drasticamente limitata. Questa asimmetria implica che ci sia un numero ridotto di fornitori di informazione e che la maggior parte di noi siano semplicemente consumatori di tali informazioni. Molte persone sostengono, invece, che in una democrazia dovremmo avere tutti la medesima opportunità di fornire informazioni. La tecnologia DSL (Digital Subscriber Line) non è necessariamente asimmetrica, ma è in grado di offrire connessioni a larga banda mediante i collegamenti telefonici esistenti soltanto ad un sottoinsieme di consumatori.

Quale sarà il risultato della battaglia commerciale che si sta combattendo tra industrie di calcolatori, compagnie telefoniche e industria televisiva è un'altra incognita (se conoscessimo la risposta, questo libro costerebbe molto di più). Tutto ciò che sappiamo è che esistono molti ostacoli tecnici (problemi di connessione, livelli di servizio, prestazioni, affidabilità ed equità) frapposti fra lo stato attuale delle cose e quella rete globale, pervasiva ed eterogenea che crediamo possibile e desiderabile. Queste sfide sono l'oggetto di questo libro.

Ulteriori letture

Le reti di calcolatori non sono la prima tecnologia orientata alle comunicazioni ad avere trovato un proprio ruolo nel tessuto sociale e nella vita quotidiana. Ad esempio, la prima parte del secolo scorso ha visto l'introduzione del telefono e durante gli anni Cinquanta si è diffusa la televisione. Quando si prende in esame il futuro delle reti, cioè si cerca di immaginare quanto si diffonderanno e quanto le useremo, è istruttivo studiare questa storia: la nostra prima citazione è un valido punto di partenza per farlo (l'intero numero è dedicato ai primi 100 anni delle telecomunicazioni).

Il secondo e il terzo lavoro citato sono, rispettivamente, le pubblicazioni di presentazione delle architetture OSI e Internet. Il lavoro di Zimmermann presenta l'architettura OSI, mentre il lavoro di Clark è una retrospettiva. Le ultime due pubblicazioni non sono specifiche delle reti, ma qualsiasi persona coinvolta nella progettazione di sistemi le dovrebbe conoscere. Il lavoro di Saltzer *et al.* descrive e motiva una delle regole di progettazione di sistema più diffuse, il *punto di vista end-to-end*. Il lavoro di Mashey descrive le idee che stanno alla base delle architetture RISC: come scopriremo presto, prendere buone decisioni in relazione a dove inserire le varie funzionalità all'interno di un sistema complesso è tutto ciò che deve fare la progettazione di sistema.

- Pierce, J. "Telephony - a personal view", *IEEE Communications* 22(5):116-120, May 1984.
- Zimmermann, H. "OSI reference model - the ISO model of architecture for open systems interconnection", *IEEE Transactions on Communications* COM-28(4):425-432, April 1980.
- Clark, D. "The design philosophy of the DARPA Internet protocols", *Proceedings of the SIGCOMM '88 Symposium*, 106-114, August 1988.
- Saltzer, J., D. Reed e D. Clark "End-to-end arguments in system design", *ACM Transactions on Computer Systems* 2(4):277-288, November 1984.
- Mashey, J. "RISC. MIPS, and the notion of complexity", *UniForum 1986 Conference Proceedings*, pagg. 116-124, 1986.

Ci sono parecchi testi introduttivi alle reti di calcolatori: Stallings presenta una trattazione encyclopedica dell'argomento, con particolare riferimento ai livelli inferiori della gerarchia OSI [Sta00a]; Tanenbaum usa l'architettura OSI come modello di riferimento [Tan02]; Comer prende in esame l'architettura Internet [Com00]; Bertsekas e Gallager parlano di reti dal punto di vista dei modelli di prestazioni [BG92].

Per inserire le reti di calcolatori in un contesto più ampio, vale la pena di leggere due testi, uno che tratta del passato e un altro che è rivolto al futuro. Il primo è *The Early History of Data Networks* di Holzmann e Pehrson [HP95]: sarete sorpresi dall'apprendere che molte delle idee che state leggendo in questo libro furono delineate nel 1700. Il secondo è *Realizing the Information Future: The Internet and Beyond*, un testo del Computer Science and Telecommunications Board del National Research Council [NRC94].

Per seguire la storia di Internet dai suoi inizi, vi invitiamo a consultare ripetutamente la serie di documenti *Request for Comments* (RFC) di Internet, che si possono trovare all'indirizzo <http://www.ietf.org/rfc.html> e comprendono tutto, dalla specifica di TCP agli scherzi del Pesce d'Aprile. Ad esempio, le specifiche dei protocolli TCP, UDP e IP si trovano, rispettivamente, in RFC 793, 768 e 791.

Per acquisire una migliore consapevolezza della filosofia e della cultura di Internet, occorre leggere due testi di riferimento, entrambi piuttosto divertenti. Padlipsky descrive bene i primi giorni di Internet e il suo libro contiene anche un confronto particolareggiato delle architetture OSI e Internet [Pad85], mentre consigliamo un articolo di Boorsook [Boo95] per una cronaca aggiornata di ciò che veramente accade dietro il sipario dell'IETF (Internet Engineering Task Force).

C'è anche un gran numero di articoli che discutono vari aspetti delle implementazioni di protocolli. Per iniziare, non è male conoscere due ambienti che realizzano dei protocolli completi: il meccanismo Stream di Unix System V [Rit84] e x-kernel [HP91]. Inoltre, [LMQ89] e [SW95] descrivono l'implementazione di TCP/IP più diffusa, nel sistema operativo Berkeley Unix.

Più in generale, esiste un ampio corpo di lavori che trattano i problemi relativi alla strutturazione e all'ottimizzazione dell'implementazione di protocolli. Clark fu uno dei primi ad affrontare le relazioni esistenti tra il progetto modulare e le prestazioni dei protocolli [Cla82]. Lavori successivi hanno introdotto l'uso di invocazioni verso l'alto nella strutturazione del codice di protocolli [Cla85] e hanno analizzato i costi aggiuntivi dovuti all'elaborazione in TCP [CJRS89]. Infine, [WM87] descrive come migliorare l'efficienza mediante una progettazione mirata e adeguate scelte implementative.

Parecchie pubblicazioni hanno introdotto tecniche specifiche e meccanismi da poter usare per migliorare le prestazioni dei protocolli. Ad esempio, [HMPT89] descrive alcuni dei meccanismi usati in x-kernel, [MD93] analizza diverse implementazioni delle tabelle usate per il demultiplexing, [VL87] introduce il meccanismo della ruota degli eventi per gestire gli eventi all'interno dei protocolli e [DP93] descrive una strategia efficiente per la gestione dei buffer. Ancora, le prestazioni dei protocolli eseguiti su processori paralleli, nei quali uno dei problemi chiave è l'acquisizione esclusiva delle risorse, sono discusse in [BG93] e [NYKT94].

Dato che molti aspetti dell'implementazione dei protocolli dipendono dalla comprensione dei meccanismi fondamentali dei sistemi operativi, raccomandiamo la lettura di Finkel [Fin88], Bic e Shaw [BS88] e Tanenbaum [Tan01], per una presentazione dei concetti relativi ai sistemi operativi.

Infine, termineremo la sezione "Ulteriori letture" di ciascun capitolo con un elenco di riferimenti attivi, cioè di URL che si riferiscono a locazioni nel World Wide Web dove potete

approfondire gli argomenti presentati nel capitolo. Essendo questi riferimenti attivi e dinamici, è possibile che non siano più validi dopo un po' di tempo; per questo motivo limitiamo questi riferimenti a siti che forniscono software o servizi, oppure danno informazioni su attività in corso all'interno di gruppi di lavoro o di organismi di standardizzazione. In altre parole, elenchiamo URL soltanto per quel tipo di materiale a cui non si possa fare riferimento tramite citazioni ordinarie. Ecco i quattro riferimenti per questo capitolo:

- <http://www.mkp.com/>: informazioni sulla versione originale di questo libro
- <http://www.acm.org/sigcomm/sos.html>: stato di diversi standard di rete, fra cui quelli di IETF, ISO e IEEE
- <http://www.ietf.org/>: informazioni sull'IETF e i suoi gruppi di lavoro
- <http://www.cs.columbia.edu/~hgs/netlib/>: materiale bigliografico relativo a pubblicazioni di ricerca sulle reti, all'interno del quale è possibile effettuare ricerche

Esercizi

1. Usate il servizio FTP anonimo per connettervi a [ftp.isi.edu](ftp://ftp.isi.edu) (directory `in-notes`) e recuperare l'indice degli RFC. Prelevate anche le specifiche per i protocolli TCP, IP e UDP.
2. Consultate il sito Web

<http://www.cs.princeton.edu/nsr>

dove potete avere informazioni in merito alle attuali ricerche in corso alla Princeton University e vedere una fotografia dell'autore Larry Peterson. Seguite i collegamenti attivi fino a trovare un'immagine dell'autore Bruce Davie.

3. Usate uno strumento di ricerca su Web per trovare informazioni utili, generali e non commerciali relative ai seguenti argomenti: MBone, ATM, MPEG, IPv6 e Ethernet.
4. Il programma di utilità di Unix `whois` serve a trovare il nome di dominio corrispondente ad una organizzazione, o viceversa. Consultate la documentazione del comando e fate esperimenti con esso: per iniziare, provate i comandi `whois princeton.edu` e `whois princeton`.
5. Calcolate il tempo totale necessario per trasferire un file di 1000 KB nei seguenti casi, ipotizzando un RTT di 100 ms, una dimensione del pacchetto di 1 KB ed un tempo iniziale utilizzato per impostare la comunicazione ("handshaking") prima di inviare i dati di durata pari a due volte RTT.
 - a) L'ampiezza di banda è 1.5 Mbps e i pacchetti di dati possono essere inviati uno dopo l'altro.
 - b) L'ampiezza di banda è 1.5 Mbps, ma dobbiamo attendere un intervallo di tempo di durata RTT dopo aver inviato un pacchetto, prima di inviare il successivo.
 - c) L'ampiezza di banda è "infinita", cioè possiamo assumere uguale a zero il tempo di trasmissione, e in ciascun intervallo di tempo di durata pari a RTT si possono inviare fino a 20 pacchetti.
 - d) L'ampiezza di banda è infinita; durante il primo intervallo di tempo di durata pari a RTT possiamo inviare un pacchetto (2^{1-1}), durante il secondo intervallo di tempo di durata pari a RTT possiamo inviare due pacchetti (2^{2-1}), durante il terzo ne possiamo inviare quattro (2^{3-1}), e così via (nel Capitolo 6 vedremo una motivazione per questo aumento esponenziale).

- ✓ 6. Calcolate il tempo totale necessario per trasferire un file di 1.5 MB nei seguenti casi, ipotizzando un RTT di 80 ms, una dimensione del pacchetto di 1 KB ed un tempo iniziale utilizzato per impostare la comunicazione ("handshaking") prima di inviare i dati di durata pari a due volte RTT.
- a) L'ampiezza di banda è 10 Mbps e i pacchetti di dati possono essere inviati uno dopo l'altro.
 - b) L'ampiezza di banda è 10 Mbps, ma dobbiamo attendere un intervallo di tempo di durata RTT dopo aver inviato un pacchetto, prima di inviare il successivo.
 - c) L'ampiezza di banda è "infinita", cioè possiamo assumere uguale a zero il tempo di trasmissione, e in ciascun intervallo di tempo di durata pari a RTT si possono inviare fino a 20 pacchetti.
 - d) L'ampiezza di banda è infinita; durante il primo intervallo di tempo di durata pari a RTT possiamo inviare un pacchetto (2^{1-1}), durante il secondo intervallo di tempo di durata pari a RTT possiamo inviare due pacchetti (2^{2-1}), durante il terzo ne possiamo inviare quattro (2^{3-1}), e così via (nel Capitolo 6 vedremo una motivazione per questo aumento esponenziale).
7. Considerate un collegamento punto-punto con lunghezza di 2 km. Quale valore di ampiezza di banda rende il ritardo di propagazione (ad una velocità di 2×10^8 m/s) uguale al ritardo di trasmissione per pacchetti di 100 byte? E per pacchetti di 512 byte?
- ✓ 8. Considerate un collegamento punto-punto con lunghezza di 50 km. Quale valore di ampiezza di banda rende il ritardo di propagazione (ad una velocità di 2×10^8 m/s) uguale al ritardo di trasmissione per pacchetti di 100 byte? E per pacchetti di 512 byte?
9. Quali proprietà degli indirizzi postali andrebbero bene anche per uno schema di indirizzamento in una rete? Quali differenze vi aspettate di trovare? Quali proprietà della numerazione telefonica andrebbero bene anche per uno schema di indirizzamento in una rete?
10. Una proprietà degli indirizzi è quella di essere unici: se due nodi avessero lo stesso indirizzo, sarebbe impossibile distinguerli. Quali altre proprietà sarebbero utili per gli indirizzi in una rete? Potete immaginare una situazione in cui gli indirizzi in una rete (o gli indirizzi postali o i numeri telefonici) potrebbero *non* essere unici?
11. Fornite un esempio di una situazione in cui gli indirizzi multicast potrebbero essere utili.
12. Quali differenze nella tipologia di traffico rendono conto del fatto che STDM è una modalità di multiplexing efficiente dal punto di vista del costo per una rete che trasporta telefonia vocale e che FDM lo è per le reti radiotelevisive, ma entrambe non lo sono per una rete di calcolatori ad utilizzo generico?
13. Quanto è "ampio" un bit su un collegamento a 1 Gbps? E quanto lo è in un cavo di rame, dove la velocità di propagazione è 2.3×10^8 m/s?
14. Quanto tempo serve per trasmettere x KB su un collegamento a y Mbps? Fornite la risposta sotto forma di rapporto fra x e y .
15. Supponete che sia stato stabilito un collegamento punto-punto a 100 Mbps fra la Terra ed una nuova colonia lunare. La distanza fra la Terra e la Luna è circa 385000 km e i dati viaggiano lungo la connessione alla velocità della luce, 3×10^8 m/s.
 - a) Calcolate il valore minimo di RTT per il collegamento.
 - b) Usando RTT come valore per il ritardo, calcolate il prodotto ritardo × ampiezza di banda per il collegamento.
 - c) Qual è il significato del prodotto ritardo × ampiezza di banda calcolato al punto precedente?

- d) Un apparecchio fotografico sulla base lunare raccoglie immagini della Terra e le memorizza in formato digitale su un disco. Supponete che il Controllo Missione sulla Terra voglia scaricare l'immagine più attuale, che ha una dimensione di 25 MB. Qual è il minimo intervallo di tempo che trascorrerà tra il momento in cui viene inviata la richiesta per tali dati ed il momento in cui il trasferimento è terminato?
- ✓ 16. Supponete che sia stato stabilito un collegamento punto-punto a 128 Kbps fra la Terra ed veicolo di esplorazione su Marte. La distanza fra la Terra e Marte (quando sono massimamente vicini) è circa 55 Gm e i dati viaggiano lungo la connessione alla velocità della luce, 3×10^8 m/s.
- Calcolate il valore minimo di RTT per il collegamento.
 - Calcolate il prodotto ritardo \times ampiezza di banda per il collegamento.
 - Un apparecchio fotografico sul veicolo di esplorazione raccoglie immagini dell'ambiente circostante e le invia sulla Terra. Dopo quanto tempo da quando è stata acquisita l'immagine giunge al Controllo Missione sulla Terra? Ipotizzate che ciascuna immagine abbia le dimensioni di 5 Mb.
17. Per ciascuna delle seguenti operazioni su un server di file remoto, argomentate sul fatto che sia più sensibile al ritardo o all'ampiezza di banda.
- Aprire un file.
 - Leggere il contenuto di un file.
 - Elencare il contenuto di una directory.
 - Visualizzare gli attributi di un file.
18. Calcolate la latenza (dal primo bit inviato all'ultimo bit ricevuto) per le reti seguenti:
- Ethernet a 10 Mbps con un solo switch lungo il percorso, di tipo store-and-forward, e con dimensione di pacchetto di 5000 bit. Ipotizzate che ciascuna linea di connessione introduca un ritardo di propagazione di 10 μ s e che lo switch, non appena ha finito di ricevere un pacchetto, inizi immediatamente ad inoltrarlo.
 - Come la rete precedente, ma con tre switch.
 - Come la rete del punto (a), ma ipotizzate che lo switch realizzi la commutazione di tipo "cut-through", che sia cioè in grado di iniziare ad inoltrare un pacchetto dopo averne ricevuto i primi 200 bit.
- ✓ 19. Calcolate la latenza (dal primo bit inviato all'ultimo bit ricevuto) per le reti seguenti:
- Ethernet a 1 Gbps con un solo switch lungo il percorso, di tipo store-and-forward, e con dimensione di pacchetto di 5000 bit. Ipotizzate che ciascuna linea di connessione introduca un ritardo di propagazione di 10 μ s e che lo switch, non appena ha finito di ricevere un pacchetto, inizi immediatamente ad inoltrarlo.
 - Come la rete precedente, ma con tre switch.
 - Come la rete precedente, ma ipotizzate che gli switch realizzino la commutazione di tipo "cut-through", che siano cioè in grado di iniziare ad inoltrare un pacchetto dopo averne ricevuto i primi 128 bit.
20. Calcolate l'ampiezza di banda effettiva nei tre casi seguenti. Per i casi (a) e (b), ipotizzate che ci sia un flusso continuo di dati da inviare; per il caso (c), calcolate semplicemente la media in un intervallo di tempo di 12 ore.
- Ethernet a 10 Mbps con tre switch di tipo store-and-forward, come nell'Esercizio 18(b). Gli switch possono inviare i dati lungo una linea mentre stanno ricevendo da un'altra linea.
 - Come la rete precedente, ma con la sorgente che deve attendere un pacchetto di conferma di 50 byte prima di inviare ciascun successivo pacchetto con 5000 bit di dati.

- c) Consegna durante la notte (12 ore) di 100 compact disk (650 MB ciascuno).
21. Calcolate il prodotto ritardo \times ampiezza di banda per i seguenti collegamenti, usando il ritardo di sola andata, misurato dal primo bit inviato al primo bit ricevuto.
- Ethernet a 10 Mbps con ritardo di 10 μ s.
 - Ethernet a 10 Mbps con un unico switch di tipo store-and-forward, come nell'Esercizio 18(a), pacchetti di 5000 bit e ritardo di propagazione di 10 μ s per ciascuna linea.
 - Collegamento T1 a 1.5 Mbps con un ritardo transcontinentale di sola andata di 50 ms.
 - Collegamento T1 a 1.5 Mbps attraverso un satellite in orbita geostazionaria, all'altitudine di 35900 km. L'unico ritardo è il ritardo di propagazione dovuto alla velocità della luce.
22. Ciascuno degli host A e B è connesso ad uno switch S tramite una linea di collegamento a 10 Mbps, come indicato in Figura 1.25. Il ritardo di propagazione di ciascuna linea è 20 μ s. S è un dispositivo di tipo store-and-forward ed inizia ad inoltrare un pacchetto 35 μ s dopo aver finito di riceverlo. Calcolate il tempo totale necessario per trasmettere 10000 bit da A a B
- con un singolo pacchetto
 - con due pacchetti di 5000 bit inviati uno dopo l'altro
23. Supponete che un host abbia un file di 1 MB da inviare ad un altro host. Serve 1 secondo di tempo di CPU per comprimerre la dimensione del file del 50%, oppure 2 secondi per comprimerla del 60%.
- Calcolate l'ampiezza di banda per la quale ciascuna opzione di compressione dà luogo allo stesso tempo totale, sommando tempo di compressione e tempo di trasmissione.
 - Spiegate perché la latenza non ha influenza sulla vostra risposta.
24. Supponete che un certo protocollo di comunicazione abbia bisogno di 100 byte aggiuntivi per ogni pacchetto, per le intestazioni e le informazioni di frame. Inviamo un milione di byte di dati usando questo protocollo, ma un byte risulta essere corrotto e l'intero pacchetto che lo contiene deve essere scartato. Calcolate, per pacchetti di dimensioni uguali a 1000, 5000, 10000 e 20000 byte, il numero totale di byte aggiuntivi + perduti. Qual è la dimensione ottimale?
25. Ipotizzate di voler trasferire un file di n byte lungo un percorso composto dalla sorgente, la destinazione, sette collegamenti punto-punto e cinque switch. Supponete che ciascun collegamento abbia un ritardo di propagazione di 2 ms, ampiezza di banda di 4 Mbps e che gli switch funzionino sia a commutazione di circuito che di pacchetto. Di conseguenza, potete scomporre il file in pacchetti di 1 KB, oppure predisporre un circuito attraverso gli switch ed inviare il file come un flusso ininterrotto di bit. Supponete che: i pacchetti abbiano 24 byte di informazioni nell'intestazione di ciascun pacchetto e 1000 byte di carico utile; l'elaborazione di tipo store-and-forward dei pacchetti in ciascuno switch provochi un ritardo di 1 ms tra il momento in cui la ricezione del pacchetto è terminata e il momento in cui inizia il suo inoltro; i pacchetti possano essere inviati in continuazione, senza attendere conferme; la predisposizione del circuito richiede che un pacchetto di 1 KB faccia un intero percorso di andata e ritorno con un

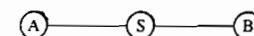


Figura 1.25 Schema per l'Esercizio 22.

- ritardo di 1 ms per ogni switch attraversato. Ipotizzate che gli switch non introducano ritardo nei dati che attraversano poi il circuito predisposto. Potete anche fare l'ipotesi che la dimensione del file sia un multiplo di 1000 byte.
- Per quale dimensione n del file il numero di byte inviati in rete è minore per il circuito rispetto a quanto necessario con i pacchetti?
 - Per quale dimensione n del file la latenza totale calcolata fino al momento in cui l'intero file è giunto a destinazione è minore per il circuito rispetto a quanto necessario con i pacchetti?
 - Quanto sono sensibili questi risultati rispetto al numero di switch presenti lungo il percorso? E rispetto all'ampiezza di banda dei collegamenti? E rispetto al rapporto tra la dimensione dei pacchetti e della loro intestazione?
 - Quanto pensate che sia accurato questo modello riguardo ai vantaggi relativi fra il circuito e i pacchetti? Sono state trascurate considerazioni importanti che sarebbero a svantaggio di uno o dell'altro approccio? Se sì, quali?
26. Considerate una rete chiusa ad anello (ad esempio, una rete token ring) con ampiezza di banda di 100 Mbps e velocità di propagazione di 2×10^8 m/s. Quale dovrebbe essere la circonferenza dell'anello per contenere esattamente un pacchetto di 250 byte, nell'ipotesi che i nodi non introducano ritardi? Quale dovrebbe essere la circonferenza se ci fosse un nodo ogni 100 m e ciascun nodo introducesse un ritardo di 10 bit?
27. Confrontate i requisiti di un canale per traffico vocale con i requisiti relativi alla trasmissione di musica in tempo reale, in termini di ampiezza di banda, ritardo e jitter. Che cosa dovrebbe essere migliorato? E di quanto, all'incirca? Sarebbe possibile rendere meno stringente qualcuno dei requisiti del canale?
28. Anche se in pratica ciò non accadrebbe mai, nei casi seguenti fate l'ipotesi che non venga effettuata alcuna compressione dei dati. Nei primi tre casi, calcolate l'ampiezza di banda necessaria per la trasmissione in tempo reale:
- di un flusso video alla risoluzione di 640×480 pixel, 3 byte/pixel, 30 frame/secondo;
 - di un flusso video 160×120 pixel, 1 byte/pixel, 5 frame/secondo;
 - di un CD-ROM musicale, nell'ipotesi che un CD contenga 75 minuti di musica in 650 MB di dati.
- Ipotizzate che un fax trasmetta un'immagine in bianco e nero di dimensioni 8×10 pollici alla risoluzione di 72 pixel per pollice. Quanto tempo è necessario per farlo con un modem a 14.4 Kbps?
- ✓ 29. Come nel problema precedente, ipotizzate che non si effettui alcuna compressione dei dati. Calcolate l'ampiezza di banda necessaria per la trasmissione in tempo reale:
- di un flusso video ad alta definizione (HDTV) alla risoluzione di 1920×1080 pixel, 24 bit/pixel, 30 frame/secondo;
 - di un flusso audio vocale POTS (Plain Old Telephone Service, vecchio servizio telefonico ordinario) con campionamenti di 8 bit a 8 KHz;
 - di un flusso audio vocale per la telefonia mobile GSM con campionamenti di 260 bit a 50 Hz;
 - di un flusso audio ad alta definizione (HDCD) con campionamenti di 24 bit a 88.2 KHz.
30. Discutete le prestazioni relative richieste dalle seguenti applicazioni, in termini di ampiezza di banda media, ampiezza di banda di picco, latenza, jitter e tolleranza alle perdite:
- server di file
 - server di stampa

- c) biblioteca digitale
 - d) monitoraggio ordinario di strumenti meteorologici remoti
 - e) voce
 - f) monitoraggio video di una sala d'attesa
 - g) diffusione televisiva broadcast
31. Supponete che un mezzo condiviso M offra agli host A_1, A_2, \dots, A_N , a turno, l'opportunità di trasmettere un pacchetto; gli host che non hanno nulla da trasmettere rifiutano immediatamente tale opportunità, liberando M. Come differisce questa strategia da SDTM? Come si confronta con SDTM l'adozione di questo schema in una rete?
- ★ 32. Considerate un semplice protocollo per il trasferimento di file lungo una linea di connessione. Dopo una negoziazione iniziale, A invia pacchetti di dati di dimensione 1 KB a B, che risponde con una conferma (ACK, acknowledgement). A attende sempre un ACK prima di inviare il pacchetto di dati successivo, una strategia che prende il nome di *stop-and-wait* (fermati-e-attendi). I pacchetti la cui conferma è attesa per troppo tempo vengono dati per smarriti e quindi ritrasmessi.
- In assenza di perdita o duplicazione di pacchetti, spiegate perché non è necessario aggiungere alcun "numero di sequenza" nell'intestazione dei pacchetti.
 - Supponete che il collegamento possa occasionalmente perdere dei pacchetti, ma che i pacchetti che arrivano siano sempre nell'ordine in cui sono stati inviati. Un numero di sequenza di 2 bit (cioè, N modulo 4) è sufficiente perché A e B possano identificare e ritrasmettere i pacchetti perduti? Un numero di sequenza di 1 bit sarebbe ugualmente sufficiente?
 - Supponete ora che il collegamento possa consegnare i pacchetti nell'ordine sbagliato e che a volte un pacchetto possa essere consegnato anche un minuto dopo pacchetti ad esso successivi. Come vengono modificati i requisiti relativi al numero di sequenza?
- ★ 33. Supponete che gli host A e B siano connessi da una linea. L'host A trasmette in continuazione l'ora attuale derivata da un orologio ad elevata precisione, con una frequenza di trasmissione costante e abbastanza velocemente da utilizzare tutta l'ampiezza di banda disponibile. L'host B legge questi valori di tempo e li scrive accoppiando ciascun valore con la propria nozione di ora attuale, derivata da un orologio locale sincronizzato con quello di A. fornite esempi qualitativi di ciò che viene scritto da B, nell'ipotesi che il collegamento abbia:
- grande ampiezza di banda, elevata latenza e basso jitter
 - piccola ampiezza di banda, elevata latenza e elevato jitter
 - grande ampiezza di banda, bassa latenza, basso jitter e perdita occasionale di dati
- Ad esempio, una connessione senza jitter, con ampiezza di banda sufficientemente ampia da poter scrivere ciascun istante fornito dall'orologio ed una latenza di uno di tali istanti potrebbe generare una sequenza cosiffatta: (0000, 0001), (0002, 0003), (0004, 0005).
34. Realizzate il programma di esempio con i socket, `simplex-talk`, presentato nel capitolo. Eseguite un server e un client, in finestre separate. Mentre il primo client è in esecuzione, fate partire altri 10 client che si connettono allo stesso server: tali client dovrebbero essere eseguiti in *background*, leggendo i dati in ingresso da un file. Cosa accade a questi 10 client? La loro invocazione di `connect` fallisce immediatamente, oppure fallisce dopo aver atteso un certo tempo (time out), oppure ha successo? Ci sono altre invocazioni che si bloccano? A questo punto fate terminare il primo client. Cosa accade? Fate questa prova usando per il server il valore di `MAX_PENDING` impostato a 1.

35. Modificate il programma che usa i socket, `simplex-talk`, in modo che ogni volta che il client invia una linea di testo al server, il server la invii al client. Il client (e il server) dovranno ora eseguire alternativamente invocazioni di `recv()` e di `send()`.
36. Modificate il programma che usa i socket, `simplex-talk`, in modo che usi UDP come protocollo di trasporto, invece di TCP. Dovrete modificare `SOCK_STREAM` in `SOCK_DGRAM` sia nel client sia nel server. Quindi, nel server, eliminate le invocazioni di `listen()` e `accept()`, sostituendo i due cicli annidati finali con un unico ciclo che invoca `recv()` con il socket `s`. Infine, osservate cosa accade quando due di tali client UDP si connettono simultaneamente allo stesso server UDP, e confrontate questo comportamento con quello dei client/server TCP.
37. Scoprite le diverse opzioni e parametri che si possono impostare in una connessione TCP (eseguendo il comando `man tcp` su un sistema Unix). Fate esperimenti con diverse impostazioni di parametri per vedere come si modificano le prestazioni di TCP.
38. Il programma di utilità di Unix, `ping`, viene usato per osservare il valore di RTT verso diversi host di Internet. Consultate la sua documentazione e usatelo per trovare il valore di RTT verso www.cs.princeton.edu nel New Jersey e verso www.cisco.com in California. Misurate i valori di RTT in diversi momenti della giornata e confrontate i risultati. A cosa pensate che siano dovute le differenze?
39. Il programma di utilità di Unix, `traceroute`, oppure la sua controparte Windows, `tracert`, viene usato per scoprire la sequenza di router attraverso cui transita un messaggio. Usatelo per scoprire il percorso dal vostro computer a qualche altra destinazione. Quanto è correlato il numero di "salti" (hop) con il tempo di RTT misurato da `ping`? E quanto è correlato il numero di salti con la distanza geografica?
40. Usate `traceroute`, come nell'esercizio precedente, per identificare alcuni dei router all'interno della rete della vostra organizzazione (oppure per verificare che non ve ne sono).

Reti a connessione diretta

Problema Connettere fisicamente i calcolatori

La più semplice rete immaginabile è quella in cui tutti gli host sono direttamente connessi mediante un mezzo fisico, che può essere un cavo o una fibra e può estendersi su una piccola superficie (come un edificio) o su un'ampia area (ad esempio transcontinentale). Tuttavia, connettere due o più nodi con un adatto supporto fisico è soltanto il primo passo: prima che i nodi si possano scambiare pacchetti con successo occorre risolvere cinque ulteriori problemi.

Il primo problema è la *codifica* (encoding) dei bit sul cavo o sulla fibra in modo che possano essere compresi dall'host ricevente. Il secondo consiste nel delineare la sequenza di bit trasmessi lungo il collegamento fisico in modo che rappresentino un messaggio completo da consegnare al nodo terminale: si tratta del problema della *tramatura* (framing) e spesso i messaggi consegnati al nodo terminale vengono chiamati *trame* (frame). Dato che a volte i frame vengono corrotti durante la trasmissione, il terzo problema da risolvere consiste nel rilevare questi errori e nell'intraprendere le azioni appropriate: si tratta del problema della *rilevazione d'errore* (error detection). Il quarto problema si occupa di far apparire affidabile un collegamento nonostante il fatto che di tanto in tanto i frame risultino corrotti. Infine, in quei casi in cui il mezzo fisico è condiviso da più host (diversamente da quanto accade per un semplice collegamento punto-punto), è necessario mediare l'accesso al collegamento stesso: si parla di problema di *controllo di accesso al mezzo* (media access control).

Nonostante questi cinque problemi (codifica, tramatura, rilevazione d'errore, consegna affidabile e accesso mediato) possano essere affrontati in astratto, sono argomenti molto concreti che vengono risolti in modi diversi da differenti tecnologie di rete. Questo capitolo considera tali problemi nel contesto di quattro specifiche tecnologie di rete: collegamenti punto-punto, reti CSMA (Carrier Sense Multiple Access, di cui Ethernet è l'esempio più noto), reti token ring (i cui esempi più famosi sono lo standard IEEE 802.5 e FDDI) e reti wireless (senza cavi, di cui 802.11 è lo standard emergente). L'obiettivo di questo capitolo è di fornire una panoramica delle tecnologie di rete disponibili e di affrontare, al tempo stesso, questi cinque problemi fondamentali.

Prima di addentrarsi nei problemi specifici della connessione di host, questo capitolo inizia con un analisi degli elementi costitutivi che verranno utilizzati: nodi e linee di collegamento. Quindi, analizziamo in dettaglio i primi tre problemi (codifica, tramatura e rilevazione d'errore) nel contesto di un semplice collegamento punto-punto. Le tecniche presentate in queste tre sezioni sono generali e, quindi, applicabili anche alle reti ad accesso multiplo. Successivamente viene preso in esame il problema della consegna affidabile dei messaggi: dato che l'affidabilità al livello della connessione fisica non viene di solito realizzata nelle reti ad accesso condiviso, questa trattazione riguarda soltanto i collegamenti punto-punto. Infine, tratteremo il problema dell'accesso al mezzo per quanto riguarda le tecnologie CSMA, token ring e wireless.

Notate che queste cinque funzioni sono generalmente implementate in un adattatore di rete, una scheda che da una parte si inserisce nel bus di input/output (I/O) di un host e dall'altra parte si collega al mezzo fisico. In altre parole, sono gli adattatori che si scambiano i bit, mentre i nodi si scambiano frame corretti. Questo adattatore funziona sotto il controllo di software in esecuzione nel nodo (il *device driver*), software che, a sua volta, viene tipicamente rappresentato come il protocollo di livello più basso nel grafo dei protocolli. Il capitolo si conclude con un esempio concreto di un adattatore di rete e delinea il device driver per tale adattatore.

2.1 Elementi hardware elementari

Come abbiamo visto nel Capitolo 1, le reti contengono due categorie di elementi hardware elementari: *nodi* e *linee di collegamento* (link). Questa frase è vera tanto per la più semplice delle reti possibili (quella in cui un'unica linea di collegamento connette una coppia di nodi) quanto per una internet mondiale. Questa sezione fornisce una breve panoramica di ciò che intendiamo per nodi e linee di collegamento e nel farlo definisce anche la tecnologia sottostante che considereremo nel resto del libro.

2.1.1 Nodi

I nodi sono solitamente calcolatori ad utilizzo generico, come una stazione di lavoro da scrivania, un calcolatore multiprocessore o un PC; per quanto ci riguarda, ipotizziamo che sia una macchina nella categoria delle stazioni di lavoro (workstation). Questa workstation può fungere da host sul quale gli utenti eseguono i propri programmi applicativi, oppure potrebbe essere usata all'interno della rete come switch (commutatore) che inoltra messaggi da una linea di connessione ad un'altra, oppure ancora potrebbe essere configurata come router (intradattatore) che inoltra pacchetti di dati da una rete ad un'altra. In alcuni casi un nodo è realizzato con hardware specializzato, e ciò avviene più spesso per uno switch o un router all'interno della rete, piuttosto che per un host. Questo accade solitamente per motivi di costi e prestazioni: in generale, è possibile costruire hardware apposito che esegua una particolare funzione più velocemente e a minor costo di quanto possa fare un processore ad utilizzo generico. Quando si verifica questa situazione, descriveremo prima la funzione di base eseguita dal nodo, come se tale funzione fosse realizzata da software in esecuzione su una workstation generica, quindi spiegheremo perché e come tale funzionalità possa invece essere implementata da hardware specializzato.

Anche se potremmo tralasciarlo, è invece utile conoscere qualche dettaglio sul funzionamento interno di una workstation, un'informazione particolarmente importante quando ci preoccuperemo delle prestazioni della rete. La Figura 2.1 delinea un semplice diagramma a blocchi dei calcolatori nella categoria delle workstation che useremo in questo libro: ci preme sottolineare tra aspetti chiave di questa figura.

Prima di tutto, la memoria di qualsiasi calcolatore ha dimensioni finite: può essere 4 MB o 128 MB, ma non è infinita. Come abbiamo messo in evidenza nella Sezione 1.2.2, ciò è importante perché la memoria risulta essere una delle risorse critiche nella rete (l'altra è l'ampiezza di banda dei collegamenti), che devono essere gestite con attenzione se vogliamo fornire a ciascun utente un'equa frazione delle capacità della rete. La memoria è una risorsa critica perché in un nodo che funziona da switch o da router i pacchetti devono essere conservati in memoria mentre aspettano il proprio turno di trasmissione lungo una linea di connessione uscente.

Secondo, ciascun nodo si connette alla rete mediante un *adattatore di rete*, che solitamente è collegato al bus di I/O di sistema e trasporta i dati fra la memoria della workstation e la connessione di rete. Un modulo software in esecuzione sulla workstation, il *device driver*, gestisce questo adattatore: invia comandi all'adattatore, dicendogli, ad esempio, da quale locazione di memoria i dati uscenti devono essere prelevati e in quale locazione di memoria i dati entranti devono essere salvati. La Sezione 2.9 presenterà gli adattatori con maggiore dettaglio.

Infine, mentre le CPU diventano sempre più veloci ad un ritmo incredibile, lo stesso non vale per la memoria. I recenti andamenti delle prestazioni mostrano che la velocità dei processori raddoppia ogni 18 mesi, mentre la latenza della memoria migliora soltanto del 7% all'anno. L'importanza di questa differenza è tale che, in prima approssimazione, una workstation funziona come nodo di rete alla velocità della memoria e non alla velocità del processore. Ciò significa che il software di rete deve essere molto attento rispetto all'utilizzo della memoria e, in particolare, rispetto al numero di volte che accede alla memoria mentre elabora ciascun messaggio. Non possiamo permetterci il lusso di essere disattenti soltanto perché i processori stanno diventando infinitamente veloci.

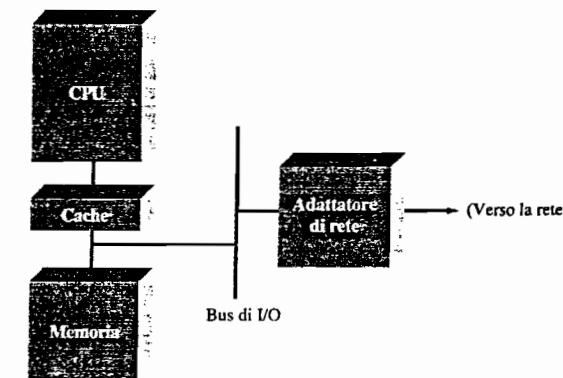


Figura 2.1 Esempio di architettura di workstation.

2.1.2 Linee di collegamento

Le linee di collegamento di una rete sono realizzate con molti mezzi fisici diversi, tra i quali i cavi in doppino (quelli a cui è collegato il vostro telefono), i cavi coassiali (quelli a cui è collegato il vostro televisore), le fibre ottiche (il mezzo fisico più comunemente utilizzato per i collegamenti ad elevata ampiezza di banda e a lunga distanza) e lo spazio (il mezzo in cui si propagano le onde radio, le microonde e i fasci all'infrarosso). Qualunque sia il mezzo fisico, esso viene usato per propagare *segnali*, che sono onde elettromagnetiche che viaggiano alla velocità della luce (che, tuttavia, è una velocità dipendente dal mezzo attraversato: le onde elettromagnetiche attraversano il rame e le fibre ad una velocità pari a circa i due terzi di quella nel vuoto).

Una caratteristica importante di un'onda elettromagnetica è la *frequenza* (frequency), misurata in hertz, con cui l'onda stessa oscilla. La distanza tra una coppia di massimi o minimi d'onda adiacenti, tipicamente misurata in metri, viene detta *lunghezza d'onda* (wavelength) dell'onda stessa: dato che tutte le onde elettromagnetiche viaggiano alla velocità della luce, tale velocità divisa per la frequenza dell'onda è uguale alla sua lunghezza d'onda. Abbiamo già visto l'esempio di una linea telefonica adatta ai segnali vocali, che trasporta segnali elettromagnetici analogici nell'intervallo compreso tra 300 Hz e 3300 Hz; un'onda che viaggia nel rame a 300 Hz ha una lunghezza d'onda uguale a:

$$\begin{aligned} \text{Velocità della Luce nel Rame / Frequenza} \\ = 2/3 \times 3 \times 10^8 / 300 \\ = 667 \times 10^3 \text{ metri} \end{aligned}$$

Più in generale, le onde elettromagnetiche spaziano su un intervallo di frequenze molto più ampio, dalle onde radio alla luce infrarossa, alla luce visibile, ai raggi X e ai raggi gamma. La Figura 2.2 mostra lo spettro elettromagnetico e i mezzi fisici solitamente utilizzati per trasportare ciascuna banda di frequenze.

Fino ad ora abbiamo visto che una linea di connessione è un mezzo fisico in grado di trasportare segnali sotto forma di onde elettromagnetiche. Tali linee di connessione forniscano le basi per la trasmissione di qualsiasi tipo di informazione, compresi i dati di cui ci

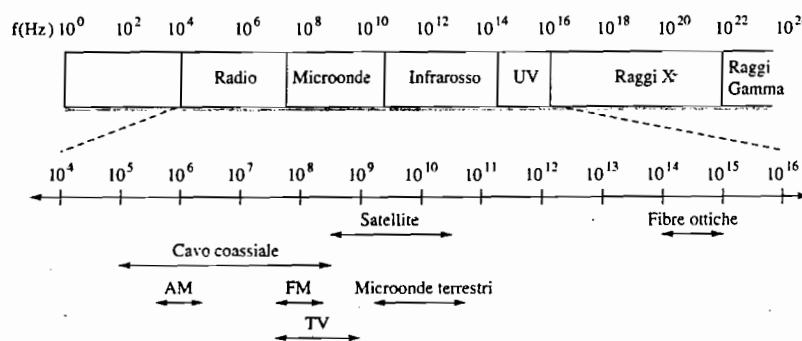


Figura 2.2 Lo spettro elettromagnetico.

2.1 Elementi hardware elementari

interessa la trasmissione: i dati binari (uni e zeri). Diciamo che i dati binari vengono *codificati* in un segnale; il problema di codificare dati binari in segnali elettromagnetiche è un argomento complesso: per facilitarne la comprensione, immaginiamo che sia suddiviso in due livelli. Il livello più basso ha a che fare con la *modulazione*, cioè la variazione della frequenza, dell'ampiezza o della fase del segnale per rendere possibile la trasmissione dell'informazione. Un semplice esempio di modulazione è la variazione della potenza (ampiezza) di una singola lunghezza d'onda, che è equivalente all'accensione e spegnimento di una lampadina. Poiché il problema della modulazione è di secondaria importanza nella nostra discussione delle linee di connessione come elementi costitutivi delle reti di calcolatori, ipotizziamo semplicemente che sia possibile trasmettere una coppia di segnali distinguibili (nei termini di segnale "alto" e "basso") e prendiamo in considerazione soltanto il livello superiore, che ha a che fare con il problema, molto più semplice, della codifica dei dati binari per mezzo di questi due segnali, codifiche che saranno presentate nella Sezione 2.2.

Un'altra caratteristica di una linea di connessione è il numero di flussi di bit che possono essere codificati nello stesso istante al suo interno. Se la risposta è "uno soltanto", allora i nodi connessi alla linea ne devono condividere l'accesso, come nel caso delle linee di connessione ad accesso multiplo descritte nelle Sezioni 2.6 e 2.7. Lungo le linee di connessione punto-punto, però, è spesso possibile trasmettere simultaneamente due flussi di bit, uno in ciascuna delle due direzioni: una linea con tali caratteristiche si dice *full-duplex*. Una linea di connessione che sia in grado di far fluire i dati in una sola direzione per volta (linea detta *half-duplex*) richiede che i due nodi ad essa connessi si alternino nel suo utilizzo, ma in questo libro ipotizzeremo sempre che le linee di connessione punto-punto siano full-duplex.

L'unica altra proprietà delle linee di connessione che ci interessa in questo momento è una caratteristica molto pratica: come facciamo ad allestirla? La risposta dipende dalla distanza che deve essere coperta dalla linea, da quanti soldi possiamo spendere e dal fatto che siamo in grado di utilizzare attrezzi per la movimentazione del terreno (escavatori, ecc.). Nel seguito trovate una panoramica dei diversi tipi di linee di collegamento che potete usare per costruire una rete di calcolatori.

Cavi

Se i nodi che volete connettere si trovano nella medesima stanza, nello stesso edificio o anche soltanto nello stesso sito (ad esempio, un campus universitario), allora potete semplicemente comprare un pezzo di cavo e stenderlo fisicamente tra i due nodi. Quale tipo di cavo scegliete con esattezza di installare, dipende dalla tecnologia che pensate di utilizzare per trasmettere i dati su quel collegamento: vedrete vari esempi nel seguito di questo capitolo. Per il momento, potete consultare in Tabella 2.1 un elenco dei tipi di cavi (e fibre) più comuni.

Tabella 2.1 I tipi più comuni di cavi e fibre per collegamenti locali.

Cavo	Ampiezza di banda tipiche	Distanze
Coppia a doppino in categoria 5	10-100 Mbps	100 m
Coassiale sottile per reti	10-100 Mbps	200 m
Coassiale grosso per reti	10-100 Mbps	500 m
Fibra multimodale	100 Mbps	2 km
Fibra monomodale	100-2400 Mbps	40 km

Tra questi, le coppie a doppino in categoria 5 (cat-5 twisted pair), caratterizzate da una guaina più spessa delle coppie in doppino che trovate a casa vostra, stanno rapidamente diventando la norma all'interno degli edifici. Per via delle difficoltà e dei costi relativi alla stesura di nuovi cavi in un edificio, molti sforzi vengono profusi per fare in modo che nuove tecnologie possano utilizzare cavi esistenti: ad esempio, Gigabit Ethernet è stata progettata per riutilizzare i cablaggi in categoria 5. Per connettere diversi edifici in uno stesso sito si usano, tipicamente, le fibre ottiche.

Linee a noleggio

Se i due nodi che volete connettere si trovano da parti opposte del Paese, oppure anche soltanto della città, non è possibile installare una linea di connessione in proprio. L'unica opzione praticabile è il noleggio di una linea di connessione dedicata presso una compagnia telefonica, nel qual caso tutto ciò che dovete fare è condurre una conversazione efficace con il personale dell'ufficio clienti della compagnia stessa. La Tabella 2.2 indica i servizi di connessione più comuni che è possibile noleggiare da una compagnia telefonica: anche in questo caso, troverete maggiori dettagli nel seguito di questo capitolo.

Anche se queste ampiezze di banda possono sembrare scelte a caso, si tratta di una stranezza che ha un significato. DS1 e DS3 (a volte chiamate, rispettivamente, T1 e T3) sono tecnologie abbastanza vecchie che furono definite, in origine, per i mezzi trasmissivi in rame. DS1 è equivalente all'aggregazione di 24 circuiti digitali per comunicazioni vocali a 64Kbps ciascuno, mentre DS3 equivale a 28 collegamenti di tipo DS1. Tutti i collegamenti di tipo STS- N sono in fibra ottica (STS significa Synchronous Transport Signal). STS-1 è la velocità della connessione base e ciascuna connessione STS- N ha un'ampiezza di banda pari a N volte quella di una connessione STS-1. Una connessione STS- N viene anche chiamata, a volte, OC- N (OC significa Optical Carrier), con una differenza marginale: il termine STS si riferisce alla trasmissione *elettrica* dei dispositivi connessi alla linea di collegamento, mentre il termine OC è relativo al segnale *ottico* che viene, in realtà, propagato lungo la fibra.

Ricordatevi che le compagnie telefoniche non "realizzano" la linea di connessione che abbiamo ordinato come un unico, ininterrotto tratto di cavo o di fibra, ma realizzano il collegamento mediante la propria rete. Nonostante la rete telefonica, storicamente, sia sempre stata molto diversa dal tipo di reti di cui parla questo libro, perché fu costruita principalmente per fornire un servizio vocale e usava la tecnologia a commutazione di circuito, la tendenza attuale va verso lo stile di reti descritto in questo libro, e in particolar modo verso la rete ATM (Asynchronous Transfer Mode) descritta nel Capitolo 3. Ciò non sorprenderà: esiste un enorme mercato per il trasporto di dati, voce e video.

Tabella 2.2 Tipiche ampiezze di banda disponibili dai fornitori di connessione.

Servizio	Ampiezza di banda
DS1	1.544 Mbps
DS3	44.736 Mbps
STS-1	51.840 Mbps
STS-3	155.250 Mbps
STS-12	622.080 Mbps
STS-48	2.488320 Gbps
STS-192	9.953280 Gbps

2.1 Elementi hardware elementari

In ogni caso, che la linea di connessione sia fisica o logica (realizzata attraverso la rete telefonica), il problema di costruire una rete di calcolatori utilizzando una serie di tali linee rimane identico, per cui procederemo come se ciascuna linea di connessione fosse realizzata da un singolo cavo o fibra. Soltanto quando avremo finito torneremo a considerare il fatto di aver costruito una rete di calcolatori sfruttando la rete telefonica esistente, oppure il fatto che la rete di calcolatori che avremo costruito potrà servire essa stessa come spina dorsale per la rete telefonica.

Connessione dell'ultimo miglio

Se non potete permettervi di noleggiare una linea dedicata (hanno prezzi variabili, approssimativamente, da un migliaio di dollari al mese per un collegamento DS1 da un punto all'altro del Paese, fino al prezzo definito come "se dovete chiedere quanto costa, significa che non potete permettervelo"), esistono alternative meno costose, che chiamiamo collegamenti "dell'ultimo miglio" ("last-mile" link) perché spesso si estendono per l'ultimo miglio che va da casa vostra al fornitore del servizio di rete. Questi servizi, riassunti in Tabella 2.3, connettono tipicamente una casa ad una rete esistente: ciò significa che probabilmente non sono adatti per costruire una rete da zero, ma se siete già riusciti a costruire una rete (perché magari "voi" siete una compagnia telefonica o una compagnia di servizi via cavo), allora potete usare questi collegamenti per raggiungere milioni di clienti.

La prima possibilità prevede l'utilizzo di un modem convenzionale sulla vecchia rete telefonica (POTS, plain old telephone service): oggi è possibile acquistare per meno di cento dollari un modem che trasmetta dati a 56 Kbps su una linea standard destinata alle comunicazioni vocali. La tecnologia, però, ha già raggiunto il limite per la sua ampiezza di banda, fenomeno che ha portato allo sviluppo della seconda alternativa: ISDN (Integrated Services Digital Network, rete digitale per servizi integrati). Una connessione ISDN comprende due canali a 64 Kbps, uno che può essere utilizzato per trasmettere dati ed un altro che viene utilizzato per la voce digitalizzata (un dispositivo che codifica la voce analogica per un collegamento ISDN digitale viene detto CODEC, *codificatore/decodificatore*). Quando il canale vocale non viene utilizzato, lo si può aggregare al canale dati per fornire un'ampiezza di banda per dati a 128 Kbps.

Per molti anni, la tecnologia ISDN è stata vista come il futuro per portare una modesta ampiezza di banda in tutte le case, ma oggi è stata ampiamente soppiantata da due nuove tecnologie: xDSL (digital subscriber line) e modem via cavo. La prima è, in realtà, un insieme di tecnologie che sono in grado di trasmettere dati ad alta velocità lungo le linee a doppio standard che già arrivano in quasi tutte le case degli Stati Uniti (e di molti altri Paesi). Quella maggiormente diffusa oggi è ADSL (asymmetric digital subscriber line): il nome stesso implica che ADSL fornisca una ampiezza di banda dal cliente verso la centrale della compagnia telefonica (flusso uscente, upstream) diversa da quella fornita dalla centrale verso

Tabella 2.3 Tipici servizi disponibili per connettere casa vostra alla rete.

Servizio	Ampiezza di banda
POTS	28.8-56 Kbps
ISDN	64-128 Kbps
xDSL	16 Kbps-55.2 Mbps
CATV	20-40 Mbps

il cliente (flusso entrante, downstream). Le ampiezze di banda esatte dipendono dalla lunghezza della linea che si trova tra il cliente e la centrale telefonica, linea che viene chiamata *anello locale* (local loop), come indicato in Figura 2.3, e che è realizzata con il doppino in rame esistente. Le ampiezze di banda di downstream variano da 1.544 Mbps (per una lunghezza di 6 Km) a 8.448 Mbps (per una lunghezza di 3 Km), mentre le ampiezze di banda di upstream variano da 16 Kbps a 640 Kbps.

Una tecnologia alternativa che non si è ancora diffusa, detta VDSL (very high data rate digital subscriber line), è simmetrica, con velocità dei dati variabile da 12.96 Mbps a 55.2 Mbps. VDSL può essere utilizzata su distanze molto più brevi, da 300 a 1500 metri, per cui tipicamente non raggiungerà la centrale telefonica da casa: la compagnia telefonica dovrà installare apparati hardware per la trasmissione VDSL in ciascun quartiere, usando qualche altra tecnologia (come, ad esempio, STS-N su fibra) per collegare tale apparato di quartiere con la centrale, come si può vedere in Figura 2.4. Questa configurazione viene a volte chiamata "fibra fino in prossimità", diversamente da schemi più ambiziosi, come "fibra a casa" e "fibra sul muretto del cortile".

I modem via cavo (coassiale) sono un'alternativa ai diversi schemi di DSL. Come suggerito dal nome, questa tecnologia sfrutta l'infrastruttura di cavi coassiali televisivi (CATV), che attualmente raggiunge il 95% delle case negli Stati Uniti (anche se soltanto il 65% delle famiglie vi è abbonata). Con questo approccio, alcuni dei canali di CATV disponibili, ciascuno dei quali ha un'ampiezza di banda di 6 MHz, vengono utilizzati per la trasmissione di dati digitali. CATV, come ADSL, è usata in modo asimmetrico, con velocità di downstream molto maggiori di quelle di upstream. La tecnologia è attualmente in grado di raggiungere i 40 Mbps in downstream su un singolo canale CATV, con 100 Mbps come limite teorico, mentre la velocità di upstream è circa la metà (cioè 20 Mbps) a causa di un decremento di un fattore 1000 del rapporto segnale-rumore. Inoltre, al traffico di upstream vengono dedicati meno canali CATV di quanti non ne siano dedicati al traffico downstream. Diversamente da DSL, l'ampiezza di banda è condivisa da tutti gli utilizzatori di un quartiere (un fatto che ha provocato alcune divertenti pubblicità da parte dei fornitori di DSL): ciò significa che bisogna usare qualche sistema per arbitrare l'accesso al mezzo condiviso, in modo simile a quanto avviene negli standard 802 descritti nel seguito del capitolo. Infine, come per la tecnologia DSL, è improbabile che i modem via cavo verranno usati per collegare un nodo A in un sito

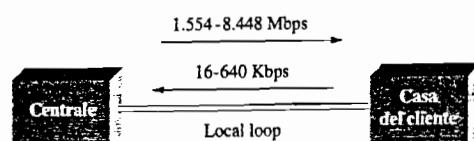


Figura 2.3 La tecnologia ADSL connette un cliente alla centrale telefonica mediante l'anello locale (*local loop*).



Figura 2.4 La tecnologia VDSL connette un cliente alla rete ottica di quartiere.

2.1 Elementi hardware elementari

Il teorema di Shannon a confronto con il vostro modem

Nelle aree correlate alla teoria dell'informazione e all'elaborazione di segnale è stato fatto un enorme lavoro per studiare tutti gli aspetti che vanno da come i segnali si degradano con la distanza a quanti dati possono effettivamente essere trasportati da un certo segnale. Il lavoro più significativo in questo settore consiste in una formula, nota come *teorema di Shannon*. In parole semplici, il teorema di Shannon fornisce un limite superiore alla capacità di una linea di connessione, in termini di bit al secondo (bps), come funzione del rapporto segnale-rumore della linea stessa, misurata in decibel (dB).

Il teorema di Shannon può essere utilizzato per determinare la velocità alla quale ci si aspetta che un modem possa trasmettere dati lungo una linea telefonica adatta ai segnali vocali senza dar luogo ad un tasso d'errore troppo elevato. Ad esempio, ipotizziamo che una tale linea di connessione ammetta frequenze nell'intervallo 300-3300 Hz.

Il teorema di Shannon viene tipicamente formulato in questo modo:

$$C = B \log_2 (1 + S/N)$$

dove C è la capacità disponibile del canale misurata in hertz, B è l'ampiezza di banda della linea (3300 Hz - 300 Hz = 3000 Hz), S è la potenza media del segnale e N è la potenza media di rumore. Il rapporto segnale-rumore (S/N) viene solitamente espresso in decibel, con la seguente relazione:

$$dB = 10 \times \log_{10} (S/N)$$

Assumendo come rapporto segnale-rumore un valore tipico di 30 dB (cioè $S/N = 1000$), si ottiene:

$$C = 3000 \times \log_2 (1000)$$

che equivale circa a 30 Kbps, molto vicino al limite di 28.8 Kbps di molti modем.

Visto questo limite fondamentale, come mai è possibile che i negozi di accessori per computer vendano modем a 56 Kbps? Uno dei motivi è che tale velocità dipende da un miglioramento della qualità delle linee di connessione, cioè da un rapporto segnale-rumore più elevato di 30 dB. Un altro motivo è che le modifiche del sistema telefonico hanno quasi del tutto eliminato le linee analogiche, che hanno l'ampiezza di banda limitata ad un massimo di 3300 Hz.

qualsiasi ad un nodo B in un sito diverso. Al contrario, i modem via cavo sono visti come un modo per collegare il nodo A di casa vostra con la compagnia che gestisce la rete via cavo, la quale definisce il modo in cui vi appare il resto della rete.

Collegamenti senza fili (wireless)

Il campo delle comunicazioni senza fili (wireless) è in forte espansione, sia economica che tecnologica. Il sistema AMPS (Advanced Mobile Phone System), che è stato per parecchi anni lo standard di telefonia cellulare negli Stati Uniti, è basato su tecnologia analogica e sta rapidamente cedendo il passo alla telefonia cellulare digitale. PCS (Personal Communication

Services) negli Stati Uniti e Canada e GSM (Global System for Mobile communication) nel resto del mondo. Tutti i tre sistemi usano, attualmente, un sistema di tralicci per trasmettere i segnali, sebbene si stia facendo qualche sforzo significativo per complementare questa infrastruttura con una griglia di satelliti orbitanti attorno alla Terra a medie e basse altitudini. Questi progetti, tra i quali citiamo ICO, Globalstar, Iridium e Teledesic, hanno avuto esiti incerti; quelli ancora attivi si sono concentrati principalmente sulla fornitura di servizi telefonici a quelle parti del globo, sempre più rare, in cui non è disponibile il servizio cellulare.

Pensando ad approcci un po' meno globali, si possono usare le bande di frequenza delle zone radio e infrarossa dello spettro elettromagnetico per realizzare collegamenti senza fili a breve distanza, come, ad esempio, all'interno di un ufficio o di un negozio, all'interno di complessi residenziali o campus. Nel caso infrarosso, si usano segnali con lunghezze d'onda di 850-950 nanometri per trasmettere dati a 1 Mbps per distanze di circa 10 metri. Questa tecnologia non richiede visibilità ottica tra i due punti collegati, ma è limitata ad ambienti interni. Nel caso di frequenze radio, sono disponibili diverse bande per la comunicazione di dati. Ad esempio, le bande intorno a 5.2 GHz e 17 GHz sono assegnate, in Europa, a HIPERLAN (High Performance European Radio LAN). Analogamente, in molti Paesi la banda intorno a 2.4 GHz è stata riservata per lo standard IEEE 802.11 per LAN senza fili (vi è un'ulteriore banda disponibile intorno a 5 GHz, ma sfortunatamente è soggetta ad interferenza da parte dei fornì a microonde). Lo standard IEEE 802.11, ancora in evoluzione, è in grado di fornire velocità di trasferimento dati fino a 54 Mbps e sarà discusso in maggiore dettaglio nella Sezione 2.8.

Un altro interessante sviluppo nel mondo wireless è l'interfaccia radio Bluetooth, che opera nella banda di frequenza attorno a 2.45 GHz. Bluetooth è stata progettata per brevi distanze, dell'ordine di 10 metri, con un'ampiezza di banda di 1 Mbps. I suoi sviluppatori hanno previsto il suo utilizzo in tutti i dispositivi (stampanti, workstation, laptop, proiettori, PDA, telefoni mobili), eliminando così la necessità di fili e cavi negli uffici (o anche fra vari dispositivi presenti sul vostro corpo, forse). Le reti di tali dispositivi vengono chiamate *piconet*.

2.2 Codifica (NRZ, NRZI, Manchester, 4B/5B)

Il primo passo per trasformare nodi e linee di collegamento in blocchi elementari utilizzabili per costruire reti richiede di capire come connetterli in modo che si possano trasmettere bit da un nodo all'altro. Come accennato nella sezione precedente, i segnali si propagano attraverso mezzi fisici, quindi l'obiettivo è quello di codificare i dati binari che il nodo sorgente vuole inviare in segnali che la linea di connessione sia in grado di trasportare, per poi decodificare il segnale nel nodo ricevente, riproducendo i dati binari corrispondenti. Non prendiamo in esame i dettagli relativi alla modulazione e ipotizziamo di lavorare con segnali a due valori: alto e basso. Dal punto di vista pratico, questi valori potrebbero corrispondere a due diverse tensioni elettriche in un collegamento di rame, oppure a due diversi livelli di potenza in un collegamento ottico.

Come abbiamo detto, la maggior parte delle funzioni di cui si parla in questo capitolo viene svolta da un adattatore di rete, un elemento hardware che collega un nodo ad una linea di connessione. L'adattatore di rete contiene un componente di segnalazione che codifica realmente i bit in segnali nel nodo sorgente e decodifica i segnali in bit nel nodo ricevente: in questo modo, come mostrato in Figura 2.5, i segnali viaggiano lungo una linea posta tra due componenti di segnalazione, mentre i bit fluiscono fra due adattatori di rete.

2.2 Codifica (NRZ, NRZI, Manchester, 4B/5B)

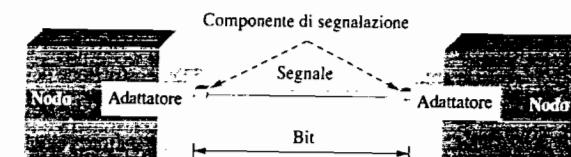


Figura 2.5 I segnali viaggiano tra due componenti di segnalazione: i bit fluiscano tra due adattatori.

Torniamo ora al problema della codifica di bit in segnali. La cosa più naturale da fare è stabilire una corrispondenza tra i dati di valore 1 ed il segnale alto e tra i dati di valore 0 ed il segnale basso: esattamente questo è lo schema usato da una strategia di codifica denominata, in modo poco comprensibile, *senza ritorno a zero* (NRZ, non-return to zero). Ad esempio, la Figura 2.6 rappresenta schematicamente il segnale codificato con NRZ (in basso) che corrisponde alla trasmissione di una certa sequenza di bit (in alto).

Il problema della codifica NRZ è che una sequenza di molti valori 1 consecutivi porta il segnale a rimanere alto sulla linea per un lungo periodo di tempo e, analogamente, molti valori 0 consecutivi lo fanno rimanere basso per lungo tempo. Queste lunghe sequenze di valori 1 o 0 provocano due problemi. Il primo consiste nel fenomeno denominato *variazione del valore di riferimento*. Nello specifico, il componente che funge da ricevitore del segnale calcola continuamente la media del segnale ricevuto e usa tale valore medio per discriminare tra segnali alti e bassi: ogni volta che il segnale è sensibilmente inferiore a tale valore medio, il ricevitore decide di aver visto un valore 0 e, analogamente, un segnale sensibilmente superiore al valore medio viene interpretato come valore 1. Il problema, ovviamente, è che troppi valori 1 o 0 consecutivi possono modificare il valore medio, rendendo più difficile la rilevazione di una variazione sensibile nel segnale.

Il secondo problema è che frequenti transizioni dal valore alto al valore basso, e viceversa, sono necessarie per la *ricostruzione del segnale di sincronismo* (clock recovery). Dal punto di vista intuitivo, il problema della ricostruzione del segnale di sincronismo consiste nel fatto che sia il processo codificatore che il processo decodificatore sono governati da un segnale di sincronismo (clock): ad ogni ciclo di clock la sorgente trasmette un bit ed il ricevitore identifica un bit. I clock del mittente e del destinatario devono essere sincronizzati in modo preciso per fare in modo che il ricevitore identifichi lo stesso bit che è stato inviato dalla sorgente: se il clock del ricevitore è appena un po' più veloce o più lento del clock della sorgente, il segnale non viene decodificato correttamente. Si potrebbe pensare di inviare il segnale di clock al ricevente lungo una connessione a sé stante, ma ciò viene tipicamente evitato perché porta ad un raddoppio dei costi di cablaggio. Diversamente, quindi, il ricevitore ricostruisce il segnale di clock dal segnale ricevuto, mediante, appunto, il processo di ricostruzione del segnale di sincronismo. Ogni volta che il segnale cambia valore, effettuan-

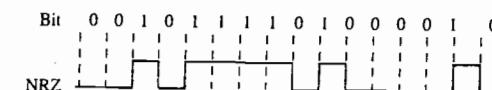


Figura 2.6 Codifica NRZ di un flusso di bit.

do una transizione da 1 a 0 o da 0 a 1, il ricevitore si accorge di trovarsi al limite di un ciclo di clock e si può sincronizzare. Tuttavia, se trascorre molto tempo senza che avvenga alcuna transizione, il clock è soggetto al fenomeno di deriva, per cui la ricostruzione del clock richiede che vi siano molte transizioni nel segnale, indipendentemente dai dati che vengono inviati.

Un approccio, chiamato *senza ritorno a zero invertito* (NRZI), illustrato in Figura 2.7) e che tenta di risolvere questo problema, prevede che la sorgente esegua una transizione di segnale per codificare un valore 1 e mantenga, invece, il segnale al livello attuale per codificare un valore 0. In questo modo si risolve il problema di valori 1 consecutivi, ma ovviamente non lo si risolve per valori 0 consecutivi. Un'alternativa, denominata *codifica Manchester*, rende più esplicito il compito di fondere il clock con il segnale, trasmettendo l'OR esclusivo del clock e dei dati codificati con NRZ (potete pensare al segnale di clock come ad un segnale interno che assume alternativamente i valori alto e basso; una coppia di periodi con valori alto e basso rappresenta un ciclo di clock). Anche la codifica Manchester è visibile in Figura 2.7. Osservate come la codifica Manchester produca come risultato che un valore 0 viene rappresentato da una transizione da basso ad alto, mentre un valore 1 viene rappresentato da una transizione da alto a basso. Poiché sia i valori 0 sia i valori 1 provocano una transizione del segnale, il ricevitore può ricostruire il clock in modo efficiente. Esiste anche una variante della codifica Manchester, che prende il nome di *Manchester differenziale*, in cui un valore 1 viene codificato con un segnale che nella prima metà del ciclo di clock ha lo stesso valore che aveva nella seconda metà del bit precedente, mentre un valore 0 viene codificato con un segnale che nella prima metà del ciclo di clock ha il valore opposto a quello assunto nella seconda metà del bit precedente.

Il problema dello schema di codifica Manchester è che si raddoppia la velocità a cui avvengono le transizioni di segnale lungo la linea di connessione: di conseguenza, il ricevitore ha a disposizione la metà del tempo per identificare ciascun impulso di segnale. La velocità di transizione di un segnale viene chiamata *velocità di baud* (baud rate) di una linea; nel caso della codifica Manchester, la velocità di bit è la metà della velocità di baud, per cui la codifica viene considerata efficiente soltanto al 50%. Tenete presente che se il ricevitore fosse in grado di mantenere il passo rispetto alla più elevata velocità di baud richiesta dalla codifica Manchester in Figura 2.7, sia NRZ sia NRZI avrebbero potuto trasmettere un numero di bit doppio nello stesso periodo di tempo.

L'ultima codifica che prendiamo in esame, chiamata 4B/5B, tenta di risolvere il problema dell'inefficienza della codifica Manchester senza presentare il problema di prolungati periodi di segnale allo stesso livello, alto o basso. L'idea della codifica 4B/5B consiste nell'inserire dei bit extra nel flusso di bit da trasmettere, in modo da interrompere eventuali

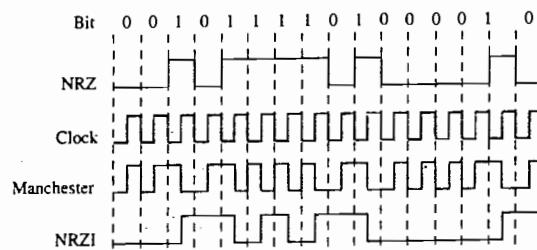


Figura 2.7 Varie strategie di codifica.

Velocità di bit e baud

Spesso si usano indifferentemente i termini *velocità di bit* (bit rate) e *velocità di baud* (baud rate), anche se, come vedremo con la codifica Manchester, non sono la stessa cosa. Mentre la codifica Manchester è un esempio in cui la velocità di baud di una linea è maggiore della sua velocità di bit, è anche possibile avere una velocità di bit maggiore della velocità di baud: ciò richiederebbe la codifica di più di un bit mediante ciascun impulso trasmesso lungo la linea.

Per capire come ciò possa avvenire, supponete di poter trasmettere su una linea quattro segnali distinguibili, anziché due. Su una linea analogica, ad esempio, questi quattro segnali potrebbero corrispondere a quattro diverse frequenze. Con quattro segnali diversi, si possono codificare due bit di informazione con ciascun segnale: il primo segnale significa 00, il secondo significa 01, e così via. Ora, una sorgente (ricevente) che sia in grado di trasmettere (identificare) 1000 impulsi al secondo, sarebbe in grado di inviare (ricevere) 2000 bit di informazioni al secondo. Sarebbe, cioè, un collegamento a 1000 baud e a 2000 bps.

lunghe sequenze di valori 1 o 0. Nello specifico, ogni sequenza di 4 bit di dati viene codificata con 5 bit che si trasmettono veramente al ricevitore, da cui il nome 4B/5B. Il codice a 5 bit viene scelto in modo che ciascuna parola di codice non abbia più di un valore 0 iniziale e non abbia più di due valori 0 finali: in questo modo, quando vengono inviate una di seguito all'altra, nessuna coppia di parole di codice a 5 bit produce una sequenza con più di tre valori 0 consecutivi. Le parole di 5 bit così identificate vengono poi trasmesse usando la codifica NRZI, dando così ragione del fatto che il codice si preoccupi soltanto degli zeri consecutivi, essendo il problema dei valori 1 consecutivi già risolto da NRZI. Notate anche che la codifica 4B/5B ha un'efficienza dell'80%.

La Tabella 2.4 fornisce le parole di codice a 5 bit che corrispondono a ciascuna delle 16 possibili sequenze di 4 bit di dati. Notate che esistono 16 parole di codice non utilizzate, che possiamo usare per altri scopi, dal momento che con 5 bit si possono codificare 32 informazioni diverse, mentre per i dati ce ne servono soltanto 16. Di questi codici inutilizzati, la parola 11111 viene usata quando la linea è inattiva (*idle*), la parola 00000 corrisponde ad una linea fuori uso e la parola 00100 ha l'effetto di interrompere le comunicazioni (*halt*). Delle 13 parole di codice rimanenti, 7 non sono valide perché violano la regola di un solo zero iniziale e due soli zeri finali, mentre le altre 6 rappresentano vari simboli di controllo. Come vedremo più avanti in questo capitolo, alcuni protocolli di framing (come, ad esempio, FDDI) fanno uso di questi simboli.

2.3 Tramatura (framing)

Ora che abbiamo visto come trasmettere una sequenza di bit lungo una linea di connessione punto-punto (cioè c'è un adattatore ad un altro), consideriamo lo scenario presentato in Figura 2.8. Ricordate, dal Capitolo 1, che ci stiamo interessando di reti a commutazione di pacchetto, per cui fra i nodi non vengono scambiati flussi di bit, ma blocchi di dati, detti *frame* a questo livello. L'adattatore di rete consente ai nodi di scambiarsi frame. Quando il nodo A vuole trasmettere un frame al nodo B, invia al suo adattatore il comando di trasmet-

Tabella 2.4 Codifica 4B/5B.

Parola di dati a 4 bit	Parola di codice a 5 bit
0000	11110
0001	01001
0010	10100
0011	10101
0100	01010
0101	01011
0110	01110
0111	01111
1000	10010
1001	10011
1010	10110
1011	10111
1100	11010
1101	11011
1110	11100
1111	11101

tere il frame, leggendolo dalla memoria del nodo, dando così luogo all'invio di una sequenza di bit lungo la linea di connessione. Quindi, l'adattatore del nodo B raccoglie la sequenza di bit che arriva dalla linea e scrive nella memoria del nodo B il frame corrispondente. L'esatta identificazione di quale insieme di bit costituisca un frame (cioè la determinazione dell'inizio e della fine del frame) è il compito primario e complesso che deve essere affrontato e risolto dall'adattatore.

Questo problema, il problema del *framing*, può essere risolto in vari modi e in questa sezione vedrete alcuni protocolli diversi che mostrano punti diversi dello spazio delle soluzioni di progetto. Notate che, sebbene noi discutiamo di framing in relazione a linee di connessione di tipo punto-punto, il problema rimane fondamentale anche per reti ad accesso multiplo, quali Ethernet e token ring.

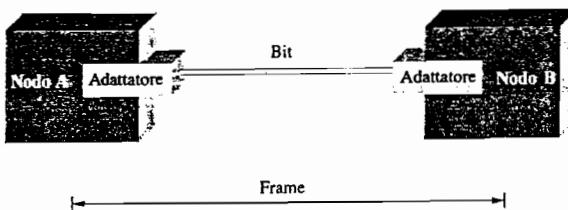


Figura 2.8 I bit fluiscono fra gli adattatori, i frame fra gli host.

2.3.1 Protocolli orientati ai byte (BISYNC, PPP, DDCMP)

Uno dei primi approcci al problema del framing, che vede le sue origini nella connessione di terminali a calcolatori di tipo mainframe, prevede di considerare ciascun frame come un insieme di byte (caratteri) piuttosto che come un insieme di bit. Tale *approccio orientato ai byte* ha come esempi il protocollo BISYNC (Binary Synchronous Communication) sviluppato da IBM verso la fine degli anni '60 e il protocollo DDCMP (Digital Data Communication Message Protocol) usato nella rete DECNET di Digital Equipment Corporation. A volte questi protocolli ipotizzano di aver a che fare con un particolare insieme di caratteri, ma ciò non è sempre vero (ad esempio, BISYNC può usare ASCII ed EBCDIC, nonché il codice Transcode a 6 bit di IBM).

Anche se per molti aspetti simili, questi due protocolli sono esempi di due diverse tecniche di framing, l'approccio con sentinella e l'approccio a conteggio di byte.

Approccio con sentinella

Il protocollo BISYNC è un esempio di soluzione del problema di framing tramite sentinella; il formato del suo frame è illustrato in Figura 2.9, che è la prima delle molte che vedrete in questo libro, usate per mostrare il formato di frame o di pacchetti, per cui sono opportune alcune parole di chiarimento. Un pacchetto viene rappresentato come una sequenza di campi etichettati, e sopra ciascun campo è posto un numero che indica la lunghezza in bit del campo stesso. Notate, anche, che i pacchetti vengono trasmessi a partire dal campo più a sinistra.

L'inizio di un frame è segnalato dall'invio del carattere speciale SYN (*synchronization*). La parte di dati del frame è poi racchiusa da speciali *caratteri sentinella*: STX (*start of text*, inizio del testo) e ETX (*end of text*, fine del testo). Il campo SOH (*start of header*, inizio dell'intestazione) serve più o meno allo stesso scopo del campo STX. Il problema relativo all'approccio con sentinella consiste, ovviamente, nel fatto che il carattere ETX potrebbe comparire anche nella porzione del frame destinata ai dati. BISYNC risolve questo problema facendo precedere il carattere ETX da un carattere DLE (*data-link escape*) ogni volta che esso appare nel corpo del frame (questa tecnica viene chiamata "escape"); anche il carattere DLE viene sostituito allo stesso modo, quando appare nel corpo del frame, facendolo precedere da un ulteriore carattere DLE (i programmati in linguaggio C noteranno che questa tecnica è analoga al modo in cui un carattere di "virgolette" viene preceduto all'interno di una stringa da un carattere di "barra rovesciata"). Questo approccio viene spesso chiamato *interposizione di caratteri* (character stuffing), perché nella porzione dedicata ai dati nel frame vengono inseriti caratteri extra.

Il formato del frame comprende anche un campo denominato CRC (*cyclic redundancy check*, verifica a ridondanza ciclica), usato per rilevare errori di trasmissione con vari algoritmi che verranno presentati nella Sezione 2.4. Infine, il frame contiene ulteriori campi di intestazione usati, fra gli altri scopi, per gli algoritmi di consegna affidabile a livello di collegamento: esempi di tali algoritmi saranno forniti nella Sezione 2.5.

Il più recente protocollo PPP (Point-to-Point Protocol), solitamente in esecuzione sui collegamenti via modem in modalità di selezione telefonica (dial-up), è simile a BISYNC, in quanto usa l'interposizione di caratteri. Il formato del frame PPP è indicato in Figura 2.10. Il

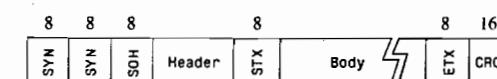


Figura 2.9 Il formato dei frame BISYNC.

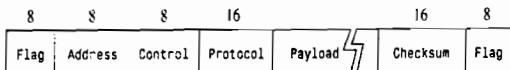


Figura 2.10 Il formato dei frame PPP.

carattere speciale che denota l'inizio del testo, rappresentato dal campo **Flag** nella Figura 2.10, è **01111110**. I campi **Address** e **Control** solitamente contengono valori predefiniti e sono, quindi, poco interessanti. Il campo **Protocol** è usato per realizzare il demultiplexing, in quanto identifica il protocollo di livello superiore, come IP o IPX (un protocollo simile a IP sviluppato da Novell). La dimensione del carico utile del frame (**Payload**) può essere negoziata, altrimenti è di 1500 byte. Il campo **Checksum** è lungo 2 (per default) o 4 byte.

Il formato del frame PPP è insolito, in quanto le dimensioni di alcuni suoi campi vengono negoziate, anziché essere prestabilite. Questa negoziazione avviene mediante un protocollo chiamato LCP (Link Control Protocol), che lavora in coppia con PPP: LCP invia dapprima messaggi di controllo incapsulati in frame PPP, contrassegnandoli opportunamente nel campo **Protocol** di PPP, quindi modifica il formato dei frame PPP in base alle informazioni contenute in tali messaggi di controllo. LCP è coinvolto anche nell'instaurazione di un collegamento tra due controparti, nel caso in cui entrambe rilevino il segnale portante della trasmissione.

Approccio a conteggio di byte

Come ben sanno tutti gli studenti di Fondamenti di Informatica, l'alternativa alla rilevazione della fine di un file mediante sentinella è l'inserimento della dimensione del file all'inizio del file stesso. Lo stesso vale per il framing: il numero di byte contenuti in un frame può essere inserito come campo nell'intestazione del frame stesso. Il protocollo DDCMP di DECNET usa proprio questo approccio, come evidenziato in Figura 2.11. In questo esempio, il campo **Count** specifica quanti bit siano contenuti nel corpo del frame.

Un pericolo insito in questo approccio è che un errore di trasmissione potrebbe corrompere il campo **Count**, nel qual caso la fine del frame non verrebbe identificata correttamente (esiste un problema simile anche nell'approccio con sentinella, nel caso in cui risulti corrotto il campo ETX). Se ciò dovesse accadere, il ricevitore immagazzinerebbe tanti byte quanti sono indicati dal campo **Count** errato e poi userebbe il campo per la rilevazione d'errore per capire che il frame è errato: questa situazione viene a volte chiamata *errore di framing*. Il ricevitore rimarrà poi in attesa di osservare il successivo carattere SYN, per poi iniziare a memorizzare i byte che formano il frame seguente. Di conseguenza, è possibile che un errore di framing provochi la ricezione errata di più frame consecutivi.

2.3.2 Protocolli orientati ai bit (HDLC)

Diversamente da questi protocolli orientati ai byte, un protocollo orientato ai bit non si interessa dei confini tra byte: considera semplicemente il frame come una sequenza di bit, che

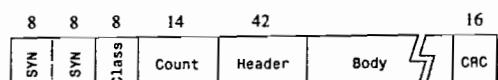


Figura 2.11 Il formato dei frame DDCMP.

possono provenire da un insieme di caratteri, come l'insieme ASCII, oppure possono essere valori di pixel in un'immagine, oppure, ancora, istruzioni e operandi di un file eseguibile. Il protocollo SDLC (Synchronous Data Link Control) sviluppato da IBM è un esempio di protocollo orientato ai bit, protocollo che venne successivamente standardizzato da ISO come protocollo HDLC (High-Level Data Link Control). Nella discussione seguente useremo come esempio il protocollo HDLC, il cui formato di frame è mostrato nella Figura 2.12.

HDLC contrassegna l'inizio e la fine del frame con la sequenza di bit **01111110**, che viene trasmessa anche durante tutto il tempo in cui la linea di connessione è inattiva, in modo che sorgente e ricevitore possano mantenere sincronizzati i propri segnali di clock. In questo modo entrambi i protocolli usano, essenzialmente, l'approccio a sentinella. Poiché tale sequenza può comparire dovunque all'interno del corpo del frame (e, inoltre, i bit **01111110** possono trovarsi a cavallo del confine tra due byte), i protocolli orientati ai bit usano una tecnica denominata *interposizione di bit* (bit stuffing), analoga all'uso del carattere DLE.

L'interposizione di bit in HDLC funziona in questo modo: ogni volta che la sorgente invia cinque valori 1 consecutivi all'interno del corpo del messaggio (senza considerare, cioè, le trasmissioni della sequenza caratteristica, **01111110**), inserisce un valore 0 prima di trasmettere il bit successivo. Alla ricezione, qualora arrivino cinque valori 1 consecutivi, il ricevitore si comporterà in modo conseguente al successivo bit ricevuto: se si tratta di un valore 0, significa che deriva dall'interposizione e viene eliminato; se, invece, si tratta di un valore 1, allora si tratta del marcatore di fine frame, oppure nel flusso di bit è stato introdotto un errore. Osservando il bit ancora successivo, il ricevitore può distinguere fra questi due ultimi casi: se osserva un valore 0 (e, quindi, gli ultimi 8 bit ricevuti sono stati **01111110**), allora si tratta del marcatore di fine frame; se, invece, osserva un valore 1 (e, quindi, gli ultimi 8 bit ricevuti sono stati **01111111**), allora deve essersi verificato un errore e l'intero frame viene eliminato. In tal caso, il ricevitore dovrà attendere il successivo **01111110** prima di poter iniziare a ricevere di nuovo: di conseguenza, è possibile che il ricevitore fallisca nella ricezione di due frame consecutivi. Naturalmente, esistono ancora situazioni in cui sono possibili errori di frame che non vengono rilevati, come quando un errore genera un'intera sequenza di fine frame indesiderata, ma questi eventi sono piuttosto improbabili. Modalità robuste di rilevazione d'errore saranno presentate nella Sezione 2.4.

Una caratteristica interessante dell'interposizione di bit, così come dell'interposizione di byte, è che la dimensione del frame dipende dai dati che vengono inviati come carico utile del frame stesso: infatti, non è possibile che tutti i frame abbiano la stessa dimensione, poiché i dati che si possono trovare in ciascun frame sono totalmente arbitrari (per convincervene, considerate ciò che accade se l'ultimo byte del corpo di un frame è il carattere ETX). Un tipo di framing che garantisce la stessa dimensione per tutti i frame è quello descritto nella sottosezione seguente.

2.3.3 Framing basato sul clock (SONET)

Un terzo approccio al framing è rappresentato dallo standard SONET (Synchronous Optical Network): in mancanza di un termine universalmente accreditato, ci riferiamo a questo ap-

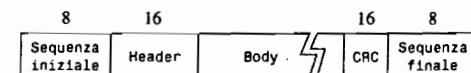


Figura 2.12 Il formato dei frame HDLC.

Cosa c'è in uno strato?

Uno dei contributi più importanti del modello di riferimento OSI presentato nel Capitolo I consiste nel mettere a disposizione un insieme di termini adatti alla discussione di protocolli e, in particolare, di strati di protocolli. Questo vocabolario ha fornito molti spunti per argomentazioni del tipo "Il tuo protocollo svolge la funzione X nello strato Y, mentre il modello di riferimento OSI dice che dovrebbe essere realizzata nello strato Z: ciò costituisce una violazione di strato". La realtà è che identificare il corretto strato in cui realizzare una certa funzione può essere molto difficile, spesso per motivi molto più sottili rispetto a "Cosa dice in proposito il modello OSI?". Proprio questo è uno dei motivi per cui questo libro evita di utilizzare un approccio rigidamente a strati: al contrario, vi mostra molte delle funzioni che devono essere realizzate dai protocolli ed esamina alcuni dei modi in cui ciò è stato fatto con successo.

Nonostante il nostro approccio non rigidamente stratificato, a volte ci serve un modo comodo per parlare di classi di protocolli: in questo caso, spesso il nome dello strato a cui essi operano è la scelta migliore. Per cui, ad esempio, questo capitolo tratta principalmente protocolli dello strato di connessione (la codifica di bit, descritta nella Sezione 2.2, costituisce un'eccezione, essendo considerata una funzione dello strato fisico). I protocolli dello strato di connessione si possono identificare per la caratteristica di operare su una singola linea di connessione, il tipo di rete di cui parla questo capitolo. I protocolli dello strato di rete, al contrario, operano su reti commutate, che contengono molte linee interconnesse da switch o router. Gli argomenti relativi ai protocolli dello strato di rete sono discussi nei Capitoli 3 e 4.

Notate come gli strati di protocolli siano utili: forniscono un modo semplice per parlare di classi di protocolli e ci aiutano a suddividere il problema della costruzione di reti in sottoproblemi gestibili. Tuttavia, gli strati non vanno considerati in modo inutilmente restrittivo: il solo fatto che qualcosa rappresenti una violazione di strato non pone termine alla discussione che riguarda la sua utilità. In altre parole, la stratificazione rappresenta un valido strumento, ma non deve essere vincolante. Nel Capitolo 6, quando parleremo di controllo di congestione, vedremo una discussione particolarmente interessante riguardo a quale sia lo strato migliore per tale funzione.

proccio chiamandolo semplicemente *framing basato sul clock*. Il primo annuncio di SONET venne da Bell Communications Research (Bellcore), per poi essere sviluppato da ANSI (American National Standards Institute) per le trasmissioni digitali in fibra ottica e successivamente adottato da ITU-T, anche se chi ha standardizzato le cose non è un argomento molto interessante. Ciò che va ricordato di SONET è che si tratta dello standard predominante per la trasmissione di dati a lunga distanza su reti in fibra ottica.

Prima di procedere oltre, bisogna rimarcare come la specifica completa di SONET sia sostanzialmente più estesa di questo stesso libro, per cui la discussione che segue coprirà, necessariamente, soltanto i punti principali dello standard. Inoltre, SONET si occupa sia del problema del framing, sia di quello della codifica, oltre ad un problema molto importante per le compagnie telefoniche, il multiplexing di diverse linee a bassa velocità su una linea ad alta velocità. Iniziamo con il framing, per discutere gli altri argomenti in seguito.

Come negli schemi di framing presentati in precedenza, anche il frame SONET contiene alcune informazioni speciali che indicano al ricevitore dove inizia e dove termina il frame, però a questo punto praticamente finiscono le somiglianze. Va notato che non viene usata

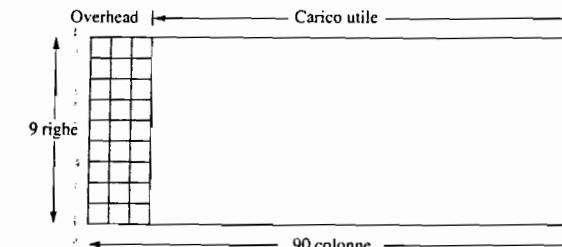


Figura 2.13 Un frame SONET STS-1.

alcuna interposizione di bit, per cui la lunghezza del frame non dipende dai dati che vengono inviati; sorge, quindi, spontanea una domanda: come fa il ricevitore a sapere dove inizia e dove termina ciascun frame? Esaminiamo questo problema per la linea SONET a più bassa velocità, nota con il nome di STS-1 e operante a 51.84 Mbps. La Figura 2.13 mostra un frame STS-1, disposto in nove righe di 90 byte ciascuna, con i primi 3 byte di ogni riga che costituiscono informazioni aggiuntive (*overhead*), mentre il resto è disponibile per i dati che si stanno trasmettendo lungo la linea. I primi 2 byte di ciascun frame contengono uno schema di bit speciale: sono questi byte che consentono al ricevitore di determinare l'inizio del frame. Tuttavia, non essendo utilizzata l'interposizione di bit, non c'è nessuna ragione per cui tale schema non possa presentarsi, di tanto in tanto, all'interno del carico utile del frame. Per proteggersi da ciò, il ricevitore cerca costantemente lo schema di bit speciale, sperando di vederlo comparire una volta ogni 810 byte, perché ciascun frame è lungo $9 \times 90 = 810$ byte. Quando lo schema speciale compare nel momento giusto per un numero di volte sufficiente, il ricevitore conclude di aver raggiunto il sincronismo e di poter quindi interpretare correttamente i frame.

Una delle cose di cui non parliamo, a causa della complessità di SONET, è l'utilizzo dettagliato di tutti gli altri bit aggiuntivi. Una parte di tale complessità si può attribuire al fatto che SONET viene utilizzato in reti ottiche che attraversano i confini di vari gestori, e non soltanto su un'unica linea di collegamento (ricordate che stiamo glissando sul fatto che le compagnie realizzino, in realtà, una rete vera e propria, concentrandoci invece sulla possibilità di noleggiare una linea SONET da esse, usando poi tale linea per costruire la nostra rete a commutazione di pacchetto). Ulteriore complessità deriva dal fatto che SONET fornisce un insieme di servizi significativamente più ricco del semplice trasporto di dati: ad esempio, un canale a 64Kbps, facente parte della capacità globale di una linea SONET, viene accantonato per un canale vocale usato per la manutenzione.

I bit aggiuntivi (*overhead*) di un frame SONET sono codificati usando NRZ, la semplice codifica descritta in una precedente sezione, dove i valori 1 sono segnali alti e i valori 0 sono segnali bassi. Tuttavia, per garantire che ci sia un numero sufficiente di transizioni per consentire al ricevitore di ricostruire il segnale di clock della sorgente, i byte del carico utile vengono *trasposti* (scrambled), calcolando l'OR esclusivo (XOR) tra i dati da trasmettere e uno schema di bit ben noto. Tale schema, lungo 127 bit, ha molte transizioni da 1 a 0, in modo che il risultato del suo OR esclusivo con i dati trasmessi fornisca con elevata probabilità un segnale con un numero di transizioni sufficiente a rendere possibile la ricostruzione del clock.

SONET fornisce supporto per il multiplexing di più linee a bassa velocità nel modo seguente: una linea SONET ha una velocità che appartiene ad un insieme finito di valori possibili, spaziando da 51.84 Mbps (STS-1) a 2488.32 Nbps (STS-48) e oltre (la Tabella 2.2 nella Sezione 2.1 contiene l'insieme completo delle velocità di dati possibili per SONET). Si noti che tutte queste velocità sono multiple di STS-1. Per il framing, la cosa importante è che un singolo frame SONET può contenere sottoframe di più canali a velocità inferiore. Una seconda caratteristica, a questa correlata, è che ciascun frame ha una durata di 125 μ s: di conseguenza, alla velocità di STS-1, un frame SONET è lungo 810 byte, mentre alla velocità di STS-3 ciascun frame SONET è lungo 2430 byte. Notate la sinergia tra queste due caratteristiche: $3 \times 810 = 2430$, per cui tre frame STS-1 trovano spazio esattamente in un solo frame STS-3.

Dal punto di vista intuitivo, si può immaginare che il frame STS- N sia composto di N frame STS-1, con i byte dei diversi frame intercalati: prima viene trasmesso un byte del primo frame, poi un byte del secondo frame, e così via. Il motivo per cui si intercalano i byte prelevandoli a rotazione da diverse linee STS-1 è per garantire che i byte provenienti da ciascuna linea STS-1 fluiscano a velocità costante, arrivando al ricevitore a 51 Mbps, piuttosto che tutti insieme durante una particolare frazione dell'intervallo di 125 μ s.

Anche se considerare un segnale STS- N come un modo da utilizzare per eseguire il multiplexing di N frame STS-1 è una visione corretta, si possono, in alternativa, raggruppare insieme i carichi utili di questi frame STS-1 per formare un più grande carico utile STS- N : una tale linea di connessione viene denominata STS- Nc (dove la lettera c significa *concatenato*): a questo scopo viene utilizzato uno dei campi presenti nello spazio dei bit aggiuntivi. La Figura 2.14 rappresenta schematicamente la concatenazione nel caso di tre frame STS-1 che vengono raggruppati per formare un unico frame STS-3c. Il motivo per cui una linea SONET viene identificata come STS-3c piuttosto che STS-3 sta nel fatto che nel primo caso l'utente può vedere la linea come un singolo flusso a 155.25 Mbps, mentre una linea STS-3 deve realmente essere considerata come tre linee a 51.84 Mbps che condividono la stessa fibra.

Per finire, la precedente descrizione di SONET è oltremodo semplificata, perché ipotizza che il carico utile di ciascun frame sia completamente contenuto all'interno del frame stesso (e perché mai non dovrebbe esserlo?). In realtà, il frame STS-1 dovrebbe essere considerato come un semplice segnaposto per il frame, il cui vero carico utile può collocarsi attorno al confine esistente tra frame diversi, come evidenziato nella Figura 2.15: qui possiamo vedere sia il carico utile del frame STS-1 posizionato a cavallo del confine fra due frame STS-1, sia il carico utile spostato a destra di qualche byte e che, quindi, viene traslato nella parte iniziale (*wrap around*). Uno dei campi nei bit aggiuntivi del frame punta proprio all'inizio del carico utile. Questa caratteristica è utile perché semplifica il problema della sincronizzazione dei clock usati nell'intera rete dei gestori, una cosa di cui essi si preoccupano molto.

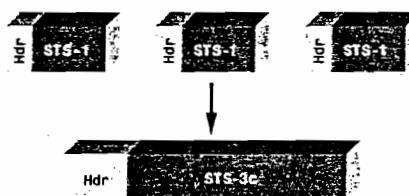


Figura 2.14 Tre frame STS-1 multiplati in un unico frame STS-3c.

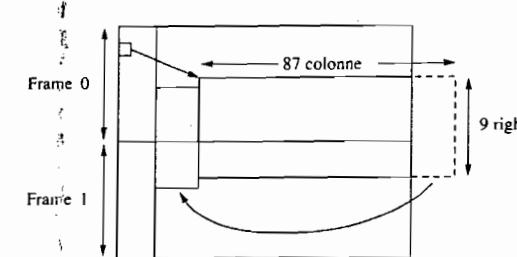


Figura 2.15 Frame SONET fuori fase.

2.4 Rilevazione d'errore

Come detto nel Capitolo 1, a volte nei frame vengono introdotti bit errati, ad esempio a causa di interferenza elettrica o rumore termico. Nonostante questi errori, specialmente sulle linee di collegamento ottiche, siano rari, è necessario disporre di qualche meccanismo per la loro rilevazione, in modo da poter intervenire con azioni correttive. In caso contrario, l'utente finale rimane stupefatto da un programma C che, compilato con successo un attimo prima, contiene ora improvvisamente un errore di sintassi, quando tutto ciò che è accaduto nel frattempo è la copiatura del file su un file system di rete.

Esiste una lunga tradizione di tecniche per la gestione di bit errati nei sistemi di calcolatori, risalente ai codici di Hamming e Reed/Solomon, sviluppati per un utilizzo nella memorizzazione dei dati su dischi magnetici e nelle prime memorie interne. In questa sezione verranno descritte alcune delle tecniche di rilevazione d'errore più comunemente utilizzate nelle reti.

Rilevare gli errori è soltanto un aspetto del problema: l'altro aspetto è la correzione degli errori individuati. Quando il destinatario di un messaggio individua un errore, possono essere intraprese, fondamentalmente, due azioni diverse. La prima consiste nell'avvertire il mittente che il messaggio è stato corrotto, in modo che possa ritrasmetterne una copia: se gli errori di bit sono rari, con buona probabilità la copia ritrasmessa ne sarà esente. In alternativa, esistono alcune categorie di algoritmi di rilevazione d'errore che consentono al destinatario di ricostruire il messaggio corretto anche dopo che è stato corrotto: tali algoritmi si basano su *codici a correzione d'errore*, come vedremo in seguito.

Una delle tecniche più usate per la rilevazione di errori di trasmissione è quella nota come *verifica di ridondanza ciclica* (cyclic redundancy check, CRC), usata praticamente in tutti i protocolli del livello di linea che abbiamo visto nella sezione precedente (come, ad esempio, HDLC e DDCMP), così come nei protocolli CSMA e token ring descritti nel seguito del capitolo. L'algoritmo CRC, nella sua forma elementare, è delineato nella Sezione 2.4.3. Prima di discutere tale approccio, consideriamo due schemi più semplici, anch'essi ampiamente utilizzati: *parità bidimensionale* e *somme di controllo* (checksum). Il primo è usato dal protocollo BISYNC quando trasmette caratteri ASCII (mentre BISYNC usa CRC come codice d'errore quando trasmette caratteri EBCDIC), il secondo è usato da molti protocolli Internet.

L'idea fondamentale degli schemi di rilevazione d'errore è l'aggiunta ad un frame di informazioni ridondanti, che possano essere utilizzate per determinare se sono stati introdot-

ti errori. Al limite, potremmo immaginare di trasmettere due copie complete dei dati: se le due copie giunte a destinazione sono identiche, probabilmente sono entrambe corrette. Se sono diverse, un errore è stato introdotto in una di esse (o in entrambe), e devono essere eliminate. Questo è uno schema di rilevazione inefficiente per due motivi: prima di tutto, vengono inviati n bit ridondanti per un messaggio di n bit; secondariamente, sfuggono alla rilevazione molti errori, tutti quelli che modificano bit nelle medesime posizioni nelle due copie del messaggio.

Fortunatamente esistono schemi assai migliori. In generale, siamo in grado di garantire una forte capacità di rilevazione d'errore inviando soltanto k bit ridondanti per un messaggio di n bit, con k molto minore di n . In una linea Ethernet, ad esempio, un frame contenente fino a 12000 bit (1500 byte) di dati richiede soltanto un codice CRC di 32 bit, o, più sinteticamente, usa CRC-32. Tale codice rileverà la stragrande maggioranza degli errori, come vedremo tra poco.

I bit aggiuntivi che vengono inviati sono detti ridondanti perché non aggiungono alcuna informazione al messaggio: al contrario, sono calcolati elaborando direttamente il messaggio originario mediante algoritmi ben definiti, conosciuti esattamente sia dal mittente che dal destinatario. Il mittente applica l'algoritmo al messaggio per generare i bit ridondanti, poi trasmette il messaggio insieme a questi pochi bit aggiuntivi. Quando il destinatario applica il medesimo algoritmo al messaggio ricevuto, in assenza di errori dovrebbe ottenere lo stesso risultato ottenuto dal mittente, per cui confronta il risultato ottenuto con quello inviatogli dal mittente: se corrispondono, il destinatario può concludere che (con elevata probabilità) nel messaggio non sono stati introdotti errori di trasmissione. Se, invece, non corrispondono, il destinatario può esser certo che il messaggio o i bit ridondanti sono stati corrotti e deve intraprendere le azioni più idonee: eliminare il messaggio o correggerlo, se ciò è possibile.

Aggiungiamo una nota relativa alla terminologia utilizzata per questi bit aggiuntivi. In generale, ci si riferisce ad essi come a codici per la rilevazione d'errore, ma in alcuni casi particolari, quando l'algoritmo che li genera è basato su addizioni, si parla di *somme di controllo* (checksum). Come vedremo, la somma di controllo usata in Internet ha un nome appropriato: è una verifica d'errore che usa un algoritmo con addizioni. Sfortunatamente, però, la parola "checksum" viene spesso usata in modo impreciso, per indicare qualsiasi forma di codice per la rilevazione d'errore, incluso il codice CRC. Ciò può creare confusione, per cui vi invitiamo ad usare la parola "checksum" soltanto quando vi riferite a codici che usano veramente l'addizione, usando invece il termine "codice per la rilevazione d'errore" quando parlate della generica categoria di codici descritti in questa sezione.

2.4.1 Parità bidimensionale

La parità bidimensionale è proprio ciò che viene suggerito dal nome: è basata sulla parità "semplice" (unidimensionale), che solitamente comporta l'aggiunta di un bit extra ad un codice a 7 bit per rendere bilanciato il numero di valori 1 nel byte. Ad esempio, la parità dispari (*odd parity*) imposta a 1 l'ottavo bit se ciò è necessario per rendere dispari il numero di valori 1 nel byte, mentre la parità pari (*even parity*) imposta a 1 l'ottavo bit se ciò è necessario per rendere pari il numero di valori 1 nel byte. La parità bidimensionale effettua un calcolo simile per ciascuna posizione di bit in tutti i byte contenuti nel frame. Ciò produce un byte aggiuntivo di parità per l'intero frame, in aggiunta al bit di parità presente in ciascun byte. La Figura 2.16 mostra come funziona la parità bidimensionale pari in un frame di

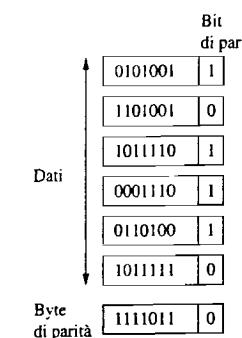


Figura 2.16 Parità bidimensionale.

esempio contenente 6 byte di dati. Notate che il terzo bit del byte di parità ha il valore 1 perché c'è un numero dispari di valori 1 nei terzi bit dei 6 byte del frame. Si può dimostrare che la parità bidimensionale rileva tutti gli errori che coinvolgono uno, due o tre bit, nonché la maggioranza degli errori di 4 bit. In questo caso abbiamo aggiunto 14 bit di informazioni ridondanti ad un messaggio di 42 bit, pur ottenendo una miglior protezione contro gli errori rispetto al "codice di ripetizione" descritto in precedenza.

2.4.2 Algoritmo di checksum di Internet

Un secondo approccio alla rilevazione d'errore ha come esempio l'algoritmo di checksum di Internet. Sebbene non sia usato al livello di linea di connessione, fornisce tuttavia lo stesso tipo di funzionalità di CRC e della parità, per cui lo presentiamo qui. Vedremo nelle Sezioni 4.1, 5.1 e 5.2 esempi del suo utilizzo.

L'idea che sta alla base della somma di controllo (checksum) di Internet è molto semplice: sommate tutte le parole che trasmettete e trasmettete, poi, il risultato di tale somma, che viene chiamato somma di controllo (checksum). Il ricevente esegue lo stesso calcolo sui dati ricevuti e confronta il risultato con il checksum ricevuto: se qualsiasi dato trasmesso, comprendendo il checksum stesso, è stato corrotto, il risultato non coinciderà, per cui il ricevente sa che si è verificato un errore.

Su questa idea di base, sono possibili molteplici varianti: lo schema esatto usato dai protocolli di Internet funziona nel modo seguente. Considerate i dati da controllare come una sequenza di interi a 16 bit e sommateli usando l'aritmetica in complemento a uno a 16 bit (spiegata nel seguito), prendendo infine il complemento a uno del risultato, che costituisce il checksum a 16 bit.

Nell'aritmetica in complemento a uno, un numero intero negativo $-x$ è rappresentato dal complemento di x , cioè ogni bit di x viene invertito. Sommando numeri con l'aritmetica in complemento a uno, eventuali riporti che provengono dal bit più significativo devono essere aggiunti al risultato. Considerate, ad esempio, l'addizione di -5 e -3 in aritmetica in complemento a uno con numeri interi di 4 bit. $+5$ è 0101, per cui -5 è 1010; $+3$ è 0011, per cui -3 è 1100. Se sommiamo 1010 e 1100 ignorando il riporto, otteniamo 0110. Nell'aritmetica in complemento a uno, il fatto che questa operazione abbiano generato un riporto dal bit più

significativo ci porta ad incrementare il risultato di un'unità, ottenendo così 0111, che è la rappresentazione in complemento a uno di -8 (ottenuta invertendo i bit di 1000), come ci aspetteremmo.

Il codice della funzione seguente fornisce un'implementazione elementare dell'algoritmo di checksum di Internet. Il parametro count fornisce la lunghezza di buf, misurata in unità di 16 bit, e la funzione ipotizza che buf sia stato precedentemente riempito con zeri fino ad una dimensione multipla di 16 bit.

```
u_short
cksum(u_short *buf, int count)
{
    register u_long sum = 0;

    while (count--)
    {
        sum += *buf++;
        if (sum & 0xFFFF0000)
        {
            /* c'è stato riporto */
            sum &= 0xFFFF;
            sum++;
        }
    }
    return ~(sum & 0xFFFF);
}
```

Questo codice garantisce che i calcoli vengano eseguiti usando soltanto l'aritmetica in complemento a uno, invece del complemento a due utilizzato nella maggior parte degli elaboratori. Notate l'enunciato `if` all'interno del ciclo `while`: se si verifica un riporto nei 16 bit più significativi di `sum`, incrementiamo `sum` di un'unità, proprio come nell'esempio precedente.

In confronto al nostro codice che duplicava le informazioni, questo algoritmo si comporta in modo migliore perché usa un numero inferiore di bit ridondanti, soltanto 16 per un messaggio di lunghezza arbitraria, ma non è molto efficiente nella rilevazione degli errori. Ad esempio, due errori su singoli bit, uno dei quali incrementa una parola e l'altro che decrementa un'altra parola della stessa quantità, non verranno rilevati. Il motivo per cui viene usato un algoritmo come questo, nonostante la sua debole protezione nei confronti degli errori (se confrontato, ad esempio, con CRC), è semplice: la realizzazione software di questo algoritmo è molto più semplice. L'esperienza di ARPANET ha evidenziato come una somma di controllo di questo tipo sia adeguata, fondamentalmente perché è soltanto l'ultima barriera difensiva in un protocollo di comunicazione tra due entità: la maggior parte degli errori vengono identificati da algoritmi di rilevazione più efficaci, come CRC, operanti al livello della linea di collegamento.

2.4.3 Verifica di ridondanza ciclica (CRC)

Dovrebbe essere ormai chiaro che uno dei principali obiettivi nella progettazione di algoritmi per la rilevazione d'errore consiste nel massimizzare la probabilità di rilevare gli errori usan-

do soltanto pochi bit ridondanti: la verifica di ridondanza ciclica usa la potenza della matematica per raggiungere tale obiettivo. Ad esempio, un codice CRC (Cyclic Redundancy Check) a 32 bit fornisce una protezione robusta contro i più comuni errori di bit che si verifichino in messaggi con lunghezza di migliaia di byte. Le basi teoriche della verifica di ridondanza ciclica trovano le proprie radici in una branca della matematica denominata campi finiti; anche se ciò può intimidire, le idee di fondo sono facilmente comprensibili.

Per iniziare, pensate ad un messaggio di $(n + 1)$ bit rappresentandolo con un polinomio di grado n , cioè un polinomio il cui termine di ordine più elevato è x^n . Il messaggio viene rappresentato con un polinomio usando il valore di ciascun bit del messaggio come coefficiente di ciascun termine del polinomio, iniziando dal bit più significativo che rappresenta il termine di ordine più elevato. Ad esempio, un messaggio di 8 bit composto dai bit 10011010 corrisponde al polinomio:

$$\begin{aligned} M(x) &= 1 \times x^7 + 0 \times x^6 + 0 \times x^5 + 1 \times x^4 + 1 \times x^3 + 0 \times x^2 + 1 \times x^1 + 0 \times x^0 \\ &= x^7 + x^4 + x^3 + x^1 \end{aligned}$$

Possiamo quindi immaginare che il mittente e il destinatario si scambino polinomi.

Allo scopo di calcolare un CRC, il mittente e il destinatario si devono accordare su un polinomio *divisore*, $C(x)$, di grado k . Supponete, ad esempio, $C(x) = x^3 + x^2 + 1$; in questo caso, $k = 3$. La risposta alla domanda "Come si sceglie $C(x)$?" è, nella maggior parte dei casi, "Lo si trova in un libro". In realtà, come vedremo in seguito, la scelta di $C(x)$ ha un impatto decisivo sul tipo di errori che vengono rilevati con una certa affidabilità. Esistono alcuni polinomi divisori che si dimostrano essere scelte valide per diverse situazioni e la scelta precisa fa normalmente parte del progetto del protocollo. Ad esempio, lo standard Ethernet usa un polinomio ben noto di grado 32.

Quando il mittente vuole trasmettere un messaggio $M(x)$ lungo $n + 1$ bit, ciò che viene realmente inviato è il messaggio di $n + 1$ bit con k bit aggiuntivi; il messaggio trasmesso, così completato e contenente anche i bit ridondanti, viene indicato come $P(x)$. Ciò che stiamo per fare, e che spiegheremo in seguito, ha l'obiettivo di rendere $P(x)$ esattamente divisibile per $C(x)$. Se $P(x)$ viene trasmesso lungo una linea e durante la trasmissione non vengono introdotti errori, il ricevente dovrebbe essere in grado di dividere $P(x)$ per $C(x)$ esattamente, con resto zero. D'altra parte, se durante la trasmissione vengono introdotti errori in $P(x)$, con tutta probabilità il polinomio ricevuto non sarà più esattamente divisibile per $C(x)$, per cui il ricevente otterrà un resto diverso da zero, cosa che implica la presenza di un errore.

Per capire ciò che segue, vi sarà di aiuto una qualche conoscenza di aritmetica dei polinomi, che è un po' diversa dalla comune aritmetica per numeri interi. Stiamo qui considerando un tipo specifico di aritmetica per polinomi, dove i coefficienti possono assumere soltanto i valori zero o uno e le operazioni sui coefficienti vengono eseguite con l'aritmetica modulo 2: parliamo, quindi, di "aritmetica per polinomi modulo 2". Essendo questo un testo di reti di calcolatori e non di matematica, concentriamo la nostra attenzione sulle proprietà di questo tipo di aritmetica che servono ai nostri scopi, proprietà che vi chiediamo di accettare senza dimostrazione:

- Qualsiasi polinomio $B(x)$ è divisibile dal polinomio divisore $C(x)$ se $B(x)$ è di grado più elevato di $C(x)$.
- Qualsiasi polinomio $B(x)$ è divisibile una volta dal polinomio divisore $C(x)$ se $B(x)$ ha lo stesso grado di $C(x)$.

Semplici calcoli di probabilità

Quando si ha a che fare con errori nelle reti e altri eventi (speriamo) improbabili, usiamo spesso delle stime di probabilità molto semplificate. Un'approssimazione che qui ci è utile è quella relativa a due eventi indipendenti con *piccole* probabilità p e q , nel qual caso la probabilità che si verifichi uno qualsiasi dei due eventi è $p + q$, mentre il valore esatto sarebbe $1 - (1 - p)(1 - q) = p + q - pq$. Con $p = q = 0.01$, la nostra stima vale 0.02, mentre il valore esatto è 0.0199.

Come semplice applicazione di ciò, supponete che il tasso di errore per bit di una linea sia $1/10^7$. Ipotizzando che gli errori di bit siano tutti indipendenti (cosa che non corrisponde al vero), possiamo stimare che la probabilità che si verifichi almeno un errore in un pacchetto di 10000 bit sia $10^4/10^7 = 10^{-3}$. Il valore esatto, calcolato come $1 - P(\text{nessun errore})$, sarebbe $1 - (1 - 10^{-7})^{10000} = 0.0009995$.

Per fare un esempio appena un po' più complesso, calcoliamo la probabilità che vi siano due errori in uno di tali pacchetti, che è la probabilità di un errore che passerebbe inosservato ad un controllo di parità eseguito con un bit. Indicando con E_{ij} l'evento in cui i bit i e j sono errati, con $0 \leq i < j < 10^4$, la probabilità di tale evento è circa $p = 10^{-7} \times 10^{-7} = 10^{-14}$. Per un certo valore di j prefissato, il numero di eventi E_{ij} aventi $i < j$ è uguale a j : sommando il numero di tali eventi per tutti i valori di $j < 10^4$, otteniamo $1 + 2 + \dots + (10^4 - 1) = \frac{1}{2} 10^8$. La probabilità complessiva è, quindi, $\frac{1}{2} 10^8 \times 10^{-14} = \frac{1}{2} 10^{-6}$.

Notate che se avessimo tentato di stimare $P(\text{due errori}) = P(\text{primo errore}) \times P(\text{secondo errore})$ e se avessimo considerato queste ultime due uguali a $P(\text{un errore}) = 10^{-7}$, avremmo ottenuto in questo caso 10^{-14} , che è abbastanza lontano dal valore esatto: il problema, con questo approccio, è che non tutti i valori di i hanno la stessa probabilità di rappresentare la posizione del primo errore, oppure, detto in altre parole, abbiamo sovrastimato la reale probabilità di un fattore due perché abbiamo contato separatamente gli errori nelle posizioni (i, j) e (j, i) , mentre avrebbero dovuto essere contati una volta sola.

- Il resto ottenuto dividendo $B(x)$ per $C(x)$ si ottiene sottraendo $C(x)$ da $B(x)$.
- Per sottrarre $C(x)$ da $B(x)$, eseguiamo semplicemente l'operazione di OR esclusivo (XOR) su ciascuna coppia di coefficienti corrispondenti.

Ad esempio, il polinomio $x^3 + 1$ può essere diviso per $x^3 + x^2 + 1$ (perché sono entrambi di grado 3) ed il resto della divisione sarebbe $0 \times x^3 + 1 \times x^2 + 0 \times x^1 + 0 \times x^0 = x^2$ (ottenuto effettuando l'OR esclusivo dei coefficienti di ciascun termine). In termini di messaggi, potremmo dire che 1001 può essere diviso per 1101, con resto 0100. Dovreste essere in grado di notare che il resto è proprio l'OR esclusivo bit a bit dei due messaggi.

Ora che conosciamo le regole basilari per dividere i polinomi, siamo in grado di eseguire le lunghe divisioni che sono necessarie per gestire lunghi messaggi, come si può vedere nell'esempio seguente.

Ricordate che avevamo l'obiettivo di creare un polinomio da trasmettere che sia derivato dal messaggio originale $M(x)$, che sia k bit più lungo di $M(x)$ e che sia esattamente divisibile per $C(x)$. Possiamo farlo nel modo seguente:

1. Moltiplichiamo $M(x)$ per x^k , cioè aggiungiamo k zeri alla fine del messaggio. Chiamiamo $T(x)$ questo messaggio espanso con zeri.

2. Dividiamo $T(x)$ per $C(x)$ e calcoliamo il resto.
3. Sottraiamo il resto da $T(x)$.

Dovrebbe essere ovvio che a questo punto abbiamo un messaggio esattamente divisibile per $C(x)$, possiamo anche notare che il messaggio risultante è composto da $M(x)$ seguito dal resto ottenuto al passo 2, perché quando abbiamo sottratto tale resto (che non può essere più lungo di k bit), in realtà abbiamo eseguito l'OR esclusivo con i k zeri aggiunti al passo 1, cosa che diventerà più chiara con un esempio.

Considerate il messaggio $x^7 + x^4 + x^3 + x^1$, cioè 10011010. Iniziamo moltiplicandolo per x^3 , perché il nostro polinomio divisore è di grado 3. Questa operazione ha come risultato 10011010000, che dividiamo per $C(x)$, corrispondente in questo caso a 1101. La Figura 2.17 illustra l'operazione di divisione in colonna dei polinomi, che, in forza delle regole dell'aritmetica dei polinomi che abbiamo prima descritto, viene eseguita in modo molto simile alla divisione fra numeri interi: nel primo passo dell'esempio vediamo che il divisore 1101 è contenuto una volta nei primi quattro bit del messaggio (1001), poiché tali valori rappresentano polinomi con lo stesso grado, e fornisce come resto 100 (1101 XOR 1001). Il passo successivo ci porta ad abbassare una cifra dal polinomio del messaggio, finché non otteniamo un altro polinomio avente lo stesso grado di $C(x)$, che in questo caso è 1001. Calcoliamo nuovamente il resto (100) e continuamo fino a completare il calcolo. Notate che il "risultato" della divisione in colonna, indicato in cima ai calcoli eseguiti, in realtà non ci interessa: siamo interessati solamente al resto finale.

Come potete vedere nella parte bassa della Figura 2.17, il resto del calcolo che abbiamo usato come esempio è 101, per cui sappiamo che 10011010000 meno 101 sarebbe esattamente divisibile per $C(x)$ e questo è ciò che inviamo. Nell'aritmetica dei polinomi, l'operazione di sottrazione è l'operazione di XOR logico, per cui ciò che inviamo realmente è 10011010101. Come abbiamo notato in precedenza, ciò risulta essere proprio il messaggio originale con il resto della divisione accodato ad esso. Il ricevente divide il polinomio ricevuto per $C(x)$ e se il risultato è 0 conclude che non ci sono stati errori. Se il risultato è diverso da zero, potrebbe essere necessario eliminare il messaggio errato, ma con alcuni codici è possibile *correggere* un piccolo errore (ad esempio, se l'errore colpisce un solo bit). Un codice che consente la correzione di errori viene chiamato *codice a correzione d'errore* (ECC, error-correcting code).

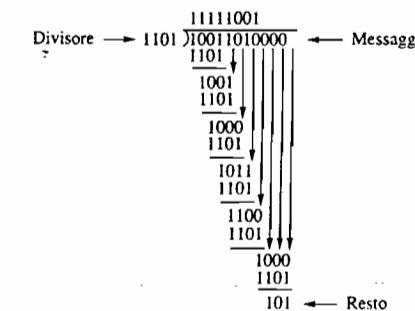


Figura 2.17 Il calcolo di CRC usando la divisione in colonna tra polinomi.

Ora ci dedicheremo al problema di come trovare il polinomio $C(x)$. Dal punto di vista intuitivo, l'idea è quella di selezionare tale polinomio in modo che sia molto improbabile che sia un divisore con resto zero di un messaggio con errori. Se il messaggio trasmesso è $P(x)$, possiamo pensare all'introduzione di errori come all'aggiunta di un altro polinomio, $E(x)$, in modo che il ricevente veda il polinomio $P(x) + E(x)$. Un errore può sfuggire alla rilevazione soltanto se tale messaggio ricevuto è divisibile esattamente per $C(x)$ e, poiché sappiamo che $P(x)$ può essere diviso esattamente per $C(x)$, ciò potrebbe accadere soltanto se $E(x)$ può essere diviso esattamente per $C(x)$. Il trucco consiste nel prendere $C(x)$ in modo tale che ciò accada molto raramente per i tipi di errori più comuni.

Un comune tipo di errore è l'errore su un singolo bit, che si può esprimere come $E(x) = x^i$ nel caso in cui colpisca il bit in posizione i . Se scegliamo $C(x)$ in modo che il primo e l'ultimo termine siano diversi da zero, abbiamo già un polinomio di due termini che non può essere un divisore esatto dell'unico termine di $E(x)$: quindi, tale $C(x)$ può rilevare tutti gli errori che colpiscono un singolo bit. In generale, è possibile dimostrare che i seguenti tipi di errori sono rilevabili da un polinomio $C(x)$ che abbia le caratteristiche enunciate:

- Tutti gli errori che colpiscono un singolo bit, se $C(x)$ ha i termini x^k e x^0 con coefficienti diversi da zero.
- Tutti gli errori che colpiscono due bit, se $C(x)$ ha un fattore con almeno tre termini.
- Qualsiasi numero dispari di errori, se $C(x)$ contiene il fattore $(x+1)$.
- Qualsiasi errore a "burst" (cioè una sequenza di bit errati consecutivi), se la lunghezza del burst è minore di k bit (vengono anche rilevati la maggior parte degli errori a burst con lunghezza superiore a k bit).

Nei protocolli al livello di linea di collegamento vengono ampiamente utilizzati sei versioni di $C(x)$, indicate nella Tabella 2.5. Ad esempio, le reti Ethernet e 802.5, descritte nel seguito del capitolo, usano CRC-32, mentre HDLC usa CRC-CCITT. Le reti ATM, descritte nel Capitolo 3, usano CRC-8, CRC-10 e CRC-32.

Infine, notiamo che l'algoritmo CRC, nonostante sembri complesso, si può realizzare facilmente in hardware, usando un registro a scorrimento a k bit e porte logiche XOR. Il numero di bit del registro a scorrimento è uguale al grado del polinomio generatore (k) e la Figura 2.18 mostra la configurazione hardware che si dovrebbe usare per il generatore $x^3 + x^2 + 1$ del nostro esempio precedente. Il messaggio viene fatto scorrere introducendolo da sinistra, iniziando con il bit più significativo e terminando con la stringa di k zeri accodata al messaggio, proprio come nell'esempio della divisione in colonna. Quando tutti i bit sono stati inseriti, facendoli scorrere, e le operazioni di XOR sono state eseguite in modo corretto, il registro contiene il resto, cioè il CRC (con il bit più significativo a destra). La

Tabella 2.5 Polinomi CRC ampiamente diffusi.

CRC-8	$x^8 + x^2 + x^1 + 1$
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^1 + 1$
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + 1$
CRC-16	$x^{16} + x^{15} + x^2 + 1$
CRC-CCITT	$x^{16} + x^{12} + x^5 + 1$
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

2.5 Trasmissione affidabile

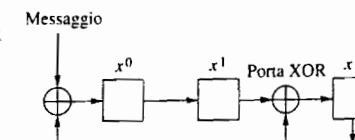


Figura 2.18 Calcolo di CRC usando un registro a scorrimento.

posizione delle porte logiche XOR si determina nel modo seguente: se i bit del registro a scorrimento vengono etichettati con i numeri interi da 0 a $k-1$, da sinistra a destra, allora se nel polinomio generatore è presente il termine x^n si inserisce una porta logica XOR davanti al bit n . Infatti, per il polinomio $x^3 + x^2 + x^0$, vediamo che è presente una porta logica XOR nelle posizioni 0 e 2.

2.5 Trasmissione affidabile

Come abbiamo visto nella sezione precedente, a volte i frame vengono corrotti durante il loro trasferimento e si usa un codice come CRC per rilevare tali errori. Mentre alcuni codici sono sufficientemente robusti da consentire anche la correzione degli errori, in pratica il loro overhead (in termini di bit aggiuntivi) è tipicamente troppo elevato per gestire gli intervalli di errori di

Rilevazione o correzione d'errore?

Abbiamo citato la possibilità di utilizzare codici che non soltanto rilevano la presenza di errori, ma consentono anche la loro correzione: non li approfondiremo perché una loro analisi in dettaglio richiede conoscenze matematiche ancora più complesse di quelle necessarie per la comprensione dei codici CRC, tuttavia è importante considerare i vantaggi della correzione rispetto alla rilevazione.

A prima vista potrebbe sembrare che la correzione sia sempre migliore, perché con la sola rilevazione siamo costretti a scaricare il messaggio e generalmente a richiedere la trasmissione di una sua copia, consumando ampia banda e introducendo latenza nell'attesa della ritrasmissione. Tuttavia, anche la correzione ha uno svantaggio: in generale, l'invio di un codice a correzione d'errore che abbia la stessa robustezza (cioè gestisca lo stesso insieme di errori) di un codice a sola rilevazione richiede un numero di bit ridondanti più elevato. Quindi, mentre la rilevazione d'errore richiede l'invio di più bit quando si verificano errori, la correzione d'errore richiede l'invio di più bit *sempre*. Di conseguenza, la correzione d'errore tende ad essere più utile quando gli errori sono assai probabili (come può accadere, ad esempio, in un ambiente wireless) oppure quando i costi di ritrasmissione sono troppo elevati (a causa, ad esempio, della latenza tipica della ritrasmissione di un pacchetto in un collegamento satellitare).

A volte, nelle reti che usano codici a correzione d'errore si parla di *correzione d'errore anticipata* (FEC, forward error correction), perché la correzione degli errori viene gestita a priori inviando informazioni supplementari, piuttosto che aspettare che l'errore accada per gestirlo successivamente con la ritrasmissione.

bit e a burst che si possono verificare su una linea di collegamento in una rete. Anche quando si usano codici a correzione d'errore (come, ad esempio, in collegamenti wireless), accadranno errori troppo gravi per poter essere corretti, per cui alcuni frame corrotti verranno comunque scartati. Un protocollo del livello di linea di connessione che voglia consegnare frame in maniera affidabile deve, in qualche modo, gestire queste situazioni in cui vi sono frame scartati (o perduti).

Ciò solitamente si realizza usando una combinazione di due meccanismi basilari: *acknowledgement* (conferma) e *timeout* ("tempo scaduto"). Un acknowledgement (ACK in breve) è un piccolo frame di controllo restituito da un protocollo alla sua entità di pari livello per segnalare la ricezione di un frame precedente, dove per "frame di controllo" intendiamo un'intestazione senza dati, anche se un protocollo può mettere un ACK "sulle spalle" di un frame di dati che sta inviando nella direzione opposta (tecnica denominata *piggyback*). Il ricevimento di una conferma segnala al mittente del frame originale che tale suo frame è stato consegnato correttamente. Se, invece, il mittente non riceve una conferma entro un intervallo di tempo ragionevole, il frame originale viene *ritrasmesso*: l'attesa di un intervallo di tempo ragionevole viene detta *timeout*.

La strategia che usa acknowledgement e timeout per realizzare la consegna affidabile viene genericamente chiamata *richiesta di ripetizione automatica* (ARQ, automatic repeat request) e questa sezione descrive tre diversi algoritmi di ARQ usando un linguaggio generico, cioè senza fornire informazioni dettagliate sui campi dell'intestazione di alcun particolare protocollo.

2.5.1 Stop-and-wait

Lo schema di ARQ più semplice è l'algoritmo *stop-and-wait* (fermati e aspetta), la cui idea è elementare: dopo aver trasmesso un frame, il mittente aspetta un acknowledgement prima di trasmettere il frame successivo. Se l'acknowledgement non arriva entro un certo periodo di tempo, il mittente dichiara "tempo scaduto" (timeout) e ritrasmette il frame originale.

La Figura 2.19 mostra quattro diversi scenari in cui si può trovare questo algoritmo elementare. La figura rappresenta una sequenza temporale di eventi (*timeline*), un modo molto comune per evidenziare il comportamento di un protocollo. La parte che invia è rappresentata sulla sinistra, mentre la parte che riceve è sulla destra; il tempo scorre dall'alto verso il basso. La Figura 2.19(a) mostra la situazione in cui il frame ACK viene ricevuto prima che scada il tempo, (b) e (c) illustrano le situazioni in cui, rispettivamente, vengono perduti il frame originale e il frame ACK, e (d) rappresenta la situazione in cui il tempo scade troppo presto. Ricordate che "perduto" in questo contesto significa che il frame è stato corrotto durante il trasferimento, che tale errore è stato rilevato dal ricevitore usando il codice a rilevazione d'errore e che, di conseguenza, il frame è stato scartato.

Occorre fare attenzione ad un piccolo dettaglio dell'algoritmo stop-and-wait. Supponete che il mittente invii un frame e che il ricevitore ne dia conferma, ma che tale conferma vada perduta o arrivi in ritardo, come illustrato nei diagrammi temporali (c) e (d) di Figura 2.19. In entrambi i casi il mittente dichiara "tempo scaduto" e ritrasmette il frame originale, ma il ricevente penserà che si tratti del frame successivo, perché aveva correttamente ricevuto e confermato il primo frame: ciò può provocare la consegna di due copie di uno stesso frame. Per risolvere questo problema, solitamente in un protocollo stop-and-wait l'intestazione contiene un numero di sequenza di un solo bit (cioè il numero di sequenza può assumere soltanto

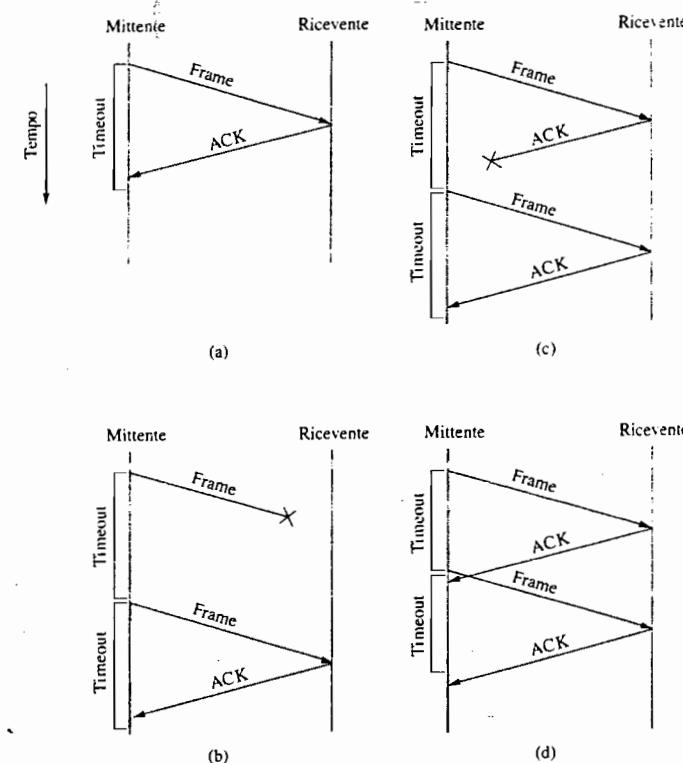


Figura 2.19 Diagrammi temporali per quattro diversi scenari relativi all'algoritmo stop-and-wait: (a) il frame ACK viene ricevuto prima che scada il tempo; (b) viene perduto il frame originale; (c) viene perduto il frame ACK; (d) il tempo scade troppo presto.

i valori 0 e 1) ed i numeri vengono usati alternativamente per ciascun frame, come indicato in Figura 2.20. In questo modo, quando il mittente ritrasmette il frame 0, il ricevitore è in grado di determinare se sta osservando una seconda copia del frame 0 piuttosto che la prima copia del frame 1 e nel primo caso ignora il frame (pur confermandolo di nuovo, dato che il primo ACK può essere andato perduto).

Il maggior difetto dell'algoritmo stop-and-wait consiste nel fatto che consente al mittente di avere un solo frame per volta sulla linea, in attesa di conferma, e ciò può essere molto meno di quanto consentito dalla capacità del collegamento. Considerate, ad esempio, una linea di collegamento a 1.5 Mbps con tempo di round-trip di 45 ms. Tale linea ha un prodotto ritardo × ampiezza di banda uguale a 67.5 Kb, cioè circa 8 KB. Poiché il mittente può inviare un solo frame per ciascun intervallo di tempo di durata RTT, ipotizzando una dimensione di frame di 1 KB ciò implica una velocità massima di invio di

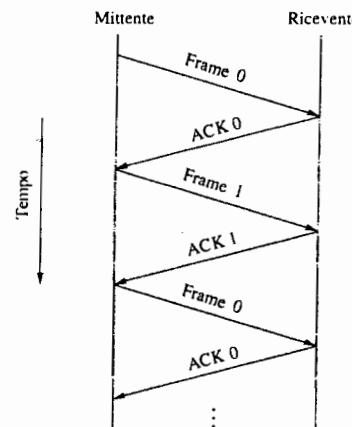


Figura 2.20 Diagramma temporale per l'algoritmo stop-and-wait con numeri di sequenza di un solo bit.

$$\begin{aligned} \text{BitPerFrame/TempoPerFrame} \\ = 1024 \times 8 / 0.045 \\ = 182 \text{ Kbps} \end{aligned}$$

cioè circa un ottavo della capacità della linea. Per usare pienamente la linea, dovremmo consentire al mittente di trasmettere fino a otto frame prima di metterlo in attesa di un acknowledgement.

Il prodotto ritardo × ampiezza di banda rappresenta la quantità di dati che possono trovarsi in transito, per cui vorremmo poter inviare tale quantità di dati senza dover aspettare il primo acknowledgement, secondo il principio di *mantenere piena la conduttrice* (*keeping the pipe full*). Gli algoritmi presentati nelle due sottosezioni seguenti fanno esattamente questo.

2.5.2 Sliding window

Prendete nuovamente in considerazione lo scenario in cui la linea di collegamento ha un prodotto ritardo × ampiezza di banda di 8 KB e i frame hanno una dimensione di 1 KB. Vorremmo che il mittente fosse pronto per trasmettere il nono frame proprio nello stesso momento in cui arriva la conferma del primo frame: l'algoritmo illustrato in Figura 2.21 è chiamato *sliding window* (finestra scorrevole) consente di fare questo.

L'algoritmo sliding window

L'algoritmo sliding window funziona nel modo seguente. Per prima cosa il mittente assegna a ciascun frame un *numero di sequenza*, indicato con SeqNum: per il momento, ignoriamo il fatto che SeqNum sia realizzato con un campo d'intestazione di dimensioni finite e ipotizziamo, invece, che possa diventare arbitrariamente grande. Il mittente gestisce tre variabili: la *di-*

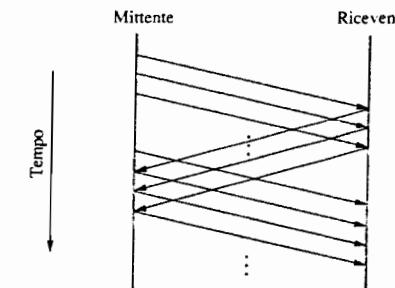


Figura 2.21 Diagramma temporale per l'algoritmo sliding window.

mensione della finestra di invio, SWS (send window size), indica il limite superiore per il numero di frame che il mittente può trasmettere sulla linea senza che questi siano confermati; LAR (last acknowledgement received) indica il numero di sequenza dell'*ultima conferma ricevuta*; LFS (last frame sent) indica il numero di sequenza dell'*ultimo frame inviato*. Il mittente soddisfa anche il seguente invarianto:

$$\text{LFS} - \text{LAR} \leq \text{SWS}$$

La situazione è rappresentata nella Figura 2.22.

Quando arriva una conferma, il mittente sposta LAR verso destra, consentendo così l'invio di un altro frame. Inoltre il mittente associa un "conto alla rovescia" a ciascun frame che trasmette, con conseguente ritrasmissione del frame se il tempo scade prima di aver ricevuto il relativo ACK. Notate che il mittente deve essere in grado di memorizzare fino a un numero massimo di frame uguale a SWS, perché deve essere pronto a ritrasmetterli finché non sono stati confermati.

Il ricevitore gestisce le seguenti tre variabili. La *dimensione della finestra di ricezione*, RWS (receive window size), indica il limite superiore per il numero di frame fuori sequenza che il ricevitore può accettare; LAF (largest acceptable frame) indica il numero di sequenza del *frame accettabile più elevato*; LFR (last frame received) indica il numero di sequenza dell'*ultimo frame ricevuto*. Il ricevitore soddisfa anche il seguente invarianto:

$$\text{LAF} - \text{LFR} \leq \text{RWS}$$

La situazione è rappresentata nella Figura 2.23.

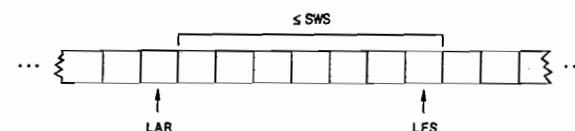


Figura 2.22 Sliding window per il mittente.

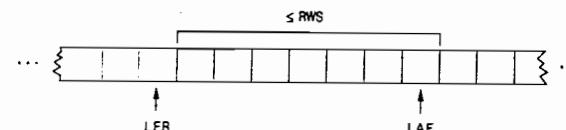


Figura 2.23 Sliding window per il ricevente.

Quando arriva un frame con numero di sequenza SeqNum, il ricevitore agisce nel modo seguente. Se $SeqNum \leq LFR$ oppure $SeqNum > LAF$, allora il frame si trova al di fuori della finestra del ricevitore e viene scartato. Se, invece, $LFR < SeqNum \leq LAF$, il frame si trova all'interno della finestra del ricevitore e viene accettato. Rimane solo da decidere se inviare un ACK oppure no. Utilizziamo SeqNumToAck per indicare il più elevato numero di sequenza non ancora confermato, in modo che tutti i frame con numeri di sequenza inferiori o uguali a SeqNumToAck siano già stati ricevuti. Il ricevitore conferma la ricezione di SeqNumToAck, anche se sono stati ricevuti pacchetti con numero più elevato: tale conferma viene detta cumulativa. Quindi, il ricevitore pone $LFR = SeqNumToAck$ e $LAF = LFR + RWS$.

Supponiamo, ad esempio, che sia $LFR = 5$ (cioè l'ultimo ACK inviato dal ricevitore era relativo al numero di sequenza 5) e $RWS = 4$: ciò implica che sia $LAF = 9$. Se dovessero arrivare i frame 7 e 8, che si trovano all'interno della finestra del ricevitore, verrebbero memorizzati, ma non verrebbe inviato alcun ACK perché il frame 6 non è ancora arrivato: si dice che i frame 7 e 8 sono arrivati fuori sequenza (teoricamente, quando riceve i frame 7 e 8 il ricevitore potrebbe inviare nuovamente un ACK per il frame 5). Se dovesse poi arrivare il frame 6, in ritardo perché ritrasmesso dopo essere andato perduto la prima volta oppure semplicemente in ritardo, il ricevitore confermerà il frame 8, spostando LFR a 8 e LAF a 12. Se il frame 6 fosse andato effettivamente perduto, il mittente avrebbe visto scadere il relativo temporizzatore e lo avrebbe ritrasmesso.

Osserviamo che quando avviene un timeout la quantità di dati in transito sulla linea diminuisce, perché il mittente non è in grado di far avanzare la propria finestra finché il frame 6 non viene confermato. Ciò significa che quando si verificano perdite di pacchetti questo schema non è più in grado di mantenere la condutture piena. Più tempo occorre per accorgersi che un pacchetto è andato perduto, più grave diventa questo problema.

Note anche che in questo esempio il ricevitore avrebbe potuto inviare una *conferma negativa* (NAK, negative acknowledgement) per il frame 6 non appena fosse giunto il frame 7: tuttavia ciò non è necessario, perché il meccanismo di timeout del mittente è sufficiente per gestire questa situazione e l'invio di conferme negative aggiunge complessità al ricevitore. Ancora, come già accennato, sarebbe stato possibile inviare conferme aggiuntive per il frame 5 quando sono stati ricevuti i frame 7 e 8: in alcuni casi il mittente può utilizzare le conferme duplicate come un segnale che un frame è andato perduto. Entrambi questi approcci aiutano a migliorare le prestazioni, consentendo una più tempestiva rilevazione delle perdite di pacchetti.

Un'altra variante ancora di questo schema richiederebbe l'utilizzo di *conferme selettive*, cioè il ricevitore potrebbe confermare i singoli pacchetti ricevuti, invece di segnalare il numero di sequenza più elevato dei frame ricevuti in ordine. In tal caso, nell'esempio precedente, il ricevitore potrebbe confermare la ricezione dei frame 7 e 8. Fornire maggiori informazioni al mittente rende potenzialmente più facile il suo compito di mantenere piena la condutture. Ma ne rende più complessa l'implementazione.

La dimensione della finestra del mittente è scelta in base al numero di frame che vogliono avere in transito sulla linea in un determinato istante: è facile calcolare SWS una volta assegnato il prodotto ritardo \times ampiezza di banda¹. Al contrario, il ricevitore può usare il valore di RWS che desidera. Due scelte frequenti sono $RWS = 1$, per cui il ricevitore non memorizza alcun frame che arrivi fuori sequenza, e $RWS = SWS$, in modo che il ricevitore memorizza tanti frame quanti ne può trasmettere il mittente. Usare un valore $RWS > SWS$ non ha senso, perché è impossibile che arrivino fuori sequenza frame in numero maggiore di SWS .

Sliding window con numeri di sequenza finiti

Torniamo ora alla semplificazione che abbiamo introdotto nell'algoritmo, la nostra ipotesi che i numeri di sequenza possano diventare indefinitamente grandi. Dal punto di vista pratico, ovviamente, il numero di sequenza di un frame viene specificato all'interno di un campo dell'intestazione, avente una qualche dimensione finita. Ad esempio, un campo di 3 bit mette a disposizione otto possibili numeri di sequenza, da 0 a 7. Ciò rende, quindi, necessario che si riutilizzino i numeri di sequenza oppure, detto in altro modo, che i numeri di sequenza tornino al loro valore iniziale, introducendo il problema di poter distinguere fra diverse incarnazioni degli stessi numeri di sequenza: di conseguenza, i numeri di sequenza possibili devono essere più del numero di frame non confermati a cui è consentito di essere in transito sulla linea. Ad esempio, l'algoritmo stop-and-wait consente un solo frame non confermato e ha, quindi, due numeri di sequenza.

Supponiamo che i numeri disponibili come numero di sequenza siano uno in più del numero di frame potenzialmente in transito, cioè che $SWS \leq \text{MaxSeqNum} - 1$, dove MaxSeqNum è il numero di numeri di sequenza disponibili. È sufficiente? La risposta dipende da RWS . Se $RWS = 1$, allora $\text{MaxSeqNum} \geq SWS + 1$ è sufficiente. Se RWS è uguale a SWS , allora avere MaxSeqNum maggiore della dimensione della finestra di invio soltanto per un'unità non è sufficiente. Per capirlo, considerate la situazione in cui abbiamo otto numeri di sequenza, da 0 a 7, e $SWS = RWS = 7$. Supponete, poi, che il mittente trasmetta i frame numerati da 0 a 6, che vengono ricevuti con successo, ma di cui vengono perse le conferme. Il ricevitore si aspetta quindi che arrivi il frame 7, seguito dai frame numerati nuovamente da 0 a 5, ma il mittente vede scadere le temporizzazioni ed invia nuovamente i frame numerati da 0 a 6. Sfortunatamente, il ricevitore sta aspettando la seconda incarnazione dei frame da 0 a 5, mentre riceve una copia dei precedenti: esattamente la situazione che volevamo evitare.

Si comprende come, nel caso in cui $RWS = SWS$, la dimensione della finestra di invio non possa essere maggiore della metà del numero di numeri di sequenza disponibili, o, più precisamente

$$SWS < (\text{MaxSeqNum} + 1)/2$$

Intuitivamente, stiamo dicendo che il protocollo sliding window considera alternativamente le due metà dello spazio dei numeri di sequenza, proprio come il protocollo stop-and-wait usa alternativamente i due valori 0 e 1 come numeri di sequenza. La sola differenza è che il

¹ È facile se conosciamo il ritardo e l'ampiezza di banda, ma a volte non li conosciamo e stimarli bene è uno dei difficili compiti dei progettisti di protocolli. Ne parleremo nuovamente nel Capitolo 5.

protocollo sliding window trasla in modo continuo tra le due metà, invece di saltare da una all'altra.

Note come questa regola sia specifica della situazione in cui $RWS = SWS$. Lasciamo come esercizio la determinazione della regola più generale, valida per valori qualsiasi di RWS e di SWS . Note anche che la relazione tra la dimensione della finestra e lo spazio dei numeri di sequenza dipende da un'ipotesi talmente ovvia da essere facilmente trascurata, cioè che i frame non vengano scambiati tra loro durante il transito lungo la linea. Ciò non può avvenire in una linea diretta di collegamento punto-punto, perché un frame non ha modo di superarne un altro durante la trasmissione, ma vedremo l'algoritmo sliding window usato in un ambiente diverso, nel Capitolo 5, e dovremo derivare una regola diversa.

Implementazione di sliding window

Le funzioni seguenti illustrano come si possano implementare le parti di invio e di ricezione dell'algoritmo sliding window. Le funzioni sono derivate da un protocollo reale e funzionante, denominato, abbastanza a tono, protocollo sliding window (SWP, sliding window protocol). Per non preoccuparci dei protocolli adiacenti nel grafo dei protocolli, indicheremo come HLP (high-level protocol) il protocollo dello strato superiore a SWP e come LINK (link-level protocol) il protocollo dello strato inferiore ad esso.

Iniziamo definendo un paio di strutture dati. Prima di tutto, l'intestazione del frame, che è molto semplice: contiene un numero di sequenza (SeqNum) ed un numero di conferma (AckNum), oltre ad un campo Flags che indica se il frame è un ACK o trasporta dei dati.

```
typedef u_char SwpSeqno;

typedef struct {
    SwpSeqno SeqNum; /* numero di sequenza di questo frame */
    SwpSeqno AckNum; /* conferma di ricezione di un frame */
    u_char Flags; /* fino a 8 bit di indicazioni */
} SwpHdr;
```

Successivamente, definiamo lo stato dell'algoritmo sliding window con la seguente struttura. Dal lato del mittente, lo stato comprende le variabili LAR e LFS, come visto in precedenza, oltre ad una coda che contiene i frame che sono stati trasmessi ma non ancora confermati (sendQ). Lo stato del mittente contiene anche un *semaforo a contatore*, chiamato sendWindowNotFull: vedremo in seguito come viene usato, ma un semaforo, in generale, è una primitiva di sincronizzazione con le operazioni semWait e semSignal. Ogni invocazione di semSignal incrementa il conteggio del semaforo di uno e ogni invocazione di semWait lo decremente di uno, con il processo invocante bloccato (tecnicamente, sospeso) nel caso in cui il decremento del semaforo portasse il contatore ad un valore negativo. Un processo bloccato durante l'invocazione di semWait potrà riprendere la propria esecuzione non appena sia stato eseguito un numero di operazioni semSignal sufficiente a riportare il conteggio ad un valore non negativo.

Dal lato ricevente di questo protocollo, lo stato comprende la variabile NFE (next frame expected), che rappresenta il *successivo frame atteso*, il frame avente un numero di sequenza superiore di un'unità rispetto all'ultimo frame ricevuto (LFR), descritto in precedenza. Anche qui c'è una coda che contiene i frame ricevuti fuori sequenza (recvQ). Infine, anche se non

viene mostrato, le dimensioni delle finestre scorrevoli del mittente e del ricevente sono definite, rispettivamente, dalle costanti SWS e RWS.

```
typedef struct {
    /* stato del mittente */
    SwpSeqno LAR; /* numero dell'ultimo ACK ricevuto */
    SwpSeqno LFS; /* ultimo frame inviato */
    Semaphore sendWindowNotFull;
    SwpHdr hdr; /* intestazione pre-inizializzata */
    struct sendQ_slot {
        Event timeout; /* associato al timeout di invio */
        Msg msg;
    } sendQ[SWS];
} SwpState;

/* stato del ricevente */
SwpSeqno NFE; /* numero del prossimo frame atteso */
struct recvQ_slot {
    int received; /* il messaggio è valido? */
    Msg msg;
} recvQ[RWS];
} SwpState;
```

Il lato mittente di SWP è implementato dalla procedura sendSWP, una funzione piuttosto semplice. Dapprima semWait provoca il blocco del processo ad un semaforo finché non è possibile inviare un altro frame. Una volta avuta l'autorizzazione a procedere, sendSWP imposta il numero di sequenza nell'intestazione del frame, memorizza una copia del frame nella coda di trasmissione (sendQ), fa partire un evento di temporizzazione per gestire il caso in cui il frame non venga confermato e invia il frame al protocollo di livello inferiore, che indicheremo come LINK.

Un dettaglio che ci interessa evidenziare è l'invocazione di store_swp_hdr subito prima dell'invocazione di msgAddHdr. Questa funzione traduce la struttura C che contiene l'intestazione di SWP (state->hdr) in una stringa di byte da anteporre al messaggio (hbuf); la funzione (che non viene mostrata) deve tradurre ciascun campo per numeri interi presente nell'intestazione in un intero rappresentato con i byte nell'ordine definito dalla rete e deve rimuovere informazioni accessorie introdotte nella struttura dati dal compilatore C. Il problema dell'ordine dei byte è discusso in maggiore dettaglio nella Sezione 7.1: per ora, è sufficiente immaginare che questa funzione disponga il bit più significativo di un numero intero a più byte nel byte avente l'indirizzo più alto. Ancora, per contenere un messaggio usiamo un tipo di dati astratto, indicato con Msg, che sia dotato di operazioni quali msgAddHdr e msgSaveCopy.

Un altro aspetto complesso di questa funzione è l'uso di semWait e del semaforo sendWindowNotFull, che viene inizializzato con la dimensione della finestra scorrevole del mittente, SWS (questa inizializzazione non viene mostrata). Ogni volta che il mittente trasmette un frame, l'operazione semWait decremente questo contatore e blocca il mittente quando il conteggio arriva a zero. Ogni volta che viene ricevuto un ACK, l'operazione semSignal invocata in deliverSWP (si veda in seguito) incrementa il contatore, sbloccando così il mittente in attesa.

```

static int
sendSWP(SwpState *state, Msg *frame)
{
    struct sendQ_slot *slot;
    hbuf[HLEN];

    /* aspetta che la finestra di invio si apra */
    semWait(&state->sendWindowNotFull);
    state->hdr.SeqNum = ++state->LFS;
    slot = &state->sendQ[state->hdr.SeqNum % SWS];
    store_swp_hdr(state->hdr, hbuf);
    msgAddHdr(frame, hbuf, HLEN);
    msgSaveCopy(&slot->msg, frame);
    slot->timeout = evSchedule(swpTimeout, slot,
        SWP_SEND_TIMEOUT);
    return sendLINK(frame);
}

```

Veniamo ora all'implementazione dell'operazione *deliver* specifica per il protocollo SWP, fornita nella procedura *deliverSWP*, che in pratica gestisce due diversi tipi di messaggi in arrivo: ACK per i frame inviati precedentemente da questo nodo e i frame di dati che arrivano in questo nodo. In un certo senso, la parte che gestisce gli ACK è la controparte del metodo *sendSWP*. La decisione relativa al tipo di messaggio (frame ACK o frame di dati) è presa in base alla verifica del campo *Flags* dell'intestazione. Notate che questa particolare implementazione non fornisce supporto al piggyback degli ACK sui frame di dati.

Quando il frame in arrivo è un ACK, *deliverSWP* cerca semplicemente la posizione, nella coda di trasmissione (*sendQ*), che corrisponde all'ACK, eliminando l'evento del temporizzatore e liberando lo spazio occupato dal frame nella coda. Tutto questo viene eseguito in un ciclo, perché la conferma contenuta in un ACK può essere cumulativa. L'unico aspetto degno di nota riguarda l'invocazione della funzione *swpInWindow*: questa funzione, mostrata nel seguito, verifica che il numero di sequenza del frame che viene confermato si trovi nell'intervallo degli ACK che il mittente si aspetta di ricevere in quel momento.

Quando il frame in arrivo contiene dei dati, *deliverSWP* invoca per prima cosa *msgStripHdr* e *load_swp_hdr*, per estrarre l'intestazione dal frame. La funzione *load_swp_hdr* è la controparte di *store_swp_hdr*, di cui abbiamo parlato precedentemente: traduce una stringa di byte nella struttura C che contiene l'intestazione SWP. Quindi, *deliverSWP* invoca *swpInWindow* per verificare che il numero di sequenza del frame sia all'interno dell'intervallo di numeri di sequenza che si aspetta di ricevere: se lo è, la funzione esegue un ciclo per esaminare l'insieme di frame consecutivi che ha ricevuto e li trasmette al protocollo di livello superiore, invocando la funzione *deliverHLP*. Invia inoltre un ACK cumulativo al mittente, ma lo fa eseguendo un ciclo sulla coda di ricezione, senza usare la variabile *SeqNumToAck* vista precedentemente.

```

static int
deliverSWP(SwpState *state, Msg *frame)
{
    SwpHdr hdr;

```

```

char *hbuf;

hbuf = msgStripHdr(frame, HLEN);
load_swp_hdr(&hdr, hbuf);
if (hdr.Flags & FLAG_ACK_VALID)
{
    /* ricevuta una conferma-comportamento da mittente */
    if (!swpInWindow(hdr.AckNum, state->LAR + 1,
                     state->LFS))
    {
        do
        {
            struct sendQ_slot *slot;

            slot = &state->sendQ[++state->LAR % SWS];
            evCancel(slot->timeout);
            msgDestroy(&slot->msg);
            semSignal(&state->sendWindowNotFull);
        } while (state->LAR != hdr.AckNum);
    }
}

if (hdr.Flags & FLAG_HAS_DATA)
{
    struct recvQ_slot *slot;

    /* ricevuto frame dati-comportamento da ricevitore */
    slot = &state->recvQ[hdr.SeqNum % RWS];
    if (!swpInWindow(hdr.SeqNum, state->NFE,
                     state->NFE + RWS - 1))
    {
        /* scarta il messaggio */
        return SUCCESS;
    }
    msgSaveCopy(&slot->msg, frame);
    slot->received = TRUE;
    if (hdr.SeqNum == state->NFE)
    {
        Msg m;
        while (slot->received)
        {
            deliverHLP(&slot->msg);
            msgDestroy(&slot->msg);
            slot->received = FALSE;
            slot = &state->recvQ[++state->NFE % RWS];
        }
        /* invia ACK */
    }
}

```

```

    prepare_ack(&m, state->NFE - 1);
    sendLINK(&m);
    msgDestroy(&m);
}
return SUCCESS;
}

```

Infine, `swpInWindow` è una semplice funzione che verifica se un certo numero di sequenza si trova tra un valore minimo ed un valore massimo.

```

static bool
swpInWindow(SwpSeqno seqno, SwpSeqno min, SwpSeqno max)
{
    SwpSeqno pos, maxpos;

    pos = seqno - min; /* pos **dovrebbe** appartenere
                           all'intervallo [0,MAX) */
    maxpos = max - min + 1; /* maxpos appartiene
                           all'intervallo [0,MAX] */

    return pos < maxpos;
}

```

Frame in ordine e controllo di flusso

Il protocollo sliding window è forse l'algoritmo più noto nel campo delle reti di calcolatori. Tuttavia, è facile fare confusione in merito a questo algoritmo, perché viene utilizzato con tre ruoli diversi. Il primo ruolo è quello sul quale ci siamo concentrati in questa sezione: consegnare in modo affidabile i frame lungo una linea di collegamento non affidabile (e, più in generale, l'algoritmo può essere utilizzato per consegnare in modo affidabile messaggi attraverso una rete non affidabile). Questa è la funzione primaria dell'algoritmo.

L'algoritmo sliding window, però, può anche essere utilizzato per mantenere l'ordine con il quale vengono trasmessi i frame. Questo compito è svolto molto semplicemente dal ricevitore: dato che ogni frame ha un numero di sequenza, basta che il ricevitore non trasferisca al protocollo di livello immediatamente superiore un frame finché non ha trasferito tutti i frame aventi un numero di sequenza inferiore. Detto in altre parole, il ricevitore memorizza, senza trasferirli oltre, i frame fuori sequenza. La versione dell'algoritmo sliding window descritta in questa sezione preserva l'ordinamento dei frame, ma potremmo immaginarne una variante in cui il ricevitore trasferisce i frame al protocollo successivo senza aspettare la consegna dei frame precedenti. Ciò che ci dovremmo chiedere è se abbiamo veramente bisogno che il protocollo sliding window preservi l'ordinamento dei frame, oppure se, invece, tale funzionalità non è necessaria al livello della linea di collegamento. Sfortunatamente, non abbiamo ancora visto abbastanza in merito all'architettura delle reti per rispondere a questa domanda: prima dobbiamo capire come una sequenza di collegamenti punto-punto sia concessa mediante switch per formare un percorso da un estremo all'altro.

Il terzo ruolo dell'algoritmo sliding window prevede che a volte venga usato per il controllo di flusso (flow control), un meccanismo di retroazione mediante il quale il ricevitore è in grado di rallentare la sorgente. Tale meccanismo viene usato per impedire alla

sorgente di sovraccaricare il destinatario, cioè per impedire la trasmissione di più dati di quelli che possono essere elaborati dal ricevente. Solitamente questo si realizza modificando il protocollo sliding window in modo che il ricevente non soltanto confermi i frame ricevuti, ma informi anche il mittente in relazione al numero di frame che è ancora in grado di accogliere, che corrisponde allo spazio di memorizzazione ancora disponibile nel ricevitore. Come nel caso della consegna ordinata, prima di inserire il controllo di flusso nel protocollo sliding window dobbiamo essere sicuri che tale funzionalità sia necessaria al livello di linea di collegamento.

Una cosa importante da ricordare in merito a questa discussione è il principio di progettazione di sistema che chiamiamo *separazione dei compiti*: in sostanza, occorre fare molta attenzione nel distinguere diverse funzioni che a volte sono compenetrate in un unico meccanismo e bisogna anche essere certi che ciascuna funzione sia necessaria e sia realizzata nel modo più efficiente. In questo caso particolare, la consegna affidabile, la consegna in ordine e il controllo di flusso sono a volte combinate in un singolo protocollo di tipo sliding window e ci dovremmo chiedere se questa sia la cosa giusta da fare al livello di linea di collegamento. Con questa domanda in mente, torneremo sull'algoritmo sliding window nel Capitolo 3 (mostrando come le reti X.25 lo usano per implementare il controllo di flusso hop-by-hop) e nel Capitolo 5 (descrivendo come TCP lo usa per realizzare un canale a flusso di byte affidabile).

2.5.3 Canali logici concorrenti

Il protocollo a livello di linea di collegamento (data link level) usato in ARPANET costituisce un'interessante alternativa al protocollo sliding window, in quanto è in grado di "mantenere piena la condutture" pur usando il semplice algoritmo stop-and-wait. Una importante conseguenza dell'uso di questo approccio è che i frame inviati su una linea non vengono mantenuti in alcun ordine particolare; inoltre, il protocollo non agisce in alcun modo sul controllo di flusso.

L'idea che sta alla base di questo protocollo di ARPANET, al quale ci riferiamo con il nome di *canali logici concorrenti* (concurrent logical channels), è il multiplexing di diversi canali logici su un singolo collegamento punto-punto, eseguendo l'algoritmo stop-and-wait su ciascuno di questi canali logici. Non viene mantenuta alcuna relazione tra i frame inviati sui diversi canali logici, tuttavia il mittente può mantenere saturata la linea di collegamento perché ciascun canale logico può avere un frame non confermato in transito sulla linea stessa.

Più precisamente, il mittente utilizza 3 bit di stato per ciascun canale: un valore booleano per indicare se il canale è attualmente occupato, il numero di sequenza ad un bit da usare per il successivo frame che verrà inviato su tale canale logico e il numero di sequenza del frame che ci si aspetta di veder arrivare dal canale. Quando il nodo ha un frame da inviare, usa il canale inattivo di numero inferiore; negli altri momenti il comportamento è identico a quello dell'algoritmo stop-and-wait.

In pratica, ARPANET usava 8 canali logici per ciascun collegamento terrestre e 16 per ciascun collegamento satellitare. Nel caso di collegamenti terrestri, l'intestazione di ciascun frame conteneva un numero di canale a 3 bit ed un numero di sequenza ad un bit, per un totale di 4 bit, che è esattamente il numero di bit richiesti dal protocollo sliding window per consentire otto frame in transito sulla linea quando $RWS = SWS$.

2.6 Ethernet (802.3)

Senza temo di smentire si può affermare che Ethernet sia la tecnologia per reti locali di maggior successo negli ultimi venti anni. Sviluppata intorno alla metà degli Anni '70 da ricercatori di Xerox Palo Alto Research Center (PARC), Ethernet è un esempio di implementazione della più generale tecnologia per reti locali denominata CSMA/CD (Carrier Sense Multiple Access with Collision Detect, accesso multiplo a sensore di portante con rilevazione di collisione).

Come indicato dal nome CSMA, Ethernet è una rete ad accesso multiplo, nel senso che un insieme di nodi inviano e ricevono frame tramite una linea di connessione condivisa; potete quindi pensare a Ethernet come ad un autobus (*bus*) con più fermate lungo la sua linea. L'espressione "carrier sense" (a sensore di portante) nell'acronimo CSMA significa che tutti i nodi sono in grado di distinguere una linea inattiva (*idle*) da una linea occupata (*busy*), mentre "collision detect" (con rilevazione di collisione) significa che un nodo rimane in ascolto mentre trasmette e può quindi capire quando un frame che sta trasmettendo subisce interferenza da (cioè collide con) un frame trasmesso da un altro nodo.

Ethernet ha le sue radici in una precedente rete radio a pacchetti, chiamata Aloha, sviluppata alla Università delle Hawaii per fornire supporto alle comunicazioni tra calcolatori situati in diverse isole dell'arcipelago. Come per la rete Aloha, il problema principale da risolvere per Ethernet è come mediare l'accesso ad un mezzo condiviso in modo equo ed efficiente (in Aloha il mezzo fisico era l'atmosfera, mentre in Ethernet si tratta di un cavo coassiale). Quindi, l'idea chiave di Aloha, come di Ethernet, consiste in un algoritmo che controlla il momento in cui ciascun nodo può trasmettere.

Nel 1978, Digital Equipment Corporation e Intel Corporation si unirono a Xerox per definire uno standard per Ethernet a 10 Mbps, che costituì la base per lo standard IEEE 802.3. Con l'unica eccezione che vedremo nella Sezione 2.6.2, è facile vedere come lo standard Ethernet del 1978 sia un sottoinsieme valido dello standard 802.3, il quale definisce una più ampia scelta di mezzi fisici sui quali Ethernet può operare e, più recentemente, è stato esteso con l'aggiunta di una versione a 100 Mbps denominata Fast Ethernet, nonché una versione a 1000 Mbps chiamata Gigabit Ethernet. La parte restante di questa sezione è focalizzata su Ethernet a 10 Mbps, poiché è la versione tipicamente usata in modalità di accesso multiplo e qui siamo interessati al modo in cui più host possono condividere una singola linea di connessione. Ethernet a 10 Mbps e a 1000 Mbps sono state progettate per essere utilizzate in configurazioni punto-punto full-duplex, per cui sono tipicamente utilizzate in reti commutate, come descritto nel capitolo successivo.

2.6.1 Proprietà fisiche

Un segmento di Ethernet si realizza con un cavo coassiale di lunghezza fino a 500 m, un cavo simile a quelli usati per la TV via cavo, tranne per il fatto che ha un'impedenza di 50 ohm anziché 75 ohm. Gli host si connettono ad un segmento di Ethernet mediante prese (*tap*), che devono essere distanti una dall'altra almeno 2.5 m. Direttamente connesso alla presa viene posto un *transceiver*, un piccolo dispositivo elettronico che rileva i momenti in cui la linea è inattiva e pilota il segnale quando, invece, l'host sta trasmettendo; inoltre, riceve i segnali in ingresso. Il transceiver è, a sua volta, collegato ad un adattatore Ethernet, infilato nell'host. Tutta la parte algoritmica che compone il protocollo Ethernet, come descritto in questa sezione, è implementata nell'adattatore (e non nel transceiver). Questa configurazione è illustrata in Figura 2.24.

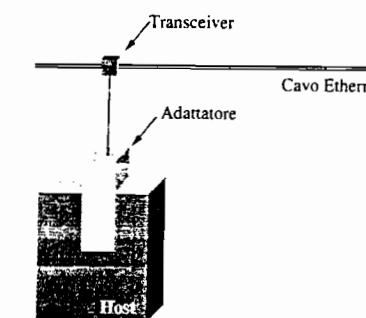
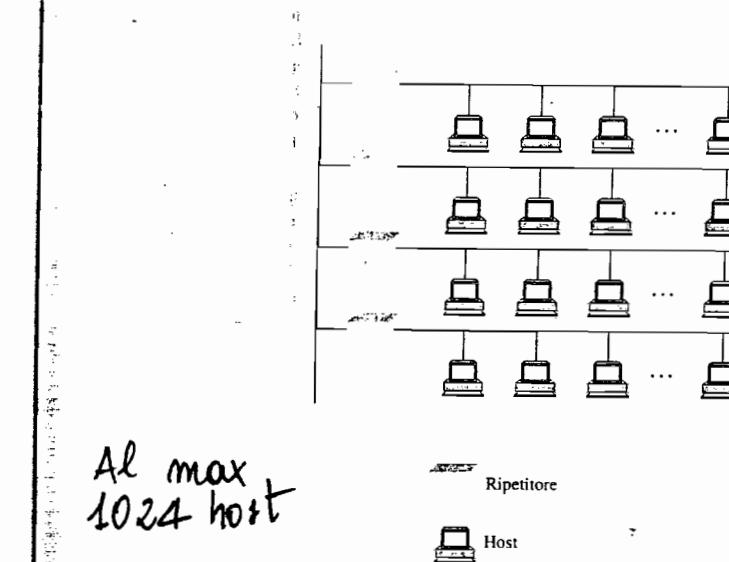


Figura 2.24 Transceiver e adattatore Ethernet.



Al max 1024 host

Ripetitore

Host

Figura 2.25 Ripetitore Ethernet.

Più segmenti Ethernet possono essere uniti insieme mediante *ripetitori*, dispositivi che inoltrano segnali digitali, più o meno come un amplificatore è in grado di inoltrare segnali analogici. In ogni caso, tra una qualsiasi coppia di host non si possono inserire più di quattro ripetitori, per cui una rete Ethernet ha una estensione massima di 2500 m. Ad esempio, usando soltanto due ripetitori tra qualsiasi coppia di host si ottiene una configurazione simile a quella illustrata in Figura 2.25, che può essere vista come un segmento che costituisce la spina dorsale verticale di un edificio, con segmenti per ciascun piano. Ciò detto, una rete Ethernet è limitata ad avere al massimo 1024 host.

Qualsiasi segnale inserito sulla Ethernet da un host viene fatto pervenire all'intera rete, cioè il segnale si propaga in entrambe le direzioni e i ripetitori inoltrano il segnale su tutti i segmenti. I terminatori connessi alla fine di ciascun segmento assorbono il segnale ed impediscono che rimbalzi, tornando indietro e interferendo con altri segnali. Ethernet usa lo schema di codifica Manchester descritto nella Sezione 2.2.

Oltre al sistema di segmenti e ripetitori appena descritto, negli anni si sono introdotte tecnologie alternative. Ad esempio, invece di usare un cavo coassiale a 50 ohm, si può costruire una Ethernet con un cavo più sottile, chiamato 10Base2 (il cavo originale viene invece chiamato 10Base5 e i due cavi sono comunemente chiamati, rispettivamente, *thin*, sottile, e *thick*, grosso). "10" in 10Base2 significa che la rete opera a 10 Mbps, "Base" si riferisce al fatto che il cavo viene usato come sistema *in banda base*, e "2" significa che un segmento non può essere lungo più di 200 m (mentre un segmento del cavo originale 10Base5 può essere lungo fino a 500 m). Oggi viene principalmente utilizzata una terza tecnologia di cavi, chiamata 10BaseT, dove "T" sta per *twisted pair*, cioè cavo in doppino intrecciato. Tipicamente vengono usati cablaggi con cavi in doppino intrecciato in Categoria 5. Un segmento 10BaseT è solitamente limitato ad una lunghezza massima di 100 m (anche Ethernet a 100 e 1000 Mbps operano su cavi in Categoria 5, fino a distanze di 100 m).

Dato che i cavi 10Base2 e 10BaseT sono ora così sottili, non ci si attacca con una presa come si farebbe con un cavo 10Base5. Con i cavi 10Base2, si usa un giunto a T inserito nel cavo; in pratica, i cavi 10Base2 sono usati per collegare un insieme di host *in daisy-chain*, cioè in cascata. La configurazione più comune con i cavi 10BaseT prevede invece la presenza di vari segmenti punto-punto connessi ad un ripetitore a molte vie, a volte chiamato *hub*, come evidenziato in Figura 2.26. Anche per Ethernet a 100 Mbps si possono connettere vari segmenti ad un hub, ma ciò non è possibile per segmenti a 1000 Mbps.

Ciò che è importante capire è che in ogni caso in una Ethernet i dati trasmessi da un host raggiungono tutti gli altri host, sia quando la Ethernet è costituita da un solo segmento, sia quando vi è una sequenza lineare di segmenti connessi da ripetitori, sia quando più segmenti sono connessi ad un hub in una configurazione a stella. E questa è una buona notizia. La cattiva notizia è che tutti questi host competono per l'accesso allo stesso mezzo fisico e, di conseguenza, si dice che appartengono allo stesso *dominio di collisione*.

realizzato in HW nell'adattatore di rete

2.6.2 Protocollo di accesso

Volgiamo ora la nostra attenzione all'algoritmo che controlla l'accesso alla linea Ethernet condivisa. Tale algoritmo viene comunemente chiamato *controllo di accesso al mezzo* (*media access control*, MAC) ed è tipicamente realizzato in hardware nell'adattatore di rete. Non descrivremo l'hardware in senso stretto, ma concentreremo la nostra attenzione sull'algoritmo che esso implementa; prima, però, descriviamo il formato del frame e dell'indirizzo Ethernet.

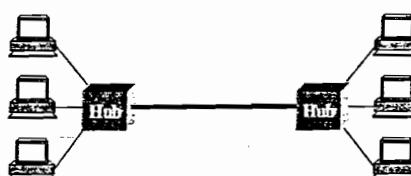


Figura 2.26 Hub Ethernet.

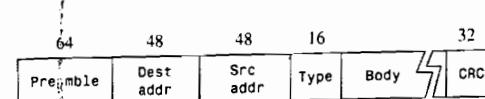


Figura 2.27 Formato del frame Ethernet.

Formato del frame

Ciascun frame Ethernet è definito dal formato illustrato nella Figura 2.27. Il preamble di 64 bit consente al ricevitore di sincronizzarsi con il segnale: è una sequenza di 0 e 1 che si alternano. Gli host sorgente e destinazione sono identificati da un indirizzo a 48 bit. Il campo del tipo di pacchetto serve come chiave di demultiplexing, cioè identifica a quale protocollo di livello superiore, tra i molti possibili, va consegnato il frame. Ciascun frame contiene fino a 1500 byte di dati e deve contenere, come minimo, 46 byte di dati, anche se per raggiungere questo scopo devono essere aggiunti dei byte riempitivi prima della trasmissione. Il motivo per cui esiste questa dimensione minima per il frame è che il frame deve essere sufficientemente lungo da consentire la rilevazione di una collisione, problema di cui parleremo più avanti. Infine, ciascun frame contiene un CRC a 32 bit. Come il protocollo HDLC descritto nella Sezione 2.3.2, Ethernet è un protocollo con framing orientato al bit. Notate che, dal punto di vista dell'host, un frame Ethernet ha un'intestazione di 14 byte: due indirizzi di 6 byte e un campo di tipo di 2 byte; il preamble, il codice CRC e gli eventuali bit riempitivi vengono aggiunti dall'adattatore sorgente prima della trasmissione e vengono rimossi dall'adattatore di destinazione.

Il formato del frame appena descritto è quello dello standard Ethernet di Digital-Intel-Xerox. Il formato del frame 802.3 è esattamente identico, tranne per la sostituzione del campo di tipo a 16 bit con un campo di lunghezza, sempre a 16 bit. Lo standard 802.3 viene solitamente usato in coppia con uno standard di encapsulamento che definisce un proprio campo di tipo usato come chiave di demultiplexing per i frame in arrivo: questo campo di tipo si trova all'inizio della porzione di dati del frame 802.3, cioè è immediatamente successivo all'intestazione. Poiché lo standard Ethernet ha evitato l'uso di valori di tipo inferiori a 1500 (la lunghezza massima che si può trovare in un'intestazione 802.3) ed i campi di tipo e di lunghezza si trovano nella stessa posizione dell'intestazione, è possibile che un dispositivo accetti entrambi i formati, con il software in esecuzione sull'host (device driver) che interpreta gli ultimi 16 bit dell'intestazione come tipo o come lunghezza. In pratica, la maggior parte degli host seguono il formato Digital-Intel-Xerox e interpretano, quindi, questo campo come tipo del frame.

Indirizzi

Ciascun host di una Ethernet ha un indirizzo Ethernet univoco (in effetti, ciò è vero per qualsiasi host Ethernet nel mondo). Tecnicamente l'indirizzo è relativo all'adattatore, non all'host, e solitamente è scritto in una ROM. Gli indirizzi Ethernet vengono solitamente visualizzati come una sequenza di sei numeri separati dal carattere "due punti"; ciascun numero corrisponde ad uno dei 6 byte dell'indirizzo ed è scritto come una coppia di cifre esadecimali, una per ciascuna delle due parti di 4 bit del byte, eliminando eventuali zeri iniziali. Ad esempio, 8:0:2b:e4:b1:2 è la rappresentazione leggibile dell'indirizzo Ethernet

00001000 00000000 00101011 11100100 10110001 00000010

Per essere certi che ciascun adattatore abbia un indirizzo univoco, a ciascun produttore di dispositivi Ethernet viene assegnato un diverso prefisso che deve costituire la parte iniziale dell'indirizzo di ogni adattatore da esso costruito: ad esempio, Advanced Micro Devices usa il prefisso a 24 bit **x080020** (o **8:0:20**). Ciascun produttore, poi, deve garantire che i suffissi degli indirizzi dei dispositivi che produce siano unici.

Ciascun frame trasmesso in un rete Ethernet viene ricevuto da tutti gli adattatori ad essa connessi, ciascun adattatore identifica i frame che sono destinati al suo indirizzo e passa, soltanto quelli all'host: un adattatore può anche essere programmato per funzionare in modo *promiscuo*, nel qual caso consegna all'host tutti i frame ricevuti, ma questo non è il funzionamento normale. Oltre a questi indirizzi *unicast*, esiste un indirizzo Ethernet speciale, avente tutti i bit al valore 1, che ha la funzione di indirizzo *broadcast*: i frame con destinazione broadcast vengono passati da tutti gli adattatori al proprio host. In modo simile, un indirizzo che abbia il primo bit al valore 1 ma che non sia l'indirizzo broadcast viene chiamato indirizzo *multicast*. Un host può programmare il proprio adattatore perché accetti alcuni insiemi di indirizzi multicast, per cui gli indirizzi multicast vengono utilizzati per inviare messaggi a sottoinsiemi degli host di una Ethernet (ad esempio, a tutti i server di file). Per riassumere, un adattatore Ethernet riceve tutti i frame, mentre accetta soltanto:

- i frame destinati al proprio indirizzo
- i frame destinati all'indirizzo broadcast
- i frame destinati ad un indirizzo multicast, se ha ricevuto istruzioni per ascoltare il traffico destinato a tale indirizzo
- tutti i frame, se è stato configurato per funzionare in modalità promiscua

L'adattatore passa all'host soltanto i frame che accetta. •

Algoritmo del trasmettitore

Come abbiamo appena visto, la parte ricevente del protocollo Ethernet è semplice; la parte interessante è quella realizzata nella parte che trasmette, il cui algoritmo di trasmissione è definito come segue.

Quando l'adattatore ha un frame da inviare e la linea è inattiva, il frame viene trasmesso immediatamente, senza alcuna negoziazione con gli altri adattatori. Il limite superiore di 1500 byte per il messaggio garantisce che l'adattatore può impegnare la linea soltanto per un periodo di tempo prefissato.

Quando un adattatore ha un frame da inviare e la linea è impegnata, aspetta finché la linea diventa inattiva e poi trasmette immediatamente². Si dice che Ethernet è un *protocollo con persistenza* perché un adattatore avente un frame da trasmettere trasmette con probabilità 1 ogni volta una linea impegnata diventa inattiva. In generale, un algoritmo *con persistenza p* trasmette con probabilità $0 \leq p \leq 1$ dopo che una linea è diventata inattiva, mentre differisce la trasmissione con probabilità $q = 1 - p$. Il motivo per cui si sceglie un valore di $p < 1$ è che ci potrebbero essere più adattatori in attesa che la linea impegnata diventi inattiva, e non si vuole che inizino tutti a trasmettere nello stesso istante. Se ciascun adattatore trasmette

² Per essere più precisi, tutti gli adattatori aspettano 9.6 µs alla fine di un frame prima di trasmettere il frame successivo. Ciò vale sia per chi ha trasmesso il primo frame, sia per tutti i nodi che stanno aspettando che la linea diventi inattiva.

immediatamente con una probabilità, ad esempio, del 33%, allora ci possono essere fino a tre adattatori in attesa di trasmettere, con una probabilità molto elevata che soltanto uno inizi a trasmettere quando la linea diventa inattiva. Nonostante questo ragionamento, un adattatore Ethernet inizia sempre a trasmettere non appena si accorge che la linea è divenuta inattiva, e ciò si è dimostrato essere molto efficiente.

Per completare il discorso sui protocolli con persistenza p nel caso in cui $p < 1$, ci potremmo chiedere per quanto tempo una sorgente che decide di differire la trasmissione deve attendere prima di poter trasmettere. La risposta per la rete Aloha, che ha sviluppato originariamente questo tipo di protocollo, consiste nel dividere il tempo in intervalli discreti, dove ciascun intervallo corrisponde al tempo necessario per trasmettere un intero frame. Ogni volta che un nodo ha un frame da inviare e si accorge della presenza di un intervallo libero (inattivo), trasmette con probabilità p ed attende l'intervalllo successivo con probabilità $q = 1 - p$. Se tale intervallo successivo è anch'esso vuoto, il nodo decide nuovamente se trasmettere oppure no, con le stesse probabilità p e q . Se, invece, l'intervalllo successivo non è vuoto, perché qualche altra stazione ha deciso di trasmettere, allora il nodo attende semplicemente il successivo intervallo inattivo e l'algoritmo viene ripetuto.

Tornando alla discussione di Ethernet, poiché non esiste un controllo centralizzato, è possibile che due (o più) adattatori inizino a trasmettere nello stesso momento, perché hanno visto la linea inattiva oppure hanno aspettato che la linea impegnata diventasse inattiva. Quando ciò accade, si dice che i due (o più) frame hanno *colliso* sulla rete; dato che Ethernet è in grado di rilevare le collisioni, ciascuna sorgente può stabilire che si sta verificando una collisione. Nel momento in cui un adattatore si accorge che il suo frame sta entrando in collisione con un altro, per prima cosa trasmette una sequenza di disturbo di 32 bit (*jammer sequence*), poi interrompe la trasmissione; in questo modo, in caso di collisione un trasmettitore invierà, come minimo, soltanto 96 bit: 64 bit di preamble e 32 bit di sequenza di disturbo.

Un caso in cui un adattatore invierà soltanto 96 bit (che vengono chiamati *runt frame*) si ha quando i due host che generano la collisione sono adiacenti. Se i due host fossero lontani, prima di osservare la collisione dovrebbero trasmettere più a lungo, e quindi inviare più bit: il caso peggiore si ha quando i due host si trovano in posizioni opposte della rete. Per avere la certezza che il frame appena inviato non ha colliso con un altro frame, può darsi che il trasmettitore debba inviare fino a 512 bit: non è un caso che qualsiasi frame Ethernet debba avere almeno 512 bit (64 byte): 14 byte di intestazione, 46 byte di dati e 4 byte di CRC.

Perché proprio 512 bit? La risposta è correlata ad un'altra domanda che vi dovreste porre a proposito di Ethernet: perché la sua lunghezza è limitata a soli 2500 m? Perché non 10 o 1000 km? La risposta ad entrambe le domande ha a che fare con il fatto che più due nodi sono distanti, più tempo serve al frame inviato da un nodo per raggiungere l'altro nodo, e in tale intervallo di tempo la rete è vulnerabile al fenomeno delle collisioni!

La Figura 2.28 mostra la situazione nel caso peggiore, quando gli host A e B si trovano ai due estremi opposti della rete. Supponete che l'host A inizi a trasmettere un frame all'istante t , come mostrato in (a); indicando la latenza del collegamento con d , il frame impiega un tempo d per raggiungere A, per cui il primo bit del frame di A arriva in B all'istante $t + d$, come si può vedere in (b). Supponete che un'istante prima che il frame di A arrivi in B (per cui B vede ancora la linea inattiva), l'host B inizi a trasmettere il proprio frame: esso colliderrà immediatamente con il frame di A e tale collisione sarà rilevata dall'host B (c). Come descritto in precedenza, quindi, l'host B invierà la sequenza di disturbo di 32 bit (e il frame di B sarà un *runt frame*): sfortunatamente, l'host A non saprà del avvenuta collisione finché non arriverà il frame di B, cosa che accadrà dopo un lasso di tempo uguale alla latenza della linea,

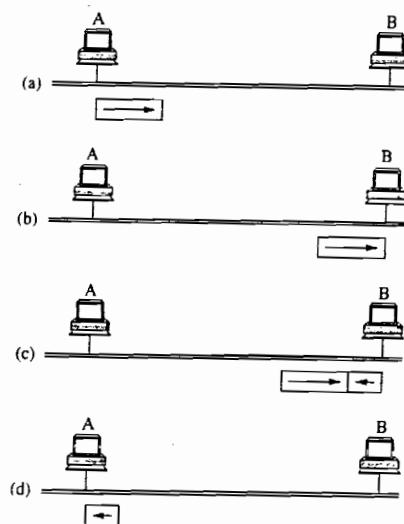


Figura 2.28 Scenario di caso peggiore: (a) A invia un frame all'istante t ; (b) il frame di A arriva in B all'istante $t + d$; (c) B inizia a trasmettere all'istante $t + d$ e genera una collisione con il frame di A; (d) il *runt frame* di B (32 bit) arriva in A all'istante $t + 2d$.

cioè all'istante $t + 2 \times d$, come mostrato in (d). Per essere certo di rilevare una collisione, l'host A deve continuare a trasmettere fino a tale istante, cioè per un tempo pari a $2 \times d$. Considerando che la Ethernet più estesa può essere di 2500 m e che tra due host qualsiasi ci possono essere fino a quattro ripetitori, il ritardo di round-trip è stato quantificato in 51.2 μs , che in una Ethernet a 10 Mbps corrispondono a 512 bit. Per vedere questa situazione in un altro modo, possiamo dire che dobbiamo limitare la latenza massima di una Ethernet ad un valore sufficientemente piccolo (ad esempio, 51.2 μs) perché l'algoritmo per l'accesso funzioni; quindi, la lunghezza massima di una Ethernet deve essere dell'ordine di 2500 m.

Una volta che un adattatore ha rilevato una collisione e ha interrotto la propria trasmissione, attende un certo periodo di tempo e riprova. Ogni volta che tenta di trasmettere e fallisce, l'adattatore raddoppia il periodo di tempo di attesa prima del tentativo successivo: questa strategia di raddoppio dell'intervallo di attesa fra due tentativi di ritrasmissione è una tecnica generale che va sotto il nome di *backoff esponenziale*. Più precisamente, il principio intervallo di attesa dell'adattatore è 0 oppure 51.2 μs , a caso. Se questo tentativo fallisce, l'attesa successiva viene selezionata a caso tra 0, 51.2, 102.4 e 153.6 μs , cioè $k \times 51.2 \mu s$, con k che va da 0 a 3. Dopo la terza collisione, l'attesa viene selezionata, nuovamente a caso, tra i valori $k \times 51.2 \mu s$, con k che va da 0 a $2^3 - 1$. In generale, quindi, l'algoritmo seleziona a caso un valore di k tra 0 e $2^n - 1$ ed aspetta un tempo uguale a $k \times 51.2 \mu s$, dove n è il numero di collisioni avvenute fino a quel momento. Dopo un certo numero di tentativi l'adattatore smette di riprovare e segnala all'host un errore; tipicamente vengono fatti 16 tentativi, anche se l'algoritmo di backoff limita il valore di n nella formula precedente al valore massimo di 10.

2.6.3 L'esperienza di Ethernet

Dato che le reti Ethernet sono in uso da tanti anni e sono assai diffuse, abbiamo una grande esperienza relativamente al loro utilizzo. Una delle osservazioni più importanti che si sono fatte sulle Ethernet è che funzionano al meglio con condizioni di basso carico. Ciò accade perché con carichi elevati (tipicamente si considera un carico elevato per Ethernet un'utilizzazione superiore al 30%) una gran parte della capacità di trasporto della rete è sprecata nelle collisioni.

Fortunatamente la maggior parte delle reti Ethernet sono usate in modo più restrittivo di quanto consentito dallo standard. Ad esempio, la maggior parte delle reti Ethernet hanno meno di 200 host connessi, un valore molto inferiore al massimo, 1024 (provate a scoprire, nel Capitolo 4, un motivo per questo limite superiore corrispondente a circa 200 host). Analogamente, la maggior parte delle reti Ethernet ha un'estensione inferiore a 2500 m, con un ritardo di round-trip più vicino a 5 μs che a 51.2 μs . Un altro fattore che rende ben utilizzabili le Ethernet consiste nel fatto che, nonostante gli adattatori Ethernet non realizzino un controllo di flusso a livello di linea di collegamento, gli host forniscono tipicamente un meccanismo di controllo di flusso da un estremo all'altro della comunicazione (*end-to-end*). Di conseguenza, è raro che si verifichino situazioni in cui un host trasmette frame in continuazione sulla rete.

Infine, è bene spendere ancora qualche parola per spiegare il motivo del successo di Ethernet, in modo che possiate capire le proprietà che dovreste imitare con qualsiasi tecnologia per rete locale che cerchi di rimpiazzare Ethernet. Innanzitutto, è estremamente facile gestire una rete Ethernet e farne la manutenzione: non ci sono switch che possano funzionare male, non ci sono tabelline di configurazione o di routing da tenere aggiornate, e aggiungere un host alla rete è molto facile: diciamo che è difficile immaginare una rete che sia più semplice da gestire. Secondo, è poco costosa: i cavi costano poco e l'unico altro costo è l'adattatore di rete in ciascun host. Qualsiasi approccio basato sulla commutazione richiede un investimento in una infrastruttura che ha un costo relativamente alto (gli switch), oltre ad un costo maggiore per ciascun adattatore. Come vedrete nel capitolo successivo, la tecnologia comunitata per reti locali in uso oggi è essa stessa basata su Ethernet.

2.7 Reti token ring (802.5 e FDDI)

Insieme a Ethernet, le reti di tipo *token ring* (anello con testimone) sono la categoria più significativa di reti con mezzo condiviso. Esistono diversi tipi di token ring: questa sezione presenterà il tipo che è stato prevalente per molti anni, con il nome di IBM Token Ring. Come la Ethernet di Xerox, anche la rete Token Ring di IBM ha uno standard IEEE quasi identico, 802.5, e quando sarà necessario metteremo in evidenza le diversità tra la rete token ring di IBM e quella dello standard 802.5.

Presentando gli standard IBM e 802.5 è possibile comprendere la maggior parte dei principi generali delle reti di tipo token ring, ma anche lo standard FDDI (Fiber Distributed Data Interface), un nuovo e più veloce tipo di token ring, merita qualche approfondimento, che forniremo alla fine di questa sezione. Al momento in cui scriviamo è in fase di definizione anche un altro standard per reti di tipo token ring, denominato Resilient Packet Ring o 802.17.

Direzione costante

Come suggerito dal nome, una rete di tipo token ring è composta da un insieme di nodi connessi ad anello (Figura 2.29); i dati fluiscono sempre in una particolare direzione lungo l'anello, con i nodi che ricevono frame dal loro vicino a monte e li inviano al loro vicino a valle. Questa topologia ad anello è in netto contrasto con la topologia a bus di Ethernet, ma, come per Ethernet, l'anello viene visto come un unico mezzo fisico condiviso e non si comporta come una collezione di linee di collegamento punto-punto indipendenti che, per caso, siano state configurate per formare un anello. Di conseguenza, un anello con testimone condivide con Ethernet due proprietà fondamentali: per prima cosa c'è la necessità di un algoritmo distribuito che controlli quando ciascun nodo è abilitato a trasmettere; secondo, tutti i nodi vedono tutti i frame, con il nodo identificato nell'intestazione del frame come destinazione che memorizza al proprio interno una copia del frame mentre questo prosegue il proprio percorso.

La parola "token" (testimone, nel senso di testimone in una staffetta) nell'espressione token ring deriva dal modo in cui viene gestito l'accesso all'anello condiviso. L'idea è che un testimone, che in realtà è soltanto una speciale sequenza di bit circola nell'anello: ciascun nodo riceve e inoltre il testimone. Quando un nodo che ha un frame da trasmettere vede il testimone, lo toglie dall'anello (cioè non inoltra la speciale sequenza di bit) e vi inserisce, invece, il proprio frame. Ciascun nodo lungo l'anello inoltra semplicemente il frame, mentre il nodo di destinazione copia il frame al proprio interno, per poi, comunque, inoltrarlo. Quando il frame ritorna alla sorgente, dopo aver viaggiato per tutto l'anello, il nodo che lo aveva inviato lo elimina dall'anello (invece di inoltrarlo nuovamente) e vi inserisce il testimone. In questo modo, qualche nodo a valle avrà l'opportunità di trasmettere un frame. L'algoritmo di accesso al mezzo è equo, nel senso che il testimone circola nell'anello e ciascun nodo ha la possibilità di trasmettere, a turno (round robin).

Ciascun nodo, a turno, può trasmettere

2.7.1 Proprietà fisiche

Una delle prime cose di cui vi dovete preoccupare quando avete a che fare con una topologia ad anello è il fatto che un guasto ad una qualsiasi linea di collegamento o ad un qualsiasi nodo rende inutilizzabile l'intera rete. Questo problema viene risolto connettendo ciascuna stazio-

→ Rete

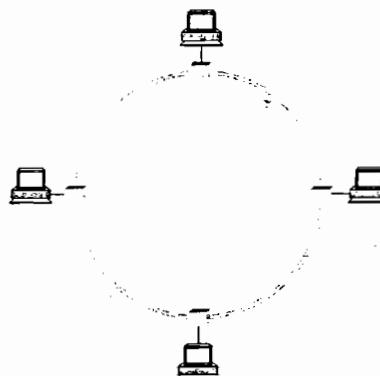


Figura 2.29 Rete token ring.

ne all'anello mediante un relè elettromeccanico: finché la stazione è ben funzionante, il relè è aperto e la stazione è inserita nell'anello; se la stazione smette di fornire energia, il relè si chiude e l'anello esclude automaticamente la stazione, come mostrato in Figura 2.30.

Solitamente alcuni di questi relè vengono raggruppati in un unico dispositivo, denominato unità di accesso multi-stazione (MSAU, multi-station access unit), con l'interessante conseguenza di far assumere all'anello una configurazione molto simile ad una stella, come evidenziato in Figura 2.31. Inoltre ciò rende assai facile l'aggiunta e la rimozione di stazioni dalla rete, poiché basta collegarle o scollarle alla MSAU più vicina, mentre il cablaggio globale della rete rimane immutato. Una delle marginali differenze che sussistono tra la specifica di Token Ring di IBM e lo standard 802.5 è che la prima richiede l'uso delle MSAU, mentre il secondo non le ritiene necessarie. In pratica, comunque, le MSAU sono utilizzate quasi sempre, sia per rendere la rete più resistente ai malfunzionamenti sia per agevolare l'inserimento e la rimozione di stazioni.

Ci sono ancora alcuni dettagli fisici da sapere relativamente a 802.5 e Token Ring di IBM. La velocità dei dati può essere 4 o 16 Mbps. La codifica dei bit avviene con codice

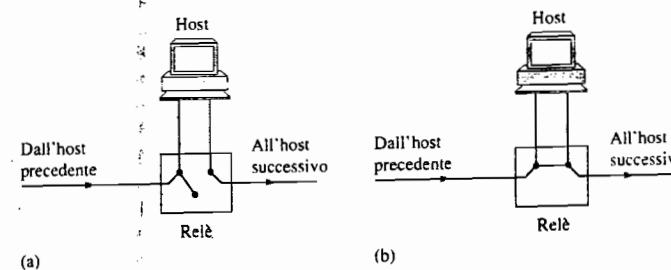


Figura 2.30 Relè usato lungo un token ring: (a) relè aperto, host attivo; (b) relè chiuso, host escluso.

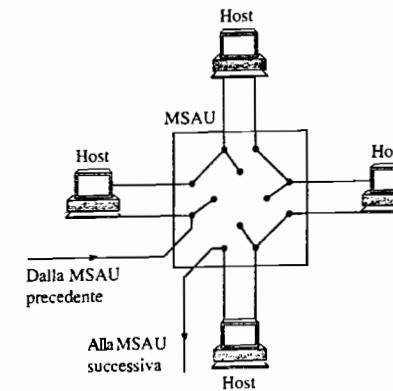


Figura 2.31 Unità di accesso multi-stazione.

Manchester differenziale, come visto nella Sezione 2.2. Le reti Token Ring di IBM possono avere fino a 260 stazioni per anello, mentre 802.5 impone il limite di 250. Il mezzo fisico è costituito da cavi in doppino per IBM, mentre non viene specificato da 802.5.

2.7.2 Controllo di accesso al mezzo in reti token ring

MAC 802.5

Ora è giunto il momento di dare uno sguardo più da vicino a come il protocollo MAC (medium access control) opera in una rete di tipo token ring. L'adattatore di rete per token ring contiene un ricevitore, un trasmettitore e la capacità di memorizzare 32 o più bit di dati fra di essi. Quando nessuna delle stazioni connesse all'anello ha qualcosa da trasmettere, nell'anello circola soltanto il testimone, per cui, ovviamente, l'anello deve avere sufficiente "capacità di memorizzazione" da contenere un intero testimone. Ad esempio, il testimone dello standard 802.5 è lungo 24 bit. Se ogni stazione potesse contenere un solo bit (come è normale in reti 802.5) e le stazioni fossero così vicine da poter trascurare il tempo di propagazione di un bit da una stazione alla successiva, dovremmo avere almeno 24 stazioni connesse all'anello perché questo possa operare correttamente. Questa situazione viene evitata attribuendo ad una predeterminata stazione, chiamata monitor, il compito di inserire nell'anello alcuni bit di ritardo, se necessario. Il funzionamento di questo monitor è descritto con maggiore dettaglio in seguito.

Mentre il testimone circola lungo l'anello, una stazione che abbia dati da inviare può impossessarsene, evitando di inoltrarlo e spedendo invece i dati. Nelle reti 802.5, impossessarsi del testimone significa semplicemente modificare un bit nel secondo byte del testimone stesso: i primi 2 byte del testimone così modificato diventano quindi il preamble del successivo pacchetto di dati. Una volta che una stazione si sia impossessata del testimone, ha la possibilità di inviare uno o più pacchetti: il numero esatto dipende da alcuni fattori descritti in seguito.

a seconda del THT

Ogni pacchetto trasmesso contiene l'indirizzo di destinazione del ricevitore previsto, oppure può contenere un indirizzo multicast (o broadcast) per raggiungere più di un ricevitore (o tutti). Mentre i pacchetti attraversano l'anello, ciascun nodo li esamina per vedere se è il ricevitore previsto: in tal caso, il nodo copia in un buffer il pacchetto mentre attraversa l'adattatore di rete, ma non lo rimuove dall'anello. Soltanto la stazione che ha inviato il pacchetto ha la responsabilità di rimuoverlo dall'anello. Se un pacchetto è più lungo del numero di bit che trovano posto lungo l'anello, la stazione di invio inizierà a rimuovere dall'anello la parte iniziale del pacchetto mentre ne sta ancora trasmettendo la parte finale.

Un problema che dobbiamo esaminare consiste nel determinare quanti dati possono essere trasmessi da un nodo ogni volta che ha il possesso del testimone, o, detto altrimenti, per quanto tempo un nodo può trattenere il token (questo tempo viene chiamato THT, token holding time, tempo di possesso del testimone). Se ipotizziamo che in un istante qualsiasi la maggior parte dei nodi della rete non abbia dati da inviare (un'ipotesi ragionevole, certamente sfruttata da Ethernet) allora potremmo fare in modo che il nodo che si è impossessato del testimone possa trasmettere dati finché ne ha, prima di passare il testimone al nodo successivo: ciò significherebbe assegnare a THT il valore infinito. In questo caso sarebbe insensato vincolare il nodo all'invio di un solo messaggio, costringendolo ad attendere che il testimone faccia il giro dell'intero anello prima di avere la possibilità di inviare un altro messaggio. Ovviamente, "trasmettere dati finché ne ha" potrebbe essere pericoloso, perché una singola stazione potrebbe trattenere il testimone per un tempo indefiniteamente lungo, però potremmo usare per THT un valore significativamente più lungo del tempo necessario per trasmettere un singolo pacchetto.

Tx

Ogni stazione esamina il traffico

2.7 Reti token ring (802.5 e FDDI)

Non è difficile osservare che quanti più byte possono essere inviati da un nodo ogni volta che si impossessava del testimone, tanto migliore è l'utilizzazione dell'anello che si ottiene nel caso in cui un solo nodo abbia dati da inviare. Lo svantaggio, ovviamente, è che questa strategia non funziona bene quando vi sono più nodi che hanno dati da inviare, perché favorisce i nodi che hanno una maggiore quantità di dati da inviare rispetto ai nodi che devono spedire un solo piccolo messaggio, anche se fosse importante consegnare tale piccolo messaggio il più presto possibile. La situazione è analoga a ciò che si verifica quando vi trovate in coda ad uno sportello bancario dietro ad un cliente che deve chiedere un prestito per acquistare un'automobile, mentre voi dovete soltanto incassare un assegno. Nelle reti 802.5, il valore normale di THT è 10 ms.

Nell'uso di THT occorre fare attenzione ad un dettaglio: prima di inviare lungo l'anello ciascun pacchetto, la stazione deve verificare che il tempo necessario a trasmettere il pacchetto non porti a superare il tempo THT concesso alla stazione stessa. Ciò significa che occorre tenere traccia di quanto tempo sia trascorso da quando il nodo si è impossessato del testimone, usando poi la lunghezza del pacchetto che si vuole inviare.

Dal tempo di possesso del testimone possiamo derivare un'altra utile quantità, il tempo di rotazione del testimone (TRT, token rotation time), che è il tempo, misurato da un certo nodo, necessario al testimone per attraversare l'intero anello. Si può facilmente vedere come sia

$$\text{TOKEN ROTATION TIME} = \frac{\text{NumeroDiNodiAttivi} \times \text{THT} + \text{LatenzaDellAnello}}{\text{NumeroDiNodiAttivi}}$$

dove LatenzaDellAnello indica il tempo necessario al testimone per attraversare l'intero anello quando nessuno ha dati da trasmettere, e NumeroDiNodiAttivi è il numero di nodi che hanno dati da trasmettere.

Il protocollo 802.5 fornisce un servizio di consegna affidabile usando 2 bit nella coda (trailer) del pacchetto, i bit A e C, che valgono inizialmente entrambi 0. Quando una stazione osserva un frame di cui è il ricevitore previsto, imposta a 1 il bit A nel frame stesso; quando copia il frame all'interno del proprio adattatore, ne imposta a 1 il bit C. Se la stazione sorgente vede ritornare, dalla parte opposta dell'anello, il pacchetto con il bit A ancora a 0, ne deduce che il ricevitore previsto non funziona o è assente del tutto. Se il bit A ha il valore 1 ma il bit C vale ancora 0, allora per qualche ragione (ad esempio, per mancanza di spazio nel buffer) il destinatario non è stato in grado di accettare il frame: di conseguenza, il frame può essere ragionevolmente ritrasmesso, nella speranza che si sia liberato spazio nel buffer.

Un altro dettaglio del protocollo 802.5 è relativo al supporto fornito per diversi livelli di priorità. Il testimone contiene un campo di priorità a 3 bit, in modo che si possa immaginare che il testimone abbia, in un determinato istante, una certa priorità n. Ogni dispositivo che voglia trasmettere un pacchetto, assegna una priorità a tale pacchetto ed il dispositivo può impossessarsi del testimone per trasmettere il pacchetto soltanto se la priorità del pacchetto non è inferiore alla priorità del testimone. La priorità del testimone viene modificata nel tempo in seguito all'utilizzo di tre bit di prenotazione (reservation) nell'intestazione del frame. Ad esempio, se una stazione Y che voglia inviare un pacchetto con priorità n vede passare un frame di dati i cui bit di prenotazione non siano già stati impostati ad un valore maggiore di n, li imposta a tale valore; a causa di tale prenotazione, la stazione che attualmente è in possesso del testimone ne aumenta la priorità, portandola a n, quando lo rilascia. Sarà cura della stazione X riportare la priorità del testimone al suo valore precedente quando ha terminato.

Note che questo è uno schema di priorità rigido, nel senso che quando ci sono in attesa pacchetti ad elevata priorità non viene inviato nessun pacchetto con priorità inferiore: in

Nella CODA Bit {C}

R: trasm PRIORITÀ 3 bit

La priorità del token cambia con le prenotazioni

questo modo pacchetti a bassa priorità possono restare bloccati fuori dall'anello per lunghi periodi, se vi sono molti pacchetti ad elevata priorità.

Per completare la nostra discussione relativa al protocollo MAC parleremo di un ultimo aspetto: quando il nodo sorgente rilascia il testimone. Come mostrato in Figura 2.32, il nodo sorgente può reinserire il testimone nell'anello immediatamente dopo il proprio frame (procedura chiamata rilascio tempestivo, early release, oppure dopo che il frame trasmesso ha completato il proprio giro attorno all'anello ed è stato rimosso (rilascio ritardato, delayed release)). Ovviamente, il rilascio tempestivo consente una miglior utilizzazione dell'ampiezza di banda, specialmente in anelli molto estesi. Lo standard 802.5 usò originariamente il rilascio ritardato, ma in seguito venne aggiunta la possibilità di usare il rilascio tempestivo.

EARLY TOKEN RELEASE

2.7.3 Gestione di una rete token ring ACTIVE MONITOR

Come abbiamo evidenziato in precedenza, le reti di tipo token ring hanno una stazione che funge da monitor, il cui compito è garantire l'integrità dell'anello. Qualsiasi stazione dell'anello può avere la funzione di monitor ed esistono procedure ben definite per determinare la scelta della stazione monitor quando l'anello viene connesso per la prima volta oppure quando il monitor esistente si guasta. Un monitor ben funzionante annuncia periodicamente la propria presenza con uno speciale messaggio di controllo: se una stazione non osserva tale messaggio per un certo periodo di tempo, ne dedurrà che il monitor si è guastato e tenerà di diventare monitor essa stessa. Le procedure per scegliere un monitor sono le stesse sia nel caso in cui l'anello sia appena divenuto operativo sia nel caso in cui il monitor attivo ha appena avuto un malfunzionamento.

Quando una stazione decide che c'è bisogno di un nuovo monitor trasmette un frame con il quale "reclama il testimone" ("claim token"), annunciando così la propria intenzione di diventare il nuovo monitor. Se tale frame compie un intero giro dell'anello e torna alla sorgente, essa acquista il diritto di diventare il nuovo monitor. Se nello stesso istante ci sono altre stazioni che stanno tentando di diventare monitor, chi ha inviato un "claim token" ne vede un altro inviato da una diversa stazione: in tal caso sarà necessario risolvere la disputa usando qualche regola ben definita, come, ad esempio, "l'indirizzo più alto vince".

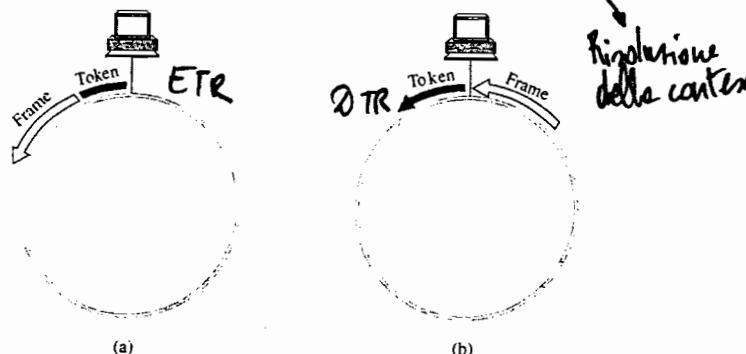


Figura 2.32 Rilascio del token: (a) tempestivo e (b) ritardato.

Raggiunto l'accordo sul monitor, esso svolge un certo numero di funzioni. Abbiamo già visto che può esserci bisogno di inserire un ritardo aggiuntivo nell'anello. Il monitor ha anche la responsabilità di garantire che esista sempre un testimone da qualche parte nell'anello, in circolazione o trattenuto da una stazione. Dovrebbe essere chiaro che il testimone può svanire per diversi motivi, come un errore di bit o un malfunzionamento della stazione che lo stava trattenendo. Per accorgersi che manca il testimone, il monitor controlla i testimoni che passano e gestisce un temporizzatore che scade dopo un intervallo di tempo uguale al valore massimo possibile per il tempo di rotazione del testimone. Questo intervallo è uguale a

$$\text{NumeroDiStazioni} \times \text{THT} + \text{LatenzaDellAnello}$$

dove NumeroDiStazioni è il numero di stazioni presenti nell'anello e LatenzaDellAnello è il ritardo di propagazione totale dell'anello. Se scade il tempo senza che il monitor abbia visto passare un testimone, ne crea uno nuovo.

Il monitor controlla anche la presenza di frame corrotti o orfani. I primi hanno errori nella somma di controllo o un formato non valido e senza l'intervento del monitor potrebbero circolare nell'anello per sempre: il monitor li elimina dall'anello evitando di inoltrarli. Un frame orfano è stato trasmesso correttamente lungo l'anello, giungendo al destinatario, ma la sua sorgente non è più attiva sull'anello, perché ha smesso di funzionare prima di rimuovere il frame dall'anello stesso. Questi frame orfani vengono identificati usando un altro bit dell'intestazione, il bit "monitor", che vale 0 quando viene trasmesso il pacchetto e cambia di valore la prima volta che il pacchetto attraversa il monitor. Se il monitor vede passare un pacchetto con questo bit al valore 1, sa che il pacchetto sta attraversando l'anello per la seconda volta e quindi lo elimina.

Un'ulteriore funzione di manutenzione dell'anello è la rilevazione di stazioni non più funzionanti. I relè nelle MSAU sono in grado di isolare automaticamente una stazione che è stata disconnessa o spenta, ma non sono in grado di rilevare malfunzionamenti più lievi. Se una qualsiasi stazione ha il sospetto che lungo l'anello ci sia un guasto, può inviare un frame di avvertimento (beacon) alla destinazione sospettata. Sulla base di quanto lontano arriva tale frame, è possibile determinare lo stato dell'anello e le stazioni guaste possono essere isolate dai relè delle MSAU.

2.7.4 Formato del frame

Siamo ora pronti per definire il formato del frame 802.5, delineato in Figura 2.33. Come già detto, 802.5 usa la codifica Manchester differenziale, e questa proprietà è sfruttata dal formato del frame, che usa codici Manchester "illegali" come marcatori di inizio e fine. Dopo il marcitore iniziale (start delimiter) si trova il byte per il controllo d'accesso (access control), che contiene la priorità del frame e la prenotazione di priorità di cui abbiamo parlato in precedenza. Il byte di controllo del frame (frame control) è una chiave di demultiplexing che identifica il protocollo di livello superiore.

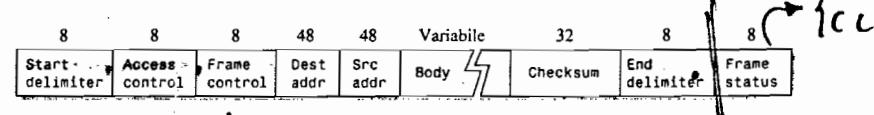


Figura 2.33 Formato del frame in reti token ring 802.5.

Indica il prot. di livello superiore

Come per Ethernet, gli indirizzi 802.5 sono lunghi 48 bit. Lo standard in realtà consente l'utilizzo di indirizzi più brevi, a 16 bit, ma solitamente vengono usati gli indirizzi a 48 bit: in questo caso, l'interpretazione dell'indirizzo avviene esattamente come nel caso di Ethernet. Il frame contiene anche un codice CRC a 32 bit, seguito dal byte di stato del frame (*frame status*), che contiene i bit A e C usati per la consegna affidabile.

2.7.5 FDDI

Per molti aspetti FDDI è simile a 802.5 e a Token Ring di IBM, tuttavia esistono differenze significative, alcune delle quali derivano dal fatto che opera su fibre ottiche e non su cavi in rame, mentre altre sono conseguenza di innovazioni apportate successivamente alla specifica di Token Ring da parte di IBM. Nel seguito evidenzieremo alcune delle differenze più importanti.

Proprietà fisiche

Diversamente dalle reti 802.5, una rete FDDI è composta da due anelli indipendenti che trasmettono dati in direzioni opposte, come illustrato in Figura 2.34(a). Il secondo anello non viene utilizzato durante il funzionamento normale, ma entra in gioco solo nel caso in cui il primo anello si interrompa, come mostrato nella Figura 2.34(b). In questo caso, l'anello si chiude sulla seconda fibra per formare un anello completo e, di conseguenza, una rete FDDI è in grado di tollerare la rottura di una linea di connessione o il guasto di una stazione.

Dato che la configurazione a doppio anello è costosa, FDDI consente ai nodi di connettersi alla rete mediante un'unica fibra: tali nodi vengono chiamati *stazioni a connessione singola* (SAS, single attachment stations), mentre le loro controparti a connessione doppia vengono chiamate, senza sorpresa, *stazioni a connessione doppia* (DAS, dual attachment stations). Per connettere diverse SAS al doppio anello si usa un concentratore, come mostrato in Figura 2.35. Notate come la connessione singola (composta da due fibre) che entra in una SAS sia una sezione dell'anello: se tale SAS dovesse avere un guasto, il concentratore rivelerebbe tale situazione e userebbe un *bypass ottico* per isolare la SAS guasta, garantendo la connessione dell'anello in modo analogo ai relè usati all'interno delle MSAU negli anelli 802.5. Notate che in questa figura il secondo anello (di backup, usato solo in caso di guasto) è indicato con una linea tratteggiata.

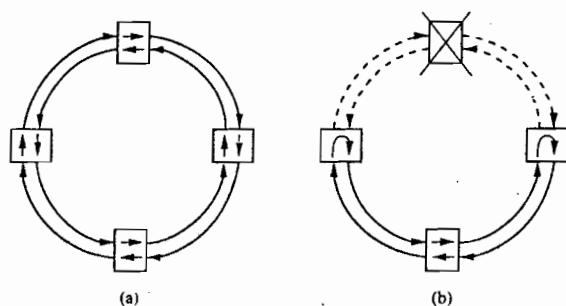


Figura 2.34 Doppio anello in fibra: (a) funzionamento normale; (b) guasto nell'anello primario.

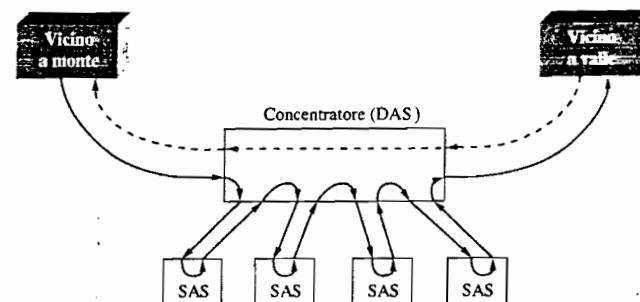


Figura 2.35 SAS connesse ad un concentratore.

Come nel caso di 802.5, ciascun adattatore di rete memorizza un certo numero di bit fra le proprie interfacce di ingresso e di uscita. Diversamente da 802.5, però, il buffer può avere dimensioni diverse nelle varie stazioni, con un minimo di 9 bit ed un massimo di 80, ed è anche possibile che una stazione inizi a trasmettere prima che il proprio buffer sia pieno. Il tempo totale necessario ad un testimone per attraversare l'intera rete è, ovviamente, una funzione della dimensione di questi buffer. Ad esempio, dato che FDDI è una rete a 100 Mbps, il suo tempo di bit è 10 nanosecondi (ns), cioè ciascun bit ha un'ampiezza di 10 ns. Se ciascuna stazione ha un buffer a 10 bit ed inizia a trasmettere quando il proprio buffer è pieno a metà, allora ogni stazione introduce nel tempo totale di rotazione del testimone un ritardo pari a $5 \times 10 \text{ ns} = 50 \text{ ns}$.

FDDI ha, poi, altre caratteristiche fisiche. Ad esempio, lo standard limita una singola rete ad avere al massimo 500 stazioni (host), con una distanza massima di 2 km tra una coppia di stazioni. La rete può avere, poi, una dimensione massima di 200 km di fibre, per cui, vista la struttura a doppio anello, la lunghezza totale dei cavi che connettono tutte le stazioni può essere al massimo 100 km. Ancora, nonostante la lettera "F" nella sigla FDDI implichi che come mezzo fisico sottostante venga usata la fibra ottica, lo standard è stato definito per poter operare su diversi mezzi fisici, compresi i cavi coassiali e le coppie in doppino: anche in questi casi occorre, ovviamente, fare attenzione alla distanza totale coperta dall'anello. Come vedremo in seguito, il tempo necessario perché il testimone attraversi l'intera rete gioca un ruolo importante nell'algoritmo di controllo d'accesso.

FDDI usa la codifica 4B/5B, vista nella Sezione 2.2. Dato che FDDI è stata la prima tecnologia per reti in fibra ad avere ampia diffusione e poiché processori per la codifica 4B/5B operanti alla velocità di FDDI sono diventati largamente disponibili, tale schema di codifica è diventata assai comune nelle fibre in generale.

Algoritmo a testimone temporizzato

Le regole che governano i tempi di possesso del testimone in FDDI sono un po' più complesse di quelle di 802.5. Il THT di ciascun nodo viene definito nello stesso modo e lo si configura ad un valore adatto. Inoltre, per garantire che un nodo abbia l'opportunità di trasmettere entro un certo tempo (cioè per porre un limite superiore al TRT osservato da un nodo qualsiasi), si definisce un *tempo obiettivo per la rotazione del testimone* (TTRT, target token rotation time) e tutti i nodi si accordano per soddisfare questo valore limite TTTRT (nella successiva sottosezione vedremo come venga raggiunto, tra i nodi, l'accordo sul valore di TTTRT). Nello

specifico, ciascun nodo misura il tempo che intercorre tra successivi arrivi del testimone: è questo il TRT *misurato* dal nodo. Se tale TRT misurato è maggiore del TTRT concordato, allora il testimone è in ritardo e il nodo non trasmette dati. Se, invece, il TRT misurato è inferiore a TTRT, allora il testimone è in anticipo e il nodo può impossessarsi del testimone per un tempo uguale alla differenza tra TTRT e il TRT misurato.

Anche se può sembrare che abbiamo terminato, l'algoritmo che abbiamo appena sviluppato non garantisce che un nodo che voglia inviare un frame entro un tempo definito sia veramente in grado di farlo. Il problema è che un nodo con molti dati da inviare ha la possibilità, vedendo arrivare il testimone in anticipo, di impossessarsene per così tanto tempo che quando un nodo a valle riceverà il testimone il suo TRT misurato sarà uguale o superiore a TTRT, impedendogli di trasmettere il proprio frame. Per tenere in considerazione questa eventualità, FDDI definisce due categorie di traffico: *sincrono* e *asincrono*³. Quando un nodo riceve il testimone, ha sempre il permesso di inviare dati sincroni, indipendentemente dalla temporizzazione del testimone stesso. Al contrario, un nodo può inviare traffico asincrono soltanto quando il testimone è in anticipo.

Notate che i termini *sincrono* e *asincrono* sono in qualche modo fuorvianti. Con il termine sincrono, FDDI indica il traffico sensibile al ritardo: ad esempio, in una rete FDDI si possono inviare dati video o audio come traffico sincrono. Al contrario, asincrono significa che l'applicazione è più interessata al throughput che al ritardo: un esempio di traffico FDDI asincrono può essere un'applicazione di trasferimento di file.

Abbiamo finito, ora? Non proprio. Dato che il traffico sincrono può essere trasmesso indipendentemente dal fatto che il testimone sia in anticipo o in ritardo, sembrerebbe che se un nodo ha un'elevata quantità di dati sincroni da trasmettere il tempo-obiettivo per la rotazione del testimone sia nuovamente privo di significato. Per tener conto di questo problema, anche il tempo totale destinato a dati sincroni che possono essere inviati durante una rotazione del testimone è limitato superiormente al valore TTRT: di conseguenza, nel caso peggiore i nodi con traffico asincrono possono trasmettere per primi per un periodo massimo uguale a TTRT, poi altrettanto possono fare i nodi con traffico sincrono, per cui il TRT misurato da un nodo può diventare, al massimo, $2 \times$ TTRT. Notate che se il traffico sincrono ha già utilizzato il periodo di tempo uguale a TTRT ad esso dedicato i nodi con traffico asincrono non invieranno dati, perché il testimone arriverà in ritardo, per cui, mentre è possibile che una rotazione del testimone impieghi un tempo massimo di $2 \times$ TTRT, non è possibile che si susseguano due rotazioni consecutive che impieghino ciascuna un tempo uguale a $2 \times$ TTRT.

Un ultimo dettaglio è relativo al modo in cui un nodo può inviare traffico asincrono. Come già detto, un nodo può trasmettere se il TRT misurato è inferiore a TTRT. La domanda che ci si pone è: cosa succede se il TRT misurato è minore di TTRT, ma di una quantità così piccola da non rendere possibile la trasmissione dell'intero messaggio senza andare oltre il periodo TTRT? La risposta è che, in tal caso, al nodo è consentita la trasmissione. Di conseguenza, il TRT misurato è, in realtà, limitato superiormente da TTRT più il tempo che serve per trasmettere un intero frame FDDI.

³ In origine FDDI definì due sottocategorie di traffico asincrono, *limitato* e *illimitato*, ma in pratica il caso asincrono limitato non è implementato, per cui descriviamo soltanto il caso illimitato e ci riferiamo ad esso, semplicemente, chiamandolo "asincrono".

Manutenzione del testimone

I meccanismi usati da FDDI per garantire che ci sia sempre un testimone valido in circolazione lungo l'anello sono diversi da quelli di 802.5, perché sono connessi al procedimento che determina TTRT. Innanzitutto, tutti i nodi lungo l'anello FDDI tengono sotto controllo l'anello stesso, per assicurarsi che il testimone non sia andato perduto. In un anello funzionante correttamente ciascun nodo dovrebbe osservare, di tanto in tanto, una trasmissione valida: o un frame di dati o il testimone. Il più lungo intervallo di tempo di inattività fra due trasmissioni valide che può essere osservato da un nodo è uguale alla latenza dell'anello più il tempo necessario a trasmettere un intero frame, che in un anello di dimensioni massime è un po' meno di 2.5 ms. Quindi, ciascun nodo usa un temporizzatore di eventi che scade dopo 2.5 ms. Se questo conteggio di tempo scade, il nodo può avere il sospetto che qualcosa sia andato storto e trasmette un frame di "reclamo" (*claim*); ogni volta che, invece, riceve una trasmissione valida, il nodo imposta nuovamente il temporizzatore a 2.5 ms.

I frame di reclamo di FDDI sono diversi da quelli di 802.5 perché contengono l'*offerta* del nodo per il TTRT, cioè il tempo di rotazione del testimone di cui il nodo ha bisogno in modo che le applicazioni in esecuzione nel nodo possano soddisfare i propri vincoli temporali. Un nodo può inviare un frame di reclamo anche senza impossessarsi del testimone e tipicamente fa proprio così ogni volta che ha il sospetto che ci sia un guasto, oppure quando entra a far parte della rete per la prima volta. Se tale frame di reclamo compie l'intero giro dell'anello, chi lo ha inviato lo rimuove, sapendo così che la propria offerta di TTRT è quella minima. Tale nodo ha ora il possesso del testimone (cioè ha la responsabilità di inserire nell'anello un testimone valido) e può procedere con l'algoritmo normale.

Quando un nodo riceve un frame di reclamo, controlla se l'offerta di TTRT ivi contenuta è minore della propria, nel qual caso il nodo imposta il valore locale di TTRT a quello contenuto nel frame di reclamo e inoltre il frame al nodo successivo. Se l'offerta di TTRT è superiore al valore di TTRT minimo richiesto dal nodo, allora il frame di reclamo viene rimosso dall'anello e il nodo inizia il processo di offerta inserendo il proprio frame di reclamo sull'anello. Se l'offerta di TTRT dovesse essere uguale a quella necessaria al nodo, il nodo confronta l'indirizzo del mittente del frame di reclamo con il proprio e la vittoria va all'indirizzo più elevato. Di conseguenza, se un frame di reclamo compie un giro intero e ritorna alla propria sorgente, il nodo che l'ha inviato sa di essere l'unico offerente attivo e può reclamare il testimone senza problemi. Al tempo stesso, tutti i nodi hanno ora raggiunto un accordo sul valore di TTRT, che sarà sufficientemente piccolo da accontentare tutti.

Formato del frame

Il formato del frame FDDI, illustrato in Figura 2.36, differisce assai poco da quello di 802.5. Dato che FDDI usa la codifica 4B/5B invece della codifica Manchester, come marcatori di inizio e di fine frame utilizza simboli di controllo di 4B/5B anziché parole di codice Manchester illecite. Altre differenze significative sono la presenza di un bit nell'intestazione per distinguere il traffico sincrono da quello asincrono e la mancanza dei bit per il controllo d'accesso, presenti invece nel frame 802.5.

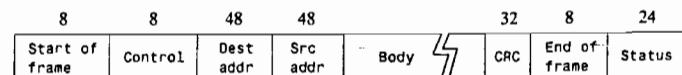


Figura 2.36 Formato del frame FDDI.

2.8 Wireless (802.11)

Le reti senza fili (*wireless*) sono una tecnologia in rapida evoluzione per la connessione di calcolatori. Come abbiamo visto in precedenza in questo stesso capitolo, le possibilità per costruire reti wireless sono praticamente infinite, spaziando dall'uso di segnali infrarossi all'interno di un singolo edificio fino alla costruzione di una rete mondiale mediante una griglia di satelliti in orbita bassa. Questa sezione dà uno sguardo più da vicino ad una specifica tecnologia, incentrata sullo standard emergente IEEE 802.11, che, come i suoi simili, Ethernet e token ring, è stato progettato per un uso in un'area geograficamente limitata (case, edifici adibiti ad ufficio, campus universitari) ed il suo obiettivo principale è quello di mediare l'accesso ad un mezzo di comunicazione condiviso, che in questo caso è costituito da segnali che si propagano nello spazio. 802.11 mette a disposizione alcune caratteristiche ulteriori (come, ad esempio, servizi a ritardo limitato, gestione del consumo di potenza e meccanismi per la sicurezza), ma noi focalizzeremo la discussione sulle sue funzionalità di base.

2.8.1 Proprietà fisiche

802.11 è stato progettato per essere eseguito su tre diversi mezzi fisici, due basati su segnali radio a spettro disperso (*spread spectrum*) e uno su segnali infrarossi diffusi. Le versioni radio operano attualmente a 11 Mbps, ma saranno presto operative a 54 Mbps.

L'idea che sta alla base dello spettro disperso è la dispersione del segnale su una banda di frequenza più ampia del normale, in modo da minimizzare l'impatto dell'interferenza provocata da altri dispositivi (la tecnologia a spettro disperso fu originariamente progettata per scopi militari, per cui questi "altri dispositivi" avevano spesso lo scopo di disturbare appositamente il segnale). Ad esempio, il *salto di frequenza* (*frequency hopping*) è una tecnica a spettro disperso che prevede la trasmissione del segnale su una sequenza casuale di frequenze, trasmettendo, cioè, prima con una certa frequenza, poi con una seconda, poi con una terza, e così via. La sequenza di frequenze non è veramente casuale, ma viene invece calcolata da un algoritmo basato su un generatore di numeri pseudocasuali. Il ricevitore usa lo stesso algoritmo della sorgente e lo inizializza con lo stesso seme, per cui è in grado di effettuare i salti di frequenza in sincronia con il trasmettitore, per ricevere correttamente il frame.

Una seconda tecnica a spettro disperso, chiamata *sequenza diretta* (*direct sequence*), ottiene lo stesso risultato rappresentando ciascun bit del frame con più bit nel segnale trasmesso. Per ciascun bit che deve essere trasmesso dalla sorgente viene in realtà inviato l'OR esclusivo di tale bit con n bit casuali. Come nel salto di frequenza, la sequenza di bit casuali viene generata mediante un generatore di numeri pseudocasuali noto sia al trasmettitore che al ricevitore. I valori trasmessi, noti come *codice chipping a n bit*, disperdon il segnale su una banda di frequenza che è n volte più ampia di quella che sarebbe stata richiesta dal frame. La Figura 2.37 fornisce un esempio di una sequenza di chipping a 4 bit.

802.11 definisce un primo strato fisico che usa il salto di frequenza (su 79 bande di frequenza, ciascuna con ampiezza di 1 MHz) ed un secondo che usa la sequenza diretta (con sequenza di chipping di 11 bit). Entrambi gli standard operano nella banda di frequenza intorno a 2.4 GHz dello spettro elettromagnetico e in entrambi i casi lo spettro disperso ha anche l'interessante caratteristica di far apparire il segnale simile a rumore a qualsiasi ricevitore che non conosca la sequenza pseudocasuale.

2.8 Wireless (802.11)

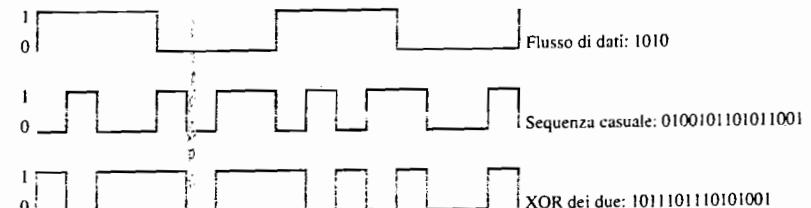


Figura 2.37 Esempio di sequenza di chipping a 4 bit.

Il terzo standard a livello fisico di 802.11 è basato su segnali infrarossi: la trasmissione è diffusa, per cui non è necessario che sorgente e ricevitore siano in linea di vista l'uno con l'altro. Questa tecnologia ha un raggio d'azione di circa 10 m ed è limitata ad un utilizzo all'interno di edifici.

2.8.2 Evitare le collisioni

A prima vista, si potrebbe avere l'impressione che un protocollo wireless debba seguire esattamente lo stesso algoritmo di Ethernet, aspettare che il mezzo fisico risulti inattivo prima di trasmettere e riprovare in caso di collisione, ed in prima approssimazione ciò è proprio quello che fa 802.11. Tuttavia, in una rete senza fili il problema è ulteriormente complicato, perché non tutti i nodi sono sempre raggiungibili da tutti gli altri.

Considerate, infatti, la situazione illustrata nella Figura 2.38, dove ciascuno dei quattro nodi è in grado di inviare e ricevere segnali che raggiungano i nodi ad essi immediatamente adiacenti, a destra e a sinistra. Ad esempio, B può scambiare frame con A e con C ma non può raggiungere D, mentre C può raggiungere B e D ma non A (i nodi raggiungibili da A e D non sono indicati in figura). Supponete che sia A sia C vogliano comunicare con B e che entrambi gli trasmettano un frame. A e C non sono consapevoli della reciproca presenza, poiché i loro segnali non arrivano a tale distanza. Questi due frame collidono l'uno con l'altro in B ma, diversamente da quanto accade in Ethernet, né A né C si accorgono di questa collisione: si dice che A e C sono *nodi nascosti* l'uno all'altro.

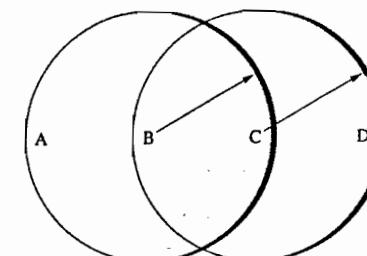


Figura 2.38 Esempio di rete wireless.

Un problema correlato, denominato *problema del nodo esposto*, si verifica nelle seguenti circostanze. Supponete che B stia trasmettendo verso A, nella Figura 2.38. Il nodo C è consapevole di tale comunicazione, perché ascolta la trasmissione di B, ma per C sarebbe un errore decidere di non trasmettere a nessuno solo perché sta assistendo alla trasmissione di B. Ad esempio, se C volesse trasmettere verso D, questo non sarebbe un problema, perché la trasmissione di C verso D non interferirebbe con la capacità di A di ricevere il segnale proveniente da B (interferirebbe, invece, con una trasmissione da A a B, ma nel nostro esempio è B che sta trasmettendo).

802.11 risolve questi due problemi con un algoritmo chiamato *accesso multiplo senza collisioni* (MACA, Multiple Access with Collision Avoidance). L'idea è che la sorgente e la destinazione si scambino frame di controllo l'uno l'altro prima che la sorgente trasmetta realmente i dati: questo scambio informa tutti i nodi vicini che sta per iniziare una trasmissione. Nello specifico, la sorgente trasmette al destinatario un frame di tipo *richiesta di invio* (RTS, Request to Send), che contiene un campo che indica per quanto tempo la sorgente ha intenzione di impegnare il mezzo fisico (cioè specifica la lunghezza del frame di dati che verrà trasmesso). Il ricevitore risponde, quindi, con un frame di tipo *libero di inviare* (CTS, Clear to Send): in tale frame viene ricopiatà la lunghezza dichiarata dalla sorgente. Un nodo che veda il frame CTS sa di essere vicino al ricevitore, per cui non potrà trasmettere per il periodo di tempo richiesto per inviare un frame della lunghezza specificata. Un nodo che veda il frame RTS ma non il frame CTS non è abbastanza vicino al ricevitore per interferire, per cui può trasmettere liberamente.

Per completare il quadro ci sono ancora due dettagli. Per prima cosa, il ricevitore invia un ACK alla sorgente dopo aver ricevuto con successo un frame e tutti i nodi devono attendere tale ACK prima di provare a trasmettere⁴. Secondo, se due o più nodi dovessero trovare il mezzo inattivo e provassero a trasmettere un frame RTS nello stesso istante, i loro frame RTS colliderebbero l'uno con l'altro. 802.11 non consente la rilevazione di collisione, ma le sorgenti capiscono che è avvenuta una collisione quando non ricevono il frame CTS dopo un certo periodo di tempo, nel qual caso ciascuna di esse aspetta un intervallo di tempo casuale prima di riprovare. La durata dell'attesa di un nodo è definita dallo stesso algoritmo di backoff esponenziale usato da Ethernet (si veda la Sezione 2.6.2).

2.8.3 Sistema di distribuzione

Come descritto fin qui, 802.11 sarebbe adatto per una configurazione ad hoc di nodi che possono essere in grado di comunicare con tutti gli altri nodi oppure no, in relazione a quanto ne sono distanti. Inoltre, poiché uno dei vantaggi di una rete wireless è che i nodi sono liberi di spostarsi, non essendo trattenuti da cavi, l'insieme di nodi direttamente raggiungibili può cambiare nel tempo. Per aiutare nella gestione di questa mobilità e parziale connessione, 802.11 definisce alcune strutture addizionali in un insieme di nodi. I nodi sono liberi di comunicare direttamente l'uno con l'altro nel modo appena descritto, ma, in pratica, essi operano invece all'interno di queste strutture.

Invece di avere nodi tutti uguali nella rete, alcuni nodi possono spostarsi (ad esempio, il vostro computer laptop) mentre altri nodi sono connessi ad una infrastruttura di rete cablata.

⁴ Questo ACK non faceva parte dell'algoritmo MACA originale, ma venne proposto in una versione estesa chiamata MACAW: MACA per reti locali Wireless.

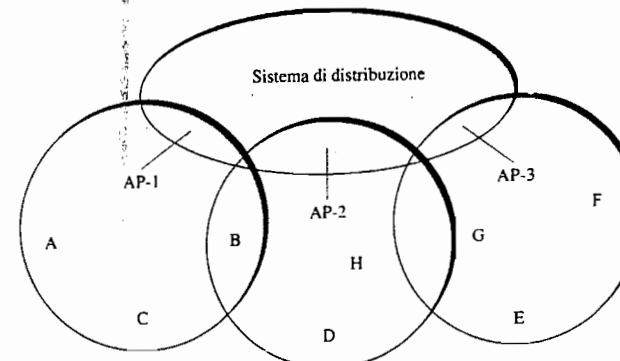


Figura 2.39 Punti di accesso connessi ad una rete di distribuzione.

Questi ultimi vengono chiamati *punti di accesso* (AP, access point) e sono connessi l'uno all'altro mediante ciò che viene chiamato sistema di distribuzione. La Figura 2.39 mostra un sistema di distribuzione che connette tre punti di accesso, ciascuno dei quali serve i nodi in una regione. Ciascuna di queste regioni è analoga ad una cella in un sistema di telefonia cellulare, con i punti di accesso che svolgono lo stesso ruolo di una stazione base. I dettagli del sistema di distribuzione non sono importanti in questa discussione: ad esempio, non ci interessa che si tratti di una Ethernet o di una rete di tipo token ring. L'unico aspetto importante è che la rete di distribuzione operi all'interno dello strato 2 dell'architettura OSI, cioè che non dipenda da alcun protocollo di livello più elevato.

Anche se due nodi sufficientemente vicini da essere raggiungibili tra loro possono comunicare direttamente l'uno con l'altro, l'idea che sta alla base di questa configurazione è che ciascun nodo si associi ad un punto di accesso. Perché il nodo A possa comunicare con il nodo E, ad esempio, A invia per prima cosa un frame al proprio punto di accesso (AP-1), il quale inoltra il frame attraverso il sistema di distribuzione ad AP-3, il quale finalmente trasmette il frame a E. Come AP-1 sappia di dover inoltrare il messaggio ad AP-3 è un problema che non riguarda lo standard 802.11: può darsi, ad esempio, che venga usato il protocollo di bridging descritto nel prossimo capitolo (Sezione 3.2). Ciò che viene specificato da 802.11 è il modo in cui i nodi selezionano il proprio punto di accesso e, cosa ancora più interessante, come funziona tale algoritmo in presenza di nodi che si spostano da una cella ad un'altra.

La tecnica per la selezione di un punto di accesso viene chiamata *scansione* (scanning) e richiede i quattro passi seguenti:

1. Il nodo invia un frame di tipo Probe.
2. Tutti i punti di accesso raggiungibili rispondono con un frame di tipo Probe Response.
3. Il nodo seleziona uno dei punti di accesso e gli invia un frame di tipo Association Request.
4. Il punto di accesso risponde con un frame di tipo Association Response.

Un nodo utilizza questo protocollo ogni volta che si unisce ad una rete, così come quando non è più soddisfatto del proprio punto di accesso. Ciò può accadere, ad esempio, perché il

segnale che proviene dal punto di accesso attuale si è indebolito perché il nodo si è allontanato da esso. Ogni volta che un nodo si associa ad un nuovo punto di accesso, tale AP segnala il cambiamento al punto di accesso precedente (durante l'esecuzione del passo 4) tramite il sistema di distribuzione.

Considerate la situazione mostrata nella Figura 2.40, dove il nodo C si sposta dalla cella servita da AP-1 verso la cella servita da AP-2. Mentre si sposta invia il frame Probe, che sollecita il frame Probe Response da parte di AP-2. Ad un certo punto C preferirà AP-2 rispetto ad AP-1 e si assocerà a tale punto di accesso.

Il meccanismo appena descritto viene definito *scansione attiva*, perché il nodo ricerca attivamente un punto di accesso. I punti di accesso inviano periodicamente un frame di tipo Beacon che pubblicizza le caratteristiche del punto di accesso stesso, tra le quali figurano la velocità di trasmissione consentita dall'AP. Questa modalità viene chiamata *scansione passiva* e un nodo può cambiare il proprio AP in base al frame Beacon ricevuto, rispondendo semplicemente con un frame Association Request al punto di accesso.

2.8.4 Formato del frame

La maggior parte del formato del frame 802.11, rappresentato in Figura 2.41, è proprio come ce lo aspetteremmo. Il frame contiene gli indirizzi dei nodi sorgente e destinazione (ciascuno dei quali è lungo 48 bit), fino a 2312 byte di dati e un codice CRC a 32 bit. Il campo Control contiene tre interessanti sottocampi (non evidenziati): un campo Type a 6 bit che indica se il frame trasporta dati, se è un frame RTS o CTS, oppure se è usato dall'algoritmo di scansione, nonché un paio di campi di un solo bit, chiamati ToDS e FromDS, che descriveremo in seguito.

La caratteristica peculiare del formato del frame 802.11 è che contiene quattro indirizzi, invece dei due solitamente presenti: come questi indirizzi vengano interpretati dipende dai

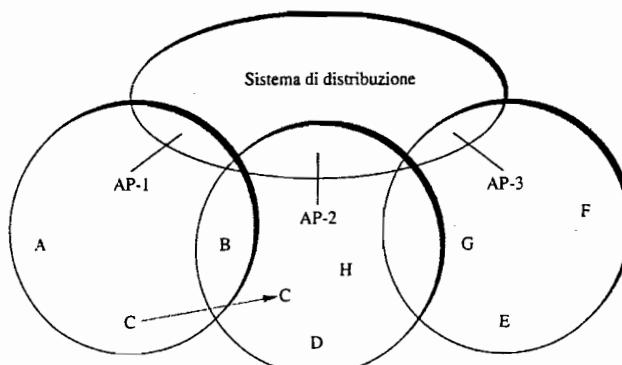


Figura 2.40 Mobilità dei nodi.

16	16	48	48	48	16	48	0-18496	32
Control	Duration	Addr1	Addr2	Addr3	SeqCtrl	Addr4	Payload	CRC

Figura 2.41 Formato del frame 802.11.

2.9 Adattatori di rete

valori dei bit ToDS e FromDS nel campo Control del frame e servono per gestire la possibilità che il frame debba essere inoltrato attraverso il sistema di distribuzione e, di conseguenza, ciò che il mittente originario non è necessariamente lo stesso nodo che ha trasmesso il frame più di recente. Un ragionamento simile è, ovviamente, applicabile anche all'indirizzo di destinazione. Nel caso più semplice, quando un nodo sta trasmettendo ad un altro direttamente, entrambi i bit DS valgono 0, Addr1 identifica il nodo di destinazione e Addr2 identifica il nodo sorgente. Nel caso più complesso, entrambi i bit DS valgono 1, indicando con ciò che il messaggio è stato trasmesso da un nodo wireless al sistema di distribuzione, e poi dal sistema di distribuzione ad un altro nodo wireless. Con entrambi i bit al valore 1, Addr1 identifica la destinazione finale, Addr2 è l'indirizzo del trasmettitore attuale (quello che ha inoltrato il frame dal sistema di distribuzione alla destinazione finale), Addr3 identifica la destinazione intermedia (quella che accetta il frame da un nodo wireless e lo inoltra attraverso il sistema di distribuzione) e Addr4 è l'indirizzo della sorgente originaria. Nell'esempio di Figura 2.39, Addr1 corrisponde a E, Addr2 identifica AP-3, Addr3 corrisponde ad AP-1 e Addr4 identifica A.

2.9 Adattatori di rete

Praticamente tutte le funzionalità di rete descritte in questo capitolo sono realizzate dall'adattatore di rete: framing, rilevazione d'errore e protocollo di accesso al mezzo. Le sole eccezioni sono gli schemi punto-punto di richiesta automatica di ripetizione (ARQ) descritti nella Sezione 2.5, che vengono tipicamente implementati nel protocollo di livello più basso tra quelli in esecuzione sull'host. Concludiamo questo capitolo descrivendo il progetto di un generico adattatore di rete e del software (*device driver*) che lo controlla e lo rende operativo.

Leggendo questa sezione, ricordatevi che non esistono due adattatori di rete esattamente uguali: sono tutti diversi in piccoli, infiniti dettagli. Di conseguenza, ci concentreremo sulle loro caratteristiche generali, sebbene presenteremo alcuni esempi relativi ad un adattatore reale per rendere più tangibile la discussione.

2.9.1 Componenti

Un adattatore di rete serve come interfaccia tra l'host e la rete e, di conseguenza, si può pensare che sia costituito da due componenti principali: un'interfaccia al bus che sappia comunicare con l'host ed un'interfaccia di linea che parli il protocollo corretto sulla rete. Deve anche esistere un percorso di comunicazione tra questi due componenti, lungo il quale vengono trasferiti i dati in entrata e in uscita. La Figura 2.42 rappresenta un semplice schema a blocchi di un adattatore di rete.

Gli adattatori di rete sono sempre progettati per uno specifico bus di I/O (Input/Output), impedendone di fatto il trasferimento dalla macchina di un produttore a quella di un produttore diverso⁵. Ciascun bus definisce un protocollo che deve essere utilizzato dalla CPU dell'host per programmare l'adattatore per inviare interruzioni alla CPU dell'host e, di nuovo, dall'adattatore per leggere e scrivere la memoria dell'host. Una delle caratteristiche

⁵ Fortunatamente, nel campo della progettazione dei bus esistono degli standard, come nelle reti, per cui alcuni adattatori possono essere utilizzati in macchine di diversi produttori.

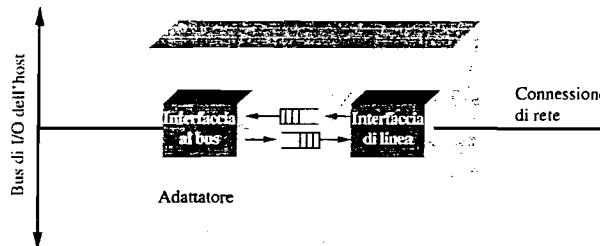


Figura 2.42 Schema a blocchi di un tipico adattatore di rete.

principali di un bus di I/O è la velocità di trasferimento di dati consentita: ad esempio, un bus tipico potrebbe avere un percorso per i dati (*data path*) a 32 bit, potendo così trasferire 32 bit di dati in parallelo, ed operare a 33 MHz, cioè con un tempo di ciclo di 33 ns, dando luogo ad una velocità di trasferimento di picco di circa 1 Gbps, che sarebbe sufficiente ad implementare un collegamento (unidirezionale) STS-12 a 622 Mbps. Come è ovvio, la velocità di picco non ci dà quasi nessuna informazione relativamente alla velocità media, che potrebbe essere molto inferiore.

La parte dell'adattatore che si connette alla linea di collegamento implementa il protocollo di linea. Per tecnologie sufficientemente mature come Ethernet, tale componente dell'adattatore è realizzata da un insieme di chip (*chip set*) che si può acquistare in un negozio di computer, ma per tecnologie di linea più nuove il protocollo relativo può essere implementato in software su un microprocessore ad utilizzo generale o, forse, in qualche forma di hardware programmatibile, come un FPGA (field-programmable gate array). Questi approcci generalmente aumentano il costo dell'adattatore ma lo rendono più flessibile, dato che è più semplice modificare il software dell'hardware ed è più facile riprogrammare un FPGA piuttosto che riprogettare una scheda.

Dato che, con grande probabilità, il bus dell'host e la linea di connessione alla rete operano a velocità diverse, c'è bisogno di interporre una piccola quantità di buffer di memorizzazione fra le due parti dell'adattatore: solitamente una piccola FIFO (una coda di byte) è sufficiente per mascherare la mancanza di sincronismo fra il bus e la linea.

2.9.2 Punto di vista dell'host

Dato che per la maggior parte del capitolo abbiamo parlato dei vari protocolli che sono implementati nella parte di adattatore che si connette alla linea di collegamento, volgiamo ora la nostra attenzione all'adattatore di rete come esso viene visto dall'host.

Registro per il controllo dello stato

Un adattatore di rete, come ogni altro dispositivo, viene sostanzialmente programmato da software eseguito sulla CPU. Dal punto di vista della CPU, l'adattatore rende visibile un *registro per il controllo dello stato* (CSR, control status register) che può essere letto e scritto dalla CPU stessa. Solitamente il CSR è accessibile mediante un indirizzo della memoria, in modo che la CPU vi possa accedere esattamente come farebbe con una normale locazione di memoria: la CPU scrive nel CSR per dargli istruzioni per la trasmissione e/o per la ricezione di un frame, mentre legge il CSR per consultare lo stato attuale dell'adattatore.

Quanto segue è un esempio di CSR del dispositivo Ethernet Lance, prodotto da Advanced Micro Devices (AMD). Il dispositivo Lance ha in realtà quattro diversi registri per il controllo dello stato, ma nel seguito mostreremo soltanto le maschere di bit usate per interpretare il significato del registro CSRO a 16 bit. Per assegnare un valore ad un bit nell'adattatore, la CPU esegue un OR inclusivo di CSRO e della maschera corrispondente al bit che vuole impostare. Per determinare il valore di un particolare bit, la CPU confronta con lo zero il risultato dell'esecuzione di un AND fra il contenuto di CSRO e la maschera.

```
/*
 * Bit di controllo e di stato nel registro CSRO.
 *
 * Legenda:
 * RO - Read Only, sola lettura
 * RC - Read/Clear, lettura/cancellazione
 * (scrivendo 1 si cancella, scrivendo 0 non succede niente)
 * RW - Read/Write, lettura/scrittura
 * W1 - Write-1-only, scrittura soltanto di valore 1
 * (scrivend 1 si scrive, scrivendo 0 non succede niente)
 * RW1 - Read/Write-1-only
 * (scrivendo 1 si scrive, scrivendo 0 non succede niente)
 */
#define LE_ERR 0x8000 /* RO BABL ; CERR ; MISS ; MERR */
#define LE_BABL 0x4000 /* RC troppi bit trasmessi */
#define LE_CERR 0x2000 /* RC nessuna attività */
#define LE_MISS 0x1000 /* RC perduto un pacchetto entrante */
#define LE_MERR 0x0800 /* RC errore in memoria: nessun ACK */
#define LE_RINT 0x0400 /* RC pacchetto ricevuto: interrompi */
#define LE_TINT 0x0200 /* RC pacchetto trasmesso: interrompi */
#define LE_IDON 0x0100 /* RC inizializzazione terminata */
#define LE_INTR 0x0080 /* RO BABL;MISS;MERR;RINT;TINT;IDON */
#define LE_INEA 0x0040 /* RW abilita interruzioni */
#define LE_RXON 0x0020 /* RO ricevitore acceso */
#define LE_TXON 0x0010 /* RO trasmettitore acceso */
#define LE_TMD 0x0008 /* W1 richiesta di trasmissione (invialo ora) */
#define LE_STOP 0x0004 /* RW1 stop */
#define LE_STRT 0x0002 /* RW1 start */
#define LE_INIT 0x0001 /* RW1 inizializza */
```

Questa definizione dice, ad esempio, che l'host scrive un valore 1 nel bit meno significativo di CSRO (0x0001) per inizializzare il chip Lance. Analogamente, se l'host osserva un valore 1 nel sesto bit (0x0020) e nel quinto bit (0x0010), significa che il chip Lance è abilitato, rispettivamente, a ricevere e a trasmettere frame.

Interruzioni

La CPU dell'host potrebbe eseguire un ciclo ad altissima velocità, leggendo continuamente il registro di controllo dello stato dell'adattatore finché non accade qualcosa di interessante, per poi intraprendere l'azione appropriata. Nel chip Lance, ad esempio, potrebbe osservare con-

tinuamente l'undicesimo bit (`0x0400`) finché non vede un valore 1, che sta ad indicare che è appena arrivato un frame. Questa attività viene chiamata *polling* e, nonostante in alcune situazioni non sia un comportamento irragionevole (come potrebbe essere, ad esempio, il caso di un router di rete che non abbia altro da fare che aspettare il successivo frame), solitamente non viene usata sugli host terminali, che potrebbero dedicare il proprio tempo all'esecuzione di programmi applicativi.

Invece di eseguire un polling, la maggior parte degli host rivolge la propria attenzione ai dispositivi di rete soltanto quando sono questi ultimi ad interrompere l'esecuzione normale dell'host. Il dispositivo lancia un'interruzione (*interrupt*) quando accade un evento che richiede l'intervento diretto dell'host, come, ad esempio, la corretta trasmissione o ricezione di un frame, oppure l'insorgere di un errore mentre il dispositivo stava tentando di trasmettere o di ricevere un frame. L'architettura dell'host è dotata di un meccanismo che provoca l'esecuzione di una particolare procedura del sistema operativo ogniqualvolta si verifica una tale interruzione: la procedura viene chiamata *gestore di interruzione* (*interrupt handler*) ed esamina il CSR per determinare la causa dell'interruzione, per poi agire di conseguenza.

Mentre sta rispondendo ad una interruzione, solitamente l'host *disabilita* ulteriori interruzioni, evitando che il software di gestione del dispositivo (device driver) debba rispondere a più interruzioni per volta. Dato che le interruzioni sono disabilitate, il driver del dispositivo deve portare a termine il proprio compito velocemente (senza, quindi, avere il tempo di eseguire l'intera pila di protocolli) e non deve bloccarsi (cioè non deve sospendere la propria esecuzione in attesa di un evento) per alcun motivo. Ad esempio, questo obiettivo potrebbe essere raggiunto facendo in modo che il gestore di interruzioni metta in esecuzione un processo per gestire il frame, terminando poi la propria esecuzione. In questo modo il gestore è certo che il frame verrà elaborato, senza dover perdere tempo prezioso per la effettiva elaborazione del frame stesso.

Accesso diretto alla memoria (DMA) e I/O programmato

Uno degli aspetti più importanti che occorre considerare durante la progettazione di un adattatore di rete è la modalità con cui i byte che costituiscono un frame vengono trasferiti dall'adattatore alla memoria dell'host, e viceversa. Esistono fondamentalmente due meccanismi: *accesso diretto alla memoria* (DMA, direct memory access) e *I/O programmato* (PIO, programmed I/O). Con la modalità DMA l'adattatore legge e scrive direttamente la memoria dell'host, senza l'intervento della CPU: semplicemente, l'host fornisce all'adattatore un indirizzo di memoria dove poter leggere (o scrivere). Con la modalità PIO, al contrario, è la CPU ad essere direttamente responsabile del trasferimento dei dati fra l'adattatore e la memoria dell'host: per inviare un frame, la CPU entra in un ciclo che legge i byte dalla memoria dell'host e li scrive nell'adattatore, mentre per ricevere un frame la CPU legge i byte dall'adattatore e li scrive in memoria. Esamineremo ora DMA e PIO in maggiore dettaglio.

Usando la modalità DMA non c'è bisogno di buffer per i frame all'interno dell'adattatore, in quanto viene usata direttamente la memoria dell'host (rimane la necessità della memorizzazione di pochi byte nell'adattatore per compensare le differenze di velocità tra il bus e la linea di collegamento alla rete, come descritto in precedenza, ma nell'adattatore non vengono memorizzati frame completi). La CPU ha, quindi, il compito di assegnare all'adattatore una coppia di *liste descrittive di buffer*: una per trasmettere e una per ricevere. Una lista descrittiva di buffer è un array di coppie indirizzo/lunghezza, come mostrato in Figura 2.43.

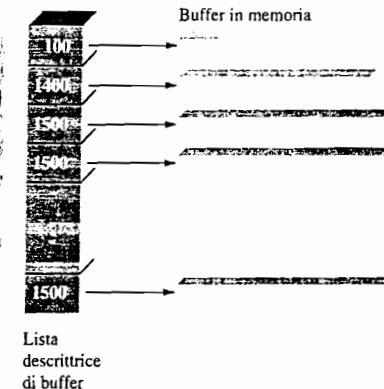


Figura 2.43 Lista descrittrice di buffer.

Quando riceve dei frame, l'adattatore usa i buffer che gli sono necessari per memorizzarli. Ad esempio, usando la lista descrittiva mostrata in Figura 2.43, un adattatore Ethernet che tentasse di ricevere un frame di 1450 byte memorizzerebbe i primi 100 byte nel primo buffer e i successivi 1350 byte nel secondo buffer. Se, subito dopo il primo, arrivasse un secondo frame di 1500 byte, verrebbe posizionato interamente nel terzo buffer: frame diversi vengono memorizzati in buffer separati, mentre un singolo frame può essere suddiviso fra più buffer. Quest'ultima caratteristica viene solitamente denominata *lettura con dispersione* (scatter-read) e viene usata, in particolare, quando la dimensione massima dei frame della rete è così grande che sarebbe uno spreco dimensionare tutti i buffer grandi abbastanza da contenere il più grande frame che può arrivare. Per collegare insieme tutti i buffer che compongono un singolo frame si userebbe, poi, una struttura dati per i messaggi specifica del sistema operativo, in modo simile a quanto visto nella Sezione 1.4.3. In Ethernet, però, solitamente non viene usata la lettura con dispersione, perché dimensionare i buffer a 1500 byte non viene considerato uno spreco eccessivo di memoria.

La gestione dei frame in uscita, anziché in ingresso, funziona in modo simile. Quando l'host ha un frame da trasmettere inserisce nella lista descrittiva dei buffer di trasmissione un puntatore al buffer che contiene il frame. I dispositivi che consentono la *scrittura con raccolta* (gather-write) permettono che il frame sia frammentato su diversi buffer distinti: tale tecnica, in pratica, è usata più frequentemente della lettura con dispersione, perché spesso i frame in uscita vengono assemblati pezzo per pezzo, da diversi protocolli, ciascuno dei quali fornisce un proprio buffer. Ad esempio, mentre un messaggio compie la propria discesa lungo la pila di protocolli, fino ad essere pronto per la trasmissione, è costituito da un buffer che contiene l'intestazione cumulativa (l'insieme delle intestazioni indicate dai vari protocolli che hanno elaborato il messaggio) e da un diverso buffer che contiene i dati dell'applicazione.

Nel caso della modalità PIO, l'adattatore di rete deve avere una certa capacità di memorizzazione mediante buffer, in quanto la CPU copia i frame dalla memoria dell'host a tale memoria dell'adattatore, e viceversa, come mostrato in Figura 2.44. Il motivo fondamentale per cui sono necessari i buffer è che, nella maggioranza dei sistemi operativi, non potete mai avere la certezza che la CPU non sia impegnata in altri compiti, per cui occorre

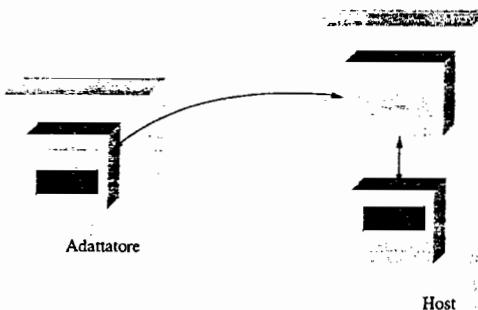


Figura 2.44 I/O programmato.

essere pronti ad aspettare: la domanda importante a cui rispondere è di quanta memoria deve disporre l'adattatore. Certamente ci deve essere almeno lo spazio per un frame, sia nella direzione di trasmissione che in quella di ricezione. Inoltre, gli adattatori che usano la modalità PIO solitamente hanno ulteriore memoria da usare per contenere un limitato numero di frame in arrivo nel caso in cui la CPU ritardi nel trasferirli nella memoria dell'host. Nonostante uno degli assiomi dei sistemi di calcolo sia che "la memoria è poco costosa" e che, quindi, potrebbe sembrare ovvio dotare l'adattatore di un'enorme quantità di memoria, tale memoria deve essere del tipo più costoso, a doppio accesso (dual-port), dato che deve essere letta/scritta sia dalla CPU sia dall'adattatore. Gli adattatori che usano la modalità PIO hanno solitamente una memoria interna di 64-256 KB, anche se esistono addirittura adattatori con 1 MB di memoria.

Frame, buffer e messaggi

Come suggerito da questa sezione, l'adattatore di rete è il punto in cui la rete viene fisicamente a contatto con l'host, ed è anche il punto in cui si intersecano tre mondi diversi: la rete, l'architettura dell'host e il suo sistema operativo. Ciascuno di questi tre mondi ha una propria terminologia per parlare della stessa cosa, per cui è importante capire ciò che accade.

Dal punto di vista della rete, l'adattatore trasmette *frame* uscenti dall'host e riceve *frame* diretti all'host: la maggior parte di questo capitolo è stata presentata dal punto di vista della rete, per cui dovreste avere ben chiaro cosa significhi il termine "frame". Dal punto di vista dell'architettura dell'host, i frame sono ricevuti in un buffer o sono trasmessi da un buffer, che è semplicemente una zona della memoria principale caratterizzata da una certa lunghezza e da un indirizzo iniziale. Infine, dal punto di vista del sistema operativo, un *messaggio* è un'astrazione che rappresenta un frame della rete. I messaggi sono implementati con una struttura dati che contiene i puntatori a diverse locazioni di memoria (buffer), struttura di cui abbiamo visto un esempio nel Capitolo 1.

Driver di dispositivi

Un driver di dispositivo (*device driver*) è un insieme di procedure del sistema operativo che hanno il compito di collegare una pila di protocolli all'hardware di rete e comprende tipicamente funzioni per inizializzare il dispositivo, per trasmettere frame lungo la linea di collegamento alla rete e per lanciare interruzioni alla CPU. Il codice è spesso di difficile lettura, perché è pieno di dettagli specifici del dispositivo, ma la struttura logica globale è spesso piuttosto semplice.

Ad esempio, una procedura di trasmissione si assicura per prima cosa che ci sia sul dispositivo un buffer di trasmissione libero che possa gestire il messaggio. Se non c'è, la procedura deve bloccare il proprio processo finché non se ne libera uno; trovato un buffer di trasmissione disponibile, il processo invocante disabilita le interruzioni per proteggersi da interferenze, quindi traduce il messaggio dal formato interno del sistema operativo al formato richiesto dal dispositivo, impostando il CSR in modo da richiedere la trasmissione, e, infine, riabilita le interruzioni.

La logica che controlla il gestore di interruzioni è altrettanto semplice. Per prima cosa disabilita ulteriori interruzioni che potrebbero interferire con l'elaborazione dell'interruzione in corso, poi esamina il CSR per determinare la causa dell'interruzione. Ci sono tre possibilità: (1) si è verificato un errore; (2) una richiesta di trasmissione è stata portata a termine; (3) è stato ricevuto un frame. Nel primo caso il gestore emette un messaggio e cancella il bit che segnala l'errore. Nel secondo caso sappiamo che una richiesta di trasmissione che era stata precedentemente accodata dalla procedura di trasmissione è stata portata a termine, per cui ora c'è un buffer di trasmissione libero che può essere riutilizzato. Nel terzo caso il gestore invoca la procedura di ricezione che estrae il frame in arrivo dalla lista dei buffer di ricezione e lo inserisce all'interno di una struttura dati del sistema operativo adatta a contenere messaggi; infine, fa partire un processo per inoltrare il messaggio verso l'alto nella pila di protocolli.

2.9.3 Collo di bottiglia nella memoria

Come visto nella Sezione 2.1.1, le prestazioni della memoria dell'host sono spesso il fattore limitante nelle prestazioni di una rete: ciò è vero soprattutto all'interfaccia fra adattatore e host. Per aiutarvi a comprendere questo fatto, consideriamo la Figura 2.45, che mostra l'ampiezza di banda disponibile fra i diversi componenti di un moderno PC. Mentre il bus di I/O è abbastanza veloce da trasferire frame fra l'adattatore di rete e la memoria dell'host con velocità dell'ordine del gigabit, ci sono due potenziali problemi.

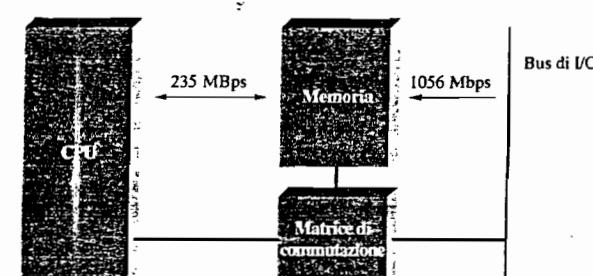


Figura 2.45 Ampiezza di banda della memoria in macchine appartenenti alla categoria dei moderni PC.

Il primo consiste nel fatto che la velocità con cui viene pubblicizzato il bus di I/O corrisponde alla sua ampiezza di banda di picco, cioè è il prodotto tra l'ampiezza del bus e la velocità del segnale di clock (ad esempio, un bus a 32 bit operante con un clock a 33 MHz ha una velocità di trasferimento di picco uguale a 1056 Mbps). Il limite reale è costituito dalla dimensione dei blocchi di dati che possono essere trasferiti dal bus di I/O, poiché ciascun trasferimento richiede una certa quantità di dati aggiuntivi da trasmettere (*overhead*). In alcune architetture, ad esempio, servono 8 cicli di clock per acquisire il bus allo scopo di trasferire dati dall'adattatore alla memoria dell'host e tale overhead è indipendente dal numero di byte di dati trasmessi, per cui se volete trasferire attraverso il bus di I/O un carico utile di 64 byte (che è proprio la dimensione del più piccolo pacchetto Ethernet) l'intero trasferimento richiede 24 cicli: 8 cicli per acquisire il bus e 16 cicli per trasferire i dati (il bus ha un'ampiezza di 32 bit, per cui può trasferire una parola di 4 byte durante ciascun ciclo di clock e 64 byte divisi per 4 byte per ciclo forniscono il risultato di 16 cicli). Ciò significa che la massima ampiezza di banda che si può ottenere è

$$16/(8 + 16) \times 1056 = 704 \text{ Mbps}$$

e non quella di picco di 1056 Mbps.

Il secondo problema consiste nel fatto che l'ampiezza di banda tra memoria e CPU, che è 235 MBps (cioè 1880 Mbps), è dello stesso ordine di grandezza dell'ampiezza di banda del bus di I/O, anche se, fortunatamente, si tratta di un valore misurato e non di una velocità di picco pubblicizzata. Di conseguenza, mentre sarebbe possibile trasferire frame lungo il bus di I/O verso la memoria e poi trasferire i dati dalla memoria ai registri della CPU operando con l'ampiezza di banda della rete, non è pensabile che ciò possa essere realizzato dal driver del dispositivo, in cooperazione con il sistema operativo e l'applicazione, accedendo più volte alla memoria per ciascuna parola di dati contenuta nel pacchetto, perché sono probabilmente necessarie più copiate da un buffer ad un altro. In particolare, se il percorso che va dalla memoria alla CPU viene attraversato n volte, allora molto probabilmente l'ampiezza di banda osservata dall'applicazione sarà $235 \text{ MBps}/n$ (le prestazioni potrebbero essere migliori nel caso in cui i dati vengano conservati in una *memoria cache*, ma spesso le memorie cache non sono di grande aiuto per i dati in arrivo dalla rete). Ad esempio, se i diversi strati software devono copiare i dati da un buffer ad un altro quattro volte (una situazione non così insolita), allora l'applicazione osserverà un valore di throughput uguale a 58.75 MBps (cioè 470 Mbps), drammaticamente inferiore al valore di 1056 Mbps che credevamo di ottenere da questa macchina.

Come nota a margine, è importante osservare che esistono molte analogie tra il trasferimento di un messaggio da e verso la memoria e la trasmissione di un messaggio attraverso la rete. In particolare, il throughput effettivo del sistema di memoria è definito dalle medesime due formule viste nella Sezione 1.5.

$$\begin{aligned} \text{Throughput} &= \text{QuantitàTrasferita}/\text{TempoDiTrasferimento} \\ \text{TempoDiTrasferimento} &= \text{RTT} + 1/\text{AmpiezzaDiBanda} \times \text{QuantitàTrasferita} \end{aligned}$$

Nel caso del sistema di memoria, però, la quantità trasferita corrisponde alla quantità di dati più grande che possiamo trasferire in modo unitario lungo il bus (che dipende dalla dimensione di una riga della cache, oppure dalla dimensione delle celle di

2.10 Riepilogo

memoria, oppure ancora dalla dimensione dei messaggi), mentre il valore di RTT corrisponde alla latenza della memoria, che dipende dal fatto che la memoria sia una cache interna al chip, una cache esterna oppure la memoria principale. Esattamente come avviene nelle reti, il miglior valore di throughput effettivo si ottiene con grandi dimensioni dei blocchi di dati trasferiti e con piccoli valori di latenza. In ulteriore analogia con le reti, il throughput effettivo della memoria non è necessariamente uguale all'ampiezza di banda di picco della memoria (che rappresenta l'ampiezza di banda che si può raggiungere con un trasferimento di dati di dimensioni infinite).

Il punto centrale di questa discussione è che occorre essere ben consapevoli dei limiti imposti alle prestazioni della rete dall'ampiezza di banda della memoria, perché, se progettato con cura, il sistema può gestire adeguatamente questi limiti. Ad esempio, è possibile far coincidere i buffer usati dal driver del dispositivo, quelli usati dal sistema operativo e quelli usati dall'applicazione in modo da minimizzare le copiate di dati. Il sistema deve anche sapere quando i dati vengono portati all'interno della cache, in modo da poter effettuare tutte le necessarie elaborazioni sui dati prima che questi escano dalla cache stessa. I dettagli relativi a come si possano implementare questi accorgimenti vanno al di là degli scopi di questo libro, ma si possono trovare nelle pubblicazioni citate alla fine del capitolo.

Infine, dobbiamo apprendere da questa discussione una seconda lezione: quando la rete non ha le prestazioni che vi aspettavate dovesse avere, non è sempre colpa della rete. In molti casi il vero collo di bottiglia del sistema è una delle macchine connesse alla rete: ad esempio, quando una pagina Web impiega molto tempo a comparire nel vostro browser, potrebbe essere l'effetto di una congestione nella rete, ma è altrettanto probabile che il server all'altro capo della rete non sia in grado di reggere il carico di lavoro a cui è sottoposto.

2.10 Riepilogo

Questo capitolo presenta i blocchi hardware elementari per la costruzione di una rete di calcolatori, i nodi e le linee di connessione, e discute i cinque problemi chiave che devono essere affrontati e risolti per fare in modo che due o più nodi direttamente connessi mediante un mezzo fisico possano scambiarsi messaggi tra loro.

Per prima cosa, i mezzi fisici veicolano segnali, per cui è necessario codificare nel nodo sorgente i bit che compongono un messaggio binario, generando un segnale che verrà poi decodificato nel nodo ricevente per ricostruire i bit del messaggio: si tratta del problema della codifica, reso complicato dalla necessità di mantenere sincronizzati il mittente e il destinatario. Abbiamo analizzato quattro diverse tecniche di codifica, NRZ, NRZI, Manchester e 4B/5B, che presentano tra loro differenze sostanziali nel modo di codificare l'informazione di sincronismo insieme ai dati che vengono trasmessi. Una delle caratteristiche principali di uno schema di codifica è la sua efficienza, rappresentata dal numero di impulsi del segnale che sono necessari per codificare un bit.

Una volta che sia possibile trasmettere bit fra i nodi, il passo successivo consiste nel capire come organizzare questi bit in frame: si tratta del problema denominato framing, che si focalizza sulla capacità di identificare l'inizio e la fine di ciascun frame. Anche in questo caso abbiamo analizzato diverse tecniche: protocolli orientati ai byte, protocolli orientati ai bit e protocolli basati sul clock.

Nell'ipotesi che ciascun nodo sia in grado di identificare l'insieme di bit che costituiscono un frame, il terzo problema consiste nel determinare se tali bit siano in effetti corretti o se

siano stati corrotti durante il trasferimento: si tratta del problema della rilevazione di errori, per il quale abbiamo analizzato tre diversi approcci, denominati verifica di ridondanza ciclica (CRC), parità bidimensionale e somma di controllo. Di questi, l'approccio CRC fornisce le garanzie migliori ed è quello più ampiamente utilizzato al livello di linea di collegamento.

Dato che alcuni frame arriveranno al nodo di destinazione con errori e verranno conseguentemente eliminati, il problema successivo consiste nel recuperare tali perdite, con l'obiettivo di far apparire affidabile la linea di collegamento. La soluzione generale a questo problema viene chiamata ARQ e richiede l'uso combinato di conferme (acknowledgement) e di temporizzazioni che scadono (timeout). Ciò che rende interessanti questi algoritmi è il loro utilizzo effettivo della linea, che ha l'obiettivo di tenere sempre la "condutture" piena.

L'ultimo problema non attiene ai collegamenti punto-punto, ma è il problema centrale nei collegamenti ad accesso multiplo: come mediare l'accesso ad un mezzo fisico condiviso in modo che tutti i nodi abbiano la possibilità di trasmettere i propri dati. In questo caso abbiamo esaminato tre diversi protocolli di accesso al mezzo, Ethernet, token ring e wireless, che sono stati utilizzati nella realizzazione pratica di reti locali. Ciò che hanno in comune queste diverse tecnologie è il controllo della rete distribuito fra tutti i nodi connessi alla rete stessa, senza nessuna dipendenza da un arbitro centrale.

Il capitolo si è poi concluso osservando che, in pratica, la maggior parte degli algoritmi che risolvono questi cinque problemi vengono implementati da adattatori che collegano gli host alla rete: di conseguenza, il progetto di tali adattatori ha un'importanza critica sulle prestazioni complessive della rete.

Problema aperto Fa parte dell'hardware?

Una delle domande più importanti che sorgono durante la progettazione di un sistema di elaborazione è: cosa fa parte dell'hardware e cosa, invece, viene realizzato in software? Nel caso delle reti, l'adattatore di rete si colloca nel cuore del problema: ad esempio, perché l'algoritmo Ethernet, presentato nella Sezione 2.6 di questo capitolo, viene tipicamente implementato in un adattatore di rete, mentre i protocolli di livello più elevato discussi nei capitoli seguenti non lo saranno?

Ovviamente, è possibile realizzare un adattatore di rete con un microprocessore di utilizzo generico, con la possibilità di trasferirvi anche protocolli di livello più elevato, come quelli della pila TCP/IP, ma ciò solitamente non viene fatto per motivi un po' complicati, ma relativi all'economia della progettazione di calcolatori: il processore principale dell'host è solitamente il processore più veloce dell'intero calcolatore e sarebbe un peccato se tale veloce processore dovesse attendere che un più lento adattatore eseguisse i protocolli TCP/IP, quando potrebbe invece svolgere da solo tale compito. D'altra parte, una parte dell'elaborazione di protocollo compete all'adattatore di rete. La regola pratica e generale prevede che qualsiasi elaborazione per cui un processore dedicato sia in grado di funzionare alla velocità richiesta dalla linea di collegamento, in modo che un processore più veloce non migliorerebbe la situazione, è una valida candidata per essere delegata all'adattatore. In altre parole, qualsiasi funzione che trovi il proprio limite nella velocità della linea, invece che nel processore che si trova all'estremità della linea stessa, può essere implementata in modo efficiente nell'adattatore.

Storicamente, la decisione relativa a quali funzionalità realizzare nell'adattatore di rete e quali nel computer host è sempre stata così complessa da generare una sostanziosa ricerca.

Nei sistemi moderni, quasi sempre accade che lo strato MAC e gli strati inferiori siano realizzati nell'adattatore, mentre lo strato IP e quelli superiori sono realizzati nell'host. Tuttavia, è interessante notare come il medesimo dibattito, relativo a quanta assistenza da parte dell'hardware sia necessaria al di sopra dello strato MAC, proseguia nel progetto di switch e router, che sarà argomento dei prossimi due capitoli.

Indipendentemente da quali siano precisamente i protocolli implementati dall'adattatore di rete, prima o poi i dati giungono al calcolatore e quando ciò avviene è molto importante l'efficienza con cui i dati vengono trasferiti dall'adattatore alla memoria del calcolatore. Ricordate che, come abbiamo visto nella Sezione 2.9.3, l'ampiezza di banda della memoria, cioè la velocità con cui i dati possono essere trasferiti da una locazione di memoria ad un'altra, è un fattore potenzialmente limitante nelle prestazioni di macchine appartenenti alla categoria delle workstation. Un meccanismo di trasferimento dei dati tra host e adattatore che sia inefficiente può, quindi, limitare la velocità di throughput osservata dai programmi applicativi in esecuzione sull'host. Innanzitutto va considerato il problema relativo all'utilizzo della modalità DMA o di I/O programmato (PIO), ciascuna avente alcuni vantaggi in situazioni diverse. Secondariamente, c'è il problema della buona integrazione dell'adattatore nel meccanismo di gestione dei buffer del sistema operativo: un sistema integrato in modo accurato è solitamente in grado di evitare la copiatura dei dati a livelli superiori del grafo di protocolli, migliorando così il throughput tra applicazioni.

Ulteriori letture

Uno dei contributi più importanti degli ultimi venti anni nel campo delle reti di calcolatori è il lavoro originale di Metcalf e Boggs (1976) che presentava Ethernet. Molti anni più tardi, nel 1988, Boggs, Mogul e Kent hanno riportato le proprie esperienze pratiche relative a Ethernet, demolendo la maggior parte dei miti che si erano fatti strada in letteratura durante gli anni. La lettura di entrambe le pubblicazioni è praticamente obbligatoria. La terza e la quarta pubblicazione in elenco discutono i problemi relativi all'integrazione tra software di sistema e adattatori per reti ad alta velocità.

- Metcalf, R., e D. Boggs "Ethernet: Distributed packet switching for local computer networks", *Communications of the ACM* 19(7):395-403, July 1976.
- Boggs, D., J. Mogul e C. Kent "Measured capacity of an Ethernet", *Proceedings of the SIGCOMM '88 Symposium*, pagg. 222-234, August 1988.
- Metcalf, R. "Computer/network interface design lessons from Arpanet and Ethernet", *IEEE Journal of Selected Areas in Communication (JSAC)* 11(2):173-180, February 1993.
- Druschel, P., M. Abbot, M. Pagels e L.L. Peterson "Network subsystem design", *IEEE Network (Special Issue on End-System Support for High Speed Networks)* 7(4):8-17, July 1993.

Esistono innumerevoli testi che pongono grande enfasi ai livelli più bassi della gerarchia di rete, focalizzandosi particolarmente sulle telecomunicazioni, cioè sulle reti dal punto di vista di una compagnia telefonica: ne sono validi esempi i libri di Spragins *et al.* [SHP91] e di Minoli [Min93]. Parecchi altri libri si concentrano sulle varie tecnologie per reti locali, tra i quali il libro di Stalling [Sta00b] è il più completo, mentre Jain fornisce una descrizione approfondita di FDDI [Jai94]. Il libro di Jain tratta anche, ad un buon livello introduttivo, i

dettagli di basso livello delle comunicazioni ottiche. Nell'articolo di Ross [Ros86] si può trovare una panoramica completa di FDDI.

Per un'introduzione sulla teoria dell'informazione, un buon punto di partenza è il libro di Blahut [Bla87], insieme al lavoro fondamentale di Shannon sulla capacità di una linea di collegamento [Sha84].

Per un'introduzione generale alla matematica che sta alla base dei codici d'errore, si raccomanda il testo di Rao e Fujiwara [RF89], mentre per una discussione più dettagliata della matematica relativa, in particolare, ai codici CRC, insieme ad altre informazioni sull'hardware usato per calcolarli, si veda Peterson e Brown [PB61].

Sull'argomento della progettazione di adattatori di rete, molto lavoro è stato fatto nei primi anni Novanta da ricercatori che avevano l'obiettivo di cercare di connettere host a reti operanti a velocità sempre più elevate. Oltre ai due esempi citati nell'elenco delle letture consigliate, si veda anche Traw e Smith [TS93], Ramakrishnan [Ram93], Edwards *et al.* [EWL⁺94], Druschel *et al.* [DPD94], Kanakia e Cheriton [KC88], Cohen *et al.* [CFFD93] e Steenkiste [Ste94a]. Recentemente è approdata sul mercato una nuova generazione di schede d'interfaccia che utilizzano *processori di rete* (network processor): Spalink *et al.* [SKPG01] hanno dimostrato come questi processori possano essere programmati per realizzare diverse funzionalità di rete.

Per informazioni generali sull'architettura dei calcolatori, un eccellente testo di riferimento è il libro di Hennessy e Patterson [HP02].

Infine, raccomandiamo il seguente riferimento attivo:

- <http://standards.ieee.org/>: stato di diversi standard IEEE relativi alle reti

Esercizi

1. Mostrate, per lo schema di bit di Figura 2.46, le codifiche NRZ, Manchester e NRZI. Ipotizzate che il segnale NRZI inizi con il valore basso.
2. Mostrate la codifica 4B/5B e il conseguente segnale NRZI per la seguente sequenza di bit:

1110 0101 0000 0011

- ✓ 3. Mostrate la codifica 4B/5B e il conseguente segnale NRZI per la seguente sequenza di bit:

1101 1110 1010 1101 1011 1110 1110 1111

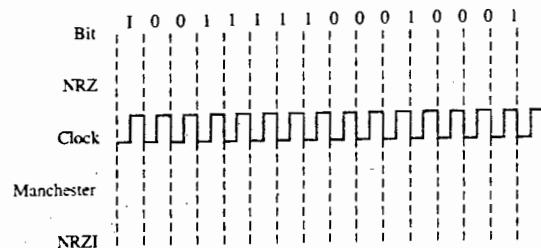


Figura 2.46 Schema per l'Esercizio 1.

4. Nella codifica 4B/5B (Tabella 2.4), soltanto due delle parole di codice a 5 bit utilizzate terminano con due zeri. Quante sono le possibili sequenze di 5 bit (usate dal codice in esame oppure no) che soddisfano il requisito più stringente di avere al massimo un solo zero iniziale e un solo zero finale? Esiste una corrispondenza tra tutte le sequenze di 4 bit e tali sequenze di 5 bit?

5. Con un protocollo di framing che usa l'interposizione di bit, mostrate la sequenza di bit trasmessa sulla linea quando il frame contiene la seguente sequenza di bit:

11010111; 10101111101011111110

Contrassegnate i bit interposti.

6. Supponete che da una linea arrivi la seguente sequenza di bit:

11010111; 1010111110010111110110

Mostrate il frame risultante dopo la rimozione dei bit interposti. Indicate anche gli errori eventualmente presenti nel frame.

- ✓ 7. Supponete che da una linea arrivi la seguente sequenza di bit:

01101011; 11010100111111011001111110

Mostrate il frame risultante dopo la rimozione dei bit interposti. Indicate anche gli errori eventualmente presenti nel frame.

8. Supponete di voler inviare dati usando il protocollo di framing BISYNC e che gli ultimi due byte dei dati siano DLE e ETX. Quale sequenza di dati dovrebbe venire trasmessa subito prima del CRC?
9. Per ciascuno dei seguenti protocolli di framing, fornite un esempio di una sequenza di byte/bit che non dovrebbe mai essere presente in una trasmissione.
 - a) BISYNC
 - b) HDLC

- ★ 10. Ipotizzate che un ricevitore SONET risincronizzi il proprio clock ogni volta che compare un valore 1, mentre il segnale viene campionato a metà dell'intervallo che esso ritiene essere l'intervallo di bit.

- a) Qual è l'accuratezza relativa tra i clock del trasmettitore e del ricevitore necessaria per ricevere correttamente 48 byte di seguito con valore 0 (che costituiscono una cella ATM AAL5)?
- b) Considerate una stazione di inoltro A su una linea SONET STS-1, che riceve frame dal flusso entrante proveniente da B e li ritrasmette sul flusso uscente. Qual è l'accuratezza relativa tra i clock di A e B necessaria per impedire che in A si accumuli più di un frame al minuto?
11. Dimostrate che la parità bidimensionale consente la rilevazione di tutti gli errori di 3 bit.
12. Fornite un esempio di un errore di 4 bit che non verrebbe rilevato dalla parità bidimensionale, come mostrato in Figura 2.16. Qual è l'insieme generico di condizioni nelle quali gli errori di 4 bit non vengono rilevati?
13. Dimostrate che la parità bidimensionale fornisce al ricevitore informazioni sufficienti per correggere qualsiasi errore di 1 bit (nell'ipotesi che il ricevitore sappia che c'è un solo bit errato), ma non un errore qualsiasi di 2 bit.

14. Dimostrate che il checksum di Internet non assume mai il valore 0xFFFF (cioè che il valore finale della somma non è mai 0x0000) tranne nel caso in cui tutti i byte del buffer hanno il valore 0 (le specifiche di Internet, infatti, richiedono che il checksum 0x0000 venga trasmesso come 0xFFFF, riservando il valore 0x0000 per l'omissione del checksum; notate che, in aritmetica in complemento a uno, 0x0000 e 0xFFFF sono due diverse rappresentazioni del numero 0).
15. Dimostrate che il calcolo del checksum di Internet mostrato nel testo è indipendente dall'ordinamento dei byte (ordinamento di host o ordinamento di rete), tranne per il fatto che i byte del checksum finale devono essere successivamente scambiati secondo l'ordinamento corretto. In particolare, dimostrate che la somma di numeri interi a 16 bit può essere eseguita in qualunque dei due schemi di ordinamento. Ad esempio, se la somma in complemento a uno (indicata con '+') di parole a 16 bit è rappresentata da

$$[A, B] +' [C, D] +' \dots +' [Y, Z]$$

allora la seguente somma scambiata è uguale alla precedente somma originale:

$$[B, A] +' [D, C] +' \dots +' [Z, Y]$$

16. Supponete che un byte contenuto in un buffer destinato ad alimentare l'algoritmo di checksum di Internet debba essere decrementato (pensate, ad esempio, al campo contatore di hop nell'intestazione). Individuate un algoritmo per calcolare il checksum conseguente alla modifica senza riesaminare l'intero buffer. L'algoritmo dovrebbe tenere in considerazione il fatto che il byte in questione sia di ordine inferiore o superiore.
- ★ 17. Dimostrate che il checksum di Internet può essere calcolato prendendo prima la somma a 32 bit in complemento a uno del buffer considerato come unità a 32 bit, poi prendendo la somma a 16 bit in complemento a uno delle semiparole superiori e inferiori, per poi complementare il risultato come in precedenza (per avere a disposizione la somma a 32 bit in complemento a uno all'interno di hardware che funziona a 32 bit in complemento a due occorre avere accesso al bit di "overflow").
18. Supponete di voler trasmettere il messaggio 11001001 e di volerlo proteggere dagli errori usando il polinomio CRC $x^3 + 1$.
- Usate la divisione in colonna di polinomi per determinare il messaggio da trasmettere.
 - Supponete che il bit più a sinistra del messaggio cambi di valore per effetto di rumore sulla linea di trasmissione. Qual è il risultato del calcolo del CRC eseguito dal ricevitore? Come fa il ricevitore a sapere che c'è stato un errore?
- ✓ 19. Supponete di voler trasmettere il messaggio 1011 0010 0100 1011 e di volerlo proteggere dagli errori usando il polinomio CRC-8 $x^8 + x^2 + x^1 + 1$.
- Usate la divisione in colonna di polinomi per determinare il messaggio da trasmettere.
 - Supponete che il bit più a sinistra del messaggio cambi di valore per effetto di rumore sulla linea di trasmissione. Qual è il risultato del calcolo del CRC eseguito dal ricevitore? Come fa il ricevitore a sapere che c'è stato un errore?
20. L'algoritmo CRC presentato in questo capitolo richiede molte manipolazioni di bit. tuttavia è possibile eseguire una divisione in colonna di polinomi considerando più bit per volta, con un metodo che usa opportune tabelle, consentendo così un'implementazione

software efficiente del calcolo di CRC. Presentiamo qui tale strategia di calcolo per la divisione con 3 bit per volta (si veda la Tabella 2.6): nella realtà, invece, considereremmo i 1 bit 8 per volta e la tabella avrebbe 256 righe. Sia $C = C(x)$ il polinomio divisorio, $x^8 + x^2 + x^1 + 1$ oppure 1101. Per costruire la tabella relativa a C , consideriamo ciascuna sequenza di 3 bit, p , con l'aggiunta di tre zeri finali, e calcoliamo il quoziente $q = p \text{ } \sim 000/C$, ignorando il resto. La terza colonna è il prodotto $C \times q$, i cui primi 3 bit devono essere uguali a p .

- Verificate per $p = 110$ che i quozienti $p \text{ } \sim 000/C$ e $p \text{ } \sim 111/C$ siano uguali, cioè che i bit finali non modificano il risultato.
 - Completate la tabella con i dati mancanti.
 - Usate la tabella per dividere 101 001 011 001 100 per C . Suggerimento: i primi 3 bit del dividendo sono $p = 101$, per cui dalla tabella si deduce che i primi 3 bit del quoziente sono $q = 110$. Scrivete 110 sopra i secondi 3 bit del dividendo e sottraete dai primi 6 bit del dividendo il valore $C \times q = 101\ 110$, nuovamente identificato nella tabella. Continuando a procedere a gruppi di 3 bit, non ci dovrebbe essere alcun resto.
- ★ 21. Con un bit di parità si possono rilevare tutti gli errori di un solo bit. Dimostrate che almeno una delle seguenti generalizzazioni è falsa:
- Dimostrate che se il messaggio m è lungo 8 bit, allora non esiste codice di rilevazione d'errore $e = e(m)$ di dimensione uguale a 2 bit che possa rilevare tutti gli errori di 2 bit. Suggerimento: Considerate l'insieme M di tutti i messaggi di 8 bit aventi un solo bit al valore 1; notate che qualunque messaggio nell'insieme M può essere trasformato in un altro messaggio ancora appartenente all'insieme con un errore di 2 bit, quindi dimostrate che alcune coppie di messaggi m_1 e m_2 appartenenti a M devono avere lo stesso codice d'errore e .
 - Trovate un valore di N (non necessariamente il minimo) in modo che nessun codice a rilevazione d'errore a 32 bit applicato a blocchi di N bit possa rilevare tutti gli errori che modificano fino a 8 bit.
 - Considerate un protocollo ARQ che usa soltanto conferme negative (NAK), senza conferme positive (ACK). Descrivete quali timeout devono essere programmati e spiegate perché solitamente viene preferito un protocollo con conferme positive rispetto ad uno con conferme negative.
 - Considerate un algoritmo ARQ operativo su una linea in fibra ottica punto-punto di 20 km.

Tabella 2.6 Calcolo di CRC mediante l'uso di tabelle.

p	$q = p \text{ } \sim 000/C$	$C \times q$
000	000	000 000
001	001	001 101
010	011	010 ____
011	0____	011 ____
100	111	100 011
101	110	101 110
110	100	110 ____
111	____	111 ____

- a) Calcolate il ritardo di propagazione per questa linea, ipotizzando che la velocità della luce nella fibra sia 2×10^8 m/s.
- b) Suggerite un valore ragionevole per il timeout da utilizzare nell'algoritmo ARQ.
- c) Come mai, nonostante tale valore per il tempo del timeout, è comunque possibile che l'algoritmo ARQ veda i temporizzatori scadere, rendendo necessaria la ritrasmissione di un frame?
24. Supponete di dover progettare un protocollo a finestra scorrevole per un collegamento punto-punto a 1 Mbps verso la Luna, con una latenza di sola andata di 1.25 secondi. Ipottizzando che ciascun frame trasporti 1 KB di dati, qual è il numero minimo di bit necessari per il numero di sequenza?
- ✓ 25. Supponete di dover progettare un protocollo a finestra scorrevole per un collegamento punto-punto a 1 Mbps verso un satellite stazionario in rotazione attorno alla Terra ad un'altitudine di 3×10^4 km. Considerando che la velocità della luce sia 3×10^8 m/s e ipottizzando che ciascun frame trasporti 1 KB di dati, qual è il numero minimo di bit necessari per il numero di sequenza nei casi seguenti?
- a) $RWS = 1$
 - b) $RWS = SWS$
26. Il testo suggerisce che il protocollo sliding window può essere utilizzato per realizzare il controllo di flusso. Possiamo immaginare di farlo, imponendo al ricevitore di ritardare l'invio delle conferme (ACK) fin quando non abbia spazio libero nel buffer per accogliere il frame successivo. In questo modo ciascun ACK confermerebbe l'ultimo frame ricevuto e, allo stesso tempo, farebbe sapere alla sorgente che si è liberato spazio nel buffer per contenere il frame successivo. Spiegate perché realizzare il controllo di flusso in questo modo non sia una buona idea.
27. Gli scenari di stop-and-wait di Figura 2.19 ipotizzano implicitamente che il ricevitore ritrasmetterà il proprio ACK immediatamente dopo aver ricevuto un frame di dati duplicato. Supponete, invece, che il ricevitore mantenga un proprio temporizzatore e che ritrasmetta il proprio ACK soltanto dopo che il frame successivo non sia giunto entro l'intervallo di timeout. Tracciate i diagrammi temporali che illustrino gli scenari di Figura 2.19, dal caso (b) al caso (d), nell'ipotesi che il timeout del ricevitore abbia una durata doppia di quella della sorgente. Tracciate nuovamente il diagramma per il caso (c) nell'ipotesi che il timeout del ricevitore abbia una durata uguale alla metà di quella della sorgente.
28. Nella trasmissione stop-and-wait, supponete che sia la sorgente che il ricevitore ritrasmettano immediatamente il proprio ultimo frame, qualora ricevano un frame di dati o un ACK duplicati; tale strategia è in prima approssimazione ragionevole, in quanto la ricezione di un tale duplicato è molto probabilmente un segnale del fatto che all'altro capo della trasmissione è scaduto un timeout.
- a) Tracciate un diagramma temporale che mostri ciò che accade se il primo frame di dati viene, per qualche motivo, duplicato, senza che alcun frame vada perduto. Quanto durerà la duplicazione? Questa situazione è nota come "problema dell'apprendista stregone".
 - b) Supponete che, come i dati, le conferme (ACK) vengano ritrasmesse se non giunge risposta entro l'intervallo del timeout. Supponete, inoltre, che le due controparti usino un intervallo di timeout di uguale durata. Identificate uno scenario ragionevolmente probabile per innescare il problema dell'apprendista stregone.

29. Indicate in dettaglio come aggiungere il controllo di flusso al protocollo sliding window, usando le conferme (ACK) per veicolare informazioni addizionali allo scopo di ridurre il valore di SWS quando il ricevitore esaurisce lo spazio disponibile nel buffer. Illustrate il vostro protocollo con un diagramma temporale della trasmissione, nell'ipotesi che SWS e RWS valgano inizialmente 4, che la velocità della linea sia infinita e che il ricevitore possa creare nuovi buffer al ritmo di uno al secondo (per cui il collo di bottiglia è proprio il ricevitore). Mostrate cosa accade agli istanti $T = 0, T = 1, \dots, T = 4$ secondi.
30. Descrivete un protocollo che combini l'algoritmo sliding window con le conferme selettive. Il protocollo dovrebbe ritrasmettere appena possibile, senza farlo, però, nel caso in cui un frame arrivi semplicemente fuori ordine di una o due posizioni. Il vostro protocollo dovrebbe anche rendere esplicito il proprio comportamento nel caso che vengano perduti parecchi frame consecutivi.
31. Tracciate un diagramma temporale per l'algoritmo sliding window con $SWS = RWS = 3$ frame, nelle due situazioni seguenti. Usate un intervallo di timeout circa uguale a $2 \times RTT$.
- a) Il frame 4 viene perduto.
 - b) I frame dal 4 al 6 vengono perduti.
- ✓ 32. Tracciate un diagramma temporale per l'algoritmo sliding window con $SWS = RWS = 4$ frame, nelle due situazioni seguenti. Ipottizzate che il ricevitore invii una conferma duplicata se non riceve il frame che sta aspettando: ad esempio, se sta aspettando FRAME[2] ma riceve FRAME[3], invia DUPACK[2]. Inoltre, dopo aver ricevuto tutti i frame fuori sequenza che stava aspettando, il ricevitore invia una conferma cumulativa: ad esempio, se riceve il frame perduto FRAME[2] dopo aver già ricevuto FRAME[3], FRAME[4] e FRAME[5], invia ACK[5]. Usate un intervallo di timeout circa uguale a $2 \times RTT$.
- a) Il frame 2 viene perduto. Come detto, la ritrasmissione avviene dopo la scadenza del timeout.
 - b) Il frame 2 viene perduto. La ritrasmissione avviene dopo la scadenza del timeout oppure in seguito alla ricezione del primo DUPACK. Questo schema riduce il tempo necessario per la transazione? Notate che alcuni protocolli end-to-end (come alcune varianti di TCP) usano uno schema simile per velocizzare la ritrasmissione.
33. Supponete di tentare l'esecuzione dell'algoritmo sliding window con $SWS = RWS = 3$ e con $\text{MaxSeqNum} = 5$. Di conseguenza, il pacchetto N -esimo, DATA[N], contiene in realtà il valore $N + 6$ nel proprio campo destinato al numero di sequenza. Fornite un esempio in cui l'algoritmo si confonde, cioè uno scenario in cui il ricevitore si aspetta di ricevere DATA[5] ma accetta invece DATA[0], che viene trasmesso con lo stesso numero di sequenza. Nessun pacchetto può arrivare fuori ordine. Notate che ciò implica che $\text{MaxSeqNum} \geq 6$ è una condizione sia necessaria che sufficiente.
34. Considerate l'algoritmo sliding window con $SWS = RWS = 3$, senza arrivi fuori ordine e con numeri di sequenza con precisione infinita.
- a) Dimostrate che se DATA[6] si trova nella finestra di ricezione, allora DATA[0] (o, in generale, qualsiasi dato anteriore ad esso) non può arrivare al ricevitore (e, quindi, $\text{MaxSeqNum} = 6$ sarebbe sufficiente).
 - b) Dimostrate che se si può inviare ACK[6] (cioè, in altre parole, DATA[5] si trova nella finestra della sorgente), allora non può accadere che si riceva ACK[2] (o una delle conferme precedenti).
- Queste dimostrazioni contribuiscono a provare la formula vista nella Sezione 2.5.2, nel caso in cui $SWS = 3$. Notate che la parte (b) implica che la situazione esaminata nel-

- l'Esercizio precedente non può essere invertita per considerare un guasto che permetta di distinguere ACK[0] da ACK[5].
35. Supponete di eseguire l'algoritmo sliding window con SWS = 5 e RWS = 3, senza arrivi fuori ordine.
- Trovate il valore minimo di MaxSeqNum. Potete ipotizzare che sia sufficiente trovare il valore minimo di MaxSeqNum per cui, se DATA[MaxSeqNum] si trova nella finestra di ricezione, allora DATA[0] non può più arrivare.
 - Fornite un esempio che mostri come MaxSeqNum - 1 non sia sufficiente.
 - Enunciate una regola generale per esprimere il valore minimo di MaxSeqNum in funzione di SWS e RWS.
36. Supponete che A sia connesso a B mediante un router intermedio R, come mostrato in Figura 2.47. Ciascuna delle linee di collegamento A-R e R-B accetta e trasmette un solo pacchetto al secondo in ciascuna direzione (per cui due pacchetti impiegano due secondi), mentre le due direzioni possono trasmettere indipendentemente tra loro. Ipotizzate che A trasmetta verso B usando il protocollo sliding window con SWS = 4.
- Indicate quali pacchetti arrivano in ciascun nodo e partono da ciascun nodo agli istanti di tempo Time = 0, 1, 2, 3, 4, 5 (oppure segnateli su un diagramma temporale).
 - Cosa accadrebbe se le linee avessero un ritardo di propagazione di un secondo, ma accettassero immediatamente tutti i pacchetti offerti (cioè, latenza = 1 secondo, ma ampiezza di banda infinita)?
37. Supponete che A sia connesso a B mediante un router intermedio R, come nel problema precedente. La linea A-R è istantanea, mentre la linea R-B trasmette un solo pacchetto al secondo, uno per volta (per cui due pacchetti impiegano due secondi). Ipotizzate che A trasmetta verso B usando il protocollo sliding window con SWS = 4. Indicate quali pacchetti arrivano in ciascun nodo e partono da ciascun nodo agli istanti di tempo Time = 0, 1, 2, 3, 4, 5. Quale dimensione raggiunge la coda in R?
38. Considerate la situazione dell'Esercizio precedente, ma questa volta ipotizzate che il router abbia una coda di dimensione 1, cioè che possa memorizzare un solo pacchetto oltre a quello che sta inviando (in ciascuna direzione). Imponete ad A un timeout di 5 secondi, con SWS = 4. Mostrate cosa accade dopo ogni secondo a partire da $T = 0$ finché non siano stati consegnati con successo tutti i quattro pacchetti presenti nella prima finestra piena.
39. Perché è importante che i protocolli soprastranti a Ethernet abbiano nella propria intestazione un campo che indichi la lunghezza del messaggio?
40. Quali problemi possono insorgere quando due host che si trovano nella stessa rete Ethernet condividono lo stesso indirizzo hardware? Descrivete ciò che accade e spiegate perché tale comportamento è un problema.
41. La specifica di Ethernet del 1982 consentiva che fra due stazioni vi fosse un cavo coassiale di lunghezza massima 1500 m, oppure un cavo di connessione punto-punto di altro tipo e lungo fino a 1000 m, nonché due ripetitori. Ciascuna stazione o ripetitore si connette al cavo coassiale tramite un "cavo drop" lungo fino a 50 m. I ritardi tipici dovuti a ciascun dispositivo sono indicati in Tabella 2.7 (dove c è la velocità della luce nel

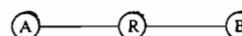


Figura 2.47 Schema per gli Esercizi 36, 37 e 38.

- vuoto, 3×10^8 m/s). Qual è il ritardo di propagazione di round-trip nel caso peggiore, misurato in bit, dovuto alle sorgenti di ritardo elencate? (l'elenco non è completo: tra le altre sorgenti di ritardo, possiamo citare il tempo di salita del segnale e il tempo necessario alla verifica dello stato elettrico della linea)
- ★ 42. Il cavo coassiale Ethernet era limitato ad una lunghezza massima di 500 m fra due ripetitori, che rigenerano il segnale riportandolo al 100% della sua ampiezza originale. Lungo un segmento di 500 m, il segnale poteva degradarsi fino a non meno del 14% del proprio valore originale (8.5 dB). Lungo 1500 m, quindi, la degradazione potrebbe essere $(0.14)^3 = 0.3\%$. Tale segnale, anche dopo 2500 m, è ancora abbastanza forte da poter essere letto: perché mai, allora, vengono richiesti ripetitori ogni 500 m?
43. Supponete che il ritardo di propagazione di round-trip per Ethernet sia $46.4 \mu s$, avendo come conseguenza una dimensione minima del pacchetto uguale a 512 bit (464 bit corrispondenti al ritardo di propagazione + 48 bit di segnale di disturbo).
- Come si modifica la dimensione minima del pacchetto se il ritardo viene mantenuto costante, mentre la velocità del segnale viene portata a 100 Mbps?
 - Quali sono gli svantaggi di una dimensione minima del pacchetto tanto grande?
 - Se la compatibilità non andasse salvaguardata, come si potrebbero riscrivere le specifiche in modo da consentire una dimensione minima del pacchetto inferiore?
- ★ 44. Siano A e B due stazioni che tentano di trasmettere su una rete Ethernet. Ciascuna ha costantemente pacchetti pronti da trasmettere in coda; i pacchetti di A vengono indicati con A_1, A_2 , e così via, e quelli di B in modo analogo. Sia $T = 51.2 \mu s$ l'unità base del backoff esponenziale. Supponete che A e B tentino simultaneamente di trasmettere il proprio frame 1, che questi collidano e che vengano scelti degli intervalli di backoff, rispettivamente, uguali a $0 \times T$ e $1 \times T$, per cui A risulta vincitrice e trasmette A_1 , mentre B aspetta. Al termine di questa trasmissione, B tenterà nuovamente di trasmettere B_1 , mentre A tenterà di trasmettere A_2 . Questi primi tentativi collideranno, ma ora A attendrà per un tempo uguale a $0 \times T$ oppure $1 \times T$, mentre l'intervalllo di backoff di B verrà scelto tra $0 \times T, \dots, 3 \times T$.
- Determinate la probabilità che A risulti nuovamente vincitrice subito dopo questa prima collisione, cioè che la prima scelta del tempo di backoff di A, $k \times 51.2 \mu s$, sia inferiore alla scelta di B.
 - Supponete che A risulti vincitrice per questa seconda trasmissione. A trasmette A_3 e, quando ha terminato, A e B generano una nuova collisione per il tentativo di A di trasmettere A_4 e quello di B di trasmettere nuovamente B_1 . Determinate la probabilità che A vinca di nuovo, immediatamente dopo questa prima collisione.
 - Determinate un limite inferiore ragionevole per la probabilità che A risulti vincitrice in tutte le successive occasioni di collisione.

Tabella 2.7 Ritardi tipici dovuti a diversi dispositivi (Esercizio 41).

Dispositivo	Ritardo
Cavo coassiale	Velocità di propagazione $0.77c$
Linea/cavo drop	Velocità di propagazione $0.65c$
Ripetitore	Circa $0.6 \mu s$ ciascuno
Transceiver	Circa $0.2 \mu s$ ciascuno

- d) Cosa accade al frame B_1 ?
 Questa situazione prende il nome di *effetto cattura* di Ethernet.
45. Supponete che l'algoritmo di trasmissione di Ethernet venga modificato come segue: dopo ciascun tentativo di trasmissione che ha successo, un host attende uno o due intervalli di tempo prefissati prima di tentare di trasmettere nuovamente, dopodiché utilizza il backoff nel solito modo.
- Spiegate perché ora l'effetto di cattura descritto nel precedente Esercizio è molto meno probabile.
 - Mostrate come la strategia appena delineata possa portare ad una situazione in cui due host catturano la rete Ethernet, alternandosi nella trasmissione ed escludendo un terzo host.
 - Proponete un approccio alternativo, ad esempio modificando il backoff esponenziale. Quali particolari della storia di una stazione potrebbero essere usati come parametri nel backoff modificato?
46. Ethernet usa la codifica Manchester. Ipotizzando che gli host che condividono una Ethernet non siano perfettamente sincronizzati, come è possibile che questa codifica consenta la rilevazione di una collisione subito dopo che questa si è verificata, senza dover attendere il CRC al termine del pacchetto?
47. Supponete che A, B e C eseguano la loro prima analisi della portante, nel corso di un tentativo di trasmissione, proprio mentre una quarta stazione, D, sta trasmettendo. Tracciate un diagramma temporale che mostri una possibile sequenza di trasmissioni, tentativi, collisioni e scelte del backoff esponenziale. Il diagramma deve anche soddisfare i seguenti requisiti: (i) i tentativi iniziali di trasmissione devono avvenire nell'ordine A, B, C, ma le trasmissioni che hanno successo devono essere, nell'ordine, quelle di C, di B e di A; (ii) si devono verificare almeno quattro collisioni.
48. Ripetete l'Esercizio precedente, con la nuova ipotesi che Ethernet sia del tipo con persistenza p , dove $p = 0.33$ (cioè una stazione in attesa trasmette immediatamente con probabilità p quando la linea diventa inattiva, altrimenti attende per un intervallo di tempo uguale a $51.2 \mu s$ e ripete la procedura). Il diagramma temporale deve ancora soddisfare il requisito (i) del problema precedente, ma, al posto del requisito (ii), deve ora mostrare almeno una collisione e almeno una serie di quattro rinvii di trasmissione con la linea inattiva. Anche in questo caso notate come siano possibili molte soluzioni diverse.
- ★ 49. Supponete che gli indirizzi fisici di Ethernet vengano scelti a caso (usando bit veramente casuali).
- Qual è la probabilità che in una rete con 1024 host vi siano due indirizzi uguali?
 - Qual è la probabilità che l'evento precedente accada in una o più reti appartenenti ad un insieme di 2^{20} reti?
 - Qual è la probabilità che vi siano due host con lo stesso indirizzo tra i 2^{30} host appartenenti a tutte le reti del punto (b)?
- Suggerimento: i calcoli per i punti (a) e (c) sono una variazione di quelli usati per risolvere il cosiddetto "problema del compleanno": in un insieme di N persone, qual è la probabilità che due di loro compiano gli anni nello stesso giorno (cioè abbiano lo stesso indirizzo)? La seconda persona ha una probabilità $1 - 1/365$ di compiere gli anni in un giorno diverso dalla prima persona, la terza ha una probabilità $1 - 2/365$ di compiere gli anni in un giorno diverso dalle prime due persone, e così via. La probabilità che le date dei compleanni siano tutte diverse è, quindi

$$\left(1 - \frac{1}{365}\right) \times \left(1 - \frac{2}{365}\right) \times \cdots \times \left(1 - \frac{N-1}{365}\right)$$

che per piccoli valori di N è circa

$$1 - \frac{1 + 2 + \cdots + (N-1)}{365}$$

50. Supponete che cinque stazioni siano in attesa che un altro pacchetto termini la propria trasmissione su una rete Ethernet, dopodiché tutte trasmettono contemporaneamente e si ha una collisione.
- Simulate questa situazione fino al punto in cui una delle cinque stazioni ha successo nel trasmettere. Usate il lancio di monete o un altro metodo veramente casuale per determinare i tempi di backoff, con le semplificazioni seguenti: trascurate la spaziatura fra i frame, trascurate la variazione dei tempi di collisione (in modo che le ritrasmissioni avvengano sempre dopo un multiplo intero di $51.2 \mu s$) e ipotizzate che ciascuna collisione riempia esattamente e completamente un intervallo di durata $51.2 \mu s$.
 - Discutete gli effetti delle semplificazioni elencate e introdotte nella vostra simulazione rispetto al comportamento che potreste verificare in una Ethernet reale.
51. Scrivete un programma che implementi la simulazione discussa nell'esercizio precedente, con un numero di stazioni in attesa uguale a N . Anche in questo caso usate i numeri interi come modello per il tempo, T , misurato in unità di intervalli temporali, e ipotizzate nuovamente che le collisioni richiedano un intero intervallo di tempo (in modo che una collisione che avvenga al tempo T , seguita da un backoff con $k = 0$, darebbe luogo ad un tentativo di ritrasmissione al tempo $T + 1$). Determinate il ritardo medio necessario perché una stazione trasmetta con successo, per $N = 20$, $N = 40$ e $N = 100$. I vostri dati corrispondono alla sensazione che il ritardo debba essere lineare in funzione di N ? Suggerimento: per ciascuna stazione, memorizzate i valori di `NextTimeToSend` e `CollisionCount`. La simulazione è terminata quando si raggiunge un tempo T per il quale esiste un'unica stazione avente `NextTimeToSend == T`. Se tale stazione non esiste, si incrementa il tempo T . Se ce ne sono due o più, si programma una ritrasmissione e si prova di nuovo.
52. Supponete che, con N stazioni Ethernet che tentano di trasmettere nello stesso momento, siano necessari $N/2$ intervalli di tempo (*slot*) per decidere quale stazione sarà la prossima a trasmettere. Ipotizzando che la dimensione media dei pacchetti sia uguale a 5 slot, esprimete l'ampiezza di banda disponibile in funzione di N .
53. Considerate il seguente modello di rete Ethernet. I tentativi di trasmissione avvengono ad istanti casuali equamente distribuiti su un intervallo pari a λ intervalli temporali (*slot*); più precisamente, l'intervallo di tempo che intercorre tra due tentativi consecutivi è una variabile casuale esponenziale $x = -\lambda \log u$, dove u è un numero casuale appartenente all'intervallo $0 \leq u \leq 1$. Un tentativo al tempo t genera una collisione se viene effettuato un altro tentativo nell'intervallo che va da $t - 1$ a $t + 1$, con t misurato in unità multiple dello slot di $51.2 \mu s$; in caso contrario il tentativo ha successo.

- a) Scrivete un programma per simulare, per un certo valore di λ , il numero medio di slot necessari per trasmettere con successo (tale numero, moltiplicato per la durata dello slot, identifica un intervallo di tempo che viene chiamato *intervallo di contesa*). Determinate il valore minimo dell'intervallo di contesa. Notate che dovete trovare un ulteriore tentativo di trasmissione successivo a quello che avviene con successo, per poter determinare che non avvenga una collisione. Ignorate le ritrasmissioni, che difficilmente rientrano nel modello casuale appena descritto.
- b) Questa rete Ethernet alterna intervalli di contesa a trasmissioni portate a termine con successo. Supponete che la trasmissione portata a termine con successo abbia una durata media di 8 slot (cioè 512 byte). Usando la lunghezza minima dell'intervallo di contesa determinato al punto precedente, quale frazione della banda teorica di 10 Mbps è disponibile per le trasmissioni?
54. Quali condizioni si dovrebbero verificare perché un frame corrotto possa circolare indefinitamente in una rete di tipo token ring priva di monitor? Come viene risolto questo problema dal monitor?
55. Una rete token ring IEEE 802.5 ha cinque stazioni ed una lunghezza totale dei cavi di 230 m. Quanti bit di ritardo devono essere inseriti nell'anello dal monitor? Eseguite il calcolo per una rete a 4 Mbps e per una a 16 Mbps, usando una velocità di propagazione uguale a 2.3×10^8 m/s.
56. Considerate una rete di tipo token ring come la rete FDDI, in cui una stazione può trattenere il testimone per un certo periodo di tempo, chiamato *token holding time* (THT). Indicate con RingLatency il tempo necessario al testimone per compiere un'intera rotazione attorno alla rete nel caso in cui nessuna delle stazioni abbia dati da inviare.
- a) Esprimete l'efficienza della rete in funzione di THT e di RingLatency quando vi è una sola stazione attiva.
- b) Quale sarebbe il valore ottimale di THT per una rete con una sola stazione attiva (cioè con dati da inviare) per volta?
- c) Nel caso in cui vi siano N stazioni attive, fornite un limite superiore al tempo di rotazione del testimone (TRT) per questa rete.
57. Considerate una rete di tipo token ring con una latenza d'anello di 200 μ s. Ipotizzando che si utilizzi la strategia del rilascio ritardato del testimone, quale effettivo throughput si può ottenere se l'anello ha un'ampiezza di banda di 4 Mbps? E se l'ampiezza di banda fosse di 100 Mbps? Rispondete sia nel caso di un solo host attivo sia nel caso di "molti" host attivi; in questo ultimo caso, fate l'ipotesi che vi sia un numero di host che trasmettono sufficientemente elevato da poter trascurare il tempo necessario per inoltrare il token. Usate una dimensione di pacchetto di 1 KB.
58. Calcolate il massimo valore effettivo del throughput che può essere raggiunto da qualsiasi host in una rete di tipo token ring a 100 Mbps con tempo di rotazione del testimone di 200 μ s, che consente a ciascuna stazione di trasmettere un pacchetto di 1 KB ogni volta che ha il possesso del testimone. Effettuate il calcolo ipotizzando: (a) rilascio immediato; (b) rilascio ritardato.
59. Supponete che una rete di tipo token ring con rilascio ritardato a 100 Mbps abbia 10 stazioni, una latenza d'anello di 30 μ s e un TTRT concordato di 350 μ s.
- a) Quanti byte di frame sincroni possono essere inviati da ciascuna stazione, nell'ipotesi che tale possibilità di trasmissione sia distribuita equamente tra tutte le stazioni?

- b) Ipotizzate che le stazioni A, B e C siano disposte in ordine crescente lungo l'anello. A causa del traffico sincrono uniforme, il valore di TRT senza dati asincroni è 300 μ s. B invia un frame asincrono di 200 μ s (2.5 Kb). In conseguenza di ciò, qual è il valore di TRT osservato da A, B e C alla loro successiva rilevazione? Chi sarà la successiva stazione a poter trasmettere un frame di quel tipo?

Commutazione di pacchetto

Problema Non tutte le reti sono a connessione diretta

Le reti a connessione diretta descritte nel capitolo precedente hanno due limitazioni. Innanzitutto, esiste un limite al numero di host che vi si possono connettere. Ad esempio, ad una linea di collegamento punto-punto si possono connettere soltanto due host, ed una rete Ethernet può connettere fino ad un massimo di 1024 host. Poi, esiste un limite alla massima estensione geografica che può essere servita da una singola rete. Ad esempio, una rete Ethernet può estendersi soltanto per 2500 m e, nonostante le linee di collegamento punto-punto possano essere abbastanza lunghe, in realtà non si può dire che svolgano un servizio per l'area attraversata che si trova tra i due estremi. Dato che abbiamo l'obiettivo di costruire reti che possano assumere dimensioni mondiali, il problema successivo da affrontare è, quindi, quello di consentire la comunicazione fra host che non siano direttamente connessi.

Questo problema non è dissimile da quello affrontato dalla rete telefonica: *vostro telefono non è direttamente collegato a ciascuna persona che potreste voler chiamare, ma è invece connesso ad un centralino che contiene un commutatore (switch)*. Sono i commutatori a dare la sensazione di essere connessi alla persona che si trova all'altro capo della chiamata. Analogamente, le reti di calcolatori usano *commutatori di pacchetti* (così chiamati per distinguere dai commutatori di circuiti usati per la telefonia) per consentire ai pacchetti di viaggiare da un host ad un altro, anche senza che esista una connessione diretta fra tali host. Questo capitolo presenta i principi fondamentali della commutazione di pacchetti, che costituisce il cuore delle reti di calcolatori.

Un commutatore di pacchetti è un dispositivo con diverse linee, entrate e uscite, che lo collegano agli host che interconnette. Il compito principale di uno switch è quello di ricevere i pacchetti in arrivo da una linea e *inoltrarli* (o *commutarli*) verso l'uscita corretta, in modo che possano raggiungere la propria destinazione. Esistono molte modalità mediante le quali uno switch può determinare l'uscita "corretta" per un pacchetto, modalità che possono essere catalogate in approcci privi di connessione (*connectionless*) e approcci orientati alla connessione (*connection-oriented*).

Uno dei problemi chiave che deve essere gestito da un commutatore è l'**ampiezza di banda** finita delle proprie linee di uscita. Se i pacchetti destinati ad una certa uscita arrivano al commutatore ad una velocità che eccede la capacità di tale linea di uscita, siamo di fronte ad un problema di *contesa* (contention). Lo switch accoda (cioè memorizza in un buffer) i pacchetti finché non termina la contesa, ma se questa dura troppo a lungo lo switch esaurirà lo spazio di memorizzazione e sarà costretto ad eliminare pacchetti. Quando i pacchetti vengono eliminati con frequenza troppo elevata, si dice che lo switch è *congestionato*. La capacità di uno switch di gestire la contesa è un aspetto chiave delle sue prestazioni e molti switch ad elevate prestazioni usano strani accorgimenti hardware per ridurre gli effetti negativi della contesa stessa.

Questo capitolo presenta i problemi connessi all'inoltro dei pacchetti e alla contesa negli switch. Iniziamo prendendo in esame diversi approcci alla commutazione, con i modelli privi di connessione e orientati alla connessione. Poi, affrontiamo con maggiore dettaglio due particolari tecnologie. La prima è la *commutazione in reti locali* (LAN switching), che è un'evoluzione dell'interconnessione (bridging) di reti Ethernet ed è diventata una delle tecnologie dominanti nelle reti locali odiene. La seconda tecnologia di commutazione degna di nota è la *modalità di trasferimento asincrono* (ATM; asynchronous transfer mode), ampiamente diffusa tra i fornitori di servizi di telecomunicazioni in reti globali. Infine, esamineremo alcuni aspetti della progettazione dei commutatori che vanno tenuti in considerazione quando si costruiscono reti su larga scala.

3.1 Commutazione e inoltro

Detto in parole semplici, un commutatore (switch) è un meccanismo che ci consente di interconnettere linee di collegamento per formare una rete più grande. Uno switch è un dispositivo a più ingressi e a più uscite che trasferisce pacchetti da un ingresso a una o più uscite. Di conseguenza, uno switch aggiunge la topologia a stella alle topologie già viste nel capitolo precedente: linee di collegamento punto-punto, bus (Ethernet) e anello (802.5 e FDDI). La topologia a stella ha diverse proprietà interessanti:

- Anche se uno switch ha un numero prefissato di ingressi e di uscite, che limita il numero di host che possono essere connessi ad un singolo commutatore, si possono costruire grandi reti interconnettendo un certo numero di switch.
- Usando linee di collegamento punto-punto, possiamo connettere uno switch ad un altro switch oppure a host, in modo da realizzare reti ad ampia estensione geografica.
- L'aggiunta di un nuovo host alla rete mediante la sua connessione ad uno switch non implica necessariamente che gli host precedentemente connessi osserveranno una diminuzione di prestazioni della rete.

Questa ultima proprietà non è valida per le reti a mezzo fisico condiviso viste nel capitolo precedente. Ad esempio, è impossibile che due host sulla stessa Ethernet trasmettano contemporaneamente a 10 Mbps, dal momento che condividono lo stesso mezzo trasmissivo. In una rete commutata, ciascun host ha la propria linea di collegamento verso lo switch, per cui è davvero possibile che molti host trasmettano alla piena velocità (ampiezza di banda) della linea, a condizione che lo switch sia stato progettato con una sufficiente capacità cumulativa (o aggregata). Garantire un elevato throughput aggregato è uno degli obiettivi della progettazione di uno switch, per cui vi torneremo in seguito. In generale, le reti commutate vengono consi-

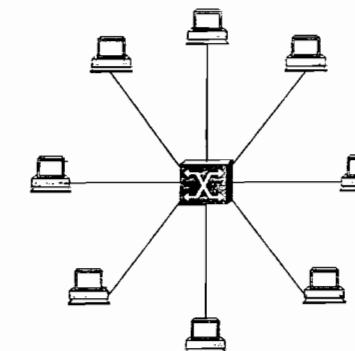


Figura 3.1 Uno switch consente la realizzazione di una topologia a stella.

derate maggiormente *scalabili* (cioè più capaci di crescere fino ad avere un gran numero di nodi) delle reti a mezzo fisico condiviso, proprio per questa capacità di consentire a molti host di raggiungere la propria velocità operativa massima.

Uno switch è connesso ad un insieme di linee di connessione e, per ciascuna di queste linee, esegue un appropriato protocollo di livello data link per comunicare con il nodo che si trova all'altro capo della linea. Il compito principale di uno switch è quello di ricevere i pacchetti in arrivo da una delle sue linee e di trasmetterli verso qualche altra linea. A volte questa funzione viene chiamata *switching* oppure *forwarding* (inoltro) ed è la funzione principale dello strato di rete dell'architettura OSI. La Figura 3.2 mostra il grafo di protocolli in esecuzione in uno switch connesso a due linee T3 e ad una linea SONET STS-1. La Figura 3.3

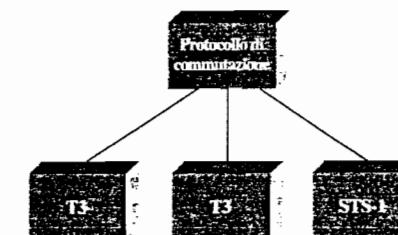


Figura 3.2 Esempio di grafo di protocolli in esecuzione in uno switch.

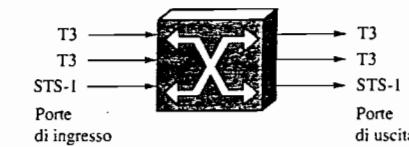


Figura 3.3 Esempio di commutatore con tre ingressi e tre uscite.

rappresenta il medesimo switch, ma abbiamo separato le parti di ingresso e di uscita relative a ciascuna linea, riferendoci a ciascun ingresso o uscita con il termine di *porta* (in generale ipotizziamo che tutte le linee di collegamento siano bidirezionali e che, quindi, siano in grado di fungere sia da ingressi sia da uscite). In altre parole, lo switch di questo esempio ha tre porte di ingresso e tre porte di uscita.

La domanda, quindi, è: come fa il commutatore a decidere quale porta di uscita utilizzare per inviare il pacchetto? La risposta generica è che si cerca nell'intestazione del pacchetto un identificatore da usare per prendere la decisione: i dettagli relativi alla modalità di utilizzo di tale identificatore variano, ma esistono due approcci molto utilizzati. Il primo approccio viene chiamato **datagram** (datagramma, parola coniata in analogia con telegramma) oppure **privi di connessione** (connectionless), mentre il secondo è l'approccio a **circuito virtuale** (virtual circuit) oppure **orientato alla connessione** (connection-oriented). Esiste anche un terzo approccio, meno comune degli altri due, ma semplice da spiegare e con alcune utili applicazioni, che viene chiamato **instradamento dalla sorgente** (source routing).

Una caratteristica comune a tutte le reti è la necessità di una modalità di identificazione di tutti i nodi terminali: tali identificatori vengono chiamati indirizzi. Abbiamo già visto esempi di indirizzi nel capitolo precedente, come l'indirizzo a 48 bit usato per Ethernet: l'unico requisito degli indirizzi Ethernet è che su una rete non devono esistere due nodi con lo stesso indirizzo, requisito che viene soddisfatto garantendo che a tutte le schede Ethernet venga assegnato un identificatore **globalmente unico**, cioè unico nel mondo. Nella discussione che segue ipotizzeremo che ciascun host sia dotato di un indirizzo globalmente unico; in seguito esamineremo altre utili proprietà che possono essere attribuite ad un indirizzo, ma l'unicità globale è sufficiente per poter iniziare.

3.1.1 Datagrammi

L'idea che sta alla base dei datagrammi è incredibilmente semplice: ciascun pacchetto deve contenere informazioni sufficienti per consentire a qualsiasi switch di decidere come fargli raggiungere la propria destinazione, cioè ogni pacchetto contiene l'indirizzo di destinazione completo. Considerate la rete di esempio di Figura 3.4, in cui gli host hanno indirizzi A, B, C e così via. Per decidere come inoltrare un pacchetto, uno switch consulta una *tavola di inoltro* (forwarding table), a volte chiamata anche *tavola di instradamento* (routing table) di cui la Tabella 3.1 costituisce un esempio. Questa particolare tabella mostra le informazioni di inoltro che consentono allo switch 2 di inoltrare datagrammi nella rete usata come esempio. Costruire tale tabella quando si ha una mappa completa di una rete semplice come quella utilizzata in questo caso è molto semplice, si può immaginare che un operatore di rete la configuri in modo statico, mentre è molto più complesso creare le tabelle di inoltro in reti grandi e articolate, con topologie che si modificano dinamicamente e con più percorsi per diverse destinazioni. Tale complesso problema è noto come *instradamento* (routing) e sarà argomento della Sezione 4.2. Possiamo immaginare il routing come un processo che viene eseguito in background, in modo che quando si presenta la necessità di inoltrare un pacchetto di dati si abbiano già le corrette informazioni presenti nella tabella di inoltro per poter inoltrare il pacchetto.

Le reti datagram (o connectionless) hanno le seguenti caratteristiche:

- Un host può inviare un pacchetto ovunque in qualsiasi momento, perché ogni pacchetto che si presenta in ingresso ad uno switch può venire inoltrato immediatamente (nell'ip-

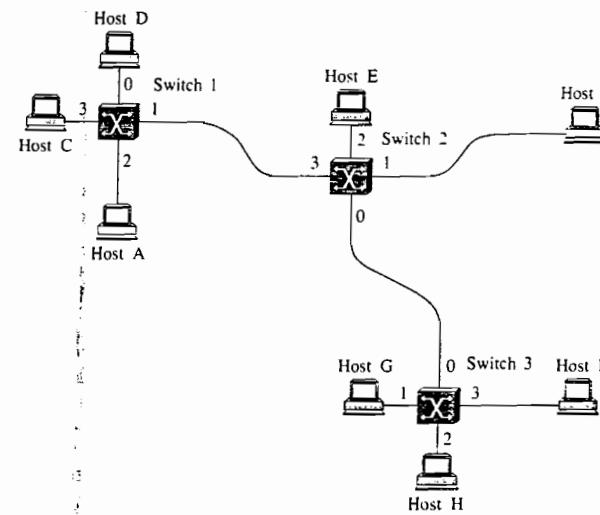


Figura 3.4 Inoltro di datagrammi: un esempio di rete.

Tabella 3.1 Tavella di inoltro per lo switch 2.

Destinazione	Porta
A	3
B	0
C	3
D	3
E	2
F	1
G	0
H	0

tesi che vi sia una tabella di inoltro popolata in modo corretto). Come vedremo, ciò contrasta con il comportamento della maggioranza delle reti orientate alla connessione, dove è richiesto lo stabilirsi di uno "stato di connessione" prima di poter inviare il primo pacchetto di dati.

- Quando un host invia un pacchetto, non ha alcun modo di sapere se la rete sia in grado di consegnarlo, né addirittura se l'host di destinazione sia operativo.
- Ciascun pacchetto viene inoltrato in modo indipendente dai pacchetti precedenti che possano essere stati inviati alla medesima destinazione, per cui due pacchetti consecutivi inviati dall'host A all'host B possono seguire due percorsi completamente diversi (magari per effetto di una modifica nella tabella di inoltro di qualche switch nella rete).

- Il malfunzionamento di uno switch o di una linea di connessione potrebbe non avere seri effetti sulle comunicazioni nella rete se è possibile trovare un percorso alternativo attorno alla sede del guasto e aggiornare di conseguenza le tabelle di inoltro.

Questa ultima caratteristica è particolarmente importante nella storia delle reti datagram: uno degli obiettivi principali di ARPANET, precursore di Internet, fu quello di sviluppare una tecnologia di rete che fosse resistente in ambiente militare, dove ci si può aspettare che nodi e linee vengano danneggiati da bombardamenti. Si giunse al progetto basato su datagrammi proprio per ottenere la capacità di aggirare le zone guaste.

3.1.2 Comutazione di circuito virtuale

Una tecnica di commutazione di pacchetto ampiamente utilizzata, che differisce in modo significativo dal modello datagram, utilizza il concetto di *circuito virtuale* (VC, virtual circuit). Tale approccio, chiamato anche *modello orientato alla connessione*, richiede che prima di inviare dati venga instaurata una connessione virtuale fra l'host sorgente e l'host destinatario. Per capire come funziona tutto ciò, considerate la Figura 3.5, dove l'host A vuole nuovamente inviare pacchetti all'host B. Possiamo immaginare che questa attività sia suddivisa in due stadi, nel primo dei quali (*connection setup*) si instaura la connessione, mentre nel secondo vengono trasferiti i dati. Li considereremo uno dopo l'altro.

Nella fase di instaurazione della connessione è necessario stabilire uno "stato di connessione" in ciascuno degli switch che si trovano tra l'host sorgente e l'host destinazione. Per una connessione, lo stato di connessione consiste in un'informazione inserita nella "tabella dei VC" di ciascuno switch attraversato dalla connessione stessa, informazione che contiene:

- un *identificatore del circuito virtuale* (VCI, virtual circuit identifier), che all'interno dello switch identifica univocamente la connessione e che verrà trasmesso nell'intestazione dei pacchetti che appartengono alla connessione stessa
- un'interfaccia di ingresso attraverso cui i pacchetti della VC arrivano allo switch
- un'interfaccia di uscita attraverso cui i pacchetti della VC escono dallo switch
- un VCI, eventualmente diverso, che verrà usato per i pacchetti uscenti

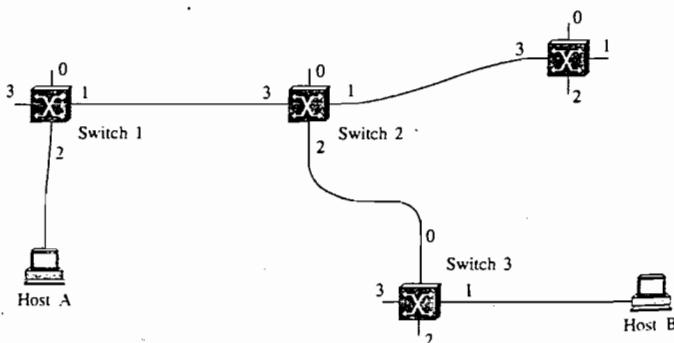


Figura 3.5 Un esempio di rete a circuito virtuale.

3.1 Comutazione e inoltro

La semantica di tali informazioni è la seguente: se dall'interfaccia di ingresso designata arriva un pacchetto che contiene nella propria intestazione il VCI designato, allora il pacchetto deve essere inviato all'interfaccia di uscita specificata dopo che il VCI specificato per l'uscita è stato inserito nella sua intestazione.

Note che una connessione virtuale viene identificata univocamente dalla combinazione del VCI dei pacchetti che vengono ricevuti dallo switch e dall'interfaccia da cui vengono ricevuti; ovviamente, è possibile che nello stesso istante vi siano molte connessioni virtuali instaurate nello switch. Osserviamo anche che, in generale, i valori di VCI di ingresso e di uscita non sono uguali, per cui il valore di VCI non ha il significato di un identificatore, globalmente univoco per la connessione, ma ha significato soltanto su una certa linea di connessione, per cui si dice che ha una visibilità localizzata alla linea (link local scope).

Ogni volta che viene creata una nuova connessione dobbiamo assegnare un nuovo valore di VCI per tale connessione in ciascuna linea che essa attraversa e dobbiamo anche garantire che il VCI scelto su una certa linea non sia attualmente utilizzato su tale linea da qualche connessione già instaurata.

Per l'instaurazione dello stato di connessione esistono due grandi categorie di approcci. Il primo prevede la presenza di un amministratore di rete che configuri lo stato, nel qual caso il circuito virtuale è "permanente", anche se, ovviamente, può anche venire eliminato dall'amministratore, per cui si dovrebbe pensare ad un circuito virtuale permanente (PVC, permanent virtual circuit) come ad un VC di lunga durata o configurato dall'amministratore. In alternativa, un host può inviare nella rete messaggi che provocano l'instaurazione dello stato: si parla, in questo caso, di *segnalazione* (signalling) ed i circuiti virtuali che ne risultano vengono detti *commutati* (SVC, switched virtual circuit). La caratteristica saliente di un circuito virtuale commutato è che un host lo può stabilire ed eliminare dinamicamente senza l'intervento di un amministratore di rete. Notate che un SVC si dovrebbe chiamare, più correttamente, "signalled" VC, perché è l'uso della segnalazione (e non della commutazione) che lo distingue da un PVC.

Immaginiamo che un amministratore di rete voglia creare manualmente una nuova connessione virtuale dall'host A all'host B. Per prima cosa l'amministratore deve identificare un percorso da A a B attraverso la rete: nella rete dell'esempio presentato in Figura 3.5 esiste uno solo di tali percorsi, ma in generale non è così. Successivamente l'amministratore sceglie per ciascuna linea un valore di VCI per la connessione che sia in quel momento inutilizzato: in questo esempio, supponiamo che venga scelto il valore di VCI uguale a 5 per la linea che va dall'host A allo switch 1 ed il valore 11 per la linea che va dallo switch 1 allo switch 2. In tal caso, nella tabella dei VC dello switch 1 deve essere presente una linea di configurazione uguale a quella mostrata in Tabella 3.2(a).

Analogamente, supponiamo che per identificare la connessione sulla linea che va dallo switch 2 allo switch 3 venga scelto il valore di VCI uguale a 7, ed il valore di VCI uguale a 4 per la linea che va dallo switch 3 all'host B. Di conseguenza, gli switch 2 e 3 devono avere la tabella dei VC configurata come mostrato nella Tabella 2.3. Notate che il valore di VCI "uscente" in uno switch è uguale al valore di VCI "entrante" nello switch successivo.

Una volta che siano state approntate le tabelle dei VC, si può procedere con la fase di trasferimento dei dati, come illustrato in Figura 3.6. Ogni volta che A vuole inviare un pacchetto all'host B, inserisce il valore di VCI uguale a 5 nell'intestazione del pacchetto e lo invia allo switch 1, che riceve il pacchetto dall'interfaccia 2 e usa la combinazione appropriata del numero di interfaccia e del valore di VCI contenuti nell'intestazione del pacchetto per trovare l'informazione corretta nella tabella dei VC. Come mostrato in Tabella 3.2, in questo

Tabella 3.2 Informazioni presenti nella tabella dei circuiti virtuali dello switch 1 (a), dello switch 2 (b) e dello switch 3 (c).

Interfaccia d'ingresso	VCI d'ingresso	Interfaccia d'uscita	VCI d'uscita
2	5	1	11
b)			
Interfaccia d'ingresso	VCI d'ingresso	Interfaccia d'uscita	VCI d'uscita
3			
c)			
Interfaccia d'ingresso	VCI d'ingresso	Interfaccia d'uscita	VCI d'uscita
0	7	1	4

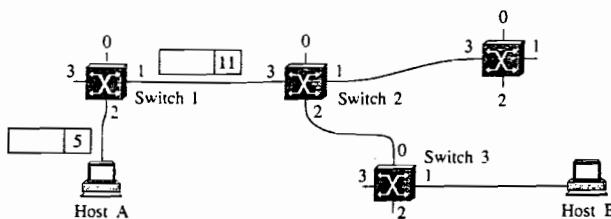


Figura 3.6 Invio di un pacchetto in una rete a circuito virtuale.

caso l'informazione indica allo switch 1 di inoltrare il pacchetto verso l'interfaccia uscente numero 1, inserendo nell'intestazione del pacchetto un valore di VCI uguale a 11. In questo modo il pacchetto arriverà allo switch 2 tramite la sua interfaccia 3, avendo un VCI uguale a 11. Lo switch 2 cerca l'interfaccia numero 3 ed il valore di VCI uguale a 11 nella propria tabella dei VC (presentata in Tabella 3.2b) ed invia il pacchetto verso lo switch 3 dopo aver aggiornato opportunamente il valore di VCI nell'intestazione del pacchetto, come mostrato in Figura 3.7. Questo procedimento continua finché il pacchetto arriva all'host B con un valore di VCI uguale a 4: tale informazione, nell'host B, identifica il pacchetto come proveniente dall'host A.

In una rete reale di dimensioni ragionevoli, il compito di configurare le tabelle dei VC in modo corretto in un gran numero di switch diventerebbe rapidamente eccessivo, se si usasse le procedure appena delineate, per cui si usa quasi sempre una qualche forma di segnalazione, anche per instaurare VC "permanenti". Nel caso dei PVC, la segnalazione viene fatta partire dall'amministratore di rete, mentre gli SVC vengono solitamente instaurati mediante la segnalazione di uno degli host. Vediamo ora come si potrebbe instaurare con la segnalazione proveniente dall'host lo stesso VC appena descritto.

Per iniziare il processo di segnalazione, l'host A invia un messaggio di configurazione (*setup*) alla rete, cioè allo switch 1. Tale messaggio di setup contiene, fra le altre cose, l'indirizzo di destinazione completo dell'host B, che deve essere raggiunto per creare lo stato di

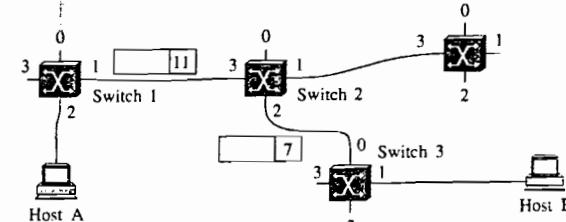


Figura 3.7 Un pacchetto si fa strada in una rete a circuito virtuale.

connessione necessario in ciascuno switch attraversato lungo il percorso. Possiamo vedere che far giungere il messaggio di setup a B è molto simile all'invio di un datagram a B, perché gli switch devono sapere verso quale uscita inviare il messaggio per fare in modo che, alla fine, raggiunga B. Per il momento ipotizziamo che gli switch abbiano abbastanza informazioni relative alla topologia della rete da poter scoprire come fare ciò, in modo che il messaggio di setup attraversi gli switch 2 e 3 per giungere, infine, all'host B.

Quando lo switch 1 riceve la richiesta di connessione, oltre ad inviarla allo switch 2 crea per questa nuova connessione una nuova riga di informazioni nella propria tabella dei VC, riga che è esattamente uguale a quella mostrata in Tabella 3.2. La differenza principale è che ora il compito di assegnare un valore di VCI che non sia usato su quella interfaccia viene svolto dallo switch: in questo esempio, lo switch sceglie il valore 5. Ora la tabella dei circuiti virtuali contiene la seguente informazione: "quando arriva sulla porta 2 un pacchetto con identificatore 5, invialo in uscita sulla porta 1". Rimane il problema che, in qualche modo, l'host A avrà bisogno di sapere di dover mettere il valore 5 nel campo VCI dei pacchetti che vuole inviare a B: vedremo più avanti una soluzione.

Quando lo switch 2 riceve il messaggio di setup, esegue una procedura simile e in questo esempio sceglie il valore 11 come VCI di ingresso. Analogamente, lo switch 3 sceglie il valore 7 come VCI di ingresso: ciascuno switch può scegliere il valore che preferisce, con l'unico vincolo che non sia attualmente utilizzato da qualche altra connessione sulla stessa porta di tale switch. Come abbiamo fatto notare in precedenza, i valori di VCI hanno una "visibilità limitata alla linea", cioè non hanno alcun significato globale.

Il messaggio di setup giunge finalmente all'host B: nell'ipotesi che B sia operativo e che voglia accettare una connessione dall'host A, anch'esso assegna un valore al VCI di ingresso, in questo caso il numero 4, valore che verrà usato da B per identificare tutti i pacchetti provenienti da A.

Ora, per completare la connessione, occorre dire a tutti quale valore di VCI è stato scelto dal nodo successivo lungo il percorso per questa connessione. L'host B invia una conferma dell'instaurazione della connessione allo switch 3, segnalando di aver scelto il valore 4 per VCI. Ora anche lo switch 3 può completare le informazioni relative a questa connessione nella propria tabella dei VC, perché ha saputo che il valore di VCI uscente deve essere 4. Lo switch 3 invia la conferma allo switch 2, specificando che VCI vale 7, quindi lo switch 2 invia la conferma allo switch 1, indicando che VCI vale 11. Infine, lo switch 1 invia la conferma all'host A, segnalandogli di usare il valore 5 per il VCI di questa connessione.

A questo punto tutti hanno le informazioni necessarie a far fluire traffico dall'host A all'host B. Ciascuno switch ha informazioni complete relative alla connessione nella propria tabella dei circuiti virtuali; inoltre, l'host A ha ricevuto conferma che tutto è stato predisposto

lungo il percorso fino all'host B. Le informazioni presenti a questo punto nelle tabelle dei circuiti virtuali dei tre switch traversati dalla connessione sono identiche a quelle configurate dall'amministratore nell'esempio precedente, ma l'intera procedura è avvenuta automaticamente in risposta al messaggio di segnalazione inviato da A. La fase di trasferimento dei dati può ora iniziare e sarà identica a quella utilizzata nel caso del PVC.

Quando l'host A non vuole più inviare dati all'host B, chiude la connessione inviando allo switch 1 un messaggio di chiusura di connessione (*teardown*). Lo switch elimina dalla propria tabella dei circuiti virtuali l'informazione corrispondente ed inoltra il messaggio agli altri switch lungo il percorso, che eliminano le informazioni allo stesso modo dalle proprie tabelle. A questo punto, se l'host A inviasse allo switch 1 un pacchetto con VCI uguale a 5, il pacchetto verrebbe eliminato come se la connessione non fosse mai esistita.

Dobbiamo notare alcune cose sulla commutazione a circuito virtuale:

- Dato che l'host A, prima di poter inviare il primo pacchetto di dati, deve attendere che la richiesta di connessione giunga all'altro capo della rete e torni indietro, prima che vengano inviati dai c'è un ritardo almeno uguale a RTT¹.
- Mentre la richiesta di connessione contiene l'indirizzo completo dell'host B (che potrebbe essere piuttosto lungo, essendo un identificatore globale nella rete), ciascun pacchetto di dati contiene soltanto un piccolo identificatore, che deve essere univoco soltanto su una linea, per cui, rispetto al modello a datagram, il valore di overhead dell'intestazione in ciascun pacchetto è ridotto.
- Se uno switch o una linea che fanno parte di un circuito virtuale hanno un malfunzionamento, la connessione si interrompe ed è necessario instaurarne una nuova. Inoltre, è necessario chiudere quella vecchia, per liberare lo spazio di memorizzazione nelle tabelle degli switch.
- Abbiamo trascurato di affrontare il problema di come uno switch decida verso quale linea inoltrare la richiesta di connessione. In sostanza, si tratta dello stesso problema visto nella costruzione delle tabelle di inoltro per reti datagram, che richiede un qualche tipo di *algoritmo di instradamento*. L'instradamento sarà descritto nella Sezione 4.2 e gli algoritmi ivi descritti sono, in generale, utilizzabili anche per instradare le richiede di setup, così come i datagrammi.

Uno degli aspetti vantaggiosi dei circuiti virtuali è che, nel momento in cui un host ottiene il permesso di inviare dati, riceve anche altre utili informazioni dalla rete: ad esempio, sa che esiste veramente un percorso verso il destinatario, e che tale destinatario è pronto ad accettare dati. Nel momento in cui viene instaurato un circuito virtuale è anche possibile assegnargli delle risorse. Ad esempio, una rete X.25 (una rete a commutazione di pacchetto che usa il modello orientato alla connessione) usa la seguente strategia in tre fasi:

1. Mentre il circuito virtuale viene inizializzato, gli si assegnano anche i buffer relativi.
2. Fra ciascuna coppia di nodi lungo il circuito virtuale viene eseguito il protocollo sliding window, a cui viene aggiunto anche il controllo di flusso per impedire al nodo sorgente di riempire i buffer allocati nel nodo ricevente.

¹ Questo non è propriamente vero. Alcune persone hanno proposto di inviare "ottimisticamente" un pacchetto di dati subito dopo aver inviato la richiesta di connessione, tuttavia la maggior parte delle attuali implementazioni attende il completamento dell'instaurazione della connessione prima di inviare dati.

3. Se un nodo non ha buffer a sufficienza da assegnare al circuito virtuale nel momento in cui viene elaborato il messaggio di richiesta di connessione, la connessione viene rifiutata.

Eseguendo queste tre fasi, ciascun nodo ha la garanzia di poter disporre di buffer a sufficienza per accodare i pacchetti che arriveranno su un certo circuito virtuale. Questa strategia basilea viene solitamente denominata *controllo di flusso segmento per segmento* (hop-by-hop).

Come termine di paragone, una rete datagram non ha alcuna fase di instaurazione della connessione e ciascuno switch elabora ogni pacchetto in modo indipendente dagli altri, rendendo meno banale il problema di come una rete datagram possa allocare le risorse in modo sensato. Al contrario, ciascun pacchetto in arrivo compete con tutti gli altri pacchetti per lo spazio all'interno dei buffer e se non c'è spazio disponibile il pacchetto in arrivo deve essere eliminato. Osserviamo, tuttavia, che anche in una rete basata sui datagrammi spesso accade che un host sorgente invii una sequenza di pacchetti allo stesso host destinazione, per cui è possibile che ciascuno switch operi delle distinzioni nell'insieme dei pacchetti che tiene in coda in un certo istante, basandosi sulla coppia di informazioni sorgente/destinazione, in modo da garantire che i pacchetti appartenenti a ciascuno di tali sottoinsiemi sia destinata una frazione equa dello spazio di memorizzazione nei propri buffer. Approfondiremo questa idea con grande dettaglio nel Capitolo 6.

Nel modello a circuito virtuale, potremmo immaginare di fornire a ciascun circuito una diversa *qualità di servizio* (QoS, Quality of Service). In questo campo, il termine "qualità di servizio" viene solitamente utilizzato per indicare che la rete fornisce all'utente una qualche forma di garanzia reativa alle prestazioni, cosa che a sua volta richiede che gli switch accantonino le risorse necessarie per soddisfare tale garanzia. Ad esempio, gli switch attraversati da un circuito virtuale potrebbero assegnare a tale circuito una percentuale dell'ampiezza di banda delle linee uscenti coinvolte. Come ulteriore esempio, una sequenza di switch lungo un circuito virtuale potrebbe garantire che i pacchetti di tale circuito non vengano ritardati (cioè accodati) per un tempo superiore ad una certa quantità. Torneremo sull'argomento della qualità di servizio nella Sezione 6.5.

Gli esempi più diffusi di tecnologie a circuito virtuale sono Frame Relay e Asynchronous Transfer Mode (ATM). ATM ha un certo numero di interessanti proprietà, di cui parleremo nella Sezione 3.3. Frame Relay è un'implementazione abbastanza semplice della tecnologia a circuito virtuale e la sua semplicità l'ha resa estremamente popolare, al punto che molti fornitori di servizi di rete offrono PVC in Frame Relay. Una delle applicazioni di Frame Relay è la costruzione di reti virtuali private (VPN, virtual private network), un argomento discusso nella Sezione 4.1.8.

La tecnologia Frame Relay fornisce alcune funzioni basilari per la qualità di servizio e per evitare la congestione, ma si tratta di funzionalità alquanto limitate se confrontate con X.25 e ATM. Il formato del pacchetto Frame Relay (in Figura 3.8) è un buon esempio di pacchetto usato per la commutazione a circuito virtuale.

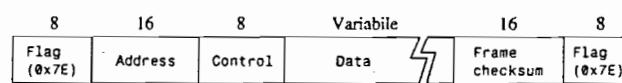


Figura 3.8 Formato del pacchetto Frame Relay.

Frame relay → utilizzato nelle VPN

Introduzione alla congestione

Ricordate la differenza tra contesa e congestione: si ha contesa quando più pacchetti vengono accodati in uno switch perché sono in lizza per uscire verso la stessa linea, mentre si ha congestione quando lo switch ha così tanti pacchetti accodati che esaurisce il proprio spazio di memorizzazione nei buffer e deve iniziare ad eliminare pacchetti. Torneremo sull'argomento della congestione nel Capitolo 6, dopo aver visto il protocollo di trasporto che fa parte dell'architettura di rete, tuttavia in questo momento possiamo osservare che la decisione di usare i circuiti virtuali o i datagrammi in una rete ha un impatto sulla modalità di gestione della congestione.

D'altra parte, supponete che ciascuno switch allochi buffer a sufficienza per gestire i pacchetti che appartengono a ciascun circuito virtuale che lo attraversa, come avviene nelle reti X.25. In questo caso, la rete ha risolto a priori il problema della congestione: uno switch non si troverà mai in una situazione in cui deve accodare più pacchetti di quanti ne possa contenere il proprio buffer, perché non consente fin dall'inizio l'instaurazione di una connessione a meno che non vi possa dedicare risorse sufficienti per evitare tale situazione. Il problema di questo approccio, tuttavia, è che si comporta in modo estremamente conservativo: è molto improbabile che tutti i circuiti si trovino a dover usare tutti i propri buffer nello stesso istante, per cui lo switch è potenzialmente sottoutilizzato.

D'altra parte, il modello datagram sembra invitare alla congestione: finché in uno switch non si esaurisce lo spazio disponibile nei buffer, non si può sapere se vi è sufficiente contesa da creare congestione. Se si giunge a quel punto, è troppo tardi per prevenire la congestione e l'unica cosa da fare è tentare di gestirla. Ovviamente, il vantaggio è che sarete in grado di utilizzare meglio gli switch, dato che non dovete riservare i buffer per una situazione di caso peggiore che accade raramente.

Come succede spesso, le cose non sono bianche o nere: ci sono vantaggi progettuali sia per l'eliminazione a priori della congestione (come nel modello X.25) sia per non far nulla finché non si verifica la congestione (come nel semplice modello datagram). Nel Capitolo 6 descriveremo alcuni di questi problemi progettuali.

3.1.3 Instradamento dalla sorgente (*source routing*)

Un terzo approccio alla commutazione, che non usa né i circuiti virtuali né i datagrammi convenzionali, è noto come **instradamento dalla sorgente** (*source routing*), il cui nome deriva dal fatto che l'**host sorgente** fornisce tutte le informazioni relative alla topologia della rete che sono necessarie per inoltrare un pacchetto all'interno della rete stessa.

Esistono vari modi per implementare il source routing. Uno di questi modi prevede di assegnare un numero a ciascuna uscita di ciascuno switch e di inserire tale numero nell'intestazione del pacchetto. La funzione di commutazione è quindi molto semplice: ogni volta che arriva un pacchetto ad un ingresso, lo switch legge nell'intestazione del pacchetto il numero della porta e inoltra il pacchetto verso tale uscita. Dato che, però, in generale vi sarà più di uno switch lungo il percorso tra la sorgente e la destinazione, l'intestazione del pacchetto deve contenere informazione sufficiente a consentire a ciascuno switch del percorso di determinare l'uscita verso cui deve essere inoltrato il pacchetto. Un modo di fare ciò sarebbe quello di inserire nell'intestazione un elenco ordinato di numeri di porta, facendo poi ruotare l'elenco in modo che il successivo switch lungo il percorso sia sempre il primo della lista. La Figura 3.9 illustra questa idea.

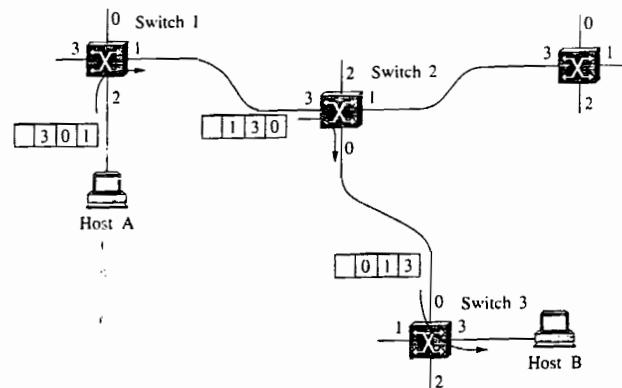


Figura 3.9 Instradamento dalla sorgente in una rete commutata (ciascuno switch legge il numero più a destra).

In questo esempio il pacchetto deve attraversare tre switch per arrivare all'host B partendo dall'host A. Nello switch 1 deve uscire dalla porta 1, nello switch successivo deve uscire dalla porta 0 e nel terzo switch deve uscire dalla porta 3, per cui l'intestazione originale, quando il pacchetto esce dall'host A, contiene l'elenco di porte (3, 0, 1), dove facciamo l'ipotesi che ciascuno switch legga l'elemento più a destra della lista. Per avere la certezza che lo switch successivo riceva l'informazione corretta, ciascuno switch fa ruotare l'elenco dopo aver letto il numero di propria competenza, per cui l'intestazione del pacchetto, quando lascia lo switch 1 in direzione dello switch 2, è (1, 3, 0); lo switch 2 esegue una rotazione analoga ed invia il pacchetto con l'elenco (0, 1, 3) nell'intestazione. Anche se non mostrato, lo switch 3 esegue un'ulteriore rotazione, ripristinando l'intestazione al suo stato iniziale, così come era stata inviata dall'host A.

Ci sono parecchie cose da notare in questo approccio. Per prima cosa, si ipotizza che l'host A abbia sufficienti conoscenze relative alla topologia della rete da poter comporre un'intestazione che ha al proprio interno le informazioni corrette per ciascuno switch che si trova lungo il percorso. Questo, in qualche modo, è analogo al problema della costruzione delle tabelle di inoltro in una rete datagram o al problema di dove inviare un pacchetto di setup in una rete a circuito virtuale. Secondo, osservate che non è possibile predeterminare la dimensione dell'intestazione, perché deve essere in grado di contenere una parola di informazioni per ciascuno switch del percorso. Ciò implica che, probabilmente, le intestazioni saranno di dimensione variabile, senza limite superiore, a meno che non siamo in grado di predire con assoluta certezza il numero massimo di switch attraverso cui un pacchetto si troverà a transitare. Terzo, esistono diverse varianti di questo approccio. Ad esempio, invece di ruotare l'intestazione, ciascuno switch potrebbe semplicemente eliminare il primo elemento dell'elenco quando lo usa, ma la rotazione ha un vantaggio rispetto all'eliminazione, in quanto l'host B riceve una copia dell'intestazione completa, che gli può essere d'aiuto nel trovare un percorso per raggiungere A. Un'altra alternativa, ancora, consiste nell'inserire nell'intestazione soltanto un puntatore alle informazioni relative alla "porta successiva", in modo che ciascuno switch aggiorni semplicemente tale puntatore senza ruotare l'intestazione, cosa che può essere più semplice da implementare. Potete vedere questi tre approcci in

Figura 3.10: in ciascun caso, le informazioni che devono essere lette dallo switch attuale sono rappresentate da A, mentre quelle per lo switch successivo sono rappresentate da B.

L'instradamento dalla sorgente si può utilizzare tanto in reti datagram quanto in reti a circuito virtuale. Ad esempio, Internet Protocol, che è un protocollo datagram, ha un'opzione per il source routing che consente di instradare dalla sorgente singoli pacchetti, mentre la maggior parte dei pacchetti vengono commutati come normali datagrammi. L'instradamento dalla sorgente viene utilizzato anche in reti a circuito virtuale come strumento per trasmettere la richiesta di setup iniziale lungo un percorso dalla sorgente alla destinazione.

Infine, notiamo che l'instradamento dalla sorgente non è facilmente scalabile: in qualsiasi rete ragionevolmente grande, è molto difficile per un host ottenere le informazioni sul percorso completo che gli servono per costruire le intestazioni corrette.

3.2 Comutatori per LAN e bridge

Avendo discusso alcune delle idee su cui si fonda la commutazione, ci concentriamo ora su alcune specifiche tecnologie di commutazione, iniziando da una categoria di commutatori che viene usata per inoltrare pacchetti fra reti locali a mezzo fisico condiviso, come Ethernet. Tali commutatori sono a volte chiamati con il loro nome più naturale, commutatori per reti locali (LAN switch), ma storicamente si trova anche il nome di *bridge*.

Supponete di avere due reti Ethernet da interconnettere. Un approccio potrebbe essere quello di provare ad inserire tra di esse un ripetitore, come descritto nel Capitolo 2, ma questa soluzione non funzionerebbe se facendo ciò si eccedessero i limiti fisici di Ethernet (ricorda che non vi possono essere più di due ripetitori fra ciascuna coppia di host e la lunghezza totale massima della rete è di 2500 m). Un'alternativa potrebbe essere quella di inserire un nodo fra le due reti Ethernet, con il compito di inoltrare i frame da una rete all'altra: questo nodo opererebbe in modalità promiscua, accettando tutti i frame trasmessi su entrambe le reti, in modo da poterli inoltrare da una all'altra.

Il nodo che abbiamo appena descritto viene tipicamente chiamato *bridge* (ponte) ed un insieme di reti locali connesse mediante uno o più bridge viene solitamente chiamato *rete locale estesa* (extended LAN). Nella loro forma più semplice, i bridge accettano semplicemente i frame delle reti locali che arrivano ai propri ingressi e li inoltrano verso tutte le proprie uscite. Tale semplice strategia è stata usata per i primi bridge, ma da allora è stata migliorata per fare dei bridge un meccanismo più efficiente per l'interconnessione di un insieme di reti locali. La restante parte di questa sezione fornisce i dettagli più interessanti al riguardo.

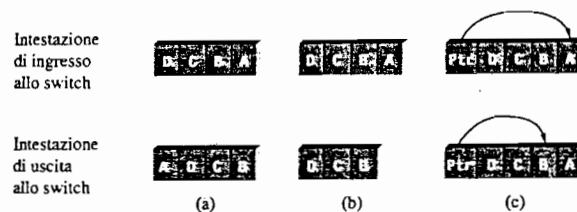


Figura 3.10 Tre modi per gestire le intestazioni nell'instradamento dalla sorgente: (a) rotazione; (b) eliminazione; (c) puntatore. Le etichette sono lette da destra a sinistra.

Commutazione ottica

Chi osservasse casualmente l'industria delle reti intorno all'anno 2000 potrebbe aver percepito che il tipo più interessante di commutatori siano quelli ottici. A dire il vero, la commutazione ottica è diventata una tecnologia importante proprio negli ultimi anni Novanta, per la concomitanza di vari fattori. Uno di tali fattori è stata la disponibilità commerciale di apparati per il multiplexing a divisione densa di lunghezza d'onda (DWDM, dense wavelength division multiplexing), che consente di inviare grandi quantità di informazioni lungo una singola fibra trasmettendo contemporaneamente un gran numero di lunghezze d'onda ottiche (o colori). Così, ad esempio, potreste inviare dati usando 100 o più lunghezze d'onda diverse, ciascuna delle quali potrebbe veicolare fino a 10 Gbps di dati.

Un secondo fattore è stata la disponibilità commerciale di amplificatori ottici. I segnali ottici vengono attenuati dal passaggio attraverso una fibra e dopo aver percorso una certa distanza (di circa 40 km) devono essere in qualche modo ripristinati. Prima di avere a disposizione gli amplificatori ottici, per ripristinare il segnale ottico era necessario interporre ripetitori lungo il percorso, convertendo il segnale ottico in un segnale elettronico digitale, per poi convertirlo nuovamente in segnale ottico. Prima di poter inviare i dati ad un ripetitore, occorre effettuarne il demultiplexing usando un terminale DWDM, per cui sarebbero necessari molti terminali DWDM soltanto per pilotare una singola coppia di fibre a lunga distanza. Gli amplificatori ottici, diversamente dai ripetitori, sono dispositivi analogici che amplificano qualsiasi segnale inviato lungo la fibra, anche se viene inviato mediante centinaia di lunghezze d'onda diverse. Gli amplificatori ottici hanno, quindi, reso molto più appetibile la tecnologia DWDM, perché ora una coppia di terminali DWDM possono scambiarsi dati anche a distanze di centinaia di chilometri. Inoltre, è anche possibile aggiornare i terminali DWDM agli estremi della fibra senza modificare gli amplificatori ottici posti lungo il percorso, dato che essi sono in grado di amplificare 100 lunghezze d'onda con la stessa facilità con cui ne amplificano 50.

Con la tecnologia DWDM e gli amplificatori ottici, è stato possibile costruire reti ottiche con capacità enormi, anche se per fare in modo che queste reti siano realmente utili è necessario un ulteriore tipo di dispositivo, il *commutatore ottico*. Oggi giorno la maggior parte dei dispositivi che vengono chiamati commutatori ottici (*optical switch*) svolgono, in realtà, la propria funzione di commutazione in modo elettronico e da un punto di vista dell'architettura sono più simili ai commutatori di circuito della rete telefonica che ai commutatori di pacchetto descritti in questo capitolo. Un tipico commutatore ottico ha un gran numero di interfacce che operano con il framing SONET ed è in grado di creare un canale SONET mettendo in comunicazione, ad incrocio, una interfaccia d'ingresso con un'interfaccia d'uscita. In questo modo, con un commutatore ottico è possibile creare canali SONET dal punto A al punto B passando attraverso il punto C, anche se non esiste una fibra che connetta direttamente A e B: è sufficiente che ci sia un percorso in fibra da A a C, un commutatore nel punto C ed un altro percorso in fibra da C a B. In questo senso, un commutatore ottico assomiglia ai commutatori di Figura 3.5, perché crea l'illusione di una connessione fra due punti anche se non esiste una connessione fisica diretta tra di essi. Tuttavia, i commutatori ottici non forniscono circuiti virtuali, ma circuiti "reali", cioè canali SONET. Esistono anche alcuni tipi di commutatori

(continua)

(seguito)

ottici più recenti che rendono disponibile un canale ottico ininterrotto tra il punto A e il punto B, usando specchi microscopici per deflettere tutta la luce proveniente da una porta verso un'altra porta.

In questo libro non trattiamo diffusamente le reti ottiche, in parte per problemi di spazio. Per molti aspetti pratici, potete pensare alle reti ottiche come ad una parte di infrastruttura che consente alle compagnie telefoniche di fornire linee di collegamento SONET o altri tipi di circuito nel luogo e nel momento in cui ne avete bisogno. Tuttavia, vale la pena di sottolineare come molte delle tecnologie che verranno discusse nel seguito del libro, come i protocolli di instradamento e il Multiprotocol Label Switching, hanno applicazioni nel mondo delle reti ottiche.

Noteate che un bridge soddisfa alla definizione di switch che abbiamo fornito nella sezione precedente: un dispositivo a più ingressi e più uscite che trasferisce pacchetti da un ingresso verso una o più uscite. Ricordate anche che in questo modo possiamo anche aumentare l'ampiezza di banda totale di una rete: ad esempio, mentre un singolo segmento di Ethernet può trasportare soltanto 10 Mbps di traffico, un bridge Ethernet può veicolare fino a $n \times 10$ Mbps di traffico, dove n è il numero di porte (di ingresso e di uscita) del bridge stesso.

3.2.1 Bridge ad apprendimento (*learning bridge*)

La prima ottimizzazione che possiamo mettere in opera in un bridge consiste nell'osservare che non c'è bisogno che il bridge inoltri tutti i frame che riceve. Considerate il bridge di Figura 3.11. Ogni volta che un frame inviato dall'host A e indirizzato all'host B arriva sulla porta 1, non c'è bisogno che il bridge lo inoltri sulla porta 2. La domanda, perciò, è: come può un bridge apprendere su quale porta risiedono i vari host?

Una delle opzioni disponibili sarebbe quella che prevede un intervento umano per caricare nel bridge una tabella simile a quella di Tabella 3.3. Successivamente, ogni volta che il bridge riceve un frame sulla porta 1 che sia indirizzato all'host A, non lo inoltra sulla porta 2, in quanto ciò non è necessario perché l'host A riceve già il frame direttamente, trovandosi

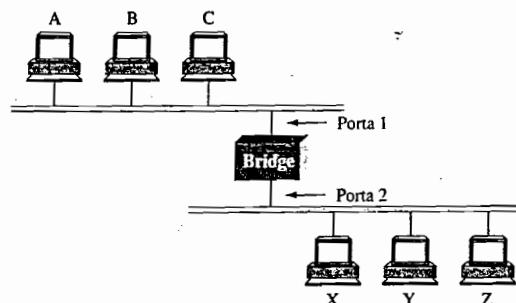


Figura 3.11 Un bridge ad apprendimento.

Tabella 3.3 Tabella di inoltro gestita da un bridge.

Host	Porta
A	1
B	1
C	1
X	2
Y	2
Z	2

esso sulla rete locale connessa alla porta 1. Ogni volta che, invece, viene ricevuto sulla porta 2 un frame indirizzato all'host A, esso viene inoltrato verso la porta 1.

Notate che un bridge che usasse questa tabella userebbe il modello datagram (o privo di connessione) per l'inoltro descritto nella Sezione 3.1.1: ciascun pacchetto contiene un indirizzo globale ed il bridge decide verso quale uscita inoltrare il pacchetto cercando tale indirizzo in una tabella.

Affidare ad una persona il compito di gestire questa tabella è molto oneroso, specialmente se si considera che esiste un semplice accorgimento che consente al bridge di ricavare autonomamente queste informazioni. L'idea è che ciascun bridge ispezioni anche l'indirizzo della sorgente di tutti i frame che riceve. In questo modo, quando l'host A invia un frame ad un host che si trova su un qualsiasi lato del bridge, il bridge riceve tale frame e memorizza il fatto che il frame proveniente dall'host A è stato ricevuto dalla porta 1. In questo modo, il bridge è in grado di costruire una tabella proprio uguale a quella presentata in Tabella 3.3.

Quando un bridge viene acceso, tale tabella è vuota e le informazioni vi vengono aggiunte a mano a mano che arrivano pacchetti. Inoltre, a ciascuna informazione presente nella tabella viene associato un temporizzatore, scaduto il quale il bridge elimina l'informazione dalla tabella stessa: questo serve a gestire il fatto che un host (e, di conseguenza, il suo indirizzo di rete locale) può venire spostato da una rete all'altra. Di conseguenza, questa tabella non è necessariamente completa: se il bridge riceve un frame indirizzato ad un host che non è presente nella tabella, semplicemente il frame viene inoltrato verso tutte le porte di uscita. In altre parole, questa tabella è soltanto un'ottimizzazione che filtra alcuni frame e non è richiesta per un funzionamento corretto.

Implementazione

Il codice che implementa l'algoritmo di apprendimento del bridge è abbastanza semplice e qui lo delineiamo. La struttura `BridgeEntry` definisce una riga di informazioni all'interno della tabella di inoltro, righe che vengono memorizzate in una struttura di tipo `Map` (che consente le operazioni `mapCreate`, `mapBind` e `mapResolve`) per consentire l'efficiente recupero delle informazioni quando arrivano pacchetti provenienti da sorgenti già presenti nella tabella. La costante `MAX_TTL` specifica il tempo di permanenza di un'informazione nella tabella.

```

#define BRIDGE_TAB_SIZE 1024 /* dimensione massima della tabella */
#define MAX_TTL 120 /* tempo (in secondi) prima che una
                     informazione in tabella venga eliminata */
  
```

```

typedef struct {
  
```

```

    MacAddr destination; /* indirizzo MAC di un nodo */
    int ifnumber; /* interfaccia per raggiungerlo */
    u_short TTL; /* tempo rimasto */
    Binding binding; /* associazione nella Map */
} BridgeEntry;

int numEntries = 0;
Map bridgeMap = mapCreate(BRIDGE_TAB_SIZE, sizeof(BridgeEntry));

```

La funzione updateTable aggiorna la tabella di inoltro quando arriva un nuovo pacchetto e riceve come argomenti l'indirizzo MAC della sorgente contenuto nel pacchetto ed il numero dell'interfaccia che ha ricevuto il pacchetto. Un'altra funzione, che qui non viene mostrata, viene invocata ad intervalli regolari e scandisce tutte le informazioni presenti nella tabella, decrementando i campi TTL (*time to live*) ed eliminando quelle informazioni il cui TTL è diventato uguale a 0. Notate che TTL viene reimpostato al valore MAX_TTL ogni volta che arriva un pacchetto che rinnova la validità di un'informazione già esistente nella tabella: l'interfaccia mediante la quale è raggiungibile la destinazione di un pacchetto viene aggiornata in base al pacchetto ricevuto più recentemente.

```

void
updateTable (MacAddr src, int inif)
{
    BridgeEntry *b;

    if (mapResolve(bridgeMap, &src, (void **)&b) == FALSE)
    {
        /* questo indirizzo non è nella tabella, quindi cerca di
         * aggiungerlo */
        if (numEntries < BRIDGE_TAB_SIZE)
        {
            b = NEW(BridgeEntry);
            b->binding = mapBind(bridgeMap, &src, b);
            /* usa l'indirizzo della sorgente del pacchetto come
             * indirizzo di destinazione nella tabella */
            b->destination = src;
            numEntries++;
        }
        else
        {
            /* non c'è spazio per questo indirizzo nella tabella,
             * non importa */
            return;
        }
    }
    /* reimposta TTL e usa l'interfaccia d'ingresso più recente */
    b->TTL = MAX_TTL;
    b->ifnumber = inif;
}

```

Note che, nel caso in cui la tabella sia piena, questa implementazione adotta una strategia molto semplice: non aggiunge il nuovo indirizzo. Ricordate che la completezza della tabella non è necessaria per l'inoltro corretto: stiamo solamente ottimizzando le prestazioni. Se qualche informazione presente nella tabella non è utilizzata, prima o poi scadrà il relativo temporizzatore e verrà eliminata dalla tabella, creando spazio per nuove informazioni. Un approccio alternativo, trovando la tabella piena, sarebbe stato quello di invocare una qualche forma di algoritmo utilizzato nella gestione delle memorie cache; ad esempio, potremmo localizzare ed eliminare l'informazione avente il più piccolo valore di TTL, per far posto alla nuova informazione.

3.2.2 Algoritmo ad albero di copertura (spanning tree)

La strategia precedente funziona veramente bene fino al momento in cui la rete locale estesa non presenta anelli (o cicli, *loop*), nel qual caso, invece, fallisce in modo drammatico: i frame circolano all'infinito nella rete locale estesa. La situazione è assai facile da comprendere esaminando l'esempio rappresentato in Figura 3.12, dove, ad esempio, i bridge B1, B4 e B6 formano un anello. Come è possibile che una rete locale estesa abbia un anello? Una possibilità è dovuta al fatto che la rete sia gestita da più amministratori, ad esempio perché serve diversi dipartimenti di un'organizzazione. In una situazione di questo tipo, può darsi che non ci sia una singola persona che conosce l'intera configurazione della rete, per cui un bridge che chiude un anello potrebbe essere stato aggiunto inconsapevolmente. Un secondo, più probabile scenario prevede che gli anelli vengano inseriti nella rete di proposito, per garantire un certo livello di ridondanza in caso di guasto.

Qualunque sia la causa, i bridge devono essere in grado di gestire correttamente le configurazioni ad anelli. Il problema viene risolto facendo eseguire ai bridge un algoritmo distribuito ad albero di copertura (spanning tree). Se immaginate che la LAN estesa venga rappresentata

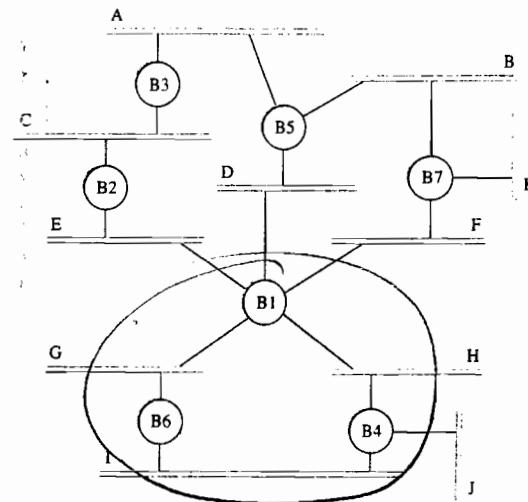


Figura 3.12 Rete locale estesa con anelli.

ia da un grafo che possa avere dei cicli, allora uno spanning tree è un sottografo di tale grafo che ne copre (contiene) tutti i vertici senza avere cicli. In altre parole, uno spanning tree di un grafo ha gli stessi vertici del grafo originario, ma gli mancano alcuni rami. Ad esempio, la Figura 3.13 mostra sulla sinistra un grafo ciclico e sulla destra uno dei molti possibili spanning tree.

L'algoritmo ad albero di copertura, sviluppato in Digital da Radia Perlman, è un protocollo utilizzato da un insieme di bridge per accordarsi su uno spanning tree di una particolare LAN estesa (la specifica IEEE 802.1 per i bridge di reti locali è basata su questo algoritmo). In sostanza, ciò significa che ciascun bridge decide verso quali porte inoltrare o non inoltrare frame; in un certo senso, la rete locale estesa viene ricondotta ad avere la topologia di un albero aciclico rimuovendo alcune porte². Addirittura, è possibile che un intero bridge non partecipi all'inoltro dei frame, cosa che può sembrare assurda se si pensa che una delle ragioni principali per cui si inseriscono intenzionalmente anelli in una rete è per creare ridondanza, ma l'algoritmo è dinamico, per cui i bridge sono sempre pronti a riconfigurarsi in un nuovo spanning tree nel caso in cui un bridge abbia un guasto.

L'idea principale dello spanning tree consiste nel fatto che sono i bridge a scegliere le porte verso le quali inoltreranno i frame. L'algoritmo seleziona le porte nel modo seguente. Ciascun bridge ha un identificatore univoco: noi useremo le etichette B1, B2, B3 e così via. Per prima cosa l'algoritmo nomina radice dello spanning tree il bridge avente l'identificatore più piccolo: come avverrà esattamente questa nomina verrà descritto in seguito. Il bridge radice inoltra sempre i frame verso tutte le proprie porte. Successivamente, ciascun bridge calcola il percorso più breve verso la radice e prende nota di quale delle proprie porte si trova su tale percorso: tale porta viene selezionata in qualità di percorso preferito dal bridge verso la radice. Infine, tutti i bridge connessi ad una certa LAN nominano un unico bridge designato, che avrà il compito di inoltrare i frame verso il bridge radice. In ciascuna LAN, il bridge designato è quello più prossimo alla radice: se due o più bridge sono equidistanti dalla radice,

①
ROOT
BRIDGE

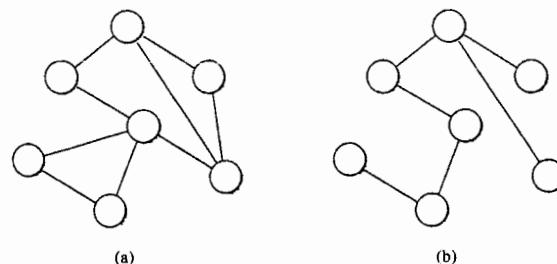


Figura 3.13 Esempio di un grafo ciclico (a) e di un corrispondente spanning tree (b).

² Rappresentare una rete locale estesa con un grafo astratto non è semplice. In sostanza, i bridge e le singole reti locali rappresentano i vertici del grafo e le porte corrispondono ai suoi rami. Tuttavia, lo spanning tree che stiamo calcolando per tale grafo deve contenere necessariamente soltanto quei nodi che corrispondono a reti, per cui è possibile che nodi che corrispondono a bridge risultino disconnessi dal resto del grafo, situazione che corrisponde alla realtà in cui tutte le porte che connettono un bridge alle varie reti vengono eliminate dall'algoritmo.

viene designato quello con l'identificatore più piccolo. Dato che ciascun bridge è connesso a più di una LAN, esso partecipa alla nomina del bridge designato in ciascuna LAN di cui fa parte, per cui, in pratica, ciascun bridge decide se essere o meno il bridge designato relativamente a ciascuna delle proprie porte, in modo da inoltrare i frame soltanto verso quelle porte per le quali è il bridge designato.

La Figura 3.14 mostra lo spanning tree corrispondente alla LAN estesa rappresentata in Figura 3.12. In questo esempio B1 è il bridge radice, poiché ha l'identificatore minore. Notate che sia B3 sia B5 sono connessi alla rete locale A, ma il bridge designato è B5 perché è più vicino alla radice. Analogamente, sia B5 che B7 sono connessi alla rete locale B, ma in questo caso il bridge designato è B5, perché ha l'identificatore minore, essendo entrambi alla medesima distanza da B1.

Mentre una persona può guardare alla rete locale estesa di Figura 3.12 e calcolare lo spanning tree rappresentato in Figura 3.14 secondo le regole sopra descritte, i bridge della rete non possono concedersi il lusso di poter vedere la topologia dell'intera rete, senza considerare il fatto di dover guardare dentro agli altri bridge per scoprire i loro identificatori. I bridge, invece, devono scambiarsi messaggi di configurazione e poi decidere, basandosi su tali messaggi, di essere la radice oppure no, oppure di essere il bridge designato per una certa rete.

Nello specifico, i messaggi di configurazione contengono tre informazioni:

1. l'identificatore del bridge che invia il messaggio
2. l'identificatore del bridge radice secondo le informazioni in possesso del bridge che invia il messaggio
3. la distanza, misurata in hop (segmenti), fra il bridge radice e il bridge che invia il messaggio

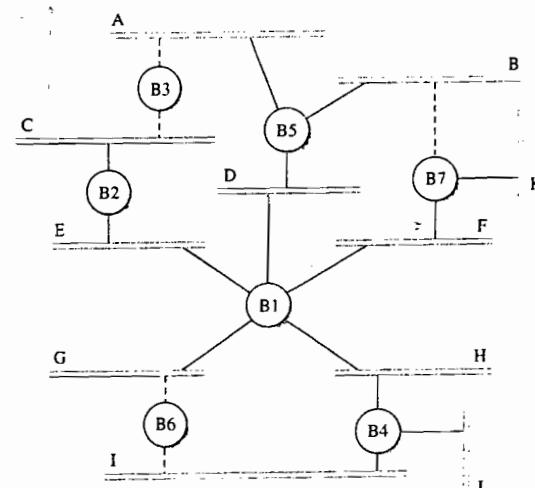


Figura 3.14 Spanning tree con alcune porte non selezionate.

Ciascun bridge memorizza il messaggio di configurazione "migliore" che ha visto su ciascuna delle proprie porte (il concetto di "migliore" è definito più avanti), tenendo conto sia dei messaggi che ha ricevuto sia di quelli che ha inviato.

All'inizio ciascun bridge ritiene di essere la radice, per cui invia un messaggio di configurazione su ciascuna delle proprie porte identificandosi come radice e segnalando una distanza dalla radice uguale a 0. Quando riceve un messaggio di configurazione tramite una porta, il bridge verifica se tale nuovo messaggio è migliore del messaggio di configurazione in quel momento memorizzato come migliore per quella porta. Il nuovo messaggio di configurazione è considerato "migliore" dell'informazione fino a quel momento memorizzata se:

- identifica una radice con un identificatore minore, oppure
- identifica una radice con lo stesso identificatore ma con una distanza minore, oppure
- l'identificatore della radice e la distanza dalla radice sono uguali, ma il bridge che ha inviato il messaggio ha un identificatore minore.

Se il nuovo messaggio è migliore dell'informazione memorizzata fino a quel momento, il bridge elimina la vecchia informazione e memorizza quella nuova, dopo aver aggiunto 1 al campo di distanza dalla radice, poiché il bridge in esame è di un segmento più distante dalla radice di quanto lo sia il bridge che ha inviato il messaggio.

Quando un bridge riceve un messaggio di configurazione da cui deduce di non essere la radice (cioè il messaggio proviene da un bridge avente un identificatore minore); il bridge smette di generare propri messaggi di configurazione e, invece, inoltra soltanto i messaggi di configurazione ricevuti da altri bridge, dopo aver aggiunto 1 al campo di distanza dalla radice. Allo stesso modo, quando un bridge riceve un messaggio di configurazione da cui deduce di non essere il bridge designato per quella porta (cioè il messaggio proviene da un bridge che si trova più vicino alla radice, oppure egualmente vicino ma con un identificatore minore), il bridge smette di inviare messaggi di configurazione da tale porta. In questo modo, quando il sistema si stabilizza, c'è soltanto il bridge radice che continua a generare messaggi di configurazione, mentre gli altri bridge inoltrano tali messaggi soltanto verso quelle porte per le quali sono i bridge designati.

Per fare un esempio concreto, considerate ciò che accadrebbe in Figura 3.14 se fosse appena tornata la tensione di alimentazione nell'edificio che ospita la rete, in modo che tutti i bridge si siano accesi nello stesso istante, iniziando a dichiarare di essere la radice. Indichiamo con (Y, d, X) un messaggio di configurazione proveniente dal nodo X , con il quale tale nodo dichiara di trovarsi alla distanza d dalla radice Y . Concentrandoci sull'attività del nodo $B3$, accadrebbe la sequenza di eventi che segue:

1. $B3$ riceve $(B2, 0, B2)$.
2. Poiché $2 < 3$, $B3$ accetta $B2$ come radice.
3. $B3$ aggiunge 1 alla distanza dichiarata da $B2$ (0), per cui invia $(B2, 1, B3)$ verso $B5$.
4. Nel frattempo, $B2$ accetta $B1$ come radice, perché ha un identificatore minore, e invia $(B1, 1, B2)$ verso $B3$.
5. $B5$ accetta $B1$ come radice e invia $(B1, 1, B5)$ verso $B3$.
6. $B3$ accetta $B1$ come radice e nota che sia $B2$ sia $B5$ si trovano più vicini di $B3$ stesso alla radice, per cui $B3$ smette di inoltrare messaggi verso entrambe le proprie interfacce.

3.2 Comutatori per LAN e bridge

Ciò lascia $B3$ con entrambe le porte non selezionate, come mostrato in Figura 3.14.

Anche dopo la stabilizzazione del sistema, il bridge radice continua ad inviare periodicamente messaggi di configurazione, e gli altri bridge continuano ad inoltrare tali messaggi come descritto nel paragrafo precedente. Se un particolare bridge dovesse avere un guasto, il bridge a valle non riceverebbe più i messaggi di configurazione e, dopo aver atteso per un periodo di tempo prefissato, inizierebbero nuovamente a dichiarare di essere la radice e l'algoritmo appena descritto partirebbe nuovamente per nominare una nuova radice e nuovi bridge designati.

Una cosa importante da notare è che, sebbene l'algoritmo sia in grado di riconfigurare lo spanning tree quando un bridge smette di funzionare, non è in grado di inoltrare frame lungo percorsi alternativi allo scopo di aggirare un bridge congestionato.

3.2.3 Broadcast e multicast

La discussione precedente si è concentrata sulla modalità di inoltro dei frame di tipo unicast da una rete locale ad un'altra. Dato che l'obiettivo di un bridge è quello di estendere in modo trasparente una rete locale coinvolgendo più reti, e poiché la maggior parte delle reti locali forniscono supporto ai frame broadcast e multicast, i bridge devono anch'essi fornire supporto per queste due caratteristiche. Per la trasmissione broadcast è semplice: ciascun bridge inoltra un frame che abbia l'indirizzo broadcast come destinazione verso tutte le proprie porte attive (selezionate) diverse da quella da cui ha ricevuto il frame.

La trasmissione multicast si può realizzare esattamente nello stesso modo e ciascun host decide per conto proprio se accettare il messaggio oppure no, e in pratica si fa proprio così, ma notate che è possibile fare di meglio, perché non necessariamente tutte le reti locali appartenenti ad una LAN estesa hanno almeno un host che appartiene ad un certo gruppo di multicast. In particolare, è possibile estendere l'algoritmo a spanning tree in modo da eliminare le reti sulle quali non è necessario inoltrare i frame di tipo multicast. Considerate un frame inviato al gruppo M da un host appartenente alla rete locale A di Figura 3.14. Se nella rete locale J non esistono host che appartengono al gruppo M, non è necessario che il bridge $B4$ inoltri i frame verso tale rete. D'altra parte, il fatto che nella rete locale H non vi siano host che appartengono al gruppo M non implica necessariamente che il bridge $B1$ possa evitare di inoltrare frame verso la rete H: dipende da fatto che nelle reti locali I o J vi siano membri del gruppo M oppure no.

Come fa un bridge a sapere se deve inoltrare un frame di tipo multicast verso una certa porta? Lo apprende esattamente nello stesso modo in cui apprende di dover inoltrare verso una certa porta i frame di tipo unicast: osservando l'indirizzo della sorgente dei pacchetti che riceve da quella porta. Il problema, però, è che tipicamente non sono i gruppi multicast ad essere sorgenti di frame, per cui dobbiamo usare un piccolo trucco: ciascun host che sia membro del gruppo M deve inviare periodicamente un frame con l'indirizzo del gruppo M nel campo di intestazione del frame che indica la sorgente e con l'indirizzo multicast che identifica i bridge nel campo che indica la destinazione.

Notate che l'estensione per la gestione del multicast che abbiamo appena descritto, nonostante sia stata proposta, non è adottata diffusamente; oggi, nelle reti locali estese il multicast viene, invece, implementato esattamente nello stesso modo in cui si realizza il broadcast.

3.2.4 Limiti dei bridge

La soluzione basata sui bridge che abbiamo appena visto è adatta ad un utilizzo in situazioni abbastanza limitate, tipicamente per connettere una manciata di reti locali simili. I principali limiti dei bridge diventano evidenti quando si considera il problema della scalabilità e della eterogeneità.

Per quanto riguarda il problema della scalabilità, connettere con bridge più di un numero limitato di reti locali non è realistico (dove in pratica "numero limitato" significa "qualche decina"), innanzitutto perché l'algoritmo che calcola lo spanning tree ha prestazioni lineari ed è quindi inutile considerare una struttura gerarchica nella rete locale estesa. Un secondo problema è dovuto ai frame broadcast, che vengono tutti inoltrati dai bridge: mentre è ragionevole che tutti gli host all'interno di un ambiente limitato (come, ad esempio, un dipartimento) vedano reciprocamente i messaggi broadcast, è assai improbabile che tutti gli host di una struttura più estesa (come può essere una grande società o un ateneo universitario) debbano occuparsi dei reciproci messaggi broadcast. Detto in altro modo, la modalità broadcast non è scalabile e, di conseguenza, le reti locali estese non sono scalabili.

Un approccio che migliora la scalabilità delle LAN estese è la rete locale *virtuale* (VLAN, virtual LAN), che consente ad una singola LAN estesa di essere suddivisa in diverse LAN che appaiono separate. A ciascuna LAN virtuale viene assegnato un identificatore (a volte chiamato *colore*) e i pacchetti possono viaggiare da un segmento ad un altro soltanto se entrambi i segmenti hanno il medesimo identificatore, limitando così il numero di segmenti di una LAN estesa che riceveranno un pacchetto broadcast.

Possiamo vedere come funzionano le LAN virtuali con un esempio. La Figura 3.15 mostra quattro host appartenenti a quattro diversi segmenti di rete locale. In assenza di VLAN, tutti i pacchetti broadcast provenienti da qualsiasi host raggiungeranno tutti gli altri host. Supponiamo ora di definire una VLAN, chiamata VLAN 100, composta dai segmenti connessi agli host W e X, ed un'altra VLAN, VLAN 200, composta dai segmenti connessi agli host Y e Z. Per far ciò dobbiamo configurare un identificatore di VLAN (VLAN ID) in ciascuna porta dei bridge B1 e B2; la linea di collegamento posta tra B1 e B2 appartiene ad entrambe le VLAN.

Quando un pacchetto inviato dall'host X arriva al bridge B2, il bridge osserva che il pacchetto proviene da una porta configurata come appartenente alla VLAN 100, per cui inserisce un'intestazione VLAN fra l'intestazione Ethernet e il carico utile del pacchetto. La parte interessante dell'intestazione VLAN è il VLAN ID, che, in questo caso, ha il valore 100. A questo punto il bridge applica le proprie normali regole per l'inoltro del pacchetto, con la restrizione aggiuntiva che il pacchetto non verrà inviato verso un'interfaccia che non

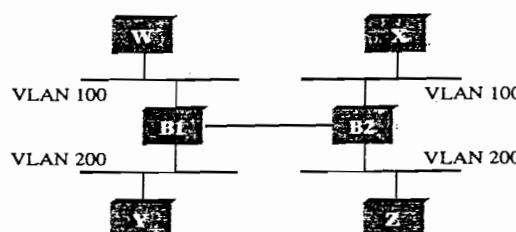


Figura 3.15 Due reti locali virtuali che condividono una dorsale comune.

3.3 Comutazione di celle (ATM)

sia parte della VLAN 100. Di conseguenza, non accadrà mai che il pacchetto, nemmeno se fosse un pacchetto broadcast, venga inviato in direzione dell'host Z, che appartiene alla VLAN 200. Il pacchetto viene, però, inoltrato verso il bridge B1, che obbedisce alle stesse regole e, quindi, inoltra il pacchetto all'host W ma non all'host Y.

Una caratteristica interessante delle reti locali virtuali è la possibilità di modificare la topologia logica senza spostare alcun collegamento fisico né modificare alcun indirizzo. Ad esempio, se volessimo fare in modo che il segmento connesso all'host Z appartenga alla VLAN 100, ottenendo quindi una rete locale virtuale con gli host X, W e Z, dovremmo soltanto modificare una parte della configurazione del bridge B2.

Per quanto riguarda il problema dell'eterogeneità, i bridge sono piuttosto limitati in merito al tipo di reti che possono interconnettere. In particolare, i bridge sfruttano l'intestazione dei frame della rete, per cui possono operare soltanto con reti che abbiano esattamente lo stesso formato per gli indirizzi. Di conseguenza, si possono usare i bridge per connettere reti Ethernet a reti Ethernet, reti 802.5 a reti 802.5 e reti Ethernet ad anelli 802.5, dato che entrambi questi tipi di rete usano lo stesso formato di indirizzo a 48 bit, ma i bridge non estendono in modo immediato la propria operatività ad altri tipi di reti, come le reti ATM³.

Nonostante i limiti, i bridge hanno un ruolo molto importante nel panorama delle reti. Il loro vantaggio principale è quello di consentire la connessione in modo trasparente di più LAN: si possono connettere reti locali senza che gli host ai capi della rete debbano eseguire protocolli aggiuntivi (e, addirittura, senza che ne siano consapevoli, per ciò che può contare). L'unica potenziale eccezione si ha quando gli host devono annunciare la propria appartenenza ad un gruppo multicast, come descritto nella Sezione 3.2.3.

In ogni caso, notate che questa trasparenza può essere pericolosa. Se un host o, più precisamente, il protocollo di trasporto e di applicazione in esecuzione in quell'host sono programmati in funzione della propria appartenenza ad un'unica rete locale, l'inserimento di bridge tra gli host sorgente e destinazione può avere conseguenze inattese. Ad esempio, se un bridge diviene congestionato, può avere la necessità di eliminare alcuni frame, mentre, al contrario, è molto raro che una singola rete Ethernet perda anche un solo frame. Un ulteriore esempio riguarda la latenza fra una coppia di host in una rete locale estesa, che diventa più elevata e soggetta a maggiore variabilità, mentre, al contrario, le limitazioni fisiche imposte ad un singolo segmento di rete Ethernet rendono la latenza piccola e prevedibile. Come ultimo esempio, in una LAN estesa può accadere (anche se è improbabile) che l'ordine dei frame venga modificato, cosa che non può accadere in una Ethernet singola. La considerazione conclusiva è che progettare software di rete facendo l'ipotesi che verrà eseguito su un singolo segmento di rete Ethernet non è salutare: capita che ci siano dei bridge.

3.3 Comutazione di celle (ATM)

Un'altra tecnologia di commutazione che merita particolare attenzione è la *modalità di trasferimento asincrono* (ATM, asynchronous transfer mode). La tecnologia ATM assunse importanza negli anni Ottanta e nei primi anni Novanta per vari motivi, non ultimo il fatto che sia stata adottata dall'industria telefonica, che storicamente era stata assai poco attiva nelle

³ Come vedremo nella Sezione 3.3.5, esistono tecniche che rendono le reti ATM molto più simili a LAN "convenzionali" come Ethernet, e i bridge hanno un proprio ruolo in tale contesto.

comunicazioni per dati, tranne che come fornitore di linee di collegamento mediante le quali altri attori avevano costruito reti. Si può anche dire che la tecnologia ATM si sia trovata nel posto giusto al momento giusto, in qualità di tecnologia di commutazione ad alta velocità apparsa sulla scena proprio quando le tecnologie a mezzo condiviso, come Ethernet e 802.5, iniziavano ad essere un po' troppo lente per molti utenti di reti di calcolatori. Per certi versi ATM è una tecnologia in competizione con la commutazione Ethernet, ma le aree di applicazione per queste due tecnologie sono solo parzialmente sovrapposte.

La tecnologia ATM è a commutazione di pacchetto e orientata alla connessione, che è come dire che usa i circuiti virtuali in un modo molto simile a quello descritto nella Sezione 3.1.2. Secondo la terminologia ATM, la fase di instaurazione della connessione viene chiamata *segnalazione (signalling)* e il principale protocollo di segnalazione di ATM è noto con la sigla Q.2931. Oltre a scoprire una via percorribile all'interno di una rete ATM, il protocollo Q.2931 ha anche il compito di allocare le risorse all'interno dei commutatori lungo il percorso, con lo scopo di garantire al circuito una certa qualità di servizio. Invero, la gestione della qualità di servizio (QoS) è uno dei punti di forza di ATM, su cui torneremo nel Capitolo 6, dove ne parleremo in un più ampio contesto dedicato a simili approcci di realizzazione della qualità di servizio.

Per instaurare qualsiasi connessione virtuale, è necessario inserire nel messaggio di segnalazione l'indirizzo del destinatario. Nella tecnologia ATM questo indirizzo può essere espresso in diversi formati, i più comuni dei quali sono E.164 e NSAP (network service access point): i relativi dettagli non sono di importanza vitale in questo contesto, tranne per il fatto che è opportuno notare come siano diversi dagli indirizzi MAC usati nelle reti locali tradizionali.*

Una caratteristica che rende la tecnologia ATM veramente insolita è il fatto che i pacchetti che vengono commutati in una rete ATM hanno una lunghezza fissa e che tale lunghezza è uguale a 53 byte (5 byte di intestazione seguiti da 48 byte di carico utile), una scelta piuttosto curiosa che verrà discussa con maggiore dettaglio in seguito. Per distinguere questi pacchetti di lunghezza fissa dai più comuni pacchetti di lunghezza variabile normalmente usati nelle reti di calcolatori, gli si attribuisce un nome specifico: *celle*. Si può pensare alla tecnologia ATM come all'esempio canonico di commutazione di celle.

3.3.1 Celle

Tutte le tecnologie di commutazione di pacchetto che abbiamo visto sin qui usavano pacchetti di lunghezza variabile. I pacchetti di lunghezza variabile hanno una dimensione solitamente limitata, sia superiormente sia inferiormente. Il limite inferiore è imposto dalla quantità minima di informazioni che devono essere necessariamente contenute nel pacchetto, che sono tipicamente costituite dall'intestazione senza estensioni opzionali. Il limite superiore può essere dettato da vari fattori; ad esempio, la massima dimensione dei pacchetti FDDI determina il massimo intervallo di tempo durante il quale una stazione può continuare a trasmettere senza rilasciare il testimone, per cui determina anche il tempo massimo di attesa dell'arrivo del testimone da parte di una stazione. Le celle, al contrario, sono sia di lunghezza fissa sia di piccole dimensioni: anche se questo schema di progetto sembra abbastanza semplice, ci sono in realtà parecchi fattori coinvolti, come illustrato nei paragrafi che seguono.

Dimensione delle celle

I pacchetti di dimensioni variabili hanno alcune comode proprietà. Se dovete inviare un solo byte (ad esempio, per confermare la ricezione di un pacchetto), lo inserite in un pacchetto di dimensione minima. Se dovete, invece, inviare un file di grandi dimensioni, lo suddividete in tanti pacchetti di dimensione massima quanti ne servono. Nel primo caso non avete bisogno di inviare alcun dato estraneo per raggiungere la lunghezza prefissata (*padding*), mentre nel secondo caso abbassate il rapporto tra il numero di byte dell'intestazione e il numero di byte di dati, aumentando così l'efficienza di banda. Inoltre, rende anche minimo il numero totale di pacchetti inviati, minimizzando di conseguenza l'elaborazione totale richiesta dalle operazioni necessarie per la gestione di ogni pacchetto. Quest'ultima proprietà può essere particolarmente importante per ottenere throughput elevati, perché molti dispositivi di rete sono limitati non nel numero di *bit* al secondo che possono elaborare, ma piuttosto nel numero di pacchetti al secondo.

Ciò detto, perché usare allora celle di lunghezza fissa? Una delle ragioni principali fu quella di rendere più agevole l'implementazione dei commutatori hardware. Quando venne ideata la tecnologia ATM, verso la metà degli anni Ottanta, la tecnologia Ethernet a 10 Mbps era lo stato dell'arte in termini di velocità: per aumentare la velocità, molte persone iniziarono a guardare all'hardware. Inoltre, nel mondo della telefonia, quando si pensa ai commutatori si pensa a grandi dimensioni: i commutatori telefonici servono spesso decine di migliaia di utenti. Il risultato fu che i pacchetti di dimensioni fisse erano una cosa molto utile per costruire switch veloci e molto scalabili, per le seguenti due ragioni principali:

1. È più facile costruire hardware che esegua compiti semplici, ed il compito di elaborare pacchetti è più semplice quando si conosce già la loro lunghezza.*
2. Se tutti i pacchetti hanno la stessa lunghezza, è possibile avere molti elementi di commutazione che fanno la stessa cosa in parallelo e ciascuno di essi impiegherà lo stesso tempo a svolgere il proprio compito.

Questa seconda motivazione, che consente il parallelismo, migliora grandemente la scalabilità dei progetti di commutatori. Sarebbe forse esagerato asserire che commutatori hardware paralleli e veloci si possono costruire soltanto usando celle di dimensioni fisse, ma è certamente vero che le celle agevolano il compito di progettare e costruire tale hardware e che nel momento in cui venne definito lo standard ATM vi erano molte conoscenze disponibili su come costruire commutatori hardware di celle.

Un'altra vantaggiosa proprietà delle celle riguarda il comportamento delle code, che si formano in un commutatore quando il traffico proveniente da diversi ingressi si dirige verso la medesima uscita. In generale, una volta che avete estratto un pacchetto da una coda ed avete iniziato a trasmetterlo, dovete continuare finché non è stato trasmesso interamente: interrompere la trasmissione di un pacchetto non ha alcun interesse pratico. Il tempo massimo per il quale l'uscita di una coda può essere bloccata è quindi uguale al tempo richiesto per trasmettere un pacchetto di dimensione massima. Usare celle di lunghezza fissa significa che l'uscita di una coda non è mai bloccata per un tempo superiore a quello necessario a trasmettere una cella, che è di dimensione quasi certamente inferiore a quella del pacchetto più grande possibile in una rete con pacchetti di dimensione variabile. Di conseguenza, le celle sono vantaggiose quando è importante avere un buon controllo sulla latenza sperimentata dalle celle quando attraversano una coda, anche se certamente è comunque possibile che si formino lunghe code e non esiste rimedio al fatto che alcune celle debbano aspettare il pro-

prio turno ciò che si ottiene usando le celle non sono code di lunghezza ridotta, ma la possibilità di un migliore controllo sul comportamento delle code stesse.

Un esempio aiuterà a chiarire questo fatto. Immaginate una rete con pacchetti di dimensione variabile, dove la lunghezza massima di un pacchetto è 4 KB e la velocità della linea è 100 Mbps. Il tempo richiesto per la trasmissione di un pacchetto di dimensione massima è $4096 \times 8/100 = 327.68 \mu s$. Quindi, un pacchetto ad elevata priorità che arriva quando il commutatore ha appena iniziato a trasmettere un pacchetto di 4 KB dovrà attendere in coda per 327.68 μs prima di avere accesso alla linea. Al contrario, se il commutatore stesse inoltrando celle di 53 byte, il tempo di attesa più lungo sarebbe di $53 \times 8/100 = 4.24 \mu s$. All'apparenza questo può non essere un vantaggio significativo, ma la capacità di controllare il ritardo e in special modo di controllare la sua variazione nel tempo (jitter) può essere importante per alcune applicazioni.

Le code di celle tendono anche ad essere un po' più corte delle code di pacchetti, per il seguente motivo: quando un pacchetto inizia ad entrare in una coda vuota, tipicamente il commutatore deve aspettare che il pacchetto sia arrivato interamente prima di iniziare a trasmetterlo verso una linea di uscita, per cui la linea rimane inattiva mentre il pacchetto arriva. Tuttavia, se immaginate che un lungo pacchetto venga sostituito da un "treno" di piccole celle, allora appena la prima cella del treno è entrata nella coda il commutatore la può trasmettere. Immaginate cosa accadrebbe nell'esempio precedente se due pacchetti di 4 KB arrivassero in coda pressappoco nello stesso istante: la linea rimarrebbe inattiva per 327.68 μs mentre i due pacchetti arrivano e al termine di tale intervallo di tempo avremmo 8 KB di dati in coda. Soltanto a questo punto la coda inizierebbe a svuotarsi. Se quegli stessi due pacchetti fossero stati inviati come treni di celle, allora la trasmissione delle celle potrebbe iniziare 4.24 μs dopo che il primo treno ha iniziato ad arrivare e, al termine di un intervallo di 327.68 μs , la linea sarebbe stata attiva per poco meno di 323 μs e ci sarebbero soltanto poco più di 4 KB di dati rimasti nella coda, non 8 KB come prima. Code più corte significano un ritardo minore per tutto il traffico.

Avendo deciso di usare piccoli pacchetti di dimensioni fisse, la domanda successiva è: qual è la lunghezza giusta? Se li fate troppo piccoli, allora aumenta la quantità di informazioni accessorie in rapporto alla quantità di dati che trovano posto in una cella e, conseguentemente, diminuisce la frazione di banda che viene realmente utilizzata per trasportare dati. Ancora più preoccupante è il fatto che in un dispositivo che elabori un certo numero massimo di celle in un secondo la velocità complessiva di elaborazione dei dati cala al diminuire delle dimensioni delle celle in funzione lineare della dimensione stessa. Un esempio di tale dispositivo potrebbe essere un adattatore di rete che riaggredisce le celle in unità più grandi prima di consegnarle all'host: le prestazioni di tale dispositivo dipendono direttamente dalla dimensione delle celle. D'altra parte, se fate le celle troppo grandi, subentra un problema di banda sprecata a causa delle necessità di inserire dati fittizi per raggiungere la dimensione completa della cella, ogni volta che i dati da trasmettere non la riempiono completamente. Se la dimensione del carico utile della cella è 48 byte e volete inviare 1 byte, dovete inviare 47 byte di dati fittizi (*padding*); se ciò accade spesso, la linea sarà utilizzata veramente male.

L'utilizzazione efficiente della linea non è l'unico fattore che influenza la dimensione delle celle. Ad esempio, la dimensione della cella ha effetti particolari sul traffico vocale: dato che la tecnologia ATM è stata prodotta dalla comunità telefonica, una delle preoccupazioni maggiori era quella che fosse in grado di trasportare la voce in modo efficace. La codifica digitale standard della voce si effettua a 64 Kbps (campioni di 8 bit presi a 8 KHz). Per maximizzare l'efficienza, si vogliono raccogliere campioni in misura sufficiente a

riempire un'intera cella prima di trasmettere la cella stessa. Ad una velocità di campionamento di 8 KHz viene campionato 1 byte ogni 125 μs , per cui il tempo necessario per riempire di campioni una cella di n byte è uguale a $n \times 125 \mu s$. Se le celle fossero lunghe, ad esempio, 1000 byte, sarebbero necessari 125 ms soltanto per raccogliere un'intera cella di campioni, ancor prima di iniziare a trasmetterli al ricevitore. Una latenza di questo ordine di grandezza inizia ad essere percepibile per un ascoltatore umano e anche latenze considerevolmente inferiori creano problemi nel caso della voce, in particolar modo per via degli echi. Gli echi si possono eliminare con uno strumento tecnologico chiamato soppressore di echi, ma ciò aggiunge ad una rete telefonica costi che molti operatori di rete vorrebbero invece evitare.

Tutti i fattori qui elencati furono oggetto di acesi dibattiti negli enti di standardizzazione internazionali dove si stava delineando lo standard ATM ed il fatto che nessuna lunghezza fosse perfetta in tutti i casi venne usato dagli oppositori di ATM per argomentare che le celle di lunghezza fissa erano, prima di tutto, una cattiva idea. Come sovente accade con gli standard, il risultato finale fu un compromesso che non piaceva quasi a nessuno: per il carico utile di una cella ATM fu scelta la lunghezza di 48 byte. Forse il peggior difetto di questa scelta è che non si tratta di una potenza di due e che quindi mal si adatta alla maggior parte delle cose che vengono gestite da calcolatori, come le pagine e le righe delle memorie cache. Con qualche controversia in meno, la lunghezza dell'intestazione venne fissata a 5 byte. Il formato di una cella ATM è mostrato in Figura 3.16, dove la lunghezza dei campi è espressa in bit.

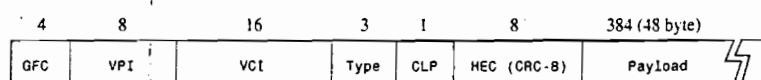


Figura 3.16 Formato UNI delle celle ATM.

Un compromesso di 48 byte

La motivazione per cui il carico utile di una cella ATM è lungo 48 byte è interessante ed è un esempio molto significativo dei processi di standardizzazione. Durante l'evoluzione dello standard ATM, le compagnie statunitensi premevano per una cella di 64 byte, mentre le compagnie europee proponevano per celle a 32 byte. Il motivo per cui le compagnie europee chiedevano una dimensione minore era che, dal momento che i Paesi da esse serviti avevano una dimensione abbastanza piccola, se fossero state in grado di contenere il valore della latenza usando celle sufficientemente piccole, non avrebbero dovuto installare i soppressori di eco. Celle a 32 byte erano adatte a questo scopo. Al contrario, gli Stati Uniti sono un Paese così esteso che le compagnie telefoniche avrebbero comunque dovuto installare i soppressori, per cui volevano celle più grandi per migliorare il rapporto fra intestazione e carico utile.

Fare la media è un metodo classico per raggiungere un compromesso: 48 byte è semplicemente il valore medio fra 64 e 32 byte. Per non dare la falsa impressione che questo uso della media come compromesso sia un incidente isolato, facciamo notare che il modello OSI a sette strati fu, in realtà, un compromesso tra un modello a sei strati ed uno a otto strati.

Formato delle celle

In realtà esistono due formati per le celle ATM, in relazione alla posizione in cui si trovano all'interno della rete. Quella mostrata in Figura 3.16 è in formato UNI (user-network interface) e viene usata, ovviamente, all'interfaccia tra l'utente e la rete, che molto probabilmente sarà anche l'interfaccia tra una compagnia telefonica e uno dei suoi clienti. L'alternativa è il formato NNI (network-network interface), usata con buona probabilità tra due compagnie telefoniche. L'unica differenza significativa tra i formati delle celle è che nel formato NNI il campo GFC viene sostituito con 4 ulteriori bit dedicati al campo VPI. Chiaramente, comprendere tutti gli acronimi di tre lettere (TLA, three-letter acronym) è fondamentale per capire la tecnologia ATM.

Iniziando dal byte più a sinistra della cella (che è il primo ad essere trasmesso), il formato UNI prevede 4 bit per un generico controllo di flusso (GFC, generic flow control): questi bit non sono stati molto utilizzati, ma il loro scopo doveva essere quello di avere un significato locale e di poter essere modificati all'interno della rete. L'idea basilare dei bit GFC era quella di fornire uno strumento di arbitraggio per l'accesso alla linea di connessione, nel caso in cui localmente venisse usato un mezzo fisico condiviso per la connessione alla rete ATM.

I 24 bit successivi contengono un identificatore di percorso virtuale (VPI, virtual path identifier) a 8 bit ed un identificatore di circuito virtuale (VCI, virtual circuit identifier) a 16 bit. La differenza tra i due è spiegata in seguito, ma per ora è sufficiente immaginarli come un singolo identificatore a 24 bit che viene usato per identificare una connessione virtuale, proprio come nella Sezione 3.1.2. Dopo i campi VPI/VCI c'è un campo di 3 bit, Type, che può assumere otto diversi valori. Quattro di questi valori, quelli con il primo bit che vale 1, sono relativi a funzioni di gestione; quando, invece, il primo bit vale 0 significa che la cella contiene dati dell'utente. In questo caso il secondo bit serve per la "indicazione esplicita di congestione nell'inoltro" (EFCL, explicit forward congestion indication), mentre il terzo serve per la "segnalazione da parte dell'utente" (user signalling). Il primo può essere utilizzato da un commutatore congestionato per segnalare tale situazione ad un nodo terminale ed ha avuto origine dal DECbit descritto nella Sezione 6.4.1; nella tecnologia ATM viene usato per il controllo di congestione, insieme alla categoria di servizio che descrive la velocità di bit disponibile (ABR, available bit rate), come discusso nella Sezione 6.5.4. Il terzo bit viene usato nello strato 5 di adattamento ATM (ATM Adaptation Layer 5) per indicare la separazione tra i frame, come vedremo nel seguito.

Il bit successivo serve per indicare la priorità di eliminazione della cella (CLP, cell loss priority): un utente o un elemento della rete possono usare questo bit per segnalare quali celle si dovrebbero preferibilmente eliminare in caso di sovraccarico. Ad esempio, un'applicazione di codifica video potrebbe impostare questo bit nelle celle che, se eliminate, non degraderebbero in modo sostanziale la qualità delle immagini, mentre un elemento di rete potrebbe impostare questo bit nelle celle che sono state trasmesse da un utente al di là della quantità di traffico che era stata negoziata.

L'ultimo byte dell'intestazione è un CRC a 8 bit, denominato "verifica d'errore nell'intestazione" (HEC, header error check). Usa il polinomio CRC-8 visto nella Sezione 2.4.3 e consente, per la sola intestazione della cella, la rilevazione d'errore, nonché la correzione di errori su un singolo bit. Proteggere l'intestazione della cella è particolarmente importante, perché un errore nel campo VCI provocherebbe un'errata consegna della cella.

3.3.2 Segmentazione e ricostruzione

Fino ad ora abbiamo ipotizzato che un protocollo di livello inferiore sia in grado di accettare il pacchetto che gli viene trasmesso da un protocollo di livello superiore, anteponendogli la propria intestazione, per poi trasferire il pacchetto verso il basso nel grafo di protocolli. Tuttavia, ciò non è consentito dalla tecnologia ATM, perché i pacchetti ricevuti dagli strati superiori sono spesso più grandi di 48 byte e, quindi, non trovano posto nel carico utile di una cella ATM. La soluzione a questo problema consiste nel frammentare nella sorgente il messaggio di alto livello in più pacchetti di basso livello, trasmettere lungo la rete i singoli pacchetti di basso livello e assemblare i frammenti una volta giunti a destinazione. Nel caso della tecnologia ATM, però, questo procedimento viene spesso chiamato *segmentazione e ricostruzione* (SAR, segmentation and reassembly).

La segmentazione non è una peculiarità di ATM, ma è un problema più frequente di quanto non lo sia in una rete che abbia, ad esempio, una dimensione massima del pacchetto di 1500 byte. Per risolvere questo problema, è stato aggiunto uno strato di protocolli situato fra ATM ed i protocolli con pacchetti di dimensione variabile che possono usare ATM, come IP. Questo strato è chiamato strato di adattamento ATM (AAL, ATM Adaptation Layer) ed in prima approssimazione l'intestazione AAL contiene le informazioni necessarie al destinatario per ricostruire le singole celle, ricomponendo il messaggio originario. La Figura 3.17 mostra la relazione esistente tra AAL e ATM.

Dato che ATM è stata progettata per fornire supporto a qualsiasi tipo di servizio, tra cui* voce, video e dati, si pensò che servizi diversi avrebbero avuto bisogno di AAL diversi, per cui furono originariamente definiti quattro strati di adattamento: gli strati 1 e 2 furono progettati per fornire supporto a quelle applicazioni, come le applicazioni vocali, che richiedono velocità di bit garantite, mentre gli strati 3 e 4 dovevano fornire il supporto per pacchetti di dati trasferiti mediante ATM. Si pensava di usare AAL3 per servizi a pacchetto orientati alla connessione (come X.25) e AAL4 per servizi privi di connessione (come IP), ma le ragioni per avere due diversi AAL per questi due tipi di servizi furono poi ritenute insufficienti e gli strati corrispondenti si fusero in un unico strato, scomodamente chiamato AAL3/4. Nel frattempo, alcuni piccoli problemi di AAL3/4 diedero luogo alla proposta di un quinto AAL, chiamato AAL5, per cui ora vi sono quattro AAL: 1, 2, 3/4 e 5. I due AAL utili alle comunicazioni tra calcolatori vengono descritti nel seguito.

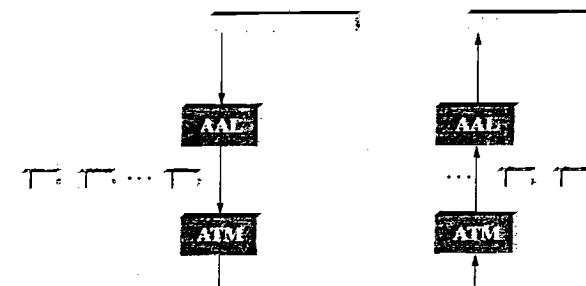


Figura 3.17 Segmentazione e ricostruzione in ATM.

Strato di adattamento ATM 3/4 (AAL3/4)

La funzione principale di AAL3/4 è quella di fornire informazioni sufficienti a consentire che pacchetti di dimensioni variabili vengano trasportati lungo reti ATM come una serie di celle di dimensione fissa: detto in altre parole, questo AAL fornisce il supporto per la procedura di segmentazione e ricostruzione. Dato che stiamo operando all'interno di un nuovo strato nella gerarchia di rete, le convenzioni richiedono che venga introdotto un nuovo nome per i pacchetti, che in questo caso verranno chiamati *unità di dati di protocollo* (PDU, protocol data unit). Il compito di segmentazione e ricostruzione viene gestito con due diversi formati di pacchetto, il primo dei quali è chiamato *unità di dati di protocollo del substrato di convergenza* (CS-PDU, convergence sublayer protocol data unit), come mostrato in Figura 3.18. Il CS-PDU definisce un modo per incapsulare PDU di lunghezza variabile prima della segmentazione in celle: il PDU giunto dall'alto nello strato AAL viene incapsulato con l'aggiunta di un'intestazione (*header*) e di una parte terminale (*trailer*), per poi segmentare in celle ATM il CS-PDU che ne risulta.

Il formato CS-PDU inizia con un campo di 8 bit detto *indicatore della parte comune* (CPI, common part indicator), che indica quale versione del formato di CS-PDU viene usata: attualmente è definito solamente il valore 0. I successivi 8 bit contengono il marcitore iniziale (Btag, beginning tag), che deve corrispondere al marcitore finale (Etag, end tag) di ciascun PDU: questo schema fornisce protezione nei confronti della situazione in cui la perdita dell'ultima cella di un PDU e della prima cella di un altro PDU provochi la fusione incontrollata in un unico PDU, trasmesso poi verso l'alto al successivo protocollo presente nella pila. Il campo per la dimensione di allocazione del buffer (BASize, buffer allocation size) non è necessariamente uguale alla lunghezza del PDU (che compare, invece, nel trailer), ma è un suggerimento fornito alla procedura di ricostruzione che indica quanto spazio assegnare nel buffer per la ricostruzione stessa. Il motivo per cui non viene indicata qui la reale lunghezza è che l'host sorgente potrebbe non conoscere la lunghezza del CS-PDU nel momento in cui ne viene trasmessa l'intestazione. Infatti, prima di aggiungere il trailer al CS-PDU, la dimensione dei dati dell'utente viene aumentata fino a raggiungere un valore inferiore di uno rispetto ad un multiplo di 4 byte, aggiungendo fino a 3 byte di dati fittizi (*padding*). Questi dati fittizi, seguiti dal byte di valore 0, garantiscono che il trailer sia allineato ad un multiplo di 32 bit, rendendo l'elaborazione più efficiente. Il trailer del CS-PDU contiene il campo Etag e la reale lunghezza del PDU (Len).

Oltre all'intestazione e al trailer del CS-PDU, AAL3/4 specifica un'intestazione e un trailer per ogni cella, come raffigurato in Figura 3.19, per cui il CS-PDU viene in realtà segmentato in pezzi di 44 byte; un'intestazione e un trailer AAL3/4 vanno poi ad incaps-

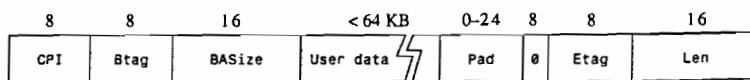


Figura 3.18 Formato del pacchetto dello strato di adattamento ATM 3/4.

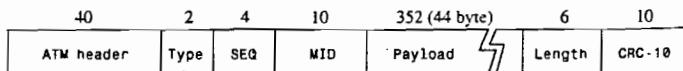


Figura 3.19 Formato della cella in AAL3/4.

3.3 Comutazione di celle (ATM)

sulare ciascuno spezzone, creando celle di 48 byte che vengono poi trasportate come carico utile in una cella ATM.

I primi due bit dell'intestazione AAL3/4 contengono il campo Type, che segnala se si tratta della prima cella di un CS-PDU, oppure dell'ultima, oppure di una cella intermedia, oppure, infine, di un PDU che trova posto in un'unica cella (nel qual caso si tratta sia della prima che dell'ultima). I nomi ufficiali per queste quattro condizioni sono indicati in Tabella 3.4, insieme ai rispettivi bit di codifica.

Successivamente troviamo un numero di sequenza a 4 bit (SEQ), usato solamente per individuare la perdita di celle o il loro errato ordinamento, in modo che si possa interrompere la ricostruzione; è ovvio che un numero di sequenza così piccolo non può rilevare perdite di celle se il numero di celle perse è abbastanza grande. Questo campo è seguito da un identificatore per il multiplexing (MID, multiplexing identifier), usato per effettuare il multiplexing di più PDU lungo un'unica connessione. Il campo Length a 6 bit indica il numero di byte del PDU che sono contenuti nella cella e deve essere uguale a 44 per le celle di tipo BOM e COM. Infine, viene usato un CRC a 10 bit per rilevare errori in qualsiasi punto dei 48 byte del carico utile della cella.

La Figura 3.20 mostra l'intero processo di incapsulamento e segmentazione in AAL3/4. In alto, i dati vengono incapsulati con l'intestazione e il trailer del CS-PDU, che viene poi segmentato in carichi utili di 44 byte, a loro volta incapsulati in celle ATM con l'aggiunta dell'intestazione e del trailer di AAL3/4, nonché dei 5 byte dell'intestazione ATM. Notate che, qualora la dimensione del CS-PDU non sia esattamente multipla di 44 byte, l'ultima cella è riempita soltanto parzialmente.

Tabella 3.4 Campo Type di AAL3/4.

Valore	Nome	Significato
10	BOM	Inizio del messaggio
00	COM	Proseuzione del messaggio
01	EOM	Fine del messaggio
11	SSM	Messaggio in un solo segmento

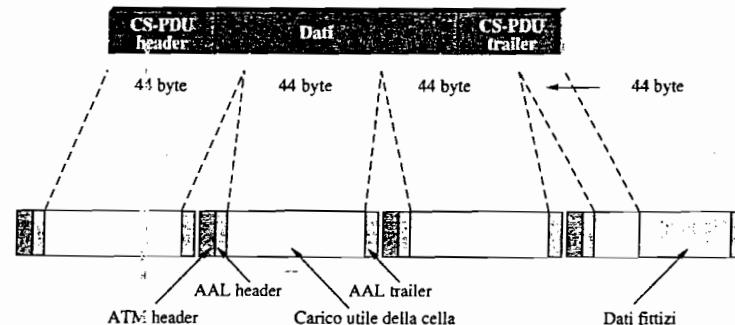


Figura 3.20 Incapsulamento e segmentazione in AAL3/4.

Una cosa da notare a proposito di AAL3/4 è che, rispetto a quanto abbiamo descritto in precedenza, incrementa notevolmente la quantità fissa di informazioni che viene aggiunta a ciascuna cella (*overhead*). Con un rapporto di 44 byte di dati contro 9 byte di intestazione, la miglior utilizzazione possibile della banda sarebbe 83%, ma notate che l'efficienza può essere considerevolmente inferiore, come mostrato in Figura 3.20, a causa dell'incapsulamento CS-PDU e del riempimento parziale dell'ultima cella.

Strato di adattamento ATM-5 (AAL5)

Una cosa che forse avrete notato nella discussione relativa a AAL3/4 è il fatto che sembra utilizzare parecchi campi e, quindi, parecchi dati aggiuntivi per portare a termine la funzione, concettualmente semplice, di frammentazione e ricostruzione. Questa osservazione fu fatta, in realtà, da parecchie persone fin dai primordi di ATM e nacquero molte proposte alternative per uno strato di adattamento (AAL) che fornisse il supporto alle comunicazioni fra calcolatori in una rete ATM. Vi fu un movimento di pensiero, ufficiosamente noto con il nome di "Back the Bit", che sosteneva che usando soltanto un bit nell'intestazione ATM (invece di usare un'intestazione AAL) per indicare la fine di un frame, la segmentazione e la ricostruzione conseguente sarebbero state possibili senza usare nessuno dei 48 byte del carico utile di ATM per informazioni relative al processo di segmentazione/ricostruzione; da ciò derivò la definizione del bit di segnalazione da parte dell'utente (*user signalling*) descritto precedentemente e la standardizzazione di AAL5.

AAL5 rimpiazza il campo *Type* a 2 bit di AAL3/4 con un bit di informazione di framing nell'intestazione della cella ATM. Impostando al valore 1 tale bit è possibile identificare l'ultima cella di un PDU; la cella successiva è, per ipotesi, la prima cella del PDU successivo e le celle seguenti sono celle COM finché non viene ricevuta un'altra cella avente il bit di *user signalling* al valore 1. Tutte le caratteristiche di AAL3/4 che forniscono protezione contro la perdita, la corruzione o l'errato ordinamento delle celle, compresa la perdita di una cella EOM, sono fornite dal formato del pacchetto CS-PDU di AAL5, delineato in Figura 3.21.

Il CS-PDU di AAL5 è composto, semplicemente, dalla parte di dati (cioè dal PDU ricevuto dal protocollo di livello superiore) e da un trailer di 8 byte. Per essere certi che il trailer si trovi sempre alla fine di una cella ATM, possono essere necessari fino a 47 byte di dati fittizi fra i dati veri e il trailer stesso; la necessità di avere il trailer alla fine di una cella è dovuta al fatto che, altrimenti, l'entità incaricata della ricostruzione del CS-PDU non avrebbe modo di trovare il trailer. I primi 2 byte del trailer sono attualmente riservati per usi futuri e devono avere il valore 0. Il campo di lunghezza (*Len*) rappresenta il numero di byte che compongono il PDU, senza contare il trailer né gli eventuali dati fittizi che precedono il trailer. Infine, c'è un CRC a 32 bit.

La Figura 3.21 mostra il processo di incapsulamento e ricostruzione per AAL5. Esattamente come in AAL3/4, i dati dell'utente vengono incapsulati per formare un CS-PDU (anche se in questo caso si usa soltanto un trailer). Il PDU risultante viene poi suddiviso in spezzoni di 48 byte, direttamente inseriti come carico utile di celle ATM senza ulteriori incapsulamenti.

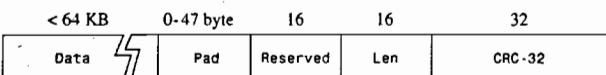


Figura 3.21 Formato del pacchetto di AAL5.

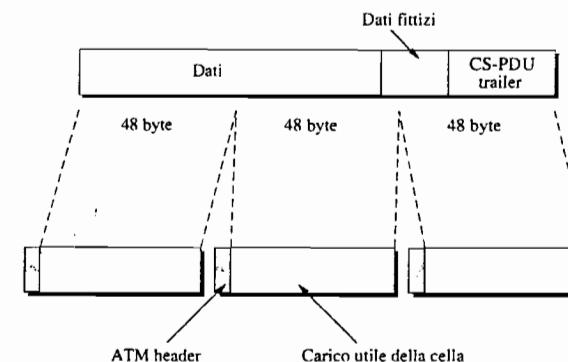


Figura 3.22 Incapsulamento e segmentazione in AAL5.

In modo alquanto sorprendente, AAL5 fornisce praticamente la stessa funzionalità di AAL3/4 senza usare 4 byte aggiuntivi per ogni cella. Ad esempio, il CRC-32 rileva celle perdute o fuori ordine, nonché errori di bit nei dati: addirittura, si può vedere che utilizzando una somma di controllo sull'intero PDU, piuttosto che sulle singole celle come in AAL3/4, si ottiene una protezione migliore. Ad esempio, si ha una protezione contro la perdita di 16 celle consecutive, un evento che non verrebbe segnalato dalla verifica del numero di sequenza di AAL3/4. Ancora, un CRC a 32 bit protegge contro errori a burst di più lunga durata rispetto a quanto può fare un CRC a 10 bit.

La principale caratteristica assente in AAL5 è la capacità di fornire un livello di multiplexing addizionale su un unico circuito virtuale usando il campo *MID*, anche se non è chiaro se questa mancanza sia significativa. Usando AAL5 è comunque possibile effettuare il multiplexing su un singolo circuito virtuale di traffico proveniente da più applicazioni e da più protocolli di livello superiore trasportando insieme ai dati una chiave di demux del tipo descritto nella Sezione 1.3.1, soltanto che è necessario eseguire il multiplexing pacchetto per pacchetto anziché cella per cella.

Effettuare il multiplexing su un singolo circuito virtuale di traffico proveniente da molte diverse applicazioni ha aspetti positivi e negativi. Ad esempio, se pagate per ogni circuito virtuale che instaurate attraverso la rete, allora il multiplexing può essere un vantaggio, ma c'è lo svantaggio che tutte le applicazioni devono funzionare con la qualità di servizio (cioè garanzia di ritardo e di ampiezza di banda) che è stata scelta per quell'unica connessione, con la conseguenza che alcune applicazioni potrebbero non ricevere un servizio adeguato.

In generale, AAL5 è stato adottato con entusiasmo dalla comunità delle comunicazioni tra calcolatori (per lo meno da quella parte della comunità che ha adottato ATM). Ad esempio, AAL5 è lo strato di adattamento per ATM scelto da IETF per la trasmissione di datagrammi IP in una rete ATM. Il suo utilizzo più efficiente della banda e lo schema progettuale semplice sono le caratteristiche che lo rendono preferibile a AAL3/4.

3.3.3 Percorsi virtuali

Come accennato in precedenza, ATM usa un identificatore a 24 bit per i circuiti virtuali, che funzionano quasi esattamente come quelli descritti nella Sezione 3.1.2. La cosa strana è che l'identificatore a 24 bit è suddiviso in due parti: un identificatore di percorso virtuale (VPI, virtual path identifier) a 8 bit e un identificatore di circuito virtuale (VCI, virtual circuit identifier) a 16 bit. In questo modo, in sostanza, si crea una gerarchia di connessioni virtuali a due livelli. Per capire come funziona tale gerarchia, considerate l'esempio seguente (trascuriamo il fatto che può esistere un'interfaccia rete-rete con VPI di dimensioni diverse, ipotizziamo semplicemente che vengano usati dappertutto VPI a 8 bit).

Supponete che un'azienda abbia due sedi connesse ad una rete ATM pubblica e che in ciascuna sede l'azienda abbia una rete di commutatori ATM. Potremmo immaginare di stabilire un percorso virtuale tra le due sedi usando soltanto il campo VPI, per cui i commutatori nella rete pubblica userebbero soltanto il valore di VPI come campo in base al quale prendere le proprie decisioni di inoltro: dal punto di vista degli switch, questa è una rete a circuito virtuale con identificatori di circuito a 8 bit. Il campo VCI a 16 bit non è di alcun interesse per questi switch pubblici, che non usano tale campo per la commutazione, né lo modificano. All'interno delle sedi dell'azienda, però, la commutazione usa l'intero spazio di indirizzamento a 24 bit. Tutto il traffico che deve fluire fra le due sedi viene instradato verso uno switch che abbia una connessione con la rete pubblica ed i suoi primi 8 bit (il campo VPI) vengono modificati nel valore corretto, che consente ai dati di giungere all'altra sede. Questa idea è illustrata nella Figura 3.23. Notate che il percorso virtuale funziona come una grossa condutture che contiene un gruppo di circuiti virtuali, aventi tutti lo stesso valore negli 8 bit del proprio byte più significativo.

Il vantaggio di questo approccio è evidente: sebbene vi possano essere migliaia o milioni di connessioni virtuali attraverso la rete pubblica, i commutatori della rete pubblica si comportano come se vi fosse un'unica connessione. Ciò significa che è necessario memorizzare negli switch una quantità di informazioni correlate allo stato delle connessioni molto minore, evitando il bisogno di tabelle enormi e costose per le informazioni relative a ciascun valore di VCI.

3.3.4 Strati fisici per ATM

Anche se l'approccio a strati del progetto di protocolli potrebbe portarvi a pensare di non dovervi preoccupare del tipo di connessione punto-punto su cui viene eseguita la tecnologia ATM, non è proprio così. Da un punto di vista semplice e pratico, quando comprate un adattatore ATM per una workstation o uno switch ATM, è previsto che utilizzi un mezzo

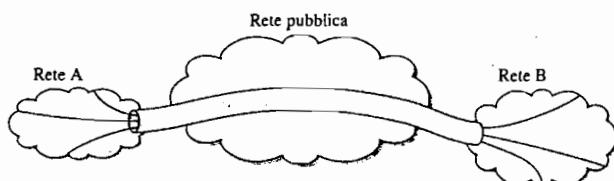


Figura 3.23 Esempio di percorso virtuale.

3.3 Commutazione di celle (ATM)

fisico ben preciso per inviare le celle ATM, così come accade, ovviamente, per altri protocolli di rete, come 802.5 e Ethernet. Come tali protocolli, anche ATM può operare con parecchi mezzi fisici diversi e diversi protocolli dello strato fisico.

Fin dai primi passi del processo di standardizzazione di ATM, si pensò che ATM sarebbe stata eseguita su uno strato fisico SONET (Sezione 2.3.3), al punto che alcuni confondono ATM e SONET, perché tali tecnologie sono strettamente correlate. Anche se è vero che sono state definite modalità standard per trasportare celle ATM all'interno di frame SONET e che oggi potete acquistare prodotti per ATM su SONET (ATM-over-SONET), le due tecnologie sono assolutamente separabili. Ad esempio, è possibile noleggiare una linea SONET da una compagnia telefonica ed inviare qualsiasi cosa attraverso essa, compresi pacchetti di lunghezza variabile. Ancora, potete inviare celle ATM attraverso molti strati di livello fisico diversi da SONET ed esistono standard definiti (o in fase di definizione) per questi incapsulamenti. Uno dei primi strati fisici per ATM degno di nota fu TAXI, lo strato fisico usato in FDDI (Sezione 2.7) e sono in corso di definizione anche strati fisici wireless per ATM.

Quando si inviano celle ATM attraverso un mezzo fisico, il problema principale consiste nell'identificare i confini delle celle ATM: si tratta, precisamente, del problema del framing descritto nel Capitolo 2. Con SONET esistono due semplici modi per trovare i confini. Uno dei byte di overhead del frame SONET può essere utilizzato come puntatore all'interno del carico utile SONET per trovare l'inizio di una cella ATM: avendo trovato l'inizio di una cella, è noto che la cella successiva inizia dopo 53 byte nel carico utile SONET, e così via. In teoria, questo puntatore deve essere letto una volta soltanto, ma in pratica ha senso leggerlo ogni volta che transita l'overhead SONET in modo da poter rilevare errori o, eventualmente, sincronizzarsi nuovamente, se necessario.

L'altro modo per identificare i confini di celle ATM sfrutta il fatto che ciascuna cella ha un CRC nel proprio quinto byte: quindi, se eseguite un calcolo di CRC sugli ultimi 5 byte ricevuti ed ottenete la risposta corretta, allora è molto probabile che abbiate appena letto un'intestazione ATM. Se ciò accade parecchie volte di seguito ad intervalli di 53 byte, potete essere ben certi di aver trovato il confine tra le celle.

3.3.5 ATM in una rete locale

Come abbiamo già detto, la tecnologia ATM è stata sviluppata dalla comunità telefonica, che l'aveva concepita come uno strumento per costruire grandi reti pubbliche in grado di veicolare traffico vocale, video e dati. Tuttavia, è stata in seguito adottata da alcuni settori industriali nel campo dei calcolatori e delle comunicazioni di dati come tecnologia da utilizzare nelle reti locali, in sostituzione di Ethernet e di 802.5. La sua popolarità in questo settore, in un certo momento storico, può essere attribuita principalmente a due fattori:

- ATM è una tecnologia commutata, mentre Ethernet e 802.5 furono inizialmente progettate come tecnologie a mezzo fisico condiviso.
- ATM fu progettata per operare su linee con velocità di 155 Mbps e oltre, in confronto ai 10 Mbps di Ethernet e ai 4 o 16 Mbps delle reti di tipo token ring.

Quando i commutatori ATM divennero disponibili, questi erano vantaggi significativi rispetto alle soluzioni esistenti. In particolare, le reti commutate hanno un grande vantaggio sulle reti a mezzo condiviso, dal punto di vista delle prestazioni: una singola rete a mezzo condiviso ha un'ampiezza di banda totale fissa, che deve essere suddivisa tra tutti gli host, mentre in

una rete commutata ciascun host ha la propria linea dedicata che lo collega allo switch. Di conseguenza, le prestazioni di reti commutate scalano meglio di quelle di reti a mezzo condiviso.

Tuttavia, dovrebbe essere evidente che le differenze tra reti a mezzo condiviso e reti commutate non sono così nette. Un bridge che connette insieme un certo numero di reti a mezzo condiviso è anche un commutatore ed è possibile (e abbastanza frequente) connettere un solo host a ciascun segmento, garantendogli un accesso dedicato alla relativa ampiezza di banda. Nello stesso momento in cui i commutatori ATM apparivano sul mercato, diventavano disponibili commutatori Ethernet ad elevate prestazioni: si tratta di dispositivi con un grande numero di porte e un throughput totale elevato. Fu definito lo standard Ethernet a 100 Mbps, per cui la velocità di una linea Ethernet ottenibile con cavi in rame iniziò ad avvicinarsi a quella di ATM.

Tutto ciò non fu sufficiente per eliminare ATM dal settore delle reti locali. Uno dei vantaggi rimanenti di ATM rispetto a Ethernet è la mancanza di limiti sulla distanza per linee di collegamento ATM. Inoltre divennero presto disponibili linee di collegamento ATM a velocità ancora più elevata (ad esempio, 622 Mbps), cosa che rese la tecnologia ATM assai popolare per le "dorsali" (backbone) ad alte prestazioni di grandi reti locali. Una configurazione comune era quella che prevedeva di connettere i singoli host a commutatori Ethernet che a loro volta venivano interconnessi tramite commutatori ATM, come illustrato in Figura 3.24. I server ad alte prestazioni potevano anche essere connessi direttamente allo switch ATM, come l'host H7 in questo esempio.

Più di recente, la tecnologia che ha probabilmente preso il sopravvento su ATM per le dorsali delle reti locali e per le connessioni dei server è Gigabit Ethernet. Le linee di collegamento Gigabit Ethernet usano la stessa tecnica di framing delle reti Ethernet a velocità inferiore, ma sono solitamente linee punto-punto in fibra ottica e possono operare su distanze relativamente lunghe (fino a parecchi chilometri). Lo stesso approccio di base sta ora per essere scalato per fornire linee a 10 Gbps.

Un problema importante quando si usa ATM in una rete locale è il fatto che non sembra una rete locale "tradizionale". Dato che molte reti locali (tra cui Ethernet e token ring) sono reti a mezzo fisico condiviso (cioè tutti i nodi della rete sono connessi alla stessa linea), è facile implementare sia il broadcast (invio a tutti) sia il multicast (invio ad un gruppo). Di

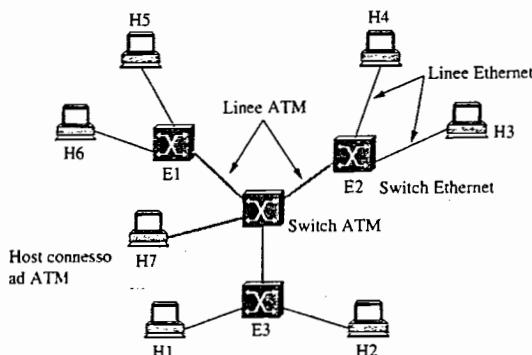


Figura 3.24 ATM usata come dorsale in una rete locale.

3.3 Comutazione di celle (ATM)

conseguenza, molti dei protocolli da cui le persone all'interno di una LAN dipendono (come, ad esempio, il protocollo ARP, Address Resolution Protocol, descritto nella Sezione 4.1.5), sono a loro volta dipendenti dalla capacità della LAN di consentire trasmissioni broadcast e multicast. A causa della propria natura commutata e orientata alla connessione, però, la tecnologia ATM si comporta in modo piuttosto diverso da una LAN a mezzo fisico condiviso. Ad esempio, come potete fare una trasmissione broadcast verso tutti i nodi di una rete locale ATM se non conoscete i loro indirizzi e non instaurate un circuito virtuale verso ciascuno di essi?

Esistono due soluzioni possibili a questo problema e sono state esplorate entrambe. La prima prevede di riiprogettare i protocolli che fanno sulle reti locali ipotesi che non sono affatto vere nel caso ATM, per cui, ad esempio, esiste un nuovo protocollo, chiamato ATMARP, che, diversamente dal tradizionale ARP, non dipende dalla trasmissione broadcast; ne parleremo nella Sezione 4.1.5. L'alternativa è quella di fare in modo che ATM si comporti in modo più simile ad una LAN a mezzo fisico condiviso, nel senso di fornire supporto alle trasmissioni multicast e broadcast, senza perdere i vantaggi in termini di prestazioni tipici di una rete commutata. Questo approccio è stato indicato dall'ATM Forum come "emulazione di LAN" o LANE (LAN emulation), anche se più precisamente si tratta di "emulazione di mezzo condiviso", con l'obiettivo di aggiungere funzionalità ad una LAN ATM in modo qualunque cosa sia in grado di operare su una LAN a mezzo fisico condiviso possa operare anche in una LAN. Nonostante oggi LANE possa essere considerata soltanto una curiosità storica, fornisce un esempio interessante di come possa funzionare la stratificazione in una rete: facendo in modo che lo "strato ATM" assomigli maggiormente ad una Ethernet, i protocolli di livello superiore che funzionano bene con Ethernet continuano a funzionare senza alcuna modifica.

Un aspetto dell'emulazione della LAN che può creare confusione è la varietà di diversi indirizzi e identificatori che vengono utilizzati. Tutti i dispositivi ATM devono avere un indirizzo ATM, che viene usato durante la fase di segnalazione per instaurare un circuito virtuale. Come notato in precedenza, questi indirizzi sono diversi dagli indirizzi MAC aderenti allo standard IEEE 802 usati da Ethernet, token ring, e così via. Se si vuole emulare il comportamento di questi tipi di reti locali, ciascun dispositivo deve avere anche un indirizzo MAC standard (cioè a 48 bit e globalmente univoco). Infine, ricordate che un identificatore di circuito virtuale è molto diverso da un indirizzo: è l'abbreviazione utilizzata per trasportare celle lungo una connessione già instaurata, ma prima occorre instaurare una connessione, e per farlo c'è bisogno di un indirizzo ATM.

L'emulazione di LAN in realtà non modifica la funzionalità dei commutatori ATM, ma aggiunge funzionalità alla rete mediante l'aggiunta di un certo numero di servizi, mentre per quanto riguarda i dispositivi che si connettono ad una rete ATM (host, bridge, router) si parla di client della LANE (LEC, LAN emulation client). Le interazioni tra i LEC e i vari server producono un comportamento della rete che, dal punto di vista di un protocollo di livello superiore, è indistinguibile da quello di una rete Ethernet o di una rete di tipo token ring. La Figura 3.25 mostra gli strati di protocolli nel caso in cui una coppia di host comunichi attraverso una rete ATM che sta emulando una LAN. Con il termine "interfaccia di tipo Ethernet" intendiamo che i servizi offerti agli strati superiori sono quelli di una rete Ethernet: si possono consegnare frame a qualsiasi indirizzo MAC appartenente alla LAN, si possono inviare frame di tipo broadcast a tutti gli host della LAN, e così via.

I server necessari per costruire una LAN emulata sono:

- il server di configurazione dell'emulazione di LAN (LECS, LAN emulation configuration server)

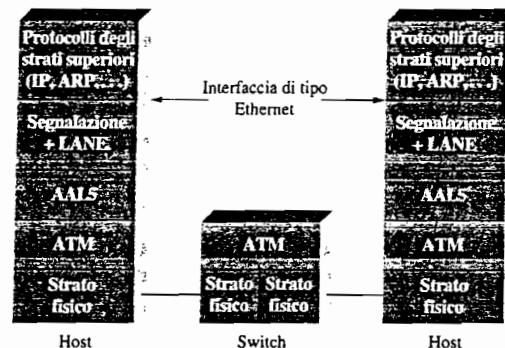


Figura 3.25 Strati di protocolli in un'emulazione di LAN.

- il server di emulazione della LAN (LES, LAN emulation server)
- il server broadcast e per indirizzi sconosciuti (BUS, broadcast and unknown server)

Questi server possono trovarsi fisicamente all'interno di uno o più dispositivi, probabilmente in uno degli host o in altri dispositivi connessi alla rete ATM. Il LECS e il LES svolgono principalmente funzioni di configurazione, mentre il BUS ha un ruolo centrale nel rendere il trasferimento di dati in una rete ATM simile a quello caratteristico di una LAN a mezzo fisico condiviso.

Il LECS consente ad un client (ad esempio, un host) dell'emulazione di LAN appena connesso o riaccesso di ottenere alcune informazioni essenziali. Prima di tutto, il client deve trovare il LECS, cosa che può fare usando un circuito virtuale ben noto e predefinito, che è sempre instaurato; in alternativa, il client deve conoscere a priori l'indirizzo ATM del LECS, in modo da instaurare un circuito virtuale verso di esso. Una volta connesso al LECS, il client fornisce al LECS il proprio indirizzo ATM ed il LECS risponde indicando al client quale tipo di LAN è in corso di emulazione (Ethernet o token ring), qual è la dimensione massima dei pacchetti e qual è l'indirizzo ATM del LES. Un LECS può fornire supporto a molte emulazioni di LAN distinte.

Il client ora agisce mediante la procedura di segnalazione per instaurare una connessione con il LES, di cui ha appreso l'indirizzo ATM. Una volta stabilita la connessione con il LES, il client registra nel LES i propri indirizzi MAC e ATM e, fra le altre cose, il LES fornisce al client l'indirizzo ATM del BUS.

Il BUS gestisce un unico circuito virtuale punto-multipunto che lo connette a tutti i client registrati. Dovrebbe essere evidente come il BUS e questo circuito virtuale multipunto siano cruciali per l'emulazione di LAN, in quanto consentono di emulare in un ambiente a circuito virtuale la tradizionale possibilità di trasmissioni broadcast delle reti locali. Quando un LEC ha l'indirizzo ATM del BUS, agisce mediante la procedura di segnalazione per instaurare una connessione con il BUS, il quale a sua volta aggiunge il LEC al circuito virtuale punto-multipunto. A questo punto, è tutto pronto perché il LEC possa partecipare ai trasferimenti di dati. La Figura 3.26 illustra una situazione in cui due host si sono connessi al LES e al BUS ed il BUS ha instaurato il circuito virtuale punto-multipunto verso entrambi, mentre il LECS non viene mostrato.

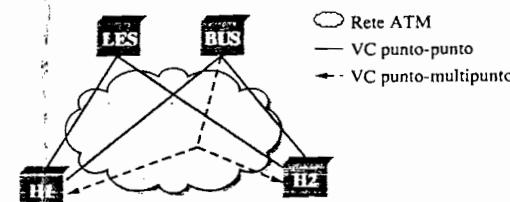


Figura 3.26 Server e client in un'emulazione di LAN.

A prima vista la connessione di un LEC al BUS richiede molto lavoro, ma la separazione delle funzioni tra i server è utile dal punto di vista della gestione della rete. Ad esempio, una grande quantità di informazioni può essere centralizzata in un singolo LECS, invece di distribuirla a molti LES; e la quantità di configurazioni specifiche necessarie in ciascun host è mantenuta veramente al minimo.*

Dovrebbe risultare evidente che il BUS rappresenta la destinazione a cui inviare qualsiasi pacchetto che debba raggiungere in modalità broadcast tutti i client della LAN; anche se lo si potrebbe usare anche per la consegna di pacchetti unicast, ciò sarebbe inefficiente, perché tale consegna avviene nel modo seguente. Immaginate che un host abbia un pacchetto da consegnare ad un certo indirizzo MAC. In una rete locale tradizionale, il pacchetto potrebbe essere semplicemente posto sulla linea e verrebbe prelevato dal destinatario previsto. In una LAN emulata, il pacchetto deve essere consegnato al destinatario mediante un circuito virtuale, ma un host appena connesso alla rete ha un circuito virtuale instaurato soltanto con il LES e il BUS, non con tale destinatario. Per peggiorare la situazione, potrebbe anche non conoscere l'indirizzo ATM del destinatario, cosa che è necessaria per instaurare un circuito virtuale. Quindi, l'host esegue i seguenti passi:

- Invia il pacchetto al BUS, perché ne conosce la capacità di consegnare il pacchetto usando il circuito virtuale punto-multipunto.
- Invia al LES una richiesta di "risoluzione di indirizzo" (address resolution): qual è l'indirizzo ATM che corrisponde a questo indirizzo MAC?

Dato che tutti i client dovrebbero aver registrato i propri indirizzi MAC e ATM presso il LES, il LES dovrebbe essere in grado di rispondere all'interrogazione e fornire l'indirizzo ATM al client, per cui ora il client è in grado di iniziare una procedura di segnalazione per instaurare un circuito virtuale verso il destinatario, circuito virtuale che potrà usare per inoltrare i successivi frame verso tale destinazione. Il motivo per cui viene usato il BUS per inviare il primo pacchetto consiste nella minimizzazione del ritardo, perché occorre un po' di tempo per ottenere la risposta dal LES e per instaurare un circuito virtuale.

Un dettaglio di questo procedimento è che, per ipotesi, le reti locali non consegnano i pacchetti fuori ordine, per cui una LAN emulata non dovrebbe comportarsi diversamente, ma se alcuni frame vengono inviati tramite il BUS e successivamente altri frame vengono inviati lungo una connessione diretta, possono verificarsi errori nell'ordinamento. Le procedure di emulazione della LAN contengono un meccanismo di "flush" (svuotamento) per garantire che l'ultimo pacchetto inviato lungo un percorso sia giunto a destinazione prima che ne venga inviato un altro lungo un diverso percorso, garantendo così la consegna ordinata.

Con la procedura sopra delineata, i client si trovano, ad un certo punto, ad avere un circuito virtuale diretto che li collega con tutte le destinazioni a cui hanno inviato dati. Questo numero di circuiti virtuali potrebbe essere eccessivo, per cui un client può usare un algoritmo di cache per eliminare quei circuiti virtuali che non ricevono più traffico. Un "cache miss" (cioè l'arrivo di un pacchetto che deve essere inviato ad una destinazione per la quale non esiste un circuito virtuale) verrà gestito inviando il pacchetto al BUS.

3.4 Implementazione e prestazioni

Fino ad ora abbiamo visto quali sono i compiti di un commutatore senza discutere come realizzarlo. Esiste un modo molto semplice per realizzare uno switch: comprare una workstation di utilizzo generale ed equipaggiarla con un certo numero di interfacce di rete. Tale apparato, con opportuno software, è in grado di ricevere pacchetti da una delle proprie interfacce, eseguire una delle funzioni di commutazione viste in precedenza ed inviare i pacchetti verso un'altra delle proprie interfacce. Questo è proprio un modo assai popolare per costruire commutatori sperimentali quando si vogliono fare cose come lo sviluppo di nuovi protocolli di instradamento, perché offre estrema flessibilità in un ambiente di programmazione ben conosciuto, e non è neppure così diverso dall'architettura di molti router di fascia bassa (che, come vedremo nel prossimo capitolo, sono molto simili agli switch).

La Figura 3.27 mostra una workstation con tre interfacce di rete usata come switch. La figura mostra anche un percorso che può essere seguito da un pacchetto dal momento in cui arriva sull'interfaccia 1 fino al momento in cui esce dall'interfaccia 2. Abbiamo qui ipotizzato che la workstation abbia modo di trasferire direttamente i dati da un'interfaccia alla propria memoria centrale, senza che siano copiati dalla CPU, cioè mediante l'accesso diretto alla memoria (DMA) descritto nella Sezione 2.9. Una volta che il pacchetto si trova nella memoria, la CPU ne esamina l'intestazione per determinare verso quale interfaccia inviarlo, poi usa il DMA per copiare il pacchetto nell'interfaccia appropriata. Notate che la Figura 3.27 non mostra il pacchetto che entra nella CPU, perché la CPU ne ispeziona soltanto l'intestazione: non deve leggere tutti i byte di dati presenti nel pacchetto.

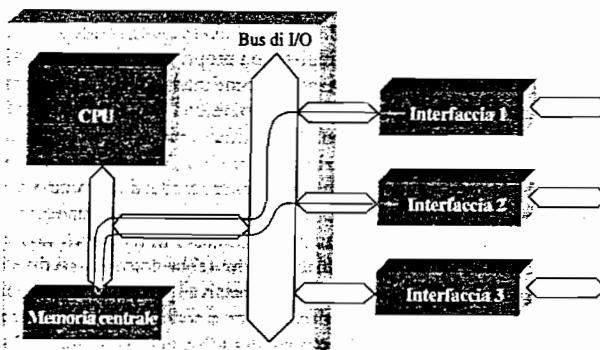


Figura 3.27 Una workstation usata come commutatore di pacchetti.

3.4 Implementazione e prestazioni

Il problema principale quando si usa una workstation come commutatore sono le sue prestazioni, che sono limitate dal fatto che tutti i pacchetti devono passare attraverso un unico punto dove avviene la contesa: nell'esempio, tutti i pacchetti attraversano il bus di I/O due volte e vengono scritti in memoria e letti dalla memoria una volta. Il limite superiore del throughput aggregato di tale dispositivo (la velocità dei dati totale, sommando quella di tutti gli ingressi, che può essere sostenuta per un tempo prolungato) è, quindi, il valore minore tra la metà dell'ampiezza di banda della memoria centrale e la metà dell'ampiezza di banda del bus di I/O (solitamente, l'ampiezza di banda del bus di I/O è inferiore). Ad esempio, una workstation con un bus di I/O a 32 bit e a 33 MHz può trasmettere dati da una velocità di picco di più di 1 Gbps. Poiché inoltrare un pacchetto richiede due attraversamenti del bus, il limite reale è 500 Mbps, che è sufficiente a far operare cinque schede d'interfaccia Ethernet a 100 Mbps. In pratica, però, l'ampiezza di banda di picco non è sostenibile per un tempo prolungato, per cui è molto più probabile che una tale workstation possa far operare soltanto tre o quattro di tali schede di interfaccia.

Inoltre, questo limite superiore è riferito anche all'ipotesi che l'unico problema sia il trasferimento dei dati, un'approssimazione realistica per pacchetti di grandi dimensioni ma pessima quando i pacchetti sono piccoli. Nel secondo caso, il costo di elaborazione del pacchetto (decodificarne l'intestazione e decidere verso quale linea d'uscita inviarlo) è probabilmente dominante. Supponete, ad esempio, che una workstation sia in grado di compiere tutte le elaborazioni necessarie per inoltrare 500000 pacchetti al secondo, una quantità che viene a volte chiamata "velocità in pacchetti al secondo" (pps, packet per second) ed il cui valore indicato è rappresentativo di quanto si possa fare con un odierno PC di fascia alta. Se la dimensione media dei pacchetti è bassa, diciamo 64 byte, si ottiene:

$$\begin{aligned} \text{Throughput} &= \text{pps} \times (\text{BitsPerPacket}) \\ &= 500 \times 10^3 \times 64 \times 8 \\ &= 256 \times 10^6 \end{aligned}$$

cioè un throughput di 256 Mbps, significativamente inferiore a quanto gli utenti oggi chiedono alle proprie reti tenendo presente che questi 256 Mbps sarebbero suddivisi tra tutti gli utenti connessi allo switch, esattamente come i 10 Mbps di una Ethernet sono suddivisi tra tutti gli utenti connessi al mezzo fisico condiviso. Quindi, ad esempio, uno switch a 10 porte con questo valore di throughput aggregato sarebbe soltanto in grado di fornire una velocità media per i dati di 25.6 Mbps su ciascuna porta.

Per risolvere questo problema, i progettisti di hardware hanno prodotto un'ampia scelta di progetti per computeratori che riducono la quantità di contesa e forniscono elevati valori di throughput aggregato. Notate che la contesa è inevitabile: se tutti gli ingressi hanno dati da inviare ad un'unica uscita, non possono farlo tutti insieme. Tuttavia, se i dati destinati ad uscite diverse giungono da ingressi diversi, un commutatore ben progettato sarà in grado di trasferire i dati dagli ingressi alle uscite in parallelo, aumentando così il valore del throughput aggregato.

3.4.1 Porte

La maggior parte dei commutatori sono concettualmente simili a quello di Figura 3.28. Sono costituiti da un certo numero di *porte d'ingresso* e di *porte d'uscita* e da una *matrice* (fabric). Solitamente vi è almeno un controllore che si occupa dell'intero commutatore e che comuni-

Definire il throughput

Non è semplice definire in modo preciso cosa sia il throughput di uno switch. Intuitivamente potremmo pensare che se uno switch ha n ingressi, ciascuno dei quali sostiene una velocità di linea s_i , allora il throughput è semplicemente la somma di tutti i valori s_i . In realtà, questo valore rappresenta il miglior throughput che può essere fornito da uno switch, ma in pratica quasi nessuno switch reale può garantire questo livello di prestazioni, per un motivo facile da comprendere: supponete che, per un certo periodo di tempo, tutto il traffico in arrivo verso lo switch debba essere inviato alla stessa uscita. Nel caso in cui l'ampiezza di banda di tale uscita sia inferiore alla somma delle ampiezze di banda degli ingressi, una parte del traffico dovrà essere memorizzato nei buffer o eliminato. Con questo particolare schema di traffico, lo switch non è in grado di fornire per un periodo di tempo prolungato un throughput più elevato della velocità della linea connessa a tale uscita. Uno switch potrebbe, invece, essere in grado di gestire il traffico in arrivo alla massima velocità da tutti gli ingressi se tale traffico si distribuisce equamente fra tutte le uscite: questa situazione è da considerarsi ottimale.

Un altro fattore che agisce sulle prestazioni degli switch è la dimensione dei pacchetti in arrivo ai loro ingressi. Per uno switch ATM, questo normalmente non è un problema, perché tutti i "pacchetti" (celle) hanno la medesima lunghezza, ma per gli switch Ethernet o i router IP sono possibili pacchetti di dimensioni fortemente variabili. Alcune delle operazioni che devono essere eseguite da uno switch richiedono un tempo costante per ciascun pacchetto, per cui è probabile che uno switch abbia prestazioni diverse in relazione al fatto che i pacchetti in arrivo siano molto piccoli, molto grandi o misti. Per questa ragione i router e gli switch che inoltrano pacchetti di dimensioni variabili vengono spesso caratterizzati dalla velocità in *pacchetti al secondo* (pps, packet per second) oltre che dall'throughput in bit al secondo. La velocità pps viene solitamente misurata con pacchetti di dimensioni minime.

La prima cosa da notare in questa discussione è che il throughput di uno switch è una funzione del traffico a cui è sottoposto. Una delle attività per le quali i progettisti di commutatori investono molto tempo è la ricerca di un modello di traffico che approssimi il comportamento del traffico di dati reale, anche se si è visto che è molto difficile identificare modelli accurati. Un modello di traffico tenta di rispondere ad alcune importanti domande: (1) Quando arrivano i pacchetti? (2) Verso quale uscita sono diretti? (3) Che dimensione hanno?

La modellistica del traffico è una scienza ben assentata che ha avuto molto successo nel mondo della telefonia, consentendo alle compagnie telefoniche di progettare le proprie reti per trasportare in modo abbastanza efficiente i carichi attesi. Ciò è in parte dovuto al fatto che il modo in cui le persone usano la rete telefonica non cambia molto nel tempo: la frequenza con cui si fanno telefonate, la quantità di tempo impiegata per una telefonata e la tendenza di ognuno a fare telefonate nel giorno della festa della mamma sono rimaste abbastanza costanti per molti anni⁴. Al contrario, la rapida evoluzione delle comunicazioni tra calcolatori, dove una nuova applicazione come Napster può modificare lo schema di traffico da un giorno all'altro, rende molto più difficoltosa la

⁴ Questa affermazione è recentemente divenuta meno vera con l'avvento del fax e delle connessioni a Internet via modem.

modellistica efficace delle reti di calcolatori. Ciò nonostante, esistono in materia alcuni eccellenti testi e articoli, che elenchiamo al termine del capitolo.

Per darvi un'idea dei valori di throughput di cui si devono occupare i progettisti, un router di fascia alta usato in Internet nel momento in cui scriviamo è in grado di fornire supporto a 10 linee OC-192 per un throughput approssimativamente uguale a 100 Gbps. Uno switch a 100 Gbps, se chiamato a gestire un flusso costante di pacchetti lunghi 64 byte, dovrebbe avere una velocità, in pacchetti al secondo, uguale a

$$100 \times 10^9 / (64 \times 8) = 195 \times 10^6 \text{ pps}$$

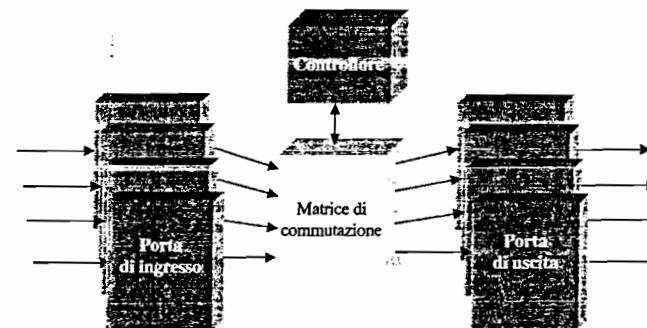


Figura 3.28 Uno switch 4 × 4.

ca con le porte direttamente oppure, come mostrato qui, tramite la matrice di commutazione. Le porte comunicano con il mondo esterno; possono contenere ricevitori per fibre ottiche e laser, buffer per memorizzare i pacchetti in attesa di essere smistati o trasmessi e spesso anche una quantità significativa di altra circuiteria che consente allo switch di funzionare. La matrice di commutazione ha un compito molto semplice e ben definito: quando le viene presentato un pacchetto, deve consegnarlo alla porta d'uscita corretta.

Uno dei compiti delle porte, quindi, consiste nella gestione della complessità del mondo reale, in modo che la matrice di commutazione possa svolgere il proprio compito relativamente semplice. Ad esempio, supponete che questo switch fornisca il supporto per un modello di comunicazione a circuito virtuale: in generale, le tabelle dei circuiti virtuali descritte nella Sezione 3.1.2 saranno localizzate nelle porte, che gestiscono elenchi degli identificatori dei circuiti virtuali attualmente in uso, con informazioni relative alla porta di uscita verso la quale va indirizzato un pacchetto avente un certo valore di VCI e come i valori di VCI vanno sostituiti per garantire la loro unicità sulla linea di uscita. Analogamente, le porte di uno switch Ethernet memorizzano tabelle che traducono indirizzi Ethernet in porte d'uscita (le tabelle di inoltro dei bridge descritte nella Sezione 3.2). In generale, quando un pacchetto viene inviato da una porta d'ingresso alla matrice di commutazione, la porta ha già individuato verso quale porta deve andare il pacchetto e configura di conseguenza la matrice comunicandole alcune informazioni di controllo, oppure aggiunge al pacchetto informazioni sufficienti (come, ad esempio, il numero della porta d'uscita) per consentire alla matrice di

svolgere il proprio compito in modo automatico. Le matrici che smistano i pacchetti guardando solamente le informazioni contenute nel pacchetto o ad esso allegate vengono chiamate "auto-instrandanti" (self-routing), poiché non necessitano di un controllo esterno per instradare i pacchetti. Nel seguito parleremo di un esempio di matrice auto-instrandante.

La porta d'ingresso è la prima cosa da prendere in esame per cercare colli di bottiglia nelle prestazioni. Una porta d'ingresso deve ricevere un flusso costante di pacchetti, analizzare le informazioni nell'intestazione di ciascuno di essi per determinare verso quale porta (o quali porte) il pacchetto deve essere inviato, e trasferire il pacchetto alla matrice di commutazione. Il tipo di analisi dell'intestazione che deve eseguire può andare da una semplice ricerca in una tabella usando un valore di VCI fino a complessi algoritmi di identificazione di corrispondenze che prendono in esame molti campi dell'intestazione. Questo è il tipo di operazioni che a volte diventano problematiche quando la dimensione media dei pacchetti è molto piccola. Ad esempio, pacchetti di 64 byte che arrivano ad una porta connessa ad una linea di collegamento OC-48 richiedono una velocità di elaborazione dei pacchetti pari a

$$2.48 \times 10^9 / (64 \times 8) = 4.83 \times 10^6 \text{ pps}$$

In altre parole, quando su questa linea arrivano piccoli pacchetti alla massima velocità possibile (la situazione peggiore per la cui gestione vengono progettate la maggior parte delle porte), la porta d'ingresso ha a disposizione circa 200 nanosecondi per elaborare ciascun pacchetto.

Un'altra funzione chiave svolta dalle porte è la memorizzazione nei buffer, che può avvenire sia nelle porte d'ingresso sia in quelle d'uscita, così come può avvenire anche all'interno della matrice di commutazione (nel qual caso si parla di *memorizzazione interna*, internal buffering). La semplice memorizzazione in ingresso presenta alcune serie limitazioni. Considerate un buffer d'ingresso realizzato mediante una coda FIFO (first-in, first-out): mentre i pacchetti giungono allo switch, vengono inseriti nel buffer d'ingresso, dopodiché lo switch tenta di inoltrare i pacchetti che si trovano al primo posto di ciascuna coda FIFO verso la loro porta d'uscita corretta. Tuttavia, se i pacchetti che si trovano nella prima posizione in diverse porte d'ingresso hanno come destinazione la medesima porta d'uscita, soltanto uno di essi può essere inoltrato⁵ e gli altri rimangono nel proprio buffer d'ingresso.

Lo svantaggio di questa configurazione è che i pacchetti rimasti al primo posto della coda d'ingresso impediscono ad altri pacchetti più interni al buffer di tentare di procedere verso la propria uscita, anche se magari verso quella direzione non c'è contesa. Questo fenomeno viene chiamato *blocco della posizione frontale* (head-of-line blocking), di cui in Figura 3.29 potete vedere un semplice esempio, dove un pacchetto destinato alla porta 1 si trova bloccato dietro un pacchetto che sta tentando di raggiungere la porta 2, per la quale c'è contesa. Si può dimostrare che, quando il traffico è distribuito uniformemente fra le uscite, il blocco della posizione frontale limita il throughput di uno switch con buffer d'ingresso al 59% del suo massimo teorico (che è la somma delle ampiezze di banda delle linee entranti nello switch). Di conseguenza, la maggioranza degli switch usa la pura memorizzazione nei buffer d'uscita oppure una combinazione di memorizzazione interna e in uscita, mentre quei

⁵ In un semplice switch con buffer d'ingresso, si può inviare ad una certa porta d'uscita un solo pacchetto per volta. È possibile progettare switch che sono in grado di inoltrare contemporaneamente più di un pacchetto verso la stessa uscita, al prezzo di una maggior complessità dello switch stesso, ma esiste sempre un limite superiore al numero di pacchetti.

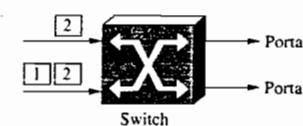


Figura 3.29 Semplice raffigurazione del blocco della posizione frontale.

pochi che si affidano ai buffer d'ingresso usano sofisticati schemi di gestione dei buffer per evitare il blocco della posizione frontale.

In realtà i buffer svolgono un compito più complesso della semplice memorizzazione dei pacchetti mentre sono in attesa di essere trasmessi. I buffer sono la fonte principale di ritardo in uno switch, ed anche il punto dove è più probabile che i pacchetti vengano eliminati per mancanza di spazio dove memorizzarli. I buffer, quindi, sono il punto fondamentale in cui si determinano le caratteristiche della qualità di servizio di uno switch. Ad esempio, se un pacchetto è stato inviato lungo un circuito virtuale che ha un ritardo garantito, non è possibile che venga trattenuto troppo a lungo in un buffer: ciò significa che i buffer, in generale, devono essere gestiti secondo le priorità dei pacchetti e con algoritmi di eliminazione che soddisfino un ampio spettro di requisiti di qualità di servizio. Ne parleremo di nuovo nel Capitolo 6.

3.4.2 Matrici di commutazione

Nonostante ci sia grande abbondanza di ricerche significative condotte in merito alla progettazione di matrici di commutazione efficienti e scalabili, per i nostri scopi è sufficiente ora capire soltanto le loro proprietà di alto livello. Una matrice di commutazione di uno switch deve essere in grado di trasferire pacchetti da porte d'ingresso a porte d'uscita con il minimo ritardo ed in un modo che soddisfi gli obiettivi di throughput dello switch. Ciò solitamente significa che le matrici mostrano un qualche grado di parallelismo: una matrice ad elevate prestazioni con n porte è spesso in grado di trasferire simultaneamente un pacchetto da ciascuna delle sue n porte ad una porta d'uscita. Un campione di tipi di matrici di commutazione comprende i seguenti:

- *A bus condiviso* (shared-bus). Questo è il tipo di "matrice di commutazione" che si trova in una normale workstation utilizzata come switch, nel modo descritto in precedenza. Dato che l'ampiezza di banda del bus determina il throughput dello switch, i commutatori ad elevate prestazioni hanno solitamente bus progettati appositamente al posto dei normali bus di cui sono dotati i PC.
- *A memoria condivisa* (shared-memory). In uno switch a memoria condivisa i pacchetti vengono scritti nella memoria principale da una porta d'ingresso e vengono letti nella memoria dalle porte d'uscita. In questo caso il throughput dello switch è determinato dall'ampiezza di banda della memoria, per cui in questo tipo di progetto si usano memorie veloci e ad elevato parallelismo. Uno switch a memoria condivisa è concettualmente simile allo switch a bus condiviso, tranne per il fatto che, invece di usare un bus di I/O, usa memorie ad alta velocità e progettate appositamente.
- *A rete di interconnessione* (crossbar). Uno switch di tipo crossbar è una matrice di percorsi che possono essere configurati per connettere qualsiasi porta d'ingresso a qualsiasi porta d'uscita. La Figura 3.30 mostra uno switch di tipo crossbar 4 × 4. Il problema

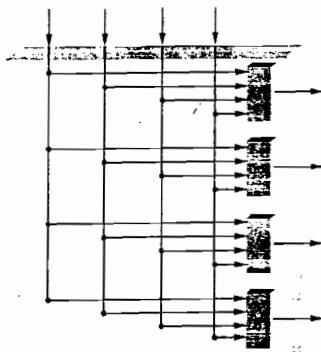


Figura 3.30 Uno switch di tipo crossbar 4×4 .

principale di queste reti di interconnessione è che, nella loro forma più semplice, richiedono che ogni porta d'uscita sia in grado di accettare pacchetti da tutti gli ingressi contemporaneamente, per cui ciascuna porta dovrebbe avere un'ampiezza di banda di memoria pari al throughput totale dello switch. In realtà, tipicamente si usano progetti più complessi per risolvere questo problema (si vedano, ad esempio, lo switch Knockout e l'approccio virtuale ad uscita con buffer di McKeown, i cui riferimenti si trovano nella sezione "Ulteriori letture" al termine del capitolo).

- **Auto-instradanti (self-routing).** Come già detto, le matrici di tipo self-routing sfruttano alcune informazioni contenute nell'intestazione del pacchetto per inviarlo all'uscita corretta. Solitamente la porta d'ingresso, dopo aver determinato l'uscita verso cui deve andare il pacchetto, aggiunge al pacchetto stesso una sorta di "intestazione per l'auto-instradamento" (self-routing header), come illustrato in Figura 3.31: questa intestazione aggiuntiva viene rimossa prima che il pacchetto abbandoni il commutatore. Le matrici di tipo self-routing sono spesso costruite mediante un gran numero di elementi di commutazione 2×2 interconnessi in uno schema regolare, come nelle matrici di commutazione *banyane* mostrate in Figura 3.32. Nella sezione "Ulteriori letture" al termine del capitolo potrete trovare alcuni esempi di progetti di matrici di tipo self-routing.

Le matrici di tipo self-routing rappresentano gli approcci maggiormente scalabili fra i progetti di matrici di commutazione ed a questo proposito sono state condotte molte ricerche, alcune delle quali sono elencate nella sezione "Ulteriori letture". Molte matrici di tipo self-routing sono simili a quella mostrata in Figura 3.32, composta di elementi di commutazione 2×2 interconnessi in modo regolare. Ad esempio, i commutatori 2×2 delle reti banyane svolgono un compito semplice: osservano un bit in ciascuna intestazione per l'auto-instradamento ed instradano i pacchetti verso l'uscita superiore se il bit vale 0 oppure verso l'uscita inferiore se il bit vale 1. Ovviamente, se due pacchetti arrivano in un elemento banyano nello stesso istante ed hanno il bit con lo stesso valore, significa che vogliono essere instradati verso la stessa uscita e si ha una collisione. Prevenire o gestire queste collisioni è il problema maggiore della progettazione degli switch di tipo self-routing. La rete banyana è un intelligente disposizione di elementi di commutazione 2×2 che instrada tutti i pacchetti verso l'uscita corretta senza collisioni nel caso in cui i pacchetti si presentino in ordine crescente.

3.4 Implementazione e prestazioni

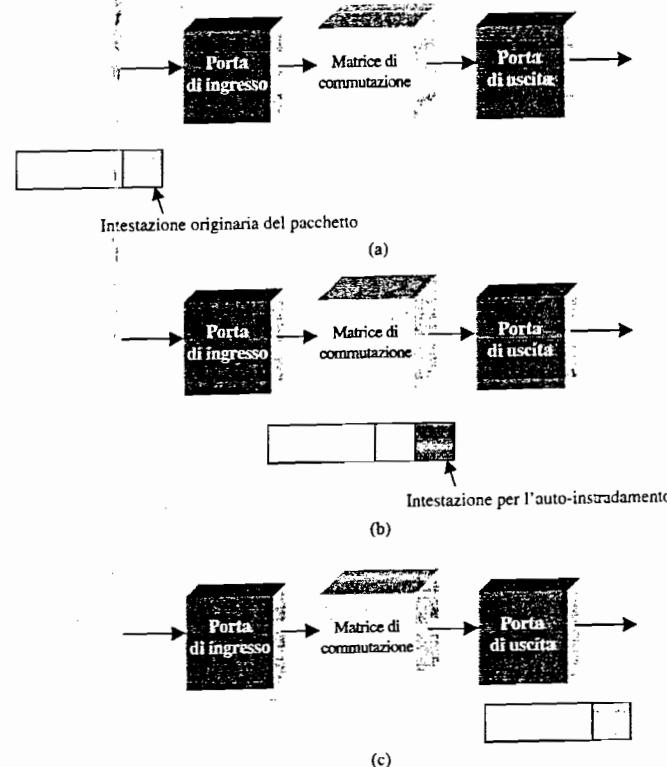


Figura 3.31 Ai pacchetti in ingresso viene applicata un'intestazione per l'auto-instradamento, che consente alla matrice di inviare il pacchetto verso l'uscita corretta, dove viene poi rimossa. (a) Il pacchetto giunge alla porta d'ingresso. (b) La porta d'ingresso aggiunge l'intestazione per l'auto-instradamento per dirigere il pacchetto verso l'uscita corretta. (c) L'intestazione per l'auto-instradamento viene eliminata dalla porta d'uscita prima che il pacchetto esca dallo switch.

Possiamo vederne il funzionamento con un esempio, relativo alla Figura 3.32, dove l'intestazione per l'auto-instradamento contiene il numero della porta d'uscita codificato in binario. Gli elementi di commutazione della prima colonna considerano il bit più significativo del numero della porta d'uscita e instradano i pacchetti verso l'alto se tale bit vale 0 e verso il basso se vale 1. Gli elementi di commutazione della seconda colonna considerano il secondo bit dell'intestazione, mentre quelli della terza colonna considerano il bit meno significativo. Osservando questo esempio potete notare che i pacchetti vengono instradati verso la porta di destinazione corretta senza collisioni. Notate come le uscite superiori degli elementi della prima colonna di commutatori portino tutte verso la metà superiore della rete, trasferendo così i pacchetti con numeri di porta che vanno da 0 a 3 nella metà corretta della rete. La

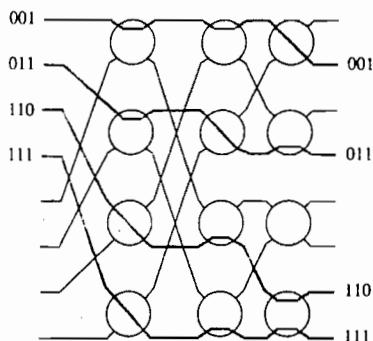


Figura 3.32 Instradamento di pacchetti attraverso una rete banyana. I numeri a 3 bit rappresentano i valori presenti nelle intestazioni di auto-instradamento di quattro pacchetti in arrivo.

colonna successiva trasferisce i pacchetti nel quarto corretto della rete, mentre la colonna terminale trasferisce i pacchetti alla porta d'uscita giusta. La parte interessante è il modo in cui gli elementi di commutazione sono disposti per evitare le collisioni; fa parte di questa strategia lo schema di collegamenti a "mescolamento perfetto" (perfect shuffle) della porzione iniziale della rete. Per costruire una matrice di commutazione completa attorno ad una rete banyana sono necessari ulteriori componenti per ordinare i pacchetti prima che si presentino agli ingressi della rete banyana. Il progetto di switch Batcher-banyano è un esempio notevole di tale approccio. La rete di Batcher, anch'essa costruita mediante l'interconnessione regolare di elementi di commutazione 2×2 , ordina i pacchetti in ordine decrescente. Quando escono dalla rete di Batcher, quindi, i pacchetti sono pronti per essere indirizzati dalla rete banyana verso l'uscita corretta, senza alcun rischio di collisione.

Una delle cose interessanti a proposito del progetto di commutatori è la grande varietà di tipi di commutatori che si possono costruire con la stessa tecnologia di base. Ad esempio, gli switch Ethernet e gli switch ATM discussi in questo capitolo, così come i router di Internet che verranno discussi nel capitolo successivo, sono tutti costruiti usando progetti simili a quelli delineati in questa sezione.

3.5 Riepilogo

Questo capitolo è iniziato con l'esame di alcuni problemi relativi alla costruzione di grandi reti scalabili usando commutatori, invece di sole linee di collegamento, per interconnettere gli host. I pacchetti si possono smistare in alcuni modi diversi: i due principali sono il modello datagram (privò di connessione) ed il modello a circuito virtuale (orientato alla connessione).

Un'importante applicazione della commutazione è l'interconnessione di LAN a mezzo fisico condiviso. I commutatori per LAN, detti anche bridge, usano tecniche quali l'apprendimento dell'indirizzo di sorgente per migliorare l'efficienza d'inoltro e algoritmi basati su spanning tree per evitare i cicli. Questi commutatori sono ampiamente utilizzati nelle reti aziendali, universitarie e di centri di elaborazione dati.

L'utilizzazione più diffusa della commutazione a circuito virtuale si ha nei commutatori Frame Relay e ATM. La tecnologia ATM presenta alcuni problemi specifici, dovuti all'uso di celle, cioè di piccoli pacchetti di lunghezza fissa. La disponibilità di commutatori ATM con throughput relativamente elevato ha contribuito alla diffusione di tale tecnologia, sebbene non abbia assolutamente spazzato via le altre tecnologie, come invece alcuni avevano predetto. Oggigiorno l'uso prevalente di ATM riguarda l'interconnessione di sedi molto distanti in reti aziendali.

Indipendentemente dalle specificità di ciascuna tecnologia di commutazione, gli switch devono inoltrare ad alta velocità pacchetti dai propri ingressi alle proprie uscite ed in alcuni casi gli switch devono avere dimensioni tali da poter gestire centinaia o migliaia di porte. La costruzione di switch che siano sia scalabili sia ad elevate prestazioni con costi accettabili è resa più complessa dal problema della contesa e, di conseguenza, spesso gli switch utilizzano hardware appositamente realizzato piuttosto di essere realizzati con workstation di uso generale.

Oltre al problema della contesa qui discusso, possiamo notare che nel corso del capitolo è emerso anche il problema correlato della congestione, la cui discussione avverrà nel Capitolo 6, dopo che avremo visto qualcosa di più in merito alle architetture di rete. Facciamo così perché è impossibile comprendere appieno la congestione (sia il problema sia le sue soluzioni) senza capire sia ciò che accade all'interno della rete (e che è argomento di questo e del prossimo capitolo) sia ciò che accade ai margini della rete (che sarà argomento del Capitolo 5).

Problema aperto Il futuro di ATM

Originariamente, la tecnologia ATM venne vista da molti dei suoi sostenitori come la base su cui fondare la "rete digitale a larga banda integrata nei servizi" (BISDN, Broadband Integrated Services Digital Network) e si fece la previsione che: ATM avrebbe sostituito tutte le altre tecnologie di rete nel giro di qualche trimestre; gli host sarebbero stati dotati di adattatori ATM invece che di porte Ethernet, consentendo di ottenere "ATM sulla scrivania"; le compagnie telefoniche avrebbero portato ATM ovunque e, in qualità di tecnologia in grado di veicolare qualsiasi tipo di informazione (voce, video, dati), avrebbe eliminato il bisogno di qualsiasi altro tipo di rete.

Oggi è evidente che è molto improbabile che si assista a questo scenario. In particolare, il successo dei commutatori Ethernet ha spiazzato la filosofia "ATM sulla scrivania" e le tecnologie Gigabit Ethernet e 10-Gigabit Ethernet hanno risolto con successo il problema di gestire connessioni ad elevata velocità verso i server, dove effettivamente si sarebbe potuto utilizzare ATM. Oggi addirittura sentiamo parlare di ATM come di un "protocollo legacy", cioè "antiquato", un termine che fu molto usato nei giorni di gloria di ATM in riferimento ai protocolli precedenti.

Un altro fattore che ha limitato la diffusione di ATM è stato il successo di Internet. Oggigiorno è assodato che molti utenti sono disposti a pagare per avere accesso a Internet, cioè a comprare un servizio di consegna di pacchetti IP. Anche se ATM può essere usata per agevolare la fornitura di tale servizio (come accade, ad esempio, in molte reti DSL), la semplice vendita di connessioni ATM agli utenti non soddisfa i loro bisogni di una rete per dati. L'unica eccezione significativa è costituita da clienti aziendali che vogliono interconnettere molte sedi, nel qual caso un circuito virtuale ATM può essere la soluzione giusta per rimpiaz-

zare, risparmiando, una linea affittata: in realtà, oggi questa è proprio la nicchia di mercato principale per ATM, che viene usata (insieme alla tecnologia Frame Relay) per fornire servizi a circuito virtuale su rete globale a clienti per reti aziendali.

Il futuro di ATM, quindi, sembra essere basato sul futuro dei servizi di rete globale basati su circuito virtuale: è improbabile che questi servizi svaniscano nell'immediato futuro, ma il ruolo di ATM è in qualche modo messo in discussione da nuove tecnologie, come i tunnel IP in codice e la commutazione con etichette multiprotocollo (MPLS, Multiprotocol Label Switching), che verranno descritte nel prossimo capitolo.

Ulteriori letture

Il lavoro fondamentale sui bridge, ed in particolare sull'algoritmo a spanning tree, è l'articolo di Perlman qui indicato. Per ATM esistono parecchie pubblicazioni che forniscono una buona panoramica: l'articolo di Turner, un pioniere di ATM, fu uno dei primi a proporre l'uso di una rete per servizi integrati basata su celle. La terza pubblicazione descrive il commutatore Sunshine ed è interessante soprattutto perché fornisce una visione approfondita del ruolo importante assunto dall'analisi di traffico nella progettazione dei commutatori. In particolare i progettisti di Sunshine furono tra i primi a capire che è molto improbabile che le celle arrivino al commutatore in modo totalmente incorrelato, per cui furono in grado di sfruttare queste correlazioni nel loro progetto. Infine, il lavoro di McKeown descrive un approccio alla progettazione di commutatori che internamente usano celle ma che sono stati usati commercialmente come base per router ad elevate prestazioni in grado di inoltrare pacchetti di dimensione variabile.

- Perlman, R. "An algorithm for distributed computation of spanning trees in an extended LAN", *Proceedings of the Ninth Data Communications Symposium*, pagg. 44-53, September 1985.
- Turner, J.S. "Design of an integrated services packet network", *Proceedings of the Ninth Data Communications Symposium*, pagg. 124-133, September 1985.
- Giacopelli, J.N. et al. "Sunshine: A high-performance self-routing broadband packet-switched architecture", *IEEE Journal of Selected Areas in Communications (JSAC)* 9(8):1289-1298, October 1991.
- McKeown, N. "The iSLIP scheduling algorithm for input-queued switches", *IEEE Transactions on Networking* 7(2):188-201, April 1999.

Una buona panoramica generale sui bridge si può trovare in un altro lavoro di Perlman [Per00]. Per una descrizione dettagliata di molti aspetti di ATM, con particolare attenzione alla costruzione di reti reali, raccomandiamo la lettura del libro di Ginsburg [Gin99]. Ancora, nella veste di uno degli enti chiave per la definizione di standard relativi ad ATM, ATM Forum fornisce nuove specifiche per ATM: la specifica UNI (User-Network Interface), nella versione 4.1, è la più recente nel momento in cui scriviamo (consultate il riferimento attivo fornito in seguito).

Sulle architetture dei commutatori sono stati pubblicati letteralmente migliaia di lavori. Una delle prime pubblicazioni che illustra bene le reti Batcher è, verosimilmente, di Batcher stesso [Bat68]. Le reti di ordinamento sono illustrate da Drysdale e Young [DY75], mentre il commutatore Knockout, un'interessante variante di crossbar, è descritto da Yeh et al. [YHA87].

Nel lavoro di Partridge [Par94] si può trovare una panoramica delle architetture dei commutatori ATM, mentre nel lavoro di Robertazzi [Rob93] compare una buona analisi delle prestazioni di diverse matrici di commutazione. In Gopal e Guerin [GG94] si può trovare un esempio del progetto di un commutatore per pacchetti di lunghezza variabile.

Le reti ottiche sono un settore assai ricco e a sé stante, con le proprie riviste, conferenze e così via. Raccomandiamo il libro di Ramaswami e Sivarajan [RS01] come valido testo introduttivo sull'argomento.

Un testo eccellente da leggere se volete saperne di più sull'analisi matematica delle prestazioni delle reti è quello di Kleinrock [Kle75], uno dei pionieri di ARPANET. Sono stati pubblicati molti lavori sulle applicazioni della teoria delle code alla commutazione di pacchetti. Raccomandiamo l'articolo di Paxson e Floyd [PF94] in qualità di contributo significativo avente Internet come argomento centrale, nonché un lavoro di Leland et al. [LTWW94] che introduce l'importante concetto di "dipendenza a lungo raggio" e mostra l'inadeguatezza di molti approcci tradizionali per la modellistica del traffico.

Infine, raccomandiamo il seguente riferimento attivo:

- <http://www.atmforum.com/>: attività in corso dell'ATM Forum

Esercizi

- Usando la rete di esempio fornita in Figura 3.33, individuate le tabelle dei circuiti virtuali per tutti i commutatori dopo che ciascuna delle seguenti connessioni è stata instaurata. Ipotizzate che la sequenza di connessioni sia cumulativa, cioè la prima connessione è ancora attiva quando si instaura la seconda, e così via. Ipotizzate anche che l'asse-

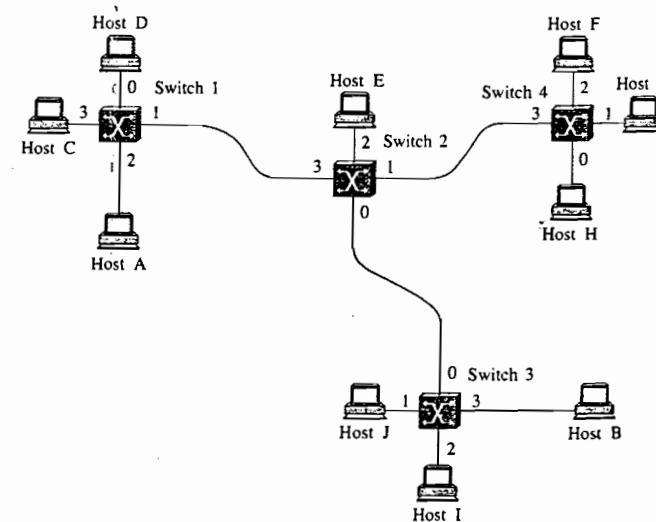


Figura 3.33 Rete di esempio per gli Esercizi 1 e 2.

gnamento del valore di VCI scelga sempre il valore minimo tra quelli inutilizzati su ciascuna linea, iniziando dal valore 0.

- a) L'host A si connette all'host B.
 - b) L'host C si connette all'host G.
 - c) L'host E si connette all'host I.
 - d) L'host D si connette all'host B.
 - e) L'host F si connette all'host J.
 - f) L'host H si connette all'host A.
- ✓ 2. Usando la rete di esempio fornita in Figura 3.33, individuate le tabelle dei circuiti virtuali per tutti i commutatori dopo che ciascuna delle seguenti connessioni è stata instaurata. Ipotizzate che la sequenza di connessioni sia cumulativa, cioè la prima connessione è ancora attiva quando si instaura la seconda, e così via. Ipotizzate anche che l'assegnamento del valore di VCI scelga sempre il valore minimo tra quelli inutilizzati su ciascuna linea, iniziando dal valore 0.
- a) L'host D si connette all'host H.
 - b) L'host B si connette all'host G.
 - c) L'host F si connette all'host A.
 - d) L'host H si connette all'host C.
 - e) L'host I si connette all'host E.
 - f) L'host H si connette all'host J.
3. Per la rete di Figura 3.34, fornite per ciascun nodo la tabella di inoltro datagram. Le linee sono etichettate con il relativo costo: le tabelle devono inoltrare ciascun pacchetto verso la sua destinazione usando il percorso di costo minimo.
4. Scrivete le tabelle di inoltro per gli switch S1, S2, S3 e S4 della Figura 3.35. Ciascuno switch deve avere un instradamento di "default", scelto per inoltrare verso OUT i pacchetti con indirizzo di destinazione non riconosciuto. Devono, quindi, essere eliminate dalla tabella eventuali informazioni che duplichino quella di default.

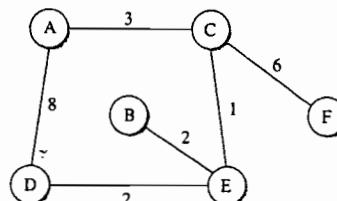


Figura 3.34 Rete per l'Esercizio 3.

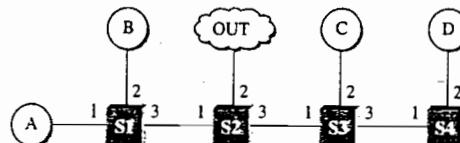


Figura 3.35 Schema per l'Esercizio 4.

5. Considerate gli switch a circuito virtuale di Figura 3.36. La Tabella 3.5 elenca, per ciascuno switch, quali coppie (porta, VCI) (oppure (VCI, interfaccia)) sono connesse a quali altre. Le connessioni sono bidirezionali. Fate un elenco di tutte le connessioni da un estremo all'altro.
6. Nell'esempio di source routing della Sezione 3.1.3 l'indirizzo ricevuto da B non è reversibile e non aiuta B a sapere come raggiungere A: proponete una modifica al meccanismo di consegna che consenta la reversibilità. Il meccanismo *non* deve richiedere l'assegnazione di un nome globalmente univoco a tutti gli switch.
7. Proponete un meccanismo che possa essere usato dagli switch a circuito virtuale in modo che, se uno switch perde la propria memorizzazione dello stato di tutte le connessioni, una sorgente che invia pacchetti lungo un percorso che attraversa tale switch venga informato del guasto.
8. Proponete un meccanismo che possa essere usato dagli switch datagram in modo che, se uno switch perde interamente o in parte la propria tabella di inoltro, le sorgenti coinvolte venganoificate del guasto.
9. Il meccanismo di circuito virtuale descritto nella Sezione 3.1.2 ipotizza che tutte le linee di collegamento siano punto-punto. Estendete l'algoritmo di inoltro in modo che funzioni anche nel caso in cui le linee siano connessioni a mezzo fisico condiviso, ad esempio, Etheræt.
10. Supponete che nella Figura 3.4 venga aggiunta una nuova linea di collegamento che connetta la porta 1 dello switch 3 (dove ora si trova G) e la porta 0 dello switch 1 (dove ora si trova D); nessuno switch è "a conoscenza" di tale linea. Inoltre, lo switch 3 pensa, erroneamente, che l'host B sia raggiungibile tramite la porta 1.
 - a) Cosa accade se l'host A tenta di inviare dati all'host B, usando l'inoltro di datagrammi?
 - b) Cosa accade se l'host A tenta di inviare dati all'host B, usando il meccanismo di instaurazione di circuiti virtuali discusso nel testo?
11. Fornite un esempio di un circuito virtuale funzionante il cui percorso attraversa alcune linee due volte. Nonostante ciò, i pacchetti inviati lungo tale percorso *non* devono viaggiare indefinitamente.

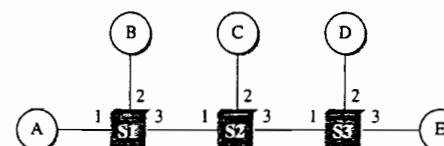


Figura 3.36 Schema per l'Esercizio 5.

Tabella 3.5 Tabelle dei VCI per gli switch di Figura 3.36.

Switch S1		Switch S2		Switch S3	
Porta VCI					
1 2	3 1	1 1	3 3	1 3	2 1
1 1	2 3	1 2	3 2	1 2	3 1
2 1	3 2	-	-	-	-

12. Nella Sezione 3.1.2 ciascuno switch sceglieva il valore di VCI per la linea d'ingresso. Dimostrate che è anche possibile che ciascuno switch scelga il valore di VCI per la linea d'uscita, e che entrambi gli approcci scelgano i medesimi valori di VCI. Se ciascuno switch sceglie il valore di VCI uscente, è ancora necessario attendere un tempo uguale a RTT prima di inviare dati?
13. Nel caso della rete locale estesa di Figura 3.37, indicate quali porte non vengono selezionate dall'algoritmo a spanning tree.
- ✓ 14. Nel caso della rete locale estesa di Figura 3.37, ipotizzate che il bridge B1 si guasti in modo irreparabile. Indicate quali porte non vengono selezionate dall'algoritmo a spanning tree dopo che si è formato un nuovo albero.
15. Considerate la disposizione di bridge ad apprendimento mostrata in Figura 3.38. Ipotizzando che inizialmente siano tutte vuote, identificate le tabelle di inoltro per i bridge B1, B2, B3 e B4 dopo le trasmissioni seguenti:
- A invia a C.
 - C invia ad A.
 - D invia a C.

Identificate le porte usando come etichetta il nome dell'unico vicino raggiungibile direttamente dalla porta stessa; ad esempio, le porte di B1 devono essere etichettate con "A" e "B2".

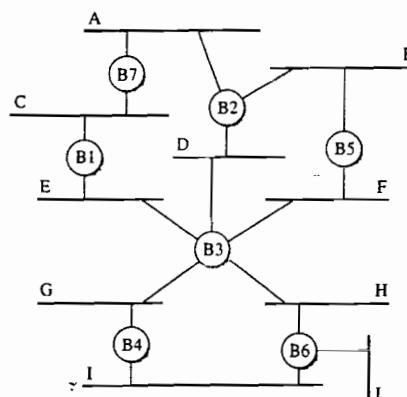


Figura 3.37 Rete per gli Esercizi 13 e 14.

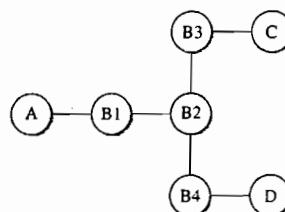


Figura 3.38 Rete per gli Esercizi 15 e 16.

- ✓ 16. Come nell'esercizio precedente, considerate la disposizione di bridge ad apprendimento mostrata in Figura 3.38. Ipotizzando che inizialmente siano tutte vuote, identificate le tabelle di inoltro per i bridge B1, B2, B3 e B4 dopo le trasmissioni seguenti:
- D invia a C.
 - C invia ad D.
 - A invia a C.
17. Considerate gli host X, Y, Z e W e i bridge ad apprendimento B1, B2 e B3, con tabelle di inoltro inizialmente vuote, come in Figura 3.39.
- Supponete che X trasmetta un pacchetto verso Z. Quali bridge apprendono la posizione di X? L'interfaccia di rete di Y vede tale pacchetto?
 - Supponete ora che Z trasmetta un pacchetto verso X. Quali bridge apprendono la posizione di Z? L'interfaccia di rete di Y vede tale pacchetto?
 - Supponete ora che Y trasmetta un pacchetto verso X. Quali bridge apprendono la posizione di Y? L'interfaccia di rete di Z vede tale pacchetto?
 - Supponete ora che Z trasmetta un pacchetto verso Y. Quali bridge apprendono la posizione di Z? L'interfaccia di rete di W vede tale pacchetto?
18. Tracciate lo spanning tree generato per la rete locale estesa mostrata in Figura 3.40 e descrivete come vengono risolte le situazioni di pareggio.
19. Supponete che due bridge ad apprendimento, B1 e B2, formino un anello come mostrato in Figura 3.41 e che non implementino l'algoritmo a spanning tree. Ciascun bridge gestisce una tabella di coppie (*indirizzo, interfaccia*).
- Cosa accade se M invia un pacchetto a L?
 - Supponete che dopo poco L risponda a M. Identificate la sequenza di eventi che porta alla situazione in cui un pacchetto proveniente da M e un pacchetto proveniente da L circolano nell'anello in direzioni opposte.
20. Supponete che, in Figura 3.41, M invii un pacchetto a se stesso (cosa che normalmente non avviene mai). Dite cosa accadrebbe, nell'ipotesi che:
- l'algoritmo di apprendimento dei bridge inserisca (o aggiorni) le coppie di informazioni (*indirizzo sorgente, interfaccia*) prima di cercare l'indirizzo di destinazione nella tabella;

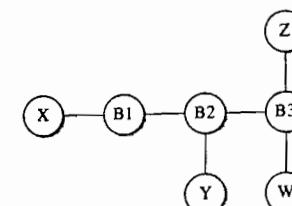


Figura 3.39 Schema per l'Esercizio 17.

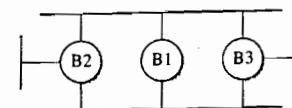


Figura 3.40 Rete locale estesa per l'Esercizio 18.

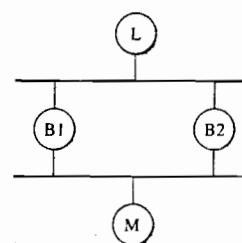


Figura 3.41 Anello per gli Esercizi 19 e 20.

- b) l'indirizzo della nuova sorgente venga inserito *dopo* la ricerca dell'indirizzo di destinazione.
21. Considerate la LAN estesa di Figura 3.12. Cosa accade nell'algoritmo a spanning tree se il bridge B1 non partecipa a?
- inoltra semplicemente tutti i messaggi relativi all'algoritmo a spanning tree?
 - ignora ed elimina tutti i messaggi relativi all'algoritmo a spanning tree?
22. Supponete di connettere ad anello alcuni ripetitori (*hub*), anziché bridge.
- Cosa succede quando qualcuno trasmette?
 - Perché sarebbe difficile o impossibile implementare il meccanismo a spanning tree per i ripetitori?
 - Proponete un meccanismo mediante il quale i ripetitori possano individuare gli anelli e chiudere alcune porte per interromperli. Non è richiesto che la vostra soluzione sia funzionante per il 100% del tempo.
23. Supponete che un bridge abbia due porte sulla stessa rete. Come se ne può accorgere e come può rimediare?
24. Quale percentuale dell'ampiezza di banda totale di una linea di collegamento ATM viene utilizzata per le intestazioni delle celle ATM? Quale percentuale dell'ampiezza di banda totale viene utilizzata da tutti i bit che non appartengono al carico utile in AAL3/4 e in AAL5, quando i dati dell'utente sono lunghi 512 byte?
25. Spiegate perché AAL3/4 non è in grado di rilevare la perdita di 16 celle consecutive appartenenti ad un singolo PDU.
26. Il datagramma IP per un messaggio di tipo ACK per TCP è lungo 40 byte: contiene 20 byte di intestazione TCP e 20 byte di intestazione IP. Nell'ipotesi che tale ACK attraversi una rete ATM che usa AAL5 per incapsulare i pacchetti IP, quanti pacchetti ATM serviranno per trasportarlo? E quanti ne sarebbero necessari usando AAL3/4?
27. Il CS-PDU di AAL5 contiene fino a 47 byte di dati fittizi (*padding*), mentre il CS-PDU di AAL3/4 ne contiene al massimo 3 byte. Spiegate perché l'ampiezza di banda effettiva di AAL5 è sempre almeno uguale o superiore a quella di AAL3/4, per un PDU di dimensione assegnata.
28. Quanto deve essere affidabile una connessione ATM per poter garantire un tasso di perdita inferiore a uno su un milione per PDU dello strato superiore aventi dimensione uguale a 20 celle? Usate AAL5.
29. Usando i pacchetti di 20 celle AAL5 dell'esercizio precedente, supponete che alla fine del PDU venga aggiunta una cella che sia il risultato dell'operazione di XOR fra tutte le celle precedenti del PDU. Tale cella consente di ricostruire una qualsiasi cella perduta.

Qual è ora il tasso di persità di celle che garantisce un tasso netto di perdita inferiore a uno su un milione per PDU a 20 celle?

- ★ 30. Ricordate che AAL3/4 ha una somma di controllo CRC-10 alla fine di ciascuna cella, mentre AAL5 ha una singola somma di controllo CRC-32 al termine del PDU. Se un PDU viene trasportato da 12 celle AAL3/4, viene dedicata alla rilevazione d'errore una quantità di bit quasi quadrupla rispetto a quanto si usa con AAL5.
- Supponete di sapere che gli errori arrivano a burst, con ogni burst sufficientemente piccolo da essere confinato all'interno di una singola cella. Determinate la probabilità che AAL3/4 fallisca nella rilevazione di un errore, nell'ipotesi che colpisca esattamente due celle. Fate lo stesso calcolo per tre celle. In queste situazioni, AAL3/4 è più o meno affidabile di AAL5? Usate l'approssimazione che un CRC a N bit fallisce nella rilevazione di un errore con una probabilità uguale a $1/2^N$ (che è il valore esatto soltanto quando tutti gli errori sono egualmente probabili).
 - Potete immaginare distribuzioni d'errore per cui è più probabile che AAL3/4 rilevi l'errore di quanto non faccia AAL5? Pensate che queste situazioni siano probabili?
31. I metodi di commutazione di celle usano essenzialmente sempre l'instradamento di circuiti virtuali piuttosto dell'instradamento di datagrammi. Fornite specifiche motivazioni per tale scelta.
32. Supponete che una workstation abbia un bus di I/O a 800 Mbps ed una memoria con ampiezza di banda di 2 Gbps. Usando il DMA per le comunicazioni con la memoria in entrambe le direzioni, un commutatore costruito con questa workstation quante interfacce verso linee T3 a 45 Mbps sarebbe in grado di gestire?
- ✓ 33. Supponete che una workstation abbia un bus di I/O a 1 Gbps ed una memoria con ampiezza di banda di 2 Gbps. Usando il DMA per le comunicazioni con la memoria in entrambe le direzioni, un commutatore costruito con questa workstation quante interfacce verso linee T3 a 45 Mbps sarebbe in grado di gestire?
34. Supponete che uno switch sia in grado di inoltrare pacchetti al ritmo di 100000 al secondo, indipendentemente dalla loro dimensione (entro limiti determinati). Usando una workstation con i parametri descritti nell'esercizio precedente, per quale dimensione dei pacchetti l'ampiezza di banda del bus diventa il fattore limitante?
35. Supponete che uno switch sia progettato per usare code FIFO per la memorizzazione, sia in ingresso sia in uscita. Quando i pacchetti arrivano ad una porta d'ingresso vengono inseriti alla fine della coda FIFO, successivamente lo switch tenta di inoltrare verso la fine della coda FIFO dell'uscita appropriata i pacchetti che si trovano all'inizio di ciascuna coda FIFO.
- Spiegate quali sono le circostanze in cui tale switch può perdere un pacchetto destinato ad una porta d'uscita la cui coda FIFO è vuota.
 - Come viene chiamato questo comportamento?
 - Nell'ipotesi che la memoria destinata alle code FIFO possa essere redistribuita liberamente, suggerite un metodo per rimescolare i buffer che elimini il problema precedente, e spiegate come vi riesce.
- ★ 36. Uno stadio di una rete banyana $n \times n$ consiste di $(n/2)$ elementi di commutazione 2×2 . Il primo stadio dirige i pacchetti verso la metà corretta della rete. Lo stadio successivo verso il quarto corretto, e così via, finché il pacchetto non viene instradato verso l'uscita corretta. Derivate un'espressione per il numero di elementi di commutazione 2×2 necessari per costruire una rete banyana $n \times n$. Verificate la risposta trovata per $n = 8$.

- ★ 37. Descrivete come funziona una rete Batcher (leggete la sezione "Ulteriori letture"). Spieghate come usare una rete Batcher insieme ad una rete banyana per costruire la matrice di commutazione di uno switch.
- 38. Uno switch Ethernet è semplicemente un bridge con la capacità di inoltrare un certo numero di pacchetti in parallelo, nell'ipotesi che le porte d'ingresso e d'uscita siano tutte distinte. Supponete che due di tali switch a N porte, con N molto grande, siano in grado, ciascuno, di inoltrare fino a tre pacchetti in parallelo. I due switch vengono connessi uno all'altro in serie collegando una coppia di porte, una di ciascuno switch; la linea di connessione fra loro è il collo di bottiglia, in quanto, ovviamente, può trasportare un solo pacchetto per volta.
 - a) Supponete di scegliere a caso due connessioni che attraversino questo switch composito. Qual è la probabilità che le due connessioni vengano inoltrate in parallelo? Suggerimento: questa è la probabilità che al massimo una delle due connessioni attraversi la linea di collegamento.
 - b) Cosa succederebbe se venissero scelte tre connessioni a caso?
- 39. Supponete che un hub (ripetitore) Ethernet a 10 Mbps venga sostituito da uno switch a 10 Mbps. In una situazione in cui tutto il traffico va da un singolo server a N client. Dato che tutto il traffico deve anche attraversare la linea di collegamento tra il server e lo switch, nominalmente la sostituzione non produce alcun miglioramento per l'ampiezza di banda.
 - a) Vi aspettate *qualche* miglioramento nell'ampiezza di banda? Se sì, perché?
 - b) Quale sarebbe la vostra risposta se il ripetitore originale fosse di tipo token ring anziché Ethernet?
 - c) Quali altri vantaggi e svantaggi rendono la soluzione con lo switch diversa da quella con il ripetitore?



4 Interconnessione di reti

Problema | Non esiste un'unica rete

Abbiamo visto come costruire una rete usando linee di collegamento punto-punto, mezzi fisici condivisi e commutatori. Il problema, ora, è che molte persone hanno costruito reti con queste diverse tecnologie e vogliono tutte essere in grado di comunicare l'una con l'altra, non soltanto con gli altri utenti di una singola rete. Questo capitolo tratta i problemi che nascono dall'interconnessione di reti diverse.

Esistono due importanti problemi da risolvere quando si connettono reti: l'*eterogeneità* e la *dimensione*. Detto semplicemente, il problema dell'eterogeneità consiste nel fatto che gli utenti di una rete di un certo tipo vogliono essere in grado di comunicare con gli utenti di reti di tipi diversi. Per complicare ulteriormente il problema, l'instaurazione di una connessione fra host appartenenti a due reti diverse può richiedere l'attraversamento di parecchie altre reti interposte, ciascuna delle quali può essere di un tipo ancora diverso. Queste diverse reti possono essere reti Ethernet, reti di tipo token ring, linee di collegamento punto-punto oppure reti comminate di vari tipi, ciascuna delle quali probabilmente avrà il proprio schema di indirizzamento, i propri protocolli di accesso al mezzo, i propri modelli di servizio e così via. L'obiettivo dell'eterogeneità è quello di fornire un servizio tra host utile e ben prevedibile, usando questo groviglio di reti diverse. Per comprendere il problema della dimensione della rete, è utile considerare la crescita di Internet, che per 20 anni ha pressappoco raddoppiato le proprie dimensioni ogni anno. Una crescita di questo tipo ci pone di fronte a parecchie sfide. Una di queste è l'*instradamento* (routing): come è possibile trovare un percorso efficiente attraverso una rete con milioni o, forse, miliardi di nodi? Un problema strettamente correlato a questo è l'*indirizzamento*, cioè il compito di fornire identificatori appropriati a tutti questi nodi.

Questo capitolo prende in esame una serie di approcci all'interconnessione di reti ed i problemi che devono essere risolti. Nel farlo, traccia l'evoluzione di Internet e della sua architettura TCP/IP, con l'obiettivo di capire in dettaglio i problemi dell'eterogeneità e della dimensione, insieme alle tecniche generali che vi si possono applicare.

La prima sezione presenta il protocollo IP (Internet Protocol) e mostra come lo si possa usare per costruire una rete interconnessa eterogenea e scalabile. La sezione contiene una discussione del modello di servizio di Internet, che è la chiave per la sua capacità di gestire l'eterogeneità, e descrive anche come lo schema di indirizzamento gerarchico di Internet abbia aiutato tale rete a scalare le proprie dimensioni fino a quelle di oggi, piuttosto grandi.

Un aspetto centrale nella costruzione di grandi reti eterogenee interconnesse è il problema di trovare percorsi efficienti e privi di cicli attraverso le reti componenti. La seconda sezione introduce i principi dell'instradamento ed esplora i problemi di scalabilità nei protocolli di routing, usando come esempi alcuni dei protocolli di routing di Internet.

La terza sezione parla di alcuni problemi (sempre più gravi) che hanno afflitto Internet negli ultimi anni e presenta varie tecniche che sono state utilizzate per risolverli. L'esperienza accumulata usando tali tecniche ha portato alla progettazione di una nuova versione di IP, la versione 6 (IPv6). In tutte queste discussioni vedremo l'importanza della gerarchia nella costruzione di reti scalabili.

Il capitolo si conclude prendendo in considerazione un paio di miglioramenti significativi per le possibilità di Internet. Il primo, la trasmissione multicast, è un miglioramento del modello di servizio di base. Mostriremo come la trasmissione multicast, che consente di consegnare pacchetti in modo efficiente ad un insieme di ricevitori, può essere incorporata in Internet e descriveremo alcuni protocolli di routing che sono stati sviluppati per fornire supporto al multicast. Il secondo miglioramento, MPLS (Multiprotocol Label Switching), modifica il meccanismo di inoltro delle reti IP: tale modifica ha consentito l'introduzione di alcuni cambiamenti nel modo in cui si effettua l'instradamento IP e nei servizi offerti dalle reti IP.

4.1 Semplice interconnessione di reti (IP)

Nel capitolo precedente abbiamo visto come sia possibile costruire reti locali ragionevolmente grandi usando bridge e commutatori per LAN, ma che tali approcci sono limitati nella loro capacità di scalare e di gestire l'eterogeneità. In questo capitolo esploriamo alcune vie per andare oltre le limitazioni delle reti con bridge, potendo così costruire grandi reti molto eterogenee con instradamento ragionevolmente efficiente: chiamiamo tali reti *internetwork* (reti interconnesse). Nelle sezioni seguenti procederemo verso reti interconnesse sempre più grandi, iniziando dalla funzionalità di base della versione attualmente in uso del protocollo IP (Internet Protocol), per poi prendere in esame, nella Sezione 4.3, diverse tecniche che sono state sviluppate per estendere la scalabilità di Internet. La discussione terminerà con una descrizione della versione 6 di IP (IPv6), nota anche come IP della "prossima generazione". Tuttavia, prima di addentrarci nei dettagli di un protocollo per reti interconnesse, dobbiamo esaminare con maggiore attenzione ciò che significa la parola "internetwork".

4.1.1 Cos'è una internetwork?

Con il termine "internetwork", o a volte soltanto "internet" con la *i* iniziale minuscola, ci riferiamo ad un insieme di reti di qualsiasi tipo interconnesse per fornire un servizio di consegna di pacchetti fra host. Ad esempio, un'azienda con molte sedi potrebbe costruire una internetwork privata interconnettendo le LAN presenti nelle diverse sedi mediante linee di collegamento punto-punto noleggiate da una compagnia telefonica. Quando parliamo, invece, della internetwork mondiale, utilizzata da tutti e a cui sono oggi connesse quasi tutte le

4.1 Semplice interconnessione di reti (IP)

reti, usiamo il termine "Internet" con la *I* iniziale maiuscola. Seguendo i principi fondamentali che guidano l'approccio utilizzato in questo libro, vogliamo principalmente farvi apprendere le funzioni delle internetwork "con la *i* minuscola", illustrando queste idee con esempi reali che provengono dalla Internet "con la *I* maiuscola".

Un altro aspetto terminologico che può creare confusione è la differenza fra reti, sottoreti, e reti interconnesse. Eviteremo del tutto l'uso di sottoreti fino alla Sezione 4.3. Per ora, usiamo il termine *rete* per indicare una rete a connessione diretta oppure una rete commutata del tipo che è stato discusso nei due capitoli precedenti. Tale rete usa un'unica tecnologia, come Ethernet, 802.5 o ATM. Una *internetwork* o rete interconnessa è un insieme di tali reti connesse tra loro; a volte, per evitare ogni ambiguità, chiameremo reti *fisiche* le reti sottostanti della cui interconnessione ci stiamo occupando. Una *internet* è una rete *logica* costituita da un insieme di reti fisiche; in questo contesto, un insieme di Ethernet connesse da bridge o switch sarebbe ancora visto come una singola rete.

La Figura 4.1 mostra un esempio di internetwork. Spesso ci si riferisce ad una internetwork come ad una "rete di reti", dato che è composta di molte reti più piccole. In questa figura vediamo reti Ethernet, un anello FDDI ed una linea di collegamento punto-punto: ciascuna di queste è una rete realizzata con una singola tecnologia. I nodi che interconnettono le reti vengono chiamati *router* (intradattori); a volte vengono anche chiamati *gateway*, ma noi useremo router perché il termine gateway ha anche altre connotazioni.

Il protocollo Internet è lo strumento chiave usato oggi per costruire reti interconnesse scalabili ed eterogenee, protocollo che fu inizialmente nato con il nome dei propri inventori, Kahn e Cerf. Si può immaginare che IP venga eseguito da tutti i nodi (sia host che router) di un insieme di reti e che definisca un'infrastruttura che consente a tali nodi e reti di funzionare dal punto di vista logico come una singola rete interconnessa. Ad esempio, la Figura 4.2 mostra come gli host H1 e H8 siano logicamente connessi alla internet di Figura 4.1, evidenziando

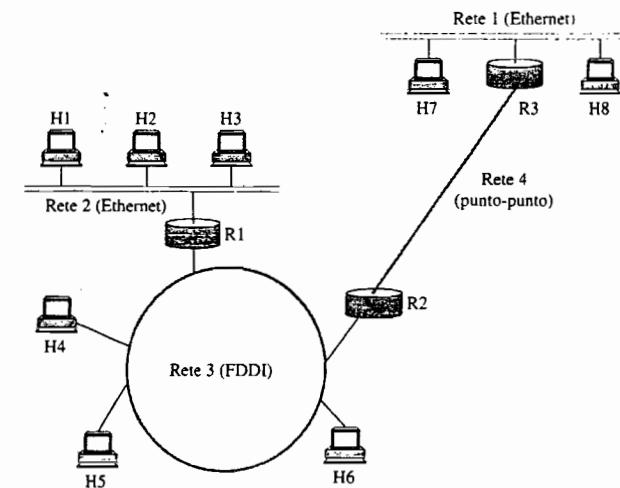


Figura 4.1 Una semplice internetwork. Hn = host; Rn = router.

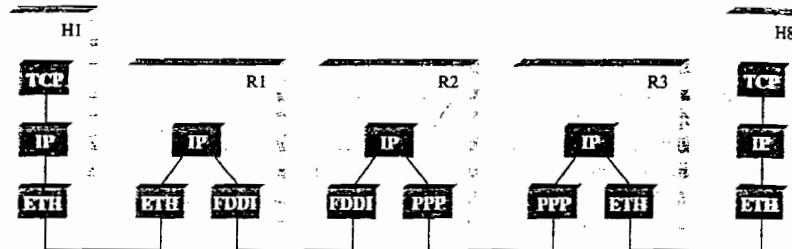


Figura 4.2 Una semplice internetwork con, evidenziati, gli strati di protocolli usati per connettere H1 a H8 nella Figura 4.1. ETH è il protocollo che viene eseguito in una Ethernet.

anche il grafo di protocolli in esecuzione in ciascun nodo. Notate che negli host, al di sopra di IP, vengono tipicamente eseguiti protocolli di livello superiore, come TCP e UDP.

La maggior parte di ciò che rimane di questo capitolo affronta diversi aspetti di IP. Mentre è certamente possibile costruire una internetwork che non usi IP (ad esempio, Novell ideò un protocollo per l'interconnessione di reti chiamato IPX, che era a sua volta basato sul protocollo XNS progettato da Xerox), IP è il caso più interessante da studiare, semplicemente a causa della dimensione di Internet. Detto in altro modo, soltanto la rete Internet, basata su IP, ha dovuto affrontare veramente il problema della dimensione, per cui costituisce il miglior caso di studio per un protocollo scalabile per l'interconnessione di reti.

4.1.2 Modello di servizio

Un buon punto da cui iniziare quando si progetta una internetwork è la definizione del suo *modello di servizio*, cioè del servizio fra host che si vuol fornire. Il problema principale nella definizione di un modello di servizio per una internetwork è che si può fornire un certo servizio fra host soltanto se tale servizio può, in qualche modo, essere fornito da tutte le sottostanti reti fisiche. Ad esempio, non sarebbe una buona cosa decidere che il modello di servizio della internetwork deve fornire la consegna garantita di ogni pacchetto in un tempo massimo di 1 ms se ci fossero reti sottostanti con tecnologie che possono ritardare arbitrariamente i pacchetti. La filosofia utilizzata nella definizione del modello di servizio di IP, quindi, è stata quella di renderlo così poco esigente da fare in modo che qualsiasi tecnologia di rete che possa comparire all'interno di una internetwork sia in grado di fornire il servizio necessario.

Il modello di servizio di IP può essere immaginato come composto da due parti: uno schema di indirizzamento, che fornisce il modo di identificare tutti gli host nella rete interconnessa, ed un modello datagram (cioè privo di connessione) per la consegna dei dati. Questo modello di servizio viene a volte chiamato *best effort* ("al meglio"), in quanto IP non fornisce alcuna garanzia sulla consegna dei datagrammi, nonostante compia tutti gli sforzi possibili. Per il momento rimandiamo la discussione sullo schema di indirizzamento e diamo prima uno sguardo al modello di consegna dei dati.

Consegna di datagrammi

Il datagramma IP è un elemento fondamentale del protocollo IP. Ricordate, dalla Sezione 3.1.1, che un datagramma è un tipo di pacchetto che viene trasmesso all'interno di una rete in

modalità priva di connessione. Ogni datagramma porta con sé sufficienti informazioni perché la rete possa inoltrare il pacchetto fino alla sua destinazione corretta, senza bisogno di alcun meccanismo preventivo che segnali alla rete cosa fare quando arriva il pacchetto: semplicemente, lo inviate, e la rete fa del suo meglio per portarlo alla destinazione desiderata. L'espressione "fa del suo meglio" (*best effort*) significa che se qualcosa non funziona bene ed il pacchetto viene perduto, corrotto, mal indirizzato o per qualsiasi motivo non riesce a raggiungere la propria destinazione, la rete non fa nulla: ha fatto del suo meglio, e ciò è tutto quanto le era richiesto. Non compie alcun tentativo per recuperare la situazione: si tratta di un servizio che a volte viene chiamato *inaffidabile*.

Un servizio privo di connessione e best-effort è praticamente il servizio più semplice che potete chiedere ad una internetwork; e ciò è un grande vantaggio. Ad esempio, se fornite un servizio best-effort mediante una rete che fornisce, invece, un servizio affidabile, va tutto bene: ottenete un servizio best-effort che riesce sempre a consegnare i pacchetti. Se, al contrario, avete un modello di servizio affidabile in una rete inaffidabile, dovreste dotare i router di molte funzioni aggiuntive per coprire l'inadeguatezza della rete sottostante. Semplificare al massimo il compito dei router era uno degli obiettivi del progetto originario di IP.

La capacità di IP di "essere eseguito ovunque" viene frequentemente citata come una delle sue caratteristiche più importanti. Può essere interessante notare come molte delle tecnologie che oggi possono essere utilizzate al di sotto di IP non esistevano nemmeno quando si inventò IP e, fino ad oggi, non è stata utilizzata nessuna tecnologia di rete che si sia dimostrata troppo bizzarra per IP: c'è chi ha dichiarato che IP è in grado di operare su una rete che trasporti i messaggi usando i piccioni viaggiatori.

La consegna best-effort non significa soltanto che i pacchetti possono essere smarriti. A volte vengono consegnati nell'ordine sbagliato e altre volte uno stesso pacchetto può essere consegnato più volte: i protocolli di livello superiore o le applicazioni che operano al di sopra di IP devono essere consapevoli di tutte queste possibilità di malfunzionamento.

Formato del pacchetto

Ovviamente, un ruolo fondamentale nel modello di servizio IP è assunto dal tipo di pacchetti che possono essere trasportati. Il datagramma IP, come la maggior parte dei pacchetti, consiste di un'intestazione seguita da un certo numero di byte di dati. Il formato dell'intestazione è mostrato in Figura 4.3. Notate che abbiamo adottato uno stile di rappresentazione dei pacchetti diverso da quello usato nei capitoli precedenti, perché i formati dei pacchetti dello strato di interconnessione di reti e degli strati superiori, sui quali focalizzeremo la nostra attenzione nei prossimi capitoli, sono quasi tutti progettati per essere allineati su confini di 32 bit, per semplificare il compito della loro elaborazione software. Di conseguenza, il modo più comune per rappresentarli (usato, ad esempio, nei documenti *Request for Comments* di Internet) consiste nel disegnarli come una successione di parole di 32 bit. La parola più in alto è quella che viene trasmessa per prima, ed il byte più a sinistra di ogni parola è quello che viene trasmesso per primo: In questa rappresentazione potete facilmente notare che i campi hanno una lunghezza che è un multiplo di 8 bit; nelle rare occasioni in cui i campi non sono un multiplo intero di 8 bit, potete determinarne la lunghezza osservando le posizioni dei bit indicate nella parte alta del pacchetto.

Osservando ciascun campo dell'intestazione IP, vediamo che il "semplice" modello di consegna best-effort e datagram ha pur sempre alcune caratteristiche complesse. Il campo Version specifica la versione di IP, che attualmente è la versione 4 e viene a volte chiamata

IPv4¹. Osservate che inserire questo campo proprio all'inizio del datagram rende facile ridefinire in successive versioni del protocollo qualsiasi altro campo del formato di pacchetto: il software che elabora l'intestazione inizia esaminando la versione, per poi usare diverse procedure per elaborare la parte rimanente del pacchetto secondo il formato appropriato. Il campo successivo, HLen, indica la lunghezza dell'intestazione, misurata in parole di 32 bit. Quando non ci sono opzioni, cosa che accade assai spesso, l'intestazione è lunga 5 parole (20 byte). Il campo TOS (type of service, tipo di servizio) di 8 bit ha assunto vari ruoli negli anni, ma la sua funzione fondamentale è quella di consentire il trattamento differenziato dei pacchetti in base alle necessità delle applicazioni. Ad esempio, il valore del campo TOS può determinare il fatto che un pacchetto debba essere inserito, oppure no, in una coda speciale che subisce un ritardo inferiore. Nella Sezione 6.5.3 parleremo con maggiore dettaglio dell'uso di questo campo (e vedremo un suo nome diverso).

I 16 bit successivi dell'intestazione contengono la lunghezza (Length) del datagramma, compresa l'intestazione. Diversamente dal campo HLen, il campo Length conta i byte invece delle parole, per cui la dimensione massima di un datagramma IP è di 65535 byte. Può darsi, però, che la rete fisica sulla quale opera IP non sia in grado di trasportare pacchetti così lunghi: per tale ragione, IP fornisce supporto per un procedimento di frammentazione e ricostruzione. La seconda parola dell'intestazione contiene informazioni relative alla frammentazione, i cui dettagli verranno presentati in seguito nel paragrafo "Frammentazione e ricostruzione".

Spostandoci sulla terza parola dell'intestazione, il byte successivo è il campo TTL (time to live, tempo di vita), il cui nome riflette il suo significato storico piuttosto che il modo in cui viene normalmente utilizzato. L'obiettivo di questo campo è quello di catturare quei pacchetti che continuano a viaggiare in percorsi circolari ed eliminarli, anziché permettere che con-

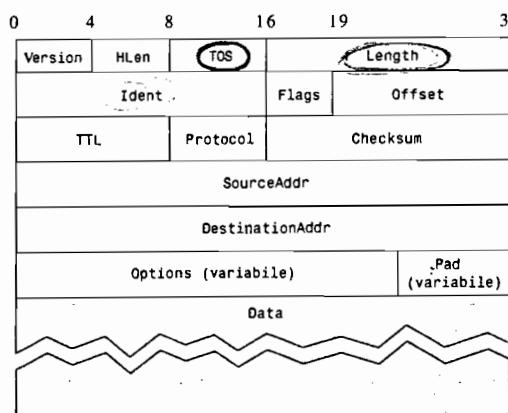


Figura 4.3 Intestazione del pacchetto IPv4.

¹ La successiva versione principale di IP, che viene discussa più avanti nel capitolo, è la versione numero 6 ed è nota come IPv6. La versione numero 5 è stata usata per un protocollo sperimentale chiamato ST-II che non ha avuto grande diffusione.

sumino risorse indefinitamente. In origine il campo TTL veniva impostato ad un valore che rappresentava proprio il numero di secondi durante i quali il pacchetto doveva rimanere in vita e i router lungo il percorso decrementavano tale campo fino a farlo diventare zero. Dato che, però, era raro che un pacchetto rimanesse all'interno di un router per più di un secondo e dato che i router non avevano accesso ad un segnale di temporizzazione comune per tutti, la maggior parte dei router prima di inoltrare il pacchetto decrementava semplicemente di un'unità il valore di TTL, che, di conseguenza, divenne in pratica un contatore di passaggi (*hop*) attraverso i router piuttosto che un temporizzatore, rappresentando comunque un buon modo per catturare i pacchetti bloccati in un percorso circolare. Rimane il dettaglio di quale valore deve impostare inizialmente in questo campo l'host sorgente: se il valore è troppo elevato i pacchetti possono circolare inutilmente per lungo tempo prima di essere eliminati, mentre se il valore è troppo basso può darsi che non riescano a giungere a destinazione. L'attuale valore di default è 64.

Il campo Protocol è semplicemente una chiave per il demultiplexing che identifica il protocollo di livello superiore a cui va consegnato il pacchetto IP. Esistono valori definiti per TCP (6), UDP (17) e per molti altri protocolli che si possono trovare al di sopra di IP nel grafo di protocolli.

Il valore del campo Checksum viene calcolato considerando l'intera intestazione IP come una sequenza di parole di 16 bit, sommandole usando l'aritmetica in complemento a uno e prendendo il complemento a uno del risultato: si tratta dell'algoritmo di somma di controllo di IP descritto nella Sezione 2.4. Di conseguenza, se uno qualsiasi dei bit dell'intestazione viene corrotto durante il trasferimento, la somma di controllo non conterrà il valore corretto al momento della ricezione del pacchetto. Dato che un'intestazione corrotta può avere un errore anche nell'indirizzo di destinazione (e, di conseguenza, essere stata consegnata al destinatario sbagliato), la cosa migliore da fare è ignorare qualsiasi pacchetto la cui somma di controllo sia errata. Questo tipo di somma di controllo non ha le stesse robuste proprietà di rilevazione d'errore viste per i CRC, ma è molto più semplice da calcolare in software.

Gli ultimi due campi obbligatori dell'intestazione sono l'indirizzo del mittente (SourceAddr) e del destinatario (DestinationAddr) del pacchetto. Quest'ultimo è l'elemento chiave per la consegna in modalità datagram: ogni pacchetto contiene un indirizzo completo della sua destinazione prevista, in modo che ogni router possa prendere le proprie decisioni sull'inoltro. L'indirizzo di sorgente è necessario per consentire al ricevente di decidere se vuole accettare il pacchetto e per metterlo in condizione di rispondere. Gli indirizzi IP vengono discussi nella Sezione 4.1.3; per il momento, la cosa importante da sapere è che IP definisce il proprio spazio di indirizzamento globale, indipendente dalla rete fisica sulla quale si trova ad operare. Come vedremo, questa è una delle caratteristiche principali che consentono l'eterogeneità.

Infine, al termine dell'intestazione si trovano un certo numero di opzioni, la cui presenza o assenza viene detta dall'esame del campo che contiene la lunghezza dell'intestazione (HLen). Nonostante le opzioni vengano usate di rado, un'implementazione completa di IP le deve gestire tutte.

Frammentazione e ricostruzione

Uno dei problemi da affrontare per fornire un modello di servizio tra host che sia uniforme su un insieme di reti eterogenee consiste nel fatto che ciascuna tecnologia di rete tende ad avere una propria personale opinione relativamente a quella che deve essere la dimensione di un pacchetto. Ad esempio, una rete Ethernet può accettare pacchetti lunghi fino ad un massimo

di 1500 byte, mentre una rete FDDI fino ad un massimo di 4500 byte. Questa situazione lascia al modello di servizio di IP soltanto due possibili scelte: garantire che tutti i datagrammi IP siano abbastanza piccoli da poter essere contenuti dai pacchetti di qualsiasi altra tecnologia di rete, oppure fornire una procedura per mezzo della quale i pacchetti, qualora siano troppo grandi per essere trasportati da una certa tecnologia di rete, possano essere frammentati e poi ricostruiti. La seconda è risultata essere la scelta vincente, soprattutto in considerazione del fatto che si affacciano sulla scena tecnologie di rete sempre nuove e IP deve essere in grado di operare su tutte: sarebbe quindi difficile scegliere un limite sufficientemente piccolo per la dimensione dei datagrammi. Ciò significa anche che un host invierà pacchetti piccoli senza averne bisogno, una cosa che sprecherebbe banda e consumerebbe risorse di elaborazione, essendo necessari più byte di intestazione per ciascun byte di dati trasmessi. Ad esempio, due host connessi a reti FDDI che siano interconnesse mediante una linea punto-punto non sono costretti a spedire pacchetti sufficientemente piccoli da poter essere trasportati da una rete Ethernet.

L'idea fondamentale, in questo caso, è che ogni tipo di rete ha una propria unità massima trasmissibile (MTU, maximum transmission unit), che è il più grande datagramma IP che può trasportare all'interno di un proprio frame. Notate che questo valore è inferiore alla dimensione massima dei pacchetti sulla rete, perché il datagramma IP deve trovare posto come carico utile del frame dello strato di linea di collegamento. Notate anche che nelle reti ATM il "frame" è il CS-PDU e non la cella ATM: il fatto che i CS-PDU vengano segmentati in celle non è visibile dal protocollo IP.

Quando un host invia un datagramma IP, quindi, può scegliere la dimensione che preferisce, anche se il valore di MTU della rete a cui l'host è direttamente connesso è una scelta ragionevole: in questo caso la frammentazione sarà necessaria soltanto se il percorso verso la destinazione attraversa una rete con un valore di MTU inferiore. Se, invece, il protocollo di trasporto che sta sopra a IP fornisce al protocollo IP un pacchetto di dimensione maggiore del valore locale di MTU, allora l'host sorgente lo deve frammentare.

Tipicamente la frammentazione avviene in un router, nel momento in cui riceve un datagramma da inoltrare verso una rete avente un valore di MTU inferiore alla dimensione del datagramma ricevuto. Per fare in modo che questi frammenti possano essere ricostruiti dall'host ricevente, devono avere tutti lo stesso identificatore nel campo Ident, identificatore che viene scelto dall'host sorgente e che deve essere univoco fra tutti i datagrammi che possono arrivare alla destinazione, provenendo da quella sorgente, in un intervallo di tempo ragionevolmente lungo. Dato che tutti i frammenti del datagramma originale contengono questo identificatore, l'host che lo deve ricostruire sarà in grado di riconoscere quei frammenti che vanno assemblati insieme; se ne dovesse mancare qualcuno, l'host abbandona il processo di ricostruzione ed elimina i frammenti che sono arrivati: il protocollo IP non tenta di recuperare i frammenti mancati.

Per vedere cosa significa tutto ciò, considerate cosa accade quando l'host H1 invia un datagramma all'host H8 nella internet di esempio mostrata in Figura 4.1. Facendo l'ipotesi che il valore di MTU sia 1500 byte per le due Ethernet, 4500 byte per la rete FDDI e 532 byte per la rete punto-punto, si ha che un datagramma di 1420 byte (20 byte di intestazione IP più 1400 byte di dati) inviato da H1 attraversa la prima Ethernet e la rete FDDI senza subire frammentazione, ma viene frammentato in tre datagrammi dal router R2. Questi tre frammenti vengono poi inoltrati dal router R3, attraverso la seconda Ethernet, verso l'host di destinazione. Questa situazione viene illustrata nella Figura 4.4, che serve anche a sottolineare due punti importanti:

4.1 Semplice interconnessione di reti (IP)

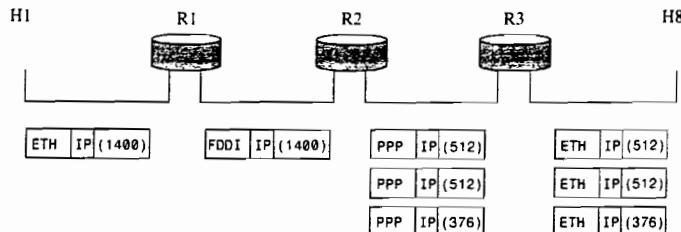


Figura 4.4 Datagrammi IP che attraversano la sequenza di reti fisiche rappresentata in Figura 4.1.

1. ciascun frammento è, a sua volta, un datagramma IP completo, che viene trasmesso attraverso una serie di reti fisiche in modo indipendente dagli altri frammenti;
2. ogni datagramma IP viene nuovamente incapsulato da ogni rete fisica sulla quale viaggia.

Il procedimento di frammentazione può essere compreso nei suoi dettagli osservando i campi dell'intestazione di ciascun datagramma, come fatto in Figura 4.5. Il pacchetto non frammentato, mostrato nella parte alta della figura, è composto da 1400 byte di dati e da un'intestazione IP di 20 byte. Quanto il pacchetto arriva al router R2, che ha un valore di MTU uguale a 532 byte, deve essere frammentato: un valore di MTU di 532 byte consente di inserire 512 byte di dati dopo l'intestazione IP di 20 byte, per cui il primo frammento contiene 512 byte di dati. Il router imposta al valore 1 il bit M (more) nel campo Flags (confrontate la Figura 4.3), stando a significare che seguiranno ulteriori frammenti, ed imposta al valore 0 il campo Offset, poiché questo frammento contiene la prima parte del datagramma originario. I dati che vengono inseriti nel secondo frammento iniziano con il 513-esimo byte dei dati originari, per cui il campo Offset di questa intestazione viene impostato al valore 64; cioè 512/8. Perché si divide per 8? Perché i progettisti di IP decisero che la frammentazione deve sempre avvenire su confini multipli di 8 byte, per cui il campo Offset conta spezzoni di 8 byte e non singoli byte (vi lasciamo trovare per esercizio una motivazione per questa decisione di progetto). Il terzo frammento contiene gli ultimi 376 byte di dati e ora il campo Offset vale $2 \times 512/8 = 128$. Dato che si tratta dell'ultimo frammento, il bit M vale 0.

Osservate che il processo di frammentazione viene fatto in modo che potrebbe anche essere ripetuto, qualora un frammento arrivasse in un'altra rete con un valore di MTU ancora più piccolo. La frammentazione produce datagrammi IP più piccoli ma perfettamente validi, che possono essere facilmente sottoposti ad un processo di ricostruzione per riottenere il datagramma originario una volta ricevuti indipendentemente dall'ordine d'arrivo. La ricostruzione viene effettuata dall'host destinatario e non dai router.

Implementazione

Terminiamo questa discussione della frammentazione e ricostruzione IP delineando il codice necessario ad eseguire la ricostruzione. Uno dei motivi per cui presentiamo questo particolare segmento di codice è il fatto che costituisce un valido esempio di una gran parte del software di rete: fa poco più di una gestione di dati tediosa e ingloriosa.

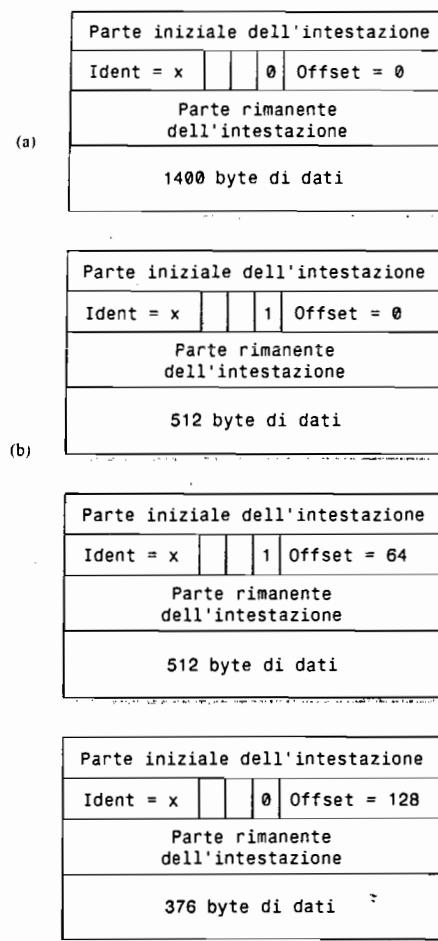


Figura 4.5 Campi dell'intestazione usati nella frammentazione IP.
(a) Pacchetto non frammentato; (b) pacchetti frammentati.

Per prima cosa definiamo la struttura dati fondamentale (`FragList`) che viene usata per contenere i singoli frammenti che arrivano a destinazione. I frammenti in arrivo vengono memorizzati in questa struttura finché non sono arrivati tutti i frammenti del datagramma originario, che a quel punto possono essere assemblati nel datagramma completo e trasferiti ad un protocollo di livello superiore. Notate che ogni elemento inserito in `FragList` contiene o un frammento o una lacuna.

4.1 Semplice interconnessione di reti (IP)

```
#define FRAGOFFMASK      0x1fff
#define FRAGOFFSET(fragflag) ((fragflag) & FRAGOFFMASK)
#define INFINITE_OFFSET    0xffff

/* struttura che contiene i campi che identificano in modo univoco i frammenti di uno stesso datagramma IP */

typedef struct fid {
    IpHost source;
    IpHost dest;
    u_char prot;
    u_char pad;
    u_short ident;
} Fragid;

typedef struct hole {
    u_int first;
    u_int last;
} Hole;

#define HOLE 1
#define FRAG 2

/* struttura che contiene un frammento o una lacuna */

typedef struct fragif {
    u_char type;
    union {
        Hole hole;
        Msg frag;
    } u;
    struct fragif *next, *prev;
} FragInfo;

/* struttura che contiene tutti i frammenti e le lacune di un singolo datagramma IP da ricostruire */

typedef struct FragList {
    u_short nholes;
    FragInfo head; /* nodo d'intestazione fittizio */
    Binding binding;
    bool gcMark; /* segnale di garbage collection */
} FragList;
```

La funzione di ricostruzione, `ipReassemble`, riceve come argomenti un datagramma in arrivo (`dg`) e la sua intestazione IP (`hdr`), nonché un terzo argomento, `fragMap`, che è una struttura di tipo `Map` (dotata delle operazioni `mapBind`, `mapRemove` e `mapResolve`) usata per creare

in modo efficiente una corrispondenza tra il datagramma in arrivo e la FragList appropriata (ricordate che il gruppo di frammenti da assemblare insieme sono identificati univocamente da alcuni campi dell'intestazione IP, come definito dalla struttura FragId descritta sopra).

Il lavoro effettivamente eseguito da ipReassemble è molto semplice: come già detto, si tratta principalmente di operazioni banali. Dapprima la funzione estrae dall'intestazione IP i campi che identificano in modo univoco il datagramma da ricostruire, poi crea una chiave con tali campi ed esegue una ricerca con tale chiave nella fragMap per trovare la FragList appropriata: se si tratta del primo frammento del datagramma, viene creata e inizializzata una nuova FragList. Successivamente, la funzione inserisce in tale FragList il nuovo frammento, cosa che richiede di confrontare la somma dell'offset e la lunghezza di tale frammento con l'offset del frammento successivo presente nell'elenco. Parte di questo lavoro viene svolto dalla funzione hole_create, il cui codice è presentato nel seguito. Infine, ipReassemble controlla se tutte le lacune sono state colmate: se sono presenti tutti i frammenti, invoca la funzione msgReassemble per assemblare effettivamente i frammenti in un datagramma completo, per poi invocare deliver per trasferire tale datagramma verso l'alto nel grafo di protocolli, ad un protocollo di livello superiore identificato come HLP (high-level protocol).

```
ipReassemble(Msg *dg, IpHdr *hdr, Map fragMap)
{
    FragId    fragid;
    FragList *list;
    FragInfo *fi, *prev;
    Hole      *hole;
    u_short   offset, len;

    /* estrae le informazioni di frammentazione dall'intestazione
       (offset e lunghezza del frammento) */
    offset = FRAGOFFSET(hdr->flag)*8;
    len = hdr->dlen - GET_HLEN(hdr) * 4;

    /* crea l'identificatore univoco per questo frammento */
    bzero((char *)&fragid, sizeof(FragId));
    fragid.source = hdr->source;
    fragid.dest = hdr->dest;
    fragid.prot = hdr->prot;
    fragid.ident = hdr->ident;

    /* cerca l'elenco di frammenti per questo frammento;
       ne crea uno se l'elenco non esiste */
    if (mapResolve(fragMap, &fragid, (void **)&list) == FALSE)
    {
        /* primo frammento del pacchetto, serve una nuova FragList */
        list = NEW(FragList);

        /* inserisce l'elenco nella struttura di tipo Map */
        list->binding = mapBind(fragMap, &fragid, list);
    }
}
```

```

    }

    /* inizializza l'elenco con una singola lacuna che si estende
       per l'intero datagramma */
    list->nholes = 1;
    list->head.next = fi = NEW(FragInfo);
    fi->next = 0;
    fi->type = HOLE;
    fi->u.hole.first = 0;
    fi->u.hole.last = INFINITE_OFFSET;
}

/* contrassegna l'attuale FragList come non disponibile per
   la garbage collection */
list->gdMark = FALSE;

/* scandisce la FragList alla ricerca della giusta lacuna per
   questo frammento */
prev = &list->head;
for (fi = prev->next; fi != 0; prev = fi, fi = fi->next)
{
    if (fi->type == FRAG)
    {
        continue;
    }
    hole = &fi->u.hole;
    if ((offset < hole->last) && ((offset + len) > hole->first))
    {
        /* controlla se il frammento si sovrappone a frammenti
           ricevuti in precedenza */
        if (offset < hole->first)
        {
            /* tronca il messaggio da sinistra */
            msgStripHdr(dg, hole->first - offset);
            offset = hole->first;
        }
        if ((offset + len) > hole->last)
        {
            /* tronca il messaggio da destra */
            msgTruncate(dg, hole->last - offset);
            len = hole->last - offset;
        }
    }
    /* controlla ora si devono creare nuove lacune */
    if (((offset + len) < hole->last) &&
        ((hdr->frag & MOREFRAGMENTS)))
    {
        /* crea una nuova lacuna al di sopra */
        hole_create(prev, fi, (offset+len), hole->last);
        list->nholes++;
    }
}

```

```

    }
    if (offset > hole->first)
    {
        /* crea una nuova lacuna al di sotto */
        hole_create(fi, fi->next, hole->first, (offset));
        list->nholes++;
    }

    /* modifica questa struttura di tipo FragInfo
       facendola diventare FRAG */
    list->nholes--;
    fi->type = FRAG;
    msgSaveCopy(&fi->u.frag, dg);
    break;
} /* se è stata trovata una lacuna */
} /* ciclo for */

/* controlla se abbiamo finito e, in tal caso, trasferisci
   il datagramma verso l'alto */
if (list->nholes == 0)
{
    Msg fullMsg;

    /* ora abbiamo un datagramma completo */
    for (fi = list->head.next; fi != 0; fi = fi->next)
    {
        msgReassemble(&fullMsg, &fi->u.frag, &fullMsg);
    }
    /* elimina la FragList ed il suo valore in Map */
    mapRemove(fragMap, list->binding);
    ipFreeFragList(list);
    deliver(HLP, &fullMsg);
    msgDestroy(&fullMsg);
}
return SUCCESS;
}

```

La funzione `hole_create` crea nella lista di frammenti una nuova lacuna che inizia all'offset `first` e si estende fino all'offset `last`, usando la funzione di utilità `NEW`, che crea un esemplare della struttura indicata.

```

static int
hole_create(FragInfo *prev, FragInfo *next, u_int first, u_int last)
{
    FragInfo *fi;
    /* crea una nuova lacuna da first a last */

```

```

    fi = NEW(FragInfo);
    fi->type = HOLE;
    fi->u.hole.first = first;
    fi->u.hole.last = last;
    fi->next = next;
    prev->next = fi;
}

```

Infine, notate che queste funzioni non rappresentano l'intero processo di ricostruzione. Ciò che non viene mostrato è un processo in background che verifica periodicamente se c'è stata attività recente su un certo datagramma (controllando il campo `gcMark`) e, in caso contrario, elimina la `FragList` corrispondente. Il protocollo IP non tenta alcun recupero nel caso in cui uno o più frammenti non arrivino: semplicemente, rinuncia e libera la memoria che era impegnata per la ricostruzione.

Una cosa che va notata in questo codice è che la ricostruzione IP non è un processo semplice. Ad esempio, se viene perduto un solo frammento, il ricevitore tenterà di ricostruire il datagramma, per poi rinunciarvi e dover liberare le risorse che erano state usate per eseguire la ricostruzione fallita. Per questa ragione, fra le altre, la frammentazione IP viene generalmente considerata una cosa che è meglio evitare. Si incoraggiano fortemente gli host a "cercare il valore di MTU del percorso" (*path MTU discovery*), un procedimento mediante il quale si evita la frammentazione inviando pacchetti abbastanza piccoli da poter attraversare la linea con il valore di MTU minore lungo l'intero percorso tra la sorgente e la destinazione.

4.1.3 Indirizzi globali

Nella precedente discussione sul modello di servizio di IP abbiamo detto che una delle cose che fornisce è uno schema di indirizzamento. Dopo tutto, se volete essere in grado di inviare dati a qualsiasi host in qualsiasi rete, c'è bisogno di un modo per identificare tutti gli host. Di conseguenza, abbiamo bisogno di uno schema di indirizzamento globale, in cui non esistano due host con lo stesso indirizzo. L'unicità globale è la prima proprietà che dovrebbe essere garantita da uno schema di indirizzamento.

Gli indirizzi Ethernet sono globalmente univoci, ma questo solo fatto non è sufficiente per uno schema di indirizzamento di una grande internetwork. Gli indirizzi Ethernet sono anche *non gerarchici* (flat), per cui non possiedono una struttura e forniscono ben poco aiuto ai protocolli di instradamento². Al contrario, gli indirizzi IP sono *gerarchici*, sono cioè composti da diverse parti che corrispondono ad una specie di gerarchia all'interno della internetwork. Nello specifico, gli indirizzi IP consistono di due parti, un indirizzo di rete e un indirizzo di host: ciò rappresenta uno schema abbastanza logico per una internetwork, che è composta di molte reti interconnesse. La parte che costituisce l'indirizzo di rete di un indirizzo IP identifica la rete a cui è connesso l'host, e tutti gli host connessi alla stessa rete hanno lo stesso indirizzo di rete come parte del proprio indirizzo IP. La parte che costituisce l'indirizzo di

² In effetti, come abbiamo già fatto notare, gli indirizzi Ethernet hanno una struttura che risponde a criteri di *assegnazione* (i primi 24 bit identificano il produttore), ma ciò non fornisce alcuna informazione utile ai protocolli di instradamento perché tale struttura non ha nulla a che vedere con la topologia di rete.

host identifica univocamente ciascun host all'interno della propria rete. Di conseguenza, nella internetwork di esempio di Figura 4.1, gli indirizzi degli host appartenenti alla rete 1, ad esempio, hanno tutti lo stesso indirizzo di rete e diversi indirizzi di host.

Noteate che i router di Figura 4.1 sono connessi a due reti: devono quindi avere un indirizzo, uno per ciascuna interfaccia. Ad esempio, il router R1, che si trova tra le reti 2 e 3, ha un indirizzo IP sull'interfaccia con la rete 2 che ha lo stesso indirizzo di rete degli host della rete 2, e ha un indirizzo IP sull'interfaccia con la rete 3 che ha lo stesso indirizzo di rete degli host della rete 3. Per cui, ricordando che un router si può realizzare mediante un host avente due interfacce di rete, è più preciso pensare agli indirizzi IP come appartenenti alle interfacce piuttosto che agli host.

Ora, come sono fatti questi indirizzi gerarchici? Diversamente da altre forme di indirizzi gerarchici, le dimensioni delle due parti non sono uguali per tutti gli indirizzi: gli indirizzi IP sono suddivisi in tre diverse classi o categorie, come mostrato in Figura 4.6, ciascuna delle quali definisce indirizzi di rete e indirizzi di host di dimensioni diverse (esistono anche indirizzi di classe D che caratterizzano un gruppo multicast, di cui parleremo nella Sezione 4.4, e indirizzi di classe E che attualmente sono inutilizzati). In tutti i casi, l'indirizzo è lungo 32 bit.

La classe di un indirizzo IP viene identificata dai suoi bit più significativi. Se il primo bit vale 0, si tratta di un indirizzo di classe A. Se il primo bit vale 1 ed il secondo vale 0, si tratta di un indirizzo di classe B. Se i primi due bit valgono 1 ed il terzo bit vale 0, si tratta di un indirizzo di classe C. Conseguentemente, esistono circa 4 miliardi di possibili indirizzi IP: metà sono di classe A, un quarto sono di classe B e un ottavo sono di classe C. Ciascuna classe assegna un certo numero di bit per l'indirizzo di rete e la restante parte dell'indirizzo è destinata all'indirizzo di host. Le reti di classe A hanno 7 bit per l'indirizzo di rete e 24 bit per l'indirizzo di host, per cui esistono soltanto 126 reti di classe A (i valori 0 e 127 sono riservati), ma ciascuna di esse può avere fino a $2^{24} - 2$ host (circa 16 milioni), dato che anche in questo caso esistono due valori riservati. Gli indirizzi di classe B assegnano 14 bit all'indirizzo di rete e 16 all'indirizzo di host, per cui ciascuna rete di classe B può avere 65534 host. Infine, una rete di classe C ha soltanto 8 bit a disposizione per l'indirizzo di host e 21 per l'indirizzo di rete, quindi può avere soltanto 256 identificatori univoci per gli host e, di conseguenza, può avere soltanto 254 host connessi (un identificatore di host, il 255, è riservato alle trasmissioni broadcast, mentre il valore 0 non viene ritenuto valido come indirizzo di host). Tuttavia, questo schema di indirizzamento consente l'esistenza di 2^{21} reti di classe C.

A prima vista questo schema di indirizzamento ha un'elevata flessibilità, consentendo la coesistenza abbastanza efficiente di reti con dimensioni fortemente diversificate. L'idea ori-

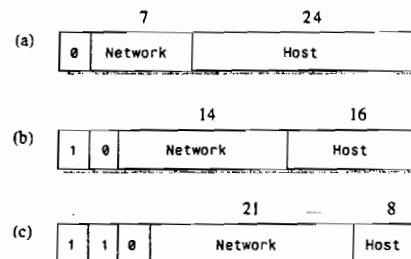


Figura 4.6 Indirizzi IP: (a) classe A; (b) classe B; (c) classe C.

ginale era quella che Internet sarebbe stata costituita da un piccolo numero di reti di ampie dimensioni (che sarebbero state le reti di classe A), un numero limitato di reti con dimensioni aziendali (che sarebbero state reti di classe B) ed un grande numero di reti locali (che sarebbero state reti di classe C). Tuttavia, come vedremo nella Sezione 4.3, si è resa necessaria una maggiore flessibilità e oggi vengono utilizzati modi innovativi per raggiungerla. Dato che una di queste tecniche, in realtà, elimina la distinzione fra classi di indirizzi, lo schema di indirizzamento appena descritto è anche noto come indirizzamento "con classi" (*classful*), per distinguere dal nuovo approccio "privo di classi" (*classless*).

Prima di vedere come vengono utilizzati gli indirizzi IP, è utile fare alcune osservazioni di carattere pratico, come, ad esempio, il modo con cui vengono scritti. Per convenzione, gli indirizzi IP vengono scritti mediante quattro numeri interi *decimali* separati da punti. Ciascun numero intero rappresenta il valore decimale contenuto in un byte dell'indirizzo, iniziando da quello più significativo. Ad esempio, l'indirizzo del calcolatore sul quale stiamo scrivendo questa frase va scritto come 171.69.210.245.

È importante non fare confusione fra gli indirizzi IP e i nomi di dominio Internet, che sono anch'essi gerarchici. I nomi di dominio sono stringhe ASCII separate da punti, come cs.princeton.edu, di cui parleremo nella Sezione 9.1. La cosa importante è che sono gli indirizzi IP ad essere presenti nelle intestazioni dei pacchetti IP e ad essere utilizzati dai router IP per prendere le decisioni di inoltro.

4.1.4 Inoltro di datagrammi in IP

Siamo ora pronti per dare un'occhiata al meccanismo di base mediante il quale i router IP inoltrano i datagrammi in una internetwork. Ricordate dal Capitolo 3 che l'*inoltro* (forwarding) è il processo che consiste nel ricevere un pacchetto da una porta d'ingresso e trasferirlo verso la porta d'uscita corretta, mentre l'*instradamento* (routing) è il processo di costruzione delle tabelle che consentono di determinare l'uscita corretta per un certo pacchetto. La discussione, qui, è incentrata sull'inoltro; esamineremo l'istradamento nella Sezione 4.2.

I punti principali da tenere presenti nel discutere di inoltro di datagrammi IP sono i seguenti:

- Ogni datagramma IP contiene l'indirizzo IP dell'host di destinazione.
- La parte chiamata "indirizzo di rete" di un indirizzo IP identifica univocamente una singola rete fisica che fa parte di Internet.
- Tutti gli host e tutti i router che condividono lo stesso indirizzo di rete nel proprio indirizzo sono connessi alla stessa rete fisica e possono quindi comunicare fra loro scambiandosi frame su tale rete.
- Ogni rete fisica che faccia parte di Internet ha almeno un router che, per definizione, è anche connesso ad almeno un'altra rete fisica; tale router è in grado di scambiare pacchetti con host e router di entrambe le reti.

L'inoltro di datagrammi IP può, quindi, avvenire nel modo seguente. Un datagramma viene inviato da un host sorgente ad un host destinazione, eventualmente attraversando dei router lungo il suo percorso. Ogni nodo, sia un router sia un host, tenta per prima cosa di stabilire se è connesso alla stessa rete fisica della destinazione: per farlo, confronta l'indirizzo di rete contenuto nell'indirizzo di destinazione con l'indirizzo di rete di ciascuna delle proprie interfacce di rete (gli host hanno normalmente una sola interfaccia di rete, mentre i router ne hanno

soltamente due o più, perché sono tipicamente connessi a due o più reti). Se viene verificata la corrispondenza, ciò significa che la destinazione si trova sulla stessa rete fisica cui è connessa l'interfaccia identificata e il pacchetto può essere consegnato direttamente da tale rete. La Sezione 4.1.5 illustra alcuni dettagli di questo processo.

Se il nodo non è connesso alla stessa rete fisica cui è connesso il nodo di destinazione, è necessario inviare il datagramma ad un router. In generale, ciascun nodo può scegliere tra vari router, per cui deve identificare il migliore o, quantomeno, uno che abbia una ragionevole probabilità di far giungere il datagramma più vicino alla sua destinazione. Il router che viene scelto prende il nome di router *del salto successivo* (next hop router), il quale troverà il successivo hop consultando la propria tabella di inoltro, che è, concettualmente, soltanto un elenco di coppie di tipo *<NetworkNum, NextHop>* (anche se, come vedremo in seguito, le tabelle di inoltro spesso contengono alcune informazioni aggiuntive sul salto successivo). Normalmente esiste anche un router di default, che viene usato se nessuna delle informazioni presenti nella tabella corrisponde al numero della rete di destinazione. Per un host può anche essere accettabile avere soltanto un default router e nient'altro: ciò significa che tutti i datagrammi destinati a host che non si trovano sulla rete fisica a cui è connesso l'host sorgente verranno inviati al router di default.

Possiamo descrivere l'algoritmo di inoltro dei datagrammi nel modo seguente:

```

se (NetworkNum della destinazione =
    NetworkNum di una delle mie interfacce)
allora
    consegna il pacchetto alla destinazione mediante tale interfaccia
altrimenti
    se (NetworkNum della destinazione è presente nella mia tabella di inoltro)
        allora
            consegna il pacchetto al router NextHop
        altrimenti
            consegna il pacchetto al router di default
    
```

Per un host con una sola interfaccia e con il solo router di default nella propria tabella di inoltro, l'algoritmo si semplifica in questo modo:

```

se (NetworkNum della destinazione = mio NetworkNum)
allora
    consegna il pacchetto direttamente alla destinazione
altrimenti
    consegna il pacchetto al router di default
    
```

Vediamo ora come funziona questo algoritmo nella internetwork di esempio della Figura 4.1. Per prima cosa, supponete che H1 voglia inviare un datagramma a H2. Dato che si trovano sulla stessa rete fisica, H1 e H2 hanno lo stesso numero di rete nel proprio indirizzo IP, per cui H1 deduce di poter consegnare direttamente il datagramma a H2 sulla rete Ethernet. Il problema che rimane da risolvere è come possa H1 trovare l'indirizzo Ethernet corretto per H2: si tratta del meccanismo di risoluzione degli indirizzi descritto nella Sezione 4.1.5.

Supponete ora che H1 voglia inviare un datagramma a H8. Dato che questi host sono connessi a reti fisiche diverse, hanno diversi numero di rete, per cui H1 deduce di dover

inviare il datagramma ad un router. R1 è l'unica scelta (è il router di default), per cui H1 invia il datagramma a R1 usando la rete Ethernet. Analogamente, R1 sa di non poter consegnare un datagramma direttamente a H8, dato che nessuna delle sue interfacce si trova sulla rete a cui appartiene H8. Supponete che il router di default di R1 sia R2, per cui R1 invia il datagramma a R2 usando la rete token ring. Ipotizzate ora che la tabella di inoltro di R2 sia quella rappresentata in Tabella 4.1: R2 cerca quindi il numero di rete di H8 (rete numero 1) nella tabella ed inoltra il datagramma a R3. Infine, R3, sapendo di essere sulla stessa rete di H8, inoltra il datagramma direttamente a H8.

Facciamo notare che si possono inserire nella tabella di inoltro anche le informazioni relative alle reti direttamente connesse. Ad esempio, potremmo etichettare le interfacce di rete del router R2, come interfaccia 0 per la linea di collegamento punto-punto (rete 4) e interfaccia 1 per la rete di tipo token ring (rete 3); quindi, R2 avrebbe la tabella di inoltro mostrata in Tabella 4.2.

In questo modo, per qualunque numero di rete che possa trovare in un pacchetto, il router R2 sa cosa fare: se la rete è direttamente connessa a R2, nel qual caso il pacchetto viene consegnato a destinazione da tale rete, oppure la rete è raggiungibile mediante qualche router di "next hop" che R2 può raggiungere attraverso una delle reti a cui è connesso. In entrambi i casi, R2 userà il protocollo ARP, descritto in seguito, per trovare l'indirizzo MAC del nodo a cui va inviato il pacchetto.

La tabella di ipotro usata da R2 è abbastanza semplice da poter essere configurata manualmente, ma di solito queste tabelle sono molto più complesse e vengono costruite da un protocollo di instradamento come uno di quelli descritti nella Sezione 4.2. Notate anche che, nella realtà, i numeri di rete sono solitamente più lunghi (ad esempio, 128.96).

Possiamo ora capire come l'indirizzamento gerarchico, con la suddivisione dell'indirizzo nelle sue parti, di rete e di host, abbia migliorato la scalabilità di una rete di grandi dimensioni. I router, in questo modo, contengono tabelle d'inoltro che elencano soltanto numeri di rete, piuttosto che tutti i nodi della rete. Nel nostro semplice esempio, ciò significa che R2 potrebbe memorizzare in una tabella di sole quattro righe le informazioni necessarie per raggiungere tutti gli host della rete (che sono otto). Anche se in ciascuna rete fisica ci fossero 100 host, R2 avrebbe comunque bisogno di quattro righe soltanto: si tratta di un primo utile passo (anche se assolutamente non quello definitivo) per raggiungere la scalabilità.

Tabella 4.1 Esempio di tabella di inoltro per il router R2 di Figura 4.1.

Numero di rete	Salto successivo
1	R3
2	R1

Tabella 4.2 Tabella di inoltro completa per il router R2 di Figura 4.1.

Numero di rete	Salto successivo
1	R3
2	R1
3	Interfaccia 1
4	Interfaccia 0

Bridge, switch e router

Non è difficile confondersi nella distinzione tra bridge, switch e router, dato che tutti inoltrano messaggi da una linea ad un'altra, sebbene a diversi livelli. Una delle distinzioni che vengono solitamente fatte è basata sugli strati: i bridge sono nodi dello strato di linea di collegamento (in quanto inoltrano frame da una linea ad un'altra per realizzare una rete locale estesa), gli switch sono nodi dello strato di rete (perché inoltrano pacchetti da una linea ad un'altra per realizzare una rete a commutazione di pacchetto) e i router sono nodi dello strato internet (dato che inoltrano datagrammi da una rete ad un'altra per realizzare una internet). In un certo senso, però, tale distinzione è artificiosa. Sicuramente nessuna azienda di apparecchiature di rete chiederà il permesso della polizia degli strati per vendere nuovi prodotti che non si configurino bene in uno strato o in altro.

Ad esempio, abbiamo già visto che un bridge a più porte viene solitamente chiamato switch Ethernet oppure switch per LAN: quindi, la distinzione tra bridge e switch ha già perso parte del suo significato. Per tale ragione spesso i bridge e gli switch vengono raggruppati nella categoria dei "dispositivi di livello 2", dove, in questo contesto, livello 2 significa "al di sopra della strato fisico e al di sotto dello strato internet".

Esiste, tuttavia, una importante distinzione tra gli switch per LAN (o bridge) e gli switch per ATM (e altri switch usati nelle WAN, come gli switch per Frame Relay e X.25). Gli switch per LAN e i bridge dipendono, per il loro funzionamento, dall'algoritmo a spanning tree, mentre gli switch per WAN in generale eseguono protocolli di instradamento che consentono ad ogni switch di apprendere la topologia dell'intera rete. Questa distinzione è importante perché la conoscenza della topologia dell'intera rete consente agli switch di discriminare fra router diversi, mentre, al contrario, l'algoritmo a spanning tree genera un unico albero, fisso, sul quale vengono inoltrati i messaggi. Inoltre, l'algoritmo a spanning tree non scala bene.

Cosa possiamo dire di switch e router? Sono fondamentalmente la stessa cosa, oppure differiscono per qualche funzionalità fondamentale? In questo caso le differenze sono molto meno evidenti. Ad esempio, dato che una singola linea di collegamento punto-punto è essa stessa una rete valida, si può utilizzare un router per connettere un insieme di tali linee. In tale situazione un router assomiglia molto ad uno switch: si tratta di uno switch che inoltra pacchetti IP usando un modello di inoltro privo di connessione e protocolli di instradamento IP. Vedremo bene questa somiglianza quando parleremo, al termine di questa sezione, della implementazione dei router.

Una sostanziale differenza tra una rete ATM basata su switch e la rete Internet basata su router è che la rete Internet fornisce supporto per l'eterogeneità, mentre una rete ATM è composta da linee di collegamento omogenee: Questo supporto per l'eterogeneità è uno dei motivi principali per cui la rete Internet si è diffusa così capillarmente.

Abbiamo così illustrato uno dei principi più importanti nella costruzione di reti scalabili: per ottenere una buona scalabilità, occorre ridurre la quantità di informazioni memorizzate in ciascun nodo e scambiate tra i nodi. Il modo più comune per farlo è l'aggregazione gerarchica. Il protocollo IP utilizza una gerarchia a due livelli, con reti al livello più alto e nodi al livello più basso. Le informazioni vengono aggregate facendo in modo che i router si preoccupino soltanto di raggiungere la rete giusta: l'informazione necessaria ad un router per consegnare un datagramma ad un nodo qualunque di una certa rete è rappresentata da un singolo dato cumulativo.

Implementazione di router

Nella Sezione 3.4 abbiamo visto vari modi per costruire uno switch, spaziando da una workstation ad utilizzo genericò dotata di un appropriato numero di interfacce di rete, fino ad alcuni sofisticati progetti hardware. In generale, lo stesso ventaglio di opzioni è disponibile per la costruzione di router, la maggior parte dei quali è simile a quanto rappresentato in Figura 4.7. Il controllore ha il compito di eseguire i protocolli di instradamento (discussi nella Sezione 4.2) e in generale agisce da punto centrale di controllo del router. La matrice di commutazione (*switching fabric*) trasferisce pacchetti da una porta all'altra, esattamente come in uno switch, e le porte forniscono funzionalità diversificate che consentono al router di interfacciarsi a linee di collegamento di vario tipo (ad esempio, Ethernet, SONET ecc.).

Vale la pena di mettere in evidenza soltanto quei pochi punti nei quali il progetto di router differisce dal progetto di switch. Innanzitutto, i router devono essere progettati per gestire pacchetti di lunghezza variabile, un vincolo che non si applica agli switch ATM ma che certamente è presente negli switch Ethernet o Frame Relay. Si osserva che molti router ad elevate prestazioni sono progettati usando una matrice di commutazione basata su celle, cioè su pacchetti di dimensioni fisse: in tali casi le porte devono essere in grado di convertire pacchetti di lunghezza variabile in celle, e viceversa, che è un problema molto simile a quello della segmentazione e ricostruzione dello standard ATM, descritto nella Sezione 3.3.2.

Ancora, la lunghezza variabile dei datagrammi IP ha come conseguenza il fatto che può essere più difficile caratterizzare le prestazioni di un router rispetto a quanto visto per uno switch che inoltra soltanto celle. Solitamente i router sono in grado di inoltrare un certo numero di pacchetti al secondo, per cui il throughput totale, espresso in bit al secondo, dipende dalla dimensione dei pacchetti. In generale, i progettisti di router devono fare una scelta in merito a quale lunghezza di pacchetto supporteranno alla *velocità della linea* (line rate): se pps (packets per second) è la velocità a cui arrivano i pacchetti ad una particolare porta per essere inoltrati e linerate è la velocità fisica della porta in bit al secondo, allora esiste una qualche packetsize in bit tale che

$$\text{packetsize} \times \text{pps} = \text{linerate}$$

Questa è la dimensione del pacchetto a cui il router può eseguire l'inoltro alla velocità della linea: è probabile che riesca a mantenere per lungo tempo la velocità della linea con pacchetti

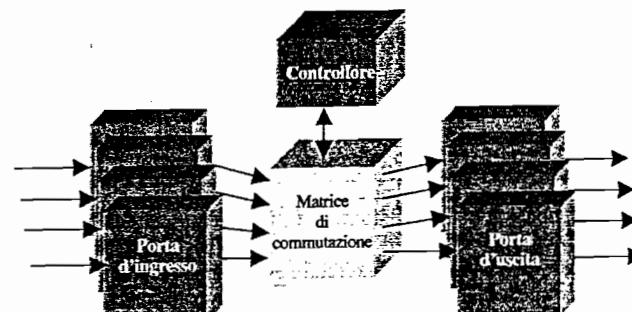


Figura 4.7 Diagramma a blocchi di un router.

più lunghi, ma non con pacchetti più corti. A volte un progettista può decidere che la giusta dimensione di pacchetto da supportare è 40 byte, perché quella è la dimensione minima di un pacchetto IP che trasporti un'intestazione TCP. Un'altra scelta potrebbe essere quella della dimensione *media* che ci si aspetta per i pacchetti, determinata, ad esempio, studiando registrazioni del traffico di rete. Per fare un esempio, misurazioni effettuate su dorsali di Internet suggeriscono che il pacchetto medio di IP ha una lunghezza che si aggira intorno ai 300 byte. Un tale tipo di router, però, avrebbe qualche problema ed inizierebbe a perdere pacchetti se si trovasse di fronte ad una lunga sequenza di pacchetti piccoli, cosa statisticamente probabile di quando in quando ed assai probabile se il router è soggetto ad un attacco attivo (si veda il Capitolo 8). Decisioni progettuali di questo tipo dipendono fortemente da considerazioni economiche e dall'applicazione a cui è destinato il router.

In merito al compito di inoltrare pacchetti IP, i router possono essere caratterizzati da un modello di inoltro *centralizzato* o *distribuito*. Nel modello centralizzato l'algoritmo di inoltro IP, delineato all'inizio della sezione, è eseguito da un singolo processo che gestisce il traffico di tutte le porte. Nel modello distribuito, invece, operano diversi processi di elaborazione, tipicamente uno per ogni porta o ancora più spesso uno per ogni scheda di linea, dove una scheda di linea può servire una o più porte fisiche. Ciascun modello ha vantaggi e svantaggi. A parità di condizioni, il modello di inoltro distribuito dovrebbe essere in grado di inoltrare un maggior numero di pacchetti al secondo attraverso il router nel suo complesso, perché la capacità di elaborazione complessiva è superiore, ma un modello distribuito rende anche più complessa l'architettura software, in quanto ciascun processo di inoltro ha tipicamente bisogno di una propria copia della tabella di inoltro, rendendo così necessario che il controllore assicuri che le tabelle di inoltro vengano aggiornate in modo coerente e con temporizzazioni controllate.

Negli anni più recenti si è creato un consistente interesse nei confronti della possibilità di creare *processori di rete* (network processor) da poter usare nella progettazione di router e di altri apparati di rete. Un processore di rete dovrebbe essere un dispositivo programmabile come un normale processore per PC o per workstation, ma maggiormente ottimizzato per eseguire compiti di rete. Ad esempio, un processore di rete potrebbe avere istruzioni particolarmente adatte all'esecuzione di ricerche in una tabella di indirizzi IP o di calcoli di somme di controllo su datagrammi IP.

Uno dei dibattiti in corso più interessanti in merito ai processori di rete riguarda il fatto che possano fare un lavoro migliore rispetto a soluzioni alternative. Ad esempio, dati i continui e significativi miglioramenti di prestazioni dei processori convenzionali e gli enormi interessi industriali che guidano tali miglioramenti, sono in grado i processori di rete di tenere il passo? Ed un dispositivo che cerca di avere una qualche generalità è in grado di svolgere meglio il proprio compito rispetto ad un chip progettato appositamente e che non faccia altro che, ad esempio, inoltrare pacchetti IP? La risposta a domande di questo tipo dipende, in parte, da ciò che significa "svolgere meglio il proprio compito". Ad esempio, ci saranno sempre compromessi da fare tra il costo dell'hardware, il tempo richiesto per approdare sul mercato (*time to market*), le prestazioni e la flessibilità (cioè la capacità di modificare le caratteristiche di un router dopo che è stato costruito). Vedremo, nel seguito del capitolo e nei capitoli successivi, come possano essere diversi i requisiti per le funzionalità dei router, per cui è ragionevole asserire che nel prossimo futuro esisteranno un gran numero di schemi di progetto per router e che anche i processori di rete avranno un proprio ruolo.

4.1.5 Traduzione di indirizzi (ARP)

Nella sezione precedente abbiamo parlato di come sia possibile far giungere i datagrammi IP alla rete fisica corretta, ma abbiamo sorvolato sul problema di come i datagrammi stessi arrivino ad un particolare host o router di tale rete. Il problema principale è che i datagrammi IP contengono indirizzi IP, mentre l'interfaccia hardware dell'host o del router al quale vogliamo inviare il datagramma è in grado di interpretare solamente lo schema di indirizzamento della particolare rete a cui appartiene. Di conseguenza, dobbiamo tradurre l'indirizzo IP in un indirizzo dello strato di linea di collegamento che abbia senso per quella particolare rete (ad esempio, un indirizzo Ethernet a 48 bit). Possiamo quindi incapsulare il datagramma IP all'interno di un frame che contenga tale indirizzo dello strato di linea di collegamento ed inviarlo o alla destinazione finale o ad un router che prometta di inoltrare il datagramma verso la sua destinazione finale.

Un modo semplice per trasformare un indirizzo IP in un indirizzo di rete fisica consiste nel codificare l'indirizzo fisico di un host nella parte di indirizzo IP riservato all'indirizzo di host. Ad esempio, un host con indirizzo fisico 00100001 01001001 (che ha il valore decimale 33 nel byte superiore e 81 in quello inferiore) potrebbe avere l'indirizzo IP 128.96.33.81. Anche se questa soluzione è stata adottata in alcune reti, essa è limitata dal fatto che gli indirizzi della rete fisica non possono essere più lunghi di 16 bit (in questo esempio) e possono essere di soli 8 bit in un rete di classe C, cosa che certamente non funziona con gli indirizzi Ethernet a 48 bit.

Una soluzione più generale potrebbe essere quella di far gestire a ciascun host una tabella di coppie di indirizzi, cioè una tabella che trasformi indirizzi IP in indirizzi fisici. Anche se una tabella di questo tipo potrebbe essere gestita centralmente da un amministratore di sistema e poi copiata in ciascun host della rete, un approccio migliore sarebbe quello far apprendere dinamicamente i contenuti della tabella a ciascun host tramite la rete stessa, obiettivo che si può raggiungere con il protocollo ARP Address Resolution Protocol. Il compito di ARP è quello di consentire a ciascun host di una rete di costruire una tabella di corrispondenze tra indirizzi IP e indirizzi dello strato di linea di collegamento: dato che queste corrispondenze possono variare nel tempo (ad esempio perché si rompe la scheda Ethernet di un host e la si sostituisce con un'altra avente un diverso indirizzo), le informazioni presenti nella tabella hanno una scadenza e vengono periodicamente eliminate ogni 15 minuti circa. L'insieme di corrispondenze memorizzate in un host in un certo istante costituisce la sua tabella ARP o cache ARP.

ARP sfrutta il fatto che molte tecnologie di rete dello strato di linea di collegamento, come Ethernet e token ring, forniscono supporto per le trasmissioni broadcast. Se un host vuole inviare un datagramma IP ad un host (o ad un router) che si trova sulla sua stessa rete (cioè il nodo sorgente e destinatario hanno lo stesso numero di rete IP), per prima cosa cerca una corrispondenza nella cache. Se non la trova, è necessario invocare sulla rete il protocollo di risoluzione dei indirizzi (ARP), inviando una richiesta ARP mediante un messaggio broadcast nella rete. Tale richiesta contiene l'indirizzo IP in questione (l'indirizzo IP "obiettivo"). Ogni host riceve la richiesta e controlla se corrisponde al proprio indirizzo IP: se è così, invia un messaggio di risposta contenente il proprio indirizzo dello strato di linea al richiedente, che così può aggiungere l'informazione alla propria tabella ARP.

Il messaggio di richiesta contiene anche l'indirizzo IP e l'indirizzo dello strato di linea dell'host sorgente; in questo modo, quando un host invia una richiesta broadcast, tutti gli host della rete vengono a conoscenza degli indirizzi IP e di strato di linea di chi l'ha inviata e

possono inserire tale informazione nella propria tabella ARP. Tuttavia, non tutti gli host aggiungono tale informazione alla propria tabella. Se la tabella dell'host ha già informazioni su quell'host, queste vengono "rinnovate", cioè si reimposta il temporizzatore che farà scadere le informazioni. Se l'host è l'obiettivo della richiesta, allora aggiunge alla propria tabella le informazioni relative alla sorgente, anche se non ne aveva ancora, perché con buona probabilità tale host sorgente sta inviando un messaggio dello strato di applicazione e si dovrà restituire una risposta o una conferma ACK, per la qual cosa ci sarà bisogno dell'indirizzo fisico dell'host sorgente. Se, infine, l'host non è l'obiettivo della richiesta e la sua tabella non contiene già informazioni su tale host sorgente, allora l'informazione non viene aggiunta, perché non vi è ragione di ritenere che l'host ne abbia bisogno: è inutile infittire la tabella.

La Figura 4.8 mostra il formato del pacchetto ARP per una corrispondenza fra indirizzi IP e Ethernet. In realtà, ARP può essere usato per molti altri tipi di corrispondenze: le differenze più significative consistono nella dimensione degli indirizzi. Oltre agli indirizzi IP e dello strato di linea della sorgente e della destinazione, il pacchetto contiene:

- un campo **HardwareType**, che specifica il tipo di rete fisica (ad esempio, Ethernet);
- un campo **ProtocolType**, che specifica il protocollo dello strato superiore (ad esempio, IP);
- i campi **HLen** (lunghezza dell'indirizzo "hardware") e **PLen** (lunghezza dell'indirizzo "di protocollo"), che indicano, rispettivamente, la lunghezza dell'indirizzo dello strato di linea di collegamento e dell'indirizzo del protocollo di livello superiore;
- un campo **Operation**, che specifica se si tratta di una richiesta o di una risposta;
- gli indirizzi hardware (Ethernet) e di protocollo (IP) della sorgente e della destinazione.

Notate che il risultato del processo di risoluzione dell'indirizzo può essere inserito, come colonna aggiuntiva, nella tabella di inoltro rappresentata in Tabella 4.1, per cui, ad esempio, quando R2 deve inoltrare un pacchetto verso la rete 2, non solo trova che il successivo destinatario è R1, ma trova anche l'indirizzo MAC da inserire nel pacchetto che viene inviato a R1.

ATMARP

Dovrebbe essere chiaro, a questo punto, che se una rete ATM deve far parte di una internetwork deve anche fornire un servizio di ARP. Tuttavia, la procedura appena descritta ovviamente

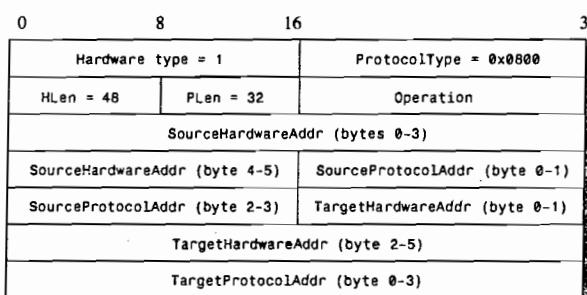


Figura 4.8 Formato del pacchetto ARP per mettere in corrispondenza indirizzi IP e indirizzi Ethernet.

non funzionerà in una semplice rete ATM, perché dipende dal fatto che i pacchetti ARP possano esser inviati in modalità broadcast a tutti gli host di una rete. Una soluzione a questo problema prevede l'utilizzo delle procedure di emulazione di LAN già viste nella Sezione 3.3.5: dato che l'obiettivo di tali procedure consiste nel fare in modo che una rete ATM abbia lo stesso comportamento di una LAN a mezzo fisico condiviso, cosa che prevede il supporto per la trasmissione broadcast, l'effetto è quello di ricondurre il problema della risoluzione degli indirizzi mediante ARP ad un problema risolto in precedenza.

Esistono, tuttavia, situazioni nelle quali non si vuole trattare una rete ATM come una LAN emulata, principalmente perché l'emulazione di LAN può essere assai inefficiente in una grande rete ATM. Ricordate, infatti, che in una LAN emulata può essere necessario inviare molti pacchetti al server che gestisce il servizio broadcast e gli host sconosciuti, il quale poi rimanda tali pacchetti a tutti i nodi della LAN emulata: comportamento che, chiaramente, limita la scalabilità. Il problema è che aggiungere la capacità di trasmissione broadcast ad una rete che ne è intrinsecamente incapace, mentre può essere utile in alcune circostanze, è davvero esagerato se l'unico motivo per cui serve il broadcast è per consentire la risoluzione degli indirizzi.

Per questa ragione esiste una diversa procedura ARP da usare in una rete ATM e che non dipende dalla trasmissione broadcast o dall'emulazione di LAN, procedura nota come **ATMARP** e facente parte del modello *Classic IP over ATM*. Il motivo per cui il modello viene chiamato "classico" diverrà evidente a breve. Come l'emulazione di LAN, per risolvere gli indirizzi ATMARP si basa sull'uso di un server, che in questo caso è detto "server ARP", ed il cui comportamento è descritto nel seguito.

Un concetto chiave del modello *Classic IP over ATM* è la **sottorete logica IP** (LIS, logical IP subnet), un'astrazione che ci consente di prendere una grande rete ATM e di suddividerla in sottoreti più piccole (la "sottorete" verrà definita con precisione nella Sezione 4.3.1, ma in questo caso una sottorete si comporta in modo molto simile ad una vera rete). Tutti i nodi appartenenti ad una stessa sottorete hanno lo stesso numero di rete IP e, proprio come nel protocollo IP "classico", due nodi (host o router) che si trovano nella stessa sottorete possono comunicare tra loro direttamente sulla rete ATM, mentre due nodi che appartengono a due sottoreti diverse dovranno comunicare tramite uno o più router. Un esempio di rete ATM suddivisa in due LIS è rappresentato in Figura 4.9. Notate che l'indirizzo IP dell'host H1 ha il numero di rete 10, come l'interfaccia del router connessa alla LIS di sinistra, mentre H2 ha il numero di rete 12, come l'interfaccia di destra del router: quindi, H1 e il router sono connessi alla stessa LIS (LIS 10), mentre H2 si trova in una diversa sottorete (LIS 12), alla quale è connesso anche il router.

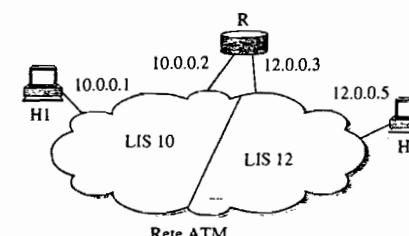


Figura 4.9 Sottoreti logiche IP.

Un vantaggio del modello LIS è che si possono connettere molti host e router ad una grande rete ATM senza dare necessariamente loro indirizzi IP che appartengano tutti alla stessa rete, cosa che semplifica la gestione dell'assegnazione degli indirizzi, ad esempio, nel caso in cui non tutti i nodi connessi alla rete ATM siano sotto il controllo della stessa entità amministrativa. La divisione della rete ATM in un certo numero di LIS migliora anche la scalabilità, limitando il numero di nodi a cui deve fornire supporto un singolo server ARP.

Il compito fondamentale di un server ARP è quello di consentire ai nodi di una LIS di tradurre indirizzi IP in indirizzi ATM senza usare la trasmissione broadcast. Ogni nodo della LIS deve essere configurato con l'indirizzo ATM del server ARP, in modo che, quando viene acceso, possa instaurare un circuito virtuale verso il server. Instaurato tale circuito virtuale, il nodo invia al server ARP un messaggio di registrazione, contenente il proprio indirizzo IP e indirizzo ATM: in questo modo il server ARP costruisce un base di dati completa di tutte le coppie (indirizzo IP, indirizzo ATM). Una volta che questa base di dati si sia costituita, qualsiasi nodo che voglia inviare un pacchetto ad un indirizzo IP può chiedere al server ARP di fornirgli il corrispondente indirizzo ATM. Dopo averlo ricevuto, il nodo sorgente può usare la segnalazione ATM per instaurare un circuito virtuale verso tale indirizzo ATM e, quindi, inviare il pacchetto. Le corrispondenze tra indirizzi IP e indirizzi ATM possono essere conservate in una cache, esattamente come nella procedura ARP convenzionale. Inoltre, il nodo può mantenere attivo il circuito virtuale che ha stabilito verso quella destinazione ATM per tutto il tempo in cui vi sia abbastanza traffico da giustificare la presenza, evitando così il ritardo necessario ad instaurare nuovamente il circuito virtuale quando arriva il pacchetto successivo.

Una interessante conseguenza del modello *Classic IP over ATM* è che due nodi della stessa rete ATM che appartengano a due diverse sottoreti non possono instaurare un circuito virtuale diretto tra loro, perché ciò violerebbe la regola che la comunicazione fra una sottorete e un'altra deve transitare attraverso un router. Ad esempio, l'host H1 e l'host H2 della Figura 4.9 non possono instaurare un circuito virtuale diretto, secondo questo modello, ma devono avere entrambi un circuito virtuale verso il router R. La semplice motivazione per questa regola è che l'instradamento IP funziona bene quando tale regola viene rispettata, come avviene nelle reti che non siano ATM. Sono state introdotte nuove tecniche per aggirare tale regola, ma sono caratterizzate da una considerevole complessità e presentano problemi di robustezza.

Abbiamo appena visto il meccanismo di base fornito da IP per gestire sia l'eterogeneità che la scalabilità. Per quanto riguarda l'eterogeneità, IP definisce, innanzitutto, un modello di servizio di tipo best-effort che fa ipotesi veramente minimali sulle reti sottostanti: la cosa più importante da ricordare è che questo modello si basa sulla consegna non affidabile di datagrammi. Successivamente, IP pone due condizioni aggiuntive a partire da quel punto fermo: (1) un formato di pacchetto unificato (e per far funzionare tale formato su reti aventi diversi valori di MTU usa il meccanismo della frammentazione/ricostruzione) e (2) uno spazio di indirizzamento globale per identificare tutti gli host (e il protocollo ARP consente a questo indirizzamento globale di funzionare su reti aventi schemi di indirizzamento fisico diversi). Sul fronte della scalabilità, IP usa l'aggregazione gerarchica per ridurre la quantità di informazioni necessarie ad inoltrare i pacchetti. In particolare, gli indirizzi IP vengono suddivisi in una componente di rete e una componente di host, con i pacchetti che vengono prima instradati verso la rete di destinazione e poi consegnati all'host corretto su tale rete.

AUTOMATICA

4.1.6 Configurazione di host (DHCP)

Nella Sezione 2.6 abbiano evidenziato che gli indirizzi Ethernet vengono configurati nell'adattatore di rete dal produttore e tale processo viene gestito in modo da garantire che questi indirizzi siano unici al mondo, una condizione evidentemente sufficiente per garantire che un insieme di host connessi ad una singola rete Ethernet (che può anche essere una LAN estesa) avranno indirizzi univoci. Inoltre, l'unicità è l'unica cosa che viene richiesta agli indirizzi Ethernet.

Gli indirizzi IP, invece, non soltanto devono essere unici in una certa internetwork, ma devono anche riflettere la struttura della rete stessa. Come notato in precedenza, gli indirizzi IP contengono una componente che identifica la rete e una che identifica l'host all'interno della rete, e la componente di rete deve essere uguale per tutti gli host di una stessa rete. Di conseguenza, non è possibile che gli indirizzi IP vengano configurati negli host una volta per tutte quando vengono costruiti, perché ciò vorrebbe dire che il costruttore dovrebbe sapere quali host andranno a finire in quali reti, e ciò vorrebbe anche dire che un host, una volta connesso ad una rete, non potrebbe più essere spostato in un'altra. Per questa ragione gli indirizzi IP devono essere riconfigurabili. → IP ruota girevole

Oltre ad un indirizzo IP, un host deve conoscere anche altre informazioni prima di poter inviare pacchetti, le più importanti delle quali è l'indirizzo di un router di default, a cui poter inviare quei pacchetti il cui indirizzo di destinazione non si trova sulla stessa rete a cui appartiene l'host sorgente.

La maggior parte dei sistemi operativi fornisce uno strumento mediante il quale un amministratore di sistema, o anche un utente, può configurare manualmente le informazioni IP necessarie all'host, tuttavia questa configurazione manuale ha alcuni ovvi svantaggi. Uno di questi svantaggi è che è semplicemente molto oneroso configurare direttamente tutti gli host di una grande rete, specialmente in considerazione del fatto che tali host non sono raggiungibili mediante la rete finché non sono stati configurati. Un problema ancora più importante è che il processo di configurazione è molto sensibile agli errori, dato che bisogna garantire che ciascun host abbia il numero di rete corretto e che non vi siano due host con lo stesso indirizzo IP. Per questi motivi si rendono necessari metodi di configurazione automatica; il principale dei quali utilizza un protocollo noto come Dynamic Host Configuration Protocol (DHCP).

Il protocollo DHCP si basa sulla presenza di un server DHCP che ha la responsabilità di fornire agli host le informazioni di configurazione. Deve esserci almeno un server DHCP per ogni dominio amministrativo. Nello schema più semplice, il server DHCP ha la funzione di banca dati centralizzata per le informazioni di configurazione degli host. Considerate, ad esempio, il problema della gestione degli indirizzi nella internetwork di una grande azienda: il protocollo DHCP consente agli amministratori della rete di non andare in giro con un elenco di indirizzi a configurare manualmente tutti gli host della rete, ma le informazioni di configurazione di ciascun host, memorizzate nel server DHCP, vengono recuperate automaticamente da ogni host quando viene acceso o viene connesso alla rete. Ad ogni modo, è sempre l'amministratore di rete a scegliere l'indirizzo che verrà ricevuto da ciascun host, solo che lo memorizza nel server. Secondo questo modello, le informazioni di configurazione di ciascun host sono memorizzate in una tabella che ha come indice una qualche forma di identificatore univoco per il client, che tipicamente è il suo "indirizzo hardware" (ad esempio, l'indirizzo Ethernet del suo adattatore di rete):

Un uso più sofisticato del protocollo DHCP consente all'amministratore di evitare anche di dover assegnare gli indirizzi ai singoli host: secondo questo modello, il server DHCP

gestisce un insieme di indirizzi disponibili, che consegna agli host su richiesta, riducendo così sensibilmente il lavoro di configurazione che deve essere svolto dall'amministratore, che deve ora soltanto allocare per ciascuna rete un intervallo di indirizzi IP (aventi tutti lo stesso numero di rete).

Dato che l'obiettivo di DHCP è quello di minimizzare la quantità di configurazioni manuali necessarie per far funzionare ciascun host, sarebbe sostanzialmente un mancato raggiungimento di tale obiettivo se ogni host dovesse essere configurato con l'indirizzo di un DHCP server, per cui il primo problema che deve essere risolto dal protocollo DHCP è la scoperta di un server.

Per contattare un server DHCP, un host appena acceso o appena connesso invia un messaggio DHCPDISCOVER ad uno speciale indirizzo IP (255.255.255.255), che è un indirizzo broadcast del protocollo IP: ciò significa che il messaggio verrà ricevuto da tutti gli host e da tutti i router di tale rete (i router non inoltrano tali pacchetti verso altre reti, impedendo che vengano trasmessi in modalità broadcast sull'intera Internet). Nel caso più semplice, uno di tali nodi è il DHCP server per la rete, che quindi risponderà all'host che aveva inviato il messaggio di identificazione (risposta che verrà ignorata da tutti gli altri nodi). Ciò nonostante, non è realistico richiedere che esista un server DHCP su ciascuna rete, perché si avrebbe un numero potenzialmente molto grande di server da configurare in modo corretto e consistente: per questo motivo il protocollo DHCP utilizza il concetto di agente di collegamento (relay agent). In ciascuna rete esiste almeno un agente di collegamento, che viene configurato con un'unica informazione: l'indirizzo IP del server DHCP. Quando un agente di collegamento riceve un messaggio DHCPDISCOVER, lo inoltra in modalità unicast al server DHCP ed attende la risposta, per inoltrarla al cliente che aveva fatto la richiesta. La Figura 4.10 mostra il processo che viene messo in atto per consegnare ad un server DHCP remoto un messaggio che proviene da un host.

La Figura 4.11 mostra il formato di un messaggio DHCP, che viene inviato usando un protocollo chiamato UDP (User Datagram Protocol), eseguito al di sopra del protocollo IP. Il protocollo UDP verrà analizzato in dettaglio nel capitolo successivo; in questo contesto, l'unica cosa da sapere è che fornisce una chiave di demultiplexing del tipo "Questo è un pacchetto DHCP".

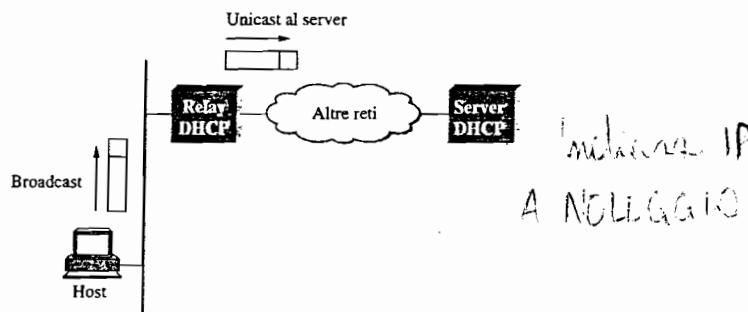


Figura 4.10 Un agente di collegamento (relay) DHCP riceve un messaggio DHCPDISCOVER broadcast da un host e invia un messaggio DHCPDISCOVER unicast al server DHCP.

Operation	HType	HLen	Hops
Xid			
Secs			Flags
			ciaddr
			yiaddr
			siaddr
			giaddr
			Chaddr (16 byte)
			sname (64 byte)
			file (128 byte)
			options

Figura 4.11 Formato del pacchetto DHCP.

Il protocollo DHCP deriva da un protocollo precedente, chiamato BOOTP, per cui alcuni dei campi del pacchetto non sono strettamente attinenti alla configurazione dell'host. Quando cerca di ottenere le informazioni di configurazione, il client inserisce nel campo chaddr il proprio indirizzo hardware (ad esempio, il proprio indirizzo Ethernet); il server DHCP risponde compilando il campo yiaddr ("il tuo" indirizzo IP, la "y" è l'iniziale di "your") e restituendo il pacchetto al client. Nel campo options si possono inserire altre informazioni, quali l'indirizzo del router di default che deve essere usato dal client.

Nei casi in cui il protocollo DHCP assegna dinamicamente gli indirizzi IP agli host, è chiaro che gli host non possono conservare indefinitamente tali indirizzi, perché ciò provocherebbe l'esaurimento dell'insieme di indirizzi a disposizione del server. Allo stesso tempo, non si può affidare al client il compito di restituire il proprio indirizzo, perché il client potrebbe essere spento o disconnesso dalla rete all'improvviso. Per questo motivo, il protocollo DHCP consegna gli indirizzi "a noleggio" per un certo periodo di tempo: terminato il periodo assegnato, il server può reinserire l'indirizzo nell'insieme degli indirizzi disponibili. Ovviamente, un host con un indirizzo a noleggio lo deve rinnovare periodicamente, qualora permanga connesso alla rete e stia funzionando correttamente.

Il protocollo DHCP evidenzia un aspetto importante della scalabilità: la scalabilità della gestione della rete. Anche se quando si parla di scalabilità ci si concentra spesso sul fatto di evitare che crescano troppo rapidamente le informazioni sullo stato della rete contenute nei singoli dispositivi, è altrettanto importante fare attenzione all'aumento della complessità di gestione della rete. Consentendo ai gestori della rete di configurare un insieme di indirizzi IP per una rete, anziché gli indirizzi IP dei singoli host, il protocollo DHCP migliora la gestione della rete.

Noteate che il protocollo DHCP introduce anche un certo grado di complessità nella gestione della rete, perché rende più dinamica la corrispondenza tra gli host fisici e i loro indirizzi IP, rendendo più difficile il lavoro del gestore della rete quando, ad esempio, vi è la necessità di identificare un host guasto.

4.1.7 Segnalazione di errori (ICMP)

Il prossimo argomento riguarda la gestione degli errori in Internet. Anche se il protocollo IP non ha problemi ad eliminare un datagramma quando il suo inoltro diventa difficile (ad esempio, quando un router non sa come inoltrare il datagramma, oppure quando un frammento di un datagramma non arriva a destinazione), non sempre fallisce silenziosamente.

Il protocollo IP viene sempre configurato con un protocollo ausiliario, noto con il nome di Internet Control Message Protocol (ICMP), che **definisce un insieme di messaggi d'errore** che vengono inviati all'host sorgente quando un router o un host non è in grado di elaborare con successo un datagramma IP. Ad esempio, il protocollo ICMP definisce alcuni messaggi d'errore per segnalare che un host di destinazione è irraggiungibile (magari a causa di un guasto ad una linea di connessione), che un processo di ricostruzione di un datagramma non è andato a buon fine, che il campo TTL ha raggiunto il valore 0, che è fallita la verifica del checksum dell'intestazione di un pacchetto IP, e così via.

Il protocollo ICMP definisce anche alcuni messaggi di controllo che un router può restituire ad un host sorgente. Uno dei più utili, tra questi ultimi, è chiamato ICMP-Redirect, che indica alla sorgente che esiste un percorso migliore verso la destinazione e viene usato nella situazione seguente. Supponete che un host sia connesso ad una rete dotata di due router, R1 e R2, e che l'host usi R1 come router di default. Se R1 riceve un datagramma dall'host e, sulla base delle proprie tabelle di inoltro, decide che R2 sarebbe una scelta migliore per quel particolare indirizzo di destinazione, allora invia all'host un messaggio ICMP-Redirect, inducendolo ad usare R2 per tutti i futuri datagrammi indirizzati a tale destinazione. In conseguenza di ciò, l'host aggiunge questo nuovo percorso alla propria tabella di instradamento.

4.1.8 Reti virtuali e tunnel

Terminiamo la nostra presentazione del protocollo IP prendendo in esame un argomento di cui forse non siete a conoscenza, ma che sta assumendo un'importanza sempre maggiore. Fino a questo punto, la nostra discussione si è incentrata sul fatto di rendere possibile la comunicazione reciproca tra due nodi appartenenti a reti diverse, senza alcuna restrizione. Solitamente, questo è proprio l'obiettivo della rete Internet: tutti vogliono essere in grado di inviare messaggi di posta elettronica a chiunque, ed il creatore di un nuovo sito Web vuole raggiungere il pubblico più ampio possibile. Tuttavia, esistono molte situazioni in cui viene richiesto un tipo di connessione più controllata: un'importante esempio di tale situazione è la *rete privata virtuale* (VPN, virtual private network).

Il termine "VPN" viene pesantemente abusato, con varie definizioni, ma possiamo definire intuitivamente una VPN considerando dapprima l'idea di una rete privata. Le aziende con molte sedi si costruiscono spesso reti private noleggiano linee di trasmissione dalle compagnie telefoniche e usandole per interconnettere le sedi. In una rete di questo tipo, la comunicazione può avvenire solamente tra le sedi dell'azienda, cosa spesso auspicabile per motivi di sicurezza. Per rendere *virtuale* una rete privata, le linee di trasmissione noleggiate (che non sono condivise con altre aziende) vengono rimpiazzate da una qualche rete condivisa. Un circuito virtuale è un sostituto molto ragionevole per una linea noleggiata, dato che fornisce un collegamento logico di tipo punto-punto fra le sedi di un'azienda. Ad esempio, se l'azienda X ha un circuito virtuale fra la sede A e la sede B, può ovviamente scambiare pacchetti tra le sedi A e B, ma non c'è alcuna possibilità per l'azienda Y di inviare pacchetti alla sede B senza aver prima instaurato il proprio circuito virtuale verso tale sede: ciò può

4.1 Semplice interconnessione di reti (IP)

essere impedito dal gestore della rete, eliminando così la possibilità di connessioni indesiderate tra l'azienda X e l'azienda Y.

La Figura 4.12(a) mostra le reti private di due aziende distinte, che, nella Figura 4.12(b), sono state trasformate in reti private virtuali, mantenendo le medesime limitazioni di connettività delle reti fisicamente private. Dato che, nel secondo caso, le reti private condividono le stesse linee di trasmissione e gli stessi commutatori, diciamo che sono state create due reti private virtuali.

Nella Figura 4.12, per fornire un servizio di connettività controllata fra le sedi viene usata una rete ATM o di tipo Frame Relay, ma è possibile fornire un servizio simile anche usando una rete IP, una internetwork: non è, però, possibile connettere semplicemente le sedi delle diverse aziende ad un'unica internetwork, poiché ciò consentirebbe anche la connessione tra l'azienda X e l'azienda Y, che vogliamo invece impedire. Per risolvere questo problema, dobbiamo introdurre un nuovo concetto, il *tunnel IP*.

Possiamo immaginare un tunnel IP come una linea virtuale di connessione punto-punto fra una coppia di nodi che sono, in realtà, separati da un numero arbitrario di reti. La linea virtuale viene realizzata dal router che si trova all'ingresso del tunnel, fornendo ad esso l'indirizzo IP del router che si trova all'uscita del tunnel. Ogni volta che il router all'ingresso del tunnel vuole inviare un pacchetto lungo tale linea virtuale di connessione, incapsula il pacchetto all'interno di un datagramma IP, la cui intestazione contiene come indirizzo di destinazione

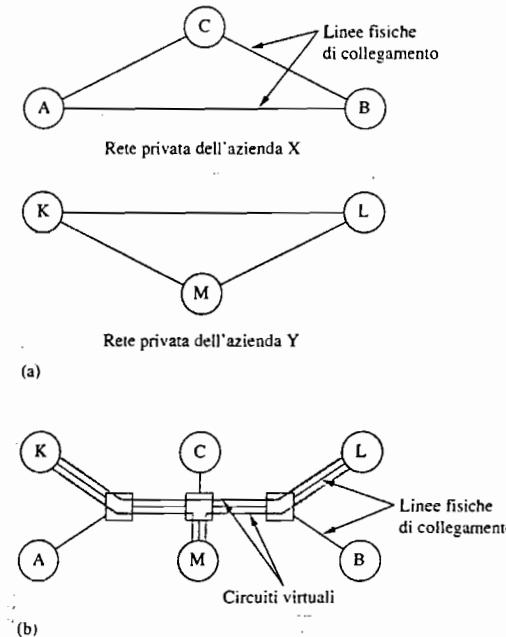


Figura 4.12 Esempio di reti private virtuali: (a) due reti private distinte; (b) due reti private virtuali che condividono alcuni commutatori.

zione l'indirizzo del router che si trova all'uscita del tunnel e come indirizzo del mittente l'indirizzo del router che effettua l'incapsulamento.

Nella tabella di inoltro del router che si trova all'ingresso del tunnel questa linea virtuale assomiglia molto ad una normale linea di collegamento. Considerate, ad esempio, la rete di Figura 4.13, dove è stato configurato fra R1 e R2 un tunnel, al quale è stato assegnato il numero di interfaccia virtuale 0. La tabella di inoltro di R1, quindi, è rappresentata in Tabella 4.3.

Il router R1 ha due interfacce fisiche. L'interfaccia 0 lo connette alla rete 1, mentre l'interfaccia 1 lo connette ad una grande internetwork ed è, quindi, la linea di default per tutto il traffico che non soddisfa una condizione più specifica nella tabella di inoltro. In aggiunta, R1 ha un'interfaccia virtuale, che è l'interfaccia del tunnel. Supponete che R1 riceva dalla rete 1 un pacchetto che contiene un indirizzo della rete 2: la tabella di inoltro afferma che tale pacchetto deve essere inviato all'interfaccia virtuale 0. Per inviare un pacchetto a tale interfaccia, il router prende il pacchetto, vi aggiunge un'intestazione IP indirizzata a R2 ed, infine, elabora tale nuovo pacchetto come se lo avesse appena ricevuto. L'indirizzo di R2 è 10.0.0.1: poiché il numero di rete di questo indirizzo è 10, e non 1 o 2, verrà inoltrato tramite l'interfaccia di default verso la internetwork.

Una volta che il pacchetto è uscito da R1, al resto del mondo appare come un normale pacchetto IP destinato a R2 e viene inoltrato di conseguenza: tutti i router della internetwork lo inoltrano nel modo usuale, finché non arriva a R2. Quando R2 riceve il pacchetto, si accorge che contiene il proprio indirizzo come destinazione, per cui elimina l'intestazione IP ed esamina il carico utile del pacchetto, trovando un pacchetto incapsulato il cui indirizzo di destinazione si trova nella rete 2: a questo punto, R2 elabora tale pacchetto con la stessa procedura che segue per ogni altro pacchetto che riceve. Poiché R2 è direttamente connesso alla rete 2, inoltra il pacchetto in tale rete. La Figura 4.13 mostra le modifiche nell'incapsulamento del pacchetto mentre si sposta attraverso la rete.

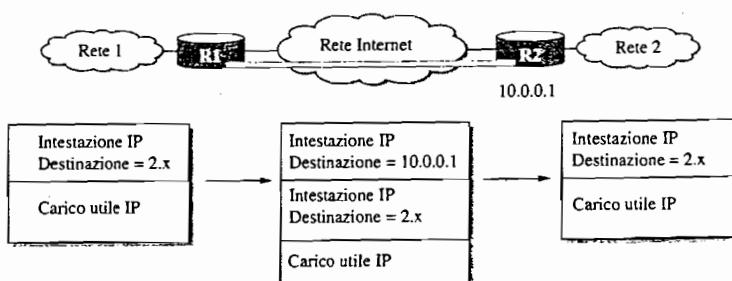


Figura 4.13 Un tunnel attraverso una internetwork.

Tabella 4.3 Tabella di inoltro per il router R1 della Figura 4.13.

Numero di rete	Salto successivo
1	Interfaccia 0
2	Interfaccia virtuale 0
Default	Interfaccia 1

4.2 Instradamento (routing)

Mentre R2 svolge la funzione di punto terminale del tunnel, niente gli impedisce di svolgere le normali funzioni di router. Ad esempio, potrebbe ricevere pacchetti che non viaggiano all'interno del tunnel, ma che sono indirizzati a reti verso la quali conosce un percorso di instradamento, e li inoltrerebbe nel modo consueto.

Vi potreste chiedere per quale motivo qualcuno può ritenere utile fare tutti questi sforzi per creare un tunnel e modificare l'incapsulamento di un pacchetto mentre si sposta all'interno di una internetwork. La sicurezza è uno di questi motivi, di cui parleremo in maggiore dettaglio nel Capitolo 8. Con l'aggiunta della codifica cifrata, un tunnel può divenire una linea di connessione molto privata all'interno di una rete pubblica. Un altro motivo può essere il fatto che R1 e R2 potrebbero disporre di alcune funzionalità non presenti, in generale, in tutte le reti interposte tra loro, come, ad esempio, l'inoltro di pacchetti multicast (multicast routing): collegando tali router con un tunnel, possiamo costruire una rete virtuale in cui tutti i router aventi queste caratteristiche appaiono direttamente connessi tra loro. La rete MBone (multicast backbone), che esamineremo nella Sezione 4.4, è proprio costruita in questo modo. Una terza motivazione per la costruzione di tunnel è il trasporto, attraverso una rete IP, di pacchetti di protocolli diversi da IP: se i router che si trovano agli estremi del tunnel sanno come gestire questi protocolli, il tunnel IP appare loro come una linea di collegamento punto-punto mediante la quale è possibile inviare pacchetti che non siano pacchetti IP. I tunnel forniscono anche uno strumento per costringere un pacchetto ad essere consegnato in un luogo particolare, anche se la sua intestazione originale (quella incapsulata all'interno dell'intestazione relativa al tunnel) potrebbe portare il pacchetto in un luogo diverso: vedremo un'applicazione di questa possibilità quando considereremo gli host mobili, nella Sezione 4.2.5. Concludendo, vediamo che la creazione di tunnel ("il tunneling") è una tecnica potente e piuttosto generale per costruire linee di collegamento virtuali attraverso internetwork.

Il tunneling ha anche alcuni svantaggi. Innanzitutto, la lunghezza dei pacchetti aumenta: di conseguenza, ciò potrebbe portare ad un significativo spreco di banda per pacchetti piccoli. Inoltre, ci possono essere problemi di prestazioni per i router che si trovano agli estremi del tunnel, che si trovano a dover fare più lavoro rispetto all'inoltro normale, aggiungendo ed eliminando le intestazioni relative al tunneling. Infine, c'è un costo di gestione per l'entità amministrativa che ha la responsabilità di instaurare il tunnel ed assicurarsi che venga gestito correttamente dai protocolli di routing.

4.2 Instradamento (routing)

Sia in questo capitolo sia nel capitolo precedente abbiamo ipotizzato che gli switch e i router abbiano sufficienti conoscenze della topologia della rete da poter scegliere la porta corretta verso cui inviare ciascun pacchetto. Nel caso di circuiti virtuali, l'instradamento è un problema soltanto per i pacchetti che richiedono una connessione: tutti i pacchetti successivi seguono lo stesso percorso seguito dal pacchetto di richiesta. Nelle reti a datagramma, come le reti IP, invece, l'instradamento è un problema che deve essere risolto per ogni pacchetto. In entrambi i casi, uno switch o un router deve saper determinare, in base all'indirizzo di destinazione, quale sia la porta d'uscita migliore per inoltrare il pacchetto verso la sua destinazione. Come abbiamo visto nella Sezione 3.1.1, uno switch prende questa decisione consultando una tabella di inoltro. A questo punto, il problema fondamentale dell'instradamento è: come fanno gli switch e i router ad acquisire le informazioni presenti nelle proprie tabelle di inoltro?

A questo punto evidenziamo nuovamente una differenza importante, spesso trascurata, che esiste tra l'inoltro (forwarding) e l'instradamento (routing): L'inoltro consiste nel prendere un pacchetto, esaminare il suo indirizzo di destinazione, consultare una tabella ed inviare il pacchetto verso la direzione indicata dalla tabella; nelle sezioni precedenti abbiamo visto parecchi esempi di inoltro. L'instradamento è, invece, il processo mediante il quale vengono costruite le tabelle di inoltro. Notiamo anche che l'inoltro richiede una procedura relativamente semplice e ben definita, eseguita localmente in un nodo, mentre l'instradamento dipende da complessi algoritmi distribuiti che hanno continuato ad evolvere nella storia delle reti di calcolatori.

Mentre i termini *tabella di inoltro* e *tabella di instradamento* vengono a volte usati come sinonimi, noi faremo distinzione fra loro. La tabella di inoltro viene usata quando si inoltra un pacchetto, per cui deve contenere informazioni sufficienti a portare a termine il compito di inoltrare un pacchetto: ciò significa che una riga nella tabella di inoltro contiene una corrispondenza tra un numero di rete e un'interfaccia d'uscita, nonché alcune informazioni a livello MAC, come l'indirizzo Ethernet della destinazione successiva. La tabella di instradamento, invece, è la tabella che viene costruita dagli algoritmi di instradamento prima di costruire la tabella di inoltro: in generale, contiene corrispondenze tra numeri di rete e destinazioni successive (*next hop*), ma può anche contenere informazioni sul modo in cui sono state acquisite tali corrispondenze, per consentire ai router di decidere sulla loro eventuale eliminazione.

Mantenere effettivamente distinte le tabelle di inoltro dalle tabelle di instradamento è una scelta di implementazione, ma vi sono parecchie buone ragioni per usare strutture dati separate. Ad esempio, la tabella di inoltro deve avere una struttura tale da ottimizzare la ricerca di un numero di rete quando si inoltra un pacchetto, mentre la tabella di instradamento deve essere ottimizzata per calcolare le modifiche nella topologia della rete. In alcuni casi, la tabella di inoltro può essere addirittura realizzata con hardware specifico, mentre ciò è assai raro per la tabella di instradamento. La Tabella 4.4 mostra un esempio di una riga di ciascun tipo di tabella. In questo caso, la tabella di instradamento dice che la rete numero 10 può essere raggiunta tramite un router di next hop con indirizzo IP 171.69.245.10, mentre la tabella di inoltro contiene l'informazione precisa relativa a come inoltrare un pacchetto verso tale salto successivo: invia il pacchetto verso l'interfaccia numero 0 con un indirizzo MAC uguale a 8:0:2b:e4:b1:2. Si noti che quest'ultima informazione è fornita dal protocollo ARP.

Prima di analizzare in dettaglio l'instradamento, dobbiamo tornare alla domanda chiave che ci dobbiamo porre ogni volta che cerchiamo di costruire qualcosa per Internet: "È una soluzione scalabile?" La risposta, per gli algoritmi e i protocolli descritti in questa sezione, è negativa: sono stati progettati per reti di dimensioni modeste (in pratica, meno di un centinaio di nodi).

Tabella 4.4 Un esempio di riga di tabella di instradamento (a) e di inoltro (b).

(a)	
Numero di rete	Salto successivo
10	171.69.245.10

Numero di rete	Interfaccia	Indirizzo MAC
10	if0	8:0:2b:e4:b1:2

io di nodi). Tuttavia, le soluzioni che descriviamo servono come blocco elementare per un'infrastruttura di instradamento gerarchica che viene usata oggi giorno da Internet. In particolare, i protocolli descritti in questa sezione sono noti, nel loro insieme, come protocolli di instradamento *intradominio*, o *interior gateway protocols* (IGP). Per comprendere il significato di questi termini, dobbiamo definire cosa sia un *dominio* di instradamento: una valida definizione operativa è "una internetwork in cui tutti i router sono gestiti dalla stessa entità amministrativa" (ad esempio, la rete di un'università o la rete di un singolo fornitore di accesso ad Internet). L'importanza di questa definizione apparirà chiara nella sezione successiva, quando esamineremo i protocolli di instradamento *interdominio*; per ora, la cosa importante da ricordare è che stiamo considerando il problema dell'instradamento nel contesto di reti di dimensione piccola o media, non di una rete delle dimensioni di Internet.³

4.2.1 La rete rappresentata con un grafo

L'instradamento è, sostanzialmente, un problema di teoria dei grafici. La Figura 4.14 mostra un grafo che rappresenta una rete. I nodi del grafo, etichettati da A a F, possono essere host, switch, router o reti. Per la nostra discussione preliminare, ci concentriamo sul caso in cui i nodi sono router. I rami del grafo corrispondono alle linee di collegamento nella rete; a ciascun ramo è associato un *costo*, che fornisce un'indicazione di quanto sia preferibile inviare traffico su tale linea. Nella Sezione 4.2.4 si affronterà il problema di come assegnare i costi ai rami del grafo.³

Il problema fondamentale dell'instradamento consiste nel trovare il percorso a costo minore fra due nodi qualsiasi, dove il costo di un percorso è uguale alla somma dei costi di tutti i rami che compongono il percorso stesso. Per una rete semplice come quella di Figura 4.14, si potrebbe ipotizzare di calcolare tutti i percorsi migliori e memorizzarli in una memoria non volatile all'interno di ciascun nodo, ma un tale approccio statico ha parecchi svantaggi:

- non gestisce i guasti di un nodo o di una linea di collegamento
- non considera l'inserimento di nuovi nodi o di nuove linee di collegamento
- implica che i costi associati a ciascun ramo non possono cambiare, anche se parrebbe

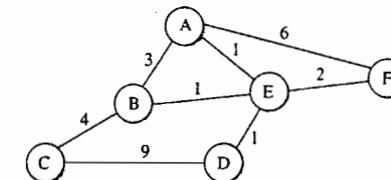


Figura 4.14 Una rete rappresentata con un grafo.

³ Nelle reti (grafici) usate come esempio in questo capitolo, usiamo rami non orientati ed assegniamo un singolo costo a ciascun ramo, facendo una piccola semplificazione. Sarebbe più accurato usare rami orientati, avendo così, tipicamente, due rami fra ciascun nodo, uno in una direzione e uno nella direzione opposta, ciascuno con il proprio costo.

sensato voler assegnare temporaneamente un costo elevato ad una linea di collegamento che sia particolarmente carica di traffico

Per questi motivi, nella maggior parte delle reti di interesse pratico si risolve il problema dell'instradamento eseguendo sui nodi protocolli di instradamento opportuni, che forniscono un approccio dinamico e distribuito alla soluzione del problema di identificare il percorso a costo minore, anche in presenza di guasti su nodi e linee e modificando i costi dei rami. Notate il termine "distribuito" in quest'ultima frase: è difficile rendere scalabili soluzioni centralizzate, per cui tutti i protocolli di instradamento più diffusi usano algoritmi distribuiti.

La natura distribuita degli algoritmi di instradamento è uno dei motivi principali per cui questo è stato un campo ricco di ricerche e sviluppi: ci sono molti problemi da risolvere per far funzionare adeguatamente gli algoritmi distribuiti. Ad esempio, gli algoritmi distribuiti non eliminano la possibilità che, ad un dato istante, due router abbiano idee diverse in merito a quale sia il percorso migliore verso una certa destinazione: ciascuno dei due può pensare che l'altro sia più vicino alla destinazione e decidere di inviarsi un pacchetto l'un l'altro. Tale pacchetto si troverebbe, chiaramente, a circolare indefinitamente, finché non fosse risolta la discrepanza di opinione tra i due router, cosa che sarebbe meglio risolvere nel più breve tempo possibile. Questo è soltanto un esempio del tipo di problemi che devono essere affrontati dai protocolli di instradamento.

Per iniziare la nostra analisi, ipotizziamo che siano noti i costi dei rami nella rete. Prenderemo in esame le due categorie principali di protocolli di instradamento: a vettore di distanza (distance vector) e a stato delle linee (link state). Nella Sezione 4.2.4 torneremo sul problema di come calcolare i costi dei rami in modo sensato.

4.2.2 Vettore di distanza (RIP)

L'idea che sta alla base dell'algoritmo a vettore di distanza (*distance vector*) è suggerita dal suo stesso nome⁴: ciascun nodo costruisce un array monodimensionale (un vettore) contenente le "distanze" (cioè i costi) relative a tutti gli altri nodi e distribuisce tale vettore ai suoi più immediati vicini. L'ipotesi iniziale per l'instradamento a vettore di distanza è che ciascun

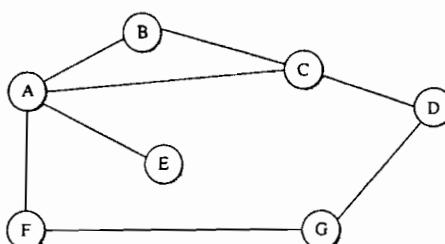


Figura 4.15 Instradamento con vettore di distanza: una rete di esempio.

⁴ Un altro nome molto usato per identificare questa categoria di algoritmi è Bellman-Ford, dal nome dei suoi inventori.

4.2 Instradamento (routing)

nodo conosca il costo delle linee che lo collegano ai vicini a cui è connesso direttamente; Una linea di collegamento guasta assume un costo infinito.

Per vedere come funziona l'algoritmo di instradamento a vettore di distanza, è opportuno considerare un esempio, come quello rappresentato in Figura 4.15, dove il costo di ciascuna linea di collegamento è assunto uguale a 1, in modo che il percorso di costo minore sia semplicemente quello con il minor numero di salti (dato che tutti i rami hanno il medesimo costo, non mostriamo i costi nel grafo). Possiamo rappresentare con una tabella come quella mostrata in Tabella 4.5 le conoscenze di ciascun nodo in merito alle distanze dagli altri nodi. Notate che ciascun nodo conosce solamente le informazioni presenti in una riga della tabella, quella che porta il nome del nodo nella colonna di sinistra. La visione globale presentata qui non è disponibile in alcun singolo punto della rete.

Possiamo considerare ciascuna riga della Tabella 4.5 come un elenco di distanze tra un nodo e tutti gli altri nodi, che rappresenta le informazioni correntemente presenti in un nodo. Inizialmente ciascun nodo imposta a 1 il costo necessario per raggiungere i vicini a cui è direttamente connesso e a ∞ quello verso tutti gli altri nodi. Così, A ritiene inizialmente che B sia raggiungibile con un solo salto e che D sia irraggiungibile. La tabella di instradamento memorizzata in A tiene conto di questo insieme di convincimenti e contiene anche il nome del nodo successivo usato da A per raggiungere uno dei nodi raggiungibili. Inizialmente, quindi, la tabella di instradamento di A sarebbe uguale a quella di Tabella 4.6.

Il passo successivo nell'instradamento a vettore di distanza consiste nell'invio, da parte di ogni nodo ai nodi ad esso direttamente connessi, di un messaggio contenente il proprio

Tabella 4.5 Distanze iniziali memorizzate in ciascun nodo (visione globale).

Informazioni memorizzate nel nodo	Distanza per raggiungere il nodo						
	A	B	C	D	E	F	G
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	1	0	∞	∞	∞
E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0

Tabella 4.6 Tabella di instradamento iniziale nel nodo A.

Destinazione	Costo	Salto successivo
B	1	B
C	1	C
D	∞	-
E	1	E
F	1	F
G	∞	-

elenco di distanze. Ad esempio, il nodo F comunica al nodo A di poter raggiungere il nodo G con costo 1; A sa anche di poter raggiungere il nodo F con costo 1, per cui addiziona tali costi per ottenere il costo da associare al raggiungimento di G tramite F. Questo costo totale vale 2, che è minore dell'attuale costo infinito, per cui A memorizza di poter raggiungere G con costo 2 passando tramite F. In modo analogo, A apprende da C che D può essere raggiunto tramite C con costo 1: addiziona tale costo al costo necessario per raggiungere C (che vale 1) e decide che D può essere raggiunto passando per C con costo 2, che è meglio del costo precedente, infinito. Al tempo stesso, A apprende da C che B può essere raggiunto tramite C con costo 1, per cui conclude che il costo necessario per raggiungere B passando per C è 2: dato che questo costo è peggiore del costo attuale per raggiungere B (che è 1), questa nuova informazione viene ignorata.

A questo punto, A può aggiornare la propria tabella di instradamento con i costi e con i next hop necessari per raggiungere tutti i nodi della rete. Il risultato è mostrato in Tabella 4.7.

In assenza di cambiamenti di topologia, bastano pochi scambi di informazioni fra nodi vicini per fare in modo che tutti i nodi abbiano una tabella di instradamento completa: il procedimento mediante il quale tutti i nodi assumono informazioni di instradamento coerenti viene chiamato *convergenza*. La Tabella 4.8 mostra l'insieme finale dei costi fra ciascun nodo ed ogni altro nodo, dopo che l'instradamento è giunto a convergenza. Vogliamo sottolineare il fatto che nessun nodo della rete possiede tutte le informazioni presenti in questa tabella: ciascun nodo conosce soltanto le informazioni presenti nella propria tabella di instradamento. L'eleganza di un algoritmo distribuito come questo sta nel consentire a tutti i nodi di ottenere una visione coerente della rete in assenza di un'autorità centralizzata.

Tabella 4.7 Tabella di instradamento finale nel nodo A.

Destinazione	Costo	Salto successivo
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

Tabella 4.8 Distanze finali memorizzate in ciascun nodo (visione globale).

Informazioni memorizzate nel nodo	Distanza per raggiungere il nodo						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

4.2 Instradamento (routing)

Prima di terminare la presentazione dell'instradamento a vettore di distanza dobbiamo completare alcuni dettagli: Innanzitutto, notiamo che un nodo decide di inviare un aggiornamento di instradamento ai propri vicini in due situazioni diverse. Il primo caso avviene in occasione di aggiornamenti *periodici*: ciascun nodo invia un messaggio di aggiornamento automatico ad intervalli regolari, anche se non ci sono state modifiche: questo fa sapere agli altri nodi che il nodo è ancora attivo e consente di continuare a ricevere informazioni che potrebbero essere loro utili nel caso che qualche percorso diventi inagibile. La frequenza di questi aggiornamenti periodici varia di protocollo in protocollo, ma è tipicamente dell'ordine di alcuni secondi, o pochi minuti. Il secondo meccanismo, a volte chiamato aggiornamento *innescato (triggered)*, entra in gioco quando un nodo riceve da uno dei suoi vicini un aggiornamento che provoca la modifica di uno dei percorsi all'interno della sua tabella di instradamento; in sostanza, ogni volta che viene modificata la tabella di instradamento di un nodo, il nodo invia un aggiornamento ai propri vicini, provocando eventualmente una modifica nelle loro tabelle, con conseguente messaggio di aggiornamento inviato ai loro vicini.

Considerate ora cosa accade in caso di guasto di una linea di connessione o di un nodo. Il nodo che per primo si accorge del guasto invia ai propri vicini nuovi elenchi di distanze e, solitamente, in un tempo molto breve il sistema si stabilizza in un nuovo stato. In merito al problema di identificare un guasto, ci sono un paio di risposte diverse. Secondo un primo approccio, un nodo tiene costantemente sotto controllo una linea di collegamento verso un altro nodo, inviando pacchetti di controllo ed osservando se viene ricevuta la relativa conferma: Un diverso approccio prevede che un nodo decida che una linea (o il nodo che si trova all'altro capo della linea) sia guasta quando non riceve per alcuni periodi consecutivi il previsto aggiornamento periodico di instradamento.

Per capire cosa accade quando un nodo identifica un guasto su una linea di collegamento, considerate cosa avviene quando F si accorge che la linea verso G non funziona correttamente. Dapprima, F imposta ad infinito la propria distanza verso G e trasmette tale informazione ad A. Poiché A sa di poter raggiungere G lungo un percorso di 2 salti che passa attraverso F, anche A imposta ad infinito la propria distanza verso G. Tuttavia, ricevendo il successivo aggiornamento da C, A viene a sapere che C può raggiungere G con un percorso di 2 salti, per cui A calcola di poter raggiungere G con 3 salti passando per C, che è meglio di infinito, e aggiorna di conseguenza la propria tabella. Quando A segnala il cambiamento ad F, F apprende di poter raggiungere G al costo di 4 salti tramite A, che è meno di infinito, ed il sistema diviene nuovamente stabile.

Sfortunatamente, circostanze poco diverse potrebbero impedire alla rete di stabilizzarsi. Supponete, ad esempio, che si guasti la linea di collegamento tra A ed E. Nel successivo ciclo di aggiornamenti, A segnala di trovarsi a distanza infinita da E, ma B e C segnalano di trovarsi a distanza 2 da E. In relazione al preciso succedersi degli eventi, può accadere quanto segue: il nodo B, dopo aver saputo che tramite C si può raggiungere E in 2 salti, deduce di poter raggiungere E in tre salti e lo segnala ad A; il nodo A deduce, quindi, di poter raggiungere E in 4 salti e lo segnala a C; il nodo C deduce di poter raggiungere E in 5 salti, e via così. Questo ciclo termina soltanto quando le distanze raggiungono un valore sufficientemente elevato da essere considerato infinito; nel frattempo, nessuno dei nodi si rende conto che E è irraggiungibile e le tabelle di instradamento della rete non si stabilizzano. Questa situazione è nota con il nome di *problema del conteggio verso l'infinito (count-to-infinity)*.

Esistono varie soluzioni parziali a questo problema. La prima consiste nell'utilizzo di un numero relativamente piccolo per rappresentare l'infinito. Ad esempio, potremmo decidere che il numero massimo di salti necessari in una rete non sia mai maggiore di 16, scegliendo

di conseguenza 16 come valore per rappresentare l'infinito. Questa tecnica, quantomeno, pone un limite ragionevole al tempo necessario per raggiungere un conteggio uguale ad infinito, ma, ovviamente, potrebbe anche porre problemi qualora la rete crescesse fino al punto in cui alcuni nodi si trovassero a distanza relativa superiore a 16.

Una tecnica che migliora il tempo necessario alla stabilizzazione dell'instradamento è detta *divisione dell'orizzonte* (split horizon), che sfrutta l'idea seguente: quando un nodo invia un aggiornamento di instradamento ai propri vicini, non invia ad un vicino quei percorsi che ha appreso da esso. Ad esempio, se B ha nella propria tabella il percorso (E, 2, A), allora sa di averlo appreso dal nodo A: quando invia un aggiornamento al nodo A, non include il percorso (E, 2) nell'aggiornamento. In una variante più robusta dello split horizon, detta *split horizon with poison reverse*, B in realtà invia anche tale percorso ad A, ma con informazioni così negative da impedire che A usi B per raggiungere E. Ad esempio, B invia ad A il percorso (E, ∞). Il problema di entrambe queste tecniche consiste nel fatto che funzionano soltanto per eliminare i cicli di instradamento che coinvolgono due soli nodi. Per cicli che coinvolgono più nodi, servono misure più drastiche. Proseguendo con l'esempio precedente, se B e C, dopo aver saputo da A del guasto sulla linea, avessero atteso un po' prima di segnalare i propri percorsi verso E, avrebbero saputo entrambi di non avere, in realtà, alcun percorso verso E. Sfortunatamente, questo approccio ritarda la convergenza del protocollo, mentre la velocità della convergenza è uno dei vantaggi più importanti del suo avversario, l'instradamento a stato della linea (link state routing), che sarà argomento della Sezione 4.2.3.

Implementazione

Il codice che implementa questo algoritmo è quasi banale, per cui ne forniamo soltanto una traccia. La struttura Route definisce ciascuna entità nella tabella di instradamento, mentre la costante MAX_TTL specifica quanto a lungo ciascuna riga venga conservata nella tabella prima di essere eliminata.

```
#define MAX_ROUTES 128 /* dimensione massima della tabella */
#define MAX_TTL 120 /* tempo (in secondi) prima di eliminare
                     un percorso */

typedef struct {
    NodeAddr Destination; /* indirizzo della destinazione */
    NodeAddr NextHop; /* indirizzo del nexthop */
    int Cost; /* distanza */
    u_short TTL; /* tempo di vita */
} Route;

int numRoutes = 0;
Route routingTable[MAX_ROUTES];
```

La procedura mergeRoute aggiorna la tabella di instradamento dei nodi locali sulla base di un nuovo percorso. Anche se non è mostrato qui, una funzione viene innescata periodicamente da un temporizzatore per scandire l'elenco dei percorsi presenti nella tabella di instradamento del nodo, decrementare il campo TTL (tempo di vita) di ciascun percorso ed eliminare i percorsi che hanno il tempo di vita uguale a 0. Notate che, però, il campo TTL viene ripristinato al valore MAX_TTL ogni volta che un percorso viene confermato da un messaggio di aggiornamento proveniente da un nodo vicino.

4.2 Instradamento (routing)

```
void mergeRoute(Route *new)
{
    int i;

    for (i = 0; i < numRoutes; ++i)
    {
        if (new->Destination == routingTable[i].Destination)
        {
            if (new->Cost + 1 < routingTable[i].Cost)
            {
                /* trovato un percorso migliore */
                break;
            }
            else if (new->NextHop == routingTable[i].NextHop)
            {
                /* potrebbe essere cambiata la distanza
                   dell'attuale next hop */
                break;
            }
            else
            {
                /* ignora il percorso */
                return;
            }
        }
        if (i == numRoutes)
        {
            /* questo è un percorso completamente nuovo: c'è spazio? */
            if (numRoutes < MAX_ROUTES)
            {
                ++numRoutes;
            }
            else
            {
                /* non c'è posto nella tabella, quindi lascia perdere */
                return;
            }
        }
        routingTable[i] = *new;
        /* ripristina TTL */
        routingTable[i].TTL = MAX_TTL;
        /* indica la distanza per raggiungere il nodo successivo */
        ++routingTable[i].Cost;
    }
}
```

Infine, la procedura updateRoutingTable è la funzione principale che invoca mergeRoute per inserire tutti i percorsi presenti in un aggiornamento di instradamento ricevuto da un nodo vicino.

```
void updateRoutingTable(Route *newRoute, int numNewRoutes)
{
    int i;
```

```

for (i = 0; i < numNewRoutes; ++i)
{
    mergeRoute(&newRoute[i]);
}
}
}

```

Routing Information Protocol (RIP)

Uno dei protocolli di instradamento maggiormente utilizzati nelle reti IP è il Routing Information Protocol (RIP). Il suo diffuso utilizzo è dovuto soprattutto al fatto che venne distribuito insieme alla popolare versione di Unix denominata BSD (Berkeley Software Distribution), da cui sono state derivate molte versioni commerciali di Unix. Si tratta, inoltre, di un protocollo molto semplice: RIP è l'esempio canonico di protocollo di instradamento basato sull'algoritmo a vettore di distanza appena descritto.

Nelle internetwork, i protocolli di instradamento sono leggermente diversi dal modello di grafo idealmente descritto in precedenza. In una internetwork, l'obiettivo dei router è quello di apprendere come inoltrare pacchetti a varie reti per cui, anziché comunicare il costo (in questo caso, la distanza) necessario per raggiungere altri router, i router comunicano il costo per raggiungere altre reti. Ad esempio, nella Figura 4.16, il router C comunicherrebbe al router A di poter raggiungere le reti 2 e 3 (alle quali è direttamente connesso) con un costo uguale a 0, le reti 5 e 6 con costo 1 e la rete 4 con costo 2.

Esaminando il formato del pacchetto RIP, in Figura 4.17, ciò appare evidente: la maggior parte del pacchetto è dedicata alle coppie *(indirizzo-di-rete, distanza)*. Ciò nonostante, i principi che stanno alla base dell'algoritmo di instradamento non mutano. Ad esempio, se il router A apprende dal router B che la rete X può essere raggiunta tramite B ad un costo inferiore rispetto a quanto possibile tramite il next hop presente nella propria tabella di instradamento, A aggiorna di conseguenza il costo e le informazioni del next hop relative a quel numero di rete.

Il protocollo RIP è, a tutti gli effetti, un'implementazione pedissequa dell'instradamento a vettore di distanza. I router che eseguono RIP inviano i loro aggiornamenti ogni 30 secondi, oltre ad inviare un messaggio di aggiornamento ogni volta che un aggiornamento ricevuto da un altro router provoca una modifica nella propria tabella di instradamento. Una caratteristica interessante del protocollo RIP è il supporto per più famiglie di indirizzi: RIP non è limitato agli indirizzi IP, la parte *indirizzo-di-rete* del messaggio è, in realtà, rappresentata da una coppia *(famiglia, indirizzo)*. La versione 2 del protocollo RIP (RIPv2) presenta anche alcune caratteristiche relative alla scalabilità, che verranno discusse nella sezione successiva.

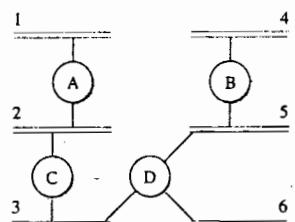


Figura 4.16 Esempio di rete su cui viene eseguito il protocollo RIP

0	8	16	31
Command	Version	Deve essere zero	
Famiglia della rete 1	Indirizzo della rete 1		
	Indirizzo della rete 1		
	Distanza della rete 1		
Famiglia della rete 2	Indirizzo della rete 2		
	Indirizzo della rete 2		
	Distanza della rete 2		

Figura 4.17 Formato del pacchetto RIP.

Come vedremo nel seguito, in un protocollo di instradamento è possibile utilizzare un insieme di metriche diverse per caratterizzare i costi delle linee di collegamento. Il protocollo RIP considera l'approccio più semplice, assegnando a tutte le linee un costo unitario, proprio come nel nostro esempio precedente: in questo modo, quindi, il protocollo cerca sempre di trovare il percorso che presenta il numero minimo di salti. I valori validi per le distanze vanno da 1 a 15, con il valore 16 che rappresenta l'infinito: ciò limita RIP all'esecuzione in reti abbastanza piccole, con percorsi non più lunghi di 15 salti.

4.2.3 Stato delle linee (OSPF)

L'instradamento mediante lo stato delle linee (*link state routing*) costituisce la seconda principale categoria dei protocolli di instradamento intradominio. Le ipotesi di partenza sono abbastanza simili a quelle dell'instradamento a vettore di distanza: si assume che ciascun nodo sia in grado di conoscere lo stato delle linee (funzionanti o non funzionanti) che lo collegano ai propri vicini e, di nuovo, vogliamo fornire a ciascun nodo informazioni sufficienti a trovare il percorso di costo minimo verso qualsiasi destinazione. L'idea che sta alla base dell'instradamento mediante lo stato delle linee è molto semplice: ciascun nodo sa come raggiungere i vicini a cui è connesso direttamente e, se garantiamo che la totalità di queste informazioni venga trasmessa ad ogni nodo, allora ogni nodo avrà sufficienti informazioni sulla rete da poter costruire una mappa completa della rete stessa. Questo costituisce, chiaramente, una condizione sufficiente (ancorché non necessaria) per trovare il percorso più breve verso qualsiasi punto interno alla rete. Quindi, i protocolli di instradamento mediante lo stato delle linee si affidano a due meccanismi: la trasmissione affidabile delle informazioni relative allo stato delle linee di collegamento ed il calcolo di percorsi a partire dall'insieme delle informazioni accumulate.

Inondazione affidabile

L'inondazione affidabile (reliable flooding) è un processo che garantisce a tutti i nodi che partecipano al protocollo di instradamento di ricevere da tutti gli altri nodi una copia delle informazioni relative allo stato delle linee di collegamento. Come suggerito dal termine "inondazione", l'idea si basa sul fatto che un nodo invii le proprie informazioni verso tutte le linee ad esso direttamente connesse, e che ogni nodo che riceve tali informazioni le inoltri verso tutte le linee ad esso connesse. Questo processo continua finché l'informazione non ha raggiunto tutti i nodi della rete.

Più precisamente, ciascun nodo crea un pacchetto di aggiornamento, chiamato anche pacchetto dello stato delle linee (LSP, link-state packet), che contiene le informazioni seguenti:

~~CONTENUTO del LSP~~

- l'identificativo (ID) del nodo che ha creato il pacchetto
- un elenco dei vicini direttamente connessi a quel nodo, con il costo della linea di collegamento verso ciascun nodo
- un numero di sequenza
- il tempo di vita del pacchetto

Le prime due informazioni sono necessarie per il calcolo dei percorsi, mentre le ultime due servono a rendere affidabile il processo di inondazione dei pacchetti verso tutti i nodi. L'affidabilità consiste, anche, nel garantire che ciascun nodo possieda la copia più recente delle informazioni, poiché la rete può essere attraversata da LSP multipli e fra loro contradditori. Si è visto, in pratica, che rendere affidabile l'inondazione è abbastanza difficile (ad esempio, un'versione precedente dell'instradamento a stato delle linee usata in ARPANET provocò un guasto nella rete stessa, nel 1981).

L'inondazione funziona nel modo seguente. Innanzitutto, viene resa affidabile la trasmissione degli LSP fra router adiacenti, usando conferme (acknowledgement) e ritrasmissioni, esattamente come nel protocollo affidabile per lo strato di linea di collegamento visto nella Sezione 2.5. Tuttavia, sono necessarie molte altre cose per consegnare in modo affidabile un LSP a tutti i nodi di una rete mediante l'inondazione.

Considerate un nodo X che riceve una copia di un LSP generato da un altro nodo Y (che può essere un qualsiasi altro router nello stesso dominio di instradamento di X). X controlla di aver memorizzato un'altra copia di un LSP proveniente da Y: se non ne ha, memorizza il pacchetto appena ricevuto, altrimenti confronta i numeri di sequenza. Se il pacchetto nuovo, ha un numero di sequenza maggiore di quello memorizzato, si assume che sia più recente e viene memorizzato, andando a sostituire quello precedente. Un numero di sequenza inferiore (o uguale) contraddistingue, invece, un LSP meno recente (o non più recente) di quello memorizzato in precedenza, per cui il pacchetto verrebbe ignorato senza alcuna azione ulteriore. Se il pacchetto ricevuto è il più recente, X invia una copia di tale LSP a tutti i propri vicini, tranne al vicino da cui ha ricevuto il pacchetto stesso (quest'ultima clausola aiuta a porre fine al processo di inondazione che coinvolge un LSP). Dato che X trasmette il pacchetto ricevuto a tutti i propri vicini, i quali a loro volta si comporteranno allo stesso modo, la copia più recente dell'LSP raggiunge, prima o poi, tutti i nodi.

La Figura 4.18 mostra un LSP che inonda una piccola rete (ciascun nodo viene evidenziato dopo aver memorizzato il nuovo LSP). In Figura 4.18(a) il pacchetto arriva al nodo X, che lo invia ai propri vicini A e C nella Figura 4.18(b). A e C non lo inviano di nuovo a X, ma lo inviano a B. Poiché B riceve due copie identiche del pacchetto, accetterà quella che arriva per

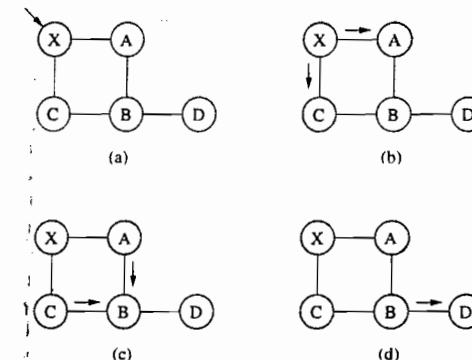


Figura 4.18 Inondazione di pacchetti con lo stato delle linee: (a) il pacchetto LSP arriva al nodo X; (b) X lo trasmette ad A e C; (c) A e C trasmettono il pacchetto LSP a B (ma non a X); (d) l'inondazione termina.

prima, ignorando la seconda in quanto duplicata. Successivamente, B trasmette il pacchetto a D, che non ha vicini verso cui trasmetterlo ed il processo termina.

Proprio come nel protocollo RIP, ciascun nodo genera nuovi LSP in due occasioni: dopo la scadenza di un temporizzatore periodico, oppure in seguito ad una modifica della topologia della rete. Tuttavia, l'unica motivazione dovuta alla topologia che induce un nodo a generare un LSP è il guasto di una linea di collegamento ad esso direttamente connessa, oppure il guasto di uno dei nodi ad esso vicini. Il guasto di una linea può, in alcuni casi, essere identificato dal protocollo dello strato di collegamento. Il guasto di un nodo vicino o la perdita di connessione verso tale nodo si può identificare usando periodici pacchetti di "saluto" ("hello" packet): ciascun nodo li invia ad intervalli predefiniti ai propri vicini e la linea di collegamento verso un vicino viene dichiarata guasta se non si ricevono pacchetti di saluto da quel vicino dopo un tempo sufficientemente lungo, generando così un nuovo LSP per testimoniare questo evento.

Uno degli obiettivi di progetto più importanti del meccanismo di inondazione di un protocollo a stato delle linee è che le informazioni più recenti vengano trasmesse a tutti i nodi il più velocemente possibile, mentre le informazioni obsolete vengano rimosse dalla rete e non fatte più circolare. Inoltre, è ovviamente auspicabile la minimizzazione della quantità totale di traffico generato nella rete: dopo tutto, si tratta solamente di "carico inutile" (*overhead*) dal punto di vista di coloro che usano la rete per le proprie applicazioni. I paragrafi successivi descrivono alcuni dei modi con cui si raggiungono questi obiettivi.

Un modo semplice per ridurre l'overhead consiste nell'*evitare* di generare LSP a meno che non sia assolutamente necessario. Ciò si può fare usando temporizzatori molto lunghi (spesso con valori di ore) per la generazione periodica di pacchetti. Dato che il protocollo di inondazione è veramente affidabile quando vi sono modifiche topologiche, è ragionevole ipotizzare che non sia necessario inviare tanto spesso messaggi che dicono che "nulla è cambiato".

Per garantire che le informazioni obsolete vengano sostituite da quelle più recenti, i pacchetti LSP contengono numeri di sequenza. Ogni volta che un nodo genera un nuovo LSP, incrementa di un'unità il numero di sequenza. Diversamente da quanto avviene con i numeri di sequenza usati in molti protocolli, non è previsto che questi numeri di sequenza

tornino al valore iniziale, per cui il campo relativo deve essere molto grande (diciamo 64 bit). Se un nodo ha un guasto e poi riprende il proprio funzionamento, riprende a numerare la sequenza a partire da 0. Se il nodo è rimasto inattivo per lungo tempo, tutti i pacchetti LSP ad esso relativi sono scaduti (come descritto in seguito), altrimenti prima o poi il nodo riceverà una copia di uno dei propri LSP avente un numero di sequenza più elevato di quello attualmente in uso, provocando un aggiornamento del numero di sequenza interno al nodo, che procederà usando il numero di sequenza ricevuto. Ciò garantisce che i suoi nuovi LSP sostituiranno tutti i propri LSP obsoleti rimasti nella rete prima che il nodo avesse il guasto.

I pacchetti LSP hanno anche un tempo di vita (TTL, *time to live*), usato per garantire che le informazioni obsolete relative allo stato delle linee vengano prima o poi eliminate dalla rete. Prima di trasmettere ai propri vicini un LSP ricevuto, un nodo ne decrementa sempre il campo TTL; inoltre, ciascun nodo "fa invecchiare" i pacchetti LSP memorizzati al proprio interno. Quando il campo TTL di un LSP raggiunge il valore 0, il nodo ritrasmette tale pacchetto con TTL a zero, evento che viene interpretato da tutti i nodi della rete come un segnale di rimozione per quel pacchetto.

Calcolo dei percorsi

Una volta che un nodo abbia ricevuto una copia del pacchetto LSP di ogni altro nodo, è in grado di calcolare una mappa completa della topologia della rete e, in base a tale mappa, può decidere quale sia il percorso migliore verso qualsiasi destinazione. Il problema, ora, è come vengano calcolati precisamente i percorsi a partire da queste informazioni. La soluzione è basata su un algoritmo ben noto della teoria dei grafi, l'**algoritmo di Dijkstra per il percorso più breve.**

Definiamo dapprima l'algoritmo di Dijkstra in termini di teoria dei grafi. Immaginate che un nodo prenda tutti i pacchetti LSP che ha ricevuto e costruisca una rappresentazione della rete, in cui N indica l'insieme dei nodi nel grafo, $l(i, j)$ indica il costo (peso) non negativo associato al ramo che esiste fra i nodi $i, j \in N$, con $l(i, j) = \infty$ se non esiste un ramo che collega i e j . Nella descrizione che segue, indichiamo con $s \in N$ tale nodo, cioè il nodo che esegue l'algoritmo per trovare il percorso più breve verso tutti gli altri nodi presenti in N . Ancora, l'algoritmo gestisce le due variabili seguenti: M indica l'insieme di nodi finora presi in considerazione dall'algoritmo e $C(n)$ indica il costo del percorso da s a ciascun nodo n . Date queste definizioni, l'algoritmo è definito come segue:

$$\begin{aligned} M &= \{s\} \\ \text{per ogni } n \text{ in } N - M \\ C(n) &= l(s, n) \\ \text{finché } (N \neq M) \\ M &= M \cup \{w\} \text{ tale che } C(w) \text{ sia il minimo per tutti } w \text{ in } (N - M) \\ \text{per ogni } n \text{ in } (N - M) \\ C(n) &= \text{MIN}(C(n), C(w) + l(w, n)) \end{aligned}$$

L'algoritmo funziona fondamentalmente in questo modo. Si parte con M contenente il nodo s e si inizializza la tabella dei costi (i valori $C(n)$) verso gli altri nodi usando i costi noti per raggiungere i nodi direttamente connessi. Successivamente, si cerca il nodo raggiungibile al costo minore (w) e lo si inserisce in M . Infine, si aggiorna la tabella dei costi considerando il costo necessario per raggiungere nodi tramite w . Nell'ultima linea di codice che descrive l'algoritmo viene scelto come nuovo percorso verso il nodo n quello che passa per il nodo w .

se il costo totale necessario per andare dalla sorgente a w e, poi, da w a n è inferiore al costo del vecchio percorso memorizzato verso n . Questa procedura viene ripetuta finché tutti i nodi non sono stati inseriti in M .

In pratica, ciascuno switch calcola la propria tabella di instradamento a partire direttamente dai pacchetti LSP che ha raccolto, usando un'implementazione dell'algoritmo di Dijkstra detta algoritmo *di ricerca in avanti* (forward search algorithm). Scendendo nei dettagli, ciascuno switch utilizza due liste, note con il nome di **Tentative** e **Confirmed**. Ciascuna di queste liste contiene un insieme di dati aventi la forma *(Destination, Cost, NextHop)* e l'algoritmo funziona in questo modo:

1. Inizializza la lista **Confirmed** con il dato relativo a me stesso, con costo 0.
2. Seleziona il pacchetto LSP relativo al nodo (chiamato **Next**) appena inserito nella lista **Confirmed** nel passo precedente.
3. Perciascun vicino (**Neighbor**) di **Next**, calcola il costo (**Cost**) necessario per raggiungere **Neighbor** come somma del costo da me stesso a **Next** e da **Next** a **Neighbor**.
 - a) se **Neighbor** non è nella lista **Confirmed** né nella lista **Tentative**, aggiungi *(Neighbor, Cost, NextHop)* alla lista **Tentative**, dove **NextHop** è la direzione verso cui devo andare per raggiungere **Next**.
 - b) Se **Neighbor** è nella lista **Tentative** e **Cost** è minore del costo attualmente associato a **Neighbor**, sostituisci il dato attualmente memorizzato per **Neighbor** con *(Neighbor, Cost, NextHop)*, dove **NextHop** è la direzione verso cui devo andare per raggiungere **Next**.
4. Se la lista **Tentative** è vuota, termina l'algoritmo; altrimenti, scegli dalla lista **Tentative** il dato avente il costo inferiore, spostalo nella lista **Confirmed** e torna al passo 2.

L'algoritmo sarà di più facile comprensione dopo aver osservato un esempio. Considerate la rete descritta in Figura 4.19. Notate che, diversamente dal nostro esempio precedente, questa rete ha costi variabili sui rami. La Tabella 4.9 tiene traccia dei passi necessari per la costruzione della tabella di instradamento per il nodo D. Le due uscite di D sono state indicate usando i nomi dei nodi a cui conducono, B e C. Notate che l'algoritmo sembra puntare a false conclusioni (come il percorso di costo 11 verso B che viene inizialmente aggiunto alla lista **Tentative**), ma termina con i percorsi di costo minimo verso tutti i nodi.

L'algoritmo di instradamento a stato delle linee ha molte caratteristiche interessanti: ha dimostrato di giungere rapidamente alla stabilità, non genera molto traffico e risponde con prontezza a modifiche nella topologia o ai guasti di nodi. Di converso, la quantità di informazioni memorizzata in ciascun nodo (un LSP per ogni altro nodo della rete) può essere molto grande. Questo è uno dei problemi principali dell'instradamento e rientra nel più generale

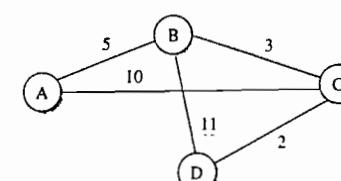


Figura 4.19 Instradamento a stato delle linee: una rete d'esempio.

Tabella 4.9 Passi eseguiti per costruire la tabella di instradamento per il nodo D (Figura 4.19).

Passo	Confirmed	Tentative	Commenti
1	(D.0.-)		Poiché D è l'unico nuovo membro della lista Confirmed , considera il suo LSP.
2	(D.0.-)	(B.11,B) (C.2,C)	Il pacchetto LSP di D dice che si può raggiungere B attraverso B con costo 11, che è meglio di qualsiasi altra informazione presente in entrambe le liste, per cui inserisce tale informazione nella lista Tentative : lo stesso vale per C.
3	(D.0.-) (C.2,C)	(B,11,B)	Inserisci nella lista Confirmed il membro di Tentative avente costo minore (C). Successivamente, esamina il pacchetto LSP del nuovo membro della lista Confirmed (C).
4	(D.0.-) (C.2,C)	(B,5,C) (A,12,C)	Il costo per raggiungere B attraverso C è 5, per cui sostituisci (B, 11, B). Il pacchetto LSP di C ci dice anche che possiamo raggiungere A con costo 12.
5	(D.0.-) (C.2,C) (B,5,C)	(A,12,C)	Sposta nella lista Confirmed il membro di Tentative avente costo minore (B), quindi considera il suo LSP.
6	(D.0.-) (C.2,C) (B,5,C)	(A,10,C)	Dato che possiamo raggiungere A con costo 5 attraverso B, sostituisci l'informazione presente in Tentative .
7	(D.0.-) (C.2,C) (B,5,C) (A,10,C)		Sposta nella lista Confirmed il membro di Tentative avente costo minore (A), dopodiché termina.

problema della scalabilità. Nella prossima sezione verranno presentate alcune soluzioni sia per il problema specifico (la quantità di memoria potenzialmente richiesta a ciascun nodo) sia per il problema generale (la scalabilità).

Quindi, la differenza fra gli algoritmi a vettore di distanza e a stato delle linee può essere riassunta come segue: **con il vettore di distanza, ciascun nodo scambia informazioni solamente con i vicini a cui è direttamente connesso, ma trasferisce loro tutte le informazioni che ha appreso (cioè la distanza verso tutti gli altri nodi); mentre con lo stato delle linee ciascun nodo scambia informazioni con tutti gli altri, trasferendo solamente le informazioni di cui è assolutamente certo (cioè soltanto lo stato delle linee a cui è direttamente connesso).**

Il protocollo OSPF (Open Shortest Path First)

Uno dei protocolli a stato delle linee più diffusi è OSPF (Open Shortest Path First). La prima parola, "Open", si riferisce al fatto che è uno standard aperto e non proprietario, creato sotto gli auspici di IETF. La parte "SPF" deriva da un nome alternativo dell'intradamento a stato delle linee. OSPF aggiunge all'algoritmo a stato delle linee descritto in precedenza parecchie caratteristiche, tra cui le seguenti:

4.2 Intradamento (routing)

- Autenticazione dei messaggi di intradamento. Si tratta di una caratteristica interessante, perché è veramente assai comune che un host configurato male possa decidere di poter raggiungere qualsiasi altro host nell'universo a costo zero. Quando un host trasmette questo tipo di informazione, tutti i router circostanti aggiornano le proprie tabelle di intradamento, puntando a tale host; di conseguenza, tale host riceve grandi quantità di dati, che, in realtà, non sa come gestire. Tipicamente li ignora tutti, bloccando la rete. Tale fenomeno disastroso può essere molte volte evitato richiedendo che gli aggiornamenti di intradamento siano autenticati. Le prime versioni di OSPF usavano per l'autenticazione una semplice **password di 8 byte**, che non è una forma di autenticazione sufficientemente robusta da impedire l'accesso ad utenti smaliziati, ma elimina molti dei problemi dovuti a banali errori di configurazione (una forma simile di autenticazione è stata aggiunta al protocollo RIP nella sua versione 2). Successivamente è stata aggiunta anche una **autenticazione con cifratura robusta**, del tipo descritto nella Sezione 8.2.1.
- Ulteriore gerarchia.** La gerarchia è uno degli strumenti fondamentali da usare per rendere più scalabili i sistemi. Il protocollo OSPF introduce un ulteriore livello gerarchico nell'intradamento, consentendo la suddivisione di un dominio in **area**. Ciò significa che un-router all'interno di un dominio non deve necessariamente conoscere come raggiungere ciascuna rete all'interno del dominio: può essere sufficiente sapere soltanto come raggiungere l'area giusta. In questo modo si assiste ad una riduzione della quantità di informazioni che devono essere trasmesse e memorizzate da ciascun nodo. Esaminemone in dettaglio le aree nella Sezione 4.3.4.
- Bilanciamento del carico.** Il protocollo OSPF consente che a più percorsi diretti verso la medesima destinazione venga assegnato lo stesso costo, distribuendo in maniera paritetica il traffico.

Esistono parecchi tipi diversi di messaggi OSPF, ma tutti iniziano con la medesima intestazione, mostrata in Figura 4.20. Il campo **Version** ha attualmente il valore 2, mentre il campo **Type** può assumere valori compresi tra 1 e 5. Il campo **SourceAddr** identifica il mittente del messaggio e il campo **AreaId** è un identificativo a 32 bit per l'area in cui si trova il nodo. L'intero pacchetto, con l'esclusione dei dati di autenticazione, è protetto da una somma di controllo (**Checksum**) a 16 bit, usando lo stesso algoritmo dell'intestazione IP (si veda la Sezione 2.4). Il campo **Authentication type** vale 0 se non viene usata alcuna autenticazione, altrimenti può valere 1 (imponendo l'utilizzazione di una semplice password) oppure 2 (che sta ad indicare l'uso di un checksum di autenticazione critografica, del tipo di quello descrit-

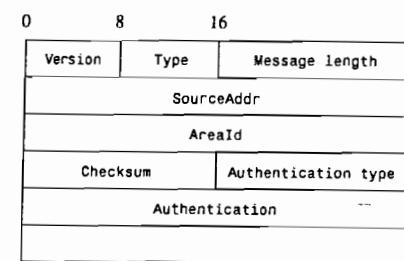


Figura 4.20 Formato dell'intestazione dei messaggi OSPF.

to nella Sezione 8.2.1). In questi ultimi casi, il campo *Authentication type* contiene la password oppure il checksum crittografico.

Tra i cinque tipi di messaggi OSPF, quello di tipo 1 è il messaggio "hello", che viene inviato da un router ai suoi vicini per segnalare il proprio corretto stato di funzionamento e connessione, come descritto in precedenza. Gli altri tipi di messaggio vengono usati per richiedere, inviare e confermare la ricezione di messaggi sullo stato delle linee. Il blocco elementare che costituisce i messaggi del protocollo OSPF viene chiamato **LSA** (*link-state advertisement*, segnalazione dello stato di una linea) e ne daremo qui alcuni dettagli. Un messaggio può contenere molti LSA.

Come qualsiasi protocollo di instradamento di una internetwork, il protocollo OSPF deve fornire informazioni in merito a come raggiungere le reti, per cui OSPF deve fornire un po' di informazioni in più rispetto al semplice protocollo basato sui grafi che abbiamo descritto in precedenza. In particolare, un router che esegue OSPF può generare pacchetti (relativi allo stato delle linee) che notificano come raggiungibili una o più delle reti direttamente connesse a quel router. Inoltre, un router che sia connesso ad un altro router mediante una linea, deve dare informazioni in merito al costo di utilizzo di tale linea per raggiungere quel router. Questi due tipi di informazioni sono necessari per consentire ai router di un dominio di determinare il costo necessario per raggiungere qualsiasi rete interna a quel dominio, nonché il corretto next hop per ciascuna rete.

La Figura 4.21 mostra il formato del pacchetto per un'informazione "di tipo 1" sullo stato della linea. I messaggi LSA di tipo 1 informano sul costo delle linee di connessione fra router, mentre quelli di tipo 2 vengono usati per indicare quali reti siano connesse ad un router; infine, gli altri tipi si usano per fornire supporto alla gerarchia addizionale, come descritto nella sezione successiva. Molti campi di questi LSA dovrebbero risultare familiari dalla discussione precedente; il campo LS Age ha lo stesso significato del campo TTL, anche se il suo valore viene incrementato e l'informazione viene eliminata quando la sua età raggiunge un valore massimo predefinito. Il campo Type con il valore 1 indica che si tratta di un LSA di tipo 1.

In un LSA di tipo 1, i campi *Link-state ID* e *Advertising router* sono identici: ciascuno contiene un identificativo a 32 bit del router che ha generato l'informazione. Anche se si possono usare diverse strategie per assegnare tale identificativo, è essenziale che sia univoco all'interno del dominio e che un certo router usi coerentemente sempre il medesimo identificativo. Un modo per scegliere un identificativo che soddisfi questi requisiti potrebbe essere quello di

LS Age	Options	Type=1
<i>Link-state ID</i>		
<i>Advertising router</i>		
LS sequence number		
LS checksum	Length	
0 Flags	0	Number of links
<i>Link ID</i>		
<i>Link data</i>		
Link type	Num_TOS	Metric
Optional TOS information		
More links		

Figura 4.21 Informazione sullo stato di una linea in OSPF.

4.2 Instradamento (routing)

prendere l'indirizzo IP di valore inferiore tra quelli assegnati al router (ricordate che un router può avere un diverso indirizzo IP su ciascuna interfaccia).

Il campo *LS sequence number* è usato proprio nel modo descritto in precedenza, per identificare LSA obsoleti o duplicati. Il campo *LS checksum* è simile ad altri che abbiamo visto nella Sezione 2.4 e in altri protocolli e viene, ovviamente, utilizzato per verificare che i dati non siano stati corrotti; considera tutti i campi del pacchetto, con la sola esclusione del campo *LS Age*: in questo modo, non è necessario ricalcolare il valore della somma di controllo ogni volta che tale campo viene incrementato. Il campo *Length* contiene la lunghezza, in byte, dell'intero LSA.

Passiamo ora alla reale informazione relativa allo stato delle linee, che è un po' complicata per la presenza dell'informazione sul tipo di servizio (*type of service*, TOS). Soprattutto per il momento su questa informazione, ciascuna linea di collegamento presente in un LSA è rappresentata da un identificativo *Link ID*, da alcuni dati denominati *Link data* e da una metrica, *Metric*. I primi due campi identificano la linea: un modo semplice per fare ciò consiste nell'utilizzare come *Link ID* l'identificativo del router presente all'altra estremità della linea, usando il campo *Link data* per risolvere l'ambiguità derivante dalla eventuale presenza di più linee in parallelo. Il campo *Metric*, ovviamente, rappresenta il costo associato alla linea, mentre il campo *Type* fornisce informazioni sulla linea (ad esempio, se sia una linea di collegamento punto-punto).

L'informazione sul tipo di servizio (TOS) è presente per consentire al protocollo OSPF di scegliere percorsi diversi per i pacchetti IP sulla base del valore presente nel loro campo TOS. Invece di assegnare un singolo costo ad una linea, è possibile assegnare costi diversi in relazione al valore di TOS dei dati. Ad esempio, se avessimo nella rete una linea particolarmente adatta al traffico che presenta elevata sensibilità ai ritardi, potremmo assegnarle un basso costo per i valori di TOS che rappresenta un basso ritardo ed un costo elevato per qualsiasi altro tipo di servizio: in questo modo il protocollo OSPF sceglierbbe un percorso diverso per quei pacchetti aventi il campo TOS contenente quel particolare valore. Vale la pena di sottolineare che, quando viene scritto questo libro, tale caratteristica non è ancora molto diffusa⁵.

4.2.4 Metriche

La precedente discussione è basata sull'ipotesi che i costi (o metriche) delle linee siano noti nel momento in cui viene eseguito l'algoritmo di instradamento. In questa sezione prendiamo in esame alcune strategie che si sono dimostrate efficienti nella pratica e che vengono utilizzate per calcolare i costi da associare alle linee. Un esempio che abbiamo già visto, abbastanza ragionevole e molto semplice, consiste nell'assegnare un costo unitario a tutte le linee: il percorso di costo minimo sarà quello con il minor numero di salti. Questo approccio, però, ha diversi svantaggi. Innanzitutto, non distingue le linee in base alla loro latenza, per cui una linea di collegamento satellitare con una latenza di 250 ms sembra utilizzabile dal protocollo di instradamento allo stesso modo di una linea di collegamento terrestre con una latenza di 1 ms. Secondariamente, non si distinguono i percorsi in base alla capacità, considerando una linea a 9.6 Kbps equivalente ad una linea a 45 Mbps. Infine, le linee non vengono distinte in base al loro carico istantaneo, rendendo impossibile l'identificazione di percorsi

⁵ Notate anche che il significato del campo TOS è cambiato da quando furono scritte le specifiche del protocollo OSPF: parleremo di questo argomento nella Sezione 6.5.3.

che aggirino le linee sovraccaricate. Dal punto di vista pratico, quest'ultimo problema è il più importante, perché si vorrebbe riassumere le caratteristiche complesse e dinamiche di una linea mediante un unico valore di costo.

La rete ARPANET fu il campo di prova per molti diversi approcci al calcolo delle metriche per le linee di collegamento (fu anche la rete in cui venne dimostrata, sul campo, la migliore stabilità dell'instradamento a stato delle linee rispetto a quello a vettore di distanza; in origine venne usato il vettore di distanza, mentre la versione seguente usò lo stato delle linee). La discussione che segue traccia l'evoluzione della metrica usata nell'instradamento in ARPANET e, nel fare ciò, delinea i dettagli del problema.

La metrica usata originariamente in ARPANET misurava il numero di pacchetti che si trovavano in coda in attesa di essere trasmessi da ciascuna linea, in modo che una linea con 10 pacchetti in coda avesse un costo maggiore di una linea con 5 soli pacchetti nella coda di trasmissione. Tuttavia, l'uso della lunghezza della coda come metrica di instradamento non funzionò bene, perché la lunghezza della coda è una misura artificiosa del carico: tende ad inoltrare i pacchetti verso la coda più breve anziché verso la destinazione, un fenomeno assai familiare a coloro che saltano da una coda all'altra all'uscita del supermercato. Più precisamente, il meccanismo di instradamento usato originariamente in ARPANET aveva il problema di non considerare l'ampiezza di banda né la latenza delle linee di collegamento.

Una seconda versione dell'algoritmo di instradamento di ARPANET, detto a volte "nuovo meccanismo di instradamento" (*new routing mechanism*), prendeva in considerazione sia l'ampiezza di banda sia la latenza e usava il ritardo, anziché la lunghezza della coda, come misura del carico, nel modo seguente: ciascun pacchetto veniva contrassegnato con l'istante di arrivo nel router (*ArrivalTime*), memorizzando anche la sua partenza dal router (*DepartTime*): nel momento in cui veniva ricevuto il pacchetto ACK nello strato di collegamento, il nodo calcolava il ritardo (*Delay*) per quel pacchetto:

$$\text{Delay} = (\text{DepartTime} - \text{ArrivalTime}) + \text{TransmissionTime} + \text{Latency}$$

dove *TransmissionTime* e *Latency* erano predefiniti e costanti per la linea, con l'intento di rappresentarne, rispettivamente, l'ampiezza di banda e la latenza. Notate che, in questo caso, *DepartTime* - *ArrivalTime* rappresenta il ritardo subito dal pacchetto mentre si trovava in coda all'interno del nodo a causa del traffico. Se non dovesse arrivare la conferma (ACK) e, quindi, il pacchetto scadesse, il valore di *DepartTime* verrebbe reimpostato al momento in cui il pacchetto viene *rtrasmesso*. In questo caso, *DepartTime* - *ArrivalTime* rappresenta l'affidabilità della linea: più sono frequenti le rtrasmissioni di pacchetti, meno è affidabile la linea e più vogliamo evitarla. Infine, il peso assegnato a ciascuna linea era ottenuto dal ritardo medio subito dai pacchetti recentemente inviati lungo tale linea.

Pur costituendo un miglioramento rispetto al meccanismo originale, questo approccio aveva ancora parecchi problemi. In caso di poco traffico funzionava ragionevolmente bene, perché i due fattori di ritardo costanti dominano il costo, ma in caso di traffico elevato una linea congestionata inizia ad avere un costo molto elevato, provocando così lo spostamento del traffico verso altre linee, fino a divenire quasi inutilizzata, per poi assumere un costo basso ed attrarre così di nuovo tutto il traffico, e così via. L'effetto di questa instabilità era tale che, in caso di traffico elevato, vi erano molte linee quasi inutilizzate per lungo tempo, che è l'ultima cosa che vogliamo accada in condizioni di forte traffico.

Un altro problema era dovuto ad un intervallo di valori possibili troppo elevato. Ad esempio, un linea a 9.6 Kbps con traffico elevato poteva avere un costo 127 volte maggiore

di quello di una linea a 56 Kbps con poco traffico. Di conseguenza, l'algoritmo di instradamento sceglierrebbe un percorso con 126 salti costituito da linee a 56 Kbps con poco traffico piuttosto di una sola linea a 9.6 Kbps. Anche se sottrarre traffico ad una linea sovraccaricata è una buona idea, renderla tanto costosa da farla perdere tutto il traffico è una misura eccessiva: usare 126 salti quando ne sarebbe sufficiente uno soltanto costituisce, in generale, un pessimo utilizzo delle risorse di rete. Inoltre, le linee satellitari risultavano inutilmente penalizzate, al punto che una linea satellitare a 56 Kbps priva di traffico aveva un costo sempre maggiore di una linea terrestre a 9.6 Kbps, anch'essa priva di traffico, anche se la prima avrebbe fornito prestazioni migliori per applicazioni ad elevata ampiezza di banda.

Un terzo approccio, denominato "metrica di instradamento di ARPANET rivista" (*revised ARPANET routing metric*), affrontò questi problemi. Le modifiche più importanti furono dirette ad una significativa compressione della dinamica dei valori di costo, alla considerazione del tipo di linea e ad appianare nel tempo le modifiche del costo.

La diminuzione della velocità con cui si modificano i costi si ottenne con vari meccanismi. Innanzitutto, la misura del ritardo venne trasformata nell'utilizzo della linea e tale numero veniva mediato con l'utilizzazione precedente, per eliminare i cambiamenti improvvisi. Inoltre, si impose un limite molto rigido alla variazione massima del costo da un ciclo di misurazione al successivo. Rendendo più lenti i cambiamenti di costo, venne assai ridotta la probabilità che tutti i nodi abbandonassero contemporaneamente un percorso.

La compressione della dinamica dei valori di costo fu ottenuta usando l'utilizzazione misurata sulla linea, il tipo di linea e la sua velocità, come parametri di una funzione che è rappresentata in Figura 4.22. Osservate che:

- Una linea di collegamento con traffico elevato non ha mai un costo superiore a tre volte il proprio costo in condizioni di assenza di traffico.
- La linea con costo maggiore è soltanto sette volte più costosa di quella con costo minore.

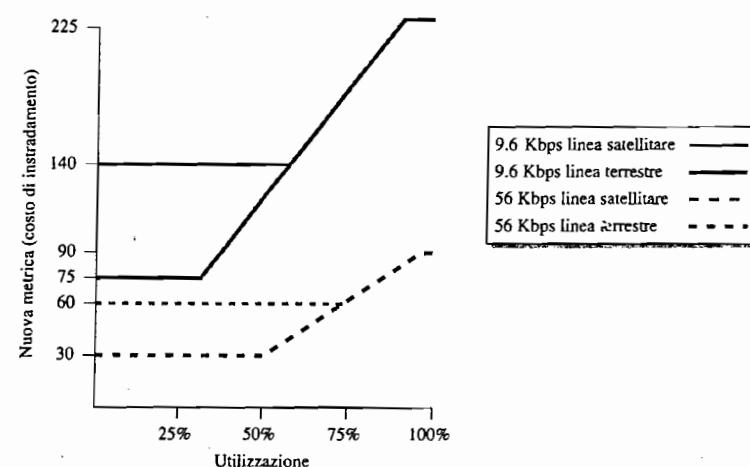


Figura 4.22 La metrica rivista dell'instradamento di ARPANET in funzione dell'utilizzazione della linea.

- Una linea di collegamento satellitare ad alta velocità è meno costosa di una linea terrestre a bassa velocità.
- Il costo è una funzione dell'utilizzazione della linea soltanto per valori di traffico abbastanza elevati.

Tutti questi fattori hanno come conseguenza che l'abbandono generalizzato di una linea diventa molto meno probabile, perché una triplicazione del costo rende probabilmente la linea poco interessante per alcuni percorsi, ma le può consentire di rimanere la scelta migliore per altri. Si giunse a definire le pendenze, i valori iniziali e i valori di cambiamento di pendenza delle curve di Figura 4.22 dopo molti tentativi e ci fu un intenso lavoro di ottimizzazione per fornire buone prestazioni.

C'è, infine, un ultimo argomento legato al calcolo dei pesi dei rami: la frequenza con cui ciascun nodo calcola i pesi delle proprie linee. A questo proposito bisogna tenere presente due fattori: quando un nodo sta misurando la lunghezza di una coda, un ritardo o un'utilizzazione, sta sempre facendo una media su un periodo di tempo; il fatto che un valore di costo si modifichi, non implica che il nodo invii un messaggio di aggiornamento. In pratica, gli aggiornamenti vengono inviati soltanto quando la modifica del peso di un ramo è maggiore di una certa soglia.

4.2.5 Instradamento per host mobili

Riesaminando la precedente discussione relativa al funzionamento dell'indirizzamento IP e dell'instradamento, noterete che viene fatta un'ipotesi implicita in merito alla mobilità degli host, o, per meglio dire, viene ipotizzata la loro immobilità. L'indirizzo di un host è composto da un numero di rete e da un numero di host, ed il numero di rete ci dice quale sia la rete a cui l'host è connesso. Gli algoritmi di instradamento IP indicano ai router come far giungere i pacchetti alla rete di destinazione corretta, migliorando così la scalabilità dell'instradamento, consentendo ai router di non considerare le informazioni specifiche relative al singolo host. Quindi, cosa accadrebbe se un host venisse disconnesso da una rete e connesso ad un'altra rete? Se non modifichiamo l'indirizzo IP dell'host, esso diverrebbe irraggiungibile. Qualsiasi pacchetto inviato a tale host verrebbe inviato alla rete avete il numero di rete corretto, ma nel momento in cui i router di tale rete cercassero di consegnare il pacchetto all'host, l'host non si troverebbe nella rete per riceverlo.

Un'ovvia soluzione a questo problema consiste nel fornire all'host un nuovo indirizzo quando esso viene connesso ad una nuova rete, e questo processo può essere reso relativamente semplice da tecniche quali il DHCP (descritto nella Sezione 4.1.6): in molte situazioni questa soluzione è adeguata, ma in altre non lo è. Ad esempio, supponete che l'utente di un PC equipaggiato con un'interfaccia di rete wireless stia eseguendo un'applicazione mentre si sposta attraverso il Paese. Il PC è in grado di disconnettersi da una rete e collegarsi ad un'altra rete che funziona sulla medesima frequenza, ma l'utente vorrebbe disinteressarsi di questo. In particolare, l'applicazione in esecuzione quando il PC era connesso alla rete A dovrebbe continuare a funzionare senza interruzioni quando il PC si connette alla rete B. Se, semplicemente, il PC modificasse il proprio indirizzo IP durante l'esecuzione dell'applicazione, l'applicazione, altrettanto semplicemente, non potrebbe funzionare, perché l'entità remota non ha modo di sapere che da un certo momento in poi i pacchetti vanno inviati ad un nuovo indirizzo IP. La cosa ideale è che il movimento del PC sia del tutto trasparente per l'applicazione remota. Le procedure che sono state progettate per risolvere questo problema

Controllare il comportamento dell'instradamento

Vista la complessità dell'instradamento di pacchetti in una rete delle dimensioni di Internet, ci potremmo chiedere quanto il sistema funzioni bene. Sappiamo che a volte funziona certamente, perché siamo in grado di connetterci con siti distribuiti in tutto il mondo. Abbiamo, tuttavia, il sospetto che non funzioni costantemente, perché a volte non siamo in grado di connetterci ad alcuni siti. Il vero problema consiste nel determinare quale parte del sistema non funziona quando le nostre connessioni falliscono: qualche apparato di instradamento non ha funzionato correttamente, il server remoto è sovraccarico, oppure semplicemente qualche linea di collegamento o qualche macchina è guasta?

Si tratta di un vero e proprio problema di gestione della rete: nonostante esistano strumenti usati dagli amministratori di rete per tenere sotto controllo le reti di loro competenza (ad esempio, il protocollo Simple Network Management Protocol, SNMP, descritto nella Sezione 9.2.3), si tratta di un problema sostanzialmente irrisolto per la rete Internet nel suo complesso, che è cresciuta fino ad avere una struttura così complessa che, pur essendo costituita da un insieme di parti costruite dall'uomo e con comportamento in massima parte deterministico, siamo portati a considerarla quasi come se fosse un organismo vivente o un fenomeno naturale da studiare. In sostanza, cerchiamo di capire il comportamento dinamico di Internet effettuando esperimenti in essa e proponendo modelli che rendano conto delle nostre osservazioni.

Un eccellente esempio di questo tipo di studi è stato condotto da Vern Paxson, che nel 1995 usò lo strumento traceroute di Unix per studiare 40000 percorsi fra 37 siti Internet, nel tentativo di rispondere a domande quali: perché un percorso non funziona? quanto sono stabili i percorsi nel tempo? i percorsi sono simmetrici oppure no? Fra le altre cose, Paxson trovò che la probabilità che un utente vada incontro ad un serio problema di instradamento era di 1 su 30, e che tali problemi avevano una durata di circa 30 secondi. Scoprì anche che due terzi dei percorsi nella rete Internet erano stabili per giorni o settimane, e che circa un terzo delle volte il percorso usato per andare dall'host A all'host B comprendeva almeno un dominio di instradamento diverso dal percorso usato per andare dall'host B all'host A. La conclusione finale di Paxson fu che l'instradamento in Internet stava diventando sempre meno prevedibile nel tempo.

vanno solitamente sotto il nome di "Mobile IP", che è anche il nome del gruppo di lavoro di IETF che le ha definite.

Il gruppo di lavoro che si è occupato di Mobile IP ha preso alcune decisioni fondamentali, fin dall'inizio. In particolare, venne posto il requisito essenziale che la soluzione dovesse funzionare senza alcuna modifica nel software degli host non mobili e della maggioranza dei router di Internet, un tipo di approccio usato molto frequentemente nella rete Internet: qualsiasi nuova tecnologia che, per poter funzionare, richieda la modifica della maggior parte dei router o degli host, deve molto probabilmente affrontare una battaglia durissima prima di essere accettata.

Nonostante la maggioranza dei router possa rimanere immutata, per fornire il supporto alla mobilità è necessaria qualche nuova funzionalità in almeno un router, che prende il nome di *home agent* (agente residenziale) del nodo mobile e che si trova nella rete "di residenza" (*home network*) dell'host mobile. L'host mobile deve avere un indirizzo IP fisso, il suo *indirizzo di residenza* (*home address*), con numero di rete uguale a quello della home network e, quindi, anche uguale a quello dell'*home agent*. Tale indirizzo verrà usato dagli altri host

quando devono inviare pacchetti all'host mobile: dato che non viene modificato, può essere usato da applicazioni di lunga durata mentre l'host si sposta.

In molti casi è anche richiesto un secondo router dotato di funzionalità aggiuntive, il *foreign agent* (agente esterno), collocato nella rete a cui si connette l'host mobile quando non si trova nella propria rete di residenza. Per prima cosa prenderemo in esame il funzionamento del Mobile IP quando viene usato il foreign agent: la Figura 4.23 mostra una rete con l'agente residenziale e l'agente esterno.

Entrambi gli agenti, residenziale ed esterno, rendono pubblica periodicamente la propria presenza nelle reti a cui sono direttamente connessi, usando messaggi di notifica degli agenti (*agent advertisement message*). Un host mobile può anche richiedere una notifica quando si connette ad una nuova rete. Il messaggio di notifica inviato dall'agente residenziale consente ad un host mobile di apprendere l'indirizzo del proprio agente residenziale prima di abbandonare la propria rete residenziale. Quando l'host mobile si connette ad una rete esterna, si pone in ascolto di messaggi di notifica da parte di un agente esterno e si registra presso di esso, fornendo l'indirizzo del proprio agente residenziale. Quindi, l'agente esterno contatta l'agente residenziale, fornendo un *care-of address* ("indirizzo di gestione"), che è solitamente l'indirizzo IP dell'agente esterno.

A questo punto, ogni host che tenti di inviare un pacchetto all'host mobile lo invierà con un indirizzo di destinazione uguale all'indirizzo residenziale di tale nodo. L'inoltro normale del protocollo IP farà arrivare tale pacchetto alla rete di residenza del nodo mobile, dove è in attesa l'agente residenziale. Possiamo quindi suddividere in tre parti il problema di consegnare il pacchetto al nodo mobile:

1. Come fa l'agente residenziale ad intercettare un pacchetto destinato al nodo mobile?
2. Come fa, poi, l'agente residenziale a consegnare il pacchetto all'agente esterno?
3. Come fa l'agente esterno a consegnare il pacchetto all'nodo mobile?

Il primo problema potrebbe sembrare semplice se si guarda soltanto alla Figura 4.23, in cui l'agente residenziale si trova chiaramente sull'unico percorso possibile tra l'host mittente e la rete residenziale, per cui deve ricevere tutti i pacchetti destinati al nodo mobile. Ma cosa succederebbe se il nodo mittente si trovasse sulla rete 10, oppure se ci fosse un altro router connesso alla rete 10 che tentasse di consegnare il pacchetto senza passare per l'agente residenziale? Per risolvere questo problema, l'agente residenziale assume effettivamente le sembianze del nodo mobile, usando una tecnica denominata "proxy ARP", che funziona proprio come il protocollo ARP descritto nella Sezione 4.1.5, tranne per il fatto che l'agente residenziale inserisce l'indirizzo IP del nodo mobile, invece del proprio, nei messaggi ARP. Usa a tale

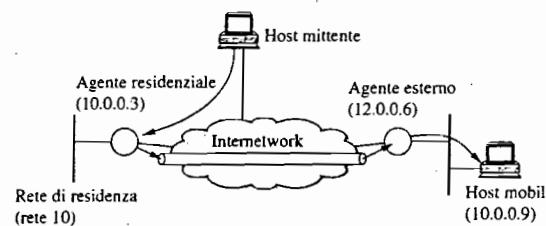


Figura 4.23 Host mobile e agenti di mobilità.

4.2 Instradamento (routing)

scopo il proprio indirizzo hardware, in modo che tutti i nodi appartenenti alla stessa rete imparino l'associazione tra l'indirizzo hardware dell'agente residenziale e l'indirizzo IP del nodo mobile. Un dettaglio di questo processo riguarda il fatto che le informazioni del protocollo ARP possono essere conservate per qualche tempo nella memoria di altri nodi della rete: per essere certo che questi dati vengano eliminati in modo tempestivo, l'agente residenziale invia un messaggio ARP non appena il nodo mobile si registra presso un agente esterno. Dato che questo messaggio ARP non è una risposta ad una normale richiesta ARP, viene detto "ARP non richiesto" (*gratuitous ARP*).

Il secondo problema riguarda la consegna del pacchetto intercettato all'agente esterno: si usa la tecnica del tunneling descritta nella Sezione 4.1.8. L'agente residenziale "avvolge" semplicemente il pacchetto con un'intestazione IP avente come destinazione l'agente esterno e lo trasmette all'interno della internetwork: tutti i router intermedi vedono solamente un pacchetto IP destinato all'indirizzo IP dell'agente esterno, come se si fosse instaurato un tunnel IP fra l'agente residenziale e l'agente esterno, tramite il quale l'agente residenziale consegna i pacchetti destinati al nodo mobile.

Quando un pacchetto arriva finalmente all'agente esterno, esso elimina l'intestazione IP che era stata aggiunta e trova all'interno un pacchetto IP destinato al nodo mobile. Ovviamente, l'agente esterno non può trattare questo pacchetto come un normale pacchetto IP, perché così facendo lo invierebbe di nuovo alla rete di residenza; al contrario, deve accorgersi che l'indirizzo appartiene ad un nodo mobile che si è registrato, e consegnare il pacchetto all'indirizzo *hardware* del nodo mobile (ad esempio, il suo indirizzo Ethernet), che ha appreso durante il processo di registrazione.

Un'osservazione che si può fare in merito a queste procedure riguarda la possibilità che l'agente esterno e il nodo mobile siano, in realtà, lo stesso calcolatore, cioè che un nodo mobile possa svolgere anche la funzione di agente esterno per se stesso. Perché ciò funzioni, però, il nodo mobile deve essere in grado di acquisire dinamicamente un indirizzo IP all'interno dello spazio di indirizzamento della rete esterna, da usare come indirizzo di gestione (*care-of address*): nel nostro esempio, quest'ultimo indirizzo apparirebbe alla rete 12. Abbiamo già visto una modalità mediante la quale un host può acquisire dinamicamente un indirizzo IP corretto, usando il protocollo DHCP (Sezione 4.1.6). Questo approccio ha la positiva caratteristica di consentire ai nodi mobili la connessione a reti private residenziali e di un po' di software nel nodo mobile (nell'ipotesi che la rete esterna usi il protocollo DHCP).

Cosa succede per il traffico nell'altra direzione, cioè dal nodo mobile a nodi fissi? Questo è molto più semplice. Il nodo mobile inserisce, semplicemente, l'indirizzo IP del nodo fisso nel campo destinazione dei suoi pacchetti IP, inserendo invece come indirizzo del mittente il proprio indirizzo residenziale permanente, dopodiché i pacchetti vengono inoltrati al nodo fisso nel modo usuale. Ovviamente, se entrambi i nodi coinvolti in un dialogo fossero mobili, le procedure appena descritte avrebbero luogo in entrambe le direzioni.

Ottimizzazione dei percorsi nel Mobile IP

L'approccio precedente ha uno svantaggio significativo, che può risultare familiare agli utenti di telefoni cellulari. Il percorso dal nodo mittente al nodo mobile può essere fortemente diverso dal percorso ottimo. Uno degli esempi più estremi si ha quando un nodo mobile e il nodo mittente si trovano nella stessa rete, ma la rete di residenza del nodo mobile è dall'altra

parte di Internet. Il nodo mittente invia tutti i pacchetti alla rete di residenza, questi attraversano Internet per giungere all'agente residenziale, che li invia all'agente esterno tramite il tunnel, di nuovo attraversando Internet. Sarebbe ovviamente meglio che il nodo mittente potesse accorgersi che il nodo mobile si trova in realtà all'interno della sua stessa rete, consegnando il pacchetto direttamente. Nel caso più generale, l'obiettivo è quello di consegnare i pacchetti nel modo più diretto possibile, dal nodo mittente al nodo mobile, senza passare per l'agente residenziale. A volte ci si riferisce a questo come al "problema della triangolazione nell'instradamento", perché il percorso dal mittente al nodo mobile attraverso l'agente residenziale si svolge lungo due lati di un triangolo, anziché lungo il terzo lato, che rappresenta il percorso diretto.

L'idea che sta alla base della soluzione dell'instradamento triangolare consiste nel far sapere al nodo mittente l'indirizzo di gestione del nodo mobile: il nodo mittente può così creare un proprio tunnel verso l'agente esterno. Ciò viene trattato come un'ottimizzazione del processo appena descritto: se il mittente è equipaggiato con l'apposito software, necessario ad apprendere l'indirizzo di gestione e a creare un proprio tunnel, allora il percorso può essere ottimizzato, altrimenti i pacchetti seguono, semplicemente, il percorso non ottimale.

Quando un agente residenziale vede un pacchetto destinato ad uno dei nodi mobili a cui fornisce supporto, deduce che il mittente non sta usando il percorso ottimale, per cui restituisce al mittente un messaggio di "aggiornamento di associazione" (*binding update*), oltre ad inoltrare il pacchetto di dati all'agente esterno. Il mittente, se è in grado di farlo, aggiunge tale informazione ad un elenco di associazioni tra nodi mobili e indirizzi di gestione (*binding cache*). La prossima volta che tale mittente avrà un pacchetto da inviare a quel nodo mobile, cercherà l'associazione nella *cache* e potrà inviare direttamente all'agente esterno il pacchetto, per mezzo di un tunnel.

Questo schema presenta un problema ovvio: le informazioni memorizzate nella binding cache possono divenire obsolete e scorrette, qualora l'host mobile si sposti in una diversa rete. Se viene usata un'informazione obsoleta, l'agente esterno riceverà tramite un tunnel i pacchetti destinati ad un nodo mobile che non è più registrato all'interno della propria rete: in tal caso, restituirà al mittente un messaggio di "problema nell'associazione" (*binding warning*), per indicare che la relativa informazione presente nella cache va eliminata. Questo schema, però, funziona solamente nel caso in cui l'agente esterno non sia il nodo mobile stesso. Per questo motivo le informazioni presenti nella cache devono essere eliminate dopo un certo tempo; la durata di questo periodo è specificata nel messaggio di binding update.

L'instradamento per nodi mobili presenta anche interessanti problemi di sicurezza. Ad esempio, un agente ostile che volesse intercettare i pacchetti destinati a qualche altro nodo della internetwork potrebbe contattare l'agente residenziale per tale nodo e proclamarsi agente esterno per esso. È quindi evidente che sia necessario un meccanismo di autenticazione, di cui parleremo nel Capitolo 8.

Infine, vogliamo notare che ci sono molti problemi aperti nelle reti mobili. Ad esempio, gli aspetti di sicurezza e di prestazioni delle reti mobili possono richiedere algoritmi di instradamento che prendano in considerazione parecchi fattori nella ricerca di un percorso verso un host mobile: per fare un esempio, potrebbe essere preferibile trovare un percorso che non attraversi reti di cui non ci si può fidare. C'è poi il problema delle reti mobili "ad hoc", che consentono ad un gruppo di nodi mobili di formare una rete in assenza di nodi fissi. Si tratta di aree in cui la ricerca è ancora attiva.

4.3 La rete Internet globale

A questo punto abbiamo visto come connettere un insieme eterogeneo di reti per creare una internetwork e come usare la semplice gerarchia degli indirizzi IP per rendere in qualche modo scalabile l'instradamento in una interconnessione di reti. Abbiamo detto "in qualche modo" scalabile perché, anche se ciascun router non ha bisogno di avere informazioni relative a tutti gli host connessi alla internetwork, ha comunque bisogno, con il modello visto fino ad ora, di avere informazioni su tutte le reti connesse. L'odierna Internet ha decine di migliaia di reti connesse ad essa, e i protocolli di instradamento di cui abbiamo appena parlato non scalano bene fino a dimensioni di questo genere. Questa sezione esamina diverse tecniche che migliorano significativamente la scalabilità e che hanno consentito ad Internet di crescere fino a questo punto.

Prima di passare a queste tecniche, abbiamo bisogno di crearcì una visione complessiva, ancorché generica, di cosa sia la rete Internet globale, che non è soltanto un'interconnessione casuale di reti Ethernet, ma assume invece una forma che è una diretta conseguenza del fatto che interconnette diverse organizzazioni. La Figura 4.24 mostra una raffigurazione semplificata dello stato di Internet nel 1990; da quel momento la topologia di Internet è diventata molto più complessa di quanto la figura possa suggerire (nella Sezione 4.3.3 e nella Figura 4.29 daremo un'immagine più accurata della Internet odierna), ma per il momento questa figura sarà sufficiente.

Una delle caratteristiche più evidenti di questa topologia riguarda la sua struttura, che consiste di siti "utenti finali" (*end user*), come la Stanford University, che si connettono a reti "fornitori di servizi" (*service provider*), come la rete BARRNET, che era un fornitore di rete nella San Francisco Bay Area. Nel 1990, molti fornitori operavano in una regione geograficamente limitata e erano quindi noti come "reti regionali" (*regional network*), a loro volta connesse tramite un *backbone* nazionale. Nel 1990 questo backbone era finanziato dalla National Science Foundation (NSF) ed era quindi chiamato *NSFNET backbone*. Anche se questa figura non mostra un tale livello di dettaglio, le reti dei fornitori di servizio erano tipicamente costituite da un gran numero di linee di collegamento punto-punto (ad esempio, linee DS3 o OC-3) connesse a router; analogamente, il sito di ciascun utente finale, tipicamente, non è costituito da una singola rete, ma consiste invece di più reti fisiche connesse da router e bridge.

Notate, nella Figura 4.24, che ciascun fornitore e ciascun utente finale sono, molto probabilmente, un'entità di gestione indipendente, cosa che ha alcune conseguenze significative

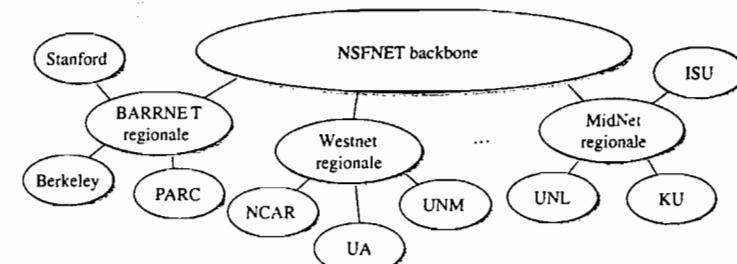


Figura 4.24 La struttura ad albero di Internet nel 1990.

sull'instradamento. Ad esempio, è abbastanza probabile che fornitori diversi abbiano idee diverse su quale sia il miglior protocollo di instradamento da usare all'interno della propria rete e su come assegnare le metriche di costo alle linee di collegamento interne alle proprie reti. A causa di questa indipendenza, la rete di ciascun fornitore è solitamente un singolo *sistema autonomo* (AS, autonomous system), che sarà definito in modo più preciso nella Sezione 4.3.3: per ora, sia sufficiente pensare ad un AS come ad una rete gestita indipendentemente da altri AS.

Il fatto che Internet abbia una struttura identificabile può essere usato a nostro vantaggio mentre affrontiamo il problema della scalabilità, perché, in realtà, siamo di fronte a due problemi di scalabilità tra loro correlati. Il primo riguarda la scalabilità dell'instradamento: dobbiamo trovare il modo di minimizzare la quantità di numeri di rete che vengono fatti circolare dai protocolli di instradamento e memorizzati nelle tabelle di instradamento dei router. Il secondo problema riguarda l'utilizzazione degli indirizzi, per essere sicuri che lo spazio degli indirizzi IP non venga consumato troppo rapidamente.

In tutta questa sezione vedremo usato più volte il principio della gerarchia per migliorare la scalabilità. Cominceremo con la suddivisione in sottoreti, che si occupa principalmente dell'utilizzazione dello spazio di indirizzamento, poi presenteremo l'instradamento senza classi (o supernetting), che affronta sia il problema dell'utilizzazione degli indirizzi, sia quello della scalabilità dell'instradamento. Successivamente, vedremo come si possa usare la gerarchia per migliorare la scalabilità dell'instradamento, sia mediante l'instradamento interdominio, sia all'interno di un singolo dominio. Il paragrafo conclusivo esamina gli standard emergenti per la versione 6 del protocollo IP, la cui progettazione è stata sostanzialmente necessaria per problemi di scalabilità.

4.3.1 Le sottoreti

L'obiettivo originale degli indirizzi IP era quello di identificare univocamente ciascuna rete fisica mediante la parte di indirizzo dedicata al numero di rete, ma si è visto che questo approccio soffre di un paio di svantaggi. Immaginate una grande struttura universitaria con molte reti al proprio interno, che decida di connettersi ad Internet: serve almeno un indirizzo di rete di classe C per ogni rete, indipendentemente dalle sue dimensioni, mentre per ogni rete avente più di 255 host serve addirittura un indirizzo di classe B. Questo potrebbe non sembrare un grande problema, e in realtà non lo era quando fu concepita Internet, ma i numeri di rete sono in numero finito e gli indirizzi di classe B sono disponibili in numero assai più limitato rispetto agli indirizzi di classe C. Gli indirizzi di classe B tendono ad essere molto richiesti, perché è difficile prevedere se una rete si espanderà fino ad avere più di 255 host oppure no, per cui è più semplice usare un indirizzo di classe B fin dall'inizio anziché rinumerare tutti gli host quando si esaurisce lo spazio in una rete di classe C. Il problema che stiamo osservando è l'inefficienza dell'assegnamento degli indirizzi: una rete con due soli nodi usa un'intera rete di classe C, sprecando così 253 indirizzi perfettamente utilizzabili; una rete di classe B con poco più di 255 host spreca più di 64000 indirizzi.

Assegnando un numero di rete per ciascuna rete fisica, quindi, si esaurisce lo spazio degli indirizzi IP in modo più veloce di quanto vorremmo. Mentre vorremmo poter connettere più di 4 miliardi di host prima di esaurire tutti gli indirizzi validi, ci basta connettere 2¹⁴ (circa 16000) reti di classe B per esaurire la parte relativa all'indirizzo di rete. Vorremmo quindi trovare il modo di usare i numeri di rete in modo molto più efficiente.

4.3 La rete Internet globale

Assegnare molti numeri di rete ha anche un altro svantaggio che diventa evidente quando si pensa all'instradamento. Ricordate che la quantità di informazioni di stato che deve essere memorizzata in un nodo che partecipa ad un protocollo di instradamento è proporzionale al numero di altri nodi, e che l'instradamento in una internetwork consiste nella costruzione di tabelle di inoltro che dicono ad un router come raggiungere reti diverse. Di conseguenza, più numeri di rete vengono utilizzati, più grandi diventano le tabelle di inoltro. Tabelle di inoltro di grandi dimensioni rendono più costosi i router e rendono potenzialmente più lente le ricerche al loro interno rispetto a tabelle di dimensioni minori degradando così le prestazioni dei router. Ciò fornisce un'altra motivazione per assegnare con cura i numeri di rete.

La suddivisione in sottoreti (*subnetting*) fornisce un modo semplice ed elegante per ridurre la quantità totale di numeri di rete che vengono assegnati. L'idea consiste nel prendere un singolo numero di rete IP ed assegnare gli indirizzi IP con quel numero di rete a diverse reti fisiche; che vengono poi chiamate *sottoreti* (*subnet*). Per far funzionare questo schema bisogna fare diverse cose. Innanzitutto, le sottoreti devono essere vicine l'una all'altra, perché per un punto lontano in Internet esse assumeranno l'aspetto di un'unica rete, avendo un unico numero di rete; ciò significa che un router sarà in grado soltanto di scegliere un percorso che porta ad una qualsiasi delle sottoreti, che quindi è meglio siano localizzate nella stessa direzione. Una condizione ideale in cui usare le sottoreti è una grande struttura universitaria o aziendale che abbia molte reti fisiche: dall'esterno, tutto ciò che avete bisogno di sapere per raggiungere una qualsiasi sottorete interna è il punto in cui la struttura globale si connette al resto di Internet. Questo è solitamente un unico punto, per cui sarà sufficiente un'unica riga nella tabella di inoltro; anche se vi fossero più punti di connessione con Internet, la conoscenza di uno solo di questi è un buon punto di partenza.

Il meccanismo mediante il quale un singolo numero di rete può essere condiviso da più reti richiede la configurazione di tutti i nodi di ciascuna sottorete mediante una *maschera di sottorete* (*subnet mask*). Quando si usano gli indirizzi IP semplici, tutti gli host che si trovano nella stessa rete devono avere lo stesso numero di rete: la maschera di sottorete ci consente di introdurre un *numero di sottorete*. Tutti gli host appartenenti alla stessa rete fisica avranno lo stesso numero di sottorete, mentre host che si trovino in reti fisiche diverse possono condividere un singolo numero di rete.

La maschera di sottorete, quindi, introduce a tutti gli effetti un ulteriore livello di gerarchia negli indirizzi IP. Ad esempio, supponete di voler condividere un singolo indirizzo di classe B fra diverse reti fisiche. Potremmo usare una maschera di sottorete uguale a 255.255.255.0 (le maschere di sottorete vengono scritte con la stessa notazione degli indirizzi IP: questa maschera ha quindi il valore 1 nei primi 24 bit e il valore 0 nei rimanenti 8 bit). In pratica, ciò significa che i primi 24 bit (dove la maschera contiene il valore 1) definiscono ora il numero di rete, mentre gli altri 8 bit (dove la maschera contiene il valore 0) vengono usati per il numero dell'host. Dato che i primi 16 bit identificano la rete come indirizzo di classe B, possiamo ora immaginare l'indirizzo come se fosse composto da tre parti, anziché due: una parte per il numero di rete, una parte per il numero di sottorete e una parte per il numero dell'host. Abbiamo quindi suddiviso la parte dedicata al numero dell'host in una parte di sottorete e una parte di host, come mostrato in Figura 4.25.

Per un host, la gestione delle sottoreti comporta che la sua configurazione contenga ora un indirizzo IP e una maschera di sottorete; per identificare la sottorete a cui è connesso. Ad esempio, l'host H1 della Figura 4.26 è configurato con l'indirizzo 128.96.34.15 e la maschera di sottorete 255.255.255.128 (tutti gli host appartenenti ad una certa sottorete sono configurati

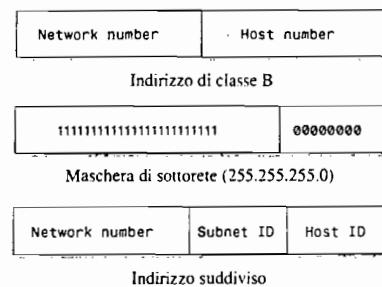


Figura 4.25 Indirizzamento con sottoreti.

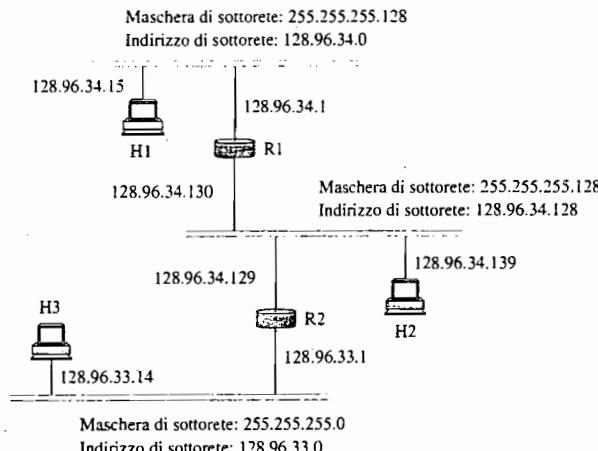


Figura 4.26 Un esempio di suddivisione in sottoreti.

con la medesima maschera, cioè esiste esattamente una maschera per ciascuna sottorete). L'operazione di AND bit a bit di questi due numeri definisce il numero di sottorete dell'host e di tutti gli altri host appartenenti alla stessa sottorete. In questo caso, 128.96.34.15 AND 255.255.255.128 ha come risultato 128.96.34.0, che è quindi il numero di sottorete della sottorete in alto nella figura.

Quando l'host vuole inviare un pacchetto ad un certo indirizzo IP, la prima cosa che fa è l'operazione di AND bit a bit fra la propria maschera di sottorete e l'indirizzo IP di destinazione. Se il risultato è uguale al proprio numero di sottorete, allora il mittente sa che l'host destinatario si trova nella stessa sottorete ed il pacchetto può essere consegnato direttamente sulla sottorete. Se il risultato è diverso, il pacchetto deve essere inviato ad un router per essere inoltrato ad un'altra sottorete. Ad esempio, se H1 sta inviando un pacchetto ad H2, allora H1 esegue l'operazione AND tra la propria maschera di sottorete (255.255.255.128) e l'indirizzo di H2 (128.96.34.139), ottenendo 128.96.34.128 che non è uguale al numero di sottorete

di H1 (128.96.34.0), per cui H1 sa che H2 si trova in una diversa sottorete. Poiché H1 non è in grado di consegnare direttamente il pacchetto sulla propria sottorete, lo invia al proprio router di default, R1.

Notate che il protocollo ARP è sostanzialmente invariante rispetto a questa modifica nella struttura degli indirizzi. Una volta che un host o un router ha capito a quale nodo deve consegnare il pacchetto per raggiungere una delle reti a cui è connesso, se necessario usa ARP per trovare l'indirizzo MAC di tale nodo.

Quando si introduce la suddivisione in sottoreti cambia anche il compito del router. Ricordate che, nel protocollo IP normale, un router usa una tabella di inoltro che contiene dati del tipo **(NetworkNum, NextHop)**. Per gestire le sottoreti, la tabella deve ora contenere dati del tipo **(SubnetNumber, SubnetMask, NextHop)**. Per trovare nella tabella il dato cercato, il router esegue, in successione, l'operazione AND fra l'indirizzo di destinazione del pacchetto e il campo **SubnetMask** di ogni dato: se il risultato corrisponde al campo **SubnetNumber** del dato, allora è stato trovato il dato cercato ed il pacchetto viene inoltrato al router del salto successivo indicato (**NextHop**). Nella rete di esempio presentata nella Figura 4.26, il router R1 avrebbe la tabella mostrata in Tabella 4.10.

Continuando con l'esempio di un datagramma che venga inviato da H1 a H2, R1 eseguirebbe l'operazione AND tra l'indirizzo di H2 (128.96.34.139) e la maschera di sottorete del primo dato (255.255.255.128), confrontando il risultato (128.96.34.128) con il numero di sottorete di quel dato (128.96.34.0). Dato che sono diversi, si procede con il dato successivo, nel qual caso si trova la corrispondenza; quindi, R1 consegna il datagramma a H2 usando l'interfaccia 1, che è l'interfaccia connessa alla rete a cui appartiene H2.

Possiamo ora descrivere l'algoritmo di inoltro dei datagrammi nel modo seguente:

```
D = indirizzo IP di destinazione
per ogni (SubnetNumber, SubnetMask, NextHop) nella tabella di inoltro
    D1 = SubnetMask & D
    se D1 = SubnetNumber
        se NextHop è un'interfaccia
            consegna il datagramma direttamente alla destinazione
        altrimenti
            consegna il datagramma al router NextHop
```

Anche se in questo esempio non viene mostrato, solitamente nella tabella sarà presente un router di default; da usare se non viene trovata nessuna corrispondenza esplicita. Si noti che sarebbe assai inefficiente un'implementazione letterale di questo algoritmo, che richiedesse l'esecuzione ripetuta di operazioni AND tra l'indirizzo di destinazione e una maschera di sottorete (che non è diversa di volta in volta), richiedendo un tempo di ricerca lineare.

Tabella 4.10 Esempio di tabella di inoltro con sottoreti relativa alla Figura 4.26.

Numero di sottorete	Maschera di sottorete	Salto successivo
128.96.34.0	255.255.255.128	Interfaccia 0
128.96.34.128	255.255.255.128	Interfaccia 1
128.96.33.0	255.255.255.0	R2

Vale la pena di soffermarci su alcuni punti relativi alle sottoreti. Abbiamo già visto che la maschera di sottorete non deve essere necessariamente allineata su confini di host (es. 255.255.255.128, costituita da 25 valori uguali a 1 e 7 valori uguali a 0); per di più, non è nemmeno necessario che tutti i valori 1 siano contigui (ad esempio, sarebbe possibile usare una maschera di sottorete uguale a 255.255.1.0). Tutti i meccanismi descritti finora continuerebbero a funzionare, ma non sarebbe più possibile prendere in esame una sezione contigua dell'indirizzo IP e dire "quello è il numero di sottorete", rendendo così la gestione più difficile. Ciò potrebbe addirittura non funzionare con alcune implementazioni che ipotizzano che nessuno usi maschere non contigue, per cui in pratica è sconsigliato farlo.

Possiamo anche usare più sottoreti sulla stessa rete fisica, con l'effetto di costringere host della stessa rete a scambiarsi pacchetti tramite un router, cosa che potrebbe essere utile per la gestione (ad esempio, per garantire l'isolamento fra diversi dipartimenti che condividono una LAN).

Un terzo punto che abbiamo solo accennato riguarda il fatto che parti diverse della internetwork vedono il mondo in modo diverso. Dall'esterno della nostra ipotetica struttura universitaria, i router vedono una singola rete. Nell'esempio visto in precedenza, i router esterni alla struttura vedono l'insieme di reti rappresentato nella Figura 4.26 come la sola rete 128.96 e conserveranno un unico dato nelle proprie tabelle di inoltro per raggiungere tale rete. I router interni alla struttura, invece, devono essere in grado di instradare i pacchetti alla sottorete giusta. Sostanzialmente, diverse parti di una internetwork vedono informazioni di instradamento diverse. La sezione successiva esamina con maggiore dettaglio la propagazione delle informazioni di instradamento all'interno di Internet.

Per riassumere, la suddivisione in sottoreti aiuta a risolvere i problemi di scalabilità in due modi: migliora l'efficienza nell'assegnazione degli indirizzi consentendo di non usare un'intera rete di classe C o di classe B ogni volta che aggiungiamo una nuova rete fisica; ed aiuta ad aggregare le informazioni, nel senso che, a partire da una distanza ragionevole, un complesso insieme di reti fisiche può essere reso assimilabile ad un'unica rete, in modo che possa essere ridotta la quantità di informazioni che i router devono memorizzare per consegnare i datagrammi verso quelle reti.

4.3.2 Instradamento senza classi (CIDR)

L'instradamento interdominio senza classi (*Classless InterDomain Routing*, CIDR, pronunciato come "cider") è una tecnica che risolve due problemi di scalabilità nella rete Internet: la crescita delle tabelle di instradamento per il backbone per effetto della sempre maggiore quantità di numeri di rete che vi si deve memorizzare, e la possibilità che lo spazio di indirizzamento di IP a 32 bit venga esaurito ben prima che quattro miliardi di host vengano connessi ad Internet. Abbiamo già presentato il problema che provocherebbe questo esaurimento dello spazio di indirizzamento: l'inefficienza nell'assegnazione degli indirizzi. Tale inefficienza nasce a causa della struttura dell'indirizzo IP, che, con gli indirizzi di classe A, B e C, ci costringe a gestire il nostro spazio di indirizzamento di rete in blocchi di dimensioni fisse, con tre dimensioni molto diverse tra loro. Una rete con due soli host necessita di un indirizzo di classe C, con un'efficienza nell'assegnazione degli indirizzi pari a $2/255 = 0.78\%$; una rete con 256 host ha bisogno di un indirizzo di classe B, con un'efficienza pari soltanto a $256/65535 = 0.39\%$. Anche se la suddivisione in sottoreti ci può aiutare ad assegnare con cura gli indirizzi, non risolve le situazioni che portano

ciascun sistema autonomo con più di 255 host (o che pensa di arrivare a tale numero) a volerle un indirizzo di classe B.

Dal punto di vista pratico, si osserva che l'esaurimento dello spazio di indirizzamento IP è dovuto all'esaurimento dei numeri di rete di classe B. Un modo di gestire questo problema potrebbe sembrare quello di negare indirizzi di classe B ad un sistema autonomo che non sia in grado di dimostrare di aver effettivamente bisogno di 64K indirizzi, assegnando invece un numero appropriato di indirizzi di classe C per contenere il numero previsto di host. Dato che in questo modo si assegnerebbe lo spazio di indirizzamento a blocchi di 256 indirizzi per volta, si potrebbe far meglio corrispondere la quantità di spazio di indirizzamento utilizzato alla dimensione del sistema autonomo. Per ciascun AS con almeno 256 host (cioè la maggioranza dei sistemi autonomi), possiamo garantire un'utilizzazione degli indirizzi almeno pari al 50%, e tipicamente più elevata.

Questa soluzione, però, solleva un problema che è almeno altrettanto serio: richiede un eccessivo spazio di memorizzazione nei router. Se ad un solo AS venissero assegnati, diciamo, 16 numeri di rete di classe C, ogni router del backbone di Internet avrebbe bisogno di 16 dati nelle sue tabelle di instradamento per rappresentare quel solo AS, anche se il percorso verso ciascuna di quelle reti fosse il medesimo. Se avessimo assegnato a quel AS un indirizzo di classe B, le stesse informazioni di instradamento sarebbero memorizzate in un solo dato nella tabella, anche se l'efficienza dell'assegnazione degli indirizzi sarebbe, in quel caso, soltanto $16 \times 255/65536 = 6.2\%$.

La strategia CIDR, quindi, cerca di bilanciare il desiderio di minimizzare il numero di percorsi che un router deve conoscere e la necessità di assegnare efficientemente gli indirizzi. Per fare ciò, CIDR aiuta ad aggregare i percorsi, cioè consente di usare un unico dato in una tabella di inoltro per indicare come raggiungere un insieme di reti diverse. Come dovreste aver intuito dal nome, realizza questo obiettivo spezzando i rigidi confini posti tra le classi di indirizzi. Per capire come ciò sia possibile, considerate il nostro ipotetico AS con 16 numeri di rete di classe C. Invece di assegnare 16 indirizzi a caso, possiamo assegnare un blocco di indirizzi di classe C contigui, ad esempio i numeri di rete di classe C che vanno da 192.4.16 a 192.4.31: osservate che i primi 20 bit di tutti gli indirizzi di questo intervallo sono uguali (11000000 00000100 0001). Di conseguenza, ciò che abbiamo creato, in realtà, è un numero di rete di 20 bit, qualcosa di intermedio, in termini di numero di host che può contenere tale rete, tra un numero di rete di classe B ed un numero di rete di classe C. In altre parole, otteniamo sia l'elevata efficienza nell'assegnazione degli indirizzi in blocchi più piccoli di una rete di classe B, sia un unico prefisso di rete da poter usare nelle tabelle di inoltro. Osservate che, perché questo schema funzioni, abbiamo bisogno di assegnare blocchi di indirizzi di classe C che condividono un prefisso comune, per cui ciascun blocco deve contenere un numero di reti di classe C che sia una potenza di due.

Tutto ciò di cui abbiamo bisogno, ora, perché CIDR possa risolvere tutti i nostri problemi è un protocollo di instradamento che possa gestire questi indirizzi "senza classe" (classless), cioè che possa capire che un numero di rete può avere una lunghezza qualsiasi. I moderni protocolli di instradamento (come BGP-4, descritto in seguito) fanno proprio questo. I numeri di rete utilizzati in questi protocolli di instradamento sono semplicemente rappresentati da una coppia di dati, $\langle \text{length}, \text{value} \rangle$, dove length (in numero di bit) del prefisso di rete, 20 nell'esempio precedente. Notate che rappresentare un indirizzo di rete in questo modo è simile all'approccio $\langle \text{mask}, \text{value} \rangle$ usato nelle sottoreti, con il vincolo che mask consista di bit contigui a partire dal bit più significativo. Notate anche che le sottoreti vengono utilizzate per condividere un indirizzo tra più reti fisiche, mentre CIDR ha l'obiettivo di

aggregare in un unico indirizzo i diversi indirizzi assegnati ad un unico AS. La somiglianza tra questi due approcci si rifletteva nel nome originale di CIDR: *supernetting*.

In realtà, la possibilità di aggregare i percorsi nel modo che abbiamo appena visto è soltanto il primo passo. Immaginate un fornitore di servizi di rete in Internet, il cui compito principale consiste nel fornire la connessione ad Internet ad un gran numero di aziende e strutture universitarie. Se assegniamo i numeri di rete alle aziende in modo che tutte le aziende diverse connesse al fornitore di rete condividano un prefisso di indirizzo comune, allora possiamo ottenere un'aggregazione di percorsi ancora maggiore. Considerate l'esempio di Figura 4.27. Alle due aziende servite dal fornitore di rete sono stati assegnati prefissi di rete a 20 bit adiacenti. Dato che entrambe le aziende sono raggiungibili tramite il medesimo fornitore di rete, quest'ultimo può pubblicizzare un unico percorso per entrambe, pubblicizzando semplicemente il prefisso comune, a 19 bit, che esse condividono. In generale, è possibile aggregare ripetutamente i percorsi, se gli indirizzi vengono assegnati con cura. Ciò significa che, se vogliamo che questo schema funzioni, dobbiamo fare attenzione a quale fornitore sia connessa un'azienda prima di assegnare ad essa un indirizzo: un modo per raggiungere questo scopo consiste nell'assegnare al fornitore una parte dello spazio degli indirizzi, lasciando che sia il fornitore stesso ad assegnare ai propri clienti indirizzi che appartengono a tale spazio.

Una rivisitazione dell'inoltro IP

Finora, in tutte le nostre analisi dell'inoltro IP, abbiamo ipotizzato di trovare un numero di rete nel pacchetto e di cercarlo in una tabella di inoltro, ma, ora che abbiamo presentato CIDR, dobbiamo riprendere in esame questa ipotesi. L'uso di CIDR implica che i prefissi possono avere una lunghezza qualsiasi, da 2 a 32 bit. Inoltre, a volte è possibile che nella tabella di inoltro siano presenti prefissi che si "sovrappongono", nel senso che alcuni indirizzi possono corrispondere a più di un prefisso. Ad esempio, potremmo avere nella tabella di inoltro di un singolo router sia 171.69 (un prefisso di 16 bit) sia 171.69.10 (un prefisso di 24 bit): in questo caso, un pacchetto destinato, ad esempio, all'indirizzo 171.69.10.5 corrisponde, ovviamente, ad entrambi i prefissi. In questo caso la regola utilizzata si basa sul principio della "maggior corrispondenza", cioè il pacchetto viene messo in corrispondenza al prefisso più lungo, che in questo caso sarebbe 171.69.10. D'altra parte, un pacchetto destinato a 171.69.20.5 troverebbe corrispondenza con 171.69 e *non* con 171.69.10: in assenza di altre corrispondenze nella tabella di instradamento, 171.69 sarebbe la corrispondenza più lunga.

Il compito di trovare la corrispondenza più lunga fra un indirizzo IP ed i prefissi di lunghezza variabile presenti in una tabella di inoltro è stato, negli ultimi anni, un campo di

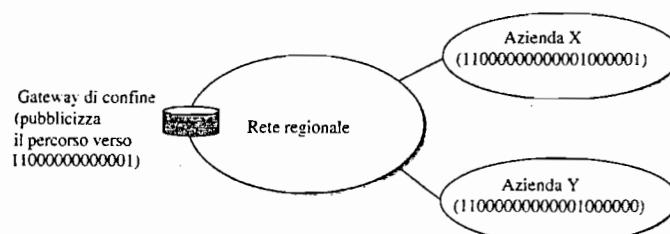


Figura 4.27 Aggregazione di percorsi con CIDR.

4.3 La rete Internet globale

ricerca molto fruttuoso, di cui troverete esempi nella sezione "Ulteriori letture" al termine del capitolo. L'algoritmo più noto usa un approccio che prende il nome di albero PATRICIA, che in realtà fu sviluppato ben prima di CIDR.

4.3.3 Instradamento interdominio (BGP)

All'inizio di questa sezione abbiamo presentato il concetto di organizzazione di Internet in sistemi autonomi, ciascuno dei quali si trova sotto il controllo di una singola entità di gestione amministrativa. Una complessa rete interna ad un'azienda potrebbe essere un unico AS. come può esserlo la rete di un singolo fornitore di servizi Internet. La Figura 4.28 mostra una semplice rete con due sistemi autonomi.

L'idea che sta alla base dei sistemi autonomi consiste nel fornire un ulteriore strumento per aggregare gerarchicamente le informazioni di instradamento in una grande internetwork, migliorando così la scalabilità. Suddividiamo ora il problema dell'instradamento in due parti: instradamento all'interno di un singolo sistema autonomo e instradamento tra sistemi autonomi. Dato che in Internet i sistemi autonomi vengono anche detti *domini*, chiameremo le due parti del problema con i nomi di instradamento intradominio e instradamento interdominio. Oltre a migliorare la scalabilità, il modello di sistema autonomo disaccoppia l'instradamento intradominio che avviene in un AS da quello che avviene in un altro AS: in modo che ciascun AS possa eseguire il protocollo intradominio che preferisce: può addirittura usare percorsi statici, se così preferisce, o perfino più protocolli di instradamento contemporaneamente. Il

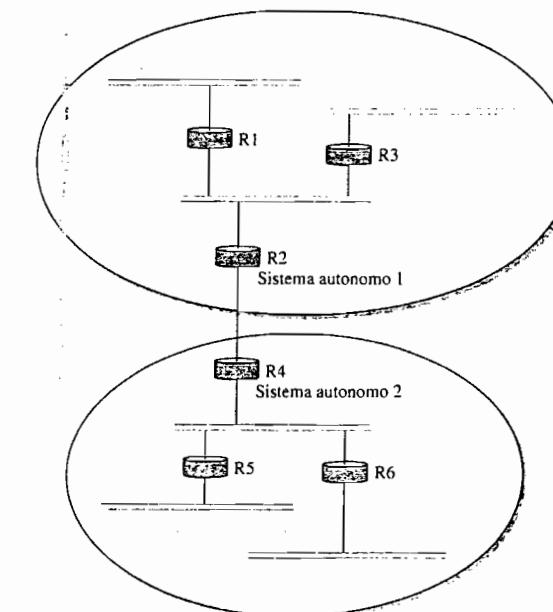


Figura 4.28 Una rete con due sistemi autonomi.

~~Orientato alla localizzazione del traffico interdominio~~
problema dell'instradamento interdominio si occupa, invece, di fare in modo che diversi AS condividano l'uno con l'altro le informazioni di raggiungibilità.

Una caratteristica dell'idea dei sistemi autonomi consiste nel consentire ad alcuni AS di ridurre drasticamente la quantità di informazioni di instradamento di cui si devono far carico mediante l'utilizzo di percorsi di default. Ad esempio, se una rete aziendale è connessa al resto di Internet mediante un unico router (che viene tipicamente chiamato *router di confine*), poiché si trova al confine fra il sistema autonomo ed il resto di Internet, allora diviene molto semplice per un host o un router interno al sistema autonomo scoprire dove inviare i pacchetti aventi destinazione esterna all'AS: per prima cosa devono andare al *router di confine* dell'AS, e questo diviene così il percorso di default. Analogamente, un fornito regionale di servizi Internet può tener traccia di come raggiungere le reti di tutti i suoi clienti direttamente connessi ed avere poi un percorso di default verso qualche altro fornito (tipicamente un fornito di backbone) per ogni altra destinazione. Questo passaggio di consegne, ovviamente, da qualche parte deve aver termine: prima o poi il pacchetto dovrebbe raggiungere un router connesso ad una rete backbone che sappia come raggiungere qualsiasi punto nella rete. La gestione della quantità di informazione di instradamento presente nei backbone è un problema di cui ora parleremo.

EGPBGP

Nella storia recente di Internet sono stati utilizzati due principali protocolli di instradamento interdominio. Il primo fu EGP (Exterior Gateway Protocol), che aveva parecchi limiti, il più severo dei quali, forse, era il fatto di costringere Internet ad assumere una topologia sostanzialmente ad albero (perché fu progettato quando Internet aveva una topologia ad albero, come in Figura 4.24). EGP non consentiva alla topologia di assumere forme più generali. Notate che in questa semplice struttura ad albero esiste un singolo backbone e i sistemi autonomi sono fra loro collegati soltanto con una relazione di tipo genitore e figlio e non come elementi di pari livelli (*peer*).

Il sostituto di EGP fu BGP (Border Gateway Protocol), che nel momento in cui scriviamo è alla sua quarta versione (BGP-4). BGP è famoso per essere un protocollo piuttosto complesso, per cui in questa sezione ne presenteremo soltanto le caratteristiche principali.

Il protocollo BGP si basa su un'ipotesi di partenza: Internet è un insieme di AS arbitrariamente interconnesse. Questo modello è ovviamente abbastanza generico da poter gestire reti interconnesse con strutture non ad albero, come la visione semplificata della rete Internet odierna, con più backbone, presentata in Figura 4.29.

Diversamente dalla semplice Internet con struttura ad albero vista in Figura 4.24, oggi Internet è formata dall'interconnessione di più reti backbone (che vengono chiamate "reti fornitrice di servizi", *service provider networks*, e sono gestite da compagnie private e non più dal governo) e da siti connessi l'uno all'altro in modi diversi. Alcune grandi aziende sono direttamente connesse ad uno o più backbone, mentre altre si connettono a più piccoli fornitori di servizi, non di tipo backbone. Molti fornitori di servizio hanno come clienti soltanto "consumer", cioè singole persone con PC in casa, e tali fornitori devono, essi stessi, connettersi a fornitori di servizi di tipo backbone. Spesso molti fornitori si interconnettono l'un l'altro in un singolo "peering point". In breve, è difficile descrivere in dettaglio la struttura odierna di Internet.

In base a questa schematica rappresentazione di Internet, se definiamo *traffico locale* il traffico che ha origine o termina in nodi che si trovano all'interno di un AS e *traffico in*

⁶ Estendendo la metafora, oggi Internet ha più spine dorsali (*backbone*), dopo averne avuta soltanto una per molti anni. Gli autori non conoscono altri animali con questa caratteristica.

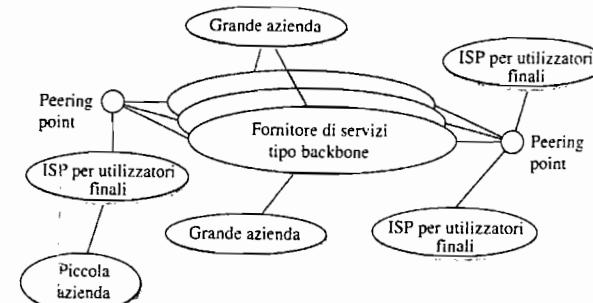


Figura 4.29 L'odierna Internet con più backbone.

transito il traffico che attraversa un AS, possiamo classificare i sistemi autonomi in tre tipi:

- **AS di tipo stub**, che ha una sola connessione verso un altro AS e, quindi, trasporta soltanto traffico locale. In Figura 4.29, la piccola azienda è un esempio di AS di tipo stub.
- **AS di tipo multihomed**, che ha connessioni con più AS ma rifiuta di trasportare traffico in transito (come la grande azienda nella Figura 4.29, in alto).
- **AS di transit**, che ha connessioni con più AS ed è progettato per trasportare sia traffico locale sia traffico in transito, come i fornitori di tipo backbone nella Figura 4.29.

Nonostante la discussione sull'instradamento della Sezione 4.2 fosse focalizzata sull'identificazione dei percorsi ottimali basati sulla minimizzazione di una qualche metrica delle linee di collegamento, il problema dell'instradamento interdominio si rivela talmente complesso che gli obiettivi sono più modesti: trovare un percorso qualsiasi verso la destinazione finale, che sia privo di cicli (loop-free). In sostanza, siamo più interessati alla raggiungibilità che all'ottimizzazione. L'identificazione di un percorso che sia in qualche modo prossimo all'ottimale è considerato un grande risultato: vedremo perché quando esamineremo i dettagli del protocollo BGP.

Esistono diversi motivi che rendono difficile l'instradamento interdominio. Prima di tutto, c'è un problema di dimensioni. Un router di una rete backbone in Internet deve essere in grado di inoltrare pacchetti destinati a qualsiasi punto nella rete, per cui deve avere una tabella di instradamento che fornisca una corrispondenza per qualsiasi indirizzo IP valido. Anche se CIDR viene in aiuto per tenere sotto controllo il numero di prefissi diversi che vengono utilizzati per l'instradamento nei backbone di Internet, è inevitabile che vi sia molta informazione di instradamento da far circolare, dell'ordine di 140000 prefissi nel momento in cui scriviamo.

La seconda sfida nell'instradamento interdominio viene dalla natura autonoma dei domini. Ciascun dominio può eseguire, al proprio interno, protocolli di instradamento diversi e usare qualsiasi schema di propria scelta per assegnare i costi ai percorsi. Ciò significa che è impossibile calcolare costi significativi per percorsi che attraversino più AS. Un costo uguale a 1000 per un fornitore potrebbe rappresentare un ottimo percorso, che potrebbe essere inaccettabile per un altro fornitore. Il risultato è che l'instradamento interdominio pubblicizza

soltanto la "raggiungibilità", cioè in sintesi, l'affermazione che "tramite questo AS si può raggiungere quella rete". Ciò significa che, per l'instradamento interdominio, scegliere un percorso ottimale è sostanzialmente impossibile.

La terza sfida è rappresentata dal problema della fiducia. Il fornitore A potrebbe essere scettico nel credere ad alcune affermazioni pubblicate dal fornitore B, per timore che esso pubblicizzi informazioni di instradamento errate. Ad esempio, confidare nel fatto che il fornitore B pubblicizza un eccellente percorso verso qualsiasi destinazione di Internet può essere una scelta disastrosa, nel momento in cui si scopre che il fornitore B ha compiuto un errore di configurazione nei propri router o non ha sufficiente capacità per trasportare il traffico.

In stretta relazione con questo problema, c'è la necessità di fornire supporto per politiche molto flessibili nell'instradamento interdominio. Una politica molto comune è il divieto per il traffico in transito: ad esempio, l'azienda multihomed di Figura 4.29 potrebbe non desiderare traffico in transito fra i due fornitori a cui è connessa. Come esempio più complesso, il fornitore A potrebbe voler implementare politiche che dicono: "usa il fornitore B soltanto per raggiungere questi indirizzi", "usa il percorso che attraversa il minor numero di AS" oppure "preferisci il sistema autonomo x rispetto al sistema autonomo y". L'obiettivo è quello di specificare politiche che forniscano "buoni" percorsi, se non i percorsi ottimi.

Per configurare il protocollo BGP, il gestore di ciascun AS sceglie almeno un nodo e gli assegna la funzione di "annunciatore BGP" (BGP speaker), che sostanzialmente effettua gli annunci per l'intero AS. Tale annunciatore BGP stabilisce sessioni con gli annunciatori BGP di altri AS, per scambiare informazioni in merito alla raggiungibilità.

Speaker BGP
GATEWAY di CONFINE

Oltre agli annunciatori BGP, ogni AS ha uno o più gateway ("cancelli") di confine, che non coincidono necessariamente con gli annunciatori. I gateway di confine (border gateway) sono i router mediante i quali i pacchetti entrano ed escono dal sistema autonomo. Nel nostro semplice esempio di Figura 4.28, i router R2 e R4 sarebbero i gateway di confine. Notate che fino a questo momento abbiamo sempre evitato l'uso della parola "gateway", perché tende a creare confusione: in questo punto non lo possiamo evitare, visto il nome del protocollo che stiamo descrivendo. La cosa importante da capire è che, nel contesto dell'instradamento interdominio, un gateway di confine è semplicemente un router IP che ha il compito di inoltrare pacchetti tra AS diversi.

Il protocollo BGP non appartiene a nessuna delle due categorie principali dei protocolli di instradamento descritti nella Sezione 4.2 (a vettore di distanza e a stato delle linee). Diversamente da questi protocolli, BGP pubblica percorsi completi, sotto forma di elenchi di AS tramite i quali si può raggiungere una particolare rete. Ciò è indispensabile per consentire quei tipi di strategie descritte in precedenza, in relazione alle scelte compiute dal singolo AS, e per consentire la pronta identificazione di cicli nei percorsi di instradamento.

Per vedere come funziona tutto ciò, considerate la rete di esempio di Figura 4.30. Ipotizzate che i fornitori siano reti di transito, mentre le reti dei clienti sono di tipo stub. Un annunciatore BGP per l'AS del fornitore A (AS 2) sarebbe in grado di fornire informazioni sulla raggiungibilità di ciascuno dei numeri di rete assegnati ai clienti P e Q, per cui, in pratica, direbbe: "le reti 128.96, 192.4.153, 192.4.32 e 192.4.3 sono direttamente raggiungibili da AS 2". La rete backbone, ricevendo questo messaggio, potrebbe dire: "le reti 128.96, 192.4.153, 192.4.32 e 192.4.3 sono raggiungibili con il percorso (AS 1, AS 2)". Analogamente, potrebbe annunciare: "le reti 192.12.69, 192.4.54 e 192.4.23 sono raggiungibili con il percorso (AS 1, AS 3)".

Uno dei compiti importanti di BGP consiste nel prevenire l'instaurarsi di percorsi ciclici. Ad esempio, considerate tre AS interconnessi, 1, 2 e 3. Supponete che AS 1 apprenda di poter

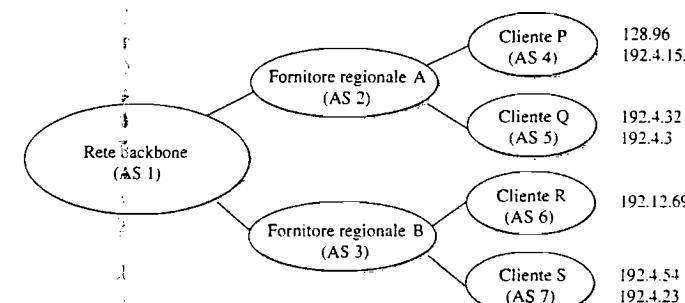


Figura 4.30 Esempio di rete che esegue il protocollo BGP.

raggiungere la rete 10.0.1 attraverso AS 2, per cui pubblicizza questa possibilità a AS 3, il quale a sua volta lo rende noto a AS 2. Ora, AS 2 potrebbe decidere di inviare a AS 3 i pacchetti destinati alla rete 10.0.1; AS 3 li invia a AS 1, il quale li restituisce a AS 2, e i pacchetti circolerebbero all'infinito. Ciò viene evitato inserendo nei messaggi di instradamento l'elenco completo degli AS coinvolti in un percorso. In questo caso, il messaggio ricevuto da AS 2 e inviato da AS 3 conterebbe un percorso di AS così composto: (AS 3, AS 1, AS 2). AS 2 vede che il percorso contiene il proprio nome, per cui trae la conclusione di ignorare tale percorso, in quanto inutile.

Dovrebbe essere evidente che i numeri di AS utilizzati nel protocollo BGP devono essere univoci. Ad esempio, AS 2 può identificare se stesso nel percorso di sistemi autonomi visto nell'esempio precedente soltanto se nessun altro AS identifica se stesso nello stesso modo. I numeri di AS sono numeri a 16 bit assegnati da un'autorità centrale, per garantirne l'unicità. Anche se 16 bit consentono l'esistenza di soli 65000 sistemi autonomi, che non sembrano molti, notiamo che i sistemi autonomi di tipo stub non necessitano di numero univoci, e questi ultimi sono la stragrande maggioranza delle reti*.

Notiamo anche che un AS pubblicherà soltanto quei percorsi che considera sufficientemente validi per sé; se un annunciatore BGP ha una scelta di diversi percorsi verso una destinazione, sceglierà il migliore in base alle politiche locali, e questo sarà l'unico ad essere pubblicizzato. Inoltre, non esiste alcun obbligo per un annunciatore BGP di pubblicizzare un percorso: in questo modo un AS può realizzare una politica che impedisce il traffico di transito, rifiutandosi di pubblicizzare percorsi verso prefissi che non appartengano al proprio AS, anche se sa come raggiungerli.

Oltre a pubblicizzare percorsi, gli annunciatori BGP devono essere in grado di cancellare percorsi precedentemente pubblicizzati, nel caso in cui lungo il percorso si sia guastata una linea critica o un nodo. Ciò avviene con una specie di informazione negativa, nota come withdrawal route (ritiro di un percorso). Sia le informazioni di raggiungibilità negative, sia quelle positive, vengono trasportate all'interno di un messaggio di aggiornamento BGP, il cui formato è mostrato in Figura 4.31 (notate che i campi, in questa figura, sono multipli di 16 bit, diversamente dai formati di altri pacchetti di questo capitolo).

Un punto di BGP-4 che è degno di nota è il fatto che questo protocollo è stato progettato per gestire gli indirizzi senza classi, descritti nella Sezione 4.3.2. Ciò significa che le "reti" che vengono pubblicate in BGP sono in realtà prefissi di lunghezza arbitraria. Di conse-

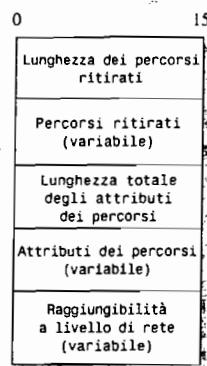


Figura 4.31 Formato del pacchetto di aggiornamento BGP.

guenza, gli aggiornamenti contengono sia il prefisso, sia la sua lunghezza in numero di bit, informazione che solitamente viene scritta nella forma prefisso/lunghezza: ad esempio, un prefisso CIDR che inizi con 192.4.16 è sia lungo 20 bit, si scrive 192.4.16/20.

L'ultima cosa da notare è che BGP è stato definito per essere eseguito al di sopra del protocollo TCP¹⁰, il protocollo di trasporto affidabile descritto nella Sezione 5.2. Dato che gli annunciatori BGP possono contare sul fatto che il protocollo TCP è affidabile, ciò significa che qualsiasi informazione che sia stata inviata da un annunciatore ad un altro non deve essere inviata di nuovo; per cui, fintanto che non ci sono modifiche, un annunciatore BGP può semplicemente inviare un messaggio periodico di tipo "keepalive", che, in sostanza, afferma: "sono ancora qui e niente è cambiato". Se tale router dovesse smettere di funzionare, smetterebbe anche di inviare tali messaggi, e gli altri router che avevano appreso percorsi da esso saprebbero che tali percorsi non sono più validi.

Non entreremo in maggiore dettaglio nel protocollo BGP-4, se non per puntualizzare che tutto ciò che fa questo protocollo è specificare come le informazioni sulla raggiungibilità debbano essere scambiate fra sistemi autonomi. Da questi scambi, gli annunciatori BGP ottengono informazioni sufficienti per calcolare percorsi aciclici che raggiungano tutte le reti raggiungibili, ma la responsabilità di scegliere il percorso "migliore" viene demandata alle strategie del singolo AS.

Torniamo ora al vero problema: come può tutto questo aiutarci a costruire reti scalabili? Innanzitutto, il numero di nodi coinvolti nel protocollo BGP è dell'ordine del numero di sistemi autonomi, che è molto inferiore al numero di reti. Secondariamente, trovare un percorso valido fra domini richiede solamente di trovare un percorso che raggiunga il corretto router di confine, di cui ne esistono pochi in ciascun AS. Di conseguenza, abbiamo elegantemente suddiviso il problema dell'instradamento in parti più gestibili, aggiungendo di nuovo un livello di gerarchia per migliorare la scalabilità. La complessità dell'instradamento interdominio è ora dello stesso ordine di grandezza del numero di AS e la complessità dell'instradamento intradominio è dello stesso ordine di grandezza del numero di reti all'interno di un singolo AS.

4.3 La rete Internet globale

Integrazione tra l'instradamento interdominio e intradominio

Mentre la discussione precedente mostra come un annunciatore BGP apprende le informazioni di instradamento interdominio, rimane aperta la domanda che riguarda come tutti gli altri router del dominio possano ottenere questa informazione. Questo problema può essere risolto in vari modi.

Abbiamo già accennato ad una possibile soluzione in una situazione molto semplice, che è anche molto comune. Nel caso di un AS di tipo stub che si connetta ad altri AS in un unico punto, il router di confine è anche, ovviamente, l'unica scelta per tutti i percorsi che escono dal dominio: tale router può "iniettare" un percorso di default nel protocollo di instradamento intradominio. A tutti gli effetti, ciò significa affermare che tutte le reti che non siano esplicitamente pubblicate dal protocollo intradominio sono raggiungibili tramite il router di confine. Ricordate, dalla discussione sull'inoltro IP vista nella Sezione 4.1, che il dato di default nella tabella di inoltro si trova al termine di tutti gli altri dati più specifici, e che costituisce una corrispondenza per qualsiasi indirizzo che non abbia trovato una corrispondenza più specifica.

Il passo successivo, di maggiore complessità, consiste nel fare in modo che i router di confine pubblicizzino all'interno del dominio alcuni percorsi specifici che hanno appreso dall'esterno del dominio. Considerate, ad esempio, il router di confine del sistema autonomo di un fornitore connesso all'AS di un cliente. Dal protocollo BGP o per un'informazione configurata al proprio interno, tale router potrebbe aver appreso che il prefisso di rete 192.4.54/24 si trova all'interno del sistema autonomo del cliente. Potrebbe, quindi, inserire nel protocollo di instradamento che viene eseguito nell'AS del fornitore un percorso verso tale prefisso, che sarebbe un messaggio del tipo: "Ho una linea di collegamento verso 192.4.54/24 con costo X".

L'ultimo livello di complessità è relativo alle reti backbone, che ricevono così tante informazioni di instradamento dal protocollo BGP che inserirle nei protocolli intradominio diviene troppo costoso. Ad esempio, se un router di confine volesse inserire i 10000 prefissi che ha appreso da un altro AS, dovrebbe inviare enormi pacchetti sullo stato delle linee agli altri router che si trovano nello stesso sistema autonomo, complicando così notevolmente il loro calcolo dei percorsi più brevi. Per questo motivo i router nelle reti backbone usano una variante di BGP denominata IBGP (*interior BGP*) per distribuire a tutti gli altri router del dominio in modo efficiente l'informazione appresa dagli annunciatori BGP che si trovano ai confini del dominio stesso. Il protocollo IBGP consente a qualsiasi router del dominio di sapere quale sia il router di confine migliore da utilizzare per inviare pacchetti ad un indirizzo qualsiasi. Al tempo stesso, ciascun router nell'AS tiene traccia di come raggiungere ciascun router di confine, usando un protocollo intradominio convenzionale, senza informazioni aggiuntive. Combinando questi due insiemi di informazioni, ogni router nell'AS è in grado di determinare il next hop appropriato per tutti i prefissi.

4.3.4 Aree di instradamento

Dato che non abbiamo ancora abbastanza gerarchie, i protocolli di instradamento intradominio basati sullo stato delle linee forniscono il modo di suddividere un dominio di instradamento in sottodomini, chiamati *aree* (la terminologia è abbastanza variabile nei diversi protocolli, usiamo qui i termini di OSPF). Aggiungendo questo ulteriore livello gerarchico, consentiamo ai singoli domini di crescere senza sovraccaricare i protocolli di instradamento intradominio.

→ Area Border Router

Un'area è un insieme di router che vengono configurati per scambiarsi reciprocamente informazioni sullo stato delle linee, ed esiste un'area speciale: l'area di backbone, nota anche come area 0. Un esempio di dominio di instradamento diviso in aree viene mostrato in Figura 4.32. I router R1, R2 e R3 sono membri dell'area di backbone, ma sono anche membri di almeno un'altra area (R1 è membro sia dell'area 1 che dell'area 2). Un router che sia membro sia dell'area di backbone sia di un'area non di backbone viene detto ABR (area border router, router di confine fra aree), notate che questi router sono diversi dai router che si trovano ai confini di un sistema autonomo, che qui verranno chiamati router di confine tra AS, per chiarezza.

L'instradamento all'interno di un'area funziona esattamente come descritto nella Sezione 4.2.3! Tutti i router dell'area si scambiano messaggi sullo stato delle linee, generando così una mappa completa e coerente dell'area, però i messaggi che provengono da router che non siano ABR non escono dall'area in cui sono stati generati. In questo modo, i processi di inondazione e di calcolo dei percorsi diventano molto più scalabili. Ad esempio, il router R4 nell'area 3 non invierà mai un messaggio sullo stato delle linee al router R8, che si trova nell'area 1. Di conseguenza, non avrà alcuna informazione sui dettagli della topologia di aree diverse dalla propria.

Ora, come fa un router di un'area a determinare il next hop corretto per un pacchetto destinato ad una rete che si trova in un'altra area? La risposta a questa domanda diviene chiara se pensiamo al percorso di un pacchetto che deve viaggiare da un'area non di backbone verso un'altra area non di backbone come se fosse suddiviso in tre parti. Dapprieta il pacchetto viaggia dalla propria rete sorgente all'area di backbone, poi attraversa il backbone e infine, viaggia dal backbone alla rete di destinazione. Per rendere possibile tutto questo, i router di confine delle aree riassumono le informazioni di instradamento che hanno raccolto da un'area e le rendono disponibili nei messaggi che inviano alle altre aree. Ad esempio, R1 riceve messaggi sullo stato delle linee da tutti i router dell'area 1 e può così determinare il costo necessario per raggiungere qualsiasi rete dell'area 1. Quando R1 invia nell'area 0 messaggi sullo stato delle linee, comunica anche il costo necessario per raggiungere le reti che si trovano nell'area 1, più o meno come se tali reti fossero direttamente connesse a R1. Ciò

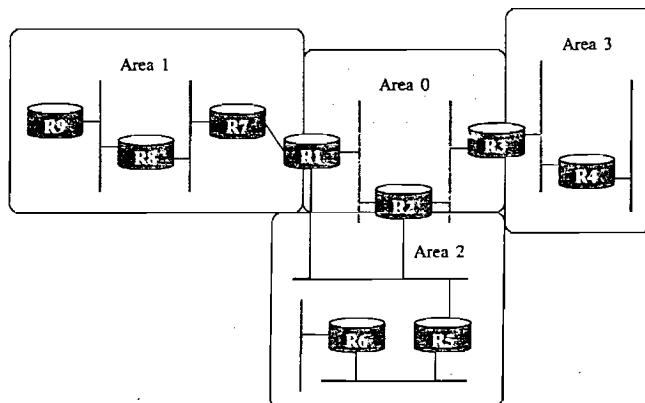


Figura 4.32 Un dominio suddiviso in aree.

consente a tutti i router dell'area 0 di sapere il costo necessario per raggiungere tutte le reti che si trovano nell'area 1. I router di confine dell'area riassumono poi questa informazione e la pubblicizzano nelle aree non di backbone, in modo che tutti i router sappiano come raggiungere tutte le reti del dominio.

Note che nell'area 2 esistono due ABR, per cui i router dell'area 2 dovranno effettuare una scelta in merito a quale dei due utilizzare per raggiungere il backbone. La scelta è abbastanza semplice, perché sia R1 sia R2 pubblicizzeranno i costi necessari per raggiungere le varie reti, in modo che divenga chiaro quale sia la scelta migliore nel momento in cui ciascun router dell'area 2 eseguirà il proprio algoritmo per l'identificazione del percorso migliore. Ad esempio, è molto evidente che R1 sarà una scelta migliore di R2 per le destinazioni che appartengono all'area 1.

Suddividendo un dominio in aree, il gestore di rete realizza un compromesso fra la scalabilità e l'ottimizzazione dell'instradamento. L'uso di aree obbliga tutti i pacchetti a seguire un percorso tra un'area ed un'altra passando sempre per l'area di backbone, anche se potrebbe esistere un percorso più breve. Ad esempio, anche se R4 e R5 fossero connessi direttamente, i pacchetti non verrebbero scambiati direttamente tra loro, perché si trovano in aree diverse e non di backbone. Nella pratica, il bisogno di scalabilità è spesso più importante della necessità di usare il percorso più breve in assoluto.

Questo fatto mette in evidenza un principio importante nella progettazione di reti di calcolatori: viene spesso ricercato un compromesso tra una qualche forma di scalabilità e di ottimizzazione. Quando si introduce una gerarchia, si nascondono informazioni ad alcuni nodi della rete, diminuendo la loro capacità di prendere decisioni veramente ottime, però nascondere informazioni è essenziale per la scalabilità, poiché elimina la necessità di avere una conoscenza globale del problema. In reti di grandi dimensioni, è innegabile che la scalabilità sia un'esigenza più sentita dell'ottimizzazione perfetta.

Infine, notiamo che esiste un trucco mediante il quale i gestori di rete possono decidere in modo più flessibile quali siano i router che appartengono all'area 0, un trucco che sfrutta il concetto di "linea di collegamento virtuale" fra router. Tale linea virtuale si ottiene configurando un router che non sia direttamente connesso all'area 0 in modo che scambi con un router che vi appartenga informazioni relative all'instradamento nel backbone. Ad esempio, si potrebbe configurare una linea virtuale da R8 a R1, rendendo così R8 parte del backbone: R8 partecerebbe, insieme agli altri router dell'area 0, all'inondazione delle informazioni relative allo stato delle linee. Il costo della linea virtuale fra R8 e R1 viene determinato dallo scambio delle informazioni di instradamento che avviene nell'area 1. Questa tecnica può essere d'aiuto nel migliorare l'ottimizzazione dell'instradamento.

4.3.5 La versione 6 di IP (IPv6)

Per molti aspetti la motivazione per una nuova versione del protocollo IP è identica alla motivazione delle tecniche descritte nella sezione precedente: gestire i problemi di scalabilità provocati dall'imponente crescita di Internet. Le sottoreti e CIDR hanno aiutato a contenere la velocità con la quale si va consumando lo spazio di indirizzamento di Internet (il problema della scarsità di indirizzi) ed ha anche aiutato a tenere sotto controllo la crescita delle informazioni presenti nelle tabelle di instradamento all'interno dei router di Internet (il problema

delle informazioni di instradamento). Tuttavia, si giungerà al punto in cui queste tecniche non saranno più adeguate. In particolare, è virtualmente impossibile ottenere un'efficienza del 100% nell'utilizzazione degli indirizzi, per cui lo spazio di indirizzamento verrà esaurito ben prima che quattro miliardi di host vengano connessi a Internet. Anche se si riuscisse ad usare tutti i quattro milioni di indirizzi, non è difficile immaginare in che modo si possano esaurire, se si pensa di assegnare indirizzi IP anche agli apparati per la TV via cavo (*set-top box*) o ai contatori dell'energia elettrica. Tutte queste eventualità suggeriscono la necessità di uno spazio di indirizzamento più vasto di quello fornito da 32 bit.

Un'analisi storica

La IETF iniziò a considerare il problema di espandere lo spazio di indirizzamento del protocollo IP nel 1991, e vennero proposte diverse alternative. Dato che l'indirizzo IP viene trasmesso all'interno di ciascun pacchetto IP, nella sua intestazione, aumentare la dimensione dell'indirizzo impone una modifica dell'intestazione stessa: ciò significa una nuova versione di Internet Protocol e, di conseguenza, il bisogno di nuovo software per ogni host e ogni router di Internet, un problema tutt'altro che banale: piuttosto, una modifica che deve essere progettata con grande attenzione.

Il tentativo di definire una nuova versione di IP prese il nome di IP Next Generation, o IPng. Con il progredire dei lavori, venne assegnato un numero di versione IP ufficiale, per cui IPng è ora noto come IPv6. Notate che la versione di IP vista finora in questo capitolo è la versione 4 (IPv4): l'apparente discontinuità nella numerazione è dovuta al fatto che la versione numero 5 venne usata qualche anno fa per un protocollo sperimentale.

Il fatto che una nuova versione di IP richiedesse un cambiamento così significativo innescò un fenomeno a valanga: il sentimento generale fra i progettisti di reti era che se si profilava un cambiamento di tale portata, valeva la pena sistemare allo stesso tempo il maggior numero dei problemi presenti nel protocollo IP. Di conseguenza, la IETF sollecitò proposte da chiunque fosse interessato a scriverne una, chiedendo di fornire idee sulle caratteristiche che avrebbero potuto essere richieste in una nuova versione di IP. Oltre al bisogno di gestire scalabilità di instradamento e indirizzamento, alcune delle altre novità richieste per IPng furono

- supporto per servizi in tempo reale (*real-time services*)
- supporto per la sicurezza
- autoconfigurazione, ossia la capacità degli host di configurare automaticamente se stessi con informazioni quali il proprio indirizzo IP e il proprio nome di dominio
- migliore funzionalità di instradamento, compreso il supporto per gli host mobili

È interessante notare come la maggior parte di queste caratteristiche fosse assente in IPv4 nel momento in cui fu progettato IPv6, ma che negli ultimi anni esse sono state tutte implementate anche in IPv4.

Oltre a questo elenco, una caratteristica assolutamente irrinunciabile per IPng era che ci dovesse essere un piano per gestire la transizione dalla versione attuale di IP (versione 4) alla nuova versione. Essendo Internet così vasta e priva di controllo centralizzato, sarebbe assolutamente impossibile immaginare di definire un momento in cui tutti spengano i propri host e router e installino una nuova versione di IP. Di conseguenza, vi sarà con ogni probabilità un lungo periodo di transizione in cui alcuni host e router eseguiranno soltanto IPv4, alcuni eseguiranno sia IPv4 sia IPv6, mentre altri eseguiranno soltanto IPv6.

La IETF nominò un comitato che prese il nome di IPng Directorate, con il compito di raccogliere tutte le idee relative a requisiti da impostare ad IPng e di valutare le proposte di protocolli che potessero divenire IPng. Durante la vita di questo comitato vi furono un certo numero di proposte, alcune delle quali si fusero con altre, e alla fine una venne scelta dal Directorate per costituire la base di IPng. Quella proposta venne chiamata SIPP (Simple Internet Protocol Plus) e, in origine, prevedeva il raddoppio della dimensione dell'indirizzo IP, portandolo a 64 bit. Quando il Directorate scelse SIPP, vennero aggiunte alcune modifiche, una delle quali fu un ulteriore raddoppio della dimensione degli indirizzi, arrivando a 128 bit (16 byte). Proprio in quel periodo venne assegnato il numero di versione 6. La parte restante di questa sezione descrive alcune delle caratteristiche principali di IPv6. Nel momento in cui questo libro viene scritto, la maggior parte delle specifiche più importanti di IPv6 sono Proposed Standard o Draft Standard all'interno di IETF.

Indirizzo IPv6 → 128 bit

Indirizzi e instradamento

Prima di tutto, IPv6 fornisce uno spazio di indirizzamento a 128 bit, mentre IPv4 usa indirizzi a 32 bit. Di conseguenza, mentre la versione 4 può potenzialmente indirizzare quattro miliardi di nodi (se l'efficienza dell'assegnazione degli indirizzi raggiungesse il 100%), IPv6 può indirizzare 3.4×10^{38} nodi, ipotizzando nuovamente un'efficienza del 100%. Come abbiamo visto, però, un'efficienza del 100% nell'assegnazione degli indirizzi è altamente improbabile. Alcune analisi di altri schemi di indirizzamento, come quelli usati nelle reti telefoniche francesi e statunitensi, così come quello di IPv4, hanno fornito alcune valutazioni empiriche per l'efficienza dell'assegnazione degli indirizzi. Basandosi sulle stime più pessimistiche dell'efficienza, derivate da questi studi, lo spazio di indirizzamento di IPv6 sarebbe in grado di fornire più di 1500 indirizzi per ogni piede quadrato di superficie terrestre, un valore che probabilmente ci sarà sufficiente anche quando i tostapane sul pianeta Venere avranno indirizzi IP.

Allocazione dello spazio di indirizzamento

Gli indirizzi di IPv6 non usano le classi, ma lo spazio di indirizzamento è comunque suddiviso in vari modi, in base ai bit iniziali. Invece di specificare diverse classi di indirizzi, i bit iniziali specificano diversi usi dell'indirizzo IP. L'attuale assegnamento di prefissi è indicato in Tabella 4.11.

Questa allocazione dello spazio degli indirizzi si rivela più semplice da spiegare di quanto sembri. Innanzitutto, l'intera funzionalità delle tre classi principali di indirizzi di IPv4 (A, B e C) viene svolta all'interno del prefisso 001. Gli indirizzi unicast globali e aggregabili (Aggregatable Global Unicast Addresses), come vedremo fra breve, assomigliano molto agli indirizzi IPv4 senza classi, sono soltanto molto più lunghi: in questo momento sono quelli più interessanti, e un ottavo dello spazio degli indirizzi è assegnato a questa importante categoria. Come è naturale che accada, grandi sezioni di spazio di indirizzamento sono state lasciate senza assegnazione, per consentire future espansioni e nuove funzionalità. Due segmenti dello spazio degli indirizzi (0000 001 e 0000 010) sono stati riservati alla codifica di altri schemi di indirizzamento, diversi da IP: gli indirizzi NSAP sono usati dai protocolli ISO, mentre gli indirizzi IPX sono utilizzati dal protocollo dello strato di rete di Novell.

L'idea di fondo per gli indirizzi di "uso su una linea locale" (*link local use addresses*) è quella di consentire ad un host di costruire un indirizzo che funzioni solamente sulla rete a cui è connesso, senza preoccuparsi dell'unicità globale dell'indirizzo stesso. Ciò può essere

Tabella 4.11 Assegnamento dei prefissi degli indirizzi in IPv6.

Prefisso	Utilizzo
0000 0000	Riservato
0000 0001	Non assegnato
0000 001	Riservato per assegnazioni NSAP
0000 010	Riservato per assegnazioni IPX
0000 011	Non assegnato
0000 1	Non assegnato
0001	Non assegnato
001	Aggregatable Global Unicast Addresses
010	Non assegnato
011	Non assegnato
100	Non assegnato
101	Non assegnato
110	Non assegnato
1110	Non assegnato
1111 0	Non assegnato
1111 10	Non assegnato
1111 110	Non assegnato
1111 1110 0	Non assegnato
1111 1110 10	Indirizzi da usare su una linea locale
1111 1110 11	Indirizzi da usare in un sito locale
1111 1111	Indirizzi di tipo multicast

utile per l'autoconfigurazione, come vedremo più avanti. In modo analogo, gli indirizzi di "uso in un sito locale" (*site local use addresses*) sono destinati a consentire la costruzione di indirizzi che siano validi soltanto in un sito (come, ad esempio, una rete aziendale privata) che non sia connesso alla Internet globale: anche in questo caso, non è necessario preoccuparsi dell'unicità globale.

Infine, lo spazio degli indirizzi di tipo multicast è destinato alla trasmissione multicast, svolgendo quindi lo stesso compito degli indirizzi di classe D in IPv4. Notate che è facile identificare gli indirizzi multicast, dato che iniziano con un byte che contiene tutti i bit al valore 1. Vedremo come utilizzare questi indirizzi nella Sezione 4.4.

All'interno dello spazio di indirizzamento riservato (costituito dagli indirizzi che iniziano con un byte avente tutti i bit al valore 0) si trovano tipi speciali di indirizzi assai importanti. Si può assegnare ad un nodo un "indirizzo IPv6 compatibile con IPv4" (*IPv4-compatible IPv6 address*) estendendo un indirizzo di 32 bit, rendendolo di 128 bit mediante l'anteposizione di zeri. Ad un nodo che sia in grado di gestire solamente la versione 4 del protocollo IP può essere assegnato un "indirizzo IPv6 trasformato in IPv4" (*IPv4-mapped IPv6 address*), anteponendo ai 32 bit dell'indirizzo IPv4 due byte aventi tutti i bit al valore 1, per poi estendere il risultato fino alla lunghezza di 128 bit mediante l'anteposizione di zeri. Questi due tipi speciali di indirizzi sono utilizzati nella transizione da IPv4 a IPv6.

4.3 La rete Internet globale

Transizione da IPv4 a IPv6

L'idea centrale che sta alla base della transizione da IPv4 a IPv6 è che la rete Internet sia troppo grande e decentralizzata da poter avere un unico "istante di transizione" (*flag day*), un momento specifico in cui tutti gli host e tutti i router vengano aggiornati da IPv4 a IPv6. Di conseguenza, è necessario che IPv6 venga installato in modo incrementale, con una metodologia che consenta agli host e ai router che gestiscono soltanto IPv4 di continuare a funzionare il più a lungo possibile. Idealmente, i nodi IPv4 dovrebbero essere in grado di colloquiare indefinitamente con altri nodi IPv4 e con alcuni insiemi di altri nodi con potenzialità di IPv6. Inoltre, gli host che usano IPv6 dovrebbero essere in grado di colloquiare con altri nodi IPv6 anche quando alcune delle infrastrutture interposte forniscono solamente il supporto a IPv4. Per aiutare questa transizione, sono stati definiti due meccanismi fondamentali: l'operatività a doppia pila (*dual stack operation*) e il tunneling.

L'idea alla base dell'operatività a doppia pila è elementare: i nodi IPv6 eseguono sia il protocollo IPv6 sia il protocollo IPv4 e usano il campo Version per decidere quale pila di protocolli debba elaborare il pacchetto in arrivo. In questo caso, l'indirizzo IPv6 potrebbe non avere alcuna correlazione con l'indirizzo IPv4, oppure potrebbe essere un "indirizzo IPv6 trasformato in IPv4", come descritto in precedenza.

La tecnica basilare di tunneling, in cui un pacchetto IP viene inviato come *carica utile* di un altro pacchetto IP, è stata descritta nella Sezione 4.1. Per la transizione verso IPv6, il tunneling viene usato per far transitare un pacchetto IPv6 all'interno di una rete che gestisce solamente IPv4. Ciò significa che il pacchetto IPv6 viene incapsulato con un'intestazione IPv4 avente come indirizzo di destinazione il punto finale del tunnel, tale pacchetto viene trasmesso sulla rete IPv4 e viene estratto al termine del tunnel. Il punto terminale del tunnel potrebbe essere un router o un host, ma in ogni caso deve essere in grado di elaborare il pacchetto IPv6 dopo che questo è stato estratto. Se tale nodo terminale è un host con un indirizzo IPv6 trasformato in IPv4, allora il tunneling può avvenire automaticamente, estraendo l'indirizzo IPv4 dall'indirizzo IPv6 e usando-lo in tale forma nell'intestazione IPv4; altrimenti, il tunnel necessita di una configurazione manuale, perché il nodo che effettua l'incapsulamento deve conoscere l'indirizzo IPv4 del nodo che si trova all'altra estremità del tunnel, indirizzo che non può essere estratto dall'intestazione IPv6. Dal punto di vista di IPv6, l'uscita del tunnel non è altro che un normale nodo IPv6 che si trova ad un solo hop di distanza, nonostante possano esistere molti hop all'interno dell'infrastruttura IPv4 fra i due estremi del tunnel.

Notazione per gli indirizzi

Proprio come in IPv4, esiste una notazione standard per scrivere gli indirizzi IPv6; tale rappresentazione è $x:x:x:x:x:x$, dove ciascuna "x" è la rappresentazione esadecimale di un segmento di 16 bit dell'indirizzo. Un esempio potrebbe essere:

47CD:1234:4422:AC02:0022:1234:A456:0124

Qualsiasi indirizzo IPv6 può essere scritto usando questa notazione, ma esistono alcune notazioni speciali, utili in alcune circostanze, dal momento che esistono alcuni tipi speciali di indirizzi IPv6. Ad esempio, un indirizzo con un gran numero di zeri contigui può essere scritto in modo più compatto omettendo i campi che valgono zero. Così

47CD:0000:0000:0000:0000:0000:A456:0124

si può scrivere

47CD::A456:0124

Ovviamente, questa forma di abbreviazione può essere utilizzata per un unico insieme di zeri contigui all'interno di un indirizzo, per evitare ambiguità.

Poiché esistono due tipi di indirizzi IPv6 che contengono un indirizzo IPv4 al loro interno, essi hanno una notazione speciale che semplifica l'estrazione dell'indirizzo IPv4. Ad esempio, l'"indirizzo IPv6 trasformato in IPv4" di un host il cui indirizzo IPv4 sia 128.96.33.81, si potrebbe scrivere in questo modo

::FFFF:128.96.33.81 → Notazioni x multietati IPv4

In pratica, gli ultimi 32 bit vengono scritti secondo la notazione IPv4, invece che come copia di numeri esadecimali separati da un carattere "due punti". Notate che il doppio carattere "due punti" iniziale sta ad indicare gli zeri iniziali.

Aggregatable Global Unicast Addresses

Per il momento, la cosa più importante che deve essere garantita da IPv6 nel momento in cui viene installato è il normale indirizzamento unicast, ma deve farlo in un modo che non ostacoli il rapido aumento di nuovi host-in Internet e consenta di effettuare l'instradamento in modo scalabile all'aumentare del numero di reti fisiche in Internet. Di conseguenza, il cuore di IPv6 è il piano di allocazione degli indirizzi unicast, che determina come gli indirizzi con il prefisso 001 vengano assegnati ai fornitori di servizi, ai sistemi autonomi, alle reti, agli host e ai router.

In pratica, il piano di assegnazione degli indirizzi proposto per gli indirizzi unicast di IPv6 è molto simile a quello sviluppato con CIDR in IPv4. Per capire come funziona e come possa agevolare la scalabilità, è utile definire alcuni nuovi termini. Possiamo immaginare che un AS non di transito (cioè uno stub o un AS multihomed) sia un *sottoscrittore (subscriber)* di servizi, mentre un AS di transito sia un *fornitore (provider)*. Inoltre, suddividiamo i fornitori in *diretti* e *indiretti*: i primi sono direttamente connessi a sottoscrittori, mentre i secondi connettono principalmente altri fornitori, non sono connessi direttamente a sottoscrittori e prendono solitamente il nome di *reti backbone*.

Con questo insieme di definizioni, possiamo vedere come Internet non sia soltanto un insieme arbitrariamente interconnesso di sistemi autonomi, ma abbia invece una sorta di gerarchia intrinseca. La difficoltà consiste nel far uso di questa gerarchia senza usare strategie che falliscano quando la gerarchia non viene rigidamente rispettata, come accade con EGP. Ad esempio, la distinzione fra fornitori diretti e indiretti diviene incerta quando un sottoscrittore si connette ad un backbone, oppure quando un fornitore diretto inizia a connettersi a molti altri fornitori.

Come per CIDR, l'obiettivo del piano di assegnazione degli indirizzi IPv6 è quello di aggregare le informazioni di instradamento per semplificare il lavoro dei router intradominio. Anche qui, l'idea chiave consiste nell'utilizzo di un prefisso all'interno dell'indirizzo (un insieme di bit contigui nella parte più significativa dell'indirizzo), per aggregare le informazioni di raggiungibilità relative ad un gran numero di reti ed anche ad un gran numero di AS. Lo schema principale mediante il quale si ottiene ciò consiste nell'assegnare un prefisso di indirizzi ad un fornitore diretto, il quale assegnerà ai propri sottoscrittori prefissi più lunghi

4.3 La rete Internet globale

che inizino con tale prefisso, esattamente come abbiamo visto in Figura 4.27. In questo modo un fornitore può pubblicizzare un singolo prefisso per tutti i propri sottoscrittori.

Ovviamente, lo svantaggio consiste nel fatto che se un sito decide di cambiare fornitore, dovrà ottenere un nuovo prefisso di indirizzi e rinumerare tutti i nodi del proprio sito: ciò costituirebbe uno sforzo colossale, tale da dissuadere la maggior parte delle persone dal cambiare fornitore. Per questa ragione continua la ricerca di schemi di indirizzamento alternativi, come l'indirizzamento geografico, nel quale l'indirizzo di un sito è funzione della sua posizione piuttosto che del fornitore a cui si connette. Al momento, però, l'indirizzamento basato sui fornitori appare necessario per rendere efficiente l'instradamento.

Notate che, mentre l'assegnamento degli indirizzi IPv6 è sostanzialmente equivalente al modo in cui viene gestito l'assegnamento degli indirizzi in IPv4 dopo l'introduzione di CIDR, IPv6 ha il significativo vantaggio di non avere una grande base di apparecchiature installate con indirizzi già assegnati che debbano essere gestiti dal piano di assegnamento.

Ci si può chiedere se abbia senso trattare l'aggregazione gerarchica ad altri livelli nella gerarchia. Ad esempio, ha senso che tutti i fornitori ottengano i propri prefissi per indirizzi dal prefisso assegnato al backbone a cui si connettono? Dal momento che la maggior parte dei fornitori si connette a più backbone, ciò probabilmente non ha senso. Inoltre, dato che il numero di fornitori è molto minore del numero di siti, i vantaggi dell'aggregazione a questo livello è molto inferiore.

Un luogo dove l'aggregazione potrebbe aver senso è al livello nazionale o continentale. I confini continentali costituiscono divisioni naturali nella topologia di Internet e se tutti gli indirizzi dell'Europa, ad esempio, avessero un prefisso comune, si potrebbe compiere un grosso lavoro di aggregazione, in modo che la maggior parte dei router di altri continenti dovrebbero avere un solo dato nelle proprie tabelle di instradamento per tutte le reti aventi il prefisso europeo. I fornitori europei sceglierebbero i loro prefissi in modo che inizino con il prefisso europeo. Usando questo schema, un indirizzo IPv6 potrebbe assomigliare a quello rappresentato in Figura 4.33. Il campo RegistryID potrebbe essere un identificativo assegnato ad un gestore di indirizzi europeo, con valori diversi assegnati agli altri continenti o nazioni. Notate che i prefissi, in questo scenario, sarebbero di lunghezze diverse; ad esempio, un fornitore con pochi sottoscrittori potrebbe avere un prefisso più lungo (e, quindi, uno spazio di indirizzamento totale più piccolo) di uno con molti clienti.

Potrebbe accadere un problema strano quando un sottoscrittore fosse connesso a più di un fornitore. Quale prefisso dovrebbe usare il sottoscrittore nel proprio sito? Non esiste una soluzione perfetta per questo problema. Ad esempio, supponete che un sottoscrittore sia connesso a due fornitori, X e Y. Se il sottoscrittore riceve il proprio prefisso da X, allora Y deve pubblicizzare un prefisso che non ha niente a che vedere con quello degli altri suoi sottoscrittori e che, di conseguenza, non può essere aggregato. Se il sottoscrittore numera parte del proprio AS con il prefisso di X e parte con il prefisso di Y, corre il rischio di avere una parte del proprio sito irraggiungibile nel momento in cui si guasti la connessione verso uno dei fornitori. Una soluzione che funziona abbastanza bene se X e Y hanno molti sottoscrittori in comune consiste nell'avere tre prefissi: uno per i sottoscrittori che sono clienti soltanto di X, uno per quelli che sono clienti soltanto di Y e uno per quei siti che sono sottoscrittori di entrambi.

3	c	m	n	o	p	125-m-n-o-p
010	RegistryID	ProviderID	SubscriberID	SubnetID	InterfaceID	

Figura 4.33 Un indirizzo unicast IPv6 basato sul fornitore.

Il formato del pacchetto

Nonostante IPv6 estenda IPv4 in molti aspetti, il formato della sua intestazione è in realtà più semplice. Questa semplicità è dovuta ad uno sforzo congiunto che è andato nella direzione di eliminare dal protocollo funzionalità non necessarie. La Figura 4.34 mostra il risultato (per fare un confronto con IPv4, osservate il formato dell'intestazione mostrata in Figura 4.3).

Come accade in molte intestazioni, anche questa inizia con un campo Version, che assume il valore 6 in IPv6. Il campo Version si trova nella stessa posizione in cui si trova il campo Version nell'intestazione di IPv4, per cui il software che elabora le intestazioni può decidere immediatamente di quale versione si tratti: I campi TrafficClass e FlowLabel servono entrambi alla qualità di servizio, come vedremo nella Sezione 6.5.

Il campo PayloadLen contiene la lunghezza del pacchetto misurata in byte, con l'esclusione dell'intestazione IPv6. Il campo NextHeader sostituisce efficientemente sia le opzioni IP sia il campo Protocol di IPv4: se servono opzioni, queste vengono inserite in una (o più) intestazioni speciali che seguono l'intestazione IP, come segnalato dal valore del campo NextHeader. Se non ci sono intestazioni speciali, il campo NextHeader è, invece, una chiave di demultiplexing che identifica il protocollo di livello più elevato che viene eseguito al di sopra di IP (ad esempio, TCP o UDP), serve cioè allo stesso scopo del campo Protocol di IPv4. Ancora, la frammentazione viene ora gestita da un'intestazione opzionale, per cui i campi relativi alla frammentazione, presenti in IPv4, non sono presenti nell'intestazione IPv6. Il campo HopLimit è semplicemente il campo TTL di IPv4, ridenominato per tener conto del modo in cui viene effettivamente usato.

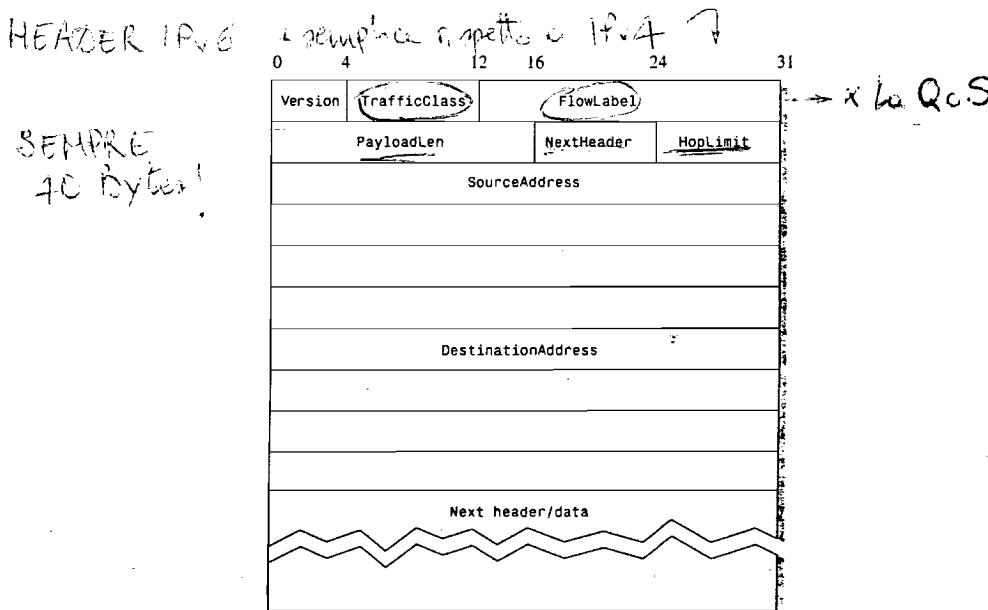


Figura 4.34 L'intestazione del pacchetto IPv6.

4.3 La rete Internet globale

Infine, la parte centrale dell'intestazione è occupata dagli indirizzi del mittente e del destinatario, ciascuno dei quali è lungo 16 byte (128 bit). Di conseguenza, l'intestazione di IPv6 è sempre lunga 40 byte. Considerando che gli indirizzi IPv6 sono quattro volte più lunghi di quelli di IPv4, questa dimensione regge bene il confronto con quella dell'intestazione di IPv4, che, in assenza di opzioni, è lunga 20 byte.

Il modo in cui IPv6 gestisce le opzioni costituisce un certo miglioramento rispetto ad IPv4, dove, se vi erano opzioni, ciascun router doveva analizzare completamente il campo delle opzioni per capire se ve ne fossero di rilevanti per svolgere il proprio compito, perché tutte le opzioni erano collocate al termine dell'intestazione IP, sotto forma di un insieme disordinato di valori del tipo (tipo, lunghezza, valore). Al contrario, IPv6 tratta le opzioni come estensioni dell'intestazione che devono, se presenti, figurare in un ordine predefinito. Ciò significa che ciascun router può rapidamente determinare se vi siano opzioni rilevanti: nella maggior parte dei casi non ve ne saranno e solitamente ciò può essere determinato guardando semplicemente il campo NextHeader. Il risultato finale è che in IPv6 l'elaborazione delle opzioni è molto più efficiente, e questo costituisce un fattore importante per le prestazioni dei router. Inoltre, considerare le opzioni come estensioni dell'intestazione ha come conseguenza il fatto che la loro dimensione sia arbitraria, mentre in IPv4 vi era un limite massimo di 44 byte. Vedremo ora come vengano usate alcune di queste opzioni.

Ciascuna opzione ha il proprio tipo di estensione dell'intestazione, che è identificato dal valore del campo NextHeader nell'intestazione precedente; ciascuna intestazione contiene, a sua volta, un campo NextHeader che identifica l'intestazione seguente. L'ultima estensione dell'intestazione sarà seguita da un'intestazione dello strato di trasporto (es. TCP) e, in questo caso, il valore del campo NextHeader è uguale al valore del campo Protocol dell'intestazione IPv4. Di conseguenza, il campo NextHeader svolge un doppio ruolo: identifica il tipo di estensione che segue, oppure, nell'ultima intestazione di estensione, serve come chiave di demultiplexing per identificare il protocollo di livello superiore in esecuzione al di sopra di IPv6.

Considerate l'esempio dell'intestazione di frammentazione, mostrata in Figura 4.35. Questa intestazione fornisce funzionalità simili ai campi che gestiscono la frammentazione nell'intestazione IPv4 descritti nella Sezione 4.1.2, ma è presente soltanto se la frammentazione è necessaria. Nell'ipotesi che sia l'unica intestazione di estensione presente, il campo NextHeader dell'intestazione IPv6 conterebbe il valore 44, che è il valore che sta ad indicare l'intestazione di frammentazione. Il campo NextHeader dell'intestazione di frammentazione contiene, invece, un valore che descrive l'intestazione seguente: ipotizzando nuovamente che non vi siano altre intestazioni di estensione, l'intestazione successiva potrebbe essere un'intestazione TCP, per cui il campo NextHeader conterebbe il valore 6, proprio come avrebbe fatto il campo Protocol in IPv4. Se l'intestazione di frammentazione fosse seguita invece da, per esempio, un'intestazione di autenticazione, allora il campo NextHeader dell'intestazione di frammentazione avrebbe contenuto il valore 51.

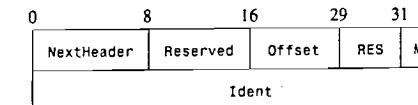


Figura 4.35 L'intestazione dell'estensione di frammentazione in IPv6.

Autoconfigurazione

Nonostante la crescita di Internet sia stata impressionante, un fattore che ha limitato un'acceptazione più rapida della tecnologia è la notevole quantità di amministrazione sistemistica richiesta per connettersi ad Internet. In particolare, ogni host connesso ad Internet deve essere configurato con una quantità minima di informazioni, tra cui un indirizzo IP valido, una maschera di sottorete per la linea a cui è connesso e l'indirizzo di un server per la risoluzione dei nomi. In sostanza, non è stato possibile fino ad ora togliere un nuovo calcolatore dalla propria confezione e connetterlo ad Internet senza alcune configurazioni preliminari. Uno degli obiettivi di IPv6, quindi, è quello di fornire supporto alla autoconfigurazione, a volte definita come operatività "plug-and-play" (inserisci la spina e opera).

Come abbiamo visto nella Sezione 4.1.6, l'autoconfigurazione è possibile anche in IPv4, ma dipende dall'esistenza di un server che sia configurato per consegnare gli indirizzi ed altre informazioni di configurazione ai client DHCP. Il più lungo formato dell'indirizzo di IPv6 consente una nuova e comoda forma di autoconfigurazione, denominata autoconfigurazione *stateless* (priva di informazioni di stato), che non richiede un server.

Ricordate che gli indirizzi unicast in IPv6 sono gerarchici, e che la parte meno significativa è l'identificativo dell'interfaccia. Di conseguenza, possiamo suddividere il problema dell'autoconfigurazione in due parti:

1. Ottenere un identificativo di interfaccia che sia unico sulla linea di collegamento a cui è connesso l'host.
2. Ottenere il corretto prefisso per gli indirizzi della sottorete.

La prima parte risulta essere piuttosto semplice, perché ogni host deve avere un indirizzo univoco a livello di linea di collegamento. Ad esempio, tutti gli host di una Ethernet hanno un indirizzo univoco a 48 bit, che può diventare un indirizzo valido per la linea locale aggiungendo il prefisso appropriato, indicato nella Tabella 4.11 (1111 1110 10), seguito da un numero di zeri sufficiente a formare un indirizzo di 128 bit. Per alcuni dispositivi, come ad esempio le stampanti o gli host di una piccola rete priva di router che non sia connessa ad alcuna altra rete, questo indirizzo può essere perfettamente adeguato. Quei dispositivi che necessitano, invece, di avere un indirizzo valido globalmente sono dipendenti dal fatto che un router presente sulla stessa linea pubblicizzi periodicamente il prefisso corretto per tale linea: ciò richiede, ovviamente, che il router sia configurato con il corretto prefisso per gli indirizzi e che tale prefisso venga scelto in modo che vi sia spazio sufficiente alla fine (per esempio 48 bit) per inserire l'indirizzo dello strato di linea di collegamento adeguato.

La capacità di includere indirizzi dello strato di linea di collegamento lunghi anche 48 bit all'interno degli indirizzi IPv6 è stata una delle ragioni per cui si è scelto una dimensione dell'indirizzo così grande. La dimensione di 128 bit non soltanto consente tale inclusione, ma lascia anche parecchio spazio libero per la gerarchia multilivello dell'indirizzamento di cui abbiamo parlato in precedenza.

Capacità di instradamento avanzate

Un'altra delle intestazioni di estensione di IPv6 è l'intestazione di instradamento. In assenza di tale intestazione, l'instradamento in IPv6 differisce assai poco da quello di IPv4 con CIDR. L'intestazione di instradamento contiene un elenco di indirizzi IPv6 che rappresentano nodi o aree topologiche che il pacchetto dovrebbe visitare lungo il suo percorso verso la destinazione. Un'area topologica può essere, ad esempio, un rete di un fornitore di tipo backbone.

4.3 La rete Internet globale

Traduzione dell'indirizzo di rete (NAT, Network Address Translation)

Mentre veniva sviluppato IPv6 per risolvere il problema dell'esaurimento dello spazio di indirizzamento dovuto all'aumentato utilizzo del protocollo IP, è divenuta popolare un'altra tecnologia che consente di risparmiare indirizzi IP. Tale tecnologia è denominata traduzione dell'indirizzo di rete (NAT, Network Address Translation) ed è possibile che la sua ampia diffusione provochi un significativo rallentamento nell'utilizzo di IPv6. La tecnologia NAT viene spesso vista come "impura" dal punto di vista architettonale, ma questo è un fatto che nelle reti può essere tranquillamente ignorato.

L'idea che sta alla base di NAT è il fatto che tutti gli host che possono comunicare tra loro attraverso Internet non hanno bisogno di un indirizzo unico globalmente. Al contrario, è possibile assegnare ad un host un "indirizzo privato" che non sia necessariamente unico globalmente, ma che sia unico in un ambito più ristretto (ad esempio, all'interno della rete aziendale in cui si trova l'host). Il numero 10, rete di classe A, viene spesso utilizzato per questo scopo, perché tale numero di rete era stato assegnato ad ARPANET e non viene più utilizzato come indirizzo unico su scala globale. Fintanto che l'host comunica solamente con altri host interni alla rete aziendale, un indirizzo localmente unico è sufficiente. Se l'host dovesse voler comunicare con un host esterno alla rete aziendale, lo farebbe tramite un "dispositivo di traduzione degli indirizzi di rete" (NAT box), un dispositivo in grado di tradurre l'indirizzo privato usato dall'host in un indirizzo unico su scala globale che è stato assegnato al dispositivo stesso. Poiché è probabile che un piccolo sottoinsieme degli host dell'azienda richieda contemporaneamente i servizi del NAT, può darsi che il dispositivo di traduzione sia in grado di svolgere il proprio compito usando un piccolo insieme di indirizzi globalmente unici, un numero di indirizzi molto minore del numero di indirizzi che sarebbero necessari se ci fossero host dell'azienda avesse un indirizzo globalmente unico.

Possiamo quindi immaginare che un dispositivo NAT riceva pacchetti IP da un host interno all'azienda e che traduca l'indirizzo IP del mittente da un indirizzo privato (ad esempio, 10.0.1.5) ad un indirizzo unico su scala globale (ad esempio, 171.69.210.246). Quando i pacchetti ritornano dall'host remoto e sono indirizzati a 171.69.210.246, il dispositivo NAT traduce l'indirizzo di destinazione in 10.0.1.5 e inoltra il pacchetto verso l'host.

Il principale svantaggio della tecnologia NAT è che invalida un'ipotesi chiave del modello di servizio del protocollo IP: tutti i nodi hanno indirizzi unici su scala globale. Si verifica che molte applicazioni e protocolli sfruttano questa ipotesi al punto da esserne dipendenti; in particolare, molti protocolli che vengono eseguiti al di sopra di IP (ad esempio, protocolli applicativi) contengono indirizzi IP nei loro messaggi. Questi indirizzi devono essere, anch'essi, tradotti dal dispositivo NAT per far funzionare in modo corretto il protocollo di alto livello, per cui tali dispositivi diventano molto più complessi di un semplice manipolatore di intestazioni IP. Devono, potenzialmente, essere in grado di interpretare un numero sempre crescente di protocolli di alto livello. Questo fenomeno, a sua volta, rappresenta un ostacolo allo sviluppo e all'installazione di nuove applicazioni. Probabilmente è corretto dire che le reti sarebbero migliori senza NAT, ma la sua scomparsa sembra essere improbabile. Un'ampia diffusione di IPv6 sarebbe quasi certamente di aiuto.

Specificare che un pacchetto deve transitare attraverso una certa rete sarebbe un modo per implementare la selezione del fornitore per ciascun pacchetto. In questo modo un host potrebbe indicare di voler inviare i propri pacchetti verso un particolare fornitore che risulta essere economico, mentre altri potrebbero voler utilizzare fornitori che garantiscono elevata affidabilità, e altri ancora fornitori di cui ci si fida per motivi di sicurezza.

Per poter specificare entità topologiche piuttosto che singoli nodi, IPv6 definisce un indirizzo di tipo *anycast*, che viene assegnato ad un insieme di interfacce: i pacchetti inviati a tale indirizzo raggiungeranno la "più vicina" di tali interfacce, secondo quanto determinato dai protocolli di instradamento. Ad esempio, a tutti i router di un fornitore di tipo backbone potrebbe essere assegnato un singolo indirizzo anycast, che verrebbe utilizzato nelle intestazioni di instradamento.

L'indirizzo anycast e l'intestazione di instradamento potranno essere usati anche per fornire un migliore supporto all'instradamento per host mobili, ma i meccanismi per fornire tale supporto non sono ancora stati definiti nel dettaglio.

Altre caratteristiche

Come accennato all'inizio di questa sezione, la motivazione principale che sta alla base dello sviluppo di IPv6 è il supporto alla continua crescita di Internet. Una volta deciso che si doveva modificare l'intestazione IP a causa degli indirizzi, tuttavia, si è spalancata la porta per un'ampia varietà di altre modifiche, due delle quali sono state appena descritte, l'autoconfigurazione e l'instradamento pilotato dal mittente. Il protocollo IPv6 contiene parecchie altre caratteristiche, la maggior parte delle quali verrà trattata in altre parti di questo libro: la mobilità è stata discussa nella Sezione 4.2.5, la sicurezza delle reti sarà argomento del Capitolo 8, mentre un nuovo modello di servizio proposto per Internet verrà descritto nella Sezione 6.5. È interessante notare come nella maggior parte di queste aree le capacità di IPv4 e di IPv6 siano divenute virtualmente indistinguibili, per cui la spinta maggiore per IPv6 rimane il bisogno di indirizzi più lunghi.

4.4 Multicast

Come abbiamo visto nel Capitolo 2, le reti ad accesso multiplo come Ethernet e token ring realizzano il multicast in hardware; questa sezione, invece, descrive come estendere, tramite software, il concetto di multicast all'interconnessione di reti. L'approccio che viene qui descritto è basato sull'attuale implementazione del multicast in Internet (IPv4); il multicast sarà possibile anche nella futura generazione del protocollo IP (IPv6) e le differenze più significative saranno confinate nel formato dell'indirizzo.

La motivazione per lo sviluppo del multicast consiste nel fatto che esistono applicazioni che vogliono inviare un pacchetto a più di un host di destinazione. Invece di costringere l'host mittente ad inviare pacchetti diversi a ciascun host di destinazione, vogliamo consentire al mittente di inviare un unico pacchetto ad un *indirizzo multicast*, lasciando alla rete (ad una internetwork, in questo caso) il compito di consegnare una copia del pacchetto a ciascun host appartenente ad un gruppo. I singoli host possono decidere di entrare a far parte di un gruppo o di uscirne autonomamente, senza alcuna sincronizzazione o negoziazione con gli altri membri del gruppo. Ancora, un host può appartenere a più gruppi contemporaneamente.

Il multicast in una internetwork può essere realizzato, per un insieme di reti che forniscono supporto hardware al multicast (o al broadcast), estendendo le funzioni di instradamento

Multicast

e inoltro svolte dai router che connettono queste reti. Questa sezione descrive tre di tali estensioni: la prima si basa sull'instradamento a vettore di distanza descritto nella Sezione 4.2.2; la seconda è basata sull'instradamento a stato delle linee descritto nella Sezione 4.2.3; la terza può essere realizzata con qualsiasi protocollo di instradamento sottostante e viene quindi denominata *Protocol Independent Multicast (PIM)*.

Prima di esaminare i protocolli di instradamento multicast, tuttavia, dobbiamo prendere in esame il modello di servizio del multicast in IP. Potremmo immaginare che un host che voglia inviare un pacchetto ad un certo numero di host di una internetwork possa elencare tutti i loro indirizzi, ma questa soluzione diventerebbe rapidamente non scalabile per grandi numeri di riceventi: considerate, ad esempio, l'utilizzo di Internet per distribuire un film in modalità *pay-per-view*. Per questa ragione il multicast IP sfrutta l'idea di un *gruppo multicast* a cui i riceventi si aggregano. A ciascun gruppo viene assegnato uno speciale indirizzo, che viene utilizzato come indirizzo di destinazione da chi voglia inviare pacchetti a tale gruppo. In IPv4, questi indirizzi sono assegnati all'interno dello spazio di indirizzamento di classe D, e anche IPv6 ha una parte del proprio spazio di indirizzamento riservata agli indirizzi dei gruppi multicast.

Gli host entrano in gruppi multicast usando un protocollo denominato Internet Group Management Protocol (IGMP), che usano per avvertire un router sulla propria rete locale che desiderano ricevere pacchetti inviati ad un certo gruppo multicast. I protocolli descritti in seguito si occupano di come i pacchetti vengano distribuiti ai router giusti, mentre la consegna dei pacchetti dal "router dell'ultimo salto" (*last hop router*) all'host viene gestita dalle capacità multicast della rete sottostante, come descritto nella Sezione 2.6.

Una cosa che suscita perplessità è il modo in cui i mittenti e i destinatari apprendano gli indirizzi multicast: ciò viene normalmente gestito con strumenti esterni ed esistono strumenti abbastanza sofisticati per consentire agli indirizzi di gruppo di essere pubblicizzati su Internet.

4.4.1 Multicast a stato delle linee

Aggiungere le potenzialità multicast all'algoritmo di instradamento a stato delle linee è abbastanza elementare, per cui lo descriviamo per primo. Ricordate che nell'instradamento a stato delle linee ciascun router tiene sotto controllo lo stato delle linee di collegamento ad esso direttamente connesse ed invia messaggi di aggiornamento a tutti gli altri router quando lo stato si modifica. Poiché ciascun router riceve sufficienti informazioni per ricostruire l'intera topologia della rete, è in grado di utilizzare l'algoritmo di Dijkstra per calcolare l'albero di copertura del percorso più breve avente se stesso come radice e che raggiunga qualsiasi destinazione possibile. Il router usa poi tale albero per determinare il miglior salto successivo per ciascun pacchetto che deve inoltrare.

Tutto ciò che dobbiamo fare per estendere questo algoritmo in modo che fornisca supporto per le trasmissioni multicast è aggiungere l'insieme di gruppi che hanno membri in una particolare LAN (o linea di collegamento) allo "stato" di tale linea; l'unica domanda che attende ancora una risposta è come faccia un router a determinare quali gruppi hanno membri in ciascuna linea. Come suggerito nella Sezione 3.2.3, la soluzione consiste nel fare in modo che ogni host annuci periodicamente sulla LAN i gruppi a cui appartiene: il router controlla semplicemente tali messaggi che vengono trasmessi sulla LAN. Se tali messaggi non arrivano più per un certo periodo di tempo, il router deduce che l'host abbia abbandonato il gruppo.

Data la conoscenza completa dei gruppi che hanno membri che appartengono a ciascuna linea, ogni router è in grado di calcolare l'*albero con i percorsi minimi per il multicast* a

partire da ogni sorgente e verso ogni gruppo, usando nuovamente l'algoritmo di Dijkstra. Ad esempio, data la internetwork di Figura 4.36, dove gli host evidenziati appartengono al gruppo G, i router calcolerebbero gli alberi presentati in Figura 4.37 per le sorgenti A, B e C. I router userebbero poi questi alberi per decidere come inoltrare i pacchetti indirizzati al gruppo multicast G. Ad esempio, il router R3 inoltrerebbe verso R6 un pacchetto spedito da A al gruppo G.

Ricordate che ciascun router deve conservare, in teoria, un diverso albero per ciascuna sorgente che voglia spedire a ciascun gruppo, un approccio ovviamente troppo costoso, per cui i router calcolano e memorizzano soltanto quegli alberi che corrispondono a coppie sorgente/gruppo che siano attive.

4.4.2 Multicast a vettore di distanza

Aggiungere le potenzialità multicast all'algoritmo di instradamento a vettore di distanza è un po' più difficile, perché i router non conoscono l'intera topologia della internetwork. Al contrario, ciascun router conserva una tabella di informazioni del tipo $\langle \text{Destination}, \text{Cost}, \text{NextHop} \rangle$ e scambia con i vicini ad esso direttamente connessi elenchi di coppie $\langle \text{Destination}, \text{Cost} \rangle$. Estendere questo algoritmo per fornire supporto alla trasmissione multicast è un processo che richiede due passi successivi. Prima di tutto, dobbiamo progettare un meccanismo broadcast che consenta di inoltrare pacchetti a tutte le reti della internetwork. Successivamente dovremo migliorare questo meccanismo in modo che elimini le reti che non hanno host appartenenti al gruppo multicast.

Broadcast a ritroso (RPB, Reverse-Path Broadcast)

Ogni router sa che, in un dato momento, il percorso più breve verso una certa destinazione passa per il corrispondente NextHop. Di conseguenza, ogni volta che riceve un pacchetto multicast dalla sorgente S, il router lo inoltra verso tutte le linee di uscita (tranne quella da cui

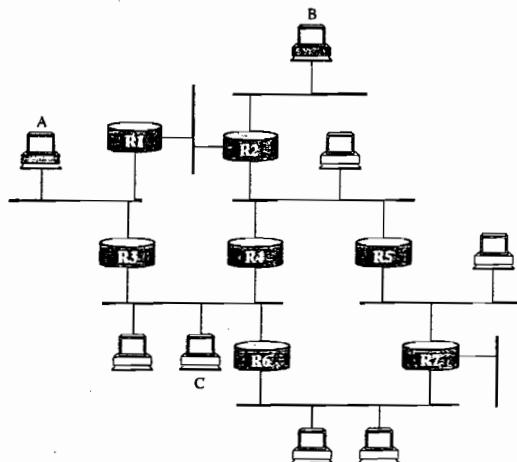


Figura 4.36 Esempio di internetwork con i membri del gruppo G in evidenza.

4.4 Multicast

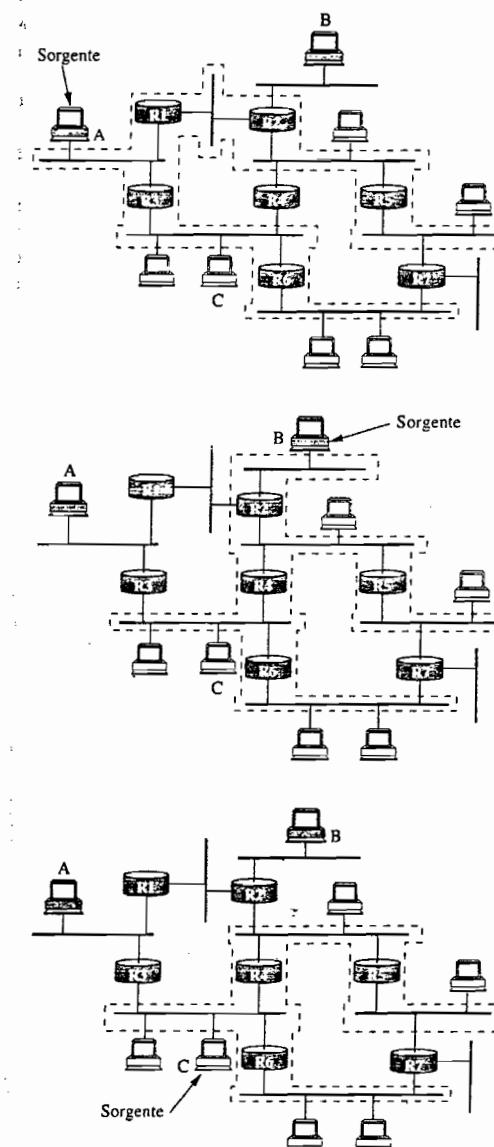


Figura 4.37 Esempio di alberi con i percorsi minimi per il multicast.

il pacchetto è arrivato) se e solo se il pacchetto è arrivato lungo la linea che si trova sul percorso più breve che porta a S (cioè il pacchetto proviene dal NextHop associato ad S nella tabella di instradamento). Questa strategia provoca un'efficiente inondazione di pacchetti a partire da S, senza far ritornare i pacchetti verso S.

Questo approccio ha due svantaggi fondamentali. Prima di tutto, inonda veramente la rete, perché non prevede alcun meccanismo che eviti le LAN che non hanno membri appartenente al gruppo multicast: ci occuperemo di questo problema nella sottosezione seguente. Il secondo limite consiste nel fatto che un pacchetto sarà inoltrato verso una certa LAN da tutti i router connessi a tale LAN, per effetto della strategia che prevede di inviare i pacchetti verso tutte le linee diverse da quella da cui è giunto il pacchetto, senza considerare se tali linee facciano parte o meno dell'albero di percorso minimo avente la sorgente come radice.

La soluzione a questa seconda limitazione consiste nell'eliminazione dei pacchetti broadcast duplicati che vengono generati quando ad una LAN sono connessi più router. Un modo per far questo prevede la designazione, per ciascuna linea, di un router come "genitore" della linea stessa, relativamente ad una certa sorgente, in modo che soltanto il router genitore possa inoltrare nella LAN pacchetti multicast che provengono da quella sorgente. Viene selezionato come genitore il router che ha verso la sorgente S il percorso di lunghezza minore, risolvendo i casi di parità assegnando la precedenza al router avente l'indirizzo minore. Un router viene a sapere di essere il genitore di una LAN (sempre relativamente a ciascuna possibile sorgente) in base ai messaggi contenenti vettori di distanza scambiati con i propri vicini.

Noteate che questo miglioramento richiede che ogni router memorizzi, per ogni sorgente, un bit per ogni linea a cui è connesso, per segnalare il fatto di essere o meno il genitore per una coppia sorgente/linea. Ricordate anche che, quando si parla di internetwork, una "sorgente" è una rete, non un host, perché un router di internetwork ha soltanto il compito di inoltrare pacchetti fra reti. Il meccanismo qui presentato viene a volte chiamato "broadcast a ritroso" (RPB, Reverse-Path Broadcast).

Multicast a ritroso (RPM, Reverse-Path Multicast)

La strategia RPB realizza il broadcast lungo il percorso più breve. Vogliamo ora ridurre le dimensioni dell'insieme di reti che ricevono ciascun pacchetto che sia indirizzato al gruppo G, escludendo quelle reti che non hanno host appartenenti al gruppo G, problema che si può risolvere in due fasi. Dapprima, dobbiamo capire quando una rete *foglia* non contiene membri del gruppo: determinare che una rete sia una foglia è facile: se il router genitore, secondo quanto descritto a proposito di RPB, è l'unico router della rete, allora la rete è una foglia. Per determinare se la rete contiene membri di un gruppo, ciascun host che sia membro del gruppo G rende periodicamente nota questa informazione sulla rete, come descritto nella nostra precedente discussione del multicast mediante stato delle linee. Il router usa poi queste informazioni per decidere se inoltrare o meno verso una certa LAN un pacchetto multicast indirizzato a G.

Il secondo passo consiste nella propagazione di questa informazione, "qui non ci sono membri di G", verso la radice dell'albero di percorso minimo. Per fare ciò, il router aggiunge alla coppia *(Destination, Cost)* che invia ai propri vicini l'insieme dei gruppi per i quali la rete foglia è interessata a ricevere pacchetti multicast. Questa informazione può, poi, essere propagata da un router all'altro, in modo che, per ciascuna delle proprie linee, ogni router sappia quali pacchetti multicast inoltrare.

Noteate che inserire tutte queste informazioni negli aggiornamenti di instradamento è una cosa abbastanza onerosa, per cui, in pratica, queste informazioni vengono scambiate

soltanto quando una sorgente inizia ad inviare pacchetti ad un gruppo. In altre parole, la strategia prevede di utilizzare RPB, che aggiunge soltanto poche informazioni all'algoritmo standard a vettore di distanza, finché un particolare indirizzo multicast non diviene attivo. In quel momento, i router che non sono interessati a ricevere pacchetti indirizzati a quel gruppo iniziano a trasmettere questo tipo di informazione, che viene poi propagata agli altri router.

4.4.3 Multicast indipendente dal protocollo (PIM)

La strategia PIM è stata sviluppata per rispondere ai problemi di scalabilità degli esistenti protocolli di instradamento per il multicast. In particolare, si è visto che i protocolli esistenti non sono ben scalabili in ambienti in cui una parte relativamente modesta di router vuole ricevere traffico per un certo gruppo. Ad esempio, inviare il traffico in modalità broadcast a tutti i router finché non chiedono esplicitamente di essere eliminati da tale distribuzione non è una buona scelta di progetto nel caso in cui la maggior parte dei router non voglia ricevere il traffico fin dall'inizio. Questa situazione è così comune che la strategia PIM suddivide il problema in "modalità sparsa" e "modalità densa": dato che i protocolli esistenti sono assai poco adatti a gestire la modalità sparsa, è quest'ultima modalità di PIM ad avere ricevuto le maggiori attenzioni e su di essa si concentrerà la nostra discussione.

Nella modalità sparsa di PIM (PIM-SM), i router entrano ed escono esplicitamente dai gruppi multicast usando, rispettivamente, i messaggi del protocollo PIM noti come *Join* e *Prune*. La domanda che nasce è: dove vanno inviati questi messaggi? Per risolvere questo problema, PIM assegna a ciascun gruppo un *punto d'incontro* (RP, *rendezvous point*). In generale, in un dominio un certo numero di router vengono configurati per essere candidati a svolgere il ruolo di RP e PIM definisce un insieme di procedure mediante le quali tutti i router di un dominio possono accordarsi su quale router usare come RP di un certo gruppo. Queste procedure sono piuttosto complesse, perché devono gestire molti diversi scenari, come il guasto di un candidato a svolgere il ruolo di RP e la suddivisione di un dominio in due reti separate per effetto di un certo numero di guasti di linee o di nodi. Nel seguito, supporremo che tutti i router del dominio conoscano l'indirizzo IP unicast del router che svolge il ruolo di RP per un certo gruppo.

Come risultato dell'invio di messaggi *Join* all'RP da parte dei router, viene costruito un albero per l'inoltro multicast. La strategia PIM-SM consente la costruzione di due tipi di alberi: un albero *condiviso*, che viene usato da tutti i mittenti, e un albero *specifico per una sorgente*, che può essere usato da un unico host nel ruolo di mittente. La normale modalità operativa crea prima l'albero condiviso, poi uno o più alberi specifici se si crea abbastanza traffico da giustificare l'esistenza. Poiché la costruzione degli alberi aggiunge informazioni di stato che devono essere memorizzate nei router lungo l'albero, è importante che la situazione di default sia quella che prevede la costruzione di un unico albero per ogni gruppo, e non uno per ogni mittente che invia pacchetti ad un gruppo.

Quando un router invia un messaggio *Join* verso il router RP di un gruppo G, ciò avviene mediante una normale trasmissione IP di tipo unicast, come evidenziato in Figura 4.38(a), dove il router R4 invia un messaggio *Join* al punto d'incontro di un gruppo. Il messaggio *Join* iniziale è un "jolly" (*wildcard*), cioè viene applicato a tutte le possibili sorgenti. Chiaramente, un messaggio *Join* deve attraversare una sequenza di router (ad esempio, R2) prima di raggiungere il punto d'incontro: ogni router lungo il percorso analizza il messaggio e crea nella propria tabella di inoltro, in corrispondenza all'albero condiviso, un dato che ha il for-

mato $(*, G)$, per rappresentare "tutti i mittenti". Per creare tale dato nella tabella di inoltro, il router osserva da quale interfaccia sia arrivato il messaggio e la contrassegna come una delle interfacce a cui inviare pacchetti di dati destinati a quel gruppo. Successivamente, viene determinata l'interfaccia da usare per inoltrare il messaggio *Join* verso il punto d'incontro, che sarà l'unica interfaccia da cui verranno accettati pacchetti destinati al gruppo. Infine, il router inoltra il messaggio *Join* verso il punto d'incontro. Prima o poi il messaggio giunge al router che svolge il ruolo di RP, completando la costruzione del ramo dell'albero. L'albero condiviso costruito in tal modo è la linea di spessore maggiore che va da RP a R4 nella Figura 4.38(a).

Mano a mano che altri router inviano messaggi *Join* verso il RP vengono aggiunti rami all'albero, come mostrato in Figura 4.38(b). Notate che, in questo caso, il messaggio *Join* deve arrivare soltanto fino a R2, che può aggiungere il ramo all'albero semplicemente aggiungendo una nuova interfaccia d'uscita al dato creato per il gruppo nella propria tabella d'inoltro: non c'è bisogno che R2 inoltri il messaggio *Join* verso il punto d'incontro. Notate anche che il risultato finale di questo procedimento è la costruzione di un albero avente RP come radice.

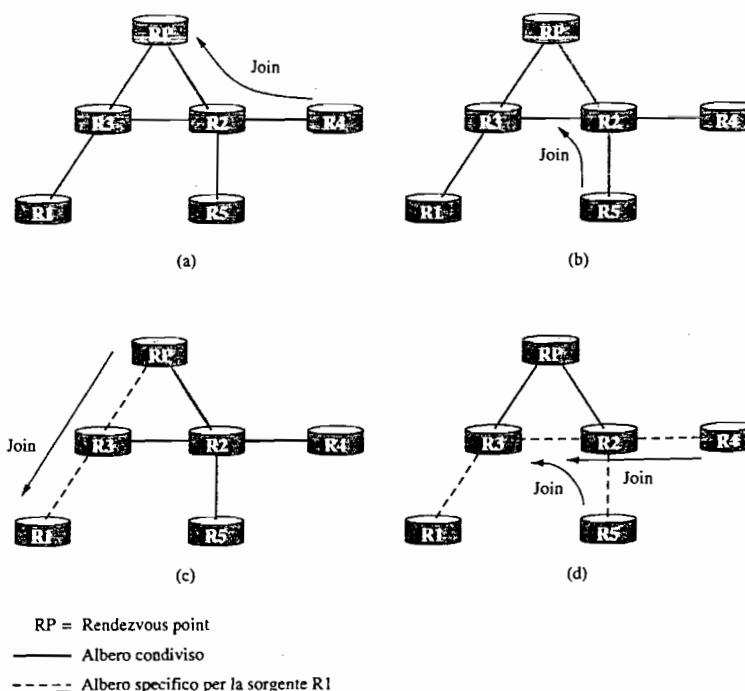


Figura 4.38 Funzionamento di PIM: (a) R4 invia un messaggio *Join* a RP ed entra a far parte dell'albero condiviso; (b) R5 viene aggiunto all'albero condiviso; (c) RP costruisce l'albero specifico per la sorgente R1 inviando un messaggio *Join* a R1; (d) R4 e R5 costruiscono alberi specifici per R1 inviando messaggi *Join* a R1.

4.4 Multicast

A questo punto supponete che un host voglia inviare un messaggio a questo gruppo: costruisce un pacchetto con l'indirizzo del gruppo multicast appropriato come destinazione e lo invia al router che nella propria rete locale è noto come *router designato* (DR, designated router). Supponete che nella Figura 4.38 il DR sia R1. A questo punto fra R1 e RP non esistono ancora informazioni di stato per questo gruppo multicast, per cui, invece di inoltrare semplicemente il pacchetto multicast, R1 lo invia a RP mediante un tunnel IP, cioè incapsula il pacchetto multicast all'interno di un pacchetto unicast che viene inviato all'indirizzo unicast di RP. Esattamente come qualsiasi punto terminale di un tunnel, come descritto nella Sezione 4.1.8, il router RP riceve il pacchetto ad esso indirizzato, esamina il carico utile del pacchetto unicast e vi trova un pacchetto IP destinato all'indirizzo multicast di questo gruppo. Il router RP, ovviamente, sa cosa fare con tale pacchetto: lo invia lungo l'albero condiviso di cui RP costituisce la radice. Nell'esempio di Figura 4.38, ciò significa che RP invia il pacchetto a R2, il quale è in grado di inoltrarlo a R4 e a R5. La Figura 4.39 mostra la consegna completa del pacchetto da R1 a R4 e R5: vediamo il pacchetto che viaggia attraverso il tunnel da R1 a RP con un'intestazione IP aggiuntiva contenente l'indirizzo unicast di RP, poi il pacchetto multicast indirizzato al gruppo G che compie il proprio percorso lungo l'albero condiviso fino a R4 e R5.

A questo punto, potremmo ritenerci soddisfatti, poiché in questo modo tutti gli host possono spedire pacchetti a tutti i ricevitori, ma c'è ancora inefficienza di banda e vi sono costi di elaborazione per l'incapsulamento (e il processo inverso) dei pacchetti che viaggiano verso RP, per cui il PIM può optare di far conoscere il gruppo ai router intermedi, per evitare l'invio lungo il tunnel. La decisione di esercitare questa opzione è basata sulla frequenza dei pacchetti che provengono da una certa sorgente: RP agisce soltanto se tale frequenza è abbastanza elevata da giustificare lo sforzo. Se decide di farlo, invia un messaggio *Join* all'host sorgente (Figura 4.38(c)). Mentre tale messaggio viaggia verso la sua destinazione, i router che si trovano lungo il percorso (R3) apprendono dell'esistenza del gruppo, per cui il DR potrà inviare pacchetti al gruppo sotto forma di pacchetti multicast "nativi", cioè non incapsulati in un tunnel.

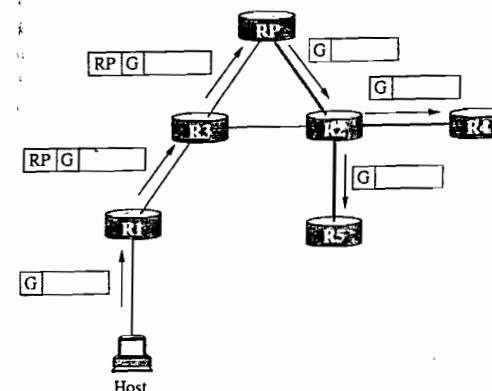


Figura 4.39 Consegnata di un pacchetto lungo un albero condiviso. R1 invia il pacchetto mediante un tunnel a RP, il quale lo inoltra lungo l'albero condiviso fino a R4 e R5.

Un dettaglio importante che va notato a questo punto è che il messaggio *Join* inviato da RP all'host mittente è specifico per quella sorgente, mentre i messaggi precedenti inviati da R4 e da R5 erano applicabili a qualsiasi sorgente. Di conseguenza, l'effetto del nuovo messaggio *Join* è quello di creare delle informazioni di stato *specifiche della sorgente* nei router che si trovano tra RP e la sorgente stessa. Questo stato viene indicato con (S, G), poiché si applica ad un solo mittente per il gruppo, diversamente dall'informazione (*, G) che era stata memorizzata fra i ricevitori e RP, che si applica a qualsiasi mittente. In Figura 4.38(d) vediamo quindi un percorso da R1 a RP (indicato con il tratteggio) che è specifico per quella sorgente, insieme ad un albero che è valido per tutte le sorgenti e va da RP ai ricevitori (indicato con una linea di spessore maggiore).

La successiva ottimizzazione possibile consiste nella sostituzione dell'intero albero condiviso con un albero specifico per la sorgente, cosa preferibile perché il percorso dal mittente al ricevente che passa per RP può essere significativamente più lungo del percorso più breve possibile. Anche questa soluzione verrà probabilmente messa in atto quando si osserva un elevato tasso di trasmissione da parte di una certa sorgente: in questo caso il router che si trova alla fine dell'albero (come R4 nel nostro esempio) invia un *Join* specifico per quella sorgente alla sorgente stessa. Mentre tale messaggio segue il percorso più breve verso la sorgente, i router lungo il percorso creano informazioni di stato del tipo (S, G) per tale albero, dando vita ad un albero che ha come radice la sorgente, invece di RP. Se ipotizziamo che sia R4 che R5 compiano la commutazione verso l'albero specifico della sorgente, ci troveremmo nella situazione illustrata in Figura 4.38(d). Notate che questo albero non coinvolge più RP: abbiamo eliminato dalla figura l'albero condiviso, per semplificare la rappresentazione, ma in realtà tutti i router che hanno ricevitori per un gruppo devono rimanere nell'albero condiviso, nell'eventualità che entrino in gioco nuove sorgenti.

Possiamo ora vedere come PIM sia "indipendente dal protocollo": tutti i meccanismi relativi alla costruzione e alla manutenzione degli alberi dipendono dal protocollo di instradamento per pacchetti unicast che viene usato all'interno del dominio. La costruzione degli alberi è completamente determinata dai percorsi seguiti dai messaggi *Join*, i quali a loro volta sono conseguenza della scelta dei percorsi più brevi effettuata dall'instradamento unicast. Volendo essere più precisi, quindi, PIM è "indipendente dal protocollo di instradamento unicast", diversamente dagli altri protocolli di instradamento multicast visti in questa sezione, che sono derivati dall'instradamento a stato delle linee o a vettore di distanza. Notate che il protocollo PIM è strettamente correlato al protocollo IP: in termini dei protocolli dello strato di rete, non è indipendente dal protocollo.

Il progetto di PIM mostra, ancora una volta, le sfide poste dalla costruzione di reti scalabili. E come la scalabilità sia a volte in contrasto con l'ottimizzazione. L'albero condiviso è certamente più scalabile di un albero specifico per la sorgente, nel senso che riconduce l'informazione di stato totale memorizzata nei router ad una funzione lineare del numero di gruppi anziché del numero di sorgenti moltiplicato per il numero di gruppi, però è probabile che sia necessario un albero specifico per la sorgente per ottenere un instradamento efficiente.

4.5 Multiprotocol Label Switching (MPLS)

Terminiamo questa presentazione del protocollo IP descrivendo un'idea che fu vista, in origine, come un modo per migliorare le prestazioni di Internet. L'idea, denominata *Multiprotocol Label Switching (MPLS, commutazione multiprotocollo a etichette)*, cerca di combinare al-

cune delle proprietà dei circuiti virtuali con la flessibilità e la robustezza dei datagrammi. D'un lato, MPLS è strettamente connessa all'architettura basata sui datagrammi dell'Internet Protocol (IP), in quanto si basa sugli indirizzi IP e sui protocolli di instradamento IP per svolgere le proprie funzioni, ma d'altra parte i router con capacità MPLS sono anche in grado di inoltrare pacchetti analizzando etichette relativamente brevi e di lunghezza fissa, con visibilità locale, proprio come accade in una rete a circuito virtuale. Forse è stato proprio questo sodalizio di due tecnologie apparentemente opposte a fare in modo che MPLS fosse accolta con sentimenti contrastanti dalla comunità dei progettisti di Internet.

Prima di esaminare il funzionamento di MPLS, è opportuno chiedersi a cosa serva. Si sono dichiarati molti obiettivi per MPLS, ma oggi viene usato principalmente per tre finalità:

- Per aggiungere alcune potenzialità di IP a dispositivi che non sono in grado di inoltrare nel modo usuale i datagrammi IP
- Per inoltrare pacchetti IP lungo "percorsi espliciti", percorsi precalcolati che non coincidono necessariamente con i percorsi che verrebbero selezionati dai normali protocolli di instradamento IP
- Per fornire supporto ad alcuni tipi di servizi per reti virtuali private

È opportuno notare che uno degli obiettivi originari, il miglioramento delle prestazioni, non figura nell'elenco, a causa dei progressi compiuti negli ultimi anni dagli algoritmi di inoltro per i router IP e per un complesso insieme di fattori che determinano le prestazioni in base all'elaborazione delle intestazioni.

Il modo migliore per capire come funzioni MPLS consiste nell'esaminare alcuni esempi del suo utilizzo. Nelle tre sezioni seguenti studieremo alcuni esempi per illustrare le tre applicazioni di MPLS che abbiamo citato.

4.5.1 Inoltro basato sulla destinazione

Una delle prime pubblicazioni che ha presentato l'idea di allegare un'etichetta ai pacchetti IP fu un lavoro di Chandranmenon e Varghese, in modo molto simile a quanto attualmente implementato nei router con capacità MPLS. L'esempio che segue mostra come funziona questa idea.

Considerate la rete di Figura 4.40. Ognuno dei due router di destra (R3 e R4) è connesso ad una rete, con prefissi 10.1.1/24 e 10.3.3/24. I rimanenti router (R1 e R2) hanno tabelle di instradamento che indicano quale interfaccia d'uscita verrà usata da ciascun router per inoltrare pacchetti verso una di tali due reti.

Quando la tecnologia MPLS viene attivata in un router, il router assegna un'etichetta ad ogni prefisso presente nella propria tabella di instradamento e comunica ai router vicini sia l'etichetta sia il prefisso da questa rappresentato, per mezzo del "Label Distribution Protocol" (*protocollo per la distribuzione delle etichette*), come illustrato in Figura 4.41. Il router R2 ha assegnato l'etichetta 15 al prefisso 10.1.1 e l'etichetta 16 al prefisso 10.3.3: queste etichette possono essere scelte dal router secondo le proprie preferenze e si possono considerare alla stregua di indici nella tabella di instradamento. Dopo aver assegnato le etichette, R2 comunica ai propri vicini gli abbinamenti tra etichette e prefissi; in questo caso, osserveremo che R2 comunica a R1 l'abbinamento tra l'etichetta 15 e il prefisso 10.1.1. Il significato di tale informazione è che R2 ha comunicato, in realtà: "Si prega di allegare l'etichetta 15 a tutti i pacchetti che mi vengono inviati e che siano destinati al

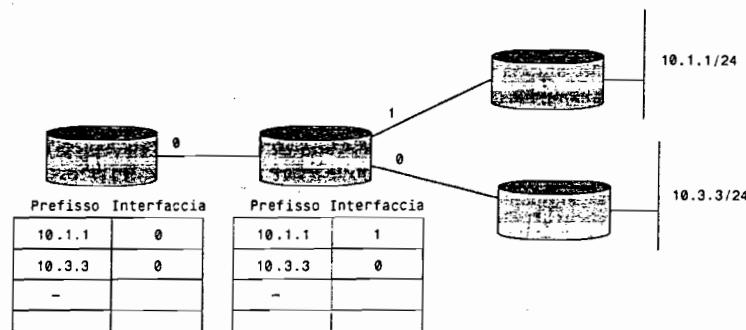


Figura 4.40 Tabelle di instradamento in una rete di esempio.

prefisso 10.1.1". R1 memorizza l'etichetta in una tabella, di fianco al prefisso che essa rappresenta, considerandola l'etichetta "remota" o "di uscita" per tutti i pacchetti che invierà verso quel prefisso.

In Figura 4.41(c) vediamo un'altra comunicazione di etichetta dal router R3 al router R2, per il prefisso 10.1.1; di conseguenza, R2 inserisce nel luogo appropriato all'interno della tabella l'etichetta "remota" che ha appreso da R3.

A questo punto, possiamo guardare a ciò che accade quando un pacchetto viene inoltrato in questa rete. Supponete che un pacchetto destinato all'indirizzo IP 10.1.1.5 giunga da sinistra al router R1. In questo caso R1 viene chiamato *label edge router* (LER): all'arrivo dei pacchetti, un LER esegue nella tabella una ricerca completa dell'indirizzo IP, quindi applica loro le etichette come risultato della ricerca nella tabella. In questo caso, R1 troverebbe nella propria tabella di inoltro una corrispondenza tra l'indirizzo 10.1.1.5 e il prefisso 10.1.1 ed osserverebbe che il dato relativo contiene sia un'interfaccia di uscita sia un valore per l'etichetta remota. Di conseguenza, R1 allega l'etichetta remota, 15, al pacchetto, prima di inviarlo.

Quando il pacchetto giunge in R2, si guarda all'etichetta presente nel pacchetto. La tabella di inoltro di R2 segnala che i pacchetti che arrivano con etichetta 15 devono essere inviati all'interfaccia 1, con etichetta 24, come comunicato dal router R3. Di conseguenza, R2 riscrive, o scambia, l'etichetta e inoltra il pacchetto verso R3.

Cosa si è ottenuto con tutte queste assegnazioni e questi scambi di etichette? Osservate che quando R2, in questo esempio, ha inoltrato il pacchetto, non ha avuto bisogno di esaminare l'indirizzo IP, ma soltanto l'etichetta: abbiamo sostituito l'usuale ricerca dell'indirizzo IP di destinazione con la ricerca in una tabella. Per capire come mai ciò sia vantaggioso, ci può essere utile ricordare che, sebbene gli indirizzi IP siano sempre della medesima lunghezza, i prefissi IP hanno lunghezze variabili e che l'algoritmo di ricerca dell'indirizzo IP di destinazione deve sempre trovare la *corrispondenza più lunga*, il più lungo prefisso che abbia i bit identici a quelli più significativi dell'indirizzo IP presente nel pacchetto che sta per essere inoltrato. Al contrario, il meccanismo di inoltro mediante etichette che abbiamo appena descritto è un algoritmo che cerca una *corrispondenza esatta*, che può essere realizzato in modo molto semplice, ad esempio usando l'etichetta come indice in un array, dove ciascun elemento dell'array è una linea della tabella di inoltro.

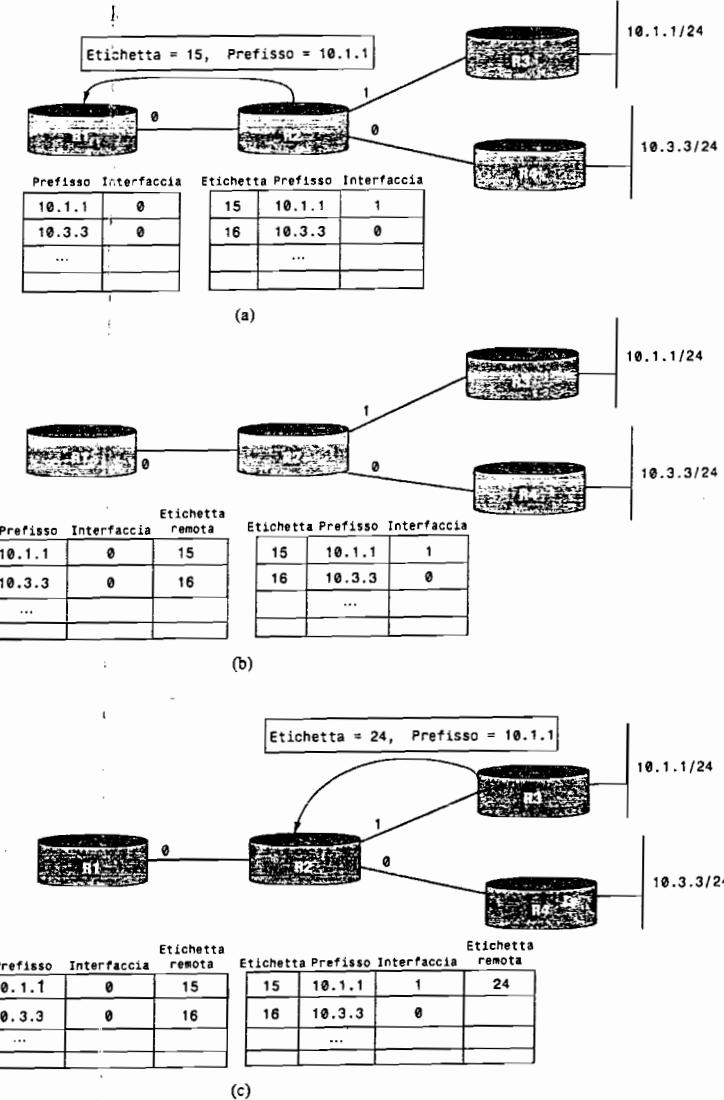


Figura 4.41 (a) R2 assegna le etichette e comunica gli abbinamenti a R1; (b) R1 memorizza in una tabella le etichette ricevute; (c) R3 comunica un altro abbinamento e R2 memorizza in una tabella l'etichetta ricevuta.

Note che, mentre l'algoritmo di inoltro è cambiato, da corrispondenza più lunga a corrispondenza esatta, l'algoritmo di instradamento può continuare ad essere un algoritmo di instradamento IP qualsiasi (ad esempio, OSPF). Il percorso che seguirà il pacchetto con questa strategia è lo stesso esatto percorso che avrebbe seguito se non si fosse usata la tecnologia MPLS, cioè il percorso scelto dagli algoritmi di instradamento IP. L'unica cosa che è cambiata è l'algoritmo di inoltro.

Il principale effetto della modifica dell'algoritmo di inoltro è che nelle reti MPLS possono essere usati dispositivi che normalmente non saprebbero come inoltrare pacchetti IP. La prima e più interessante applicazione di questo risultato avvenne nei commutatori ATM, che sono in grado di utilizzare MPLS senza alcuna modifica al proprio hardware di inoltro. I commutatori ATM sono in grado di eseguire l'algoritmo di scambio delle etichette proprio come è stato descritto e, dotando tali commutatori di algoritmi di instradamento IP e di un metodo per distribuire gli abbinamenti di etichette, possono essere trasformati in *router a commutazione di etichetta (LSR, label switching router)*, dispositivi che eseguono i protocolli di controllo di IP ma usano l'algoritmo di inoltro mediante scambio di etichette. Più recentemente, la stessa idea è stata applicata ai commutatori ottici del tipo descritto nella Sezione 3.1.2.

Prima di prendere in esame i decantati benefici della trasformazione di un commutatore ATM in un LSR, dobbiamo riesaminare alcune cose rimaste in sospeso. Abbiamo detto che le etichette vengono "allegate" ai pacchetti, ma dove vengono esattamente inserite? La risposta dipende dal tipo di linea di collegamento su cui vengono trasportati i pacchetti. Due comuni metodi per trasportare etichette insieme ai pacchetti sono illustrate in Figura 4.42. Quando i pacchetti IP sono trasportati come frame completi, come avviene nella maggior parte dei tipi di linea, come Ethernet, token ring e PPP, l'etichetta viene inserita come una "guarnizione" interposta tra l'intestazione del livello 2 e l'intestazione di IP (o di un diverso strato 3), come mostrato in Figura 4.42(b).

Ora che abbiamo identificato uno schema mediante il quale un commutatore ATM può funzionare da LSR, cosa abbiamo ottenuto? Si può mettere in evidenza che ora potremmo costruire una rete che usa router IP convenzionali, LER e commutatori ATM funzionanti come LSR, e che tutti questi apparati userebbero gli stessi protocolli di instradamento. Per capire i vantaggi derivanti dall'utilizzo di protocolli comuni, consideriamo quale sarebbe l'alternativa: in Figura 4.43(a) vediamo un insieme di router interconnessi tramite circuiti virtuali all'interno di una rete ATM, una configurazione che prende il nome di rete "sovraposta"

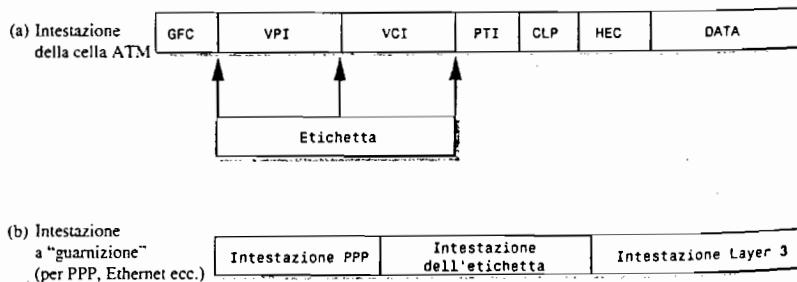


Figura 4.42 (a) Etichetta in un pacchetto incapsulato in ATM. (b) Etichetta in un pacchetto incapsulato in un frame.

In quale strato si colloca MPLS?

Ci sono state molte discussioni in merito allo strato in cui collocare la tecnologia MPLS nell'architettura di protocolli stratificata presentata nella Sezione 1.3. Poiché l'intestazione MPLS si trova solitamente, nei pacchetti, fra l'intestazione di strato 3 e l'intestazione di strato 2, a volte si parla di protocollo dello strato 2.5. Alcuni sostengono che, essendo i pacchetti IP incapsulati all'interno delle intestazioni MPLS, allora MPLS deve trovarsi "al di sotto" di IP, rendendolo in tal modo un protocollo di livello 2. Altri, invece, affermano che MPLS deve trovarsi allo stesso livello di IP (cioè nello strato 3) perché i protocolli di controllo di MPLS sono, in massima parte, gli stessi protocolli utilizzati da IP, in quanto MPLS usa i protocolli di instradamento di IP e l'indirizzamento IP. Come abbiamo fatto notare nella Sezione 1.3, le architetture a strati sono validi strumenti, ma non sempre descrivono esattamente la realtà dei fatti: MPLS costituisce un ottimo esempio di come una visione rigidamente stratificata possa difficilmente conciliarsi con la realtà.

(*overlay network*). Si sono costruite diverse reti di questo tipo, perché in un certo momento erano commercialmente disponibili commutatori ATM in grado di sostenere un throughput totale più elevato di quanto non fossero in grado di fare i router. Oggi, invece, le reti di questo tipo sono meno diffuse, perché i router hanno raggiunto e anche superato, in termini di prestazioni, i commutatori ATM, tuttavia queste reti esistono ancora, per via dei molti commutatori ATM installati nelle reti backbone, cosa che a sua volta è parzialmente dovuta alla capacità della tecnologia ATM di fornire supporto ad un ampio ventaglio di caratteristiche, quali l'ermazione di circuito e i servizi a circuito virtuale.

In una rete sovrapposta ogni router sarebbe idealmente connesso con ogni altro router mediante un circuito virtuale, ma in questo caso, per chiarezza, abbiamo messo in evidenza solamente i circuiti che vanno dal router R1 a tutti i router suoi pari. R1 ha cinque router adiacenti e ha la necessità di scambiare messaggi di instradamento con tutti loro: diciamo, quindi, che R1 ha cinque adiacenze di instradamento. Nella Figura 4.43(b), i commutatori ATM sono stati, invece, sostituiti da LSR, e non vi sono più circuiti virtuali che interconnettono i router. Di conseguenza R1 ha una sola adiacenza, nei confronti di LSR1. In reti di grandi dimensioni, l'esecuzione di MPLS nei commutatori porta ad una significativa riduzione nel numero di adiacenze che deve essere gestita da ciascun router e può ridurre notevolmente il lavoro svolto dai router per tenersi reciprocamente informati delle modifiche topologiche.

Un secondo vantaggio derivante dall'esecuzione dei medesimi protocolli nei router di confine (LER) e negli LSR consiste nel fatto che i router di confine acquisiscono, in tal modo, una visione completa della topologia della rete. Ciò significa che se nella rete si guasta una linea o un nodo i router di confine saranno di fronte ad un compito più facile nella scelta di un nuovo percorso rispetto a quanto farebbero i commutatori ATM nel reinstradare i circuiti virtuali colpiti senza disporre delle informazioni dei router di confine.

Notate che la "sostituzione" dei commutatori ATM con gli LSR si effettua, in realtà, modificando i protocolli eseguiti dai commutatori, tipicamente senza che sia necessaria alcuna modifica hardware: un commutatore ATM può spesso essere convertito in un LSR della tecnologia MPLS aggiornando solamente il suo software. Inoltre, un LSR potrebbe, allo stesso tempo, continuare a svolgere le funzioni previste dallo standard ATM, pur eseguendo anche i protocolli di controllo di MPLS.

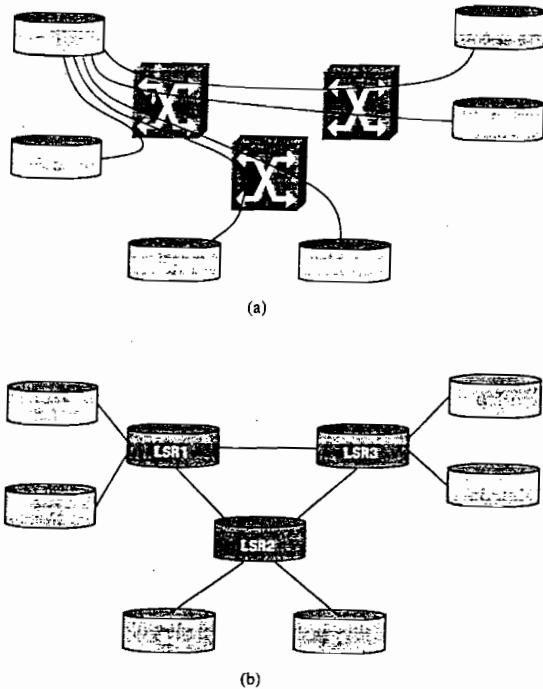


Figura 4.43 (a) Router interconnessi mediante la sovrapposizione (“overlay”) di circuiti virtuali.
 (b) I router scambiano informazioni direttamente con i LSR.

Più recentemente, l’idea di eseguire i protocolli di controllo di IP in dispositivi che non sono in grado di inoltrare in modo nativo pacchetti IP è stata estesa ai commutatori ottici e a dispositivi TDM come i multiplexer SONET: si tratta di *MPLS generalizzata* (GMPLS, Generalized MPLS). Una delle motivazioni alla base di GMPLS è stata quella di fornire ai router la conoscenza della topologia di una rete ottica, esattamente come nel caso di ATM, ma ancora più importante è stato il fatto che non vi fossero protocolli di controllo standard per dispositivi ottici, per cui la tecnologia MPLS è apparsa come naturale per svolgere tale compito.

4.5.2 Instradamento esplicito

Nella Sezione 3.1.3 abbiamo presentato il concetto di instradamento dalla sorgente. Il protocollo IP ha un’opzione per l’instradamento dalla sorgente, ma non è molto utilizzata, per vari motivi, tra cui il fatto che si possono specificare solamente un numero limitato di salti: inoltre, solitamente tale opzione viene elaborata, nella maggior parte dei router, al di fuori del “percorso veloce”.

La tecnologia MPLS fornisce un sistema conveniente per aggiungere alle reti IP delle potenzialità simili all’instradamento dalla sorgente, anche se tale potenzialità viene più spesso chiamata “instradamento esplicito” (*explicit routing*) piuttosto che “instradamento dalla sorgente”. Uno dei motivi per questa distinzione risiede nel fatto che spesso il percorso non viene realmente scelto dalla sorgente: più spesso questo compito viene svolto da uno dei router che si trovano all’interno della rete del fornitore di servizi. La Figura 4.44 mostra un esempio di come si possa applicare la capacità di MPLS di instradare in modo esplicito. Questo tipo di rete viene a volte chiamata rete “a pesce” (*fish network*) a causa della sua forma (i router R1 e R2 costituiscono la coda del pesce, mentre R7 ne è la testa).

Supponete che il gestore della rete rappresentata in Figura 4.44 abbia stabilito che il traffico che fluisce da R1 a R7 debba seguire il percorso R1-R3-R6-R7, mentre il traffico tra R2 e R7 debba seguire il percorso R2-R3-R4-R7. Una possibile motivazione per tale scelta sarebbe quello di fare buon uso della capacità disponibile lungo i due diversi percorsi che vanno da R3 a R7, cosa che non si può realizzare facilmente con il normale instradamento IP perché R3 non guarda alla provenienza del traffico quando prende le proprie decisioni di inoltro.

Dato che, invece, MPLS usa lo scambio di etichette per inoltrare i pacchetti, se i router sono in grado di funzionare con la tecnologia MPLS è abbastanza facile ottenere l’instradamento voluto. Se R1 e R2 allegano ai pacchetti etichette diverse, prima di inoltrarli verso R3, allora R3 può inoltrare lungo percorsi diversi i pacchetti che provengono da R1 e R2. La domanda a cui si deve poi rispondere è: come fanno tutti i router della rete ad accordarsi sulle etichette da utilizzare e su come inoltrare i pacchetti aventi certe etichette? È ovvio che non si possano usare, per distribuire le etichette, le stesse procedure descritte nella sezione precedente, perché tali procedure attribuiscono etichette che inducono i pacchetti a seguire i percorsi normalmente identificati dall’instradamento, che è proprio ciò che vogliamo evitare. Serve, invece, un nuovo e diverso meccanismo. Il protocollo utilizzato per risolvere questo problema è il Resource Reservation Protocol (RSVP, protocollo per prenotazione di risorse). Presenteremo più diffusamente questo protocollo nella Sezione 6.5.2, ma per ora ci sarà sufficiente dire che è possibile inviare un messaggio RSVP lungo un percorso specificato esplicitamente (ad esempio, R1-P3-R6-R7) e usarlo per impostare le etichette nelle tabelle di inoltro lungo l’intero percorso, in modo molto simile al processo usato per instaurare un circuito virtuale che è stato descritto nella Sezione 3.1.2.

Una delle applicazioni dell’instradamento esplicito è la “gestione ingegneristica del traffico” (*traffic engineering*), che si riferisce al compito di assicurare, in una rete, una disponibilità di risorse sufficiente alle richieste: controllare con precisione quali siano i percorsi lungo cui fluisce il traffico è una parte importante della gestione del traffico stesso. L’instradamento

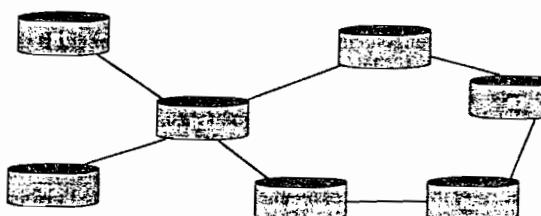


Figura 4.44 Una rete che necessita di instradamento esplicito.

esplicito può anche aiutare a rendere le reti più robuste in caso di guasto, usando una caratteristica nota come *reinstradamento veloce*. Ad esempio, è possibile precalcolare un percorso che vada dal router A al router B e che eviti esplicitamente una certa linea L: nel caso in cui la linea si guasti, il router A potrebbe instradare lungo il percorso precalcolato tutto il traffico destinato a B. La combinazione del calcolo preventivo del "percorso alternativo" e l'instradamento esplicito dei pacchetti lungo il percorso consente ad A di non dover attendere che i pacchetti del protocollo di instradamento si facciano strada nella rete, o che gli algoritmi di instradamento vengano eseguiti dai vari nodi della rete. In alcune circostanze ciò può ridurre significativamente il tempo richiesto per reinstradare i pacchetti per aggirare un guasto.

Un ultimo punto degno di nota riguardo all'instradamento esplicito è che non è necessario che i percorsi esplicativi in una rete vengano calcolati da un operatore, come nell'esempio precedente. Esistono algoritmi mediante i quali i router possono calcolare automaticamente i percorsi esplicativi, tra i quali il più comune è detto CSPF (*constrained shortest path first*), che è simile agli algoritmi a stato delle linee che abbiamo descritto nella Sezione 4.2.3, ma prende anche in considerazione dei "vincoli" (*constraint*). Ad esempio, se vi fosse il problema di trovare un percorso da R1 a R7 che possa trasportare un traffico offerto di 100 Mbps, potremmo impostare il vincolo che ciascuna linea abbia una capacità disponibile di almeno 100 Mbps: CSPF risolve questo tipo di problemi. La sezione "Ulteriori letture" al termine del capitolo fornisce maggiori dettagli su CSPF e sulle applicazioni dell'instradamento esplicito.

4.5.3 Reti private virtuali e tunnel

Nella Sezione 4.1.8 abbiamo parlato per la prima volta di reti private virtuali (VPN, *virtual private network*) e abbiamo messo in evidenza che l'utilizzo dei tunnel forniva uno strumento per costruire tali reti. Adesso abbiamo visto che la tecnologia MPLS può essere utilizzata per costruire tunnel, per cui è, di riflesso, adatta a costruire VPN di vari tipi.

La più semplice forma di VPN realizzabile con MPLS è una VPN "di livello 2". In questo tipo di VPN la tecnologia MPLS viene usata per inviare in un tunnel dati dello strato 2 (come i frame Ethernet o le celle ATM) attraverso una rete di router che siano in grado di gestire MPLS. Ricordate, dalla Sezione 4.1.8, che una delle motivazioni che stanno alla base dell'utilizzo dei tunnel consiste nel fornire una qualche forma di servizio di rete (come la trasmissione multicast) per il quale alcuni router della rete non forniscano il supporto necessario. Qui ci troviamo nella stessa situazione: i router IP non sono commutatori ATM, per cui non è possibile fornire un servizio di circuito virtuale attraverso una rete di router convenzionali. Tuttavia, mediante una coppia di router interconnessi da un tunnel, è possibile inviare celle ATM attraverso il tunnel ed emulare un circuito ATM. Il termine usato da IETF per questa tecnica è *emulazione di pseudo-collegamento (pseudowire emulation)*, di cui la Figura 4.45 mostra l'idea di fondo.

Abbiamo già visto come vengono instaurati i tunnel IP: il router che si trova all'ingresso del tunnel incapsula i dati che devono essere inviati nel tunnel in un'intestazione IP (detta "intestazione del tunnel", *tunnel header*), che rappresenta l'indirizzo del router che si trova all'altra estremità del tunnel, dopodiché i dati vengono trasmessi come normali pacchetti IP. Il router che si trova all'uscita del tunnel riceve il pacchetto che presenta il proprio indirizzo nell'intestazione, dopo di che elimina tale intestazione relativa al tunnel, ottenendo così i dati inviati lungo il tunnel, da elaborare successivamente. In cosa consiste l'elaborazione dei dati, dipende dalla natura dei dati stessi. Ad esempio, se i dati consistono in un pacchetto IP, esso verrà inoltrato come ogni altro pacchetto IP, ma non è necessario che i dati siano un pacchetto

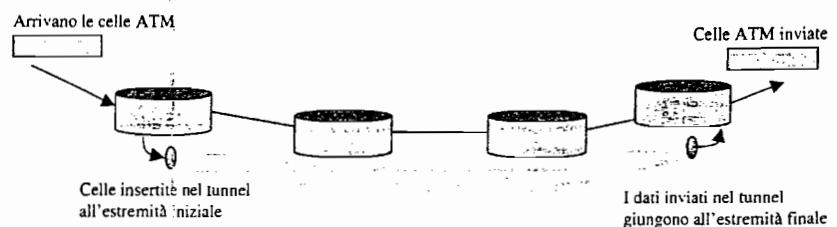


Figura 4.45 Un circuito ATM emulato mediante un tunnel.

IP, a patto che il router che li riceve sappia cosa fare con pacchetti che non appartengano al protocollo IP. A breve torneremo sull'argomento relativo alla gestione di pacchetti che non siano pacchetti IP.

Un tunnel MPLS non è molto diverso da un tunnel IP, tranne per il fatto che la "intestazione del tunnel" è costituita da un'intestazione MPLS piuttosto che da un'intestazione IP. Tornando al nostro primo esempio, in Figura 4.41, abbiamo visto che il router R1 allega un'etichetta (15) ad ogni pacchetto che viene inviato verso il prefisso 10.1.1, dopodiché il pacchetto seguirà il percorso R1-R2-R3 e ciascun router lungo il percorso esaminerà solamente l'etichetta MPLS. Di conseguenza, osserviamo che non c'è alcun vincolo sul fatto che il router R1 possa inviare lungo tale percorso solamente pacchetti IP: all'interno dell'intestazione MPLS possono essere incapsulati dati di qualsiasi natura, che seguirebbero lo stesso percorso perché i router interposti non analizzano altro che l'intestazione MPLS. Da questo punto di vista, un'intestazione MPLS svolge la stessa funzione dell'intestazione di un tunnel IP⁷. L'unica domanda che dobbiamo porci quando inviamo traffico che non sia di tipo IP lungo un tunnel, sia esso un tunnel MPLS oppure no, è la seguente: come si gestisce il traffico che non sia di tipo IP quando viene raggiunta la fine del tunnel? La soluzione generale consiste nel trasportare, nel carico utile che viaggia lungo il tunnel, una qualche forma di identificativo di demultiplexing che dica al router che si trova alla fine del tunnel cosa fare, e si può vedere che un'etichetta MPLS svolge perfettamente la funzione di tale identificativo, come risulterà chiaro dall'esempio che segue.

Ipotizziamo di voler inviare, tramite un tunnel, celle ATM da un router ad un altro, attraverso una rete di router capaci di gestire la tecnologia MPLS, come rappresentato in Figura 4.45. Ipotizziamo anche che l'obiettivo sia quello di emulare un circuito virtuale ATM: la cella giunge all'ingresso del tunnel, arrivando ad una certa porta con un qualche valore di VCI, e deve uscire dal tunnel tramite una porta assegnata e con un valore di VCI potenzialmente diverso. Ciò si può ottenere configurando i router "di entrata" e "di uscita" del tunnel nel modo seguente:

- Il router di entrata deve essere configurato con le informazioni relative alla porta da cui giungono le celle, il valore di VCI di ingresso, la "etichetta di demultiplexing" relativa al circuito da emulare e l'indirizzo del router di uscita del tunnel.

⁷ Notate, però, che un'intestazione MPLS è lunga solamente 4 byte, mentre un'intestazione IP è lunga 20 byte, garantendo così un significativo risparmio di banda quando viene usata la tecnologia MPLS.

- Il router di uscita deve essere configurato con le informazioni relative alla porta di uscita, al valore di VCI di uscita e all'etichetta di demultiplexing.

Una volta che queste informazioni siano state fornite ai router, possiamo vedere come possa essere inoltrata una cella ATM, con i passi evidenziati in Figura 4.46.

- Una cella ATM giunge alla porta designata con il valore di VCI appropriato (101 in questo esempio).
- Il router d'entrata allega l'etichetta di demultiplexing che identifica il circuito emulato.
- Il router d'entrata aggiunge poi una seconda etichetta, relativa al tunnel che porterà il pacchetto verso il router di uscita. Il valore di questa etichetta viene appreso mediante il meccanismo appena descritto nella Sezione 4.5.1.
- I router interposti tra l'entrata e l'uscita del tunnel inoltrano il pacchetto usando solamente l'etichetta del tunnel.
- Il router di uscita elimina l'etichetta relativa al tunnel, identifica l'etichetta di demultiplexing e riconosce il circuito emulato.
- Il router di uscita modifica il valore di VCI, inserendo il valore corretto (202 in questo caso) e invia la cella ATM verso la porta d'uscita corretta.

In questo esempio, ci si potrebbe sorprendere del fatto che il pacchetto viaggi con due etichette indicate. Questa è una delle caratteristiche interessanti di MPLS: le etichette in un pacchetto possono essere "impilate" fino ad una dimensione arbitraria. Questo meccanismo fornisce potenzialità utili per la scalabilità: in questo esempio consente ad un singolo tunnel di trasportare un numero potenzialmente elevato di circuiti emulati.

La stessa tecnica qui descritta può essere applicata per l'emulazione di molti altri servizi di livello 2, tra cui Frame Relay e Ethernet. Vale forse la pena di mettere in evidenza che mediante i tunnel IP si possono avere a disposizione, virtualmente, le medesime potenzialità: il vantaggio principale della tecnologia MPLS è la piccola dimensione dell'intestazione relativa al tunnel.

Prima che MPLS venisse usata per fornire servizi di livello 2 mediante i tunnel, è stata anche usata per fornire supporto alle reti private virtuali (VPN) di livello 3. Non entriremo nei dettagli delle VPN di livello 3, che sono piuttosto complesse (consultate la sezione "Ulteriori letture" per alcune valide fonti di informazione), ma noteremo che esse rappresentano l'utilizzo più diffuso di MPLS al giorno d'oggi. Anche le VPN di livello 3 usano pile di etichette MPLS per inoltrare pacchetti IP lungo tunnel in una rete, ma i pacchetti che vengo-

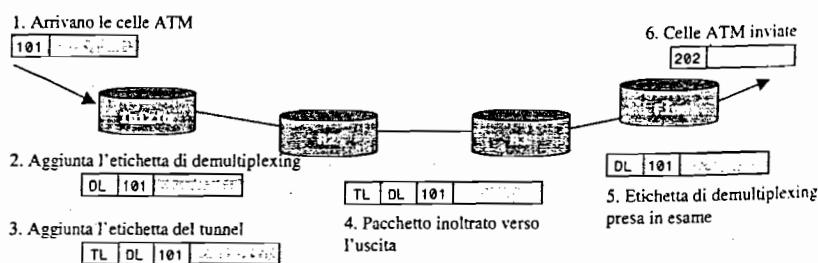


Figura 4.46 Inoltro di celle ATM lungo un tunnel.

no inoltrati nel tunnel sono i pacchetti stessi, da cui il nome "VPN di livello 3". In una VPN di livello 3, un singolo fornitore di servizi gestisce una rete di router con capacità MPLS e fornisce un servizio di rete IP "virtualmente privato" ad un numero qualunque di clienti distinti: ogni cliente del fornitrone ha un certo numero di siti ed il fornitrone di servizio crea al cliente l'illusione di essere l'unico cliente della rete. Il cliente vede una rete IP che interconnette i propri siti, senza vedere alcun altro sito: ciò significa che ciascun cliente è isolato da tutti gli altri clienti, sia in termini di instradamento, sia di indirizzamento. Il cliente A non può inviare direttamente pacchetti al cliente B, e viceversa⁸. Il cliente A può anche usare indirizzi IP che sono utilizzati anche dal cliente B. L'idea di fondo viene illustrata in Figura 4.47: come per le VPN di livello 2, la tecnologia MPLS viene utilizzata per inviare i pacchetti da un sito all'altro mediante dei tunnel, però la configurazione dei tunnel viene eseguita automaticamente da un uso abbastanza elaborato del protocollo BGP, che va al di là degli scopi di questo libro.

Riassumendo, la tecnologia MPLS è uno strumento abbastanza versatile che è stato applicato ad un'ampia varietà di problemi di rete diversi, combinando il meccanismo di inoltro mediante lo scambio di etichette, normalmente associato alle reti a circuito virtuale, con i protocolli di instradamento e controllo delle reti a datagrammi IP, per dar luogo ad una cate-

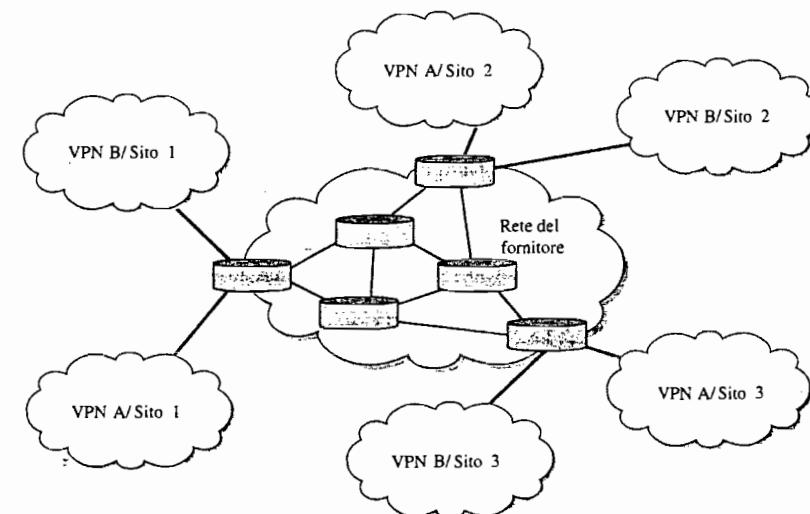


Figura 4.47 Esempio di una VPN di livello 3. Entrambi i clienti, A e B, ottengono da un unico fornitrone un servizio IP virtualmente privato.

⁸ In realtà, il cliente A solitamente può inviare dati al cliente B, con alcune limitazioni. Nella maggior parte dei casi, entrambi i clienti, A e B, hanno connessioni verso la rete Internet globale, per cui è probabile che il cliente A possa inviare, ad esempio, messaggi di posta elettronica al server di posta elettronica che si trova all'interno della rete del cliente B. La "privacy" offerta da una VPN impedisce al cliente A di avere accesso illimitato a tutte le macchine e alle sottoreti che si trovano all'interno della rete del cliente B.

goria di reti che si colloca, in qualche modo, in una posizione intermedia rispetto ai due estremi più convenzionali. Ciò estende le possibilità di una rete IP, consentendo, fra le altre cose, un controllo più preciso dell'instradamento e il supporto di un vasto insieme di servizi di rete privata virtuale.

4.6 Riepilogo

Il tema principale di questo capitolo è stato quello di costruire grandi reti interconnettendo reti più piccole. Nel capitolo precedente avevamo visto l'interconnessione di reti mediante il *bridging*, ma tale tecnica viene utilizzata principalmente per interconnettere un numero medio/piccolo di reti simili, perché non è in grado di risolvere in modo soddisfacente due dei problemi strettamente correlati alla costruzione di reti molto grandi: l'*eterogeneità* e la *scalabilità*. La chiave di volta per gestire questi problemi è l'Internet Protocol (IP), su cui si è posta l'attenzione in questo capitolo.

Il protocollo IP affronta il problema dell'*eterogeneità* definendo per una internetwork un modello di servizio semplice e condivisibile, basato sulla consegna di datagrammi IP in modalità *best-effort*. Un elemento importante del modello di servizio è lo schema di indirizzamento globale, che consente a due qualsiasi nodi di una internetwork di identificarsi univocamente l'un l'altro, allo scopo di scambiarsi dati. Il modello di servizio IP è sufficientemente semplice da poter essere realizzato con qualsiasi tecnologia di rete conosciuta, usando il protocollo ARP per tradurre gli indirizzi IP globali in indirizzi locali dello strato della linea di collegamento.

Un aspetto cruciale delle operazioni all'interno di una internetwork è la determinazione di percorsi efficienti per raggiungere qualsiasi destinazione nella rete stessa. Gli algoritmi di instradamento di Internet risolvono questo problema in modo distribuito; questo capitolo ha presentato due categorie principali di algoritmi, a stato delle linee e a vettore di distanza, insieme ad esempi della loro applicazione (OSPF e RIP), oltre ad aver preso in esame le estensioni dell'instradamento IP che forniscono supporto a host mobili.

Successivamente abbiamo preso coscienza di vari problemi relativi alla scalabilità e al modo in cui IP li gestisce. Il problema di scalabilità più importante è l'utilizzo efficiente dello spazio di indirizzamento e la crescita delle tabelle di instradamento al crescere di Internet. Il formato gerarchico dell'indirizzo IP, con le sue parti di rete e di host, ci fornisce un primo livello di gerarchia per gestire la scalabilità. La suddivisione in sottoreti consente un uso più efficiente dei numeri di rete e aiuta ad aggregare le informazioni di instradamento: a tutti gli effetti, la suddivisione in sottoreti aggiunge un nuovo livello di gerarchia negli indirizzi. L'instradamento senza classi (CIDR) introduce ulteriori livelli di gerarchia e ottiene un'aggregazione ancora migliore. I sistemi autonomi ci consentono di suddividere il problema di instradamento in due parti, instradamento interdominio e intradominio, ciascuna delle quali ha dimensione inferiore a quella del problema di instradamento complessivo. Questi meccanismi hanno consentito alla rete Internet di sostenere una crescita assai significativa fino ad oggi.

Prima o poi anche questi meccanismi non saranno più in grado di gestire la crescita di Internet e sarà necessario un nuovo formato di indirizzo. Ciò richiederà un nuovo formato del datagramma IP e una nuova versione del protocollo, che, originariamente noto come IPng (IP Next Generation), ha preso oggi il nome di IPv6 e usa indirizzi di 128 bit con indirizzamento e instradamento simili a quelli di CIDR. Nonostante siano state annunciate molte nuove

potenzialità di IPv6, il vantaggio fondamentale rimane quello di poter indirizzare un numero estremamente grande di dispositivi.

Problema aperto L'installazione di IPv6

Sono trascorsi più di dieci anni da quando la scarsità di indirizzi IPv4 è divenuta abbastanza seria da spingere verso proposte di una nuova versione di IP: la prima specifica di IPv6 è di sette anni fa. Sono oggi ben disponibili sistemi operativi con potenzialità IPv6 per host e i principali produttori di router offrono varie opzioni di supporto ad IPv6 per i loro prodotti. Ciò nonostante, l'installazione di IPv6 in Internet non è ancora iniziata in modo sostanziale, nel momento in cui scriviamo. Vale la pena di chiedersi quando inizierà seriamente la diffusione di IPv6, e cosa la provocherà.

Un motivo per cui *non c'è* stato subito bisogno di IPv6 è l'intenso utilizzo della tecnica di traduzione degli indirizzi di rete (NAT), descritta in precedenza nel capitolo. Quando i fornitori di rete avvertirono la scarsità degli indirizzi di rete di tipo IPv4, iniziarono ad assegnarne pochi ai loro clienti, oppure a farsi pagare in base al numero di indirizzi utilizzati; i clienti risposero nascondendo molti dei propri dispositivi dietro ad un dispositivo NAT, usando un singolo indirizzo IPv4. Ad esempio, è assai comune che la maggior parte delle reti domestiche, con più dispositivi in grado di utilizzare il protocollo IP, usino qualche forma di NAT nella rete per fare economia di indirizzi. Di conseguenza, uno dei fattori che potrebbero portare alla diffusione di IPv6 sarebbero applicazioni che non funzionassero bene con la tecnica NAT. Mentre le applicazioni di tipo client/server funzionano ragionevolmente bene quando l'indirizzo del client è "nascosto" dietro ad un dispositivo NAT, le applicazioni di tipo peer-to-peer lo fanno molto meno. Esempi di applicazioni che funzionerebbero meglio senza NAT e trarrebbero quindi beneficio da politiche di assegnazione degli indirizzi più libere sono i giochi a più giocatori e la telefonia IP.

Ottenere blocchi di indirizzi IPv4 è diventato, negli anni, sempre più difficile, particolarmente in Paesi al di fuori degli Stati Uniti d'America. All'aumentare delle difficoltà, aumenta la spinta verso i fornitori ad offrire indirizzi IPv6 ai propri clienti. Allo stesso tempo, per i fornitori esistenti, offrire IPv6 è sostanzialmente un costo aggiuntivo, perché non smetteranno di fornire supporto ad IPv4 nel momento in cui inizieranno ad utilizzare IPv6. Ciò significa, ad esempio, che la dimensione delle tabelle di instradamento dei fornitori possono, inizialmente, soltanto aumentare, perché devono contenere tutti i prefissi IPv4 esistenti, più alcuni nuovi prefissi IPv6.

Al momento, l'installazione di IPv6 sta avvenendo quasi esclusivamente in reti di ricerca. Pochi fornitori di servizi stanno iniziando ad offrire queste caratteristiche, spesso con qualche sovvenzione da parte dei governi nazionali. Sembra difficile immaginare che Internet possa continuare a crescere indefinitamente senza che si assista ad un'installazione significativa di IPv6, ma sembra altrettanto probabile che la stragrande maggioranza degli host e delle reti continuerà ad essere dotata, nel prossimo futuro, delle sole capacità IPv4.

Ulteriori letture

Non ci meraviglia il fatto che siano stati scritti innumerevoli lavori sui vari aspetti di Internet. Tra questi, ne raccomandiamo due, la cui lettura è veramente obbligatoria: il lavoro di Cerf e Kahn è quello che ha originariamente presentato l'architettura TCP/IP e vale la pena leggerlo

da un punto di vista storico; il lavoro di Bradner e Mankin fornisce una panoramica su come la rapida crescita di Internet abbia messo a dura prova la scalabilità dell'architettura originaria, dando infine luogo alla futura generazione di IP. Il lavoro di Paxson presenta uno studio di come i router si comportino all'interno di Internet, e costituisce anche un buon esempio di come i ricercatori stiano oggi studiando il comportamento dinamico di Internet. L'ultima pubblicazione discute la trasmissione multicast, presentando l'approccio al multicast usato originariamente da MBone.

- Cerf, V., e R. Kahn. "A protocol for packet network intercommunication". *IEEE Transactions on Communications* COM-22(5):637-648, May 1974.
- Bradner, S., e A. Mankin. "The recommendation for the next generation IP protocol". *Request for Comments* 1752, January 1995.
- Paxson, V. "End-to-end routing behavior in the Internet", *SIGCOMM 96*, pagg. 25-38. August 1996.
- Deering, S., e D. Cheriton. "Multicast routing in datagram internetworks and extended LANs", *ACM Transactions on Computer Systems* 8(2):85-110, May 1990.

Oltre a questi lavori, Perlman fornisce un'eccellente spiegazione dell'instradamento in una internetwork, trattando sia i bridge sia i router [Per00]. Ancora, il libro di Lynch e Rose presenta informazioni generali sulla scalabilità di Internet [Cha93], mentre in Labovitz *et al.* [LAAJ00] vengono presentati alcuni interessanti studi sperimentali sul comportamento dell'instradamento in Internet.

Molte delle tecniche e dei protocolli sviluppati per migliorare la scalabilità di Internet sono descritti in documenti RFC: la suddivisione in sottoreti è descritta in Mogul e Postel [MP85], CIDR è descritto in Fuller *et al.* [FLYV93], RIP è definito in Hedrick [Hed88] e in Malkin [Mal93], OSPF è definito in Moy [Moy98] e BGP-4 è definito in Rekhter e Li [RL95]. La specifica di OSPF, di circa 200 pagine, è uno dei documenti RFC più lunghi, ma contiene anche un insieme di dettagli fuori dall'usuale in merito a come implementare un protocollo. In Bradner e Mankin [BM95] si può trovare una raccolta di RFC relativi a IPv6, mentre la specifica più recente di IPv6 è quella di Deering e Hinden [DH98]. Le motivazioni per cui la frammentazione IP è da evitare sono prese in esame in Kent e Mogul [KM87], mentre la tecnica per trovare il valore di MTU viene descritta in Mogul e Deering [MD90]. Il protocollo PIM (Protocol Independent Multicast) è descritto in Deering *et al.* [DEF+96] e [EFH+98].

Molto lavoro è stato fatto per sviluppare algoritmi che possano essere utilizzati dai router per cercare rapidamente gli indirizzi IP nelle tabelle (ricordate che il problema sta nel fatto che il router deve cercare la corrispondenza più lunga fra i prefissi presenti nella tabella di inoltro). Gli alberi PATRICIA sono fra i primi algoritmi ad essere stati applicati a questo problema [Mor68]. Lavori più recenti sono presentati in [DBCP97], [WVTP97], [LS98] e [SVSW98]. Per una panoramica di come questi tre algoritmi si possano usare per costruire un router ad alta velocità, consultate il lavoro di Partridge *et al.* [Par98].

La tecnologia MPLS e i protocolli connessi, che ne hanno favorito lo sviluppo, sono descritti in Chandranmenon e Varghese [CV95], Rekhter *et al.* [RDR+97] e Davie e Rekhter [DR00]. L'ultimo lavoro citato descrive molte applicazioni di MPLS, come la gestione del traffico, il rapido recupero di guasti di rete e le reti private virtuali. Il lavoro [RR99] fornisce le specifiche delle VPN basate su MPLS/BGP, un tipo di VPN di livello 3 che si può realizzare in reti MPLS.

Infine, raccomandiamo i seguenti riferimenti attivi:

- <http://www.ietf.org/>: la pagina di IETF, dove si possono consultare RFC, bozze preliminari di documenti Internet e statuti dei gruppi di lavoro;
- <http://playground.sun.com/pub/ipng/html/ipng-main.html>: lo stato attuale di IPv6.

Esercizi

1. Quali caratteristiche degli indirizzi IP richiedono che ci sia un indirizzo per ogni interfaccia di rete, invece di un indirizzo per ogni host? Alla luce della vostra risposta, perché IP tollera interfacce punto-punto che abbiano indirizzi non univoci, o addirittura senza indirizzo?
2. Perché il campo *Offset* dell'intestazione IP è misurato in unità di 8 byte? (Suggerimento: ricordate che il campo *Offset* è lungo 13 byte).
3. Alcuni errori di segnale possono provocare la sostituzione di interi gruppi di bit di un pacchetto mediante una sequenza di valori 0 oppure una sequenza di valori 1. Supponete che vengano sostituiti in tal modo tutti i bit di un intero pacchetto, compresa la somma di controllo di Internet. Un pacchetto composto da soli valori 0 o soli valori 1 può essere un pacchetto IPv4 lecito? La somma di controllo di Internet segnala l'errore? Perché o perché no?
4. Supponete che un messaggio TCP che contiene 2048 byte di dati e 20 byte di intestazione TCP venga trasmesso al protocollo IP per la consegna attraverso due reti di Internet (cioè dall'host sorgente ad un router, e poi da questo all'host di destinazione). La prima rete usa intestazioni di 14 byte e ha un MTU di 1024 byte; la seconda usa intestazioni di 8 byte e ha un MTU di 512 byte. Il valore di MTU di ciascuna rete indica la dimensione del più grande pacchetto IP che può viaggiare all'interno di un frame nello strato di linea di collegamento. Individuate le dimensioni e gli offset della sequenza di frammenti che vengono consegnati allo strato di rete nell'host di destinazione. Ipotizzate che tutte le intestazioni IP siano di 20 byte.
5. Il valore di *MTU di percorso* è il più piccolo valore di MTU tra quelli di tutte le linee che si trovano lungo il percorso fra due host. Ipotizzate di poter determinare il valore dell'MTU di percorso del percorso usato nell'esercizio precedente e di usare tale valore come MTU di tutti i segmenti del percorso stesso. Individuate le dimensioni e gli offset della sequenza di frammenti che vengono consegnati allo strato di rete nell'host di destinazione.
6. Ipotizzate che un pacchetto IP venga scomposto in 10 frammenti, ciascuno con una probabilità di perdita (indipendente) uguale all'1%. In ragionevole approssimazione, ciò significa che la probabilità di perdere l'intero pacchetto a causa della perdita di un frammento è uguale al 10%. Qual è la probabilità netta di perdere l'intero pacchetto se questo viene trasmesso due volte
 - a) nell'ipotesi che tutti i frammenti ricevuti abbiano fatto parte della stessa trasmissione?
 - b) nell'ipotesi che ciascun frammento abbia fatto parte di una trasmissione qualsiasi?
 - c) Spiegate perché in questo caso si può usare il campo *Ident*.
7. Supponete che tutti i frammenti della Figura 4.5(b) passino attraverso un altro router lungo una linea avente un MTU di 380 byte, senza contare l'intestazione della linea di

- collegamento. Mostrate quali frammenti vengono prodotti. Se per tale valore di MTU il pacchetto fosse stato frammentato all'origine, quanti frammenti sarebbero stati generati?
8. Qual è la massima ampiezza di banda a cui un host può inviare pacchetti IP di 576 byte senza che il campo Ident torni al proprio valore iniziale entro 60 secondi? Supponete che il tempo di vita massimo di un segmento (*MSL, maximum segment lifetime*) sia 60 secondi, cioè che i pacchetti in ritardo possano arrivare con un ritardo massimo di 60 secondi, ma non di più. Cosa potrebbe succedere se questa ampiezza di banda venisse superata?
 9. Lo strato ATM AAL3/4 usa i campi Btag/Etag, BASize/Len, Type, SEQ, MID, Length e CRC-10 per realizzare la frammentazione delle celle, mentre IPv4 usa, fra gli altri, i campi Ident e Offset e il bit M nel campo Flags. Qual è, in IP, il campo analogo a ciascun campo di AAL3/4, se esiste? Esiste in AAL3/4 un campo analogo ad ognuno dei campi IP elencati? Quanto è buona la corrispondenza tra questi campi?
 10. Secondo voi, perché il protocollo IPv4 esegue la ricostruzione dei frammenti al termine del percorso, piuttosto che in ogni router attraversato? Perché IPv6 ha abbandonato completamente la frammentazione? Suggerimento: Pensate alle differenze esistenti fra la frammentazione nello strato IP e la frammentazione nello strato della linea di collegamento.
 11. Il fatto che i dati presenti in una tabella ARP scadano dopo 10-15 minuti è un tentativo di ottenere un compromesso ragionevole. Descrivete i problemi che si possono verificare se tale durata è troppo breve o troppo lunga.
 12. Attualmente IP usa indirizzi di 32 bit. Se potessimo riprogettare IP in modo che usasse l'indirizzo MAC a 6 byte invece dell'indirizzo a 32 bit, saremmo in grado di eliminare la necessità di disporre del protocollo ARP? Spiegate perché o perché no.
 13. Supponete che agli host A e B sia stato assegnato lo stesso indirizzo IP sulla stessa Ethernet, all'interno della quale viene usato ARP. B inizia a funzionare dopo A. Cosa accadrà alle connessioni di A già esistenti? Spiegate come la soluzione di questo problema possa essere agevolata dall'uso dell'"auto-ARP" (l'interrogazione della rete all'accensione in merito al proprio indirizzo IP).
 14. Supponete che un'implementazione del protocollo IP segua alla lettera il seguente algoritmo ogniqualvolta viene ricevuto un pacchetto, P, destinato all'indirizzo IP D:

```

    se (l'indirizzo Ethernet per D è nella cache di ARP)
        (invia P)
    altrimenti
        (emetti una richiesta ARP per D)
        (inserisci P in una coda finché non arriva la risposta)

```

- a) Se lo strato IP riceve una rapida sequenza (*burst*) di pacchetti destinati a D, per quale motivo questo algoritmo spreca inutilmente risorse?
- b) Delineate una versione migliore.
- c) Supponete che, quando la ricerca nella cache fallisce, dopo aver emesso una richiesta il pacchetto P venga semplicemente eliminato. Quale sarebbe il comportamento risultante? (Alcune delle prime implementazioni di ARP si comportavano in questo modo)
15. Per la rete di Figura 4.48, indicate le tabelle globali dei vettori di distanza simili alle Tabelle 4.5 e 4.8, nel caso in cui

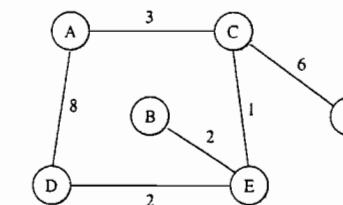


Figura 4.48 Rete per gli Esercizi 15, 17 e 20.

- a) ogni nodo conosca solamente le distanze dai nodi adiacenti.
 - b) ogni nodo abbia segnalato ai nodi adiacenti le informazioni apprese nel passo precedente.
 - c) il passo (b) venga eseguito una seconda volta.
- ✓ 16. Per la rete di Figura 4.49, indicate le tabelle globali dei vettori di distanza simili alle Tabelle 4.5 e 4.8, nel caso in cui
- a) ogni nodo conosca solamente le distanze dai nodi adiacenti.
 - b) ogni nodo abbia segnalato ai nodi adiacenti le informazioni apprese nel passo precedente.
 - c) il passo (b) venga eseguito una seconda volta
17. Per la rete di Figura 4.48, mostrate come l'algoritmo *a stato delle linee* costruisca la tabella di instradamento per il nodo D.
18. Supponete di conoscere le tabelle di inoltro mostrate in Tabella 4.12 per i nodi A e F, in una rete dove tutte le linee hanno costo unitario. Tracciate uno schema della più piccola rete coerente con tali tabelle.

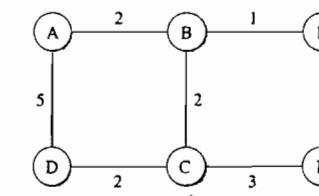


Figura 4.49 Rete per l'Esercizio 16.

Tabella 4.12 Tabelle di inoltro per l'Esercizio 18.

Nodo	A		Nodo	F	
	Costo	Salto successivo		Costo	Salto successivo
B	1	B	A	3	E
C	2	B	B	2	C
D	1	D	C	1	C
E	2	B	D	2	E
F	3	D	E	1	E

- ✓ 19. Supponete di conoscere le tabelle di inoltro mostrate in Tabella 4.13 per i nodi A e F. in una rete dove tutte le linee hanno costo unitario. Tracciate uno schema della più piccola rete coerente con tali tabelle.
20. Per la rete di Figura 4.48, supponete che tutte le tabelle di inoltro siano state determinate come nell'Esercizio 15 e che, poi, la linea C-E abbia un guasto. Scrivete
- le tabelle di A, B, D e F dopo che C ed E hanno annunciato la notizia.
 - le tabelle di A e D dopo il loro successivo scambio reciproco di informazioni.
 - la tabella di C dopo che A ha scambiato informazioni con C stesso.
21. Supponete che un router abbia costruito la tabella di instradamento mostrata in Tabella 4.14. Il router può consegnare pacchetti direttamente alle interfacce 0 e 1, oppure può inviare pacchetti ai router R2, R3 o R4. Descrivete cosa fa il router quando deve inoltrare un pacchetto indirizzato ad ognuna delle seguenti destinazioni.
- 128.96.39.10
 - 128.96.40.12
 - 128.96.40.151
 - 192.4.153.17
 - 192.4.153.90
- ✓ 22. Supponete che un router abbia costruito la tabella di instradamento mostrata in Tabella 4.15. Il router può consegnare pacchetti direttamente alle interfacce 0 e 1, oppure può inviare pacchetti ai router R2, R3 o R4. Descrivete cosa fa il router quando deve inoltrare un pacchetto indirizzato ad ognuna delle seguenti destinazioni.
- 128.96.171.92
 - 128.96.167.151
 - 128.96.163.151
 - 128.96.169.192
 - 128.96.165.121

Tabella 4.13 Tabelle di inoltro per l'Esercizio 19.

A			F		
Nodo	Costo	Salto successivo	Nodo	Costo	Salto successivo
B	1	B	A	2	C
C	1	C	B	3	C
D	2	B	C	1	C
E	3	C	D	2	C
F	2	C	E	1	E

Tabella 4.14 Tabella di instradamento per l'Esercizio 21.

Numero di sottorete	Maschera di sottorete	Salto successivo
128.96.39.0	255.255.255.128	Interfaccia 0
128.96.39.128	255.255.255.128	Interfaccia 1
128.96.40.0	255.255.255.128	R2
192.4.153.0	255.255.255.192	R3
(Default)		R4

Tabella 4.15 Tabella di instradamento per l'Esercizio 2.

Numero di sottorete	Maschera di sottorete	Salto successivo
128.96.170.0	255.255.254.0	Interfaccia 0
128.96.168.0	255.255.254.0	Interfaccia 1
128.96.166.0	255.255.254.0	R2
128.96.164.0	255.255.252.0	R3
(Default)		R4

- ★ 23. Considerate la semplice rete di Figura 4.50, in cui A e B si scambiano informazioni di instradamento mediante vettori di distanza. Tutte le linee hanno costo unitario. Supponete che la linea A-E abbia un guasto.
- Fornite una sequenza di aggiornamenti delle tabelle di instradamento che genera un instradamento ciclico fra A e B.
 - Stimate la probabilità dello scenario (a), ipotizzando che A e B inviano aggiornamenti di instradamento ad istanti casuali, con la medesima frequenza media.
 - Stimate la probabilità che si generi un ciclo nel caso in cui A emetta un aggiornamento broadcast entro 1 secondo dal momento in cui si accorge del guasto sulla linea A-E, mentre B emette aggiornamenti broadcast costantemente ogni 60 secondi.
24. Considerate la situazione in cui nella rete di Figura 4.15 si viene a creare un instradamento ciclico quando la linea A-E si guasta. Elencate *tutte* le sequenze di aggiornamenti delle tabelle scambiate fra A, B e C, relativamente alla destinazione E, che provocano il ciclo. Ipotizzate che gli aggiornamenti delle tabelle avvengano uno per volta, che tutti i nodi coinvolti usino la tecnica *split horizon* e che A invii a B una iniziale segnalazione sull'irraggiungibilità di E prima che lo faccia C. Potete ignorare gli aggiornamenti che non provocano modifiche.
25. Supponete che un insieme di router usi la tecnica dello *split horizon*; vogliamo qui considerare in quali circostanze l'uso, in aggiunta, della tecnica di *poison reverse* introduce differenze.
- Mostrate che, nei due esempi descritti nella Sezione 4.2.2 e nell'ipotesi che gli host coinvolti usino lo *split horizon*, il *poison reverse* non introduce differenze nell'evoluzione della situazione di instradamento ciclico.
 - Supponete che i router di suddivisione dell'orizzonte, A e B, raggiungano in qualche modo lo stato in cui inoltrano l'uno verso l'altro il traffico destinato ad un nodo X. Descrivete come evolve questa situazione nel caso venga o non venga usata la tecnica di *poison reverse*.
 - Descrivete una sequenza di eventi che portano A e B in uno stato ciclico come quello del caso (b), anche se viene usato il *poison reverse*. Suggerimento: supponete che A e B siano connessi mediante una linea molto lenta. Ciascuno di essi raggiunge X attraverso un terzo nodo, C, e pubblicizzano simultaneamente l'un l'altro i propri percorsi.

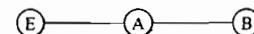


Figura 4.50 Semplice rete per l'Esercizio 23.

26. La tecnica di *hold-down* è un'altra tecnica a vettore di distanza utilizzata per evitare i cicli, nella quale gli host ignorano gli aggiornamenti per un certo periodo di tempo, durante il quale le nuove notizie di guasto di linee hanno la possibilità di propagarsi. Considerate le reti di Figura 4.51, dove tutte le linee hanno costo unitario, tranne E-D che ha costo 10. Supponete che la linea E-A si interrompa e che B segnali immediatamente ad A il proprio percorso verso E, che va a generare un ciclo (si tratta del falso percorso, quello che passa per A). Specificate i dettagli per un'interpretazione della tecnica *hold-down* e usatela per descrivere l'evoluzione dell'instradamento ciclico in entrambe le reti. In quale modo la tecnica di *hold-down* è in grado di prevenire la formazione del ciclo nella rete EAB senza ritardare la scoperta del percorso alternativo nella rete EABD?
27. Considerate la rete di Figura 4.52, dove viene usato l'instradamento a stato delle linee. Supponete che la linea B-F si guasti e che accadono i seguenti eventi, in sequenza:
- Il nodo H viene aggiunto sul lato destro mediante una connessione a G.
 - Il nodo D viene aggiunto sul lato sinistro mediante una connessione a C.
 - Viene aggiunta una nuova linea D-A.
- La linea guasta, B-F, viene ora ripristinata. Descrivete quali pacchetti relativi allo stato delle linee verranno inondati in avanti e all'indietro. Ipotizzate che il numero di sequenza iniziale sia 1 in tutti i nodi, che per nessun pacchetto scada il timeout e che entrambi gli estremi di una linea usino lo stesso numero di sequenza nei propri LSP per tale linea, numero maggiore di qualsiasi altro numero di sequenza mai usato prima.
28. Indicate, come nella Tabella 4.9, i passi eseguiti dall'algoritmo di ricerca in avanti mentre viene costruita la base di dati di instradamento per il nodo A nella rete mostrata in Figura 4.53.

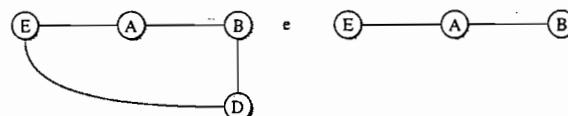


Figura 4.51 Rete per l'Esercizio 26.

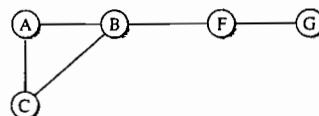


Figura 4.52 Rete per l'Esercizio 27.

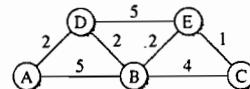


Figura 4.53 Rete per l'Esercizio 28.

- ✓ 29. Indicate, come nella Tabella 4.9, i passi eseguiti dall'algoritmo di ricerca in avanti mentre viene costruita la base di dati di instradamento per il nodo A nella rete mostrata in Figura 4.54.
30. Supponete che i nodi della rete mostrata nella Figura 4.55 partecipino all'instradamento a stato delle linee e che C riceva LSP contradditori: uno arriva da A e afferma che la linea A-B è guasta, mentre un altro arriva da B e afferma che la linea A-B è operativa.
- Come può accadere ciò?
 - Cosa dovrebbe fare C? Cosa si può aspettare che succeda C?
- Fate l'ipotesi che gli LSP non contengano alcuna informazione temporale sincronizzata.
31. Considerate la rete mostrata in Figura 4.56, in cui le linee orizzontali rappresentano fornitori di transito e le linee verticali numerate sono linee di collegamento fra fornitori.
- Quanti percorsi verso P può ricevere l'annunciatore BGP del fornitore Q?
 - Supponete che Q e P adottino la politica che il traffico uscente viene instradato verso la linea più vicina al fornitore del destinatario, minimizzando così i propri costi. Quali percorsi verranno seguiti dal traffico proveniente da A e diretto a B e proveniente da B e diretto ad A?
 - Cosa potrebbe fare Q perché il traffico diretto da B ad A usi la linea 1, più vicina?
 - Cosa potrebbe fare Q perché il traffico diretto da B ad A passi attraverso R?
32. Fornite un esempio di disposizione di router raggruppati in sistemi autonomi che dia luogo ad un percorso tra il punto A e il punto B che abbia il minor numero di salti e che attraversi due volte lo stesso AS. Spiegate cosa farebbe il protocollo BGP in questa situazione.
- ★ 33. Sia *A* il numero di sistemi autonomi nella rete Internet e *D* la massima lunghezza di un percorso in termini di numero di AS.

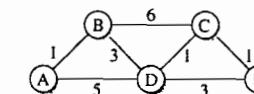


Figura 4.54 Rete per l'Esercizio 29.

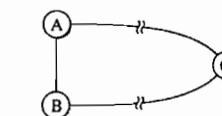


Figura 4.55 Rete per l'Esercizio 30.

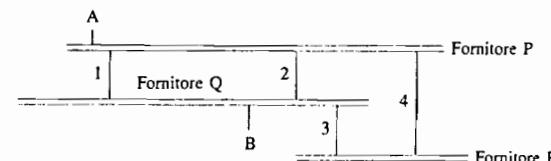


Figura 4.56 Rete per l'Esercizio 31.

- a) Descrivete un modello di connettività per il quale D sia dell'ordine del logaritmo di A ed un altro per cui D sia dell'ordine della radice quadrata di A .
- b) Ipotizzando che il numero di ogni AS sia composto da 2 byte e che il numero di ogni rete sia di 4 byte, fornite una stima della quantità di dati che un annunciatore BGP deve ricevere per tenere traccia del percorso di sistemi autonomi che porta ad ogni rete. Esprimete la vostra risposta in termini di A , D ed N (il numero di reti).
34. Supponete che i router IP assumano informazioni relative alle reti IP e alle sottoreti nello stesso modo in cui i *learning bridge* di Ethernet assumono informazioni relative agli host: annotando la comparsa di nuovi host e l'interfaccia da cui si rendono evidenti. Confrontate questo apprendimento con l'apprendimento caratteristico dei router che usano i vettori di distanza
- per un sito foglia con un'unica connessione ad Internet.
 - per l'uso interno di una organizzazione che non si connette ad Internet.
- Supponete che i router ricevano dagli altri router solamente informazioni relative alla comparsa di nuove reti e che i router che generano tali informazioni ricevano informazioni sulla propria rete IP per mezzo di opportuna configurazione.
35. Gli host IP che non sono router designati *devono* eliminare i pacchetti ad essi erroneamente indirizzati, anche nel caso in cui siano in grado di inoltrarli correttamente. In assenza di questo obbligo, cosa accadrebbe se un pacchetto indirizzato all'indirizzo IP A venisse inavvertitamente trasmesso in modalità broadcast dallo strato di linea di collegamento? Quali altre giustificazioni potete addurre per questo obbligo?
36. Consultate la documentazione (pagina man o altro) del programma netstat di Linux/Windows. Usate netstat per visualizzare l'attuale tabella di instradamento del vostro host e spiegate il significato di ogni informazione presente. Qual è, dal punto di vista pratico, il numero minimo di informazioni che sono presenti?
37. Usate il programma Unix traceroute (tracert in Windows) per determinare quanti salti siano necessari dal vostro host per raggiungere altri host in Internet (ad esempio cs.princeton.edu o www.cisco.com). Quanti router attraversate, soltanto per uscire dal vostro sito locale? Consultate la documentazione (pagina man o altro) del programma traceroute e spiegate come sia implementato.
38. Cosa accade se traceroute viene usato per trovare il percorso verso un indirizzo non assegnato? Ha importanza che sia non assegnata la parte di rete dell'indirizzo, oppure la sola parte di host?
39. La Figura 4.57 mostra un sito, dove R1 e R2 sono router e RB è un *bridge router*: instrada il traffico ad esso indirizzato e svolge la funzione di bridge per traffico diverso. All'interno del sito ven-

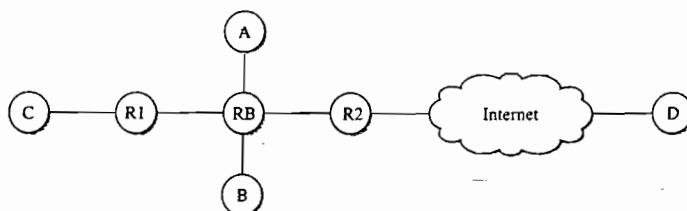


Figura 4.57 Rete per l'Esercizio 39.

- gono usate sottoreti e in ciascuna sottorete si usa ARP. Sfortunatamente, l'host A è stato configurato male e non usa le sottoreti. Quali host, tra B, C e D, può raggiungere A?
40. Un'azienda ha una rete di classe C, 200.1.1, e vuole creare sottoreti per quattro dipartimenti, con i seguenti host:
- 72 host
 - 35 host
 - 20 host
 - 18 host
- per un totale di 145 host.
- Descrivete una possibile configurazione di maschere di sottorete che risolva il problema.
 - Suggerite cosa può fare l'azienda se il dipartimento D cresce fino ad avere 34 host.
41. Supponete che gli host A e B si trovino in una LAN Ethernet con indirizzo di rete IP di classe C uguale a 200.0.0. Si vuole connettere un host C alla rete mediante una connessione diretta verso B (come in Figura 4.58). Spiegate come farlo usando sottoreti e fornite un esempio di assegnazione delle sottoreti stesse. Ipotizzate che non sia disponibile un ulteriore indirizzo di rete. Cosa comporta questo per la dimensione della LAN Ethernet?
42. Un metodo alternativo per connettere l'host C del precedente esercizio consiste nell'utilizzo della tecnica di *proxy ARP* con instradamento: B instrada il traffico proveniente da C e diretto a C e risponde anche alle richieste ARP relative a C che riceve sulla rete Ethernet.
- Indicate tutti i pacchetti inviati, con i relativi indirizzi fisici, mentre A usa ARP per localizzare C e, poi, inviargli un pacchetto.
 - Descrivete la tabella di instradamento di B. Cosa deve contenere di particolare?
43. Proponete uno schema di indirizzamento plausibile per IPv6 che, però, rimanga presto privo di bit disponibili. In particolare, fornite uno schema come quello di Figura 4.33, magari con ulteriori campi identificativi, che raggiunga un totale di più di 128 bit, con giustificazioni plausibili per la dimensione di ogni campo. Potete immaginare che i campi siano tra loro suddivisi sui confini tra byte e che il campo InterfaceID sia lungo 64 bit. Suggerimento: Considerate i campi che raggiungono la loro occupazione massima soltanto in condizioni molto particolari e poco probabili. Potete farlo se il campo InterfaceID è di 48 bit?
44. Supponete che due sottoreti condividano la stessa LAN fisica; gli host di ciascuna sottorete vedranno, quindi, i pacchetti broadcast dell'altra sottorete.
- Come si comporterà il protocollo DHCP se sulla LAN condivisa coesistono due server, uno per ciascuna sottorete? Quali problemi potrebbero nascere [ne nascono!]?
 - Tale condivisione crea problemi al protocollo ARP?

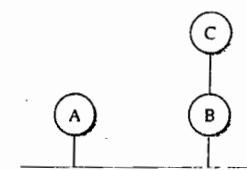


Figura 4.58 Rete per l'Esercizio 41.

45. La Tabella 4.16 è una tabella di instradamento che usa CIDR. I byte degli indirizzi sono espressi in esadecimale. La notazione “/12” in C4.50.0.0/12 indica una maschera di sottorete con 12 bit iniziali al valore 1, cioè FF.F0.0.0. Noteate che le ultime tre righe della tabella corrispondono a qualsiasi indirizzo, per cui svolgono la funzione di percorso di default. Indicate verso quale salto successivo verranno inoltrati i pacchetti destinati agli indirizzi seguenti.

- a) C4.5E.13.87
- b) C4.5E.22.09
- c) C4.41.80.02
- d) 5E.43.91.12
- e) C4.6D.31.2E
- f) C4.6B.31.2E

✓ 46. La Tabella 4.17 è una tabella di instradamento che usa CIDR. I byte degli indirizzi sono espressi in esadecimale. La notazione “/12” in C4.50.0.0/12 indica una maschera di sottorete con 12 bit iniziali al valore 1, cioè FF.F0.0.0. Indicate verso quale salto successivo verranno inoltrati i pacchetti destinati agli indirizzi seguenti.

- a) C4.4B.31.2E
- b) C4.5E.05.09
- c) C4.4D.31.2E
- d) C4.5E.03.87
- e) C4.5E.7F.12
- f) C4.5E.D1.02

Tabella 4.16 Tabella di instradamento per l'Esercizio 45.

Lunghezza della maschera di sottorete	Salto successivo
C4.50.0.0/12	A
C4.5E.10.0/20	B
C4.60.0.0/12	C
C4.68.0.0/14	D
80.0.0.0/1	E
40.0.0.0/2	F
00.0.0.0/2	G

Tabella 4.17 Tabella di instradamento per l'Esercizio 46.

Lunghezza della maschera di sottorete	Salto successivo
C4.5E.2.0/23	A
C4.5E.4.0/22	B
C4.5E.C0.0/19	C
C4.5E.40.0/18	D
C4.4C.0.0/14	E
C0.0.0.0/2	F
80.0.0.0/1	G

47. Supponete che P, Q e R siano fornitori di servizi di rete, con l'assegnazione, rispettivamente, degli indirizzi CIDR C1.0.0.0/8, C2.0.0.0/8 e C3.0.0.0/8 (usando la notazione dell'Esercizio 45). I clienti di ciascun fornitore ricevono, inizialmente, assegnazione di indirizzi che sono sottoreti di quelli del fornitore. P ha i seguenti clienti:

- PA, con assegnazione C1.A3.0.0/16
- PB, con assegnazione C1.B0.0.0/12

Q ha i seguenti clienti:

- QA, con assegnazione C2.0A.10.0/20
- QB, con assegnazione C2.0B.0.0/16

Ipotizzate che non ci siano altri clienti, né fornitori.

- a) Fornite le tabelle di instradamento di P, Q e R, nell'ipotesi che ogni fornitore si connetta ad entrambi gli altri fornitori.
- b) Ipotizzate ora che P sia connesso a Q e che Q sia connesso a R, ma che P e R non siano direttamente connessi. Fornite le tabelle per P e R.
- c) Supponete che il cliente PA acquisti una linea diretta verso Q e che QA acquisti una linea diretta verso P, oltre alle linee esistenti. Fornite le tabelle per P e Q, ignorando R.

48. Nel problema precedente, ipotizzate che ogni fornitore sia connesso ad entrambi gli altri fornitori. Supponete che il cliente PA cambi fornitore, passando a Q, e che il cliente QB cambi anch'esso fornitore, passando a R. Usate la regola CIDR della più lunga corrispondenza per fornire le tabelle di instradamento per tutti i tre fornitori, che consentano a PA e QB di cambiare fornitore senza cambiamenti di numerazione.

49. Supponete che la maggior parte della rete Internet usi qualche forma di indirizzamento geografico, ma che una grande organizzazione internazionale abbia un unico indirizzo di rete IP e instradì il proprio traffico interno lungo le proprie linee.

- a) Spiegate l'inefficienza di instradamento del traffico entrante nell'organizzazione in seguito a questa configurazione.
- b) Spiegate come l'organizzazione potrebbe risolvere il problema per il traffico uscente.
- c) Cosa dovrebbe accadere perché il vostro metodo proposto al punto precedente possa funzionare per il traffico entrante?
- d) Supponete che la grande organizzazione modifichi ora il proprio indirizzamento, usando indirizzi geograficamente distinti per ciascuna sede. Come dovrebbe essere strutturato l'instradamento interno per continuare ad instradare internamente il traffico interno?

50. Il sistema telefonico usa un indirizzamento geografico. Per quale motivo pensate che questo schema non sia stato adottato in Internet?

51. Supponete che un sito A sia *multihomed*, avendo due connessioni ad Internet verso due diversi fornitori, P e Q. Usate l'indirizzamento basato sul fornitore, come nell'Esercizio 47, considerando che A riceva l'assegnazione dei propri indirizzi da P. Q ha una riga corrispondente ad A che indica l'instradamento mediante CIDR con il prefisso più lungo.

- a) Descrivete quale traffico entrante fluisce lungo la connessione A-Q. Considerate i casi in cui Q pubblicizza e non pubblicizza A al resto del mondo, tramite BGP.
- b) Qual è la minima informazione del proprio percorso verso A che Q deve effettuare per fare in modo che tutto il traffico entrante raggiunga A attraverso Q nel caso in cui si guasti la linea P-A?
- c) Quali problemi devono essere risolti se A vuole usare entrambe le proprie linee per il traffico uscente?

52. Un ISP con un indirizzo di classe B sta collaborando con una nuova azienda per assegnare ad essa una porzione di spazio di indirizzamento basata su CIDR. La nuova azienda necessita di indirizzi IP per calcolatori di tre reparti della propria rete aziendale: Sviluppo, Marketing e Vendite. Questi reparti prevedono di crescere nel modo seguente: il reparto Sviluppo ha 5 calcolatori all'inizio dell'anno 1 e prevede di aggiungere un calcolatore ogni settimana; il reparto Marketing non avrà mai bisogno di più di 16 calcolatori; il reparto Vendite ha bisogno di un calcolatore ogni due clienti. All'inizio dell'anno 1 l'azienda non ha clienti, ma il modello di vendita indica che all'inizio dell'anno 2 l'azienda avrà sei clienti e da quel momento in poi ogni settimana avrà un nuovo cliente con probabilità uguale al 60%, perderà un cliente con probabilità uguale al 20% e manterrà inalterato il numero di clienti con probabilità uguale al 20%.
 a) Quale intervallo di indirizzi sarebbe necessario per fornire supporto ai piani di crescita dell'azienda per almeno sette anni se il reparto Marketing usa tutti i propri 16 indirizzi e gli altri reparti rispettano le previsioni?
 b) Per quanto tempo risulterà adeguata questa assegnazione di indirizzi? Quando l'azienda esaurirà il proprio spazio di indirizzamento, come saranno stati assegnati gli indirizzi ai tre reparti?
 c) Se non fosse disponibile un indirizzamento CIDR per il piano settennale, quali opzioni sarebbero disponibili per l'azienda in termini di ottenimento di spazio di indirizzamento?
53. Proponete un algoritmo di ricerca per una tabella di inoltro CIDR che non richieda una ricerca lineare dell'intera tabella per trovare la corrispondenza più lunga.
- ★ 54. Supponete che una rete N all'interno di una grande organizzazione A acquisisca una propria connessione diretta ad un fornitore di servizi Internet, in aggiunta alla connessione esistente tramite A. Sia R1 il router che connette N al proprio fornitore e sia R2 il router che connette N alla parte restante di A.
 a) Ipotizzando che N continui ad essere una sottorete di A, come dovrebbero essere configurati R1 e R2? Quali limitazioni continuerebbero a sussistere in merito all'utilizzo da parte di N della propria connessione privata? Sarebbe impedito ad A l'utilizzo della connessione privata di N? Specificate la vostra configurazione in termini di cosa R1 e R2 dovrebbero pubblicizzare, e con quali percorsi. Ipotizzate che sia disponibile un meccanismo simile a BGP.
 b) Supponete ora che N usi un proprio numero di rete: come cambia la vostra risposta al punto (a)?
 c) Descrivete una configurazione dei router che consenta ad A di usare la linea di N quando la propria linea è guasta.
- ✓ 55. Considerate la internetwork mostrata in Figura 4.59, in cui le sorgenti D ed E inviano pacchetti al gruppo multicast G, i cui membri sono indicati in grigio chiaro. Mostrate, per ciascuna sorgente, gli alberi con i più brevi percorsi multicast.
56. Considerate la internetwork mostrata in Figura 4.60, in cui le sorgenti S1 ed S2 inviano pacchetti al gruppo multicast G, i cui membri sono indicati in grigio chiaro. Mostrate, per ciascuna sorgente, gli alberi con i più brevi percorsi multicast.
57. Supponete che l'host A voglia inviare pacchetti ad un gruppo multicast, i cui riceventi sono i nodi foglia di un albero avente A come radice, con profondità N e con ciascun nodo con figli aventi k figli (ci sono quindi k^N riceventi).
 a) Quante singole linee di trasmissione sono coinvolte se A invia un messaggio multicast a tutti i riceventi?

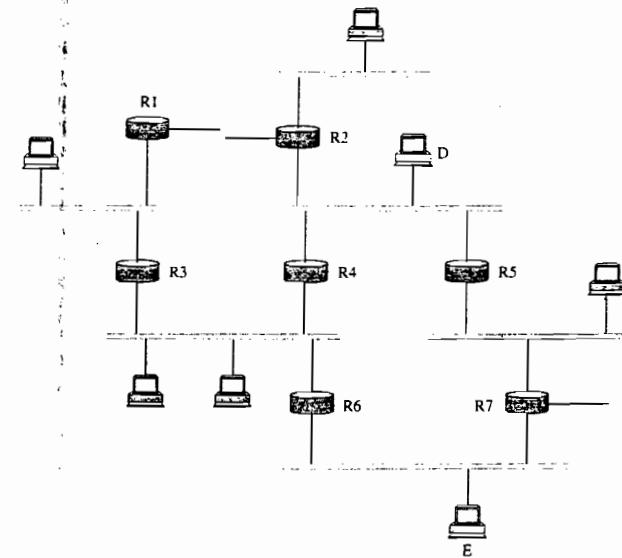


Figura 4.59 Internetwork per l'Esercizio 55.

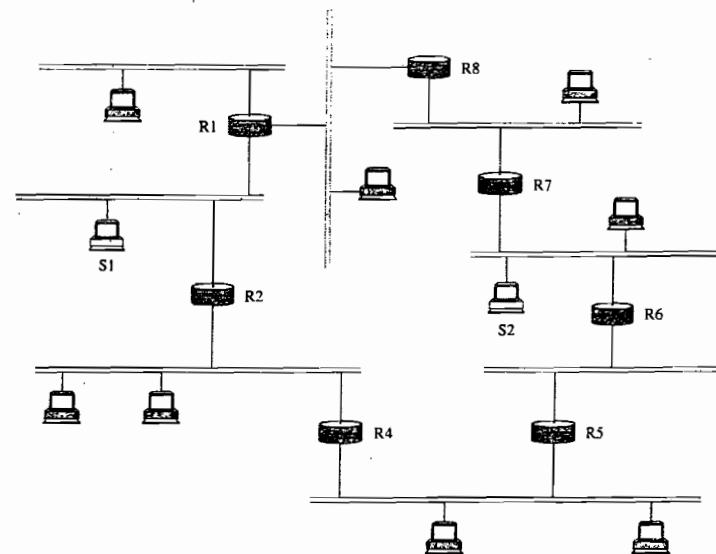


Figura 4.60 Internetwork per l'Esercizio 56.

- b) Quante singole linee di trasmissione sono coinvolte se A invia messaggi unicast a ciascun singolo ricevente?
- c) Supponete che A spedisca messaggi a tutti i riceventi, ma che alcuni messaggi vengano perduti e sia necessaria una ritrasmissione. La trasmissione unicast verso quale frazione dei riceventi è equivalente, in termini di singole linee di trasmissione, alla ritrasmissione multicast a tutti i riceventi?
58. Determinate se i seguenti indirizzi IPv6 sono corretti oppure no.
- ::0F53:6382:AB00:67DB:BB27:7332
 - 7803:42F2::88EC:D4BA:B75D:11CD
 - ::4BA8:95CC::DB97:4EAB
 - 74DC::02BA
 - ::00FF:128.112.92.116
59. Determinate se il vostro sito è connesso alla rete MBone. Se lo è, fate esperimenti con gli strumenti di MBone, come *sdr*, *vat* e *vic*.
60. Le etichette MPLS sono solitamente lunghe 20 bit. Spiegate per quale motivo vengono così fornite sufficiente etichette quando MPLS viene usato per l'inoltro basato sulla destinazione.
61. A volte è stato dichiarato che MPLS migliora le prestazioni dei router. Spiegate perché ciò potrebbe essere vero, e suggerite motivi per cui in pratica ciò potrebbe non accadere.
62. Ipotizzate che, quando viene usata l'intestazione "a guarnizione" di Figura 4.42(b), servano 32 bit per trasportare ciascuna etichetta MPLS allegata ad un pacchetto.
- Quanti ulteriori byte sono necessari per inviare un pacchetto lungo un tunnel usando la tecnica MPLS descritta nella Sezione 4.5.3?
 - Quanti ulteriori byte sono necessari, come minimo, per inviare un pacchetto lungo un tunnel usando un'ulteriore intestazione IP come descritto nella Sezione 4.1.8?
 - Calcolate l'efficienza dell'utilizzo di banda per ciascuno dei due approcci di tunneling quando la dimensione media del pacchetto è 300 byte. Ripetete il calcolo per pacchetti di 64 byte. L'efficienza di banda è definita come il numero di byte trasportati come carico utile diviso per il numero di byte trasportati in totale.
63. Il documento RFC 791 descrive Internet Protocol e contiene due opzioni per l'instradamento dalla sorgente. Descrivete tre svantaggi derivanti dall'utilizzo delle opzioni IP per l'instradamento dalla sorgente rispetto all'uso di MPLS per l'instradamento esplicito. Suggerimento: l'intestazione IP che contiene le opzioni può essere lunga, al massimo, 15 parole.

Protocolli di trasporto

Problema Far comunicare i processi

I tre capitoli precedenti hanno descritto diverse tecnologie che possono essere utilizzate per connettere insieme più calcolatori: linee di collegamento dirette (tra cui le tecnologie LAN, come Ethernet e token ring), reti a commutazione di pacchetto (tra cui le reti basate su celle, come ATM) e internetwork. Il problema successivo consiste nella trasformazione di questo servizio di consegna di pacchetti tra host in un canale di comunicazione tra processi: questo è il ruolo svolto dal livello di *trasporto* dell'architettura di rete, che viene anche detto protocollo *end-to-end*, "da un estremo all'altro", al momento che fornisce il supporto alla comunicazione tra programmi applicativi posti ai due estremi del canale.

Il protocollo di trasporto assume forma per effetto di due esigenze. Da un lato (il "lato superiore"), i processi a livello applicativo che usano i suoi servizi hanno certi requisiti. L'elenco seguente schematizza alcune delle proprietà più comuni che ci si aspetta siano fornite da un protocollo di trasporto:

- garanzia di consegna del messaggio
- consegna dei messaggi nello stesso ordine in cui vengono inviati
- consegna di una sola copia di ciascun messaggio
- supporto per i messaggi di dimensione arbitraria
- supporto per la sincronizzazione fra il mittente e il destinatario
- possibilità, per il ricevente, di applicare un controllo di flusso nei confronti del mittente
- supporto per più processi applicativi in ciascun host

Noteate che questo elenco non contiene tutte le funzionalità che i processi applicativi possono richiedere alla rete. Ad esempio, non comprende la sicurezza, che viene solitamente fornita da protocolli situati al di sopra del livello di trasporto.

Dal "lato inferiore", la rete sottostante su cui opera il protocollo di trasporto ha precise limitazioni in relazione al livello di servizio che può fornire.

Alcune delle più tipiche limitazioni delle reti consistono nel fatto che possono:

- perdere messaggi
- modificare l'ordine dei messaggi
- consegnare più copie di uno stesso messaggio
- impostare un limite finito alla dimensione dei messaggi
- consegnare i messaggi con un ritardo indefinitamente lungo

Una rete di questo tipo fornisce un livello di servizio di tipo *best-effort*, di cui la rete Internet costituisce un classico esempio.

La sfida, quindi, consiste nello sviluppo di algoritmi che trasformino le proprietà, assai poco lusinghiere, della rete sottostante nel servizio di alto livello richiesto dai programmi applicativi: diversi protocolli di trasporto usano diverse combinazioni di questi algoritmi. Questo capitolo esamina questi algoritmi nel contesto di tre servizi rappresentativi: un semplice servizio di demultiplexing asincrono, un servizio di flusso affidabile di byte e un servizio di tipo richiesta/risposta.

Nel caso dei servizi di demultiplexing e di flusso di byte, usiamo, rispettivamente, i protocolli UDP e TCP di Internet per illustrare, dal punto di vista pratico, come vengano forniti questi servizi. Nel terzo caso esaminiamo dapprima una raccolta di algoritmi che implementano i servizi di tipo richiesta/risposta (e altri servizi correlati), per poi mostrare come questi algoritmi si possano combinare per implementare un protocollo di invocazione remota di procedure (RPC, Remote Procedure Call). Questa presentazione termina con la descrizione di due protocolli RPC molto utilizzati, SunRPC e DCE-RPC, in termini di questi algoritmi constituenti. Infine, il capitolo si conclude con una sezione che discute le prestazioni dei diversi protocolli di trasporto.

5.1 Semplice demultiplexing (UDP)

Il più semplice protocollo di trasporto che si possa immaginare estende il servizio di consegna fra host svolto dalla rete sottostante in un servizio di comunicazione tra processi. Essendo molto probabile che vi siano molti processi in esecuzione in un host qualsiasi, il protocollo deve aggiungere un livello di demultiplexing, consentendo così la condivisione della rete fra più processi applicativi presenti in ciascun host. Oltre a questo requisito, il protocollo di trasporto non aggiunge alcuna altra funzionalità al servizio best-effort fornito dalla rete sottostante. Il protocollo User Datagram Protocol (UDP) di Internet è un esempio di questo tipo di protocollo di trasporto.

L'unica cosa interessante in questo protocollo è la forma di indirizzo usato per identificare il processo che svolge attività in rete. Sebbene sia possibile che i processi si identifichino l'uno l'altro *in modo diretto*, mediante un identificativo di processo (pid, *process id*) assegnato dal sistema operativo, tale approccio ha interesse pratico soltanto in un sistema distribuito chiuso, in cui tutti gli host eseguono un unico sistema operativo, che assegna a ciascun processo un identificativo univoco. Un approccio molto più comune, che è quello utilizzato da UDP, prevede che i processi si identifichino l'uno l'altro *indirettamente*, usando un localizzatore astratto, spesso chiamato *porta* o *casella postale* (*mailbox*): un processo sorgente invia un messaggio ad una porta e il processo destinatario riceve il messaggio da una porta.

L'intestazione per un protocollo di trasporto (*end-to-end*) che realizzzi questa funzione di demultiplexing contiene tipicamente un identificativo (porta) sia per il mittente (sorgente) sia per il destinatario (ricevente) del messaggio. Ad esempio, la Figura 5.1 mostra l'intestazione UDP; notate che il campo che ospita la porta UDP è lungo soltanto 16 bit: ciò significa che esistono fino a 64K possibili porte, ovviamente non sufficienti ad identificare tutti i processi di tutti gli host di Internet. Fortunatamente, le porte non assumono significato sull'intera Internet, ma su un singolo host: in sostanza, un processo viene effettivamente identificato da una porta su un certo host, cioè da una coppia (porta, host). A tutti gli effetti, questa coppia costituisce le chiavi di demultiplexing per il protocollo UDP.

Il problema successivo riguarda il modo in cui un processo apprende il valore della porta del processo a cui vuole inviare un messaggio. Tipicamente, un processo con funzioni di *client* inizia uno scambio di messaggi con un processo *server*. Una volta che il client ha contattato il server, il server conosce la porta del client (perché era contenuta nell'intestazione del messaggio) e può rispondere. Un approccio assai diffuso consiste nel fatto che il server accetti messaggi da una qualche porta *ben nota* (*well-known port*), cioè ogni server riceve i propri messaggi da una porta prefissata che viene ampiamente pubblicizzata, in modo simile al servizio telefonico di emergenza, che è disponibile al numero di telefono ben conosciuto 113 (NdT: 911 negli Stati Uniti). Nella rete Internet, ad esempio, il Domain Name Server (DNS) riceve messaggi dalla ben nota porta 53 su ciascun host, il servizio di posta elettronica è in ascolto per ricevere messaggi dalla porta 25 e il programma *talk* di Unix accetta messaggi in arrivo alla ben nota porta 517; e così via. Questa corrispondenza viene periodicamente pubblicata in un RFC ed è disponibile nella maggioranza dei sistemi Unix all'interno del file */etc/services*. Alcune volte una porta ben nota rappresenta soltanto il punto iniziale per una comunicazione: il client e il server usano la porta ben nota per accordarsi su qualche altra porta che useranno per le comunicazioni successive, lasciando la porta ben nota libera per altri client.

Una strategia alternativa consiste nella generalizzazione di questa idea, in modo che vi sia un'unica porta ben nota, quella da cui accetta messaggi il servizio di "corrispondenza delle porte" (Port Mapper). Un client invia un messaggio alla ben nota porta del Port Mapper, chiedendo di conoscere la porta da usare per parlare con un "qualsiasi" servizio, e il Port Mapper risponde fornendo la porta appropriata. Questa strategia rende semplice modificare nel tempo la porta associata a servizi diversi e consente a ciascun host di usare una diversa porta per lo stesso servizio.

Come appena detto, una porta è semplicemente un'astrazione. Come sia implementata esattamente dipende dal sistema o, più precisamente, dal sistema operativo. Ad esempio, la

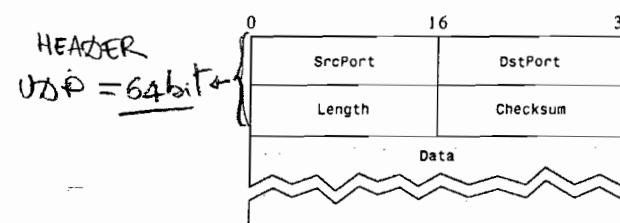


Figura 5.1 Formato dell'intestazione UDP.

API *socket* descritta nel Capitolo 1 è un'implementazione delle porte. Tipicamente una porta viene implementata mediante una coda di messaggi, come illustrato in Figura 5.2. Quando arriva un messaggio, il protocollo (ad esempio, UDP) accoda il messaggio al termine della coda; se la coda fosse piena, il messaggio verrebbe ignorato (non esiste alcun meccanismo di controllo di flusso che dica al mittente di rallentare). Quando un processo applicativo vuole ricevere un messaggio, ne viene prelevato uno dal fronte della coda (se la coda è vuota, il processo rimane bloccato finché non vi è un messaggio disponibile).

In fine, anche se UDP non implementa il controllo di flusso, né la consegna affidabile o in sequenza corretta, fa qualcosa in più del semplice demultiplexing dei messaggi destinati a processi applicativi: assicura anche la correttezza del messaggio, per mezzo di una somma di controllo (la somma di controllo in UDP è attualmente facoltativa in Internet, ma diverrà obbligatoria con IPv6). Il protocollo UDP calcola la propria somma di controllo sull'intera intestazione UDP, sul contenuto del corpo del messaggio e su una parte chiamata *pseudointestazione (pseudoheader)*. La pseudointestazione è composta da tre campi dell'intestazione IP, il numero di protocollo, l'indirizzo IP del mittente e l'indirizzo IP del destinatario, e da un campo che contiene la lunghezza del pacchetto UDP (quindi, in effetti, il campo "lunghezza" viene incluso due volte nel calcolo della somma di controllo). La somma di controllo di UDP viene calcolata con lo stesso algoritmo usato da IP, come definito nella Sezione 2.4.2. La motivazione che giustifica la presenza della pseudointestazione consiste nell'esigenza di verificare che il messaggio sia stato trasmesso tra i due estremi finali corretti. Ad esempio, se l'indirizzo IP del destinatario fosse stato modificato durante il trasporto del pacchetto, provocando l'errata consegna del pacchetto stesso, questo fatto verrebbe scoperto dalla somma di controllo di UDP.

*PSEUDO
HEADER*

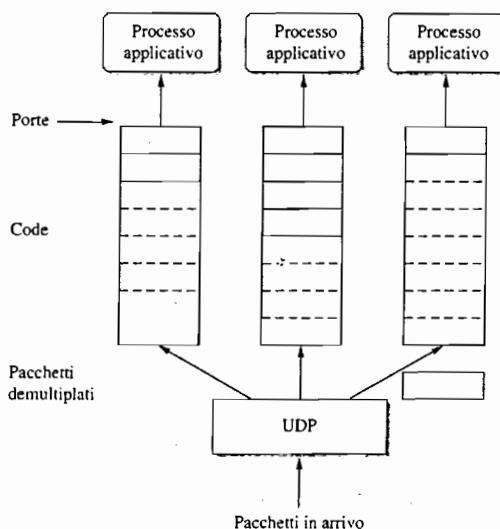


Figura 5.2 La coda dei messaggi UDP.

FULL-DUPLEX

5.2 Flusso affidabile di byte (TCP)

Diversamente da un semplice protocollo di demultiplexing, quale è UDP, un protocollo di trasporto più sofisticato offre un servizio di flusso di byte affidabile e orientato alla connessione. Tale servizio si è dimostrato utile in un'ampia varietà di applicazioni, poiché libera l'applicazione dal compito di gestire i dati fuori ordine o mancanti. Il Transmission Control Protocol (TCP) di Internet è probabilmente il più utilizzato protocollo di questo tipo, ed è anche quello ottimizzato con maggior cura. Per queste due ragioni questa sezione studia in dettaglio il protocollo TCP, anche se al termine della sezione identifieremo e discuteremo scelte di progetto alternative.

In termini delle proprietà dei protocolli di trasporto enunciate all'inizio di questo capitolo, TCP garantisce la consegna affidabile e in sequenza di un flusso di byte. È un protocollo full-duplex, per cui ciascuna connessione TCP fornisce una coppia di flussi di byte, una in ciascuna direzione. Il protocollo TCP prevede anche un meccanismo di controllo di flusso per ciascuno di questi flussi di byte, consentendo al ricevitore di limitare la quantità di dati che il mittente può inviare in un certo istante. Infine, come UDP, il protocollo TCP fornisce il supporto ad un meccanismo di demultiplexing che consente a più programmi applicativi in un host di sostenere simultaneamente una conversazione con le loro pari entità. Oltre alle caratteristiche appena elencate, il protocollo TCP realizza anche un meccanismo di controllo della congestione messo a punto con grande cura. L'idea su cui si basa questo meccanismo è quella di controllare la velocità con cui TCP invia i dati, non per evitare che il mittente sovraccarichi il ricevente, ma per impedire al mittente di sovraccaricare la rete. La descrizione del meccanismo di controllo della congestione di TCP è posposto al Capitolo 6, dove verrà presentato nel più ampio contesto dell'allocazione equa delle risorse di rete.

Z.H. controllo di flusso / di congestione

Dato che molte persone confondono il controllo di congestione e il controllo di flusso, enunciamo nuovamente la differenza: Il controllo di flusso si occupa di evitare che il mittente esaurisca la capacità dei ricevitori. Il controllo di congestione si occupa di evitare che vengano inseriti nella rete troppi dati, provocando così un sovraccarico dei commutatori o delle linee di collegamento. Di conseguenza, il controllo di flusso è un problema della comunicazione end-to-end, mentre il controllo della congestione ha a che fare con l'interazione tra gli host e le reti.

5.2.1 Problemi end-to-end

Nel cuore di TCP si colloca l'algoritmo a finestra scorrevole (*sliding window*). Nonostante si tratti, fondamentalmente, dello stesso algoritmo che abbiamo visto nella Sezione 2.5.2, esistono molte differenze importanti, dovute al fatto che TCP viene eseguito in Internet piuttosto che su una linea di collegamento punto-punto. Questa sottosezione identifica queste differenze e spiega come, di conseguenza, TCP risulti complicato. Le sottosezioni seguenti, poi, descrivono come il protocollo TCP risolva queste e altre complicazioni.

Innanzitutto, mentre l'algoritmo sliding window presentato nella Sezione 2.5.2 viene eseguito su una singola linea fisica di connessione che connette sempre gli stessi due calcolatori, TCP consente connessioni logiche fra processi che sono in esecuzione su due calcolatori qualsiasi in Internet. Ciò significa che TCP necessita di un'esplicita fase di instaurazione della connessione, durante la quale le due parti coinvolte nella connessione si accordano per scambiarsi dati reciprocamente. Questa differenza è analoga alla necessità di comporre un

numero di telefono invece di avere una linea telefonica dedicata. Il protocollo TCP ha anche un'esplicita fase di terminazione della connessione. Una delle cose che accadono durante l'instaurazione della connessione è la condivisione di informazioni di stato fra le due parti, in modo da consentire all'algoritmo sliding window di iniziare la propria esecuzione. La terminazione della connessione è, invece, necessaria perché ciascun host sappia di poter eliminare queste informazioni di stato.

La seconda differenza consiste nel fatto che, mentre una singola linea fisica che connette continuamente gli stessi due calcolatori ha un prefissato valore di RTT, le connessioni TCP hanno, molto probabilmente, valori molto diversi di tempi di round-trip. Ad esempio, una connessione TCP fra un host a San Francisco e un host a Boston, che sono separate da diverse migliaia di chilometri, potrebbe avere un RTT di 100 ms, mentre una connessione TCP fra due host che si trovano nella stessa stanza, distanti soltanto pochi metri, potrebbe avere un RTT di 1 ms soltanto. Il medesimo protocollo TCP deve essere in grado di consentire il funzionamento di entrambe queste connessioni. Per peggiorare ancora le cose, la connessione TCP fra un host a San Francisco e un host a Boston potrebbe avere un RTT di 100 ms alle 3 del mattino, ma un RTT di 500 ms alle 3 del pomeriggio. Variazioni del valore di RTT sono addirittura possibili durante una singola connessione TCP che duri anche soltanto pochi minuti. Ciò significa, per l'algoritmo sliding window, che il meccanismo di timeout che fa scattare le ritrasmissioni deve essere adattativo (il timeout per una linea di connessione punto-punto deve essere, certamente, un parametro che si possa impostare, ma non è necessario che venga adattato per ciascuna particolare coppia di nodi).

Una terza differenza consiste nel fatto che i pacchetti possono essere riordinati mentre attraversano Internet, mentre la cosa non è possibile in una linea punto-punto, dove il primo pacchetto inserito ad un estremo della linea deve essere il primo che compare all'altro estremo. Pacchetti che siano fuori ordine di poco non provocano problemi, perché l'algoritmo sliding window è in grado di riordinare correttamente i pacchetti usando il numero di sequenza. Il vero problema è quanto possano essere fuori ordine i pacchetti, oppure, detto in altro modo, con quanto ritardo un pacchetto possa arrivare a destinazione. Nel caso peggiore, un pacchetto può venire ritardato da Internet finché il suo campo IP che contiene il tempo di vita (TTL, *time to live*) non scade, perché in tal caso il pacchetto viene eliminato, eliminando anche il pericolo dovuto al suo arrivo ritardato. Sapendo che il protocollo IP elimina i pacchetti dopo che il loro TTL è scaduto, il protocollo TCP ipotizza che ciascun pacchetto abbia un tempo di vita massimo. Il valore esatto di questo tempo di vita, denominato *tempo di vita massimo per un segmento* (MSL, *maximum segment lifetime*), è una scelta progettuale: l'impostazione attualmente consigliata è 120 secondi. Ricordate che il protocollo IP non impone direttamente questo valore di 120 secondi: si tratta, semplicemente, di una stima conservativa fatta da TCP di quanto un pacchetto possa vivere in Internet. La conseguenza di ciò è importante: il protocollo TCP deve essere pronto a gestire pacchetti molto vecchi che si presentino improvvisamente al ricevitore, creando potenzialmente confusione nell'algoritmo sliding window.

Quarta differenza: i calcolatori connessi a linee punto-punto sono, solitamente, progettati appositamente per gestire quella particolare linea. Ad esempio, se si calcola che il prodotto ritardo × ampiezza di banda di una linea sia 8 KB (per cui viene selezionata una finestra di dimensioni tali da consentire, in ogni istante, la presenza di 8 KB, al massimo, di dati non ancora confermati), allora è probabile che i calcolatori che si trovano ai due capi della linea abbiano la capacità di memorizzare in un buffer fino a 8 KB di dati: progettare il sistema in modo diverso sarebbe sbagliato. D'altra parte, praticamente qualsiasi tipo di

5.2 Flusso affidabile di byte (TCP)

calcolatore può essere collegato a Internet, rendendo assai variabile la quantità di risorse dedicate ad una connessione TCP, specialmente in considerazione del fatto che un host può potenzialmente sostenere centinaia di connessioni TCP contemporaneamente. Ciò significa che il protocollo TCP deve prevedere un meccanismo che ciascuna parte coinvolta nella comunicazione possa usare per "apprendere" quali risorse (ad esempio, quanto spazio di memorizzazione) possano essere destinate alla connessione dall'altra entità. Si tratta del problema del controllo di flusso.

Infine, dato che la parte trasmittente di una linea a connessione diretta non può inviare dati più velocemente di quanto consentito dall'ampiezza di banda della linea stessa e un solo host invia dati sulla linea, è impossibile congestionare la linea senza saperlo. Detto in altro modo, il carico sulla linea può essere visto dalla sorgente come una coda di pacchetti. Al contrario, da parte del mittente di una connessione TCP non si ha alcuna idea in merito a quali linee saranno attraversate per raggiungere la destinazione. Ad esempio, la macchina che spedisce potrebbe essere direttamente connessa ad una Ethernet relativamente veloce, e, quindi, in grado di inviare dati alla velocità di 100 Mbps, ma da qualche parte all'interno della rete deve essere attraversata una linea T1 a 1.5 Mbps. Per peggiorare le cose, i dati generati da molte diverse sorgenti potrebbero cercare di attraversare quella stessa linea lenta, dando luogo al fenomeno della congestione, la cui discussione è rimandata al Capitolo 6.

Terminiamo questa discussione relativa ai problemi dei protocolli di trasporto confrontando l'approccio usato da TCP per fornire un servizio di consegna affidabile e in sequenza con l'approccio usato dalle reti X.25. TCP assume che la rete IP sottostante non sia affidabile e possa consegnare i messaggi fuori sequenza, per cui usa l'algoritmo sliding window su ciascuna connessione per garantire la consegna affidabile e in sequenza. Al contrario, le reti X.25 usano il protocollo sliding window all'interno della rete, tra un salto e l'altro lungo il percorso. L'ipotesi che sta alla base di questo approccio è che se i messaggi vengono consegnati in modo affidabile e in sequenza fra ciascuna coppia di nodi lungo il percorso fra l'host sorgente e l'host destinatario, allora anche il servizio end-to-end garantirà la consegna affidabile e in sequenza.

Il problema di quest'ultimo approccio è che una sequenza di garanzie salto per salto non fornisce necessariamente una garanzia da un estremo all'altro della comunicazione: innanzitutto, se viene aggiunta ad un estremo del percorso una linea di collegamento eterogenea (ad esempio, una Ethernet), non c'è alcuna garanzia che tale salto mantenga il medesimo servizio svolto dagli altri salti. Secondariamente, il solo fatto che il protocollo sliding window garantisca che i messaggi vengano consegnati correttamente dal nodo A al nodo B, e poi dal nodo B al nodo C, non garantisce il perfetto funzionamento del nodo B. Ad esempio, si sa che i nodi di una rete possono introdurre errori nei messaggi mentre li trasferiscono da un buffer di ingresso ad un buffer di uscita, così come possono riordinare accidentalmente i messaggi. In conseguenza di queste piccole finestre di vulnerabilità, è necessario prevedere anche un vero controllo end-to-end per garantire un servizio affidabile e in sequenza, anche se i livelli inferiori del sistema implementano questa stessa funzionalità.

Questa discussione serve ad illustrare uno dei principi più importanti nella progettazione di sistemi: il *ragionamento end-to-end*. In sintesi, "ragionare end-to-end" significa affermare che una funzione (nel nostro esempio, la consegna affidabile e in sequenza) non dovrebbe essere fornita dai livelli inferiori del sistema a meno che non possa essere implementata completamente e correttamente in tali livelli. Di conseguenza, questa regola è a favore dell'approccio seguito in TCP/IP, anche se non si

tratta di una regola assoluta, perché ammette che alcune funzioni siano fornite in modo incompleto da un livello inferiore per motivi di ottimizzazione delle prestazioni. Per questo motivo, eseguire il controllo d'errore (es. CRC) sulla base di ogni salto è perfettamente coerente con il ragionamento end-to-end: rilevare un singolo pacchetto corrotto e ritrasmetterlo lungo un solo salto è preferibile a dover ritrasmettere un intero file da un estremo all'altro della comunicazione.

5.2.2 Formato del segmento

Il protocollo TCP è orientato ai byte: ciò significa che il mittente invia byte in una connessione TCP e il destinatario riceve byte dalla connessione TCP. Anche se il "flusso di byte" descrive il servizio offerto da TCP ai processi applicativi, il protocollo TCP non trasmette singoli byte su Internet, ma memorizza un numero sufficiente di byte ricevuti dal processo applicativo sull'host mittente finché non ha riempito un pacchetto di dimensioni ragionevoli, dopodiché invia tale pacchetto al proprio pari sull'host destinatario. Sull'host di destinazione, il protocollo TCP svuota quindi all'interno di un buffer il contenuto del pacchetto ricevuto, dopodiché il processo ricevente leggerà da questo buffer quando vorrà. Questa situazione è rappresentata in Figura 5.3, che, per semplicità, mostra i dati che fluiscono in una sola direzione; ricordate che, in generale, una singola connessione TCP consente al flusso di byte di fluire in entrambe le direzioni. — full duplex

I pacchetti scambiati fra pari entità del protocollo TCP in Figura 5.3 sono detti segmenti, perché ognuno di essi trasporta un segmento del flusso di byte. Ciascun segmento TCP contiene l'intestazione schematicamente raffigurata in Figura 5.4. L'importanza della maggior parte di questi campi diverrà chiara in seguito, in questa sezione: per ora, li presentiamo solamente.

I campi SrcPort e DstPort identificano, rispettivamente, le porte della sorgente e della destinazione, proprio come nel protocollo UDP. Questi due campi si combinano con gli indirizzi di sorgente e di destinazione per identificare univocamente ciascuna connessione TCP, cioè la chiave di demultiplexing di TCP è data dalla quaterna

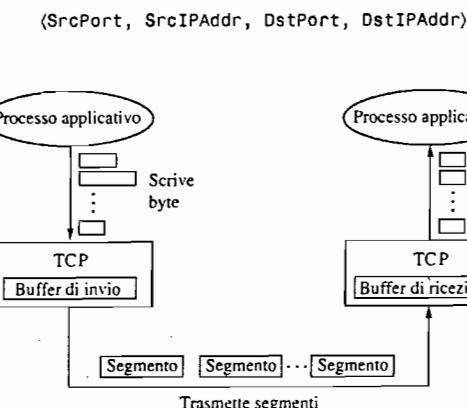


Figura 5.3 Come viene gestito un flusso di byte dal protocollo TCP.

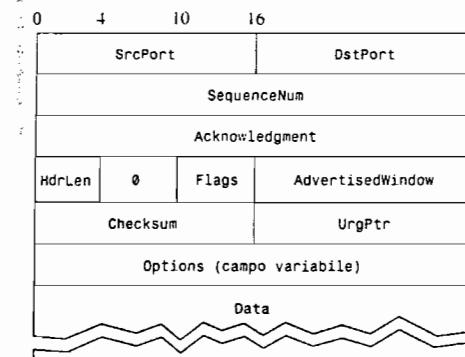


Figura 5.4 Il formato dell'intestazione TCP.

Dal momento che le connessioni TCP nascono e muoiono, è possibile che fra due porte una connessione venga instaurata, venga usata per inviare e ricevere dati e poi venga chiusa, per poi creare una seconda connessione fra le medesime porte. In questa situazione si parla, a volte, di due diverse incarnazioni della stessa connessione.

I campi Acknowledgement, SequenceNum e AdvertisedWindow sono coinvolti nella gestione dell'algoritmo sliding window di TCP. Data che TCP è un protocollo orientato ai byte, ogni byte di dati ha un numero di sequenza: il campo SequenceNum contiene il numero di sequenza del primo byte di dati trasportato in quel segmento. I campi Acknowledgement e AdvertisedWindow contengono informazioni in merito al flusso dei dati che scorre nell'altra direzione. Per semplificare la nostra discussione, ignoriamo il fatto che i dati possano fluire in entrambe le direzioni e ci concentreremo sui dati che vanno in una direzione con un certo valore di SequenceNum e sui valori Acknowledgement e AdvertisedWindow che vanno nella direzione opposta, come illustrato in Figura 5.5. L'uso di questi tre campi verrà descritto più compiutamente nella Sezione 5.2.4.

Il campo di 6 bit, Flags, viene usato per trasportare informazioni di controllo fra pari entità di una connessione TCP. I segnali possibili sono SYN, FIN, RESET, PUSH, URG e ACK: I segnali SYN e FIN sono usati quando si instaura e si termina, rispettivamente, una connessione TCP: il loro utilizzo è descritto nella Sezione 5.2.3. Il segnale ACK assume il valore 1 ogni volta che il campo Acknowledgement è valido, e, conseguentemente, il ricevitore vi deve porre attenzione. Il segnale URG al valore 1 significa che il segmento contiene dati urgenti: in questo caso il campo UrgPtr indica dove iniziano i dati non urgenti contenuti nel segmento.

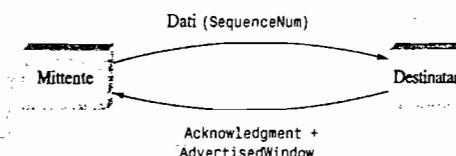


Figura 5.5 Diagramma semplificato (perché mostra una sola direzione del flusso) del processo TCP, con i dati che fluiscono in una direzione e le conferme (ACK) nell'altra.

mentre i dati urgenti sono contenuti all'inizio del corpo del segmento, fino al valore di UrgPt, byte all'interno del segmento stesso. Il segnale PUSH indica che il mittente ha invocato l'operazione "push", chiedendo all'entità TCP ricevente di segnalare questo fatto al processo ricevente. Queste due ultime funzionalità verranno discusse ulteriormente nella Sezione 5.2.7. Infine, il segnale **RESET** al valore 1 significa che il ricevente è stato confuso (ad esempio, perché ha ricevuto un segmento che non si aspettava di ricevere), per cui vuole terminare bruscamente la connessione.

Per concludere, il campo **Checksum** viene usato esattamente come descritto per UDP: viene calcolato sull'intestazione TCP, i dati TCP e la pseudointestazione, che è composta dall'indirizzo del mittente, l'indirizzo del destinatario e i campi che esprimono la lunghezza dell'intestazione IP. La somma di controllo è obbligatoria, per il protocollo TCP, sia in IPv4 sia in IPv6. Inoltre, essendo l'intestazione TCP di lunghezza variabile (a causa delle opzioni che possono essere indicate dopo i campi obbligatori), è presente un campo HdrLen che indica la lunghezza dell'intestazione in parole di 32 bit. Tale campo è anche noto con il nome di Offset, perché misura l'offset dall'inizio del pacchetto all'inizio dei dati.

5.2.3 Instaurazione e terminazione della connessione

Una connessione TCP inizia quando un client (il *chiamante*) effettua l'apertura attiva di un canale di comunicazione verso un server (il *chiamato*). Nell'ipotesi che il server abbia effettuato in precedenza un'apertura passiva, le due parti iniziano uno scambio di messaggi per stabilire la connessione (ricordate dal Capitolo 1 che un'entità che voglia iniziare una connessione esegue un'apertura attiva, mentre un'entità che desidera accettare connessioni esegue un'apertura passiva). Le due parti iniziano ad inviare dati soltanto dopo che questa fase di instaurazione della connessione è terminata. Analogamente, quando una delle due parti ha terminato di inviare dati, chiude la connessione nella direzione corrispondente, spingendo il protocollo TCP ad iniziare uno scambio di messaggi per la chiusura della connessione. Notate che, mentre l'apertura della connessione è un'attività asimmetrica (un'entità esegue un'apertura passiva e l'altra esegue un'apertura attiva), la chiusura della connessione è simmetrica (ciascuna entità deve chiudere la connessione indipendentemente)¹. Di conseguenza, è possibile che un'entità abbia effettuato la chiusura, per cui non può più inviare dati, mentre l'altra entità continua a tenere aperta l'altra metà della connessione bidirezionale e continua ad inviare dati.

Accordo in tre fasi (*three-way handshake*)

L'algoritmo usato dal protocollo TCP per instaurare e terminare una connessione viene chiamato *three-way handshake* ("stretta di mano" in tre fasi, nel senso di "raggiungimento di un accordo" in tre fasi). Dapprima descriviamo l'algoritmo di base, poi vediamo come viene usato nel protocollo TCP. L'handshake in tre fasi richiede lo scambio di tre messaggi fra il client e il server, come mostrato nel diagramma temporale di Figura 5.6.

Lo schema di base prevede che le due parti vogliano accordarsi su un insieme di parametri, che, nel caso dell'apertura di una connessione TCP, sono i numeri iniziali di sequenza che,

¹ Volendo essere più precisi, anche l'instaurazione della connessione può essere simmetrica, con entrambe le parti che tentano di aprire la connessione nello stesso momento, ma il caso più comune prevede che un'entità esegua un'apertura attiva e l'altra esegua un'apertura passiva.

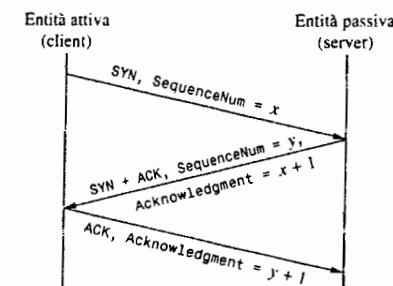


Figura 5.6 Diagramma temporale per l'algoritmo di accordo in tre fasi (*three-way handshake*).

le due parti useranno per i rispettivi flussi di byte; in generale, i parametri potrebbero essere qualsiasi cosa che ciascuna entità voglia far sapere all'altra entità. Prima di tutto, il client (l'entità attiva) invia un segmento al server (l'entità passiva) annunciando il numero iniziale di sequenza che pensa di usare (`Flags = SYN, SequenceNum = x`). Il server, quindi, risponde con un singolo segmento che svolge due funzioni: conferma il numero di sequenza del client (`Flags = ACK, Ack = x + 1`) e annuncia il proprio numero iniziale di sequenza (`Flags = SYN, SequenceNum = y`); per cui nel campo `Flags` di questo secondo messaggio sono impostati al valore 1 sia il bit `SYN` sia il bit `ACK`. Infine, il client risponde con un terzo segmento che conferma il numero di sequenza del server (`Flags = ACK, Ack = y + 1`). Ciascuna entità conferma il numero di sequenza ricevuto inviandolo aumentato di uno, perché il campo `Acknowledgement` identifica in ogni momento il "successivo numero di sequenza che ci si aspetta di ricevere", confermando così, implicitamente, di aver ricevuto tutti i numeri di sequenza precedenti. Sebbene non sia mostrato in questo diagramma temporale, per ognuno dei primi due segmenti viene impostato un tempo di scadenza, per ritrasmettere il segmento se non viene ricevuta in tempo la risposta che ci si attende.

Può darsi che vi stiate chiedendo perché il client e il server debbano scambiarsi i numeri iniziali di sequenza al momento dell'instaurazione della connessione: sarebbe più semplice se ciascuna entità iniziasse, semplicemente, con un "ben noto" numero di sequenza, ad esempio 0. In realtà, invece, la specifica del protocollo TCP richiede che ciascuna entità coinvolta in una connessione selezioni a caso un numero iniziale di sequenza, per proteggere il protocollo dal fatto che due incarnazioni della stessa connessione riutilizzino troppo presto gli stessi numeri di sequenza, mentre c'è ancora la possibilità che un segmento di una precedente incarnazione della connessione possa interferire con l'incarnazione successiva.

Diagramma delle transizioni di stato

Il protocollo TCP è sufficiente complesso da contenere, nella propria specifica, un diagramma delle transizioni di stato, presentato nella Figura 5.7. Questo diagramma mostra soltanto gli stati coinvolti nell'apertura di una connessione (tutto quanto si trova al di sopra di ESTABLISHED) e nella chiusura della connessione (tutto quanto al di sotto di ESTABLISHED). Tutto ciò che accade mentre la connessione è aperta (cioè il funzionamento dell'algoritmo sliding window) è nascosto all'interno dello stato ESTABLISHED.

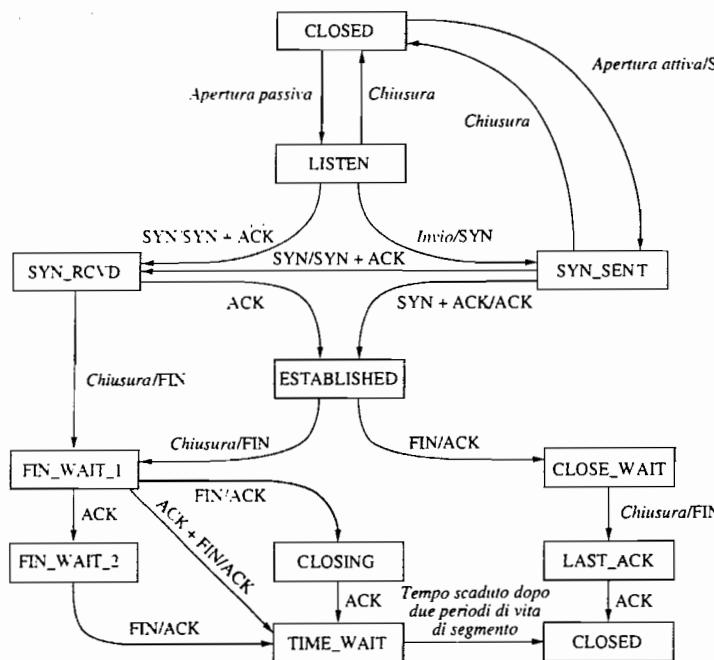


Figura 5.7 Diagramma delle transizioni di stato per il protocollo TCP.

Il diagramma delle transizioni di stato di TCP è di facile comprensione. Ciascun riquadro indica uno stato in cui si può trovare una delle due entità di una connessione TCP. Tutte le connessioni iniziano dallo stato CLOSED e, al progredire dell'instaurazione della connessione, si spostano da uno stato all'altro seguendo gli archi orientati, ognuno dei quali è etichettato con un coppia *evento/azione*. Ad esempio, se una connessione si trova nello stato LISTEN e riceve un segmento SYN (cioè un segmento con il bit SYN impostato al valore 1), allora la connessione stessa compie una transizione verso lo stato SYN_RCVD e intraprende l'azione di inviare un segmento di risposta contenente ACK + SYN.

Noteate che una transizione di stato può essere provocata da due tipi di eventi: (1) arriva un segmento dall'entità di pari livello (ad esempio, l'evento che si trova sull'arco che va da LISTEN a SYN_RCVD), oppure (2) il processo applicativo locale richiede un'azione al protocollo TCP (ad esempio, l'evento *apertura attiva* sull'arco che va da CLOSED a SYN_SENT). In altre parole, il diagramma delle transizioni di stato di TCP definisce, in sostanza, sia la *semantica* dell'interfaccia tra pari entità sia quella dell'interfaccia del servizio, secondo le definizioni viste nella Sezione 1.3.1. La *sintassi* di queste due interfacce è data, rispettivamente, dal formato del segmento (visto nella Figura 5.4) e da alcune interfacce per i programmi applicativi (API, *application programming interface*).

Seguiamo ora, sul diagramma di Figura 5.7, le transizioni che tipicamente avvengono, ricordando che il protocollo TCP compie transizioni di stato diverse a ciascun estremo della

connessione. Quando apre una connessione, il server invoca, prima di tutto, un'operazione di apertura passiva sul protocollo TCP, che di conseguenza si sposta verso lo stato LISTEN. Dopo qualche tempo, il client esegue un'apertura attiva, che provoca l'invio al server, da parte della propria entità, di un segmento SYN e lo spostamento verso lo stato SYN_RCVD e risponde con un segmento SYN + ACK. L'arrivo di questo segmento induce il client ad effettuare una transizione verso lo stato ESTABLISHED, con il conseguente invio di un ACK al server. Quando questo ACK viene ricevuto, il server entra finalmente nello stato ESTABLISHED. In altre parole, abbiamo appena seguito l'handshake in tre fasi.

Ci sono tre cose da notare a proposito della porzione di diagramma che si riferisce all'instaurazione della connessione (la metà superiore). Prima di tutto, se l'ACK inviato dal client al server viene perduto, cioè viene a mancare la terza fase dell'handshake, la connessione funziona comunque correttamente, perché il client si trova già nello stato ESTABLISHED e, di conseguenza, il processo applicativo locale può iniziare ad inviare dati all'altro estremo della connessione. Ciascuno di questi segmenti di dati avrà il bit ACK impostato al valore 1 ed il valore corretto nel campo Acknowledgement, per cui il server si sposterà nello stato ESTABLISHED non appena riceve il primo segmento di dati. Questa caratteristica è un punto molto importante del protocollo TCP: ogni segmento indica quale numero di sequenza si aspetta di ricevere il mittente del segmento stesso, anche se questo valore ripete il m^o desimo numero di sequenza contenuto in uno o più dei segmenti precedentemente inviati.

La seconda cosa da notare in relazione del diagramma di transizione degli stati è che esiste una strana transizione in uscita dallo stato LISTEN ogniqualvolta il processo locale invoca un'operazione di invio sul protocollo TCP: è quindi possibile, per un'entità passiva, impersonare entrambe le parti di una connessione (cioè essere anche la parte remota che vuole connettersi a sé stessa) e, di conseguenza, smettere di attendere passivamente una connessione e, invece, stabilire attivamente la connessione. Per quanto ne sappiamo, si tratta di una caratteristica di TCP che non viene utilizzata da alcun processo applicativo.

L'ultima cosa da notare nel diagramma sono gli archi che non sono stati tracciati: in particolare, la maggior parte degli stati che prevedono l'invio di un segmento all'altra entità fanno anche partire un temporizzatore che, quando eventualmente scade, provoca la ritrasmissione del segmento, qualora non sia stata ricevuta la risposta attesa. Queste ritrasmissioni non sono indicate nel diagramma delle transizioni di stato. Se dopo alcuni tentativi la risposta attesa non arriva, il protocollo TCP abbandona la trasmissione e ritorna nello stato CLOSED.

Rivolgendo ora la nostra attenzione al processo che pone termine ad una connessione, la cosa importante da ricordare è che il processo applicativo di entrambe le entità della connessione deve chiudere indipendentemente la propria metà della connessione stessa. Se una sola parte chiude la connessione, ciò significa che non ha più dati da inviare, ma è ancora disponibile a ricevere dati dall'altra entità. Ciò complica il diagramma delle transizioni di stato, perché deve tener conto della possibilità che le due parti invochino l'operazione di chiusura contemporaneamente, così come della possibilità che sia la prima entità ad invocare la chiusura, seguita, dopo un certo tempo, dall'invocazione di chiusura della seconda entità. Di conseguenza, su ciascun estremo della connessione esistono tre combinazioni di transizioni che portano una connessione dallo stato ESTABLISHED allo stato CLOSED:

- Questa entità chiude per prima:
ESTABLISHED → FIN_WAIT_1 → FIN_WAIT_2 → TIME_WAIT → CLOSED

- L'altra entità chiude per prima:
ESTABLISHED → CLOSE_WAIT → LAST_ACK → CLOSED
- Entrambe le parti chiudono contemporaneamente:
ESTABLISHED → FIN_WAIT_1 → CLOSING → TIME_WAIT → CLOSED

Esiste, in realtà, una quarta, anorché rara, sequenza di transizioni che porta allo stato CLOSED, seguendo l'arco che va da FIN_WAIT_1 a TIME_WAIT. Vi lasciamo come esercizio l'individuazione di quale combinazione di circostanze porti a questa quarta possibilità.

Una cosa importante da sapere in merito alla chiusura della connessione è che una connessione che si trova nello stato TIME_WAIT non può andare nello stato CLOSED prima di aver atteso un tempo uguale al doppio del massimo tempo di vita di un datagramma IP in Internet (ad esempio, 120 secondi), perché, dopo aver inviato un ACK in risposta al segmento FIN che era stato inviato dall'altra entità, l'entità locale non sa se il proprio ACK sia stato consegnato con successo. Di conseguenza, l'altra entità potrebbe ritrasmettere il proprio segmento FIN, che potrebbe a sua volta essere ritardato dalla rete. Se fosse consentito alla connessione di entrare direttamente nello stato CLOSED, un'altra coppia di processi applicativi potrebbe intervenire ed aprire la medesima connessione (cioè usare la stessa coppia di porte) e il segmento FIN della precedente incarnazione, in ritardo, provocherebbe immediatamente l'inizio del processo di chiusura della nuova connessione.

5.2.4 Una rivisitazione dell'algoritmo *sliding window*

Siamo ora pronti per discutere la variante dell'algoritmo sliding window usata dal protocollo TCP, che svolge diverse funzioni: (1) garantisce la consegna affidabile dei dati, (2) garantisce che i dati vengano consegnati nella sequenza corretta, (3) consente il controllo di flusso tra il mittente e il destinatario. Per quanto riguarda le prime due funzioni, l'uso che TCP fa dell'algoritmo sliding window è uguale a quanto visto nella Sezione 2.5.2, mentre ne differisce per l'introduzione della funzione di controllo di flusso. In particolare, invece di avere una finestra scorrevole (*sliding window*) di dimensione fissa, il ricevitore *comunica* al mittente una certa dimensione per la finestra, usando il campo AdvertisedWindow nell'intestazione TCP. Il mittente, di conseguenza, si deve limitare fino al punto di avere, in ogni momento, un valore massimo di byte non confermati uguale al valore di AdvertisedWindow. Il ricevitore sceglie il valore adatto per il campo AdvertisedWindow in base alla quantità di memoria che è stata assegnata per la memorizzazione dei dati relativi alla connessione, con l'obiettivo di impedire al mittente di esaurire la capacità di memorizzazione del ricevitore. Ne parleremo con grande dettaglio in seguito.

Consegna affidabile e in sequenza

Per vedere come il lato del mittente e del ricevente di TCP interagiscono l'un l'altro nella realizzazione di una consegna affidabile e in sequenza, considerate la situazione illustrata in Figura 5.8. Sul lato del mittente, TCP gestisce un buffer di spedizione, che viene usato per memorizzare i dati inviati ma non ancora confermati, oltre ai dati consegnati dall'applicazione mittente ma non ancora trasmessi. Sul lato ricevente, TCP gestisce un buffer di ricezione, che contiene i dati che arrivano fuori ordine, oltre ai dati che sono in ordine corretto (cioè che non hanno byte mancanti nella parte precedente del flusso) ma che non sono ancora stati letti dal processo applicativo.

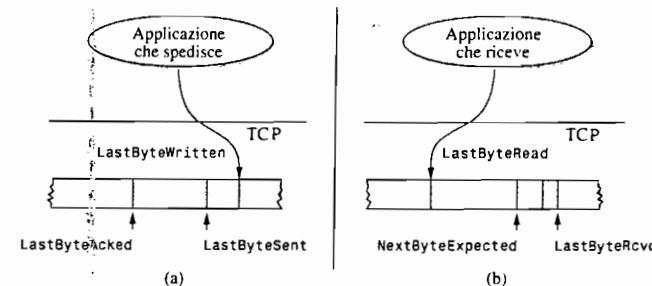


Figura 5.8 Relazione esistente tra il buffer di spedizione (a) e il buffer di ricezione (b) di TCP.

Per rendere più semplice la discussione, ignoriamo inizialmente il fatto che sia i buffer sia i numeri di sequenza abbiano una dimensione finita e, quindi, prima o poi tornino al loro valore iniziale. Ancora, non distinguiamo fra un puntatore che segnala dove sia memorizzato un particolare byte di dati all'interno di un buffer e il numero di sequenza che identifica tale byte.

Esaminando prima il lato di chi invia, vengono gestiti tre puntatori per il buffer di invio, ciascuno con un significato ovvio: LastByteAcked (ultimo byte confermato), LastByteSent (ultimo byte inviato) e LastByteWritten (ultimo byte scritto). Chiaramente

$$\text{LastByteAcked} \leq \text{LastByteSent}$$

perché il ricevitore non può aver confermato un byte che non è ancora stato inviato, e

$$\text{LastByteSent} \leq \text{LastByteWritten}$$

perché il protocollo TCP non può inviare un byte che non è ancora stato scritto nel buffer dal processo applicativo. Notate anche che nessuno dei byte che si trovano a sinistra di LastByteAcked ha bisogno di essere memorizzato nel buffer, perché sono stati tutti già confermati, così come nessuno dei byte a destra di LastByteWritten, dato che tali dati non sono ancora stati generati.

Dal lato ricevente viene gestito un insieme di puntatori (numeri di sequenza) analogo: LastByteRead (ultimo byte letto), NextByteExpected (prossimo byte atteso) e LastByteRcvd (ultimo byte ricevuto). Tuttavia, le relative diseguaglianze sono un po' meno intuitive, per via del problema della consegna fuori ordine. La prima relazione

$$\text{LastByteRead} < \text{NextByteExpected}$$

è vera perché un byte non può essere letto dall'applicazione finché non è stato ricevuto e sono stati ricevuti anche tutti i byte precedenti: NextByteExpected punta al byte immediatamente successivo all'ultimo che soddisfa questo requisito. Inoltre

$$\text{NextByteExpected} \leq \text{LastByteRcvd} + 1$$

perché: se i dati sono arrivati in ordine, `NextByteExpected` punta al byte successivo a `LastByteRcvd`; altrimenti, se i dati sono arrivati fuori ordine, `NextByteExpected` punta all'inizio della prima interruzione nei dati, come in Figura 5.8. Notate che i byte che si trovano a sinistra di `LastByteRead` non necessitano di essere memorizzati nel buffer, perché sono già stati letti dal processo applicativo locale, così come i byte a destra di `LastByteRcvd`, che non sono ancora arrivati.

Controllo di flusso

La maggior parte della discussione precedente è simile a quella vista nella Sezione 2.5.2; l'unica differenza sostanziale sta nel fatto che questa volta ci siamo occupati di come i processi applicativi mittente e ricevente riempiono e, rispettivamente, svuotano i propri buffer locali (la discussione appena condotta ha glissato sul fatto che i dati in arrivo da un nodo a monte del flusso stavano riempiendo il buffer di invio, mentre i dati trasmessi verso un nodo a valle del flusso stavano svuotando il buffer di ricezione).

Prima di procedere dovreste avere la certezza di aver compreso bene questo punto, perché ora i due algoritmi si differenziano in modo significativo. Nel seguito torniamo a considerare buffer di dimensioni finite, rispettivamente `MaxSendBuffer` e `MaxRcvBuffer`, anche se non ci occupiamo dei dettagli realizzativi dei buffer stessi. In altre parole, ci interessa solamente il numero di byte che vengono memorizzati, non dove vengono realmente memorizzati.

Ricordate che in un protocollo di tipo sliding window la dimensione della finestra determina la quantità di dati che possono essere inviati senza aspettare la conferma del ricevente. Di conseguenza, il ricevente controlla la velocità del mittente comunicando una finestra che non sia più grande della quantità di dati che può memorizzare. Osservate che, dal lato ricevente, il protocollo TCP deve fare in modo che sia sempre

$$\text{LastByteRcvd} - \text{LastByteRead} \leq \text{MaxRcvBuffer}$$

per evitare che il buffer trabocchi. Per far ciò comunica una dimensione di finestra uguale a

$$\text{AdvertisedWindow} = \text{MaxRcvBuffer} - ((\text{NextByteExpected} - 1) - \text{LastByteRead})$$

che rappresenta la quantità di spazio rimasto libero nel buffer di ricezione. Mano a mano che i dati arrivano, il ricevitore li conferma, insieme a tutti i byte già arrivati in precedenza, e sposta verso destra (cioè incrementa) `LastByteRcvd`, provocando, potenzialmente, una contrazione della finestra disponibile. Il fatto che la finestra di contraggia effettivamente, oppure no, dipende da quanto velocemente il processo applicativo locale utilizza i dati. Se il processo locale legge i dati alla stessa velocità con cui arrivano (facendo aumentare `LastByteRead` con la stessa velocità con cui aumenta `LastByteRcvd`), allora la finestra che viene pubblicizzata rimane aperta (cioè `AdvertisedWindow = MaxRcvBuffer`). Se, invece, il processo ricevente si attarda, magari perché esegue operazioni computazionalmente molto onerose per ciascun byte che legge, allora la finestra pubblicizzata diventa più piccola all'arrivo di ogni segmento, fino a raggiungere il valore 0.

Dal lato del mittente, il protocollo TCP deve utilizzare la finestra che viene comunicata dal ricevente, per cui, ad ogni istante, deve garantire che

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{AdvertisedWindow}$$

5.2 Flusso affidabile di byte (TCP)

Detto in altro modo, il mittente calcola la finestra *effettiva*, che limita la quantità di dati che possono essere inviati:

$$\text{EffectiveWindow} = \text{AdvertisedWindow} - (\text{LastByteSent} - \text{LastByteAcked})$$

Chiaramente, il mittente può inviare dati soltanto se `EffectiveWindow` è maggiore di zero. Quindi, è possibile che arrivi un segmento che conferma x byte, consentendo al mittente di aumentare `LastByteAcked` di una quantità pari a x , ma, poiché il processo ricevente non sta leggendo dati, la finestra che viene comunicata è ora x byte più piccola di prima. In tale situazione, il mittente potrebbe liberare un po' di spazio nel proprio buffer, ma non potrebbe inviare ulteriori dati.

Mentre accade tutto ciò, il lato trasmittente deve anche assicurarsi che il processo applicativo locale non faccia traboccare il buffer di trasmissione, cioè che

$$\text{LastByteWritten} - \text{LastByteAcked} \leq \text{MaxSendBuffer}$$

Se il processo trasmittente tenta di inviare y byte al protocollo TCP, ma

$$(\text{LastByteWritten} - \text{LastByteAcked}) + y > \text{MaxSendBuffer}$$

allora TCP blocca il processo trasmittente e non gli permette di generare altri dati.

A questo punto si può capire come sia possibile che un processo ricevente lento provochi il blocco totale di un processo trasmittente veloce. Dapprima, il buffer di ricezione si riempie, facendo restringere la finestra fino a zero. Una finestra completamente chiusa impedisce al mittente di inviare alcun dato, anche se i dati che ha trasmesso in precedenza sono stati confermati con successo. Infine, non essendo in grado di trasmettere alcun dato, il buffer di trasmissione si riempie, costringendo il protocollo TCP a bloccare il processo trasmittente. Non appena il processo ricevente ricomincia a leggere dati, la parte ricevente di TCP è in grado di aprire nuovamente la propria finestra, consentendo all'entità trasmittente di inviare dati, prelevandoli dal proprio buffer. Quando questi dati vengono, poi, confermati, il valore di `LastByteAcked` aumenta, viene liberato nel buffer lo spazio che era occupato da questi dati confermati e si sblocca il processo trasmittente, consentendogli di procedere.

Ci rimane da risolvere soltanto un ulteriore dettaglio: come fa l'entità trasmittente a sapere che la finestra disponibile non è più di dimensione zero? Come detto, il protocollo TCP invia *sempre* un segmento in risposta alla ricezione di un segmento di dati: questa risposta contiene i valori più recenti dei campi `Acknowledge` e `AdvertisedWindow`, anche se tali valori non sono stati modificati dall'ultima volta in cui sono stati inviati. Ora, il problema è: una volta che il ricevente ha comunicato una finestra di dimensione 0, il mittente non può più inviare alcun dato, per cui non avrà modo di scoprire, in futuro, che la finestra disponibile non è più 0, perché la parte ricevente di TCP non invia spontaneamente segmenti privi di dati, ma invia segnali solamente in risposta a segmenti di dati in arrivo.

Il protocollo TCP gestisce questa situazione nel modo seguente. Ogniqualvolta l'altra entità pubblicizza una finestra di dimensione 0, la parte mittente continua ad inviare periodicamente un segmento con 1 byte di dati. Sa che questi dati probabilmente non verranno accettati, ma ci prova lo stesso, perché ognuno di questi segmenti di 1 byte richiede una risposta che contiene la finestra attualmente pubblicizzata. Prima o poi, uno di questi byte darà luogo ad una risposta che porta la finestra ad una dimensione diversa da zero.

Noteate che il motivo per cui la parte trasmittente invia periodicamente questo segmento "di stimolo" sta nel fatto che il protocollo TCP è stato progettato per rendere assai semplice la parte ricevente: risponde soltanto ai segmenti inviati dal mittente e non prende mai proprie iniziative. Questo è un esempio di una regola di progettazione di protocolli ben assodata (anche se non sempre applicata) che, in mancanza di un nome migliore, chiamiamo **regola del mittente intelligente e del ricevitore stupido**. Ricordate che abbiamo visto un altro esempio di questa regola quando abbiamo discusso l'uso dei NAK nella Sezione 2.5.2.

Protezione dal ritorno al valore iniziale (*wraparound*)

Questa sottosezione e la successiva parlano della dimensione dei campi SequenceNum e AdvertisedWindow e dell'effetto che queste loro dimensioni hanno sulla correttezza e sulle prestazioni del protocollo TCP. Il campo SequenceNum di TCP è lungo 32 bit e il campo AdvertisedWindow è lungo 16 bit, per cui è agevolmente verificata la condizione imposta dall'algoritmo sliding window per la quale lo spazio dei numeri di sequenza deve essere almeno il doppio della dimensione della finestra: $2^{32} \gg 2 \times 2^{16}$. Tuttavia, questo requisito non è ciò che ci interessa in merito a questi due campi, che ora prenderemo in esame singolarmente.

L'importanza di avere uno spazio a 32 bit per i numeri di sequenza consiste nel fatto che i numeri di sequenza usati per una connessione potrebbero tornare al valore iniziale (*wraparound*): in un certo momento potrebbe essere inviato un byte avente il numero di sequenza x e poi, più tardi, potrebbe essere inviato un secondo byte con lo stesso numero di sequenza, x . Ipotizziamo nuovamente che i pacchetti non possano sopravvivere nella rete Internet per un periodo più lungo del valore consigliato di MSL. Di conseguenza, dobbiamo avere la garanzia che il numero di sequenza non torni al proprio valore iniziale in un periodo di 120 secondi. Il fatto che ciò accada oppure no dipende dalla velocità a cui si possono trasmettere i dati in Internet, cioè da quanto velocemente possa essere utilizzato lo spazio a 32 bit dei numeri di sequenza (alla base di questa discussione c'è l'ipotesi che stiamo cercando di utilizzare lo spazio dei numeri di sequenza il più velocemente possibile, ma lo staremo facendo di certo se ci stiamo impegnando per mantenere pieno il canale). La Tabella 5.1 mostra quanto tempo sia necessario a far tornare il numero di sequenza al proprio valore iniziale in reti con diverse ampiezze di banda.

SEQ# → 32 bit AdvW → 16 bit

Tabella 5.1 Tempo necessario perché i numeri di sequenza a 32 bit tornino al proprio valore iniziale.

Ampiezza di banda	Tempo per tornare al valore iniziale
T1 (1.5 Mbps)	6.4 ore
Ethernet (10 Mbps)	57 minuti
T3 (45 Mbps)	13 minuti
FDDI (100 Mbps)	6 minuti
STS-3 (155 Mbps)	4 minuti
STS-12 (622 Mbps)	55 secondi
STS-24 (1.2 Gbps)	28 secondi

Come potete vedere, lo spazio dei numeri di sequenza a 32 bit è adeguato per le reti odierne, ma, dal momento che attualmente nel backbone di Internet esistono linee di collegamento OC-48, non ci vorrà molto tempo perché singole connessioni TCP vogliano viaggiare a velocità di 622 Mbps o ancora maggiori. Fortunatamente, IETF ha già individuato un'estensione del protocollo TCP che estende, a tutti gli effetti, lo spazio dei numeri di sequenza, per fornire protezione contro il fenomeno del ritorno al valore iniziale: questa estensione ed altre ad essa correlate sono descritte nella Sezione 5.2.8.

Mantenere pieno il canale di trasmissione

L'importanza del campo AdvertisedWindow a 16 bit consiste nel fatto che esso deve essere sufficientemente grande da consentire al mittente di tenere pieno il canale di trasmissione. Sappiamo che, ovviamente, il ricevitore è libero di non aprire la finestra fino al valore massimo del campo AdvertisedWindow, ma qui siamo interessati alla situazione in cui il ricevitore ha spazio di memorizzazione a sufficienza per gestire tanti dati quanti ne sono consentiti da AdvertisedWindow.

In questo caso, non è soltanto l'ampiezza di banda della rete, ma anche il prodotto ritardo × ampiezza di banda che specifica quanto deve essere grande il campo AdvertisedWindow: la finestra deve poter essere aperta a sufficienza per consentire la trasmissione di una quantità di dati uguale al prodotto ritardo × ampiezza di banda. Ipotizzando un valore di RTT uguale a 100 ms (un valore tipico per una connessione che attraversi gli Stati Uniti), la Tabella 5.2 fornisce il prodotto ritardo × ampiezza di banda di diverse tecnologie di rete.

Come potete vedere, il campo AdvertisedWindow del protocollo TCP si trova in una situazione ancora peggiore del campo SequenceNum: non è grande a sufficienza nemmeno per gestire una connessione T3 che attraversi gli Stati Uniti, perché un campo a 16 bit consente di usare una finestra di dimensione massima uguale a 64 KB. La medesima estensione TCP appena citata (descritta nella Sezione 5.2.8) fornisce un meccanismo per aumentare, a tutti gli effetti, la dimensione della finestra che può essere comunicata.

5.2.5 Stimolare la trasmissione

Consideriamo ora un problema sorprendentemente subdolo: come faccia il protocollo TCP a decidere di trasmettere un segmento. Come abbiamo visto, TCP fornisce supporto all'astrazione di flusso di byte: i programmi applicativi scrivono byte nel flusso, e sta al protocollo

Tabella 5.2 Dimensione di finestra necessaria per un RTT di 100 ms.

Ampiezza di banda	Prodotto ritardo × ampiezza di banda
T1 (1.5 Mbps)	18 KB
Ethernet (10 Mbps)	122 KB
T3 (45 Mbps)	549 KB
FDDI (100 Mbps)	1.2 MB
STS-3 (155 Mbps)	1.8 MB
STS-12 (622 Mbps)	7.4 MB
STS-24 (1.2 Gbps)	14.8 MB

TCP decide di aver byte a sufficienza per inviare un segmento. Quali fattori controllano questa decisione?

Se ignoriamo che esista il controllo di flusso, cioè ipotizziamo che la finestra sia abbastanza aperta, come accade quando viene instaurata una connessione, allora TCP prevede tre meccanismi per stimolare la trasmissione di un segmento. Innanzitutto, TCP gestisce una variabile, tipicamente denominata dimensione massima del segmento (MSS, maximum segment size) ed invia un segmento non appena ha raccolto dal processo mittente un numero di byte uguale a MSS: che solitamente viene impostato al valore del più grande segmento TCP che può essere inviato senza provocare la frammentazione locale del corrispondente pacchetto IP (cioè MSS è uguale al valore di MTU della rete direttamente connessa, diminuito della dimensione delle intestazioni TCP e IP). La seconda cosa che provoca la trasmissione di un segmento è l'esplicita richiesta di invio da parte del processo mittente, tramite l'operazione push: il processo mittente invoca questa operazione per provocare il reale svuotamento del buffer contenente tutti i byte non ancora inviati. L'ultima causa che provoca la trasmissione di un segmento è lo scadere di un temporizzatore: il segmento che viene così prodotto contiene tanti byte quanti ne sono presenti, al momento, nel buffer di trasmissione, anche se, come vedrete presto, questo "temporizzatore" non si comporta proprio come ci si potrebbe immaginare.

Sindrome della finestra futile (*silly window syndrome*)

È però ovvio che non possiamo ignorare il controllo di flusso, che gioca un ruolo importante nel controllo della velocità di trasmissione del mittente. Se il mittente ha MSS byte pronti da spedire e la finestra è aperta almeno di tale quantità, allora il mittente trasmette un segmento completo. Supponete, invece, che il mittente stia accumulando byte da spedire, ma la finestra, sia chiusa. Supponete ora che arrivi una conferma che apre la finestra di una quantità tale da consentire la trasmissione, ad esempio, di MSS/2 byte. Il mittente deve trasmettere un segmento incompleto, oppure aspettare che la finestra si apra fino a consentire la trasmissione di un segmento di dimensione uguale a MSS? La specifica originaria taceva su questo punto e le prime implementazioni del protocollo TCP decisero di andare avanti e di trasmettere un segmento incompleto: dopo tutto, non c'è modo di sapere quanto bisognerà aspettare perché la finestra si apra di più.

Si è visto che questa strategia, di avvantaggiarsi in modo attivo di qualsiasi finestra disponibile, porta ad una situazione che prende il nome di *sindrome della finestra futile* (silly window syndrome). La Figura 5.9 aiuta a visualizzare cosa accade. Se pensate ad un flusso TCP come ad un convoglio che trasporta contenitori " pieni" (segmenti di dati) in una direzione e contenitori vuoti (conferme) nell'altra direzione, i segmenti di dimensione uguale a MSS corrispondono a contenitori grandi, mentre i segmenti contenenti un solo byte corrispondono a contenitori molto piccoli. Se il mittente riempie un contenitore vuoto appena questo arriva, allora tutti i piccoli contenitori introdotti nel sistema vi rimangono indefinitamente, perché vengono immediatamente riempiti e svuotati a ciascun estremo e non si fondono mai con contenitori adiacenti per creare contenitori più grandi. Questo scenario fu scoperto nel momento in cui le prime implementazioni del protocollo TCP si trovavano regolarmente a riempire la rete di piccoli segmenti.

Note che la sindrome della finestra futile è un problema soltanto quando il mittente trasmette un segmento di piccole dimensioni oppure il ricevitore apre la finestra di una quantità modesta. Se nessuna di queste due cose accade, allora nel flusso non vengono mai intro-

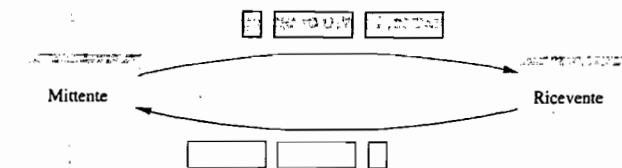


Figura 5.9 La sindrome della finestra futile.

dotti contenitori piccoli. È impossibile impedire l'invio di segmenti piccoli: ad esempio, l'applicazione potrebbe invocare l'operazione push dopo aver scritto un solo byte. È però possibile impedire al ricevitore di introdurre piccoli contenitori nel flusso (cioè di effettuare piccole aperture della finestra): la regola prevede che, mentre sta comunicando una finestra di dimensione zero, il ricevitore debba attendere di avere uno spazio libero di dimensione MSS prima di comunicare un'apertura della finestra.

Dato che non possiamo eliminare la possibilità che vengano introdotti nel flusso contenitori piccoli, abbiamo bisogno di un meccanismo che li fonda. Il ricevitore può farlo ritardando i segnali di conferma, inviando una conferma complessiva piuttosto che più conferme di piccole dimensioni, ma questa è soltanto una soluzione parziale, perché il ricevitore non ha modo di sapere per quanto tempo sia ragionevole attendere per l'arrivo del segmento successivo oppure perché l'applicazione legga un po' di dati (aprendo così la finestra). La soluzione definitiva spetta al mittente, cosa che ci riporta al nostro problema iniziale: quand'è che il protocollo TCP decide di trasmettere un segmento?

Algoritmo di Nagle

Tornando al mittente di TCP, se ci sono dati da inviare ma la finestra è aperta per una dimensione inferiore al valore di MSS possiamo voler attendere un po' di tempo prima di inviare i dati, ma la domanda è: quanto a lungo? Se aspettiamo troppo, danneggiamo le applicazioni interattive, come Telnet. Se non aspettiamo abbastanza, allora rischiamo di spedire un sacco di piccoli pacchetti e ricadiamo nella sindrome della finestra futile. La risposta consiste nell'utilizzo di un temporizzatore e nel trasmettere quando questo scade.

Anche se potremmo usare un temporizzatore basato sull'orologio di sistema (*clock*), ad esempio un temporizzatore che scatti ogni 100 ms, Nagle presentò un'elegante soluzione *auto-temporizzata*. L'idea sfrutta il fatto che, mentre il protocollo TCP ha dati in viaggio, il mittente può ricevere una conferma, che può venire considerata come un temporizzatore che scade, provocando la trasmissione di ulteriori dati. L'algoritmo di Nagle fornisce una regola semplice e uniforme per decidere quando sia il caso di trasmettere:

```

Quando l'applicazione produce dati da inviare
se sia i dati disponibili sia la finestra sono ≥ MSS
    invia un segmento completo
altrimenti
    se ci sono dati non confermati in viaggio
        memorizza i dati nel buffer finché non arriva una conferma
    altrimenti
        invia ora tutti i dati
  
```

In altre parole, se la finestra lo consente, l'invio di un segmento completo è sempre una scelta valida, così come è permesso inviare immediatamente una piccola quantità di dati se non ci sono segmenti in viaggio, ma se ci sono dati in viaggio il mittente deve attendere una conferma prima di trasmettere il segmento successivo. Di conseguenza, un'applicazione interattiva come Telnet, che scrive continuamente un byte alla volta, invierà i dati alla velocità di un segmento per ogni intervallo di tempo uguale a RTT. Alcuni segmenti conterranno un unico byte, mentre altri conterranno tanti byte quanti l'utente è riuscito a scriverne in un intervallo di tempo uguale al tempo di round trip. Dato che alcune applicazioni non sono in grado di accettare tale ritardo per ciascuna azione di scrittura che eseguono verso una connessione TCP, l'interfaccia socket consente alle applicazioni di disattivare l'algoritmo di Nagle, imponendo l'opzione `TCP_NODELAY`. Impostando tale opzione i dati vengono trasmessi appena possibile.

5.2.6 Ritrasmissione adattativa

Dato che il protocollo TCP garantisce la consegna affidabile dei dati, ogni segmento per il quale non sia stata ricevuta conferma entro un certo periodo di tempo viene ritrasmesso. Questo temporizzatore viene impostato da TCP in funzione del valore di RTT che ci si attende fra i due estremi della connessione. Sfortunatamente, dato l'ampio intervallo di possibili valori di RTT fra una coppia qualsiasi di host in Internet, oltre alla variabilità di RTT nel tempo anche fra i due medesimi host, scegliere un valore appropriato per il temporizzatore non è affatto facile. Per risolvere questo problema, il protocollo TCP usa un meccanismo di ritrasmissione adattativo, che ora descriviamo insieme al modo in cui si è evoluto nel tempo, accumulandosi l'esperienza della comunità di Internet in merito all'utilizzo del protocollo TCP.

Algoritmo originario

Iniziamo con un semplice algoritmo che calcola un valore di temporizzazione fra una coppia di host: si tratta dell'algoritmo che fu originariamente descritto nelle specifiche di TCP. La descrizione seguente lo presenta in quei termini, ma potrebbe essere utilizzato da qualsiasi protocollo di trasporto.

L'idea è quella di valutare continuamente il valore medio di RTT e di calcolare il valore della temporizzazione in funzione di tale RTT. In particolare, ogni volta che TCP invia un segmento di dati, memorizza l'orario attuale e, quando poi riceve la conferma dell'arrivo di tale segmento, legge nuovamente l'orario e considera la differenza tra questi due orari come nuovo campione dei valori di RTT, `SampleRTT`. Il protocollo calcola poi un valore stimato di RTT, `EstimatedRTT`, come media pesata fra la stima precedente e questo nuovo campione, in questo modo:

$$\text{EstimatedRTT} = a \times \text{EstimatedRTT} + (1 - a) \times \text{SampleRTT}$$

Il parametro `a` viene scelto per rallentare le modifiche al valore `EstimatedRTT`. Un piccolo valore di `a` segue le modifiche di RTT ma è forse troppo influenzato da fluttuazioni temporanee, mentre un valore elevato di `a` è più stabile ma forse non sufficientemente veloce da adattarsi alle modifiche della realtà. La specifica originaria di TCP raccomandava di impostare `a` ad un valore compreso tra 0.8 e 0.9. Il protocollo TCP usa poi il valore di `EstimatedRTT` per calcolare il valore della temporizzazione (`TimeOut`) in un modo piuttosto conservativo:

$$\text{TimeOut} = 2 \times \text{EstimatedRTT}$$

5.2 Flusso affidabile di byte (TCP)

Algoritmo di Karn/Partridge

Dopo parecchi anni di utilizzo in Internet, in questo semplice algoritmo venne scoperto un errore abbastanza ovvio. Il problema sta nel fatto che una conferma tramite ACK non è in realtà la conferma di una particolare trasmissione, ma conferma semplicemente la ricezione di dati. In altre parole, ogni volta che un segmento viene ritrasmesso e, successivamente, arriva al mittente la relativa conferma, è impossibile determinare se tale conferma debba essere associata alla prima o alla seconda trasmissione del segmento ai fini della misurazione di un nuovo campione di RTT: per calcolare un valore accurato per `SampleRTT` è necessario sapere a quale trasmissione va associata ciascuna conferma. Come evidenziato in Figura 5.10, se si ipotizza che la conferma sia riferita alla trasmissione originale, mentre in realtà è riferita alla seconda trasmissione, il valore di `SampleRTT` è troppo grande (a), mentre se si ipotizza che la conferma sia riferita alla seconda trasmissione ma in realtà è relativa alla prima, si ottiene un valore di `SampleRTT` troppo piccolo (b).

La soluzione è sorprendentemente semplice. Ogni volta che TCP ritrasmette un segmento, smette di registrare campioni di RTT: misura `SampleRTT` soltanto per quei segmenti che sono stati inviati una volta sola. Questa soluzione è nota come algoritmo di Karn/Partridge, dal nome dei suoi inventori. La soluzione che hanno proposto prevede anche una seconda piccola modifica al meccanismo delle temporizzazioni in TCP: ogni volta che TCP effettua una ritrasmissione, l'intervallo di temporizzazione viene posto uguale al doppio del valore usato in precedenza, piuttosto che basarsi sul più recente valore di `EstimatedRTT`. In sostanza, Karn e Partridge hanno proposto che TCP usi un backoff esponenziale, in modo simile a quanto fatto da Ethernet. La motivazione per l'uso del backoff esponenziale è semplice: la perdita di segmenti è quasi certamente dovuta a congestione, per cui la sorgente TCP non dovrebbe reagire troppo aggressivamente alla scadenza di una temporizzazione. Al contrario, più spesso accade che la connessione veda scadere le proprie temporizzazioni, più lenta dovrebbe divenire la sorgente nel ritrasmettere. Vedremo di nuovo questa idea nel Capitolo 6, utilizzata in un meccanismo molto più sofisticato.

Algoritmo di Jacobson/Karels

L'algoritmo di Karn/Partridge fu introdotto in un periodo in cui Internet soffriva di elevati livelli di congestione di rete. Il loro approccio venne progettato per porre rimedio ad alcune delle cause di tale congestione e, sebbene ciò abbia apportato un miglioramento, la conge-

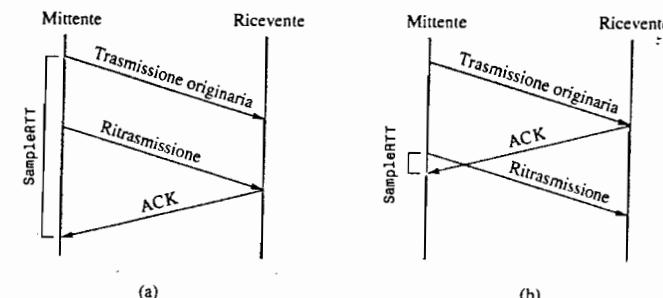


Figura 5.10 Associazione di una conferma (ACK) con la trasmissione originaria (a) o con una ritrasmissione (b).

stione non venne eliminata. Un paio d'anni più tardi, altri due ricercatori, Jacobson e Karels, proposero una più drastica modifica al protocollo TCP, per combattere la congestione. Il cuore di quella proposta è descritto nel Capitolo 6, mentre qui rivolgiamo la nostra attenzione agli aspetti di quella proposta che sono correlati alla decisione di quando sia ora di ritrasmettere un segmento.

Come nota a margine, dovrebbe essere chiaro come il meccanismo della scadenza di temporizzazioni sia correlato alla congestione: se si usano temporizzazioni troppo brevi, può darsi che un segmento venga ritrasmesso inutilmente, aggiungendo ulteriore carico alla rete. Come vedrete nel Capitolo 6, un altro motivo per cui è necessario avere un valore di temporizzazione accurato è che la scadenza di una temporizzazione viene interpretata come un segnale di congestione, innescando l'azione di un meccanismo di controllo della congestione stessa. Infine, notate che nel calcolo delle temporizzazioni con il metodo di Jacobson/Karels non vi è niente che sia specifico del protocollo TCP: potrebbe essere utilizzato per qualsiasi protocollo di trasporto.

Il problema principale del metodo di calcolo originario consisteva nella mancata considerazione della varianza dei campioni di RTT. Intuitivamente, se i campioni variano poco l'uno rispetto all'altro, allora il valore di EstimatedRTT può essere considerato più affidabile e non c'è ragione per moltiplicare questa stima per 2 quando si calcola il valore della temporizzazione. D'altra parte, campioni con grande varianza suggeriscono che il valore della temporizzazione non dovrebbe seguire rigidamente il valore di EstimatedRTT.

Nel nuovo approccio il mittente misura un nuovo valore di SampleRTT allo stesso modo, per poi inserire tale nuovo campione nel calcolo della temporizzazione nel modo seguente:

$$\begin{aligned} \text{Difference} &= \text{SampleRTT} - \text{EstimatedRTT} \\ \text{EstimatedRTT} &= \text{EstimatedRTT} + (\delta \times \text{Difference}) \\ \text{Deviation} &= \text{Deviation} + \delta (|\text{Difference}| - \text{Deviation}) \end{aligned}$$

dove δ è una frazione compresa tra 0 e 1. In sostanza, calcoliamo sia il valor medio di RTT sia la variazione di tale valore medio.

Il protocollo TCP calcola, poi, il valore della temporizzazione come una funzione di EstimatedRTT e di Deviation, con la formula seguente:

$$\text{TimeOut} = \mu \times \text{EstimatedRTT} + \phi \times \text{Deviation}$$

dove, in base all'esperienza, μ viene tipicamente impostato al valore 1 e ϕ al valore 4. Di conseguenza, quando la varianza è piccola, TimeOut è prossimo a EstimatedRTT, mentre una varianza elevata porta il termine Deviation a dominare il calcolo.

Implementazione

In merito all'implementazione delle temporizzazioni nel protocollo TCP ci sono due aspetti degni di nota. Il primo è che è possibile implementare il calcolo di EstimatedRTT e di Deviation senza usare l'aritmetica in virgola mobile: l'intero calcolo viene effettuato dopo aver scalato tutti i valori di un fattore 2^n , scegliendo δ uguale a $1/2^n$. In questo modo possiamo usare l'aritmetica intera, effettuando moltiplicazioni e divisioni mediante scorimenti, ottenendo così prestazioni migliori. La restante parte del calcolo viene svolta dal seguente frammento

di codice, dove $n = 3$ (cioè $\delta = 1/8$). Notate che EstimatedRTT e Deviation sono memorizzati nella loro forma scalata, mentre il valore di SampleRTT all'inizio del codice è il valore di TimeOut sono numeri veri, non scalati. Se il codice vi sembra difficile da seguire, potreste provare ad inserirvi alcuni numeri veri e verificare che fornisce gli stessi risultati delle espressioni precedenti.

```
{
    SampleRTT -= (EstimatedRTT >> 3);
    EstimatedRTT += SampleRTT;
    if (SampleRTT < 0)
        SampleRTT = -SampleRTT;
    SampleRTT -= (Deviation >> 3);
    Deviation += SampleRTT;
    TimeOut = (EstimatedRTT >> 3) + (Deviation >> 1);
}
```

Il secondo punto degno di nota è che l'algoritmo di Jacobson/Karels è efficiente quanto l'orologio di sistema che viene usato per valutare i tempi: in una tipica implementazione di Unix, la granularità di tale segnale di sistema è di 500 ms, un valore significativamente maggiore del tempo di round trip medio di una linea continentale, che è generalmente compreso tra 100 e 200 ms. Per peggiorare ancora le cose, l'implementazione di TCP presente in Unix verifica se è scaduta una temporizzazione soltanto ogni volta che l'orologio di sistema cambia valore, cioè ogni 500 ms, e prende un solo campione del tempo di round trip per ogni RTT. La combinazione di questi due fattori comporta che si abbia, abbastanza spesso, la scadenza di una temporizzazione un secondo dopo la trasmissione del segmento. Di nuovo, le estensioni di TCP contengono un meccanismo che rende un po' più preciso questo calcolo di RTT.

5.2.7 Confini tra gruppi di dati (record)

Dato che TCP è un protocollo a flusso di byte, il numero di byte che vengono scritti dal mittente non è necessariamente uguale al numero di byte che vengono letti dal ricevente. Ad esempio, può darsi che un'applicazione scriva, in una connessione TCP, prima 8 byte, poi 2 byte, poi 20 byte, mentre sul lato del ricevente l'applicazione può leggere 5 byte alla volta all'interno di un ciclo che viene eseguito 6 volte. Il protocollo TCP non inserisce confini tra gruppi di dati (*record*), cioè tra l'ottavo e il nono byte, né tra il decimo e l'undicesimo, al contrario di quanto avviene in un protocollo orientato ai messaggi, come UDP, in cui il messaggio che viene inviato ha esattamente la stessa lunghezza del messaggio che viene ricevuto.

Nonostante TCP sia un protocollo a flusso di byte, prevede due diverse proprietà che possono essere utilizzate dal mittente per inserire, all'interno del flusso di byte, confini fra record, segnalando così al ricevitore come scomporre il flusso di byte in record (ad esempio, in molte applicazioni di basi di dati è utile poter contrassegnare i confini fra record successivi). Entrambe queste caratteristiche furono originariamente incluse nel protocollo TCP per motivi completamente diversi, ma sono state utilizzate per questo scopo soltanto nel tempo. Il primo meccanismo è la possibilità di gestire dati urgenti, mediante il bit URG e il campo UrgPtr nell'intestazione TCP. In origine, il meccanismo dei dati urgenti era stato progettato

per consentire all'applicazione mittente di inviare alla pari entità dati *fuori linea* (*out-of-band data*), cioè dati separati dal normale flusso dei dati (ad esempio, un comando che interrompa un'operazione già intrapresa). Questi dati fuori linea venivano identificati, nel segmento, mediante il campo *UrgPtr*, che richiedeva la consegna di tali dati al processo ricevente non appena questi arrivassero a destinazione, anche se ciò volesse dire consegnarli prima di altri dati aventi un numero di sequenza precedente. Nel tempo, però, questa caratteristica non è stata usata, per cui, invece di rappresentare dati "urgenti", è stata utilizzata con il significato di dati "speciali", come quelli che contrassegnano i record. Questo uso si è sviluppato perché, come per l'operazione *push*, sul lato ricevente il protocollo TCP deve informare l'applicazione del fatto che sono arrivati "dati urgenti", che non sono importanti in sé: ciò che è importante è che il processo mittente abbia inviato un "segnale" al ricevente.

Il secondo meccanismo per inserire marcatori di terminazione di record in corrispondenza di un particolare byte è l'operazione *push*. In origine questo meccanismo fu pensato per consentire al processo mittente di ordinare al protocollo TCP di inviare alla propria pari entità tutti i byte memorizzati fino a quel momento (*flush*). L'operazione *push* può essere usata per contrassegnare i confini tra record perché la specifica dice che il protocollo TCP, quando riceve un comando *push*, deve inviare tutti i dati presenti nel buffer di trasmissione e, facoltativamente, la parte ricevente del protocollo TCP notifica all'applicazione il fatto di aver ricevuto un segmento con il bit *PUSH* al valore 1. Se il lato ricevente supporta questa opzione (ma, ad esempio, l'interfaccia socket non lo fa), allora si può usare l'operazione *push* per scomporre in record il flusso TCP.

Naturalmente il programma applicativo ha sempre la possibilità di inserire confini tra record senza l'aiuto del protocollo TCP. Ad esempio, può inviare un campo che indichi la lunghezza del record seguente, oppure può inserire nel flusso di dati i propri marcatori di confine tra record.

5.2.8 Estensioni al protocollo TCP

In tre diversi punti di questa sezione abbiamo menzionato il fatto che esistono oggi estensioni al protocollo TCP che aiutano ad alleviare alcuni dei problemi che TCP sta affrontando con l'aumentare della velocità delle reti sottostanti. Queste estensioni sono progettate per avere il minimo impatto possibile sul protocollo TCP: in particolare, sono realizzate come opzioni da aggiungere all'intestazione TCP (abbiamo sorvolato, in precedenza, su questo punto, ma il motivo per cui l'intestazione TCP ha un campo *HdrLen* sta nel fatto che l'intestazione può essere di lunghezza variabile, per via della parte che contiene le opzioni che sono state aggiunte al protocollo). L'importanza di aver aggiunto queste estensioni come opzioni anziché modificare il nucleo dell'intestazione TCP sta nel fatto che, in questo modo, gli host possono ancora comunicare usando il protocollo TCP senza dover necessariamente implementare le opzioni. Tuttavia, gli host che implementano le estensioni opzionali ne possono trarre vantaggio. Le due entità all'estremità di una connessione si accordano, durante la fase di instaurazione della connessione TCP, sulle opzioni da usare.

La prima estensione aiuta a migliorare il meccanismo di scadenza delle temporizzazioni previsto da TCP. Invece di misurare il valore di RTT usando un evento a grossa granularità, il protocollo TCP può consultare l'orologio di sistema mentre sta per inviare un segmento ed inserire nell'intestazione del segmento tale *orario* (*timestamp*), sotto forma di numero a 32 bit. Il ricevente, poi, restituisce questo valore orario all'interno della corrispondente conferma e il mittente lo sottrae dall'orario in cui viene ricevuta la conferma, per misurare di con-

5.2 Flusso affidabile di byte (TCP)

seguenza il valore di RTT. In sostanza, l'opzione dell'orario fornisce al protocollo TCP un luogo comodo per "memorizzare" l'istante di trasmissione del segmento: nel segmento stesso. Notate che le entità terminali della connessione non necessitano di orologi sincronizzati, poiché l'indicazione del tempo viene scritta e letta allo stesso capo della connessione.

La seconda estensione si occupa del problema del campo *SequenceNum* a 32 bit, che torna al proprio valore iniziale troppo rapidamente in una rete ad alta velocità. Invece di definire un nuovo campo di 64 bit per il numero di sequenza, TCP usa l'orario a 32 bit appena descritto per estendere, a tutti gli effetti, lo spazio dei numeri di sequenza. In altre parole, il protocollo TCP decide se accettare o rifiutare un segmento basandosi su un identificativo a 64 bit, avente il campo *SequenceNum* come 32 bit di valore inferiore e l'indicazione oraria come 32 bit di valore superiore. Poiché l'indicazione oraria è sempre crescente, serve a distinguere fra due diverse incarnazioni dello stesso numero di sequenza. Notate che l'indicazione oraria viene qui usata soltanto per proteggersi contro il ritorno al valore iniziale, e non viene considerata parte integrante del numero di sequenza per quanto riguarda l'ordinamento o la conferma dei dati.

La terza estensione consente al protocollo TCP di comunicare una finestra più grande, consentendo così lo sfruttamento dei più alti valori di ritardo × ampiezza di banda che sono resi disponibili da reti ad alta velocità. Questa estensione prevede un'opzione che definisce un fattore di scala per la finestra che viene comunicata, cioè, invece di interpretare il numero che appare nel campo *AdvertisedWindow* come un'indicazione del numero di byte che possono essere inviati dal mittente senza averne conferma, questa opzione consente alle due parti coinvolte di accordarsi sul fatto che il campo *AdvertisedWindow* rappresenti un conteggio di spezzoni di maggiori dimensioni (ad esempio, unità di 16 byte). In altre parole, l'opzione di finestra scalata specifica di quanti bit il campo *AdvertisedWindow* deve essere fatto scorrere verso sinistra prima di usare il suo contenuto per calcolare l'effettiva dimensione della finestra.

5.2.9 Scelte di progetto alternative

Sebbene il protocollo TCP abbia dimostrato di essere un protocollo robusto che soddisfa le necessità di un ampio ventaglio di applicazioni, lo spazio di progetto per protocolli di trasporto è molto grande: non c'è nessun motivo per cui il protocollo TCP debba essere l'unico punto valido all'interno di tale spazio di progetto. Terminiamo la nostra discussione di TCP considerando scelte di progetto alternative. Mentre daremo una spiegazione del motivo per cui i progettisti di TCP hanno fatto determinate scelte, lasceremo a voi decidere se ci possa essere spazio per protocolli di trasporto alternativi.

Prima di tutto, fin dal primo capitolo di questo libro abbiamo sottolineato come ci siano almeno due categorie interessanti di protocolli di trasporto: protocolli orientati al flusso, come TCP, e protocolli di tipo richiesta/risposta, come RPC. In altre parole, abbiamo implicitamente suddiviso lo spazio di progettazione a metà, e abbiamo posizionato il protocollo TCP nella porzione di spazio destinata ai protocolli orientati al flusso. I protocolli orientati al flusso possono essere ulteriormente suddivisi in due gruppi, affidabili e non affidabili, il primo dei quali contiene il protocollo TCP, mentre il secondo è più adatto ad applicazioni video interattive che preferiscono perdere un riquadro d'immagine (*frame*) piuttosto che incorrere nel ritardo dovuto ad una ritrasmissione.

Questo esercizio di costruzione di una tassonomia dei protocolli di trasporto è interessante e potrebbe continuare con dettaglio ulteriore, ma il mondo non è bianco e nero come

potrebbe sembrare. Considerate, ad esempio, l'utilizzabilità di TCP come protocollo di trasporto per applicazioni di tipo richiesta/risposta. TCP è un protocollo full-duplex, per cui sarebbe semplice aprire una connessione TCP fra il client e il server, inviare il messaggio di richiesta in una direzione e, poi, inviare il messaggio di risposta nell'altra direzione. Purtroppo, ci sono due complicazioni. La prima è che TCP è un protocollo orientato *ai byte*, piuttosto che *ai messaggi*, e le applicazioni di tipo richiesta/risposta gestiscono sempre messaggi (torniamo tra poco su questa diversità tra byte e messaggi). La seconda difficoltà sta nel fatto che, in quelle situazioni in cui sia il messaggio di richiesta sia il messaggio di risposta trovano posto in un unico pacchetto di rete, un protocollo di tipo richiesta/risposta che sia ben progettato richiede solamente due pacchetti per realizzare lo scambio di informazioni, mentre il protocollo TCP ne richiederebbe almeno nove: tre per instaurare la connessione, due per lo scambio di messaggi e quattro per chiudere la connessione. Ovviamente, se i messaggi di richiesta o di risposta sono abbastanza grandi da richiedere più pacchetti di rete (ad esempio, potrebbero essere necessari 100 pacchetti per inviare un messaggio di risposta di 100000 byte), allora quanto richiesto in aggiunta per instaurare e terminare la connessione è ininfluente. In altre parole, non è sempre vero che un certo protocollo non è in grado di fornire il supporto per una determinata funzionalità, ma spesso è vero che un particolare schema di progetto è più efficiente di un altro, in particolari circostanze.

Secondo: come appena detto, vi potreste chiedere perché per il protocollo TCP si sia scelto di fornire un servizio affidabile a flusso *di byte* piuttosto che un servizio affidabile a flusso *di messaggi*: i messaggi sarebbero la scelta naturale per un'applicazione di basi di dati che voglia scambiare record. Questa domanda ha due risposte. La prima è che un protocollo orientato ai messaggi deve, per definizione, stabilire un limite superiore alla dimensione dei messaggi. Dopo tutto, un messaggio infinitamente lungo è un flusso di byte. Per qualunque dimensione del messaggio che venga scelta dal protocollo, esisteranno applicazioni che vorranno inviare messaggi più lunghi, rendendo inutilizzabile il protocollo di trasporto e forzando l'applicazione ad implementare i propri servizi di trasporto. Il secondo motivo è che, anche se i protocolli orientati ai messaggi sono decisamente più adeguati per applicazioni che vogliono scambiarsi record di dati, è sempre possibile inserire confini fra record all'interno di un flusso di byte per realizzare questa funzionalità, come descritto nella Sezione 5.2.7.

Terzo: per il protocollo TCP sono state definite fasi esplicite per l'instaurazione e la terminazione della connessione, ma ciò non è richiesto. Nel caso dell'instaurazione della connessione, sarebbe certamente possibile inviare tutti i parametri necessari alla connessione insieme al primo messaggio contenente dati, ma il protocollo TCP ha scelto un approccio più conservativo che da al ricevente l'opportunità di rifiutare la connessione prima che arrivino i dati. Nel caso della terminazione, potremmo silenziosamente porre termine ad una connessione che sia stata inattiva per un lungo periodo di tempo, ma ciò renderebbe complicata la vita di applicazioni, come Telnet, che vogliono tenere in piedi una connessione per settimane: tali applicazioni sarebbero costrette ad inviare messaggi fuori linea, di tipo "keepalive", al solo scopo di impedire che all'altro capo venga chiusa la connessione.

Infine, TCP è un protocollo basato su finestra, ma questa non è l'unica possibilità. L'alternativa è un progetto *basato sulla velocità*, in cui il ricevente indica al mittente la velocità, espresso in byte al secondo o in pacchetti al secondo, alla quale è in grado di accettare dati in ingresso: ad esempio, il ricevente potrebbe informare il mittente di poter accogliere 100 pacchetti al secondo. Esistente un'interessante dualità tra finestre e velocità, poiché il numero di pacchetti (o di byte) di una finestra, diviso per il valore di RTT, fornisce esattamente la velocità. Ad esempio, una finestra con una dimensione di 10 pacchetti e un RTT di 100 ms

implicano che il mittente possa trasmettere ad una velocità di 100 pacchetti al secondo. Aumentando o diminuendo la dimensione della finestra pubblicizzata, il ricevitore aumenta o diminuisce, in effetti, la velocità a cui il mittente può trasmettere. Nel protocollo TCP questa informazione viene restituita al mittente nel campo *AdvertisedWindow* della conferma di ogni segmento. Uno dei problemi fondamentali di un protocollo basato sulla velocità è quanto spesso la velocità desiderata, che può cambiare nel tempo, venga comunicata al mittente: ad ogni pacchetto, una volta per ogni RTT, oppure soltanto quando viene modificata? Anche se abbiamo confrontato finestre e velocità soltanto nel contesto del controllo di flusso, l'argomento è ancora più caldo nel contesto del controllo di congestione, che verrà discusso nel Capitolo 6.

5.3 Remote Procedure Call

Come detto nel Capitolo 1, un frequente schema di comunicazione usato da programmi applicativi è il paradigma richiesta/risposta, detto anche transazione di messaggio: un client invia ad un server un messaggio di richiesta, il server risponde con un messaggio di risposta, e il client si blocca (cioè sospende la propria esecuzione) in attesa di tale risposta. La Figura 5.11 mostra l'interazione di base fra il client e il server in tale transazione di messaggio.

Un protocollo di trasporto che fornisca supporto al paradigma richiesta/risposta è molto più sofisticato di un messaggio UDP che va in una direzione, seguito da un messaggio UDP che va nella direzione opposta: richiede il superamento di tutte le limitazioni imposte dalla rete sottostante che sono state delineate nell'enunciazione del problema all'inizio di questo capitolo. Mentre il protocollo TCP supera queste limitazioni fornendo un servizio a flusso di byte affidabile, non soddisfa molto bene il paradigma di richiesta/risposta, perché affrontare il problema di stabilire una connessione TCP soltanto per scambiare un paio di messaggi sembra un'esagerazione. Questa sezione descrive un terzo protocollo di trasporto, che chiamiamo Remote Procedure Call (RPC, *invocazione di procedura remota*) e che soddisfa meglio i bisogni di un'applicazione coinvolta nello scambio di messaggi di tipo richiesta/risposta.

RPC è, in realtà, qualcosa di più di un semplice protocollo: è un meccanismo molto diffuso per strutturare sistemi distribuiti. RPC è assai popolare perché si basa sulla semantica di un'invocazione di procedura locale: il programma applicativo invoca una procedura, senza preoccuparsi del fatto che sia locale o remota, e si blocca finché la procedura non termina. Anche se, detto così, può sembrare semplice, ci sono due problemi principali che rendono

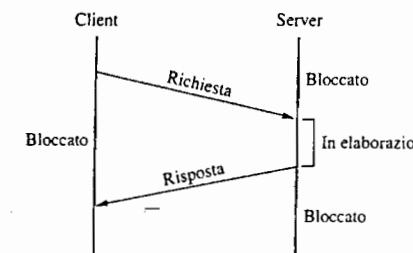


Figura 5.11 Diagramma temporale per RPC.

l'invocazione di una procedura remota più complicata dell'invocazione di una procedura locale:

- La rete fra il processo invocante e il processo invocato possiede proprietà molto più complesse della scheda principale di un calcolatore. Ad esempio, è probabile che limiti la dimensione dei messaggi e ha la tendenza a perdere e riordinare i messaggi.
- I calcolatori su cui sono in esecuzione i processi invocante e invocato possono avere architetture significativamente diverse e diversi formati di rappresentazione dei dati.

Di conseguenza, un meccanismo di RPC completo richiede due componenti principali:

1. Un protocollo che gestisce i messaggi scambiati fra i processi client e server e che si occupa delle proprietà, potenzialmente negative, della rete sottostante.
2. Il supporto di un linguaggio di programmazione e di un compilatore per trasformare i parametri in un messaggio di richiesta sulla macchina client, trasformare nuovamente questo messaggio in parametri sulla macchina server, e fare la stessa cosa con il valore restituito dall'invocazione (questa parte del meccanismo RPC viene solitamente chiamata *compilatore di adattatore, stub compiler*).

La Figura 5.12 rappresenta schematicamente ciò che accade quando un client invoca una procedura remota. Dapprima il client invoca un adattatore locale della procedura, fornendo i parametri richiesti dalla procedura stessa. Questo adattatore nasconde il fatto che la procedura sia remota, traducendo gli argomenti in un messaggio di richiesta e poi invocando un protocollo RPC per inviare il messaggio di richiesta alla macchina server. Sul server, il pro-

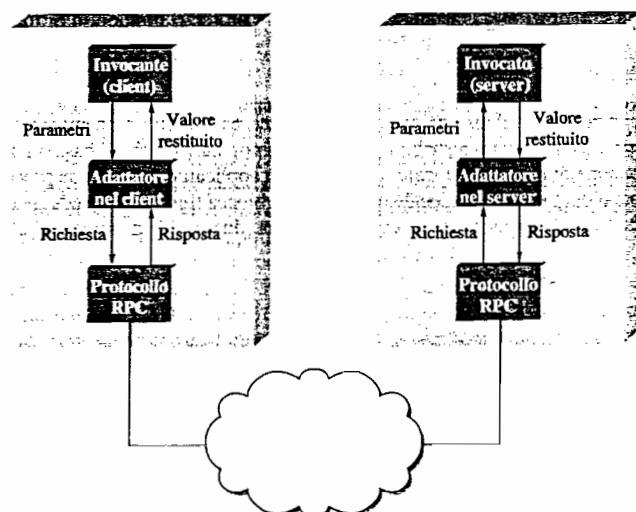


Figura 5.12 Funzionamento completo di RPC.

tocollo RPC consegna il messaggio di richiesta all'adattatore del server, che traduce il messaggio e fornisce i parametri alla procedura, per poi invocarla localmente. Dopo che la procedura termina la propria esecuzione sul server, restituisce la risposta all'adattatore del server, che inserisce tale valore restituito in un messaggio di risposta da consegnare al protocollo RPC per l'invio al client. Il protocollo RPC sul client trasferisce il messaggio all'adattatore del client, che lo traduce in un valore restituito da fornire al programma client.

Questa sezione considera il funzionamento di RPC soltanto in relazione agli aspetti del protocollo, cioè ignora gli adattatori e si concentra, invece, sul protocollo RPC che trasmette i messaggi fra il client e il server; la trasformazione dei parametri in messaggi, e viceversa, è trattata nel Capitolo 7. Inoltre, poiché RPC è un termine generico, piuttosto che uno standard specifico come TCP, seguiremo un approccio diverso da quello della sezione precedente. Invece di incentrare la discussione su uno standard esistente (come TCP) e poi segnalare, alla fine, alternative di progetto, vi guideremo nel processo cognitivo che porta alla progettazione di un protocollo RPC, cioè progetteremo da zero un nostro protocollo, considerando le opzioni di progetto disponibili ad ogni passaggio, per poi tornare indietro e descrivere alcuni dei protocolli RPC più diffusi, confrontandoli e mettendoli in contrapposizione con il protocollo da noi progettato.

Prima di iniziare, però, notiamo che un protocollo RPC esegue un insieme di funzioni piuttosto complicate, per cui, invece di trattare RPC come un singolo protocollo monolitico, lo svilupperemo come una "pila" di tre protocolli più piccoli: BLAST, CHAN e SELECT. Ognuno di questi protocolli più piccoli, che a volte chiamiamo *microprotocollo*, è composto da un unico algoritmo che risolve uno dei problemi delineati all'inizio del capitolo. In breve:

- BLAST: frammenta e ricostruisce grandi messaggi
- CHAN: sincronizza i messaggi di richiesta e risposta
- SELECT: smista i messaggi di richiesta al processo corretto

Questi microprotocolli sono protocolli completi e autonomi che si possono usare in diverse combinazioni per fornire diversi servizi di trasporto. La Sezione 5.3.4 mostra come possano essere combinati per realizzare un protocollo RPC.

Giusto per essere chiari, BLAST, CHAN e SELECT non sono protocolli standard come lo sono TCP, UDP e IP. Sono, semplicemente, protocolli di nostra invenzione, ma mettono in evidenza gli algoritmi necessari per implementare un protocollo RPC. Dato che questa sezione non è vincolata a ciò che è stato progettato in passato, fornisce un'opportunità particolarmente favorevole per prendere in esame i principi della progettazione di protocolli.

5.3.1 Trasferimento a blocchi (BLAST)

Il primo problema che vogliamo risolvere consiste nel trasformare una rete sottostante che consegna messaggi di piccole dimensioni (ad esempio, 1 KB) in un servizio che consegna messaggi di dimensione molto maggiore (ad esempio, 32 KB). Anche se 32 KB non sembra una dimensione "arbitrariamente grande", è abbastanza grande da poter essere usata, in pratica, da molte applicazioni, tra cui la maggior parte dei *file system* distribuiti. Alla fine, comunque, per consentire lo scambio di messaggi di dimensioni qualsiasi, sarà necessario un protocollo basato sul flusso, come TCP (si veda la Sezione 5.2), perché qualsiasi protocollo orientato ai messaggi avrà necessariamente un limite superiore per la dimensione dei messaggi che è in grado di gestire, e si può sempre immaginare di aver bisogno di trasmettere un messaggio che sia più grande di tale limite.

Abbiamo già preso in esame la tecnica di base che viene utilizzata per trasmettere un messaggio di grandi dimensioni in una rete che può ospitare soltanto messaggi più piccoli: la frammentazione e la ricostruzione. Descriviamo ora il protocollo BLAST, che usa proprio questa tecnica. Una delle proprietà peculiari di BLAST è il tentativo strenuo di consegnare tutti i frammenti di un messaggio. Diversamente dal meccanismo AAL per la segmentazione e la ricostruzione usato nella tecnologia ATM (si veda la Sezione 3.3) o il meccanismo di frammentazione e ricostruzione del protocollo IP (si veda la Sezione 4.1), BLAST tenta di recuperare le situazioni in cui vengono perduti frammenti, ritrasmettendoli. Tuttavia, BLAST non procede fino al punto da garantire la consegna del messaggio. L'importanza di questa scelta di progetto diverrà chiara in seguito.

Algoritmo BLAST

L'idea che sta alla base di BLAST è che il mittente spezzi in un insieme di frammenti più piccoli un messaggio di grandi dimensioni che gli sia stato trasmesso da un protocollo di livello superiore, per poi trasmettere tramite la rete questi frammenti, uno dopo l'altro, consecutivamente, da cui il nome "blast", esplosione: il protocollo non attende che un frammento venga confermato prima di inviare il successivo. Il ricevente, quindi, invia al mittente una richiesta di ritrasmissione selettiva (SRR, selective retransmission request), per indicare quali frammenti siano arrivati e quali non lo siano (il messaggio SRR viene a volte chiamato *conferma parziale o selettiva*). Infine, il mittente ritrasmette i frammenti mancanti. Nel caso in cui siano arrivati tutti i frammenti, il messaggio SRR serve a confermare interamente il messaggio. La Figura 5.13 presenta un diagramma temporale del protocollo BLAST.

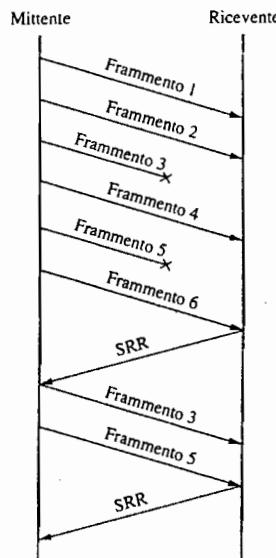


Figura 5.13 Diagramma temporale rappresentativo di BLAST.

In quale strato si colloca RPC?

Ancora una volta ci poniamo la domanda "A quale strato appartiene questo?". Per molte persone, in special modo per quelle che seguono da vicino l'architettura di Internet, il protocollo RPC è realizzato al di sopra di un protocollo di trasporto (solitamente UDP) e, quindi, non può (per definizione) essere anch'esso un protocollo di trasporto. Tuttavia, è ugualmente corretto argomentare che Internet dovrebbe avere un protocollo RPC, perché esso offre un servizio tra processi che differisce in maniera fondamentale da quello offerto da TCP e UDP. La risposta più comune a questo è che l'architettura di Internet non impedisce ai progettisti di reti di implementare il proprio protocollo RPC al di sopra di UDP (in generale, UDP viene visto come la "scappatoia" dell'architettura Internet, perché, a tutti gli effetti, aggiunge al protocollo IP solamente uno strato di demultiplexing). Indipendentemente da ciò che pensate in merito al fatto che Internet debba avere un protocollo RPC ufficiale, il punto importante è che il modo in cui implementate RPC all'interno dell'architettura Internet non dice nulla sul fatto che RPC debba essere considerato un protocollo di trasporto oppure no.

Assai interessante è notare come esistano altre persone che credono che RPC sia il protocollo più interessante del mondo e che l'architettura TCP/IP è soltanto ciò che usate quando volete "uscire dal sito": si tratta della visione predominante nella comunità dei sistemi operativi, che ha costruito innumerevoli nuclei di sistemi operativi per sistemi distribuiti che contengono un solo protocollo (ovviamente, il protocollo RPC), che viene eseguito al di sopra del software di controllo di un dispositivo di rete (*device driver*).

Le cose si complicano ulteriormente quando si implementa RPC come una combinazione di tre diversi microprotocolli, come nel caso di questa sezione. In tale situazione, quale dei tre è il protocollo "di trasporto"? La nostra risposta a questa domanda è che qualsiasi protocollo che offre un servizio tra processi, diversamente da un servizio tra nodi o tra host, si qualifica come un protocollo di trasporto, per cui RPC è un protocollo di trasporto e, in realtà, lo si può realizzare mediante una combinazione di microprotocolli che sono, essi stessi, validi esempi di protocolli di trasporto.

Consideriamo ora in maggiore dettaglio i lati trasmittente e ricevente di BLAST. Dal lato trasmittente, dopo aver frammentato il messaggio e aver trasmesso ciascun frammento, il mittente imposta un temporizzatore denominato DONE. Ogni volta che arriva un SRR, il mittente ritrasmette i frammenti richiesti e reimposta il temporizzatore DONE. Quando arriva il messaggio SRR che comunica la ricezione di tutti i frammenti, il mittente elimina la propria copia del messaggio e annulla il temporizzatore DONE. Se ad un certo punto DONE scade, il mittente elimina la propria copia del messaggio, cioè abbandona il tentativo di spedizione.

Sul lato ricevente, quando arriva il primo frammento di un messaggio, il ricevitore crea una struttura dati per contenere i singoli frammenti a mano a mano che arrivano, ed imposta un temporizzatore LAST_FRAG. Questo temporizzatore conta il tempo che è trascorso dal ricevimento del frammento più recente. Ogni volta che arriva un frammento di quel messaggio, il ricevitore lo aggiunge alla propria struttura dati; se sono presenti tutti i frammenti, li assembla per dare l'orma al messaggio completo e lo trasmette al protocollo di livello superiore.

Esistono, però, quattro casi eccezionali a cui il ricevitore deve prestare attenzione:

- Se arriva l'ultimo frammento (che è contrassegnato in modo particolare) ma il messaggio non è completo, allora il ricevitore determina quali siano i frammenti mancanti ed invia un SRR al mittente, oltre ad impostare un temporizzatore denominato RETRY.
- Se scade il temporizzatore LAST_FRAG, allora il ricevitore determina quali siano i frammenti mancanti ed invia un SRR al mittente, oltre ad impostare, anche in questo caso, il temporizzatore RETRY.
- Se scade, per la prima o per la seconda volta, il temporizzatore RETRY, allora il ricevitore determina quali siano i frammenti ancora mancanti e trasmette nuovamente un messaggio SRR.
- Se il temporizzatore RETRY scade per la terza volta, allora il ricevitore elimina i frammenti già arrivati ed annulla il temporizzatore LAST_FRAG, cioè abbandona il tentativo di ricezione.

Tre aspetti di BLAST sono degni di nota. Prima di tutto, ci sono due diversi eventi che provocano un primo invio di un messaggio SRR: l'arrivo dell'ultimo frammento e lo scadere del temporizzatore LAST_FRAG. Nel primo caso, poiché la rete può modificare l'ordine tra i pacchetti, l'arrivo dell'ultimo frammento non implica necessariamente che uno dei frammenti mancanti sia andato perduto (potrebbe essere semplicemente in ritardo), ma, essendo questa la motivazione più probabile, BLAST usa un approccio "aggressivo" ed invia un messaggio SRR. Nel secondo caso, possiamo dedurre che l'ultimo frammento è andato perduto o sta subendo un pesante ritardo.

Il secondo aspetto riguarda il fatto che le prestazioni di BLAST non dipendono in modo critico dai valori a cui vengono impostati i temporizzatori. Il temporizzatore DONE viene usato solamente per decidere se sia ora di abbandonare il tentativo di trasmissione ed eliminare il messaggio su cui si sta operando, e può quindi essere impostato ad un valore sufficiente elevato, dato che serve solamente a recuperare spazio in memoria. Il temporizzatore RETRY viene utilizzato solamente per ritrasmettere un messaggio SRR: in questo caso la situazione è talmente compromessa che il protocollo si trova ad eseguire nuovamente una procedura che tenta di ripristinare il processo, per cui le prestazioni sono l'ultima cosa che interessa. Infine, il temporizzatore LAST_FRAG può, effettivamente, avere qualche impatto sulle prestazioni, dato che a volte provoca l'invio, da parte del ricevitore, di un messaggio SRR, ma si tratta di un evento poco probabile: accade solamente quando l'ultimo frammento di un messaggio è andato perduto nella rete.

Il terzo aspetto degnò di nota è che, nonostante BLAST proceda di continuo a chiedere e a ritrasmettere i frammenti mancanti, non fornisce la garanzia che il messaggio completo venga, prima o poi, consegnato. Per capire come ciò accada, ipotizzate che un messaggio contenga solamente uno o due frammenti e che questi frammenti vadano perduti. Il ricevitore non invierà mai un SRR e, prima o poi, scadrà il temporizzatore DONE del mittente, provocando l'abbandono del tentativo di spedizione del messaggio. Per garantire la consegna, BLAST dovrebbe andare incontro alla scadenza di un temporizzatore nel caso in cui non ricevesse un SRR, ritrasmettendo, di conseguenza, l'ultimo insieme di frammenti che aveva trasmesso. Anche se certamente BLAST avrebbe potuto essere progettato in questo modo, abbiamo scelto di non farlo, perché lo scopo di BLAST è quello di consegnare messaggi di grandi dimensioni, non di garantire la consegna del messaggio: al di sopra di BLAST potranno essere presenti altri protocolli che si occupino di garantire la consegna del messaggio.

Potreste, a questo punto, chiedervi perché mai abbiamo inserito in BLAST la possibilità di effettuare ritrasmissioni, dal momento che dobbiamo in ogni caso inserire al di sopra di esso un meccanismo di garanzia della consegna. Il motivo risiede nel fatto che abbiamo preferito ritrasmettere soltanto i frammenti che sono andati perduti, piuttosto di dover ritrasmettere completamente un lungo messaggio ogni volta che ne viene perduto un frammento. In questo modo sfruttiamo le garanzie offerte dal protocollo di livello superiore, ma forniamo, mediante la ritrasmissione di frammenti in BLAST, una miglior efficienza.

Il formato dei messaggi in BLAST

L'intestazione di BLAST deve veicolare diverse informazioni. Innanzitutto, deve contenere una sorta di identificativo del messaggio, in modo che si possano identificare tutti i frammenti che appartengono ad uno stesso messaggio. Secondariamente, ci deve essere il modo di identificare la posizione dei singoli frammenti all'interno del messaggio originario e, analogamente, un SRR deve essere in grado di specificare quali frammenti siano arrivati correttamente e quali siano mancanti. Terzo, deve esistere il modo di distinguere l'ultimo frammento dagli altri, in modo che il ricevitore sappia quando è giunto il momento di controllare se tutti i frammenti sono arrivati. Infine, deve essere possibile distinguere un messaggio SRR da un messaggio contenente dati. Alcune di queste proprietà sono codificate in un campo dell'intestazione in modo ovvio, ma per altre si può procedere in vari modi. La Figura 5.14 presenta il formato dell'intestazione utilizzata da BLAST, mentre la discussione che segue ne illustra i vari campi e prende in esame progetti alternativi.

Il campo MID identifica univocamente il messaggio, e tutti i frammenti che appartengono al medesimo messaggio hanno lo stesso valore nel campo MID. L'unica domanda che ci dobbiamo porre è quanti bit siano necessari per questo campo, una domanda simile a quella che ci siamo posti per il campo SequenceNum del protocollo TCP. Il nodo centrale nel decidere quanti bit usare per il campo MID è correlato con il tempo necessario perché questo campo torni al proprio valore iniziale e il protocollo inizi a riutilizzare gli stessi valori del campo identificativo dei messaggi. Se ciò accade troppo presto (cioè il campo MID ha un numero di bit insufficiente), è possibile che il protocollo venga confuso da un messaggio che viaggia in ritardo nella rete e che una vecchia incarnazione dello stesso identificativo di messaggio venga scambiata per una nuova incarnazione dello stesso identificativo. Quindi, quanti bit

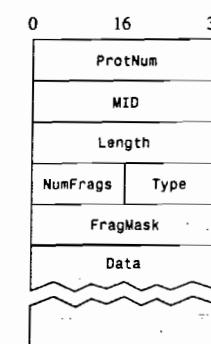


Figura 5.14 Formato dell'intestazione dei messaggi BLAST.

sono sufficienti per avere la certezza che il tempo necessario al campo MID per tornare al proprio valore iniziale sia maggiore del massimo ritardo che può colpire un messaggio nella rete?

Nello scenario peggiore, ciascun messaggio BLAST contiene un singolo frammento lungo 1 byte, per cui il protocollo BLAST deve poter generare un nuovo valore di MID per ogni byte che invia. In una Ethernet a 10 Mbps, ciò significa generare un nuovo valore di MID ogni microsecondo (circa), mentre in una linea STS-24 a 1.2 Gbps servirebbe un nuovo valore di MID ogni 7 nanosecondi. Questo è, ovviamente, un calcolo talmente conservativo da risultare quasi ridicolo, perché la quantità di *overhead* necessaria alla preparazione del messaggio sarebbe, con buona probabilità, maggiore di un microsecondo. Supponiamo, quindi, che serva un nuovo valore di MID ogni microsecondo e che un messaggio possa essere ritardato nella rete fino a 60 secondi (che è la nostra ipotesi standard per il caso peggiore in Internet): di conseguenza, dobbiamo garantire che esistano più di 60 milioni di diversi valori di MID. Anche se un campo di 26 bit sarebbe sufficiente ($2^{26} = 67108864$), è più semplice gestire campi la cui dimensione sia multipla di un byte, per cui usiamo un campo MID a 32 bit.

Questa analisi conservativa (si potrebbe quasi dire paranoica) del campo MID mette in evidenza un punto importante. Quando si progetta un protocollo di trasporto, si potrebbe avere la tentazione di "prendere una scorciatoia" o di fare ipotesi semplificative, perché non tutte le reti presentano i problemi elencati all'inizio di questo capitolo. Ad esempio, in una Ethernet i messaggi non possono rimanere bloccati per 60 secondi e, analogamente, in un segmento di rete Ethernet non è possibile che venga alterata la sequenza dei messaggi. Tuttavia, questo modo di ragionare diventa un problema qualora si voglia che il protocollo di trasporto funzioni su qualsiasi tipo di rete: in tal caso occorre *progettare per il caso peggiore*, perché il vero pericolo è che, non appena ipotizzate che una Ethernet non modifichi l'ordine tra i pacchetti, qualcuno vi inserisca un bridge o un router.

Proseguiamo ora con i successivi campi dell'intestazione di BLAST. Il campo Type indica se si tratta di un messaggio di tipo DATA o di tipo SRR. Notate che, nonostante siamo certi di non aver bisogno di 16 bit per rappresentare questi due tipi, una regola generale ci consiglia di mantenere i campi dell'intestazione allineati su confini di 32 bit (parole), per migliorare l'efficienza di elaborazione. Il campo ProtNum identifica il protocollo che si trova al di sopra di BLAST: i messaggi in arrivo usano questo campo come chiave di demultiplexing. Il campo Length indica quanti byte di dati siano presenti in *questo* frammento e non ha niente a che vedere con la lunghezza del messaggio. Il campo NumFrgs indica di quanti frammenti sia composto il messaggio, e viene usato per determinare se si è ricevuto l'ultimo frammento; un'alternativa consiste nel inserire un valore di flag che segnali l'ultimo frammento.

Infine, il campo FragMask è usato per selezionare frammenti ed è un campo a 32 bit che viene utilizzato come maschera di bit. Per messaggi aventi Type = DATA, il bit *i*-esimo vale 1 (e tutti gli altri valgono 0) per indicare che il messaggio trasporta il frammento *i*-esimo. Per messaggi aventi Type = SRR, il bit *i*-esimo viene posto a 1 per indicare che il frammento *i*-esimo è arrivato, mentre viene posto a 0 per indicare che il frammento *i*-esimo è mancante. Notate che i frammenti possono essere identificati in vari modi. Ad esempio, l'intestazione potrebbe contenere, semplicemente, un campo "identificativo del frammento", che assume il valore *i* per indicare l'*i*-esimo frammento. Con questo approccio diventa difficile, rispetto

alla soluzione che usa un vettore di bit, indicare in un SRR quali frammenti sono arrivati e quali no. Se per identificare ciascun frammento mancante serve un numero di *n* bit, invece di un singolo bit in un vettore di bit di dimensioni fisse, allora il messaggio SRR avrebbe dimensioni variabili, dipendentemente dal numero di frammenti mancanti. Le intestazioni di lunghezza variabile sono consentite, ma sono un po' più complesse da elaborare. D'altra parte, una delle limitazioni dell'intestazione di BLAST che abbiamo appena descritto sta nel fatto che la lunghezza del vettore di bit limita ciascun messaggio ad avere soltanto 32 frammenti. Se la rete sottostante ha un MTU di 1 KB, questa limitazione consente di inviare messaggi di dimensione massima uguale a 32 KB.

5.3.2 Richiesta/Risposta (CHAN)

Il microprotocollo successivo, CHAN, realizza l'algoritmo di richiesta/risposta che costituisce il nucleo di RPC. In termini delle proprietà comuni ai protocolli di trasporto che abbiamo enunciato all'inizio del capitolo, CHAN garantisce la consegna dei messaggi, garantisce che venga consegnata una sola copia di ciascun messaggio e consente la mutua sincronizzazione dei processi comunicanti. Per quanto riguarda quest'ultimo aspetto, la sincronizzazione di cui parliamo realizza il comportamento dell'invocazione di una procedura: l'invocante (il client) rimane bloccato in attesa di una risposta dall'invocato (il server).

La semantica "al massimo una volta"

Il nome CHAN deriva dal fatto che il protocollo realizza un *canale* (*channel*) logico di tipo richiesta/risposta fra una coppia di entità: in ogni istante, ci può essere un'unica transazione di messaggio attiva in un canale. Come il protocollo a canale logico concorrente descritto nella Sezione 2.5.3, i programmi applicativi, se vogliono attivare contemporaneamente più di una sola transazione di tipo richiesta/risposta tra di essi, devono aprire più canali.

La proprietà più importante di ogni canale è che viene preservata una semantica che prende il nome di "al massimo una volta" (*at-most-once*): ciò significa che viene consegnata al server al massimo una sola copia di ogni messaggio di richiesta inviato dal client. In termini del meccanismo RPC, per il cui funzionamento è stato progettato CHAN, ogni volta che il client invoca una procedura remota, tale procedura viene invocata sul server al massimo una volta. Diciamo "al massimo una volta" anziché dire "esattamente una volta" perché è sempre possibile che ci sia un guasto nella rete o nel server, rendendo impossibile la consegna di una copia del messaggio di richiesta.

Nonostante la semantica "al massimo una volta" sembra ovvia, non tutti i protocolli RPC si comportano in questo modo. Alcuni hanno una semantica che viene, spiritosamente, chiamata "zero o più", cioè ogni invocazione di un client genera zero o più invocazioni della procedura remota. Non è difficile capire come questo possa causare problemi ad una procedura remota che modifichi alcune variabili di stato locali (ad esempio, incrementi un contatore) o che abbia qualche effetto visibile esternamente (ad esempio, provochi il lancio di un missile) ogni volta che viene invocata. D'altra parte, se la procedura remota che viene invocata è *idempotente*, cioè più invocazioni hanno lo stesso effetto di una sola invocazione, allora il meccanismo RPC non ha bisogno di fornire supporto alla semantica "al massimo una volta" ed è possibile che un'implementazione più semplice (e potenzialmente più veloce) sia sufficiente.

L'algoritmo CHAN

L'algoritmo di richiesta/risposta ha parecchi dettagli che vanno considerati, per cui lo svilupperemo in più fasi. L'algoritmo di base è banale, come illustrato nel diagramma temporale di Figura 5.15: il client invia un messaggio di richiesta e il server conferma (ACK) di averlo ricevuto, poi, dopo aver eseguito la procedura, il server invia un messaggio di risposta e il client conferma di averlo ricevuto.

Dato che il messaggio di risposta spesso arriva con pochissimo ritardo e a volte accade che il client effettui una seconda richiesta subito dopo aver ricevuto la prima risposta, lo scenario di base può essere ottimizzato usando una tecnica detta *conferma implicita*. Come rappresentato in Figura 5.16, il messaggio di risposta serve per confermare il messaggio di richiesta ed una successiva richiesta conferma la risposta precedente.

Ci sono due fattori che possono complicare il quadro roseo che abbiamo appena presentato. Il primo è che può andare perduto nella rete un messaggio che trasporta dati (un messaggio di richiesta o un messaggio di risposta) oppure un messaggio ACK inviato per confermare un messaggio di dati. Per tener conto di ciò, il client e il server memorizzano una copia di ciascun messaggio che inviano finché non ricevono la conferma corrispondente. Inoltre, cia-

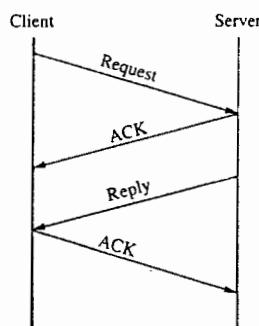


Figura 5.15 Semplice diagramma temporale di CHAN.

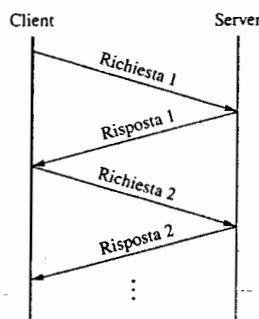


Figura 5.16 Diagramma temporale di CHAN con l'utilizzo di conferme implicite.

scuna entità imposta un temporizzatore, RETRANSMIT, e invia nuovamente il messaggio nel caso in cui tale temporizzatore scada; entrambe le entità reimpostano questo temporizzatore ed effettuano un certo numero, prefissato, di tentativi prima di eliminare il messaggio.

Ricordate dalla Sezione 2.5.1 che questa strategia di conferme e temporizzazioni comporta la possibilità che arrivino a destinazione più copie di un messaggio: arriva il messaggio originario, viene perduta la conferma, quindi arriva il messaggio ritrasmesso. Di conseguenza, il ricevitore deve ricordare quali messaggi ha già visto arrivare e ignorare ogni duplicato: si raggiunge questo scopo mediante l'utilizzo di un campo MID nell'intestazione. Ogni messaggio il cui campo MID non contenga il valore successivo che ci si aspetta per MID viene ignorato, anziché venir trasferito verso il protocollo di livello superiore che si trova al di sopra di CHAN.

La seconda complicazione deriva dal fatto che il server può aver bisogno di un tempo arbitrariamente lungo per produrre il risultato e, cosa ancor peggio, può interrompere bruscamente la propria esecuzione (sia il processo che l'intero elaboratore) prima di aver generato la risposta. Tenete presente che stiamo discutendo di ciò che può accadere nell'intervallo di tempo che inizia con la conferma inviata dal server in seguito alla richiesta e che dovrebbe terminare con l'invio della risposta. Per aiutare il client a riconoscere un server lento da un server non operativo, l'entità client di CHAN invia periodicamente al server un messaggio "Sei vivo?", e l'entità server di CHAN risponde con un messaggio ACK di conferma. In alternativa, il server potrebbe inviare ai client messaggi del tipo "Sono ancora vivo", senza che il client li debba richiedere, ma preferiamo l'approccio che prevede l'iniziativa del client perché rende il server il più semplice possibile (dovendo gestire un temporizzatore in meno).

Il formato del messaggio di CHAN

Il formato del messaggio del protocollo CHAN è illustrato in Figura 5.17. Come nel caso di BLAST, il campo Type identifica il tipo di messaggio; in questo caso, i tipi possibili sono REQ, REP, ACK e PROBE (quest'ultimo è il messaggio "Sei vivo?" di cui abbiamo parlato prima). Analogamente, il campo ProtNum identifica il protocollo di livello superiore che usa i servizi di CHAN.

Il campo CID identifica univocamente il canale logico a cui appartiene il messaggio; si tratta di un campo a 16 bit, per cui CHAN può gestire fino a 64K transazioni concorrenti di

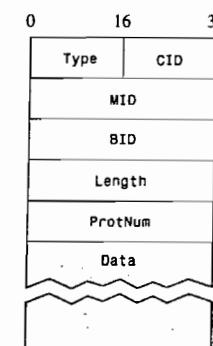


Figura 5.17 Formato dell'intestazione del messaggio di CHAN.

tipo richiesta/risposta fra qualsiasi coppia di host, e, ovviamente, un host può usare canali verso più host contemporaneamente.

Il campo **MID** identifica univocamente ciascuna coppia richiesta/risposta: il messaggio di risposta ha lo stesso valore di **MID** che era presente nella richiesta. Notate che, poiché CHAN permette l'esistenza di un'unica transazione di messaggio su un certo canale in un dato istante, si potrebbe pensare che un campo **MID** di un solo byte sia sufficiente, esattamente come nell'algoritmo *stop-and-wait* presentato nella Sezione 2.5.1. Tuttavia, come in BLAST, dobbiamo preoccuparci dei messaggi che vagano nella rete per un lungo periodo di tempo e, improvvisamente, fanno la loro comparsa a destinazione, confondendo il protocollo CHAN. Di conseguenza, con ragionamenti molto simili a quelli visti nella Sezione 5.3.1, CHAN usa un campo **MID** a 32 bit.

Infine, il campo **BID** indica il *boot id* (*identificativo di accensione*) dell'host. Il *boot id* di un host è un numero che viene incrementato ogni volta che la macchina si accende: viene letto dal disco rigido, incrementato e scritto di nuovo sul disco durante la procedura di avvio dell'elaboratore. Questo numero viene inserito in ogni messaggio inviato dall'host ed il suo ruolo è molto simile a quello svolto dal campo **MID**: fornire protezione contro i messaggi vecchi che possono improvvisamente comparire a destinazione, anche se in questo caso il messaggio vecchio non è dovuto ad un ritardo nella rete ma, piuttosto, ad una macchina che si è riavviata.

Per capire l'utilizzo del *boot id*, considerate la seguente situazione patologica. Una macchina client invia un messaggio di richiesta con **MID** = 0, poi si blocca e viene riavviata. quindi invia un messaggio che non ha niente a che vedere con il precedente, ma che ha nuovamente **MID** = 0. Il server potrebbe non essere al corrente del fatto che il client è stato riavviato e, ricevendo un messaggio di richiesta con **MID** = 0, potrebbe semplicemente confermarlo e ignorarlo in qualità di duplicato. Per evitare che ciò accada, ognuna delle due entità di CHAN verifica che la coppia **(BID, MID)**, e non soltanto il campo **MID**, corrisponda a quanto ci si aspetta. Anche **BID** è un campo a 32 bit, per cui, se ipotizziamo che ci vogliano almeno 10 minuti per ripristinare il funzionamento di un elaboratore, tale campo riassume il suo valore iniziale ogni 40 miliardi di minuti (circa 80000 anni). In realtà, i campi **BID** e **MID** formano, a tutti gli effetti, un identificativo unico, a 64 bit, per le transazioni: i 32 bit meno significativi vengono incrementati ad ogni transazione ma ripartono da zero ogni volta che la macchina viene riavviata, mentre i 32 bit più significativi vengono incrementati ogni volta che la macchina viene, appunto, riavviata.

Temporizzazioni

Il protocollo CHAN utilizza tre diversi temporizzatori: un temporizzatore RETRANSMIT sul client e uno sul server, più un temporizzatore PROBE che viene gestito dal client. Il temporizzatore PROBE non è vitale per le prestazioni e può quindi essere impostato ad un valore sufficientemente elevato, dell'ordine di alcuni secondi. Il temporizzatore RETRANSMIT, invece, influenza le prestazioni di CHAN: se viene impostato ad un valore troppo elevato, il protocollo CHAN potrebbe attendere un tempo inutilmente lungo prima di ritrasmettere un messaggio che è andato perduto nella rete, peggiorando evidentemente le prestazioni; se, invece, viene impostato ad un valore troppo basso, CHAN potrebbe appesantire il traffico di rete inutilmente.

Se CHAN viene progettato per essere eseguito soltanto in una rete locale, o anche in una LAN che si estende all'interno di un'università, allora RETRANSMIT può assumere un valore fisso, probabilmente dell'ordine di 20 millisecondi, perché il valore di RTT in una

5.3 Remote Procedure Call

LAN non è variabile. Se, invece, CHAN deve essere eseguito in Internet, allora la scelta del valore adatto per RETRANSMIT è un problema simile a quello affrontato da TCP. Di conseguenza, CHAN potrebbe calcolare la durata del temporizzatore RETRANSMIT usando una strategia simile a quella descritta nella Sezione 5.2.6: l'unica differenza consiste nel fatto che CHAN deve prendere in considerazione la dimensione del messaggio che viene inviato, che varia da 1 byte a 32 KB, mentre TCP trasmette sempre segmenti che hanno pressappoco la medesima dimensione.

Protocolli sincroni e asincroni

Un modo per caratterizzare un protocollo sta nel fatto che sia *sincrono* oppure *asincrono*, due termini che hanno significati sostanzialmente diversi in relazione a dove vengono usati nella gerarchia di protocolli. Nello strato di trasporto, è più corretto pensare al sincronismo come ad uno spettro di possibilità piuttosto che a due sole alternative: il fattore caratterizzante di ciascun punto di questo spettro è la quantità di informazione nota al mittente nel momento in cui termina l'operazione di invio di un messaggio. In altre parole, se ipotizziamo che un programma applicativo invochi l'operazione *send* di un protocollo di trasporto, allora la domanda è: quando l'operazione *send* termina, quali informazioni ha l'applicazione in merito al successo dell'operazione?

All'estremo *asincrono* dello spettro, l'applicazione non sa assolutamente nulla nel momento in cui *send* termina: non solo non sa se il messaggio sia stato ricevuto dalla pari entità, ma non sa nemmeno se il messaggio sia uscito con successo dall'elaboratore locale. All'estremo *sincrono* dello spettro, tipicamente l'operazione *send* restituisce un messaggio di risposta, cioè l'applicazione non solo sa che il messaggio è stato ricevuto dalla pari entità, ma sa anche che è stata restituita una risposta. Di conseguenza, i protocolli sincroni realizzano l'astrazione richiesta/risposta, mentre i protocolli asincroni vengono usati quando il mittente vuole poter trasmettere molti messaggi senza dover aspettare una risposta. Usando questa definizione, CHAN è ovviamente un protocollo sincrono.

Sebbene non ne abbiamo parlato in questo capitolo, esistono posizioni interessanti tra questi due estremi. Ad esempio, un protocollo di trasporto potrebbe implementare l'operazione *send* in modo che sia bloccante (cioè non termini) finché il messaggio non è stato correttamente ricevuto dalla macchina remota, ma termini prima che l'entità mittente che si trova su tale macchina abbia elaborato la richiesta e abbia fornito una risposta: un protocollo di questo tipo viene detto *protocollo a datagrammi affidabile*.

Implementazione di CHAN

Terminiamo la nostra discussione di CHAN fornendo alcuni frammenti di codice C che realizzano il lato client del protocollo. Leggere il codice può essere noioso, ma se lo si fa con attenzione può essere d'aiuto per rafforzare la comprensione del funzionamento di un sistema. Nel caso del protocollo CHAN, il codice serve ad illustrare tutti i diversi componenti che vanno a costituire l'implementazione di un protocollo e come questi interagiscano tra loro: la funzione che invia un messaggio in uscita, la funzione che ritrasmette i messaggi e la funzione che elabora i messaggi in arrivo.

Iniziamo con le due strutture dati fondamentali di CHAN: *chanHdr* e *chanState*. I campi di *chanHdr* sono già stati illustrati, mentre i campi di *chanState* verranno spiegati nel codice che segue. Notate che *chanState* contiene un campo *hdr_template*, che è una copia dell'intestazione di CHAN. Molti dei campi dell'intestazione di CHAN rimangono gli stessi per tutti i messaggi inviati lungo uno stesso canale: questi campi vengono impostati quando il

canale viene creato (un'operazione che non viene descritta), mentre soltanto i campi variabili vengono modificati prima di trasmettere un messaggio.

```
typedef struct {
    u_short Type; /* tipo di messaggio: REQ, REP, ACK, PROBE */
    u_short CID; /* identificativo univoco del canale */
    int MID; /* identificativo univoco del messaggio */
    int BID; /* boot id */
    int Length; /* lunghezza del messaggio */
    int ProtNum; /* numero del protocollo di più alto livello */
} ChanHdr;

typedef struct {
    u_char type; /* tipo di sessione: CLIENT o SERVER */
    u_char status; /* stato della sessione: BUSY o IDLE */
    Event event; /* evento di temporizzazione */
    int timeout; /* valore del temporizzatore */
    int retries; /* numero di tentativi di ritrasmissione */
    int ret_val; /* valore da restituire */
    Msg *request; /* messaggio di richiesta */
    Msg *reply; /* messaggio di risposta */
    Semaphore reply_sem; /* semaforo su cui si blocca il client */
    int mid; /* identificativo di messaggio del canale */
    int bid; /* boot id del canale */
    ChanHdr hdr_template; /* schema dell'intestazione del canale */
    BlastState blast; /* puntatore al protocollo BLAST */
} ChanState;
```

Volgiamo ora la nostra attenzione alla funzione che invia i messaggi di richiesta. Dato che CHAN presenta ai protocoli di livello superiore un'interfaccia sincrona (l'invocante rimane bloccato finché non viene restituita una risposta), l'operazione send che abbiamo ipotizzato fin dal Capitolo 1 non funzionerà. Di conseguenza, introduciamo una nuova operazione nell'interfaccia, a cui diamo il generico nome di call, che blocca l'invocante finché non è disponibile un messaggio di risposta e restituisce tale messaggio all'invocante. Il suo primo parametro identifica il canale in uso: a tutti gli effetti, contiene tutte le informazioni necessarie per inviare il messaggio alla sua corretta destinazione. Il secondo ed il terzo argomento corrispondono al tipo di dati astratto che rappresenta i messaggi, rispettivamente di richiesta e di risposta, con l'assunzione che sia fornito il supporto per le operazioni consuete (ad esempio, msgSaveCopy e msgLength).

```
int
callCHAN(ChanState *state, Msg *request, Msg *reply)
{
    ChanHdr *hdr;
    char hbuf[HLEN];
    /* garantisce che ci sia una sola transazione per canale */
```

```
if (state->status != IDLE)
    return FAILURE;
state->status = BUSY;

/* memorizza una copia del messaggio di richiesta ed un puntatore
   al messaggio di risposta */
msgSaveCopy(&state->request, request);
state->reply = reply;

/* imposta i campi dell'intestazione */
hdr = state->hdr_template;
hdr->Length = msgLength(request);
if (state->mid == MAX_MID)
    state->mid = 0;
hdr->MID = ++state->mid;

/* allega l'intestazione al messaggio e invialo */
store_chan_hdr(hdr, hbuf);
msgAddrHdr(request, hdr, HLEN);
sendBLAST(request);

/* fa partire il primo temporizzatore di eventi */
state->retries = 1;
state->event=eventSchedule(retransmit, state, state->timeout);

/* rimani bloccato in attesa del messaggio di risposta */
semWait(&state->reply_sem);

/* sistema lo stato e termina */
flush_msg(state->request);
state->status = IDLE;
return state->ret_val;
```

La prima cosa da notare è che la struttura ChanState fornita come argomento a callCHAN contiene un campo di nome status che indica se il canale sia in uso oppure no. Se il canale è in uso, allora la funzione callCHAN restituisce una segnalazione d'errore: una diversa scelta di progetto potrebbe bloccare l'invocante finché il canale non diviene libero: abbiamo scelto di delegare al protocollo di livello superiore, in questo caso SELECT, la responsabilità di bloccare un processo che voglia usare un canale impegnato.

La successiva cosa che va notata in merito all'operazione call è che, dopo aver inserito le informazioni nell'intestazione del messaggio e aver trasmesso il messaggio di richiesta mediante BLAST, il processo invocante rimane bloccato ad un semaforo (reply_sem). Quando arriva il messaggio di risposta, questo viene elaborato dalla funzione deliverCHAN del medesimo protocollo CHAN (si veda in seguito), che copia il messaggio di risposta nella variabile di stato reply e segnala questo fatto al processo bloccato, che finalmente termina la propria esecuzione. Se il messaggio di risposta non dovesse arrivare, verrebbe invocata la funzione

temporizzata `retransmit` (si veda in seguito): questo evento viene predisposto nel corpo di `callCHAN`.

La funzione successiva, `retransmit`, viene invocata ogni volta che scade il temporizzatore di ritrasmissione, che viene fatto partire per la prima volta all'interno di `callCHAN` e viene poi fatto ripartire ogni volta che viene invocata la funzione `retransmit`. Dopo che il messaggio di richiesta è stato ritrasmesso quattro volte, CHAN lo abbandona: imposta a `FAILURE` il valore da restituire e riprende l'esecuzione al processo client che era bloccato in attesa. Infine, ogni volta che viene eseguita la funzione `retransmit` viene inviata una nuova copia del messaggio di richiesta, tale messaggio viene nuovamente memorizzato nella variabile di stato `request`: ciò avviene perché ipotizziamo che, ogni volta che un protocollo invia un messaggio al protocollo di livello inferiore, il riferimento al messaggio venga eliminato.

```
static void
retransmit(Event ev, int *arg)
{
    ChanState *state = (ChanState *)arg;
    Msg      tmp;

    /* se sono stati fatti 4 tentativi, sblocca il processo client */
    if (++stat->retries > 4)
    {
        state->ret_val = FAILURE;
        semSignal(state->rep_sem);
        return;
    }
    /* ritrasmetti il messaggio di richiesta */
    msgSaveCopy(&tmp, &state->request);
    sendBLAST(&tmp);

    /* pianifica nuovamente l'evento, con backoff esponenziale */
    state->timeout = 2 * state->timeout;
    state->event = eventSchedule(retransmit, state, state->timeout);
}
```

Infine, consideriamo la funzione `deliver` di CHAN. La prima cosa da osservare è che CHAN è un protocollo asimmetrico: il codice che realizza CHAN sul client è completamente diverso dal codice che realizza CHAN sul server. Questo comportamento dipende dalla variabile di stato di CHAN (`type`). Di conseguenza, la funzione `deliver` per prima cosa verifica di essere su un server (aspettandosi quindi messaggi di tipo `REQ`) oppure su un client (aspettandosi invece messaggi di tipo `REP`), dopodiché invoca le funzioni specifiche appropriate per il client o per il server. In questo caso, presentiamo la funzione specifica per il client, `deliverClient`.

```
static int
deliverClient(ChanState *state, Msg *msg)
{
    ChanHdr hdr;
    char     *hbuf;
```

5.3 Remote Procedure Call

```
/* elimina l'intestazione e verificane la correttezza */
hbuf = msgStripHdr(msg, HLEN);
load_chan_hdr(&hdr, hbuf);
if (!clnt_msg_ok(state, &hdr))
    return FAILURE;

/* annulla l'evento di temporizzazione per la ritrasmissione */
eventCancel(state->event);

/* se si tratta di un ACK, fa partire il temporizzatore PROBE
   e termina */
if (hdr.Type == ACK)
{
    state->event = eventSchedule(probe, s, PROBE);
    return SUCCESS;
}

/* msg è un REP; memorizzalo e sveglia il client bloccato */
msgSaveCopy(state->reply, msg);
state->ret_val = SUCCESS;
semSignal(&state->reply_sem);

return SUCCESS;
}
```

La funzione `deliverClient` verifica, per prima cosa, di aver ricevuto il messaggio atteso: ad esempio, verifica che il messaggio abbia il valore corretto di `MID` e di `BID`, e che sia di tipo `REP` o `ACK`. Questa verifica viene eseguita dalla funzione `clnt_msg_ok` (che non viene mostrata qui). Se si tratta di un messaggio di conferma valido, allora la funzione `deliverClient` annulla il temporizzatore `RETRANSMIT` e fa partire il temporizzatore `PROBE`: quest'ultimo non viene mostrato, ma è simile al temporizzatore `RETRANSMIT` di cui abbiamo parlato in precedenza. Se il messaggio è, invece, una risposta valida, allora la funzione `deliverClient` annulla il temporizzatore `RETRANSMIT`, memorizza nella variabile di stato `reply` una copia del messaggio di risposta e sblocca il processo client che era bloccato in attesa. È proprio questo processo client che restituisce, in realtà, il messaggio di risposta al protocollo di livello più elevato, mentre il processo che aveva invocato `deliverClient` torna semplicemente verso il basso della pila di protocolli.

5.3.3 Smistamento (SELECT)

L'ultimo microprotocollo, chiamato `SELECT`, smista i messaggi di richiesta alla procedura appropriata: si tratta, per la pila di protocolli RPC, di un protocollo che ha la stessa funzione di demultiplexing che abbiamo visto essere svolta da UDP, con una sostanziale differenza, in quanto questo è un protocollo sincrono, laddove quello era asincrono. Questo significa che, dal lato del client, `SELECT` riceve il numero identificativo della procedura che il client vuole invocare, inserisce tale numero nella propria intestazione e, poi, invoca l'operazione `call` di

un protocollo di livello inferiore come CHAN. Quando questa invocazione termina, SELECT trasferisce semplicemente il valore restituito al client, senza dover fare, in realtà, alcuna operazione di demultiplexing. Sul lato del server, SELECT usa il numero di procedura che trova nella propria intestazione per selezionare la procedura locale corretta da invocare. Quando questa procedura termina, SELECT trasferisce semplicemente il valore restituito al protocollo di livello inferiore che l'aveva invocato.

Può sembrare che SELECT sia così semplice da non richiedere una trattazione in forma di protocollo a sé stante: dopo tutto, CHAN ha già un proprio campo di demultiplexing che potrebbe essere utilizzato per smistare i messaggi di richiesta in arrivo alla procedura appropriata. Nonostante ciò, ci sono due motivi per i quali abbiamo preferito isolare SELECT nella forma di un protocollo autonomo.

Il primo motivo consiste nel fatto che, così facendo, è possibile modificare lo spazio di indirizzamento in cui sono definite le procedure usando semplicemente una diversa versione di SELECT all'interno del grafo di protocolli. In alcune situazioni è sufficiente definire per le procedure uno spazio di indirizzamento *non gerarchico (flat)*, ad esempio un campo di selezione a 16 bit che consenta di identificare univocamente 64K diverse procedure. In altre situazioni, invece, è difficile gestire uno spazio di indirizzamento non gerarchico: chi decide quale procedura assume un certo numero identificativo? In questo caso potrebbe essere meglio poter disporre di uno spazio di indirizzamento gerarchico, cioè di un numero di procedura scomposto in due parti. Prima di tutto, ad ogni programma si potrebbe fornire un numero di programma: in questa trattazione, un programma corrisponde a qualcosa come un "server di file" (*file server*) o un "server di nomi" (*name server*). Successivamente, si potrebbe assegnare ad ogni programma la responsabilità di dare un numero univoco a ciascuna delle proprie procedure. Ad esempio, nel programma che svolge la funzione di server di file, `read` potrebbe essere la procedura 1, `write` potrebbe essere la procedura 2, `seek` potrebbe essere la procedura 3, e così via, mentre nel programma *name server*, `insert` potrebbe essere la procedura 1 e `lookup` la procedura 2.

Il secondo motivo per implementare SELECT come protocollo autonomo sta nel fatto che questo fornisce un valido strumento per gestire più canali concorrenti (*parallel*). Ricordate che CHAN consente soltanto canali con comportamento "al massimo una volta": supponete, invece, di voler consentire alle applicazioni in esecuzione su un host di effettuare più invocazioni della stessa procedura remota prima che questa restituisca un risultato. Poiché CHAN consente l'esistenza di un'unica invocazione attiva per volta, l'unico modo per realizzare questa semanticità è l'apertura di più canali verso lo stesso server. Ogni volta che un processo invocante chiama SELECT, il processo viene instradato verso un canale inattivo; se tutti i canali sono in uso, allora SELECT blocca il processo invocante finché un canale non diviene inattivo.

5.3.4 Mettere tutto insieme (SunRPC, DCE)

Siamo finalmente pronti per costruire una pila RPC a partire dai microprotocolli descritti nelle tre sottosezioni precedenti. Questa sezione illustra anche due protocolli RPC ampiamente diffusi, SunRPC e DCE, in termini dei nostri tre microprotocolli.

Una semplice pila RPC

La Figura 5.18 mostra una semplice pila di protocolli che implementa RPC. Nella parte inferiore troviamo i protocolli che realizzano la rete sottostante: anche se questa pila po-



Figura 5.18 Una semplice pila RPC.

trebbe contenere protocolli corrispondenti ad una qualunque delle tecnologie di rete di cui abbiamo parlato nei capitoli precedenti, per fare un esempio usiamo IP al di sopra di Ethernet.

Al di sopra di IP si trova BLAST, che trasforma la piccola dimensione dei messaggi della rete sottostante in un servizio di comunicazione che può trasmettere messaggi lunghi fino a 32 KB. Notate che non è sempre vero che la rete sottostante trasporti soltanto piccoli messaggi: IP può gestire messaggi la cui dimensione massima è 64 KB. Tuttavia, dato che IP deve frammentare tali grandi messaggi prima di inviarli lungo la rete Ethernet, e l'algoritmo di frammentazione/ricostruzione di BLAST è migliore di quello di IP (perché è in grado di ritrasmettere i frammenti mancati in modo selettivo), preferiamo considerare che IP abbia esattamente lo stesso valore di MTU della rete fisica sottostante. In questo modo il compito della frammentazione/ricostruzione viene affidato a BLAST, a meno che IP non si trovi a dover effettuare la frammentazione nel mezzo della rete, da qualche parte.

Successivamente, CHAN implementa l'algoritmo di richiesta/risposta. Ricordate che abbiamo deciso di non implementare la consegna affidabile in BLAST, rimandando la soluzione di questo problema ad un protocollo di livello superiore. In questo caso, il meccanismo di temporizzazione e conferma utilizzato in CHAN fornisce la garanzia che i messaggi vengano consegnati. Altri protocolli potrebbero decidere di non implementare affatto la consegna affidabile. Questo è un esempio di applicazione del ragionamento *end-to-end*: non fare nei livelli inferiori del sistema (es: BLAST) ciò che può essere fatto a livelli superiori (es: CHAN).

Infine, SELECT definisce uno spazio di indirizzamento per l'identificazione delle procedure remote. Come abbiamo evidenziato nella Sezione 5.3.3, al di sopra di CHAN possono essere utilizzate diverse versioni di SELECT, ognuna delle quali definisce un diverso metodo per identificare le procedure. In realtà, sarebbe addirittura possibile scrivere una versione di SELECT che implementi l'indirizzamento per procedure di qualche pacchetto RPC esistente (come SunRPC), e poi usare CHAN e BLAST al di sotto di questo nuovo SELECT per implementare la parte restante della pila RPC. Questa nuova pila non potrebbe cooperare con il protocollo originale, ma consentirebbe di inserire un nuovo sistema RPC al di sotto di un insieme esistente di procedure remote senza doverne modificare l'interfaccia. Inoltre, SELECT gestisce il parallelismo concorrente.

SunRPC

SELECT, CHAN e BLAST, anche se sono protocolli completi e correttamente funzionanti, non sono mai stati standardizzati né adottati diffusamente, per cui rivolgiamo ora la nostra attenzione ad un protocollo RPC assai diffuso, SunRPC. Ironia della sorte, anche SunRPC non è mai stato approvato da alcun ente di standardizzazione, ma è divenuto uno standard di fatto, grazie alla sua ampia diffusione all'interno delle stazioni di lavoro Sun ed al ruolo centrale che svolge nel famoso *file system* di rete di Sun, NFS (Network File System). Nel momento in cui scriviamo, IETF sta valutando l'adozione ufficiale di SunRPC come protocollo standard di Internet.

Sostanzialmente, qualsiasi protocollo RPC si deve preoccupare di tre aspetti: frammentare grandi messaggi, sincronizzare i messaggi di richiesta e risposta e smistare i messaggi di richiesta alla procedura appropriata. SunRPC non fa eccezione, ma, diversamente dalla pila SELECT/CHAN/BLAST, SunRPC affronta questi tre problemi in un ordine diverso e usa algoritmi un po' diversi. Il grafo di protocolli di SunRPC è presentato in Figura 5.19.

Per prima cosa SunRPC implementa l'algoritmo che sta alla base di tutto, la gestione di richiesta e risposta: si tratta della controparte di CHAN, anche se SunRPC si differenzia da CHAN per il fatto che, tecnicamente, non garantisce la semantica "al massimo una volta", in quanto esistono oscure circostanze nelle quali viene consegnata al server una copia duplicata di un messaggio di richiesta (si veda in seguito). Una seconda differenza sta nel fatto che il ruolo di SELECT è suddiviso tra UDP e SunRPC: UDP smista al programma corretto, mentre SunRPC smista alla corretta procedura all'interno del programma (in seguito diremo con maggiore dettaglio come sono identificate le procedure). Infine, la capacità di inviare messaggi di richiesta e di risposta che siano più grandi del valore di MTU della rete, corrispondente alla funzionalità realizzata da BLAST, è gestita da IP. Tuttavia, ricordate che IP non è efficiente come BLAST nell'implementare la frammentazione: BLAST usa la ritrasmissione selettiva, mentre IP non lo fa.

Come abbiamo appena detto, SunRPC usa indirizzi suddivisi in due parti per identificare le procedure remote: un numero di programma a 32 bit e un numero di procedura, pur esso a 32 bit (c'è anche un numero di versione a 32 bit, ma lo ignoreremo nella discussione che segue). Ad esempio, al server NFS è stato assegnato il numero di programma **x00100003** e, all'interno di questo programma, **getattr** è la procedura 1, **setattr** è la procedura 2, **read** è la procedura 6, **write** è la procedura 8, e così via. Ogni programma è raggiungibile inviando un messaggio ad una certa porta UDP: quando arriva un messaggio di richiesta su tale porta, SunRPC lo preleva e invoca la procedura appropriata.

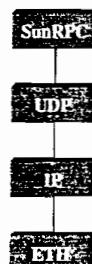


Figura 5.19 Il grafo di protocolli di SunRPC.

Per determinare quale porta corrisponda ad un particolare numero di programma SunRPC, viene utilizzato uno specifico programma SunRPC, chiamato "Port Mapper", che stabilisce una corrispondenza tra numero di programma e numeri di porta. Anche il Port Mapper stesso ha un numero di programma (**x00100000**) che deve essere tradotto in un numero di porta UDP, ma, fortunatamente, questo numero di porta è sempre una porta ben nota (111). Il programma Port Mapper fornisce il servizio tramite diverse procedure, una delle quali (la procedura numero 3) è quella che fornisce la corrispondenza tra numero di programma e numero di porta.

Di conseguenza, per inviare un messaggio di richiesta alla procedura **read** di NFS, per prima cosa un client invia un messaggio di richiesta al Port Mapper, sulla ben nota porta 111, chiedendo l'invocazione della procedura 3 per conoscere la porta UDP corrispondente al numero di programma **x00100003** (anche se, in pratica, NFS è un programma talmente importante che gli è stata assegnata una porta UDP ben nota, per cui non c'è bisogno di invocare il Port Mapper per trovarla). Quindi, il client invia un messaggio di richiesta SunRPC con il numero di procedura 6 a questa porta UDP e il modulo SunRPC in ascolto su tale porta invoca la procedura NFS **read**. Il client memorizza anche, in una cache, la corrispondenza tra il numero di programma e il numero di porta, in modo da non dover passare per il Port Mapper ogni volta che vuol parlare con il programma NFS.

L'attuale intestazione di SunRPC è definita come un complesso insieme di strutture dati annidate dentro l'altra. La Figura 5.20 fornisce i dettagli essenziali nel caso in cui l'invocazione vada a buon fine senza problema. Il campo **XID** è un identificativo univoco di transazione, molto simile al campo **MID** di CHAN. Il motivo per cui SunRPC non può garantire la semantica "al massimo una volta" è dovuto al fatto che, sul lato server, SunRPC non tiene traccia di aver già visto un particolare valore di **XID** dopo aver completato con successo una transazione, un comportamento che pone problemi nel caso in cui il client ritrasmetta un messaggio di richiesta per effetto della scadenza di un temporizzatore e tale richiesta viaggi

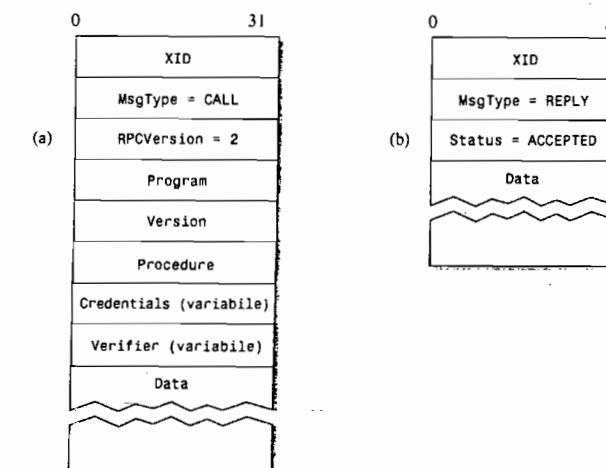


Figura 5.20 Formato dell'intestazione SunRPC: (a) richiesta; (b) risposta.

esattamente nello stesso momento in cui il server sta inviando al client la risposta alla richiesta originaria. Quando la richiesta ritrasmessa giunge al server, assume l'aspetto di una nuova transazione, poiché il server pensa di aver già completato la transazione avente tale valore di XID. Chiaramente, se la risposta arriva al client prima che scada la temporizzazione, la richiesta non verrà ritrasmessa; analogamente, se la richiesta ritrasmessa arriva al server prima che sia stata generata la risposta, allora il server si accorgerà che la transazione corrispondente a tale valore di XID è già in atto e ignorerà il messaggio di richiesta duplicato, per cui è veramente improbabile che accada questo comportamento erroneo. Notate anche che questa memoria a breve termine del server in merito ai valori di XID comporta anche il fatto che il server non possa proteggersi da messaggi che sono stati ritardati a lungo dalla rete, anche se questo non è un vero problema per SunRPC, che è stato progettato per essere utilizzato all'interno di LAN.

Tornando al formato dell'intestazione di SunRPC, il messaggio di richiesta contiene campi **Credentials** e **Verifier** di lunghezza variabile, entrambi utilizzati dal client per autenticarsi presso il server, cioè per fornire evidenza del fatto di avere il diritto di invocare il server. Come possa un client autenticarsi presso un server è un problema generale che deve essere risolto da ogni protocollo che voglia fornire un ragionevole livello di sicurezza: parleremo in maggiore dettaglio di ciò nel Capitolo 8.

DCE

Il Distributed Computing Environment (DCE, un insieme di standard e di software per costruire sistemi distribuiti) definisce un altro protocollo RPC ampiamente utilizzato, che chiamiamo DCE-RPC. DCE è stato definito dalla Open Software Foundation (OSF), un consorzio di produttori di computer che comprendeva, in origine, IBM, Digital e Hewlett-Packard; oggi OSF ha preso il nome di Open Group. DCE-RPC è il protocollo RPC che sta alla base del sistema DCE: può essere utilizzato con il compilatore di stub NDR (Network Data Representation) descritto nel Capitolo 7, ma serve anche come protocollo RPC sottostante in CORBA (Common Object Request Broker Architecture), che è uno standard industriale per la progettazione di sistemi distribuiti e orientati agli oggetti.

DCE-RPC è progettato per essere eseguito al di sopra di UDP ed è simile a SunRPC per il fatto che definisce uno schema di indirizzamento in due parti: UDP esegue il demultiplexing verso il server corretto, mentre DCE-RPC smista le richieste verso la procedura appropriata tra quelle rese disponibili ("esportate") dal server; per sapere come raggiungere un particolare server, il client consulta un "servizio di corrispondenza tra punti estremi" ("endpoint mapping service", simile al Port Mapper di SunRPC). Diversamente da SunRPC, però, DCE-RPC implementa la semantica "al massimo una volta", in un unico protocollo che, sostanzialmente, combina gli algoritmi di BLAST e CHAN. Concentriamo ora la nostra attenzione su questo aspetto di DCE-RPC (in realtà, DCE-RPC consente diverse semantiche di invocazione, tra cui anche una semantica idempotente come quella di SunRPC, ma il comportamento di default è la semantica *at-most-once*).

La Figura 5.21 delinea un diagramma temporale del tipico scambio di messaggi, ciascuno dei quali è etichettato dal proprio tipo in DCE-RPC. Lo schema è simile a quello di CHAN: il client invia un messaggio **Request**, il server prima o poi risponde con un messaggio **Response** e il client conferma la risposta (**Ack**). Invece di far inviare un messaggio di conferma dal server per l'avvenuta ricezione della richiesta, il client invia periodicamente un messaggio **Ping** al server, che risponde con un messaggio **Working** per indicare che la procedura remota è ancora in fase di elaborazione. Sebbene non mostrati nella figura, sono possibili altri tipi di

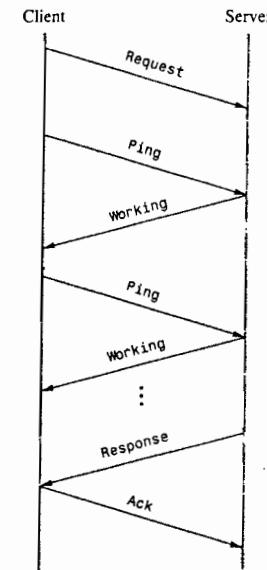


Figura 5.21 Tipico scambio di messaggi in DCE-RPC.

messaggi. Ad esempio, il client può inviare al server un messaggio **Quit**, chiedendo di terminare bruscamente una precedente invocazione che sia ancora in fase di elaborazione: il server risponde con un messaggio di tipo **Quack (quit acknowledgement)**. Ancora, il server può rispondere ad un messaggio **Request** con un messaggio **Reject** (segnalando che la richiesta è stata rifiutata) e può rispondere ad un messaggio **Ping** con un messaggio **Nocall** (segnalando che il server non ha mai ricevuto una richiesta dal client).

Oltre al tipo di messaggio, i messaggi di richiesta e risposta contengono quattro campi chiave (**ServerBoot**, **ActivityId**, **SequenceNum** e **FragmentNum**) che vengono usati per implementare sia gli aspetti relativi alla frammentazione/ricostruzione, tipici di BLAST, sia quelli relativi alle transazioni, tipici di CHAN.

Il campo **ServerBoot** ha lo stesso scopo del campo **BID (boot id)** di CHAN: il server memorizza il proprio orario di accensione in una variabile globale ogni volta che viene fatto ripartire, e inserisce tale valore in ogni invocazione a cui risponde. Il campo **ActivityId** è simile al campo **CID (channel id)** di CHAN: identifica una connessione logica tra il client e il server, lungo la quale è possibile effettuare una sequenza di invocazioni. Il campo **SequenceNum**, poi, distingue diverse invocazioni che fanno parte della medesima attività e ha lo stesso scopo del campo **MID (message id)** di CHAN e del campo **XID (transaction id)** di SunRPC. Come CHAN (ma diversamente da SunRPC), DCE-RPC tiene traccia dell'ultimo numero di sequenza usato come parte di una certa attività, in modo da garantire la semantica "al massimo una volta".

Dato che sia i messaggi di richiesta sia quelli di risposta possono avere dimensione maggiore del pacchetto della rete sottostante, è possibile che vengano frammentati in più pac-

chetti: il campo `FragmentNum` identifica ciascun frammento che faccia parte di un certo messaggio di richiesta o di risposta. Diversamente da BLAST, che usa un vettore di bit per identificare i frammenti, ad ogni frammento DEC-RPC viene assegnato un numero di frammento univoco (ad esempio, 0, 1, 2, 3, e così via): il client e il server implementano un meccanismo di conferma selettiva, che funziona come segue (descriviamo il meccanismo dal punto di vista di un client che invia al server un messaggio di richiesta frammentato, ma lo stesso ragionamento si applica quando un server invia una risposta frammentata al client).

Prima di tutto, ogni frammento che compone il messaggio di richiesta contiene sia un `FragmentNum` univoco sia un indicatore del fatto che il pacchetto sia un frammento di un'invocazione (`frag`) o sia l'ultimo frammento di un'invocazione (`last_frag`); i messaggi di richiesta che trovano posto in un singolo pacchetto hanno l'indicatore `no_frag`. Il server sa di aver ricevuto il messaggio di richiesta completo quando riceve il pacchetto `last_frag` senza aver osservato interruzioni nella sequenza dei numeri di frammento. Secondariamente, in risposta ad ogni frammento arrivato, il server invia un messaggio di tipo `Fack` (*fragment acknowledgement*, conferma di frammento), che segnala il numero di frammento più alto che il server ha ricevuto con successo. In altre parole, si tratta di una conferma cumulativa, in modo molto simile a quanto avviene nel protocollo TCP. Come ulteriore segnalazione, però, il server conferma selettivamente ogni numero di frammento più elevato che ha ricevuto fuori sequenza, mediante un vettore di bit che identifica questi frammenti fuori sequenza relativamente al più elevato frammento che è stato ricevuto in sequenza. Infine, il client risponde ritrasmettendo i frammenti mancanti.

La Figura 5.22 mostra come funziona tutto ciò. Supponete che il server abbia ricevuto con successo i frammenti fino al numero 20, più i frammenti 23, 25 e 26. Il server risponde con un `Fack` che segnala il frammento 20 come il frammento più elevato tra quelli ricevuti in

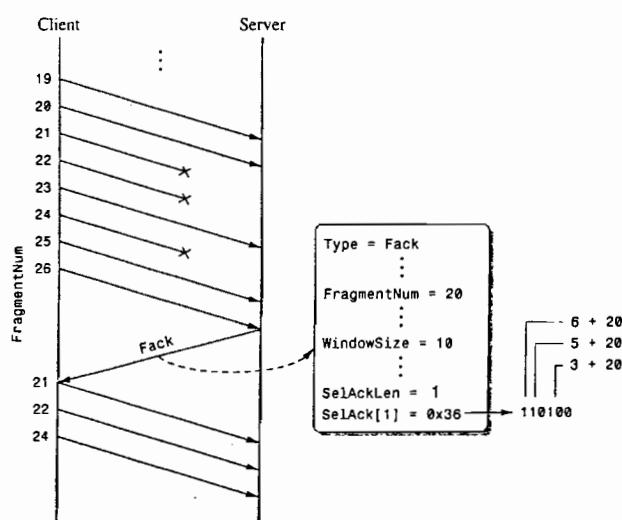


Figura 5.22 Frammentazione con conferme selettive.

sequenza, con l'aggiunta di un vettore di bit (`SelAck`) che presenta con il valore 1 il terzo ($23 = 20 + 3$), il quinto ($25 = 20 + 5$) e il sesto ($26 = 20 + 6$) bit. Per poter usare un vettore di bit di lunghezza (quasi) arbitraria, la dimensione del vettore (misurata in parole di 32 bit) è indicata nel campo `SelAckLen`.

Dato che DCE-RPC consente l'invio di messaggi molto grandi (il campo `FragmentNum` è lungo 16 bit, per cui può gestire 64K frammenti), non è appropriato che il protocollo trasmetta il più velocemente possibile tutti i frammenti che compongono un messaggio, come fa BLAST, perché ciò potrebbe sovraccaricare il ricevitore. Al contrario, DCE-RPC implementa un algoritmo di controllo di flusso che è molto simile a quello usato dal protocollo TCP. Specificatamente, ogni messaggio `Fack` non soltanto conferma i frammenti ricevuti, ma informa anche il mittente in merito a quanti frammenti possono essere inviati in quel momento. A ciò serve il campo `WindowSize` di Figura 5.22, che ha esattamente lo stesso significato del campo `AdvertisedWindow`, con l'unica differenza che conta i frammenti anziché contare byte. DCE-RPC implementa anche un meccanismo di controllo della congestione simile a quello di TCP, che vedremo nel Capitolo 6.

5.4 Prestazioni

Ricordate che il Capitolo 1 ha introdotto due metriche quantitative per valutare le prestazioni delle reti: latenza e throughput. Come accennato in quella discussione, tali metriche sono influenzate non soltanto dall'hardware sottostante (ad esempio, ritardo di propagazione e ampiezza di banda della linea di collegamento) ma anche dagli *overhead* software. Ora che abbiamo a disposizione un grafo di protocolli basato completamente sul software, che contiene diversi protocolli di trasporto alternativi, possiamo discutere in merito a come misurare in modo significativo le sue prestazioni, che sono assai importanti perché rappresentano le prestazioni percepite dai programmi applicativi.

Iniziamo, come dovrebbe fare qualsiasi relazione sperimentale, descrivendo il nostro metodo sperimentale, che comprende gli apparati utilizzati negli esperimenti. Abbiamo eseguito i nostri esperimenti su una coppia di stazioni di lavoro equipaggiate con Pentium a 733 MHz e connesse da una rete Ethernet a 100 Mbps isolata. La rete Ethernet si estendeva all'interno di un'unica sala macchine, per cui la propagazione non è un problema da prendere in considerazione, rendendo la prova una misura degli *overhead* dovuti al software e al processore. Ciascuna stazione di lavoro eseguiva il sistema operativo Linux (*kernel* 2.4) ed un programma di prova, in esecuzione al di sopra dell'interfaccia socket, faceva rimbalzare messaggi fra le due macchine. La Figura 5.23 mostra un percorso di andata e ritorno (*round trip*) tra i due programmi di prova.

Ciascun esperimento è consistito di tre esecuzioni identiche del medesimo programma di prova. Ogni prova, a sua volta, prevedeva l'invio di un messaggio di una qualche dimensione specifica, avanti e indietro fra le due macchine, 10000 volte. L'orologio di sistema veniva consultato all'inizio e alla fine di ogni prova e la differenza tra le due letture veniva divisa per 10000 per determinare il tempo richiesto da ciascun round trip. La media di questi tre valori (corrispondenti alle tre esecuzioni del programma di prova) viene riportata nel seguito per ciascun esperimento. Ogni esperimento ha usato una diversa dimensione di messaggio. I valori di latenza sono relativi a messaggi di dimensioni uguali a 1 byte, 100 byte, 200 byte, ..., 1000 byte. I risultati di throughput sono relativi a messaggi di 1 KB, 2 KB, 4 KB, 8 KB, ..., 32 KB.

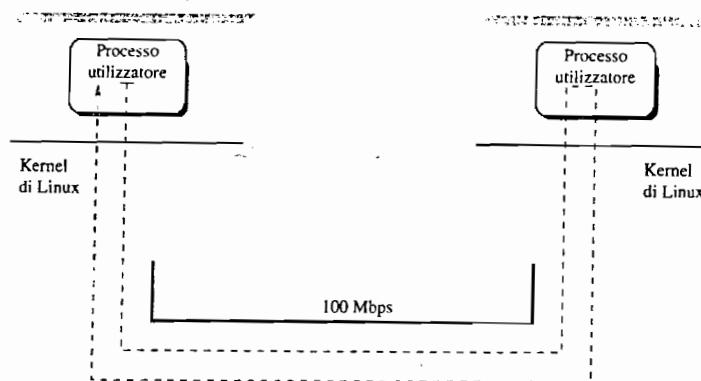


Figura 5.23 Sistema di misura: due stazioni di lavoro Pentium allestite con Linux e connesse mediante una Ethernet a 100 Mbps.

La tabella 5.3 presenta i risultati delle misurazioni di latenza. Come ci si potrebbe attendere, la latenza aumenta con la dimensione del messaggio. Nonostante esistano alcune situazioni speciali in cui potreste essere interessati alla latenza di un messaggio di 100 byte, ad esempio, tipicamente la latenza più importante è quella relativa a messaggi di 1 byte, perché ciò rappresenta l'*overhead* necessario per l'elaborazione di ogni messaggio, indipendentemente dalla quantità di dati contenuta nel messaggio stesso. Si tratta, tipicamente, del limite inferiore alla latenza, dipendente da valori come il ritardo dovuto alla velocità finita della luce e al tempo necessario per l'elaborazione dell'intestazioni. Notate come ci sia un piccola differenza fra la latenza rilevata sperimentalmente per le due diverse pile di protocolli, con il protocollo UDP che richiede un tempo di round trip lievemente inferiore a quello richiesto da TCP: il risultato era prevedibile, perché TCP fornisce maggiori funzionalità.

Tabella 5.3 Latenza di *round trip* misurata (in ms) per diverse dimensioni di messaggi e diversi protocolli.

Dimensione del messaggio (byte)	UDP	TCP
1	58	66
100	76	84
200	93	104
300	111	124
400	132	136
500	150	159
600	167	176
700	184	194
800	203	210
900	223	228
1000	239	249

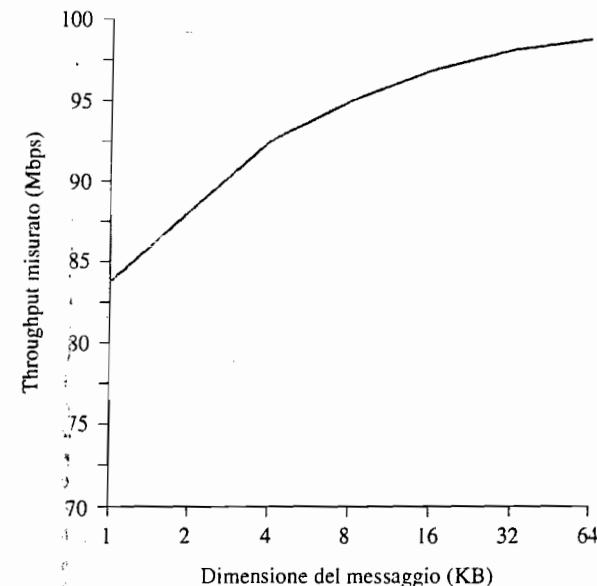


Figura 5.24 Throughput misurato usando UDP, per diverse dimensioni di messaggio.

I risultati della prova di throughput sono presentati in Figura 5.24, dove mostriamo solamente i dati relativi a UDP. Il fatto più importante da notare in questo grafico è che il throughput migliora all'aumentare della dimensione dei messaggi, osservazione assolutamente sensata, dato che ogni messaggio richiede una certa quantità di *overhead*, che viene ammortizzato su più byte nel caso di messaggi più grandi. La curva relativa al throughput si appiattisce al di sopra di 16 KB, il punto in cui l'*overhead* specifico di ciascun messaggio diviene trascurabile in confronto al gran numero di byte che devono essere elaborati dalla pila di protocolli.

Una seconda cosa da notare è che la curva del throughput non raggiunge i 100 Mbps. Anche se non è possibile dedurlo da queste misurazioni, si è visto che il fattore che impedisiva al nostro sistema di sfruttare appieno la velocità della rete Ethernet era una limitazione dell'adattatore di rete piuttosto che del software.

5.5 Riepilogo

Questo capitolo ha descritto tre protocolli di trasporto molto diversi. Il primo protocollo che abbiamo considerato esegue semplicemente la funzione di demultiplexing e UDP ne è un esemplare significativo. Tutto ciò che deve fare un protocollo di questo tipo è lo smistamento di messaggi al processo applicativo appropriato, in base ad un numero di porta, senza apportare alcun miglioramento al modello di servizio *best-effort* della rete sottostante: detto in altro modo, offre ai programmi applicativi un servizio a datagrammi non affidabile e non orientato alla connessione.

Il secondo tipo è un protocollo affidabile a flusso di byte e TCP ne costituisce l'esempio che abbiamo esaminato. I problemi che devono essere risolti da un protocollo di questo tipo sono il recupero di messaggi che possono essere andati perduti nella rete, la consegna dei messaggi nello stesso ordine in cui sono stati inviati e la possibilità, per il ricevente, di controllare il flusso inviato dal mittente. Il protocollo TCP realizza questa funzionalità usando la versione di base dell'algoritmo *sliding window*, migliorata con la comunicazione della dimensione della finestra. L'altro aspetto degno di nota in relazione a questo protocollo è l'importanza di un accurato meccanismo di ritrasmissione, mediante temporizzazioni che scadono. Anche se TCP è un unico protocollo, abbiamo visto che impiega almeno cinque diversi algoritmi, tutti potenzialmente adatti a qualsiasi protocollo di trasporto: *sliding window*, Nagle, three-way handshake, Karn/Partridge e Jacobson/Karels.

Il terzo protocollo di trasporto che abbiamo preso in esame è un protocollo di richiesta/risposta che costituisce la base di RPC. In questo caso viene utilizzata una combinazione di tre diversi algoritmi per realizzare il servizio di richiesta/risposta: un algoritmo di ritrasmissione selettiva che viene usato per frammentare e ricostruire messaggi di grandi dimensioni, un algoritmo a canale sincrono che accoppia il messaggio di richiesta con il messaggio di risposta e un algoritmo di smistamento che provoca l'invocazione della procedura corretta.

Problema aperto

Protocolli per applicazioni specifiche

Ciò che dovrebbe risultare chiaro dopo aver letto questo capitolo è che la progettazione di un protocollo di trasporto è un problema complesso. Come abbiamo visto, è abbastanza difficile progettare al primo tentativo un protocollo di trasporto corretto, ma ciò è reso ancora più complicato dalla modifica delle condizioni operative. Il problema è trovare il modo di adattarsi a questi cambiamenti.

La nostra esperienza relativa all'utilizzo del protocollo può modificarci. Come abbiamo visto con il meccanismo delle temporizzazioni in TCP, l'esperienza ha portato ad una serie di modifiche nel modo in cui TCP prende la decisione di ritrasmettere un segmento, anche se nessuna di queste modifiche ha provocato cambiamenti nel formato dell'intestazione di TCP e, quindi, è stato possibile incorporarle nel protocollo una alla volta: non c'è stato bisogno che, in un dato momento, tutti aggiornassero la propria versione di TCP.

Anche le caratteristiche della rete sottostante possono cambiare. Per molti anni il numero di sequenza a 32 bit e la finestra comunicata con 16 bit del protocollo TCP sono stati adeguati, ma recentemente le reti con più elevata ampiezza di banda hanno reso non più sufficientemente grande il numero di sequenza per fornire protezione contro il ritorno al valore iniziale, mentre la finestra che può essere comunicata è troppo piccola per consentire al mittente di mantenere pieno il canale attraverso la rete. Mentre la soluzione più ovvia sarebbe stata quella di ridefinire l'intestazione di TCP, prevedendo un campo a 64 bit per il numero di sequenza e un campo a 32 bit per il valore della finestra, ciò avrebbe provocato un serio problema per la transizione simultanea di milioni di host di Internet verso questa nuova intestazione. Anche se queste trasformazioni sono state attuate in alcune reti, come la rete telefonica, non sono banali: si è quindi deciso di implementare le necessarie estensioni come opzioni, permettendo agli host di concordare vicendevolmente se usare o meno le opzioni per ciascuna connessione.

Tuttavia, questo approccio non funzionerà indefinitamente, perché l'intestazione TCP ha spazio per soli 44 byte di opzioni (perché il campo `HdrLen` è lungo 4 bit, dando luogo ad una lunghezza massima dell'intestazione TCP che non può eccedere il numero di 16 parole

di 32 bit ciascuna, ovvero 64 byte). Ovviamente, è possibile che un'opzione TCP estenda lo spazio disponibile per le opzioni, ma vale la pena di chiedersi quanto bisogna spingersi in avanti nella ricerca della compatibilità con il passato.

Forse, le modifiche che faticano di più a trovare una collocazione sono gli adattamenti del livello di servizio richiesto dai programmi applicativi. È inevitabile che alcune applicazioni possano avere buone ragioni per richiedere piccole variazioni rispetto ai servizi standard: ad esempio, alcune applicazioni potrebbero aver bisogno di RPC per la maggior parte del tempo, ma potrebbero volerlo, di tanto in tanto, inviare un flusso di messaggi di richiesta senza attendere per le relative risposte. Anche se questo, tecnicamente, non appartiene più alla semantica di RPC, è pratica comune modificare un protocollo RPC esistente per consentire questa flessibilità. Come ulteriore esempio, dato che le applicazioni video sono orientate ai flussi, vi è la tentazione di usare TCP come protocollo di trasporto: sfortunatamente, il protocollo TCP garantisce l'affidabilità, che non è importante per le applicazioni video, che preferiscono perdere un *frame* di immagine (un segmento) piuttosto che attendere la sua ritrasmissione. Tuttavia, piuttosto che inventare da zero un nuovo protocollo di trasporto, alcuni progettisti hanno proposto che il protocollo TCP debba fornire il supporto per un'opzione che, a tutti gli effetti, disabilita la sua caratteristica di affidabilità. Sembra difficile poter continuare a chiamare TCP un protocollo di questo tipo, ma stiamo parlando degli aspetti pratici che consentono ad un'applicazione di funzionare.

Come sviluppare protocolli di trasporto che possano evolvere per soddisfare i requisiti di diverse applicazioni, molte delle quali non sono ancora state immaginate, è un problema difficile. Può darsi che la risposta definitiva a questi problemi consista nello stile illustrato dai microprotocolli che abbiamo usato per implementare RPC, che prevede un'unica funzione per ciascun protocollo, o altri meccanismi simili, mediante i quali il programmatore di applicazioni possa programmare, configurare o, in ogni caso, personalizzare il protocollo di trasporto.

Ulteriori letture

Non c'è dubbio che TCP sia un protocollo complesso e che sia ricco di dettagli che non sono stati presi in esame in questo capitolo, per cui le letture che raccomandiamo in relazione a questo capitolo contengono la specifica originaria del protocollo TCP. Il motivo per cui consigliamo tale lettura non consiste tanto nel completamento dei dettagli mancati, quanto nel farvi vedere come dovrebbe essere una "buona e onesta" specifica di protocollo. Gli altri due lavori che fanno parte della lista consigliata trattano l'argomento RPC: il lavoro di Birrell e Nelson è il lavoro principe sull'argomento, mentre l'articolo di O'Malley e Peterson descrive in maggiore dettaglio la filosofia di progetto "una funzione per ogni protocollo".

- USC-ISI "Transmission Control Protocol", *Request for Comments 793*, September 1981.
- Birrell, A., e B. Nelson "Implementing remote procedure calls", *ACM Transactions on Computer Systems* 2(1):39-59, February 1984.
- O'Malley, S., e L. Peterson "A dynamic network architecture", *ACM Transactions on Computer Systems* 10(2):110-143, May 1992.

Al di là della specifica del protocollo, la descrizione più completa di TCP, comprendente anche la sua implementazione in Unix, si trova in [Ste94b] e [SW95]. Ancora, il terzo volume della serie di libri su TCP/IP di Comer e Stevens descrive come scrivere applicazioni

client/server basate su TCP e UDP, usando l'interfaccia socket Posix [CS00], l'interfaccia socket di Windows [CS97] e l'interfaccia TLI di Unix System V [CS94]. Parecchi lavori valutano le prestazioni di diversi protocolli di trasporto ad un livello di dettaglio molto elevato. Ad esempio, l'articolo di Clark *et al.* [CJRS89] misura gli *overhead* di elaborazione di TCP, un lavoro di Mosberger *et al.* [MPBO96] esplora i limiti degli *overhead* nell'elaborazione del protocollo e Thekkath e Levy [TL93] e Schroeder e Burrows [SB89] esaminano in grande dettaglio le prestazioni di RPC.

Il calcolo originale delle temporizzazioni di TCP era descritto nelle specifiche di TCP (si veda in precedenza), mentre l'algoritmo di Karn/Partridge è stato descritto in [KP91] e l'algoritmo di Jacobson/Karels è stato proposto in [Jac88]. Le estensioni TCP sono state definite da Jacobson *et al.* [JBB92], mentre O'Malley e Peterson [OP91] hanno teorizzato che l'estensione di TCP in tal modo non fosse l'approccio corretto per la soluzione del problema.

Infine, esistono numerosi sistemi operativi distribuiti che hanno definito propri protocolli RPC. Esempi significativi sono: il sistema V, descritto da Cheriton e Zwaenepoel [CZ85], Sprite, descritto da Ousterhout *et al.* [OCD+88], e Amoeba, descritto da Mullender [Mul90]. L'ultima versione di SunRPC, come è stata definita da Srinivasan [Sri95a], è una proposta di standard per Internet.

Esercizi

- Se un datagramma UDP viene inviato dall'host A, porta P, all'host B, porta Q, ma sull'host B non c'è nessun processo in ascolto sulla porta Q, allora B invia in risposta ad A un messaggio ICMP di tipo Port Unreachable. Come tutti i messaggi ICMP, questo viene indirizzato ad A, non alla porta P di A.
 - Fornite un esempio di applicazione che possa voler ricevere tali messaggi ICMP.
 - Identificate cosa debba fare l'applicazione, su un sistema operativo di vostra scelta, per ricevere tali messaggi.
 - Perché potrebbe non essere una buona idea inviare tali messaggi direttamente alla porta P di A da cui è partito il messaggio originario?
- Considerate un semplice protocollo, basato su UDP, per richiedere dei file (che ricorda, in qualche modo, il protocollo TFTP, Trivial File Transfer Protocol). Il client invia un'iniziale richiesta di file e il server risponde (se il file può essere inviato) con il primo pacchetto di dati. Client e server, poi, procedono con un meccanismo di trasmissione stop-and-wait.
 - Descrivete uno scenario in cui un client potrebbe richiedere un file ma riceverne uno diverso; potete consentire all'applicazione client di terminare bruscamente e di iniziare nuovamente l'esecuzione sulla medesima porta.
 - Proponete una modifica del protocollo che renda meno probabile tale situazione.
- Progettate un semplice protocollo, basato su UDP, per recuperare file da un server, senza la necessità di prevedere autenticazione e potendo usare una trasmissione dei dati di tipo stop-and-wait. Il vostro protocollo dovrebbe risolvere i problemi seguenti:
 - Una duplicazione del primo pacchetto non deve provocare una duplicazione della "connessione".
 - La perdita della conferma finale non deve necessariamente lasciare al server il dubbio che il trasferimento sia avvenuto con successo.
 - Un pacchetto di una precedente connessione che arrivi in ritardo non deve poter essere interpretato come parte della connessione attuale.

- Questo capitolo ha illustrato tre sequenze di transizioni di stato durante la terminazione di una connessione TCP. Esiste una quarta sequenza possibile, che percorre un'ulteriore arco da FIN_WAIT_1 a TIME_WAIT, etichettato con FIN + ACK/ACK. Spiegate cosa possa innescare questa quarta sequenza di terminazione.
- Quando si chiude una connessione TCP, perché nella transizione da LAST_ACK a CLOSED non è necessaria la temporizzazione di durata uguale al doppio del tempo di vita di un segmento?
- Il mittente di una connessione TCP che riceva la comunicazione di una finestra di dimensione 0 stimola periodicamente il ricevitore per scoprire se la finestra ha assunto dimensione diversa da zero. Perché il ricevitore avrebbe bisogno di un temporizzatore supplementare nel caso in cui avesse il compito di segnalare al mittente che la propria finestra è diventata di dimensione diversa da zero (cioè se il mittente non stimolasse il ricevitore)?
- Consultate la documentazione man (o l'equivalente Windows) per il programma di utilità netstat. Usate netstat per scoprire lo stato delle connessioni TCP locali. Scoprite quanto tempo venga dedicato alla chiusura delle connessioni, permanendo nello stato TIME_WAIT.
- Il campo del numero di sequenza nell'intestazione TCP è lungo 32 bit, una dimensione sufficiente a gestire più di 4 miliardi di byte di dati. Come mai il numero di sequenza può comunque passare dal valore $2^{32} - 1$ al valore 0, anche se lungo una singola connessione non è mai stato trasferito un numero di byte così elevato?
- Vi è stato assegnato il compito di progettare un protocollo affidabile a flusso di byte che usi l'algoritmo sliding window (come TCP). Il protocollo verrà eseguito in una rete a 100 Mbps, con RTT di 100 ms e MSL di 60 secondi.
 - Quanti bit inserireste nei campi AdvertisedWindow e SequenceNum dell'intestazione del vostro protocollo?
 - Come determinereste i numeri forniti al punto precedente, e quali valori sarebbero più incerti?
- ✓ Vi è stato assegnato il compito di progettare un protocollo affidabile a flusso di byte che usi l'algoritmo sliding window (come TCP). Il protocollo verrà eseguito in una rete a 1 Gbps, con RTT di 140 ms e MSL di 60 secondi. Quanti bit inserireste nei campi AdvertisedWindow e SequenceNum dell'intestazione del vostro protocollo?
- Supponete che un host voglia stabilire l'affidabilità di una linea di connessione inviando pacchetti e misurando la percentuali di quelli che vengono ricevuti; ad esempio, i router fanno in questo modo. Spiegate quanto sia difficile farlo mediante una connessione TCP.
- Supponete che il protocollo TCP operi in una linea di connessione a 1 Gbps.
 - Ipotizzando che TCP possa utilizzare continuativamente l'intera ampiezza di banda, quanto tempo è necessario perché i numeri di sequenza tornino al proprio valore iniziale?
 - Supponete che un campo aggiuntivo che riporta un'indicazione oraria a 32 bit si incrementi 1000 volte durante il tempo appena determinato al punto precedente. Quanto tempo è necessario perché tale indicazione oraria torni al proprio valore iniziale?
- ✓ Supponete che il protocollo TCP operi in una linea di connessione STS-768 a 40 Gbps.
 - Ipotizzando che TCP possa utilizzare continuativamente l'intera ampiezza di banda, quanto tempo è necessario perché i numeri di sequenza tornino al proprio valore iniziale?

- b) Supponete che un campo aggiuntivo che riporta un'indicazione oraria a 32 bit si incrementi 1000 volte durante il tempo appena determinato al punto precedente. Quanto tempo è necessario perché tale indicazione oraria torni al proprio valore iniziale?
14. Se l'host A riceve due pacchetti SYN dalla stessa porta, provenienti dall'host B, il secondo potrebbe essere una ritrasmissione dell'originale oppure, se B è stato riavviato, una richiesta di connessione completamente nuova.
- Descrivete come l'host A veda diversamente questi due casi.
 - Fornite una descrizione algoritmica di cosa debba fare lo strato TCP nel momento in cui riceve un pacchetto SYN. Considerate i casi di pacchetto duplicato/nuovo sopra descritto, nonché la possibilità che nessun processo sia in ascolto nella porta di destinazione.
15. Supponete che x e y siano due numeri di sequenza TCP. Scrivete una funzione per determinare se x precede y (nella notazione del Request for Comments 793, " $x < y$ ") oppure segue y . La soluzione deve funzionare anche quando i numeri tornano al loro valore iniziale.
16. Supponete che fra i socket A e B esista una connessione TCP inattiva e che un intruso si sia posto in ascolto e conosca il numero di sequenza corretto per entrambe le entità.
- Supponete che l'intruso invii ad A un pacchetto manipolato che appaia provenire da B, con 100 byte di nuovi dati. Cosa accade? Suggerimento: Cercate nel Request for Comments 793 quali azioni intraprende il protocollo TCP quando riceve un ACK che non sia un "ACK accettabile".
 - Supponete che l'intruso invii tale pacchetto di 100 byte ad entrambe le entità, manipolandolo in modo che, per ciascuna delle due entità, appaia provenire dall'entità che si trova all'altro estremo della connessione. Cosa accade ora? Cosa accadrebbe se, successivamente, A inviasse a B 200 byte di dati?
17. Supponete che l'host A si connetta ad Internet mediante un server IP con connessione telefonica (usando, ad esempio, SLIP o PPP), che abbia diverse connessioni Telnet aperte (usando TCP) e che perda il collegamento. Successivamente si collega l'host B con lo stesso sistema e gli viene assegnato il medesimo indirizzo IP che era stato precedentemente assegnato ad A. Ipotizzando che B sia in grado di stabilire a quali host si era connesso A, descrivete una sequenza di tentativi mediante i quali B potrebbe ottenere informazioni di stato sufficienti a continuare le connessioni di A.
18. Sono normalmente disponibili programmi di diagnostica che memorizzano, ad esempio, i primi 100 byte di ogni connessione TCP diretta ad una certa coppia (host, port). Indicate cosa si deve fare con ogni pacchetto TCP ricevuto, P, per determinare se contiene dati che appartengono ai primi 100 byte di una connessione diretta all'host HOST e alla porta PORT. Ipotizzate che l'intestazione IP sia P. IPHEAD, che l'intestazione TCP sia P. TCPHEAD e che i campi delle intestazioni abbiano i nomi indicati nelle Figure 4.3 e 5.4. Suggerimento: per determinare i numeri di sequenza iniziali (ISN, initial sequence number) dovete esaminare ogni pacchetto che abbia il bit SYN impostato al valore 1. Trascurate il fatto che i numeri di sequenza vengono, prima o poi, riutilizzati.
19. Se arriva all'host A un pacchetto con l'indirizzo di sorgente B, potrebbe trattarsi facilmente di un pacchetto manipolato da un terzo host C. Se, però, A accetta una connessione TCP da B, allora durante il three-way handshake A invia ISNA all'indirizzo di B e riceve da esso una conferma. Se C non si trova in un punto della rete che gli consenta

- di osservare ISNA, potrebbe sembrare impossibile che C sia in grado di manipolare la risposta di B.
- Tuttavia, l'algoritmo utilizzato per scegliere ISNA fornisce, in realtà, ad altri host che non hanno alcuna relazione con le due entità terminali qualche possibilità di indovinare tale valore. In particolare, A sceglie ISNA in base al valore del proprio orologio di sistema nel momento della connessione. Il documento Request for Comments 793 specifica che questo orologio di sistema deve essere incrementato ogni 4 µs, ma la maggior parte delle implementazioni derivate da quella di Berkeley hanno semplificato questo requisito incrementando il valore di 250000 (o 256000), una volta ogni secondo.
- In base a questa implementazione semplificata, con incremento una volta al secondo, spiegate come un host C qualsiasi possa prendere il posto di B, per lo meno durante l'apertura di una connessione TCP. Potete ipotizzare che B non risponda ai pacchetti SYN + ACK che A tenta di inviare.
 - Ipotizzando che valori reali di RTT si possano stimare intorno ai 40 ms, quanti tentativi sarebbero necessari per realizzare la strategia del punto (a) con l'implementazione non semplificata di TCP, che prevede un incremento ogni 4 µs?
20. L'algoritmo di Nagle, implementato dalla maggior parte delle realizzazioni di TCP, richiede che il mittente memorizzi un segmento di dati parziale (anche se viene invocata l'operazione PUSH) finché non si accumula un intero segmento oppure arriva il segnale ACK relativo al segmento più recente tra quelli in viaggio.
- Supponete che vengano inviati i caratteri abcdefghi, uno al secondo, lungo una connessione TCP con un valore di RTT uguale a 4.1 secondi. Tracciate un diagramma temporale che indichi quando viene inviato ciascun pacchetto, e cosa contiene.
 - Se quanto sopra fosse stato digitato in una connessione Telnet full-duplex, cosa avrebbe visto l'utente?
 - Supponete che i cambiamenti della posizione del mouse vengano inviati lungo la connessione. Ipotizzando che durante ogni RTT vengano inviate più modifiche di posizione, come verrebbe percepito il movimento del mouse in presenza e in assenza dell'algoritmo di Nagle?
21. Supponete che un client C si connetta ripetutamente, mediante il protocollo TCP, ad una data porta di un server S, e che ogni volta sia C ad iniziare la procedura di chiusura della connessione.
- Quante connessioni TCP può fare C in un secondo prima di portare tutte le porte disponibili nello stato TIME_WAIT? Ipotizzate che le porte disponibili per l'utente C siano quelle nell'intervallo 1024-5119 e che TIME_WAIT duri 60 secondi.
 - Le implementazioni di TCP derivate da quella di Berkeley consentono, solitamente, la riapertura di un socket che si trova nello stato TIME_WAIT anche prima che il temporizzatore TIME_WAIT scada, a patto che il numero di sequenza più elevato usato dalla vecchia incarnazione della connessione sia inferiore al valore di ISN usato dalla nuova incarnazione. Questa concessione risolve il problema dei vecchi dati che possono venire accettati come nuovi, però TIME_WAIT serve anche a gestire i segnali finali FIN in ritardo. Come dovrebbe fare tale implementazione per risolvere questo problema e ottenere comunque il rispetto del requisito di TCP secondo il quale un FIN inviato in qualsiasi momento, prima o durante un TIME_WAIT di una connessione, deve ricevere la medesima risposta?

22. Spiegate perché TIME_WAIT costituisce un problema più serio quando la chiusura è iniziata dal server anziché dal client. Descrivete una situazione in cui ciò potrebbe ragionevolmente accadere.
23. Qual è la giustificazione per l'aumento esponenziale del valore della temporizzazione proposto da Karn e Partridge? Perché, in particolare, un aumento lineare (o anche più lento) potrebbe essere migliore?
- ★ 24. L'algoritmo di Jacobson/Karels impone TimeOut in modo che sia 4 deviazioni medie al di sopra del valore medio. Ipotizzate che i tempi di round trip per il singolo pacchetto seguano una distribuzione statistica normale, per cui 4 deviazioni medie sono uguali a π deviazioni standard. Usando, ad esempio, tabelle statistiche, qual è la probabilità che un pacchetto arrivi in un tempo superiore al valore di TimeOut?
25. Supponete che una connessione TCP, con finestra di dimensione 1, perda ogni ulteriore pacchetto e che quelli che arrivano abbiano RTT di 1 secondo. Cosa accade? Cosa accade al valore di TimeOut? Considerate questi due casi:
 - a) Dopo che un pacchetto è stato ricevuto, riprendiamo dal punto in cui eravamo rimasti, inizializzando EstimatedRTT al valore che aveva prima della scadenza del temporizzatore e TimeOut al doppio di tale valore.
 - b) Dopo che un pacchetto è stato ricevuto, riprendiamo con TimeOut inizializzato all'ultimo valore del back-off esponenziale che era stato usato come intervallo di temporizzazione.
26. Nel meccanismo di ritrasmissione adattativa di TCP, supponete che EstimatedRTT assuma ad un certo punto il valore 4.0 e che i successivi valori misurati di RTT siano tutti 1.0. Quanto tempo occorre perché il valore di TimeOut, calcolato con l'algoritmo di Jacobson/Karels, scenda al di sotto di 4.0? Ipotizzate un valore iniziale plausibile per Deviation: quanto è sensibile la vostra risposta a questa scelta? Usate $\delta = 1/8$. (*questi calcoli, come quelli dei successivi tre esercizi, sono di facile realizzazione mediante un foglio elettronico*).
- ✓ 27. Nel meccanismo di ritrasmissione adattativa di TCP, supponete che EstimatedRTT assuma ad un certo punto il valore 90 e che i successivi valori misurati di RTT siano tutti 200. Quanto tempo occorre perché il valore di TimeOut, calcolato con l'algoritmo di Jacobson/Karels, scenda al di sotto di 300? Assumete per Deviation il valore iniziale 25; usate $\delta = 1/8$.
28. Supponete che il valore misurato di RTT per TCP sia 1.0, tranne che ogni N valori, nel qual caso RTT vale 4.0. Qual è, approssimativamente, il più grande valore di N che non provoca scadenza di temporizzatori in condizioni stazionarie (cioè per il quale il valore di TimeOut, calcolato con l'algoritmo di Jacobson/Karels, rimane superiore a 4.0)? Usate $\delta = 1/8$.
29. Supponete che TCP stia misurando valori di RTT uguali a 1.0 secondi, con una deviazione media di 0.1 secondi. Improvvistamente RTT balza a 5.0 secondi, senza alcuna deviazione. Confrontate i comportamenti dell'algoritmo originale e dell'algoritmo Jacobson/Karels nel calcolo di TimeOut. In particolare, quante scadenze di temporizzatori si verificano con ciascun algoritmo? Qual è il valore massimo di TimeOut che viene calcolato? Usate $\delta = 1/8$.
30. Supponete che, quando un segmento TCP viene inviato più di una volta, il valore di SampleRTT sia il tempo trascorso fra la trasmissione originaria e la conferma ACK, come in Figura 5.10(a). Dimostrate che se una connessione con finestra di un solo pacchetto perde ogni altro pacchetto (cioè ogni pacchetto viene trasmesso due volte).

- allora EstimatedRTT aumenta all'infinito. Ipotizzate che sia TimeOut = EstimatedRTT; entrambi gli algoritmi presentati nel testo impostano TimeOut a valori anche maggiori. Suggerimento: EstimatedRTT = EstimatedRTT + $\beta \times (\text{SampleRTT} - \text{EstimatedRTT})$.
31. Supponete che, quando un segmento TCP viene inviato più di una volta, il valore di SampleRTT sia il tempo trascorso fra la trasmissione più recente e la conferma ACK, come in Figura 5.10(b). Per rendere definito il calcolo, ipotizzate che sia TimeOut = $2 \times \text{EstimatedRTT}$. Delineate una situazione in cui nessun pacchetto venga perduto ma EstimatedRTT converga ad un terzo del valore reale di RTT, e tracciate un diagramma che illustri lo stato stazionario finale. Suggerimento: Iniziate con un improvviso balzo del vero valore di RTT, che lo porti appena al di sopra del valore determinato per TimeOut.
 32. Consultate il Request for Comments 793 per capire come è previsto che TCP risponda ad un FIN o ad un RST che arrivano con un numero di sequenza diverso da quello atteso, NextByteExpected. Considerate sia il caso in cui il numero di sequenza si trova all'interno della finestra di ricezione, sia il caso in cui non lo sia.
 33. Uno degli scopi di TIME_WAIT è quello di gestire il caso di un pacchetto di dati di una prima incarnazione di una connessione che arriva in grande ritardo e venga accettato come contenente dati validi per una seconda incarnazione.
 - a) Spiegate per quale motivo, perché ciò accada (in assenza di TIME_WAIT), gli host coinvolti dovrebbero scambiarsi diversi pacchetti in sequenza *dopo* che il pacchetto in ritardo è stato inviato, ma prima che venga consegnato.
 - b) Proponete uno scenario di rete che potrebbe tener conto di tale consegna ritardata.
 34. Proponete un'estensione di TCP mediante la quale un estremo della connessione può "passare la mano" ad un terzo host, cioè: se A era connesso a B e A trasferisce a C la propria connessione, allora C risulta essere connesso a B, mentre A non lo è più. Specificate i nuovi stati e le nuove transizioni che sono necessarie nel diagramma delle transizioni di stato di TCP, nonché qualsiasi nuovo tipo di pacchetto che si renda necessario. Potete ipotizzare che tutte le entità coinvolte siano in grado di gestire questa nuova opzione. In quale stato si porterebbe A subito dopo aver passato la mano?
 35. La caratteristica di TCP che consente l'apertura simultanea viene usata raramente.
 - a) Proponete una modifica di TCP in cui questa caratteristica diventi proibita. Indicate quali modifiche si renderebbero necessarie nel diagramma degli stati (e, se necessario, negli eventi di risposta non rappresentati nel diagramma).
 - b) Sarebbe ragionevole, per il protocollo TCP, impedire la chiusura simultanea?
 - c) Proponete una modifica di TCP nella quale segnali SYN scambiati simultaneamente da due host danno luogo a due connessioni separate. Indicate quali modifiche del diagramma degli stati sono richieste e anche quali modifiche si rendono necessarie all'intestazione. Notate anche che ciò significa che, ora, possono esistere più connessioni per una certa coppia $(\text{host}, \text{port})$. Potete anche leggere la parte "Discussion" a pagina 87 del Request for Comments 1122.
 36. Il protocollo TCP è fortemente simmetrico, ma il modello client/server non lo è. Considerate un protocollo asimmetrico simile a TCP, in cui soltanto alla parte server viene assegnato un numero di porta visibile agli strati applicativi. I socket sul lato client sarebbero semplici astrazioni da poter connettere a porte del server.
 - a) Proponete i dati per l'intestazione e la semantica di connessione che possano realizzare ciò. Cosa userete per sostituire il numero di porta del client?
 - b) Quale forma assume ora TIME_WAIT? Come sarebbe visto tutto questo tramite l'interfaccia di programmazione? Ipotizzate che, ora, un socket di tipo client possa

- riconnettersi ad una data porta del server un numero arbitrario di volte, risorse permettendo.
- Studiate il protocollo rsh/login. Come sarebbe messo in condizioni di non poter funzionare da questo nuovo protocollo di trasporto?
37. L'esercizio seguente tratta dello stato del protocollo TCP denominato FIN_WAIT_2 (vedere la Figura 5.7).
- Descrivete come un client possa lasciare indefinitamente nello stato FIN_WAIT_2 un server opportunamente predisposto. Quale caratteristica del protocollo sul lato server è necessaria in questa situazione?
 - Sperimentate questa caratteristica con alcuni server esistenti, scrivendo lo stub per un client oppure usando un client Telnet che sia in grado di connettersi a qualsiasi porta. Usate il programma netstat per verificare che il server si trovi nello stato FIN_WAIT_2.
38. Il documento *Request for Comments* 1122 afferma (relativamente al protocollo TCP):
- Un host PUÒ implementare una sequenza di chiusura di TCP "half-duplex", in modo che un'applicazione che abbia invocato CLOSE non possa continuare a leggere dati dalla connessione. Se tale host invoca CLOSE mentre sta eseguendo una lettura di dati in TCP, oppure se vengono ricevuti nuovi dati dopo che è stata invocata l'operazione CLOSE, DOVREBBE essere inviato un segnale RST per segnalare la perdita di dati.
- Delineate uno scenario i cui si verifichi tale situazione, in cui i dati inviati dall'host (*non all'host!*) in chiusura vengono perduti. Potete ipotizzare che l'host remoto, dopo aver ricevuto un RST, ignori tutti i dati ricevuti che si trovino ancora nei buffer, anche se non sono stati letti.
39. Quando TCP invia $\langle\text{SYN, SequenceNum} = x\rangle$ oppure $\langle\text{FIN, SequenceNum} = x\rangle$, il conseguente ACK ha Acknowledgement = $x + 1$, cioè sia SYN che FIN utilizzano un'unità nello spazio dei numeri di sequenza. È necessario? Se lo è, fornite un esempio di un'ambiguità che sorgerebbe se il corrispondente valore di Acknowledgement fosse x anziché $x + 1$; se non lo è, spiegate perché.
40. Cercate nel documento *Request for Comments* 793 il formato generico delle opzioni dell'intestazione TCP.
- Delineate una strategia che espanda lo spazio disponibile per le opzioni oltre l'attuale limite di 44 byte.
 - Suggerite un'estensione di TCP che consenta a chi invia un'opzione di specificare cosa dovrebbe fare il ricevente qualora non comprenda l'opzione. Elencate alcune azioni utili che il ricevente potrebbe intraprendere e cercate di fornire un esempio di applicazione per ciascuna di esse.
41. L'intestazione TCP non ha un campo BID, mentre CHAN lo ha. Come fa il protocollo TCP a proteggersi dalla situazione di riavvio improvviso dei calcolatori, che ha motivato la presenza del campo BID in CHAN? Perché CHAN non usa la medesima strategia?
42. Supponete di dover implementare l'accesso ad un *file system* remoto usando un protocollo RPC non affidabile che abbia la semantica "zero o più". Se un messaggio di risposta viene ricevuto, la semantica migliora, diventando "almeno una volta". Definiamo la funzione `read()` che restituisce il blocco N -esimo specificato, invece del successivo blocco della sequenza: in questo modo leggere una volta è come leggere due

- volte e la semantica "almeno una volta" funziona come quella "esattamente una volta".
- Per quali altre operazioni sul file system non esistono differenze fra la semantica "almeno una volta" e la semantica "esattamente una volta"? Considerate `open`, `create`, `write`, `seek`, `opendir`, `readdir`, `mkdir`, `delete` (chiamata anche `unlink`) e `rmdir`.
 - Per le restanti operazioni, quali possono essere modificate nella propria semantica per ottenere l'equivalenza tra "almeno una volta" e "esattamente una volta"? Quali operazioni sul file system non sono conciliabili con la semantica "almeno una volta"?
 - Supponete che la semantica dell'operazione `rmdir` preveda, ora, che la cartella (*directory*) indicata venga rimossa se esiste, e che non venga fatto nulla in caso contrario. Come potreste scrivere un programma che elimina cartelle, facendo distinzione tra questi due casi?
43. A volte il file system remoto "NFS", basato su RPC, viene considerato più lento del previsto rispetto alle prestazioni dell'operazione `write`. In NFS, la risposta RPC di un server ad una richiesta di tipo `write` proveniente da un client ha il significato di conferma che il dato sia stato fisicamente scritto sul disco del server, non soltanto inserito in una coda.
- Fornite spiegazioni sul collo di bottiglia che vi aspettate di osservare, anche con ampiezza di banda illimitata, nel caso in cui il client invii tutte le proprie richieste `write` attraverso un unico canale logico di CHAN, e spiegate perché l'utilizzo di un insieme di canali può essere d'aiuto. Suggerimento: dovete avere qualche idea in merito al funzionamento dei controllori di disco.
 - Supponete che la risposta del server significhi solamente che i dati sono stati inseriti nella coda di scrittura del disco. Spiegate come ciò possa provocare perdite di dati che non avverrebbero con un disco locale. Notate che la brusca interruzione dell'attività del sistema (*crash*) subito dopo l'inserimento dei dati in coda non rientra in queste cause, perché ciò provocherebbe perdita di dati anche in un disco locale.
 - In alternativa, il server potrebbe inviare immediatamente una risposta per confermare la richiesta di tipo `write`, inviando poi, mediante CHAN, una richiesta di conferma della scrittura fisica. Proponete una diversa semantica RPC in CHAN che ottenga il medesimo effetto, usando un'unica richiesta/risposta logica.
44. Sia il protocollo BLAST sia il protocollo CHAN hanno un campo `VID`.
- In quali circostanze questi possono avere lo stesso valore, per alcuni pacchetti in sequenza?
 - Nel testo, questi campi sono stati incrementati in modo sequenziale. Uno di questi campi potrebbe essere, invece, un numero casuale?
- ★ 45. Supponete che BLAST venga usato su una linea di collegamento con un tasso di perdita di pacchetti del 10%, con le perdite dovute ad eventi indipendenti. Tuttavia, l'ordine dei frammenti che arrivano non viene mutato rispetto all'ordine di spedizione. I messaggi sono composti di sei frammenti.
- Qual è, approssimativamente, la probabilità che scada il temporizzatore `LAST_FRAG`? Ipotizzate che ciò accada solamente quando viene perduto l'ultimo frammento.
 - Qual è la probabilità che l'ultimo frammento arrivi ma che qualcun altro non sia arrivato, provocando l'invio di un SRR?
 - Qual è la probabilità che non arrivi alcun frammento?
46. Considerate un client e un server che usano un meccanismo RPC che comprende CHAN.
- Delineate uno scenario in cui il server viene riavviato, una richiesta RPC viene

- invia due volte dal client e viene eseguita due volte dal server, con un unico ACK.
- Come potrebbe il client accorgersi che ciò è accaduto? Sarebbe sicuro dell'avvenimento?
47. Supponete che una richiesta RPC abbia la forma "incrementa del 10% il valore del campo X del blocco N del disco". Indicate un meccanismo da usare nell'esecuzione sul server che possa garantire che una richiesta in arrivo venga eseguita esattamente una volta, anche se il server viene riavviato improvvisamente nel corso dell'operazione. Ipotizzate che le scritture di singoli blocchi del disco siano complete oppure che il blocco non venga modificato affatto. Potete anche ipotizzare che siano disponibili alcuni blocchi con la funzione di "registro di azioni da annullare" (*undo log*). La vostra soluzione deve comprendere una descrizione del comportamento del server RPC nel momento in cui viene avviato (o riavviato).
48. Considerate un client SunRPC che invia una richiesta ad un server.
- In quali circostanze il client può esser certo che la sua richiesta sia stata eseguita esattamente una volta?
 - Supponete di voler aggiungere a SunRPC la semantica "al massimo una volta". Quali modifiche dovrebbero essere apportate? Spiegate perché l'aggiunta di uno o più campi alle intestazioni esistenti non sarebbe sufficiente.
49. Supponete che TCP debba essere usato come trasporto sottostante in un protocollo RPC: una connessione TCP trasporta, quindi, un flusso di richieste e di risposte. Quali sono le analogie, se ve ne sono, fra i campi di CHAN (CID, MID e BID) e i valori di Type (REQ, REP, ACK e PROBE)? Quali di questi dovrebbero essere forniti dal protocollo RPC soprastante? Esisterebbe qualche forma di conferma implicita?
50. Supponete che BLAST venga eseguito in una Ethernet a 10 Mbps, inviando messaggi di 32 K.
- Se i pacchetti Ethernet possono contenere 1500 byte di dati, e si usano intestazioni IP senza opzioni, oltre alle intestazioni di BLAST, quanti pacchetti Ethernet servono per ogni messaggio?
 - Calcolate il ritardo dovuto all'invio di un messaggio di 32 K in una rete Ethernet
 - diretta
 - scomposta in più spezzoni, come in (a), con un bridge
- Ignorate i ritardi di propagazione, le intestazioni, le collisioni e gli spazi fra i pacchetti.
51. Scrivete un programma di prova che usi l'interfaccia socket per inviare messaggi fra una coppia di stazioni di lavoro Unix connesse ad una LAN (ad esempio, Ethernet, ATM o FDDI). Usate questo programma di prova per eseguire i seguenti esperimenti.
- Misurare la latenza di round trip di TCP e di UDP per diverse dimensioni di messaggio (per esempio: 1 byte, 100 byte, 200 byte, ..., 1000 byte).
 - Misurare il throughput di TCP e UDP per messaggi di 1 KB, 2 KB, 3 KB, ..., 32 KB. Tracciate un grafico del throughput misurato in funzione della dimensione del messaggio.
 - Misurare il throughput di TCP inviando 1 MB di dati da un host all'altro. Ottenete lo scopo mediante un ciclo che invia messaggi di dimensione prefissata, ad esempio 1024 iterazioni di un ciclo che invia messaggi lunghi 1 KB. Ripetete l'esperimento con diverse dimensioni del messaggio e tracciate un grafico che riporti i risultati.

Controllo della congestione e allocazione di risorse

Problema Allocazione di risorse

Per il momento avete visto un numero di strati della gerarchia dei protocolli di rete sufficiente a capire come i dati possano essere trasmessi da un processo ad un altro attraverso reti eterogenee. Rivolgiamo ora la nostra attenzione ad un problema che si estende sull'intera pila di protocolli: come allocare risorse in modo efficace ed equo ad un insieme di utenti in competizione. Le risorse da condividere sono l'ampiezza di banda delle linee di collegamento e i buffer interni ai router o agli switch dove i pacchetti vengono accodati in attesa di essere trasmessi. I pacchetti *competono* in un router per l'utilizzo di una linea di collegamento, mentre si trovano in una coda ad aspettare il proprio turno per essere trasmessi sulla linea stessa. Quando troppi pacchetti competono per la medesima linea, la coda trabocca e alcuni pacchetti devono essere eliminati. Quando tale eliminazione diviene un fenomeno frequente, si dice che la rete è *congestionata*. La maggior parte delle reti ha un meccanismo di *controllo della congestione* per gestire tale situazione.

Il controllo della congestione e l'allocazione delle risorse sono due facce della stessa medaglia. Da una parte, se la rete svolge un ruolo attivo nell'allocation delle risorse (ad esempio, pianificando quale circuito virtuale possa utilizzare una certa linea fisica in un certo intervallo di tempo), allora la congestione può essere evitata, rendendo così inutile il controllo di congestione. Tuttavia, l'allocazione delle risorse di rete con tale precisione è un problema complesso, perché le risorse in questione sono distribuite nella rete: occorre pianificare l'utilizzo di più linee che collegano una sequenza di router. D'altra parte, è sempre possibile consentire alle sorrenti di pacchetti di inviare tutti i dati che vogliono, gestendo poi la congestione quando questa si verifica. Questo approccio è più semplice, ma può essere poco efficiente perché, prioritaria che la congestione possa essere controllata, la rete può aver eliminato molti pacchetti. Inoltre, il bisogno di allocazione di risorse fra utenti in competizione viene percepito in modo più stringente proprio quando la rete diviene congestionata, cioè quando le risorse sono divenute scarse rispetto alla richiesta. Esistono anche soluzioni intermedie, nelle quali vengono prese decisioni imperfette per l'allocation di risorse e la congestione può

comunque avvenire, rendendo quindi comunque necessario qualche meccanismo per la sua gestione. Non è veramente importante se queste soluzioni combinate vengano chiamate controllo della congestione o allocazione delle risorse: in un certo senso, si tratta di entrambe le cose.

Il controllo della congestione e l'allocazione delle risorse coinvolgono sia gli host sia gli apparati di rete, come i router. Negli apparati di rete, possono essere messe in atto diverse discipline di gestione delle code per controllare l'ordine in cui i pacchetti vengono trasmessi o eliminati. La disciplina di gestione della coda può anche tenere separato il traffico, cioè impedire che i pacchetti di un utente possano provocare problemi ai pacchetti di un altro utente. Negli host terminali, il meccanismo di controllo della congestione determina la velocità a cui le sorgenti possono inviare pacchetti, nel tentativo, in primo luogo, di evitare che la congestione si verifichi e, nel caso dovesse comunque accadere, di eliminarla.

Questo capitolo inizia con una panoramica del controllo di congestione e dell'allocazione di risorse, per poi discutere diverse discipline di gestione delle code che si possono implementare nei router interni alla rete, proseguendo con una descrizione dell'algoritmo di controllo della congestione che viene eseguito negli host dal protocollo TCP. La quarta sezione esplora varie tecniche che coinvolgono sia i router sia gli host, con l'obiettivo di evitare la congestione prima che questa diventi un problema. Infine, prendiamo in esame le necessità delle applicazioni di ricevere differenti livelli di allocazione delle risorse in una rete, descrivendo alcune modalità per la richiesta di risorse e alcune soluzioni mediante le quali le reti possono soddisfare queste richieste.

6.1 Problemi nell'allocazione delle risorse

L'allocazione delle risorse e il controllo della congestione sono problemi complessi e sono stati oggetto di molti studi fin dalle prime progettazioni di reti di calcolatori: sono tuttora aree di ricerca molto attive. Uno dei fattori che rende complessi questi problemi è il fatto che non sono localizzati in un unico strato della gerarchia di protocolli. L'allocazione delle risorse viene realizzata in parte dai router o dagli switch interni alla rete e in parte dal protocollo di trasporto che viene eseguito sugli host terminali. I sistemi che si trovano agli estremi di una comunicazione (*end systems*) usano protocolli di segnalazione per veicolare le proprie richieste di risorse ai nodi della rete, i quali rispondono con informazioni relative alla disponibilità di risorse. Uno degli obiettivi principali di questo capitolo è la definizione di un'infrastruttura (*framework*) all'interno della quale possano essere compresi questi meccanismi, oltre a fornire i dettagli più significativi di alcuni di questi meccanismi.

Prima di procedere oltre dobbiamo chiarire la terminologia che useremo. Con il termine "allocazione di risorse" intendiamo il processo mediante il quale gli apparati di una rete cercano di soddisfare le richieste delle applicazioni in merito a risorse di rete, richieste che vanno ad essere in competizione tra loro; le risorse di rete di cui parliamo sono, principalmente, l'ampiezza di banda delle linee e lo spazio di memorizzazione nei buffer di router e switch. Ovviamente, spesso non sarà possibile soddisfare tutte le richieste, per cui alcuni utenti o applicazioni possono ricevere meno risorse di rete di quelle che desiderano. Una parte del problema dell'allocazione di risorse consiste nel decidere quando, e a chi, dire "no". Usiamo il termine "controllo di congestione" per descrivere gli sforzi compiuti dai nodi della rete per prevenire le condizioni di sovraccarico o per reagire a tali situazioni. Poiché la congestione è generalmente una condizione negativa per tutti, la prima cosa da fare è smorzarla.

6.1 Problemi nell'allocazione delle risorse

o addirittura prevenirla: ciò si può fare semplicemente persuadendo alcuni host ad interrompere la trasmissione, migliorando così la situazione per tutti gli altri. Tuttavia, è molto comune che i meccanismi di controllo della congestione debbano utilizzare concetti di equità, cercando di suddividere i disagi fra tutti gli utenti, piuttosto che concentrare un grande disagio su pochi utenti. Di conseguenza, vedremo che molte strategie di controllo di congestione hanno, al proprio interno, una qualche nozione di allocazione di risorse.

È anche importante capire le differenze che esistono fra il controllo di flusso e il controllo di congestione. Il controllo di flusso, come abbiamo visto nella Sezione 2.5, si occupa di impedire ad una sorgente veloce di sovraccaricare un ricevitore lento. Il controllo di congestione, al contrario, si occupa di impedire che un insieme di sorgenti inviano *nella rete* troppi dati, in seguito alla scarsità di risorse in qualche punto della rete stessa. Questi due concetti vengono spesso confusi e, come vedremo, condividono anche alcune strategie.

6.1.1 Modello della rete

Cominciamo definendo tre caratteristiche salienti dell'architettura di rete: si tratta, per la maggior parte, di un riassunto del materiale presentato nei capitoli precedenti che ha interesse in relazione al problema dell'allocazione delle risorse.

Rete a commutazione di pacchetto

Consideriamo l'allocazione di risorse in una rete (o una internet) a commutazione di pacchetto, composta da più linee di collegamento e switch (o router). Poiché molte delle strategie descritte in questo capitolo furono progettate per Internet, e quindi originariamente definite in termini di router piuttosto che di switch, usiamo nella discussione il termine "router", anche se il problema, in una rete o in una internetwork, è sostanzialmente lo stesso.

In tale situazione, può darsi che una sorgente sia collegata ad una linea con capacità più che sufficiente per poter inviare un pacchetto, ma i suoi pacchetti possono attraversare, da qualche parte all'interno della rete, una linea di collegamento utilizzata da molte altre sorgenti di traffico. La Figura 6.1 illustra questa situazione: due linee ad alta velocità che forniscono traffico ad una linea a bassa velocità. Questa situazione contrasta con le reti ad accesso condiviso, come Ethernet e reti token ring, nelle quali la sorgente può osservare direttamente il traffico nella rete e decidere, di conseguenza, se inviare o meno un pacchetto: abbiamo già visto gli algoritmi utilizzati per allocare banda in reti ad accesso condiviso (Capitolo 2). Questi algoritmi di controllo dell'accesso sono, in un certo senso, analoghi agli algoritmi di controllo della congestione in una rete commutata.

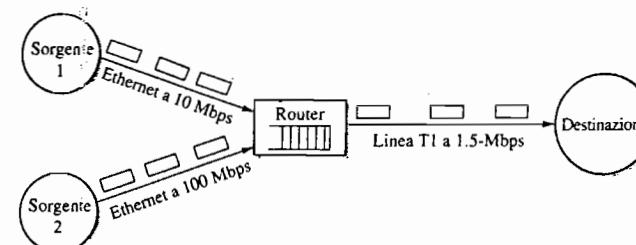


Figura 6.1 Congestione in una rete a commutazione di pacchetto.

Note che il controllo di congestione non è l'instradamento. Anche se è vero che al ramo corrispondente ad una linea congestionata si potrebbe assegnare un peso elevato nel protocollo di propagazione dei percorsi e, di conseguenza, i router instraderebbero i pacchetti aggirando tale linea, "aggirare" una linea congestionata non risolve il problema della congestione. Per capire meglio questa affermazione, non abbiamo bisogno di esaminare situazioni più complesse di quella rappresentata in Figura 6.1, dove tutto il traffico deve fluire attraverso il medesimo router per raggiungere la destinazione. Anche se si tratta di un esempio estremo, avere un router che non si possa aggirare è una situazione comune¹: tale router può divenire congestionato e la strategia di instradamento non può far niente. Questo tipo di router congestionato viene a volte chiamato *collo di bottiglia*.

Flussi privi di connessione

Per la maggior parte della nostra discussione ipotizzeremo che la rete sia sostanzialmente priva di connessioni, con i servizi orientati alla connessione che vengono realizzati dal protocollo di trasporto in esecuzione sugli host terminali (il termine "sostanzialmente" verrà spiegato meglio fra poco). Si tratta, in effetti, del modello di Internet, dove il protocollo IP fornisce un servizio di consegna di datagrammi privo di connessione e il protocollo TCP implementa l'astrazione della connessione di trasporto tra due estremi (*end-to-end*). Note che questa ipotesi esclude le prime reti, come X.25, nelle quali un insieme di router si prendeva carico della gestione dell'astrazione di circuito virtuale (si veda la Sezione 3.1.2). Quando viene creato un circuito, una rete di questo tipo è attraversata da un messaggio di instaurazione della connessione che riserva, in ciascun router, un insieme di buffer da utilizzare per la connessione stessa, fornendo così una forma di controllo della congestione: si stabilisce una connessione soltanto se in ciascun router può essere assegnato ad essa uno spazio sufficiente nei buffer. Il principale svantaggio di questo approccio sta nel fatto che porta ad una sottoutilizzazione delle risorse: i buffer riservati ad un particolare circuito non sono disponibili per altro traffico, nemmeno nel caso in cui non siano utilizzati, in un determinato momento, da quel circuito. L'attenzione di questo capitolo è focalizzata sugli approcci di allocazione delle risorse che si applicano ad una internetwork, per cui ci concentriamo principalmente sulle reti che offrono servizi privi di connessione. L'unica eccezione a questo sarà la nostra discussione sulla qualità di servizio della tecnologia ATM, nella Sezione 6.5.4, che fornisce un interessante quadro di confronto per il modello di Internet.

Dobbiamo ora specificare meglio il termine "privi di connessione" (*connectionless*), perché la nostra classificazione delle reti in "prive di connessione" e "orientate alla connessione" è un po' troppo restrittiva: esiste una zona grigia intermedia. In particolare, l'implicazione per cui in una rete priva di connessione tutti i datagrammi sono completamente indipendenti l'uno dall'altro è troppo rigida. Certamente i datagrammi vengono commutati indipendentemente, ma solitamente un flusso di datagrammi fra una particolare coppia di host segue lo stesso percorso attraverso un insieme di router. Questa idea di un *flusso* come di una sequenza di pacchetti inviati da una sorgente ad una destinazione e che seguono lo stesso percorso all'interno di una rete è un'importante astrazione nel contesto dell'allocatione di risorse, ed è quella che useremo in questo capitolo.

¹ Vale anche la pena di notare che la complessità dell'instradamento in Internet è tale che la cosa migliore che si può sperare di ottenere è, semplicemente, un percorso privo di cicli e ragionevolmente diretto: aggirare la congestione sarebbe la ciliegina sulla torta.

6.1 Problemi nell'allocazione delle risorse

Uno dei punti di forza dell'astrazione di flusso è che i flussi possono essere definiti con diversi livelli di dettaglio. Ad esempio, un flusso può essere di tipo *host-to-host* (cioè essere un insieme di pacchetti che hanno gli stessi indirizzi di host come sorgente e destinazione) oppure *process-to-process* (cioè avere le stesse coppie host/porta come sorgente e destinazione). Nel secondo caso, un flusso coincide sostanzialmente con un canale, e abbiamo usato questo termine in tutto il libro. Il motivo per cui presentiamo qui un nuovo termine sta nel fatto che un flusso è visibile ai router che si trovano in una rete, mentre un canale è un'astrazione tra host terminali, cioè è un'astrazione di tipo *end-to-end*. La Figura 6.2 mostra alcuni flussi che attraversano una sequenza di router.

Dato che più pacchetti, fra loro correlati, fuiscono attraverso ciascun router, a volte ha senso gestire informazioni di stato relative a ciascun flusso, che possono essere utilizzate per prendere decisioni in merito all'allocazione di risorse per i pacchetti che appartengono al flusso stesso. Si parla, in questo caso, di *stato soft*: la principale differenza tra lo stato *soft* e lo stato "hard" risiede nel fatto che lo stato *soft* non ha bisogno di essere esplicitamente creato e rimosso mediante un'attività di segnalazione. Lo stato *soft* rappresenta le situazioni intermedie fra una rete assolutamente priva di connessione, che non gestisce *alcuna* informazione di stato nei router, e una rete completamente orientata alla connessione, i cui router memorizzano informazioni di stato "hard". In generale, il funzionamento corretto di una rete non dipende dal fatto che lo stato *soft* sia presente (ogni pacchetto viene instradato correttamente anche in assenza di tali informazioni di stato), ma, quando un pacchetto appartiene ad un flusso per il quale un router ha memorizzato informazioni di stato *soft*, allora la gestione del pacchetto può essere fatta in modo migliore.

Note che un flusso può essere definito in modo implicito o esplicito. Nel primo caso, ciascun router osserva i pacchetti che viaggiano fra la stessa coppia di sorgente/destinazione, ispezionando gli indirizzi presenti nell'intestazione, e, per il controllo di congestione, considera che tali pacchetti appartengano al medesimo flusso. Nel secondo caso, la sorgente invia attraverso la rete un messaggio di instaurazione del flusso, dichiarando che sta per iniziare un flusso di pacchetti. Mentre si potrebbe ritenere che i flussi creati esplicitamente non siano diversi da una connessione all'interno di una rete orientata alla connessione, richiamiamo l'attenzione sul fatto che, ancorché definito esplicitamente, un flusso non implica alcuna semantica fra i nodi terminali e, in particolare, non implica la consegna affidabile e in se-

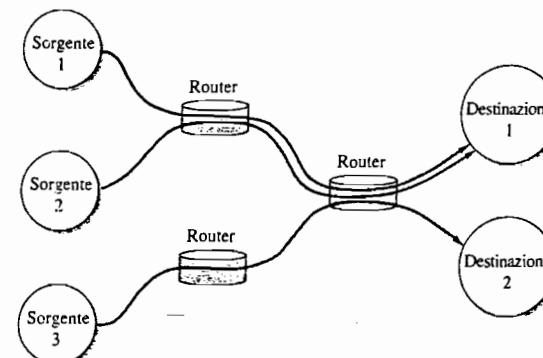


Figura 6.2 Più flussi che attraversano un insieme di router.

quenza, che è tipica di un circuito virtuale: il flusso esiste al solo scopo dell'allocazione di risorse. In questo capitolo vedremo esempi di flussi impliciti ed esplicativi.

Modello del servizio

Nella prima parte di questo capitolo ci siamo concentrati sulle strategie che si basano sull'ipotesi del modello di servizio *best-effort* di Internet. Con il servizio best-effort, tutti i pacchetti sono gestiti allo stesso modo e agli host terminali non viene dato alcuno strumento per chiedere alla rete di fornire certe garanzie ad uno dei propri flussi. La Sezione 6.5 avrà come argomento la definizione di un modello di servizio che fornisce alcune forme di garanzie: ad esempio, l'ampiezza di banda garantita per le necessità di un flusso video. Un modello di servizio di quel tipo fornisce diverse *qualità di servizio* (QoS, *quality of service*). Come vedrete, esiste effettivamente un ampio spettro di possibilità, da un modello di servizio puramente best-effort, fino ad arrivare ad un modello in cui i singoli flussi ricevono garanzie quantitative sulla propria QoS. Una delle sfide più interessanti consiste nel definire un modello di servizio che soddisfi le necessità di un ampio spettro di applicazioni, comprese le applicazioni che verranno ideate in futuro.

6.1.2 Tassonomia

Esistendo innumerevoli dettagli per i quali le strategie di allocazione di risorse differiscono l'una dall'altra, la definizione di una tassonomia completa è un proposito arduo. Per il momento descriviamo tre dimensioni lungo le quali possono essere caratterizzate le strategie di allocazione delle risorse, lasciando alcune più sottili distinzioni alle discussioni che saranno svolte nel capitolo.

Focalizzata sui router o focalizzata sugli host

Le strategie di allocazione di risorse possono essere classificate in due grandi categorie: quelle che risolvono il problema all'interno della rete (cioè nei router o negli switch, *router-centric*) e quelle che lo risolvono ai confini della rete (cioè negli host, probabilmente nel protocollo di trasporto, *host-centric*). Dato che, solitamente, sia i router interni alla rete sia gli host ai suoi confini partecipano all'allocazione delle risorse, il vero problema riguarda chi si faccia carico della responsabilità maggiore.

In un progetto focalizzato sui router, ogni router si assume la responsabilità di decidere quando i pacchetti vengano inoltrati e di selezionare quali pacchetti vengano eliminati, oltre ad informare gli host che stanno generando il traffico di rete in merito a quanti pacchetti sia loro concesso di spedire. In un progetto focalizzato sugli host, gli host terminali osservano le condizioni della rete (ad esempio, quanti pacchetti stanno attraversando con successo la rete) e modificano di conseguenza il proprio comportamento. Notate che queste due categorie non sono mutuamente esclusive. Ad esempio, una rete che assegna ai router il compito primario di gestire la congestione, si aspetta comunque che gli host rispettino i messaggi di informazioni inviati dai router, mentre i router di una rete che usa un controllo di congestione focalizzato sugli host hanno comunque delle politiche, ancorché semplici, per decidere quali pacchetti eliminare qualora le proprie code trabocchino.

Basata sulla prenotazione o basata sui segnali ricevuti

Un secondo modo per catalogare le strategie di allocazione delle risorse si basa sul fatto che usino un meccanismo di *prenotazione* (*reservation based*) o che si basino sui *segnali ricevuti*

6.1 Problemi nell'allocazione delle risorse

(*feedback based*). In un sistema basato sulla prenotazione, un host terminale richiede alla rete una certa capacità nel momento in cui viene creato un flusso; in conseguenza di ciò, ciascun router alloca risorse (buffer e/o porzione dell'ampiezza di banda della linea) sufficienti per soddisfare tale richiesta. Se la richiesta non può essere soddisfatta da qualche router, perché nel farlo userebbe più risorse di quelle disponibili, il flusso viene rifiutato dal router. Ciò è analogo alla situazione in cui si riceve il segnale di linea occupata sollevando la cornetta del telefono. In un approccio basato sui segnali ricevuti, un host terminale inizia ad inviare dati senza prenotare anticipatamente alcuna capacità nella rete; in seguito, modifica la propria velocità di trasmissione in base ai segnali (*feedback*) che riceve. Tale feedback può essere *esplicito* (ad esempio, un router congestionato invia all'host un messaggio del tipo "per favore rallenta") oppure *implicito* (ad esempio, l'host modifica la propria velocità di trasmissione in base al comportamento esternamente osservabile della rete, come la perdita di pacchetti). Notate che un sistema basato sulla prenotazione implica sempre una strategia di allocazione delle risorse focalizzata sui router, perché ogni router ha la responsabilità di tenere traccia di quale frazione della propria capacità sia già stata riservata e di assicurarsi che ogni host rimanga entro i limiti delle proprie prenotazioni. Se un host tenta di inviare dati più velocemente di quanto ha dichiarato di voler fare all'atto della prenotazione delle risorse, allora i pacchetti di tale host sono buoni candidati per l'eliminazione, nel caso in cui il router diventi congestionato. D'altra parte, un sistema basato sul feedback è compatibile sia con una strategia focalizzata sui router sia con una focalizzata sugli host. Tipicamente, se il feedback è esplicito il router è coinvolto, in qualche modo, nello schema di allocazione delle risorse, mentre se il feedback è implicito la maggior parte degli oneri gravano sugli host: i router, qualora divengano congestionati, eliminano pacchetti in silenzio.

Basata su finestra o basata sulla velocità

Un terzo modo per catalogare le strategie di allocazione delle risorse si basa sull'utilizzo di *finestre* o di *velocità*. Come abbiamo detto prima, questa è un'area in cui si usano meccanismi e terminologia simili per il controllo di flusso e per il controllo di congestione: entrambi i problemi necessitano di un modo per comunicare alla sorgente la quantità di dati che può trasmettere. Ciò si può fare, in generale, in due modi diversi: con una *finestra* e con una *velocità*. Abbiamo già visto protocolli di trasporto basati su finestra, come TCP, in cui il ricevitore comunica al mittente la dimensione di una finestra, che corrisponde alla quantità di spazio disponibile nel buffer del ricevitore e pone un limite alla quantità di dati che può essere trasmessa dalla sorgente: in sostanza, fornisce il supporto per il controllo di flusso. Una strategia simile, la pubblicizzazione di una finestra (*window advertisement*) può essere utilizzata in una rete per riservare spazio nei buffer, cioè per fornire supporto all'allocazione di risorse: così si comportano le reti X.25.

Si può controllare il comportamento di una sorgente anche usando un valore di velocità, cioè indicando quanti bit al secondo possono essere assorbiti dal ricevitore o dalla rete. Anche se in questo libro non abbiamo ancora studiato nessun protocollo di trasporto basato sulla velocità (lo faremo nel Capitolo 9), possiamo immaginare un protocollo di questo tipo utilizzato per la trasmissione di flussi video: il ricevitore segnala di poter elaborare immagini (*frame*) video alla velocità di 1 Mbps e la sorgente si adeguà a tale velocità. Come vedremo più avanti in questo capitolo, la caratterizzazione di un flusso in base alla velocità è una scelta naturale in un sistema basato sulla prenotazione che consenta diverse *qualità di servizio*: la sorgente effettua la prenotazione di un certo numero di bit al secondo e ciascun router lungo il percorso determina se è in grado di sostenere tale velocità, in base agli altri flussi che deve gestire.

Riassunto della tassonomia per l'allocazione di risorse

La catalogazione delle strategie per l'allocazione di risorse mediante due diversi punti lungo tre dimensioni, come abbiamo appena fatto, sembrerebbe suggerire otto strategie possibili: anche se sono tutte certamente possibili, notiamo che, in pratica, sembrano generalmente prevalere due strategie, strettamente connesse al modello di servizio della rete sottostante. Da un lato, un modello di servizio di tipo best-effort implica l'utilizzo di una strategia basata sul feedback, perché tale modello non consente agli utenti di effettuare prenotazioni di capacità di rete. Ciò, a sua volta, significa che la maggior parte della responsabilità del controllo di congestione è in carico agli host terminali, eventualmente con un po' di assistenza da parte dei router. Questa è la strategia adottata in Internet e sarà oggetto delle Sezioni 6.3 e 6.4.

D'altro canto, un modello basato sulla qualità di servizio implica, quasi sempre, una qualche forma di prenotazione di risorse². Il supporto per queste prenotazioni richiede, probabilmente, un elevato coinvolgimento dei router, per, ad esempio, accodare diversamente i pacchetti in base al livello di prenotazione delle risorse che hanno richiesto. Inoltre, risulta naturale esprimere tali prenotazioni in termini di velocità, perché le finestre sono correlate soltanto indirettamente alla quantità di banda di cui necessita un utente. Parleremo di questo argomento nella Sezione 6.5.

6.1.3 Criteri di valutazione

L'ultimo problema riguarda la possibilità di valutare se un meccanismo di allocazione delle risorse sia valido oppure no. Ricordate che nell'enunciazione del problema all'inizio di questo capitolo ci siamo chiesti quanto *efficacemente* ed *equamente* una rete allochi le proprie risorse. Ciò suggerisce almeno due vaste categorie di parametri in base ai quali si possono valutare gli schemi di allocazione delle risorse. Le consideriamo ora, una per volta.

Allocazione di risorse efficace

Un buon punto di partenza per la valutazione dell'efficacia di uno schema di allocazione di risorse è l'esame delle due metriche principali relative alle reti di calcolatori: il throughput e il ritardo. Chiaramente, vogliamo avere il massimo throughput e il minimo ritardo possibili. Anche se, superficialmente, potrebbe sembrare che all'aumentare del throughput si riduca il ritardo, ciò non è sempre vero. Un modo sicuramente funzionale perché un algoritmo di allocazione delle risorse possa aumentare il throughput consiste nel consentire l'invio nella rete del massimo numero possibile di pacchetti, andando verso un'utilizzazione al 100% di tutte le linee di collegamento, per evitare che una linea diventi inattiva, perché una linea inattiva si ripercuote negativamente sul throughput. Il problema di questa strategia è che l'aumento del numero di pacchetti nella rete provoca l'aumento della lunghezza delle code di ogni router: code più lunghe, a loro volta, provocano maggiori ritardi per l'attraversamento della rete da parte dei pacchetti. Per descrivere questa relazione, alcuni progettisti di reti hanno proposto l'utilizzo del rapporto tra il throughput e il ritardo come metrica per la valutazione dell'efficacia di uno schema di allocazione delle risorse. Questo rapporto viene a volte chiamato *potenza* (*power*) della rete³:

² Come vedremo nella Sezione 6.5, la prenotazione di risorse può essere effettuata da gestori di rete anziché dagli host.

³ La vera definizione è $\text{Power} = \text{Throughput}^\alpha / \text{Delay}$, dove $0 < \alpha < 1$; $\alpha = 1$ rende massima la potenza in corrispondenza del ginocchio della curva del ritardo. Il throughput viene misurato in unità di dati (ad esempio, in bit) al secondo; il ritardo in secondi.

6.1 Problemi nell'allocazione delle risorse

$$\text{Power} = \text{Throughput}/\text{Delay}$$

Notate come non sia così ovvio che la potenza rappresenti la metrica corretta per valutare l'efficacia dell'allocazione di risorse. Da un lato, la teoria che sta dietro alla potenza è basata su una rete di accodamento⁴ M/M/1, che ipotizza code di lunghezza infinita, mentre le reti reali hanno buffer di dimensioni finite e ogni tanto sono costrette ad eliminare pacchetti. Di converso, la potenza viene tipicamente definita in relazione ad una singola connessione (un flusso) e non è chiaro come si estenda a più connessioni che competono tra loro. Nonostante queste limitazioni, piuttosto severe, nessuna alternativa ha acquisito la stessa popolarità, per cui si continua ad usare la potenza.

L'obiettivo è quello di rendere massimo questo rapporto, che è funzione del carico della rete, che, a sua volta, viene impostato dal meccanismo di allocazione delle risorse. La Figura 6.3 presenta un grafico rappresentativo della potenza, dove, idealmente, il meccanismo di allocazione delle risorse opererebbe vicino al picco di questa curva. Alla sinistra del picco, la strategia si comporta in modo troppo cauto, cioè non consente di inviare pacchetti a sufficienza per tenere impegnate le linee. Alla destra del picco, si consente di inviare nella rete un numero troppo elevato di pacchetti e gli aumenti del ritardo, dovuti alla permanenza nelle code, iniziano a dominare qualsiasi piccolo guadagno nel valore del throughput.

È interessante notare come questa curva della potenza assomigli molto alla curva del throughput di sistema in un calcolatore multiprogrammato. Il throughput di sistema aumenta all'aumentare del numero di programmi in esecuzione nel sistema, finché raggiunge un punto in cui ci sono talmente tanti programmi in esecuzione che il sistema inizia a boccheggiare, trascorrendo tutto il proprio tempo cambiando posizione a pagine di memoria (*swapping*), e il valore del throughput inizia a diminuire.

Come vedremo nelle successive sezioni di questo capitolo, molti schemi di controllo della congestione sono in grado di controllare il carico soltanto in modo molto brutale, cioè non è possibile "aprire un po' il rubinetto" e consentire un piccolo aumento di nuovi pacchetti inviati nella rete. Di conseguenza, i progettisti di reti devono preoccuparsi anche di cosa

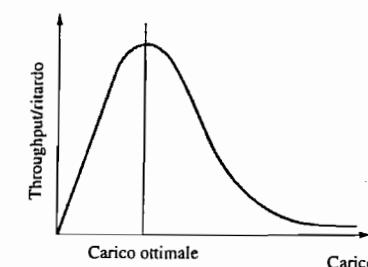


Figura 6.3 Rapporto tra il throughput e il ritardo in funzione del carico.

⁴ Dato che questo non è un libro di teoria delle code, diamo solamente questa sintetica descrizione di una coda M/M/1. Il numero 1 significa che esiste un unico server, mentre le lettere M significano che è "Markoviana", cioè esponenziale, sia la distribuzione degli arrivi dei pacchetti, sia quella dei tempi di servizio.

accade quando il sistema si trova ad operare in condizioni di carico molto elevato, cioè nella parte più a destra della curva di Figura 6.3. Idealmente vorremmo evitare la situazione in cui il throughput di sistema va a zero perché il sistema boccheggia. Nella terminologia delle reti, vogliamo un sistema che sia *stabile*, dove i pacchetti continuano ad attraversare la rete anche quando essa si trova ad operare in condizioni di carico molto elevato. Se una strategia non è stabile, la rete può andare incontro ad un *collasso per congestione*.

Allocazione di risorse equa

L'utilizzo efficace delle risorse di rete non è l'unico criterio per valutare uno schema di allocazione di risorse, dobbiamo considerarne anche l'equità, anche se ci impantaneremmo rapidamente se cercassimo di definire cosa sia, esattamente, l'equità nell'allocazione delle risorse. Ad esempio, uno schema di allocazione delle risorse basato sulla prenotazione fornisce un modo esplicito per creare una controllata mancanza di equità: si può usare la prenotazione per consentire ad un flusso video di ricevere 1 Mbps tramite una particolare linea, mentre un trasferimento di file riceve soltanto 10 Kbps sulla stessa linea.

In assenza di esplicite informazioni contrarie, quando diversi flussi condividono una certa linea, vorremmo che ogni flusso ricevesse la stessa frazione di banda. Questa definizione sottintende l'ipotesi che un' *equa* frazione di banda significhi un' *eguale* frazione di banda, ma, anche in assenza di prenotazioni, frazioni eguali possono non equivalere a frazioni eque. Dovremmo considerare anche la lunghezza dei percorsi che stiamo confrontando? Ad esempio, come mostrato in Figura 6.4, cos'è l'equità quando un flusso con quattro salti viene confrontato con tre flussi aventi un solo salto?

Ipotizzando che equità implichi egualianza e che tutti i percorsi abbiano la stessa lunghezza, Raj Jain ha proposto una metrica che può essere utilizzata per quantificare l'equità di una strategia di controllo della congestione. L'indice di equità di Jain è così definito: dato un insieme di throughput di flussi (x_1, x_2, \dots, x_n), misurati in unità coerenti (ad esempio, bit/secondo), la seguente funzione assegna ai flussi un indice di equità:

$$f(x_1, x_2, \dots, x_n) = \frac{\left(\sum_{i=1}^n x_i\right)^2}{n \sum_{i=1}^n x_i^2}$$

L'indice di equità è sempre un numero compreso tra 0 e 1, con il valore 1 che rappresenta la massima equità. Per capire l'intuizione che sta alla base di questa metrica, considerate il caso in cui tutti gli n flussi ricevono un throughput di 1 unità di dati al secondo. Possiamo vedere che, in questo caso, l'indice di equità vale

$$\frac{n^2}{n \times n} = 1$$

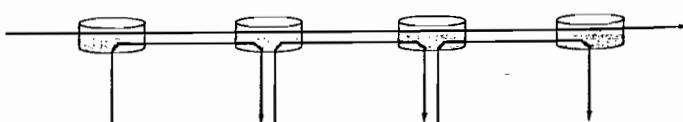


Figura 6.4 Un flusso con quattro salti in competizione con tre flussi aventi un solo salto.

6.2 Politiche di gestione delle code

Supponete ora che un flusso riceva un throughput uguale a $1 + \Delta$: l'indice di equità diventa

$$\frac{((n-1)+1+\Delta)^2}{n(n-1+(1+\Delta)^2)} = \frac{n^2 + 2n\Delta + \Delta^2}{n^2 + 2n\Delta + n\Delta^2}$$

Noteate che il denominatore supera il numeratore per la quantità $(n-1)\Delta^2$. Di conseguenza, indipendentemente dal fatto che il flusso diverso dagli altri abbia un valore di throughput maggiore o minore di quello di tutti gli altri (cioè un valore di Δ positivo o negativo), l'indice di equità è divenuto minore di uno. Un altro caso semplice da considerare è quello in cui soltanto k degli n flussi ricevono lo stesso throughput, mentre i rimanenti $n-k$ ricevono un throughput uguale a zero: in questo caso l'indice di equità scende al valore k/n .

6.2 Politiche di gestione delle code

Indipendentemente da quanto sia semplice o complesso il meccanismo di allocazione delle risorse, ogni router deve implementare una politica di gestione delle code che stabilisca come vengono memorizzati nei buffer i pacchetti mentre attendono di essere trasmessi. Si può pensare che l'algoritmo di gestione delle code allochi sia banda (in base alla quale i pacchetti vengono trasmessi) sia spazio nei buffer (in base al quale i pacchetti vengono eliminati). Inoltre, influenza anche, in modo diretto, la latenza di un pacchetto, in quanto determina il tempo di attesa di un pacchetto prima della propria trasmissione. Questa sezione presenta due algoritmi di gestione delle code assai comuni (oltre ad esaminare alcune varianti che sono state proposte): FIFO (*first-in, first-out*, "il primo a entrare è il primo a uscire") e gestione equa delle code (FQ, *fair queueing*).

6.2.1 FIFO

L'idea delle code FIFO, chiamate anche "il primo che arriva è il primo ad essere servito" (FCFS, *first-come, first-served*), è semplice: il primo pacchetto che arriva ad un router è il primo pacchetto ad essere trasmesso da quel router. Questa strategia è illustrata in Figura 6.5(a), che mostra una coda FIFO avente "spazi" (slot) per contenere fino a otto pacchetti. Dato che la quantità di spazio di memorizzazione (buffer) in ciascun router è limitato, se un pacchetto arriva quando la coda (cioè lo spazio di memorizzazione) è piena, allora il router ignora il pacchetto, come evidenziato in Figura 6.5(b), senza considerare a quale flusso appartenga il pacchetto o quanto importante sia il pacchetto stesso. In questi casi si parla anche di *tail drop* ("eliminazione terminale"), perché vengono eliminati i pacchetti che arrivano all'estremo terminale (tail) della coda.

Noteate che l'eliminazione terminale e la politica FIFO sono due concetti separabili. FIFO è una *disciplina di pianificazione*, che determina l'ordine in cui vengono trasmessi i pacchetti, mentre l'eliminazione terminale (*tail drop*) è una *strategia di eliminazione*, che determina quali pacchetti vengono eliminati. Dal momento che FIFO e tail drop sono, rispettivamente, la disciplina di pianificazione e la strategia di eliminazione più semplici, sono spesso viste insieme: l'implementazione più semplice delle code. Sfortunatamente, spesso si parla di questo insieme come di "gestione di code con modalità FIFO", mentre si dovrebbe parlare, più precisamente, di "FIFO con eliminazione terminale". La Sezione 6.4 fornisce un esempio di una diversa strategia di eliminazione, che usa un algoritmo più complesso di quello "c'è un

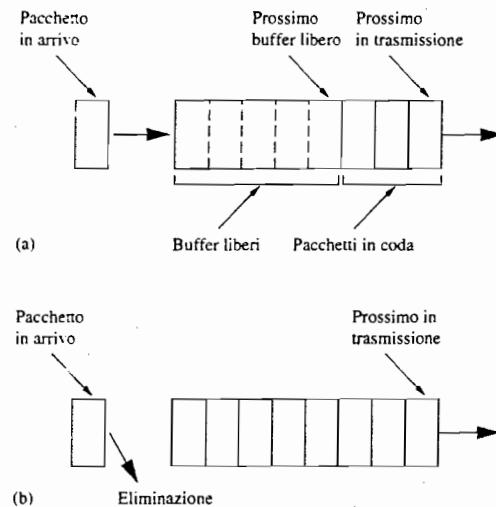


Figura 6.5 (a) Coda FIFO; (b) tail drop in una coda FIFO.

buffer libero?" per decidere quando eliminare pacchetti. Tale strategia di eliminazione può essere usata insieme a FIFO, o con discipline di pianificazione più complesse.

FIFO con *tail drop*, essendo il più semplice fra tutti gli algoritmi di gestione delle code, è quello più diffusamente utilizzato nei router di Internet nel momento in cui scriviamo. Questo semplice approccio al problema della gestione delle code sposta verso i confini della rete tutta la responsabilità per il controllo della congestione e l'allocazione delle risorse. Di conseguenza, la forma di controllo di congestione più utilizzata oggi in Internet ipotizza che non vi sia alcun aiuto da parte dei router: il protocollo TCP si assume la responsabilità di individuare e di gestire la congestione, nel modo che esamineremo nella Sezione 6.3.

Le code a gestione prioritaria sono una semplice variante del meccanismo basilare delle code FIFO. L'idea consiste nel contrassegnare ciascun pacchetto con una priorità e il contrassegno potrebbe trovar posto, ad esempio, nel campo IP denominato *Type of Service* (TOS). I router, poi, implementano più code FIFO, una per ciascuna classe di priorità: un router trasmette sempre pacchetti estratti dalla coda a priorità più elevata, se tale coda non è vuota, prima di passare a prendere in esame la cosa con la priorità successiva. All'interno di ciascuna classe di priorità, i pacchetti continuano ad essere gestiti in modalità FIFO. Questa idea costituisce una piccola deviazione dal modello best-effort, ma non fa molti passi avanti nel fornire garanzie ad alcuna particolare classe di priorità: semplicemente, consente ai pacchetti con priorità più elevate di balzare nelle prime posizioni della coda di trasmissione.

Il problema delle code prioritarie, ovviamente, sta nel fatto che la coda a priorità più elevata può bloccare tutte le altre code, cioè le code a priorità inferiore non vengono mai servite finché c'è almeno un pacchetto presente nella coda a più elevata priorità. Per rendere praticabile questo approccio, deve esistere un limite rigido per la quantità di traffico ad elevata priorità che viene inserito nella coda. Dovrebbe essere assolutamente evidente che non

possiamo consentire ai singoli utenti di assegnare in modo incontrollato la priorità ai propri pacchetti: dobbiamo proibire loro di farlo oppure dobbiamo poter esercitare una qualche forma di "pressione" sugli utenti stessi. Un modo ovvio consiste nella leva economica: la rete potrebbe far pagare una cifra più elevata per la consegna di pacchetti ad elevata priorità, rispetto al costo dei pacchetti di priorità inferiore. Tuttavia, in un ambiente decentralizzato come Internet, realizzare uno schema di questo tipo è veramente una sfida.

In Internet, vengono usate le code prioritarie per proteggere i pacchetti più importanti, tipicamente gli aggiornamenti di stradamento che sono necessari alla stabilizzazione delle tabelle di stradamento dopo una modifica alla topologia della rete. Spesso esiste una coda speciale per tali pacchetti, che si possono identificare mediante il campo TOS nell'intestazione IP: si tratta, infatti, di un caso semplificato dell'idea di "Differentiated Services" (*servizi differenziati*), che sarà argomento della Sezione 6.5.3.

6.2.2 Fair queueing

Il problema principale della gestione FIFO delle code sta nel fatto che non fa distinzione tra diverse sorgenti di traffico: usando la terminologia presentata nella sezione precedente, non separa i pacchetti in base al flusso a cui appartengono. Questa situazione pone problemi a due livelli diversi. Innanzitutto, non è chiaro se un algoritmo di controllo della congestione che venga implementato totalmente nella sorgente possa effettuare un controllo adeguato con un aiuto così minimo da parte dei router. Sospenderemo il giudizio su questo aspetto fino alla sezione successiva, dove esamineremo il controllo di congestione del protocollo TCP. Come secondo problema, dato che l'intera strategia di controllo della congestione viene implementata alle sorgenti e la gestione FIFO delle code non fornisce alcuna possibilità per stabilire quanto le sorgenti siano efficienti nel realizzare tale strategia, è possibile che una sorgente (o un flusso) che si comporta male possa impegnare una frazione arbitrariamente grande della capacità della rete. Considerando nuovamente Internet, è certamente possibile che un'applicazione non utilizzi il protocollo TCP e, di conseguenza, sia esente dal relativo meccanismo di controllo della congestione tra entità terminali (cioè che fanno, oggi, le applicazioni di telefonia su Internet). Un'applicazione di questo tipo è in grado di inondare i router di Internet con i propri pacchetti, provocando così l'eliminazione dei pacchetti di altre applicazioni.

La gestione equa delle code (FQ, *fair queueing*) è un algoritmo che è stato proposto per risolvere questo problema. L'idea di FQ consiste nella gestione di una diversa coda per ogni flusso gestito dal router in un certo istante: il router, quindi, serve queste code a rotazione (cioè in modalità *round-robin*), come mostrato in Figura 6.6. Quando un flusso invia pacchetti troppo rapidamente, la sua coda si riempie; quando una coda raggiunge una certa lunghezza, ulteriori pacchetti in arrivo che appartengano a quella coda vengono eliminati. In questo modo, una certa sorgente non può aumentare arbitrariamente la propria frazione della capacità della rete a spese degli altri flussi.

Note che la strategia FQ non prevede che i router segnalino alcunché alle sorgenti di traffico in merito allo stato del router stesso, né che limitino in alcun modo la velocità alla quale una sorgente può inviare pacchetti. In altre parole, anche la strategia FQ è stata progettata per un utilizzo congiunto ad un meccanismo di controllo della congestione end-to-end: semplicemente, suddivide il traffico in modo che sorgenti che non si comportano correttamente non interferiscono con quelle che implementano fedelmente l'algoritmo di controllo della congestione end-to-end. FQ assicura anche l'equità di trattamento di un insieme di flussi gestiti da un algoritmo di controllo della congestione ben progettato.

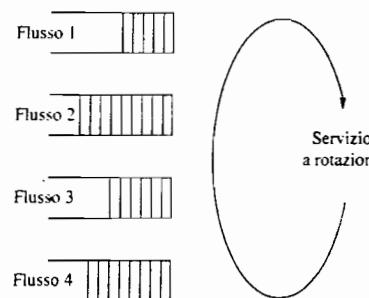


Figura 6.6 Gestione equa delle code (FQ, fair queueing) in un router.

Nonostante l'idea sia molto semplice, ci sono ancora alcuni dettagli che dovete conoscere. La complicazione maggiore consiste nel fatto che i pacchetti elaborati da un router non hanno necessariamente tutti la stessa lunghezza, per cui, per assegnare in modo equo l'ampiezza di banda di una linea di uscita, è necessario prendere in considerazione la lunghezza dei pacchetti. Ad esempio, se un router sta gestendo due flussi, uno con pacchetti di 1000 byte e un altro con pacchetti di 500 byte (magari a causa di una frammentazione avvenuta a monte del router in esame), allora un semplice prelievo di pacchetti a rotazione dalla coda di ciascun flusso darà al primo flusso i due terzi dell'ampiezza di banda della linea e al secondo flusso soltanto un terzo di tale banda.

Ciò che vorremmo ottenere, in realtà, è un servizio a rotazione bit per bit, cioè vorremmo che il router trasmettesse un bit dal flusso 1, poi un bit dal flusso 2, e così via, anche se, ovviamente, intervallare i bit di pacchetti diversi non è ragionevole. La strategia FQ, quindi, simula questo comportamento, determinando per prima cosa quando terminerebbe la trasmissione di un pacchetto se venisse inviato usando una rotazione bit per bit, e poi usando questo intervallo di tempo per porre in sequenza i pacchetti da trasmettere.

Per capire l'algoritmo che simula la rotazione bit per bit, considerate il comportamento di un singolo flusso e immaginate un orologio la cui lancetta si sposta di una tacca ogni volta che viene inviato un bit da tutti i flussi attivi (un flusso è attivo quando ha dati nella coda). Per il flusso in esame, sia P_i la lunghezza del pacchetto i , sia S_i l'istante in cui il router inizia la trasmissione del pacchetto i e sia F_i l'istante in cui il router termina la trasmissione del pacchetto i . Se P_i viene espresso come numero di tacche dell'orologio necessarie per trasmettere il pacchetto i (ricordando che l'orologio avanza di una tacca ogni volta che il flusso in esame viene servito per la trasmissione di un bit), allora è facile verificare che $F_i = S_i + P_i$.

Quando iniziamo la trasmissione del pacchetto i ? La risposta a questa domanda dipende dal fatto che il pacchetto i arrivi prima o dopo l'istante in cui il router ha terminato la trasmissione del pacchetto $i - 1$ del flusso in esame. Se arriva prima, allora il primo bit del pacchetto i viene trasmesso, logicamente, immediatamente dopo l'ultimo bit del pacchetto $i - 1$. D'altra parte, è possibile che il router abbia terminato di trasmettere il pacchetto $i - 1$ ben prima che arrivasse il pacchetto i , per cui c'è stato un periodo di tempo durante il quale la coda del flusso in esame era vuota, impedendo che il meccanismo di trasmissione a rotazione potesse trasmettere alcunché di relativo al flusso stesso. Se indichiamo con A_i l'istante in cui il pacchetto i arriva al router, allora $S_i = \max(F_{i-1}, A_i)$; di conseguenza, possiamo calcolare:

$$F_i = \max(F_{i-1}, A_i) + P_i$$

6.2 Politiche di gestione delle code

Ora prendiamo in esame la situazione in cui ci sia più di un flusso e cerchiamo un modo per determinare il valore di A_i : non possiamo guardare l'orologio quando arriva il pacchetto. Come abbiamo notato in precedenza, vogliamo che il tempo del nostro orologio avanzi di una tacca ogni volta che tutti i flussi attivi vengono serviti per un bit con il meccanismo a rotazione bit per bit, per cui ci serve un orologio che si muova più lentamente quando ci sono più flussi. In particolare, se ci sono n flussi attivi, l'orologio deve avanzare di una tacca quando vengono trasmessi n bit: per calcolare A_i useremo questo orologio.

A questo punto, per ciascun flusso, calcoliamo F_i per ogni pacchetto che arriva, usando la formula precedente. Successivamente, consideriamo tutti i valori F_i come indicazioni orarie (timestamp) ed il successivo pacchetto da trasmettere è sempre quello con l'indicazione oraria minore, quel che, in base al ragionamento precedente, terminerebbe la propria trasmissione prima di tutti gli altri.

Noteate che ciò significa che può arrivare, in un flusso, un pacchetto che, essendo più corto di un altro pacchetto dello stesso flusso che si trova già in coda in attesa di essere trasmesso, venga inserito nella coda davanti a tale pacchetto più lungo. Tuttavia, un pacchetto appena arrivato non può prendere il posto del pacchetto che si trova in fase di trasmissione: proprio questa mancanza di "prevaricazione" impedisce all'implementazione di FQ appena descritta di simulare in modo esatto lo schema di rotazione bit per bit che stiamo tentando di approssimare.

Per vedere meglio come lavora questa implementazione della gestione equa delle code, considerate l'esempio di Figura 6.7. La parte (a) mostra le code di due flussi: l'algoritmo decide che entrambi i pacchetti del flusso 1 vengano trasmessi prima del pacchetto che si trova nella coda del flusso 2. In (b), il router ha già iniziato a trasmettere un pacchetto del flusso 2, quando arriva il pacchetto del flusso 1: sebbene, nel caso in cui usassimo una gestione equa perfettamente bit per bit, la trasmissione del pacchetto appena arrivato per il flusso 1 terminerebbe prima di quella del pacchetto del flusso 2, l'implementazione che abbiamo visto non prevarica il pacchetto del flusso 2.

In relazione alla gestione equa delle code dobbiamo notare due cose. Prima di tutto, finché c'è almeno un pacchetto in coda la linea non viene mai lasciata inattiva: si dice che uno schema di gestione delle code che abbia questa proprietà è *work-conserving*. Un effetto di questa proprietà è che se sto condividendo una linea con molti flussi che non stanno inviando dati, posso usare per il mio flusso la piena capacità della linea. Non appena gli altri flussi iniziano a trasmettere, però, inizieranno ad usare la loro frazione di banda e la capacità disponibile per il mio flusso calerà.

La seconda cosa da notare è che se la linea è completamente caricata e ci sono n flussi che inviano dati, non posso usare più di una frazione di banda pari a $1/n$. Se provo ad inviare

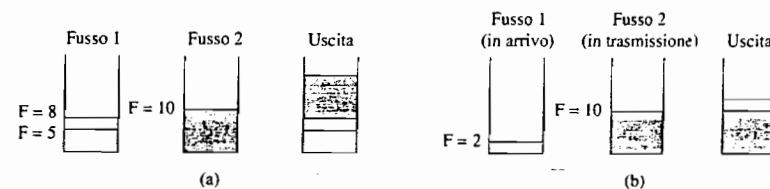


Figura 6.7 Esempio di gestione equa delle code in azione: (a) i pacchetti più corti vengono spediti per primi; (b) la trasmissione di pacchetti più lunghi, se già iniziata, viene portata a termine.

più dati, ai miei pacchetti verrà assegnata un'indicazione temporale di valore sempre più elevato, facendoli rimanere in coda sempre più a lungo, in attesa di essere trasmessi. Prima o poi la coda traboccherà, anche se l'eliminazione dei miei pacchetti o dei pacchetti di altri flussi è una decisione che non dipende dal fatto che stiamo usando la gestione *fair queueing*, ma viene determinata dalla strategia di eliminazione. FQ è un algoritmo di pianificazione che, come FIFO, può essere combinato con diverse strategie di eliminazione.

Dato che FQ è *work-conserving*, la banda che non viene usata da un flusso diviene automaticamente disponibile per gli altri flussi. Ad esempio, se abbiamo quattro flussi che attraversano un router e tutti stanno inviando pacchetti, allora ognuno di essi riceverà un quarto della banda totale. Ma se uno di loro diviene inattivo per un tempo sufficiente a fare in modo che la sua coda nel router venga svuotata di pacchetti, allora la banda disponibile verrà suddivisa fra i tre flussi restanti, ognuno dei quali riceverà un terzo dell'ampiezza di banda totale. Possiamo quindi pensare che FQ garantisca una frazione minima della banda a ciascun flusso, con la possibilità di averne di più di quanto garantito nel caso altri flussi non stiano usando la loro quota.

È possibile implementare una variante di FQ, chiamata *gestione delle code equa e pesata* (WFQ, *weighted fair queueing*), che consente l'assegnazione di un peso ad ogni flusso (o ad ogni coda). Tale peso specifica, dal punto di vista logico, il numero di bit che vengono trasmessi ogni volta che il router serve la coda, controllando così, in pratica, la percentuale di ampiezza di banda che viene assegnata al flusso corrispondente. La strategia FQ semplice assegna il peso unitario ad ogni coda, per cui, dal punto di vista logico, ogni volta che ciascuna coda viene servita si trasmette un solo bit: di conseguenza, ad ogni flusso viene assegnata una frazione dell'ampiezza di banda uguale a $1/n$, se n è il numero di flussi presenti. Con WFQ, invece, una coda potrebbe avere peso uguale a 2, una seconda coda potrebbe avere peso uguale a 1 e una terza coda potrebbe avere peso uguale a 3. Ipotizzando che tutte le code abbiano sempre almeno un pacchetto in attesa di essere trasmesso, il primo flusso otterrà un terzo della banda disponibile, il secondo ne otterrà un sesto e il terzo otterrà la metà della banda disponibile.

Anche se abbiamo descritto la strategia WFQ in termini di flussi, la si può anche implementare in termini di "classi" di traffico, definendo le classi in un modo un po' diverso da come abbiamo definito i semplici flussi presentati all'inizio di questo capitolo. Ad esempio, potremmo usare il bit *Type of Service* (TOS) dell'intestazione IP per identificare le classi e assegnare una coda e un peso a ciascuna classe: esattamente ciò che è stato proposto all'interno dell'architettura "Differentiated Services" (*servizi differenziati*), che sarà argomento della Sezione 6.5.3.

Note che un router che realizzi la strategia WFQ deve apprendere in qualche modo i pesi da assegnare a ciascuna coda, o tramite una configurazione manuale o con qualche forma di segnalazione proveniente dalla sorgente. Nel secondo caso, significa muoversi verso un modello basato sulla prenotazione: la semplice assegnazione di un peso ad una coda fornisce, però, un meccanismo di prenotazione piuttosto debole, perché questi pesi sono correlati alla banda ricevuta dal flusso soltanto in modo indiretto (l'ampiezza di banda disponibile per un flusso dipende anche, ad esempio, da quanti altri flussi condividono la linea in oggetto). Vedremo nella Sezione 6.5.2 come si possa usare WFQ come componente di una strategia di allocazione delle risorse basata sulla prenotazione.

Infine, osserviamo che questa intera discussione della gestione delle code illustra un importante principio di progettazione di sistemi, noto come *separazione delle discipline*.

pline dai meccanismi. L'idea è quella di vedere ciascun meccanismo come una scatola nera che fornisca un servizio multiforme, controllato da un insieme di manopole, mentre una disciplina specifica un particolare insieme di impostazioni di tali manopole, ma non sa (o non vuol sapere) come è realizzata la scatola nera. In questo caso, il meccanismo di cui parliamo è la gestione delle code, mentre la disciplina è una particolare impostazione che assegna a ciascun flusso un certo livello di servizio (una priorità o un peso). Nella Sezione 6.5 vedremo altre discipline che si possono usare congiuntamente al meccanismo WFQ.

6.3 Controllo di congestione in TCP

no i messaggi di ACK per controllare la velocità dei pacchetti
Questa sezione descrive il più importante esempio di controllo di congestione di tipo *end-to-end* usato attualmente, quello implementato all'interno del protocollo TCP. La strategia basale di TCP consiste nell'invio di pacchetti nella rete senza effettuare prenotazioni e, in seguito, reagire agli eventi osservabili che accadono. Il protocollo TCP ipotizza che i router della rete usino, semplicemente, una gestione FIFO delle code, ma può funzionare anche con una gestione FQ.

Il controllo di congestione di TCP è stato introdotto in Internet verso la fine degli anni Ottanta da Van Jacobson, circa otto anni dopo che la pila di protocolli TCP/IP era divenuta operativa. Subito prima di allora, Internet soffriva di collassi dovuti a congestione: gli host inviavano i propri pacchetti in Internet tanto velocemente quanto era consentito dalla finestra annunciata, qualche router andavano incontro al fenomeno di congestione e iniziava ad eliminare pacchetti, per cui gli host ritrasmettevano i propri pacchetti dopo la scadenza delle relative temporizzazioni, provocando ancora maggiore congestione.

Parlando in generale, l'idea del controllo di congestione in TCP affida ad ogni sorgente la responsabilità di determinare quanta capacità sia disponibile nella rete, in modo da sapere quanti pacchetti in transito sia possibile avere senza provocare problemi. Quando una sorgente sa di avere in transito un tale numero di pacchetti, usa l'arrivo di un ACK come segnalazione del fatto che uno dei propri pacchetti è uscito dalla rete, per cui ne può inserire in rete un altro senza timore di aumentare il livello di congestione. Dato che usa i messaggi ACK per controllare la velocità di trasmissione dei pacchetti, si dice che TCP è *self-clocking* (autotemporizzato). Ovviamente, determinare prima di tutto la capacità disponibile non è compito facile e, per rendere peggiore la situazione, la banda disponibile cambia nel tempo, a causa di altre connessioni che vengono instaurate e terminate: ciò significa che una sorgente deve essere in grado di modificare il numero di propri pacchetti in transito nella rete. Questa sezione descrive gli algoritmi usati dal protocollo TCP per risolvere questi e altri problemi. Notate che, anche se descriviamo questi meccanismi uno alla volta, dando così l'impressione che si tratti di tre strategie indipendenti, si ottiene il controllo di congestione di TCP soltanto quando li si considera in modo complessivo.

6.3.1 Aumento additivo/diminuzione moltiplicativa

Il protocollo TCP gestisce, per ogni connessione, una nuova variabile di stato, chiamata CongestionWindow, che viene usata dalla sorgente per limitare la quantità di propri dati in transito nella rete in un certo istante. La finestra di congestione è, per il controllo di congestione, la controparte della finestra annunciata (AdvertisedWindow) nel controllo di flusso. Il protocollo TCP viene modificato in modo che il numero massimo consentito di byte di dati

non confermati sia, ora, il minimo fra la finestra di congestione e la finestra annunciata. In tal modo, usando le variabili definite nella Sezione 5.2.4, la finestra effettiva di TCP diventa così calcolata:

$$\begin{aligned} \text{MaxWindow} &= \text{MIN}(\text{CongestionWindow}, \text{AdvertisedWindow}) \\ \text{EffectiveWindow} &= \text{MaxWindow} - (\text{LastByteSent} - \text{LastByteAcked}) \end{aligned}$$

In sostanza, MaxWindow sostituisce AdvertisedWindow nel calcolo di EffectiveWindow. In questo modo una sorgente TCP non può spedire più velocemente di quanto sia accettabile per il componente più lento: la rete o l'host di destinazione.

Il problema, ovviamente, è come TCP apprenda il valore appropriato per CongestionWindow. Diversamente da AdvertisedWindow, che viene inviato dall'estremo ricevente della connessione, nessuno invia un valore plausibile di CongestionWindow al lato mittente della connessione TCP. La risposta è che la sorgente TCP imposta il valore di CongestionWindow basandosi sul livello di congestione di rete da essa percepito: questo richiede la diminuzione della finestra di congestione quando il livello di congestione sale e l'aumento della finestra di congestione quando, viceversa, il livello di congestione scende. Considerato nella sua globalità, questo meccanismo viene comunemente chiamato aumento additivo / diminuzione moltiplicativa (AIMD, Additive Increase, Multiplicative Decrease): a breve diverrà evidente la giustificazione per un nome tanto altisonante.

La domanda chiave, ora, è: come può la sorgente determinare che la rete è congestionata e che bisogna diminuire la finestra di congestione? La risposta si basa sull'osservazione che il motivo principale per cui i pacchetti non vengono consegnati, provocando la scadenza delle relative temporizzazioni, è la loro eliminazione in seguito a congestione: è raro che un pacchetto venga eliminato per effetto di un errore di trasmissione. Quindi, il protocollo TCP interpreta la scadenza delle temporizzazioni come un segnale di congestione e riduce la velocità di trasmissione. In particolare, ogni volta che scade una temporizzazione, la sorgente imposta CongestionWindow alla metà del suo valore precedente: questo dimezzamento del valore di CongestionWindow in risposta ad ogni temporizzazione che scade corrisponde alla parte "diminuzione moltiplicativa" di AIMD.

Sebbene il valore di CongestionWindow sia definito in termini di byte, è più facile capire il significato della diminuzione moltiplicativa se ragioniamo in termini di interi pacchetti. Ad esempio, supponete che CongestionWindow valga 16 (pacchetti). Se viene rilevata la perdita di un pacchetto, il valore di CongestionWindow passa a 8 (solitamente viene rilevata la perdita di un pacchetto quando scade una temporizzazione, ma, come vedremo in seguito, TCP ha un altro meccanismo per accorgersi dell'eliminazione di pacchetti). Ulteriori perdite di pacchetti portano CongestionWindow ai valori 4, 2 e, infine, 1 (pacchetto). Il valore di CongestionWindow non può scendere al di sotto di un singolo pacchetto, o, nella terminologia di TCP, al di sotto del valore di MSS (dimensione massima di un segmento).

Una strategia di controllo della congestione che sappia solo diminuire la dimensione della finestra è, ovviamente, troppo cautelativa: dobbiamo anche essere in grado di aumentare la dimensione della finestra di congestione, per sfruttare nuove capacità che si rendano disponibili nella rete. Stiamo parlando della sezione di "aumento additivo" di AIMD, che funziona come descritto in seguito. Ogni volta che una sorgente invia una quantità di pacchetti pari al valore di CongestionWindow con successo (ossia tutti i pacchetti inviati durante l'ultimo periodo di durata RTT sono stati confermati), incrementa il valore di CongestionWindow per l'equivalente di un pacchetto: questo aumento lineare è mostrato in Figura 6.8. Notate

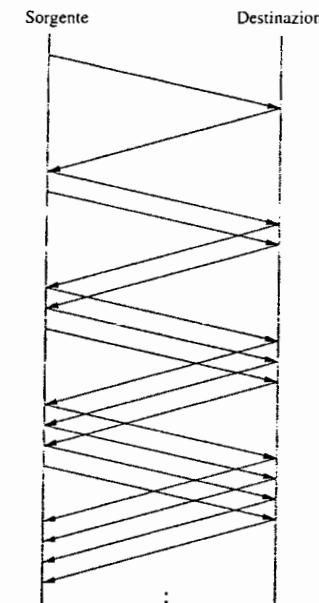


Figura 6.8 Pacchetti in transito durante una fase di aumento additivo, con l'aggiunta di un pacchetto per ogni RTT.

che, in pratica, TCP non attende la conferma per un'intera finestra di pacchetti prima di aggiungere un pacchetto alla finestra di congestione, ma incrementa un po' CongestionWindow ogni volta che arriva una conferma. In particolare, la finestra di congestione viene aumentata, ogni volta che arriva un ACK, come segue:

$$\begin{aligned} \text{Increment} &= \text{MSS} \times (\text{MSS}/\text{CongestionWindow}) \\ \text{CongestionWindow} &+= \text{Increment} \end{aligned}$$

In sostanza, invece di aumentare CongestionWindow di un intero MSS per ogni RTT, lo incrementiamo per una frazione di MSS ogni volta che viene ricevuto un ACK: nell'ipotesi che ciascun ACK confermi la ricezione di un numero di byte uguale a MSS, tale frazione vale MSS/CongestionWindow.

Questo andamento, fatto di continui aumenti e diminuzioni della finestra di congestione, continua per tutta la durata della connessione: se si traccia un grafico del valore di CongestionWindow in funzione del tempo, si osserva uno schema a dente di sega (*sawtooth pattern*), come evidenziato in Figura 6.9. Il concetto importante da capire in merito a AIMD è che la sorgente è disposta a ridurre la propria finestra di congestione molto più rapidamente di quanto la aumenti, in netto contrasto con la strategia ad aumento additivo / diminuzione additiva che porterebbe ad aumentare la finestra di un pacchetto ogni volta che si riceve una conferma ACK e a diminirla di un pacchetto quando scade una temporizzazione. È stato

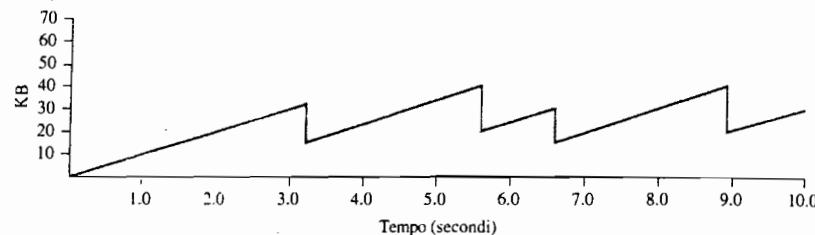


Figura 6.9 Tipico diagramma a dente di sega in TCP.

dimostrato che AIMD è una condizione necessaria perché un meccanismo di controllo della congestione sia stabile (si veda la sezione "Ulteriori Letture" al termine del capitolo). Una motivazione intuitiva che giustifica la diminuzione aggressiva della dimensione della finestra e l'aumento cauto della stessa sta nel fatto che avere una finestra troppo ampia ha conseguenze assai peggiori del fatto di averla troppo piccola. Ad esempio, quando la finestra è troppo ampia, i pacchetti eliminati devono essere ritrasmessi, rendendo ancora peggiore la congestione: è quindi importante uscire il più rapidamente possibile da tale stato.

Dato che, infine, la scadenza di una temporizzazione è un segnale di congestione che innesca una diminuzione moltiplicativa della finestra, il protocollo TCP necessita del meccanismo di temporizzazione più accurato possibile. Abbiamo già trattato il meccanismo delle temporizzazioni di TCP nella Sezione 5.2.6 e non lo riprenderemo qui, ma le due cose principali da ricordare in relazione ad esso sono che le temporizzazioni vengono impostate come funzione del valore medio di RTT e della deviazione standard di tale valore medio, e che, a causa dell'elevato costo di una misurazione accurata del tempo necessario per ciascuna trasmissione, il protocollo TCP effettua un solo campionamento del tempo di round-trip per ogni intervallo di tempo di durata RTT (invece che per ogni pacchetto), usando un orologio piuttosto grossolano (500 ms).

6.3.2 Partenza lenta → ~~Unito all'ultimo alla connessione~~

Il meccanismo di aumento additivo appena descritto è l'approccio corretto da usare quando la sorgente si trova ad operare in condizioni prossime alla capacità disponibile della rete, ma richiede troppo tempo per far decollare la connessione quando essa inizia la trasmissione. Il protocollo TCP, quindi, dispone di un secondo meccanismo, chiamato ironicamente partenza lenta (slow start), che viene usato per aumentare rapidamente la dimensione della finestra di congestione all'inizio della connessione: la partenza lenta aumenta la finestra di congestione esponenzialmente, anziché linearmente.

In particolare, la sorgente inizia impostando CongestionWindow al valore corrispondente ad un solo pacchetto. Quando arriva la conferma ACK per tale pacchetto, TCP aggiunge 1 al valore di CongestionWindow e invia due pacchetti. Dopo aver ricevuto le due conferme corrispondenti, TCP aumenta il valore di CongestionWindow di 2, un'unità per ogni ACK ricevuto, inviando successivamente quattro pacchetti. Il risultato finale è che TCP, in pratica, raddoppia il numero di pacchetti in transito ad ogni intervallo di tempo uguale a RTT. La Figura 6.10 mostra la crescita del numero di pacchetti in transito durante la partenza lenta: confrontatela con la crescita lineare dell'aumento additivo mostrata in Figura 6.8.

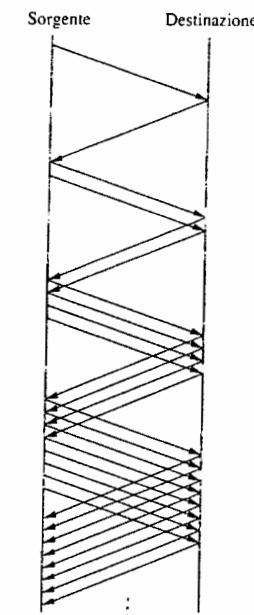


Figura 6.10 Pacchetti in transito durante la partenza lenta.

Perché mai un meccanismo esponenziale sia stato chiamato "lento" può sembrare assai strano, a prima vista, ma lo si può spiegare inserendo la cosa nel giusto contesto storico. Occorre confrontare la "partenza lenta" non con il meccanismo lineare della sottosezione precedente, ma con il comportamento originario di TCP. Considerate cosa accade quando viene instaurata una connessione e la sorgente inizia ad inviare pacchetti per la prima volta, cioè quando non ha pacchetti in transito. Se la sorgente invia tanti pacchetti quanti ne sono consentiti dalla finestra annunciata, che era proprio ciò che faceva TCP prima che fosse sviluppata la partenza lenta, allora i router potrebbero non essere in grado di assorbire questa improvvisa quantità di nuovi pacchetti (*burst*), anche se nella rete fosse disponibile un'ampiezza di banda sufficiente: dipende tutto da quanto spazio è disponibile nei buffer dei router. **La partenza lenta fu quindi progettata per distanziare i pacchetti fra loro, in modo che non si verificassero burst di quel genere.** In altre parole, anche se l'aumento esponenziale è più veloce dell'aumento lineare, la partenza lenta è molto "più lenta" dell'invio improvviso di una quantità di dati che riempia l'intera finestra consentita.

Esistono, in realtà, due situazioni in cui viene usata la partenza lenta. La prima è all'inizio di una connessione, nel momento in cui la sorgente non ha alcuna idea in merito a quanti pacchetti si possa permettere di mantenere in transito nella rete (ricordate che il protocollo TCP viene eseguito in qualsiasi tipo di rete, con linee a 9600 bps e linee a 2.4 Gbps, per cui non c'è modo, per la sorgente, di conoscere la capacità della rete). In questa situazione, la strategia di partenza lenta continua a raddoppiare il valore di CongestionWindow ad ogni RTT finché non si verificano perdite di pacchetti: in quel momento, la scadenza di una

temporizzazione provoca una diminuzione moltiplicativa, dividendo per 2 il valore di CongestionWindow.

La seconda situazione in cui viene usata la partenza lenta è un po' più delicata e accade quando la connessione si interrompe mentre la sorgente è in attesa della scadenza di una temporizzazione. Ricordate come funziona l'algoritmo sliding window di TCP: quando si perde un pacchetto, prima o poi la sorgente arriva ad un punto in cui ha inviato tutti i dati consentiti dalla finestra e si blocca in attesa di un ACK che non arriverà. Presto o tardi si avrà la scadenza di una temporizzazione, ma a quel punto non ci saranno più pacchetti in transito, per cui la sorgente non riceverà alcun ACK che possa "stimolare" la trasmissione di nuovi pacchetti. Al contrario, la sorgente riceverà un unico ACK cumulativo, che riapre l'intera finestra annunciata: in questo caso, come spiegato in precedenza, la sorgente fa ripartire il flusso di dati usando la partenza lenta, piuttosto che scaricare improvvisamente nella rete una quantità di dati tale da riempire completamente la finestra.

Sebbene la sorgente stia usando nuovamente la partenza lenta, ha ora maggiori informazioni di quante ne avesse all'inizio della connessione. In particolare, la sorgente ha un valore attuale (e utilizzabile) di CongestionWindow, il valore esistente prima dell'ultima perdita di pacchetto, diviso per 2 per effetto della perdita stessa. Possiamo considerare questo valore come l'obiettivo della finestra di congestione: si usa la partenza lenta per aumentare rapidamente la velocità di trasmissione fino a tale punto, usando l'aumento additivo da quel punto in poi. Notate che ci dobbiamo far carico di un piccolo problema di gestione, perché vogliamo ricordare sia la finestra di congestione risultante dalla diminuzione moltiplicativa sia quella attualmente utilizzata dalla partenza lenta. Per risolvere questo problema, il protocollo TCP introduce una variabile temporanea che memorizza la "finestra obiettivo", denominata solitamente CongestionThreshold, a cui viene assegnato il valore risultante dalla diminuzione moltiplicativa di CongestionWindow. La variabile CongestionWindow viene poi ad assumere il valore iniziale corrispondente ad un solo pacchetto e viene incrementata di un pacchetto per ogni ACK che si riceve, fino al raggiungimento del valore CongestionThreshold: da quel momento in poi, viene incrementata di un pacchetto per ogni RTT.

In altre parole, il protocollo TCP aumenta la finestra di congestione con la modalità definita dal seguente frammento di codice:

```
{
    u_int cw = state->CongestionWindow;
    u_int incr = state->maxseg;

    if (cw > state->CongestionThreshold)
        incr = incr * incr / cw;
    state->CongestionWindow = MIN(cw + incr, TCP_MAXWIN);
}
```

dove state rappresenta lo stato di una connessione TCP e TCP_MAXWIN definisce il limite superiore della dimensione fino a cui può crescere la finestra di congestione.

La Figura 6.11 mostra come il valore di CongestionWindow in TCP aumenti e diminuisca nel tempo e ha lo scopo di illustrare l'interazione tra la partenza lenta e la strategia di aumento additivo / diminuzione moltiplicativa. Si tratta di un grafico derivato da una vera connessione TCP e mostra l'andamento temporale del valore di CongestionWindow (linea di spessore maggiore).

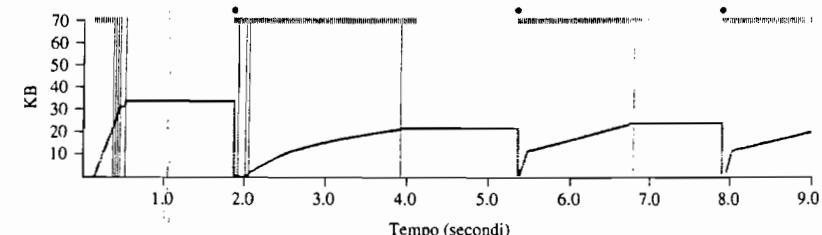


Figura 6.11 Comportamento del controllo di congestione di TCP. La linea di spessore maggiore raffigura l'andamento del valore di CongestionWindow nel tempo; i pallini nella parte alta del grafico indicano gli istanti in cui scadono le temporizzazioni; i trattini verticali nella parte alta del grafico indicano le trasmissioni di pacchetti; le linee verticali che attraversano il grafico segnalano gli istanti in cui furono trasmessi per la prima volta i pacchetti che subiscono ritrasmissioni.

Vogliamo far notare diverse cose in questo diagramma. Prima di tutto, il rapido aumento della finestra di congestione all'inizio della connessione, che corrisponde alla fase iniziale della strategia di "partenza lenta". Tale fase prosegue finché non vengono perduti alcuni pacchetti, dopo circa 0,4 secondi dall'inizio della connessione: a quel punto, il valore di CongestionWindow rimane costante, intorno a 34 KB (parleremo dopo del motivo per cui nella fase di partenza lenta vengono perduti così tanti pacchetti). La finestra di congestione rimane costante perché non arrivano conferme ACK, dato che parecchi pacchetti sono andati perduti: a dire il vero, durante questi istanti non viene inviato alcun pacchetto, come si può notare osservando la mancanza di trattini nella parte alta del grafico. Dopo circa 2 secondi dall'inizio della connessione scade una temporizzazione, che provoca il dimezzamento della finestra di congestione (che passa da un valore di circa 34 KB ad un valore di circa 17 KB), portando CongestionThreshold a questo stesso valore. La partenza lenta, poi, riporta al valore unitario iniziale la variabile CongestionWindow, che ricomincia ad aumentare da quel punto.

Nel grafico non sono rappresentati sufficienti dettagli per vedere cosa accade esattamente ad un paio di pacchetti che vengono perduti poco dopo i 2 secondi, per cui andiamo oltre ed osserviamo l'aumento lineare della finestra di congestione che avviene fra i 2 e i 4 secondi: si tratta della fase di aumento additivo. Dopo circa 4 secondi dall'inizio della connessione, il valore di CongestionWindow diviene nuovamente costante, ancora una volta per effetto della perdita di un pacchetto. Ora, dopo circa 5,5 secondi:

1. Scade una temporizzazione, provocando il dimezzamento della finestra di congestione, che passa dal valore di circa 22 KB al valore di circa 11 KB; a tale valore finale si porta anche la variabile CongestionThreshold.
2. Il mittente entra in una nuova fase di partenza lenta e CongestionWindow riparte dal valore iniziale unitario.
3. La partenza lenta fa crescere esponenzialmente CongestionWindow finché non raggiunge CongestionThreshold.
4. Successivamente, CongestionWindow cresce linearmente.

Dopo circa 8 secondi dall'inizio della connessione, scade un'altra temporizzazione e si ripete il medesimo schema.

Torniamo ora alla domanda che ci siamo posti, in merito al motivo per cui nel periodo iniziale di partenza lenta vengano perduti così tanti pacchetti. In quei momenti, il protocollo TCP sta cercando di capire quanta banda sia disponibile nella rete, che è un compito veramente difficile. Se in questa fase la sorgente non fosse sufficientemente aggressiva, ad esempio aumentando soltanto linearmente la finestra di congestione, occorrerebbe molto tempo per scoprire quale ampiezza di banda sia disponibile, un fatto che potrebbe avere un impatto molto negativo sul throughput raggiunto dalla connessione. D'altra parte, se la sorgente in questa fase è aggressiva, come lo è il protocollo TCP durante la crescita esponenziale, si corre il rischio di perdere, per eliminazione all'interno della rete, una quantità di pacchetti pari alla metà della finestra.

Per vedere cosa accade durante la crescita esponenziale, considerate la situazione in cui la sorgente è appena riuscita ad inviare con successo 16 pacchetti attraverso la rete, provocando il raddoppio della finestra di congestione, che assume il valore 32. Supponete, però, che la capacità della rete possa ricevere solamente 16 pacchetti da tale sorgente: il risultato più probabile è che la rete elimini 16 dei 32 pacchetti inviati per effetto del nuovo valore della finestra di congestione (anche se si tratta del caso peggiore, perché in realtà alcuni dei pacchetti potranno essere conservati nei buffer di qualche router). Questo problema diventerà sempre più grave all'aumentare del prodotto ritardo \times ampiezza di banda delle reti. Ad esempio, con un prodotto ritardo \times ampiezza di banda di 500 KB si corre il rischio di perdere fino a 500 KB di dati all'inizio di ogni connessione, ovviamente nell'ipotesi che sia la sorgente che la destinazione implementino l'estensione delle "finestre di grandi dimensioni" (*big windows extension*).

Alcuni progettisti di reti hanno proposto alternative alla partenza lenta, nelle quali la sorgente cerca di stimare l'ampiezza di banda disponibile in modi che siano migliori rispetto al fatto di spedire gruppi di pacchetti e stare a vedere quanti riescono ad arrivare a destinazione. Un tecnica, denominata *packet pair* (*coppia di pacchetti*), è rappresentativa di questa strategia generale. In poche parole, l'idea è quella di inviare una coppia di pacchetti senza alcun intervallo interposto e osservare, da parte della sorgente, quale sia il ritardo relativo fra i due pacchetti di conferma ACK. L'intervallo fra tali pacchetti ricevuti viene preso come misura della congestione presente nella rete, provocando un aumento conseguente della finestra di congestione. Non è ancora stato emanato un verdetto definitivo sull'efficacia di approcci di questo tipo, anche se i risultati sembrano essere promettenti.

6.3.3 Ritrasmissione veloce e recupero veloce

Il meccanismo descritto fino ad ora faceva parte della proposta originaria che introduce il controllo di congestione nel protocollo TCP. Tuttavia, ben presto ci si accorse che la grossolana implementazione delle temporizzazioni in TCP portava a lunghi periodi di tempo durante i quali la connessione andava incontro a fenomeni di stagnazione in attesa che scadesse una temporizzazione. A causa di ciò, fu aggiunto al protocollo TCP un nuovo meccanismo, denominato *ritrasmissione veloce* (*fast retransmit*): un approccio euristico che a volte innescava la ritrasmissione di un pacchetto perduto prima che scada la normale temporizzazione. Questo meccanismo non va a rimpiazzare le normali temporizzazioni, ma viene solo usato per migliorarne la funzionalità.

L'idea della ritrasmissione veloce è molto semplice: ogni volta che un pacchetto di dati arriva a destinazione, il ricevente risponde con una conferma, anche se con un numero di

sequenza che è già stato confermato. In questo modo, quando arriva un pacchetto fuori sequenza, che non può quindi essere confermato da TCP perché non sono ancora arrivati i pacchetti precedenti, il protocollo rispedisce la stessa conferma spedita la volta precedente: questa seconda trasmissione della stessa conferma viene chiamata *ACK duplicato*. Quando il mittente vede un ACK duplicato, sa che il ricevente ha certamente ricevuto un pacchetto fuori sequenza, da cui si deduce che un pacchetto precedente potrebbe essere andato perduto. Dato che tale pacchetto precedente potrebbe essere stato semplicemente ritardato dalla rete, anziché perduto, il mittente attende di vedere un certo numero di ACK duplicati, dopodiché ritrasmette il pacchetto mancante. In pratica, il protocollo TCP attende di vedere tre ACK duplicati prima di ritrasmettere il pacchetto.

La Figura 6.12 mostra come i pacchetti ACK duplicati portano ad una ritrasmissione veloce. In questo esempio, il destinatario riceve i pacchetti 1 e 2, ma il pacchetto 3 viene perduto dalla rete. Di conseguenza, quando arriva il pacchetto 4, il destinatario invierà un ACK duplicato per il pacchetto 2, e lo farà di nuovo quando arriva il pacchetto 5, e così via (per semplificare l'esempio, ragioniamo in termini di pacchetti 1, 2, 3 e così via, invece di preoccuparci dei numeri di sequenza di ciascun byte). Quando il mittente vede il terzo ACK duplicato per il pacchetto 2, quello inviato dal ricevente all'arrivo del pacchetto 6, viene ritrasmesso il pacchetto 3. Notate che quando arriva a destinazione la copia del pacchetto 3 il ricevitore invia al mittente un ACK cumulativo per tutto quanto ricevuto fino al pacchetto 6 incluso.

La Figura 6.13 mostra il comportamento di una versione di TCP che implementa il meccanismo di ritrasmissione veloce: è interessante confrontare questo grafico con quello di Figura 6.11, dove non era implementata la ritrasmissione veloce. Sono spariti i lunghi periodi di tempo durante i quali la finestra di congestione rimaneva costante e non venivano spediti pacchetti. In generale, questa tecnica è in grado di eliminare circa la metà delle scadenze di temporizzazioni grossolane di una tipica connessione TCP, dando luogo ad un miglioramen-

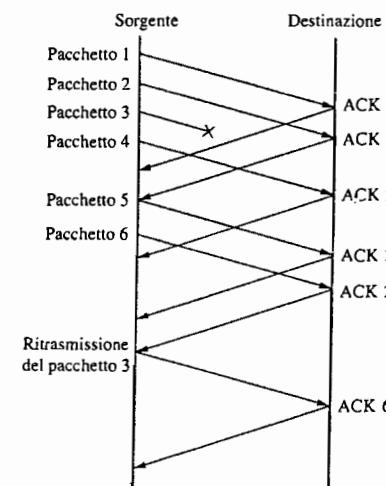


Figura 6.12 Ritrasmissione veloce basata su ACK duplicati.

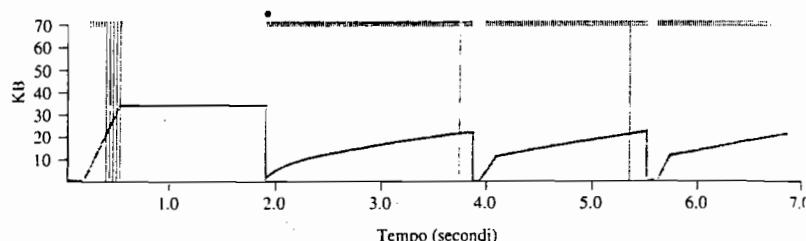


Figura 6.13 Funzionamento di TCP con la ritrasmissione veloce. La linea di spessore maggiore raffigura l'andamento del valore di CongestionWindow nel tempo; i pallini nella parte alta del grafico indicano gli istanti in cui scadono le temporizzazioni; i trattini verticali nella parte alta del grafico indicano le trasmissioni di pacchetti; le linee verticali che attraversano il grafico segnalano gli istanti in cui furono trasmessi per la prima volta i pacchetti che subiscono ritrasmissioni.

to di circa il 20% del throughput rispetto a quanto si potrebbe ottenere altrimenti. Notate, però, che la strategia di ritrasmissione veloce non elimina tutte le scadenze di temporizzazioni, perché, con una finestra di piccole dimensioni, non si ha un numero di pacchetti in transito sufficiente a generare abbastanza ACK duplicati. Con un numero sufficientemente elevato di pacchetti perduti, come accade, ad esempio, durante la fase iniziale di partenza lenta, l'algoritmo sliding window provoca, prima o poi, il blocco del mittente fino alla scadenza di una temporizzazione. Con l'attuale dimensione massima della finestra pubblicizzata da TCP, 64 KB, il meccanismo di ritrasmissione veloce è in grado, in pratica, di rilevare fino a tre pacchetti eliminati per ogni finestra.

Infine, possiamo realizzare un ultimo miglioramento. Quando il meccanismo di ritrasmissione veloce segnala una congestione, invece di riportare la finestra di congestione al suo valore iniziale unitario ed eseguire nuovamente la partenza lenta, si possono usare i pacchetti di conferma, ACK, che si trovano ancora in transito per pianificare l'invio dei pacchetti. Questo meccanismo, solitamente chiamato *recupero veloce (fast recovery)*, a tutti gli effetti elimina la fase di partenza lenta che avviene tra il momento in cui la ritrasmissione veloce rileva la perdita di un pacchetto e quello in cui inizia l'aumento additivo. Ad esempio, il recupero veloce evita l'intervallc di partenza lenta che si trova, nella Figura 6.13, tra 3.8 e 4 secondi dall'inizio della connessione, dimezzando semplicemente la finestra di congestione (da 22 KB a 11 KB) e riprendendo l'aumento additivo. In altre parole, la partenza lenta viene usata solamente all'inizio della connessione oppure quando si ha la scadenza di una temporizzazione: in tutte le altre occasioni, la finestra di congestione segue fedelmente lo schema di aumento additivo e diminuzione moltiplicativa.

6.4 Strategie per evitare la congestione

È importante capire che la strategia del protocollo TCP punta a controllare la congestione quando questa accade, piuttosto che cercare, prima di tutto, di evitarla. Infatti, TCP aumenta ripetutamente il carico imposto alla rete, nel tentativo di trovare il punto in cui si verifica la congestione, per poi spostarsi all'indietro da tale punto. Detto in altro modo, il protocollo

6.4 Strategie per evitare la congestione

TCP ha bisogno di provocare la perdita di pacchetti per determinare quale sia la banda disponibile per la connessione. Un'attrattiva alternativa, che non ha però ancora avuto ampia diffusione, consiste nel prevedere un'imminente congestione e ridurre la velocità di spedizione dei pacchetti prima che questi vengano eliminati. In riferimento a questa strategia, parleremo di *impedire la congestione (congestion avoidance)*, per distinguerla dal controllo di congestione.

Questa sezione descrive tre diversi meccanismi per impedire l'insorgere della congestione. I primi due seguono un approccio simile: aggiungono al router poche funzionalità che assistano il nodo terminale nella previsione della congestione. Il terzo meccanismo è molto diverso dai primi due e tenta di evitare la congestione senza alcuna assistenza da parte della rete.

6.4.1 DECbit

La prima strategia fu sviluppata per la Digital Network Architecture (DNA), una rete non orientata alla connessione con un protocollo di trasporto orientato alla connessione: di conseguenza, tale meccanismo può essere applicato anche al protocollo TCP, che si trova in un contesto analogo in una rete IP. Come abbiamo già detto, in questo caso l'idea consiste nel suddividere maggiormente la responsabilità del controllo di congestione tra i router e i nodi terminali. Ciascun router controlla il carico a cui è sottoposto e segnala esplicitamente ai nodi terminali quando sta per accadere una congestione. Questa segnalazione viene implementata mediante un bit di congestione nei pacchetti che fluiscono attraverso il router, da cui il nome DECbit. L'host destinatario, quindi, copia questo bit di congestione nel pacchetto ACK che restituisce al mittente e, infine, la sorgente modifica la propria velocità in modo da evitare la congestione. La discussione seguente descrive l'algoritmo in maggiore dettaglio, iniziando da ciò che accade nel router.

Nell'intestazione del pacchetto viene aggiunto un singolo bit di congestione, che viene impostato al valore 1 dal router se la lunghezza media delle proprie code è maggiore o uguale a 1 nel momento in cui arriva il pacchetto. Questa lunghezza media delle code viene misurata su un intervallo di tempo che copre l'ultimo ciclo di attività e di inattività, esteso all'attuale ciclo di attività (il router è attivo, *busy*, quando sta trasmettendo, ed è inattivo, *idle*, quando non lo sta facendo). La Figura 6.14 mostra la lunghezza delle code in un router in funzione del tempo. In sostanza, il router calcola l'area sottesa dalla curva e la divide per la durata dell'intervallo di tempo, calcolando così la lunghezza media delle code. L'uso di una lunghezza unitaria delle code come valore che innesta la segnalazione nel bit di congestione è un compromesso fra una significativa occupazione delle code (e quindi un throughput elevato) e un aumento del tempo di inattività (e quindi un minore ritardo). In altre parole, sembra che una coda di lunghezza unitaria ottimizzi la funzione potenza.

Volgendo ora la nostra attenzione alla parte del meccanismo che compete agli host, la sorgente memorizza il numero di pacchetti che hanno provocato l'impostazione del bit di congestione da parte di qualche router. In particolare, la sorgente gestisce una finestra di congestione, esattamente come nel protocollo TCP, ed osserva quale frazione di pacchetti dell'ultima finestra hanno avuto il bit impostato al valore 1. Se tale frazione è inferiore al 50%, allora la sorgente aumenta di un pacchetto la propria finestra di congestione, altrimenti la decremente ad un valore uguale a 0.875 moltiplicato per il valore precedente. Il valore di 50% è stato scelto come soglia in base ad un'analisi che ha mostrato la sua corri-

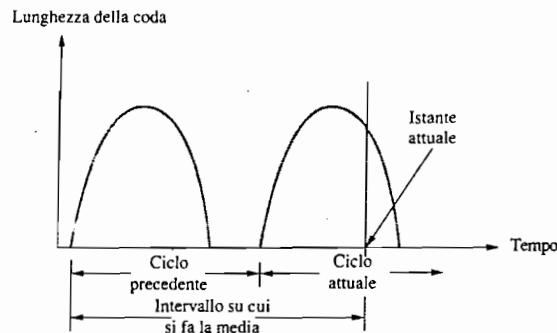


Figura 6.14 Calcolo della lunghezza media della coda di un router.

spondenza con il picco della curva della funzione potenza. La regola "aumenta di 1 e diminisci moltiplicando per 0.875" è stata scelta perché l'aumento additivo e la diminuzione moltiplicativa rende stabile il meccanismo.

6.4.2 Random early detection (RED)

Un secondo meccanismo, denominato *rilevazione casuale anticipata* (RED, *random early detection*), è simile allo schema DECbit, in quanto ogni router viene programmato perché tenga sotto controllo la lunghezza delle proprie code e, quando si accorge che una congestione è imminente, segnali alla sorgente di modificare la propria finestra di congestione. L'approccio RED, inventato da Sally Floyd e Van Jacobson nei primi anni Novanta, differisce dallo schema DECbit in due aspetti fondamentali.

Innanzitutto, l'invio di un messaggio di segnalazione di congestione ad una sorgente è tutt'altro che esplicito, in quanto solitamente RED viene realizzato in modo da segnalare *implicitamente* alla sorgente la congestione, eliminando uno dei suoi pacchetti. La sorgente, quindi, riceve a tutti gli effetti la segnalazione nel momento della successiva scadenza della propria temporizzazione, oppure mediante un ACK duplicato. Nel caso in cui non l'aveste ancora intuito, RED è stato progettato per essere utilizzato insieme al protocollo TCP, che sostanzialmente si accorge della congestione tramite la scadenza delle proprie temporizzazioni (o in qualche altro modo di rilevazione della perdita di pacchetti, come l'uso di ACK duplicati). Come suggerito dal termine "anticipata" (*early*) presente nell'acronimo RED, il router elimina un pacchetto prima che ciò sia realmente necessario, in modo da segnalare alla sorgente l'opportunità di diminuire la propria finestra di congestione prima che ciò si renda veramente necessario. In altre parole, il router elimina alcuni pacchetti prima di esaurire completamente il proprio spazio di memorizzazione nei buffer, per provocare un rallentamento della sorgente, con la speranza di non dover eliminare un maggior numero di pacchetti più tardi. Notate che lo schema RED si può facilmente adattare per funzionare con una segnalazione esplicita, semplicemente contrassegnando un pacchetto invece di *eliminarlo*, come discusso nel riquadro relativo alla segnalazione esplicita di congestione (ECN).

Segnalazione esplicita di congestione (ECN, explicit congestion notification)

Anche se la maggior parte delle implementazioni di RED segnalano la congestione eliminando pacchetti, recentemente è stata dedicata molta attenzione alla valutazione di strategie di segnalazione esplicita, che ha portato verso uno sforzo di standardizzazione di ECN per Internet.

L'osservazione di base è che, nonostante l'eliminazione di un pacchetto agisca certamente come segnale di congestione e sia probabilmente la cosa migliore da fare per trasferimenti di ingenti quantità di dati e di lunga durata, questa strategia crea problemi ad applicazioni che siano sensibili ai ritardi o alla perdita di uno o più pacchetti. Il traffico interattivo, come quello di applicazioni Telnet e navigatori Web, è uno degli esempi principali: per tali applicazioni, apprendere di un'imminente congestione tramite una segnalazione esplicita è più appropriato.

Tecnicamente, ECN richiede 2 bit e lo standard proposto usa i bit 6 e 7 del campo TOS di IP. Uno di questi bit viene impostato dalla sorgente per segnalare di essere in grado di gestire il meccanismo ECN, cioè di saper reagire alle segnalazioni di congestione; l'altro viene impostato dai router lungo il percorso tra nodi terminali quando si verifica una congestione. Quest'ultimo bit viene anche restituito alla sorgente dall'host di destinazione. Il protocollo TCP in esecuzione nella sorgente risponde all'impostazione del bit ECN esattamente come risponde alla perdita di un pacchetto.

Come accade per ogni idea valida, questa recente attenzione nei confronti di ECN ha indotto molte persone a soffermarsi a riflettere su quali altri modi, all'interno delle reti, si possa trarre beneficio da uno scambio di informazioni tra gli host che si trovano ai confini della rete e i router al suo interno. Analogamente a quanto avviene in ECN, con un meccanismo di *piggyback* sui pacchetti di dati. Questa generica strategia viene a volte chiamata *gestione attiva della coda* (*active queue management*) e recenti ricerche sembrano indicare che sia particolarmente adatta a flussi TCP, che abbiano un elevato prodotto ritardo × ampiezza di banda. Il lettore che ne fosse interessato può rintracciare materiale rilevante nelle citazioni fornite alla fine del capitolo.

La seconda differenza tra RED e DECbit consiste nei dettagli relativi a come RED decide che sia giunto il momento di eliminare un pacchetto e quale sia il pacchetto da eliminare. Per capire l'idea che sta alla base di tutto, considerate una semplice coda FIFO. Invece di aspettare che la coda si riempia completamente, essendo così costretti ad eliminare tutti i pacchetti in arrivo (la strategia di eliminazione terminale della Sezione 6.2.1), potremmo decidere di eliminare ciascun pacchetto in arrivo in base ad una qualche *probabilità di eliminazione*, ogni volta che la coda supera un certo *livello di eliminazione*. Questa idea viene chiamata *early random drop* (eliminazione casuale anticipata). L'algoritmo RED definisce nei dettagli relativi a come vada tenuta sotto controllo la lunghezza della coda e quando sia opportuno eliminare un pacchetto.

Nei paragrafi seguenti, descriviamo l'algoritmo RED così come è stato originariamente proposto da Floyd e Jacobson, pur segnalando che sono state proposte, in seguito, molte modifiche, sia dagli stessi inventori sia da altri ricercatori: alcune di queste sono discusse nella sezione "Ulteriori letture". Tuttavia, le idee principali sono quelle presentate qui e le implementazioni più recenti sono molto simili all'algoritmo che segue.

Dapprima RED calcola una lunghezza media della coda usando una media pesata corrente simile a quella usata dal calcolo delle temporizzazioni nel protocollo TCP originario. cioè AvgLen viene calcolato come

$$\text{AvgLen} = (1 - \text{Weight}) \times \text{AvgLen} + \text{Weight} \times \text{SampleLen}$$

dove $0 < \text{Weight} < 1$ e SampleLen è la lunghezza della coda in un istante di campionamento. Nella maggior parte delle implementazioni la lunghezza della coda viene misurata ogni volta che arriva un nuovo pacchetto, mentre in hardware potrebbe essere calcolata ad intervalli fissi.

Il motivo che porta ad utilizzare una lunghezza media delle code piuttosto che un valore istantaneo sta nel fatto che il primo rappresenta molto meglio il concetto di congestione. A causa della natura "a raffiche" del traffico di Internet, le code si possono riempire molto velocemente, per poi svuotarsi altrettanto rapidamente. Se una coda è vuota per la maggior parte del tempo, probabilmente non è corretto concludere che il router sia congestionato e chiedere agli host di rallentare. Di conseguenza, il calcolo della media pesata corrente cerca di identificare situazioni di congestione durature, come indicato nella parte destra della Figura 6.15, dove i cambiamenti di breve durata della lunghezza della coda vengono filtrati. Potete pensare alla media corrente come ad un filtro passa-basso, dove Weight determina la costante di tempo del filtro. In seguito discuteremo di come scegliere il valore di questa costante di tempo.

Il secondo punto riguarda il fatto che RED ha due soglie per la lunghezza della coda che innescano precise azioni: MinThreshold e MaxThreshold. Quando un pacchetto arriva al router, RED confronta il valore attuale di AvgLen con queste due soglie, secondo queste regole:

Se $\text{AvgLen} \leq \text{MinThreshold}$
→ accoda il pacchetto

Se $\text{MinThreshold} < \text{AvgLen} < \text{MaxThreshold}$
→ calcola la probabilità P
→ elimina, con probabilità P , il pacchetto in arrivo

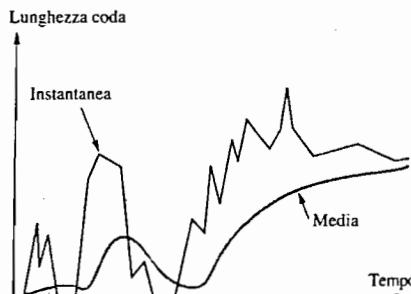


Figura 6.15 Media pesata corrente della lunghezza della coda.

6.4 Strategie per evitare la congestione

Se $\text{MaxThreshold} \leq \text{AvgLen}$
→ elimina il pacchetto in arrivo

In sintesi, se la lunghezza media della coda è inferiore alla soglia più bassa, non viene intrapresa alcuna azione, mentre se la lunghezza media della coda è superiore alla soglia più alta il pacchetto viene sempre eliminato. Se la lunghezza media della coda si trova fra le due soglie, allora il nuovo pacchetto in arrivo viene eliminato con una determinata probabilità, P . Tale situazione è rappresentata in Figura 6.16, mentre la Figura 6.17 presenta un'approssimazione della relazione esistente fra P e AvgLen. Notate che la probabilità di eliminazione di un pacchetto aumenta lentamente quando AvgLen si trova fra le due soglie, raggiungendo il valore MaxP in corrispondenza della soglia superiore, quando poi balza al valore unitario. La motivazione per questo comportamento sta nel fatto che quando AvgLen raggiunge la soglia superiore l'approccio "morbido" (cioè l'eliminazione di pochi pacchetti) non funziona più e si rendono necessarie misure più drastiche, che consistono nell'eliminazione di tutti i pacchetti in arrivo. Alcuni ricercatori hanno suggerito che sarebbe più appropriata una transizione più graduale fra l'eliminazione probabilistica e l'eliminazione completa, piuttosto dell'approccio discontinuo visto qui.

Anche se la Figura 6.17 mostra la probabilità di eliminazione come una funzione della sola variabile AvgLen, la situazione è un po' più complicata, perché P è, in realtà, una funzione di AvgLen e del tempo trascorso dalla precedente eliminazione di un pacchetto.

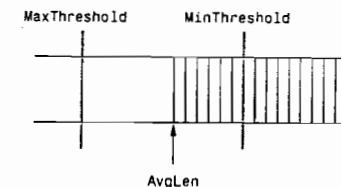


Figura 6.16 Soglie di RED in una coda FIFO.

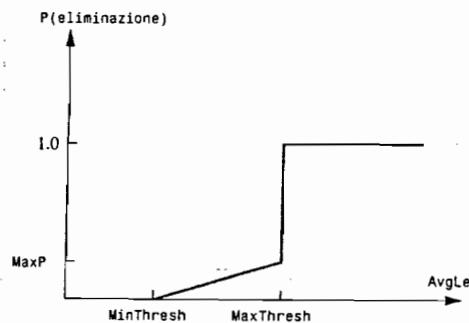


Figura 6.17 Funzione di probabilità dell'eliminazione di pacchetti in RED.

In particolare, viene calcolata come segue:

$$\begin{aligned} \text{TempP} &= \text{MaxP} \times (\text{AvgLen} - \text{MinThreshold}) / (\text{MaxThreshold} - \text{MinThreshold}) \\ P &= \text{TempP} / (1 - \text{count} \times \text{TempP}) \end{aligned}$$

TempP è la variabile che rappresenta l'asse y nella Figura 6.17, mentre count tiene traccia di quanti nuovi pacchetti in arrivo sono stati inseriti in coda (e, quindi, non eliminati) nel periodo in cui AvgLen ha assunto un valore all'interno dell'intervallo delimitato dalle due soglie. P aumenta lentamente all'aumentare di count , in modo che l'eliminazione di un pacchetto diventi sempre più probabile all'aumentare del tempo trascorso dall'ultima eliminazione avvenuta: in questo modo eliminazioni ravvicinate sono meno probabili di eliminazioni distanti nel tempo. Questo passaggio ulteriore nel calcolo di P fu introdotto dagli inventori del meccanismo RED dopo aver osservato che, senza di esso, le eliminazioni di pacchetti non erano ben distribuite nel tempo, ma tendevano a concentrarsi in aggregazioni (*cluster*). Dato che i pacchetti relativi ad una certa connessione tendono ad arrivare in rapida sequenza (*burst*), questo fenomeno di eliminazioni ravvicinate aumentava la probabilità di eliminare più pacchetti da un'unica connessione, una cosa non desiderabile, perché un'unica eliminazione in un periodo di tempo uguale a RTT è sufficiente a provocare una riduzione della finestra di una connessione, mentre più eliminazioni potrebbero riportarla in una situazione in cui si renda necessaria la fase di "partenza lenta".

Come esempio, supponete di impostare MaxP al valore 0.02 e di partire dal valore count uguale a 0. Se la lunghezza media della coda si trovasse a metà dell'intervallo fra le due soglie, allora TempP , e il conseguente valore iniziale di P , sarebbe uguale alla metà di MaxP , cioè 0.01: ovviamente, un pacchetto in arrivo ha 99 probabilità su 100 di essere inserito in coda. Ad ogni successivo pacchetto che non viene eliminato, si ha un piccolo aumento di P : dopo che sono arrivati 50 pacchetti e nessuno è stato eliminato, il valore di P è raddoppiato, portandosi a 0.02. Nell'evento, improbabile, che arrivino 99 pacchetti senza che avvenga alcuna eliminazione, P raggiunge il valore 1, garantendo così che il successivo pacchetto verrà eliminato. La cosa importante di questa parte dell'algoritmo è che garantisce, nel tempo, una distribuzione abbastanza uniforme delle eliminazioni.

Ovviamente, ci si auspica che, nel momento in cui RED elimina una piccola frazione di pacchetti (perché AvgLen ha superato MinThreshold), l'effetto sia quello di indurre alcune connessioni TCP a ridurre la propria finestra, cosa che, a sua volta, ridurrà la velocità di arrivo dei pacchetti nel router. Andando tutto bene, il valore di AvgLen diminuisce e si evita la congestione: si mantiene corta la lunghezza della coda, mentre il throughput rimane elevato a causa dei pochi pacchetti perduti.

Notate che, operando RED su una lunghezza della coda che viene mediata nel tempo, è possibile che i valori istantanei della lunghezza della coda siano molto più elevati di AvgLen . In questo caso, se arriva un pacchetto che non trova posto in coda, dovrà essere eliminato. Quando ciò accade, RED si trova ad operare in modalità di "eliminazione terminale" nella coda, anche se uno degli obiettivi di RED è quello di prevenire, per quanto possibile, questo comportamento.

La natura casuale di RED conferisce all'algoritmo un'interessante proprietà: eliminando pacchetti in modo casuale, la probabilità che RED decida di eliminare un pacchetto da un certo flusso è sostanzialmente proporzionale alla porzione di banda che il flusso sta sfruttando nel router, perché un flusso che sta inviando un numero relativamente grande di pacchetti sta anche fornendo più candidati per l'eliminazione casuale. Di conseguenza, RED ha una qualche cognizione di allocazione equa di risorse, anche se non in modo preciso.

6.4 Strategie per evitare la congestione

Noteate che è necessaria un'intensa attività di analisi per impostare i vari parametri di RED (ad esempio, `MaxThreshold`, `MinThreshold`, `MaxP` e `Weight`) con lo scopo di ottimizzare la funzione potenza (cioè il rapporto tra il throughput e il ritardo). Le prestazioni derivanti da queste configurazioni di parametri sono state anche confermate da simulazioni e si è visto che l'algoritmo non è troppo sensibile alle variazioni di tali parametri. Tuttavia, è importante ricordare che tutte queste analisi e simulazioni si basano su una particolare caratterizzazione del traffico di rete: il vero contributo di RED è un meccanismo mediante il quale il router può gestire la lunghezza della propria coda in modo più accurato. La definizione precisa di quale sia la lunghezza ottimale della coda dipende dalla combinazione di vari schemi di traffico ed è ancora argomento di ricerca, mentre si raccolgono informazioni realistiche dall'utilizzazione operativa di RED in Internet.

Considerate i valori delle due soglie, `MinThreshold` e `MaxThreshold`. Se il traffico è "a raffica", il valore di `MinThreshold` dovrebbe essere abbastanza elevato da consentire un livello accettabile di utilizzazione della linea. Ancora, la differenza tra le due soglie dovrebbe essere maggiore dell'aumento tipico della lunghezza media della coda che può accadere durante un

Tahoe, Reno e Vegas

Il nome "TCP Vegas" proviene dalla naturale prosecuzione delle prime implementazioni di TCP che vennero distribuite nella versione di Unix 4.3 BSD. Queste versioni erano note con i nomi Tahoe e Reno (che, come Las Vegas, si trovano in Nevada) e le versioni di TCP divennero famose con i nomi delle rispettive versioni di BSD. La versione TCP Tahoe, nota anche con il nome di BNR1, BSD Network Release 1.0, corrisponde all'implementazione originale del meccanismo di controllo della congestione di Jacobson, e contiene tutti i meccanismi descritti nella Sezione 6.3 tranne il recupero veloce. La versione TCP Reno, nota anche con il nome di BNR2, BSD Network Release 2.0, aggiunge il meccanismo di recupero veloce, insieme ad un'ottimizzazione nota come *previsione dell'intestazione (header prediction)*, che si occupa di ottimizzare il caso molto comune in cui i segmenti arrivano in ordine. La versione TCP Reno fornisce anche il supporto per le *conferme ritardate (delayed ACK)*, anche se si tratta di un'opzione che viene a volte disabilitata. Una versione più recente di TCP, distribuita all'interno della versione 4.4 BSD di Unix, aggiunge le estensioni di "finestre grandi" (*big windows*) descritte nella Sezione 5.2.

Dalla descrizione di questa linea evolutiva di TCP, dovreste aver capito che questo protocollo è stato piuttosto fluido negli ultimi anni, in special modo per quanto riguarda il suo meccanismo di controllo della congestione. In pratica, non esiste nemmeno un'opinione universalmente accettata che dica quali tecniche sono state introdotte in ciascuna versione, a causa del fatto che sono state rese disponibili versioni di codice intermedie e spesso si sono fatte modifiche a partire da altre modifiche.

Ciò che si può dire con relativa certezza è che due qualsiasi implementazioni di TCP che seguano le specifiche originarie, nonostante possano sicuramente operare insieme, non hanno necessariamente buone prestazioni. Identificare come l'interazione tra TCP Tahoe e TCP Reno possa influenzare le prestazioni è compito assai arduo. In altre parole, si potrebbe dire che il protocollo TCP non sia più definito da una specifica, ma da un'implementazione. L'unica domanda, a questo punto, è: quale implementazione?

RTT. Con gli schemi di traffico dell'odierna Internet, sembra ragionevole assegnare a MaxThreshold un valore doppio di MinThreshold. Dato che, poi, ci aspettiamo di vedere la lunghezza media della coda muoversi fra le due soglie durante i periodi di carico elevato, ci dovrebbe essere uno spazio sufficiente nei buffer *al di sopra* di MaxThreshold per assorbire le normali raffiche che accadono nel traffico di Internet senza costringere il router ad usare la modalità di eliminazione terminale.

Abbiamo notato in precedenza che Weight determina la costante di tempo per il filtro passa-basso che esegue la media corrente, suggerendoci così un'idea di come potremmo scegliere un suo valore ragionevole. Ricordate che RED cerca di mandare dei segnali ai flussi TCP eliminandone pacchetti durante i momenti di congestione. Supponete che un router elimini un pacchetto da una connessione TCP e che poi inoltri immediatamente alcuni pacchetti della stessa connessione. Quando tali pacchetti arrivano a destinazione, il ricevitore inizia ad inviare ACK duplicati al mittente, che, quando avrà visto un numero sufficiente di ACK duplicati, ridurrà la dimensione della propria finestra. In questo modo, deve trascorrere un tempo almeno uguale al tempo di round trip per quella connessione dal momento in cui il router elimina un pacchetto fino al momento in cui il router stesso inizia ad osservare qualche miglioramento in relazione a quella connessione, in termini di una ridotta dimensione di finestra. Probabilmente non ha molto senso che un router reagisca alla congestione su una scala temporale molto inferiore al valore del tempo di round-trip delle connessioni che lo attraversano: come abbiamo notato in precedenza, 100 ms è una stima abbastanza buona dei valori medi di round trip in Internet, per cui si dovrebbe scegliere il valore di Weight in modo che vengano filtrate eventuali modifiche della lunghezza della coda su una scala temporale molto inferiore a 100 ms.

Dal momento che RED funziona inviando segnali ai flussi TCP, dicendo loro di rallentare, vi potreste chiedere cosa accadrebbe se questi segnali venissero ignorati. Questo problema viene spesso chiamato *flusso inadempiente (unresponsive flow)* ed è stato un argomento che ha suscitato qualche interesse per alcuni anni. I flussi inadempienti usano una quantità di risorse di rete maggiore della "quota equa" e se ve ne fossero troppi si potrebbe assistere ad un collasso per congestione, proprio come nei giorni in cui TCP non aveva controllo di congestione. Alcune delle tecniche descritte nella Sezione 6.5 possono essere d'aiuto in questa situazione, isolando alcune classi di traffico dalle altre. Esiste anche la possibilità che una variante di RED possa eliminare più pacchetti dai flussi che risultano inadempienti in seguito ai suggerimenti iniziali che vengono inviati loro: si tratta di un'area in cui la ricerca è ancora attiva.

Concludiamo la nostra discussione in merito a RED considerando il problema più generale: quando è una buona idea eliminare pacchetti, prima di essere costretti a farlo in seguito al riempimento di una coda? Considerate una rete ATM, ad esempio: se inviate pacchetti AAL5 attraverso un commutatore ATM congestionato e questo è costretto ad eliminare una delle celle di un pacchetto, allora le altre celle dello stesso pacchetto saranno inutili per l'host terminale, perché quest'ultimo dovrà comunque chiedere la ritrasmissione dell'intero pacchetto AAL5. Ha senso, quindi, eliminare anche le altre celle, anche se il commutatore ha spazio sufficiente per memorizzarle nel proprio buffer: questa tecnica è stata, infatti, proposta e viene chiamata *eliminazione parziale di un pacchetto (PPD, partial packet discard)*. Si può rendere ancora più aggressivo il commutatore, combinando il meccanismo RED con l'idea di PPD: quando un commutatore ATM è prossimo alla congestione e arriva la prima cella di un nuovo pacchetto AAL5, viene eliminata tale cella e tutte le celle successive che appartengono allo stesso pacchetto AAL5. In questo modo viene eliminato l'intero pacchetto

e non soltanto la sua parte conclusiva: una tecnica denominata *eliminazione anticipata di un pacchetto (EPD, early packet discard)*. Anche se la tecnica EPD viene spesso confusa con il meccanismo RED, è importante notare che EPD è specifica della tecnologia ATM e che la decisione di eliminare un pacchetto viene solitamente presa con un algoritmo meno sofisticato di RED, reagendo all'occupazione istantanea dei buffer piuttosto che alla congestione di lungo periodo.

6.4.3 Impedire la congestione alla sorgente

Diversamente dai due schemi precedenti, che impediscono l'insorgere della congestione basandosi su nuovi meccanismi implementati nei router, descriviamo ora una strategia implementata negli host terminali per rilevare un imminente stato di congestione prima che avvengano perdite di pacchetti. Per prima cosa vediamo una panoramica di un insieme di meccanismi correlati che usano diverse informazioni per individuare i primi sintomi di congestione, poi descriveremo con maggiore dettaglio uno specifico meccanismo.

L'idea generale che sta alla base di queste tecniche consiste nell'osservazione di alcuni segnali che si evidenziano nella rete quando un router sta entrando in crisi e si avrà presto una congestione se non si prendono contromisure. Ad esempio, la sorgente potrebbe notare un aumento misurabile del valore di RTT per ogni successivo pacchetto che invia, mano a mano che le code di pacchetti si riempiono nei router della rete. Un particolare algoritmo sfrutta questa osservazione nel modo seguente: normalmente, la finestra di congestione aumenta con lo stesso meccanismo visto per il protocollo TCP, ma dopo ogni intervallo di durata uguale a due volte il tempo di round trip l'algoritmo verifica se l'attuale valore di RTT sia maggiore della media tra i valori minimo e massimo di RTT osservati fino a quel momento: se lo è, l'algoritmo diminuisce di un ottavo la dimensione della finestra di congestione.

Un secondo algoritmo si comporta in modo simile, basando la propria decisione di modificare o meno l'attuale dimensione della finestra sulle modifiche del valore di RTT e della finestra stessa. Ogni due periodi di round trip, la finestra viene modificata in base al prodotto:

$$(CurrentWindow - OldWindow) \times (CurrentRTT - OldRTT)$$

Se il risultato è positivo, la sorgente diminuisce di un ottavo la dimensione della finestra, altrimenti aumenta la dimensione della finestra di una quantità pari alla dimensione massima di un pacchetto. Notate che la finestra viene modificata in ogni caso, cioè oscilla attorno al proprio valore ottimale.

Un altro segnale visibile quando la rete si avvicina alla congestione è l'appiattimento della velocità di spedizione: un terzo meccanismo sfrutta questa osservazione. Ad ogni intervallo di durata RTT, la dimensione della finestra viene aumentata di un pacchetto e si confronta il throughput ottenuto in questo modo con il throughput precedente, quando la finestra era di un pacchetto più piccola. Se la differenza è inferiore alla metà del throughput che si otteneva avendo un solo pacchetto in transito, come accade all'inizio della connessione, allora l'algoritmo diminuisce di un pacchetto la dimensione della finestra. Questo schema calcola il throughput dividendo per RTT il numero di byte in transito nella rete.

Un quarto meccanismo, quello che descriveremo con maggiore dettaglio, è simile a quest'ultimo algoritmo per il fatto che si occupa di osservare le modifiche del throughput o, per essere più precisi, le modifiche della velocità di spedizione. Tuttavia, differisce dal terzo algoritmo per la modalità di calcolo del throughput e, invece di tener conto dei cambiamenti

di pendenza nel throughput, confronta il throughput misurato con un valore di throughput atteso. L'algoritmo, chiamato TCP Vegas, non è ancora molto diffuso in Internet, ma si continua a studiarne la strategia (si veda la sezione "Ulteriori Letture" per informazioni più complete).

L'intuizione che sta alla base dell'algoritmo Vegas si può riscontrare nei comportamenti del protocollo TCP standard rappresentati in Figura 6.18 (nel riquadro trovate spiegazioni in merito al nome Vegas). Il grafico superiore mostrato in Figura 6.18 contiene l'andamento della finestra di congestione della connessione: la stessa informazione fornita dai grafici visti precedentemente in questa sezione. I grafici centrale e inferiore rappresentano, invece, informazioni nuove: il grafico centrale mostra la velocità media di trasmissione misurata alla sorgente, mentre il grafico inferiore mostra la lunghezza media della coda misurata nel router.

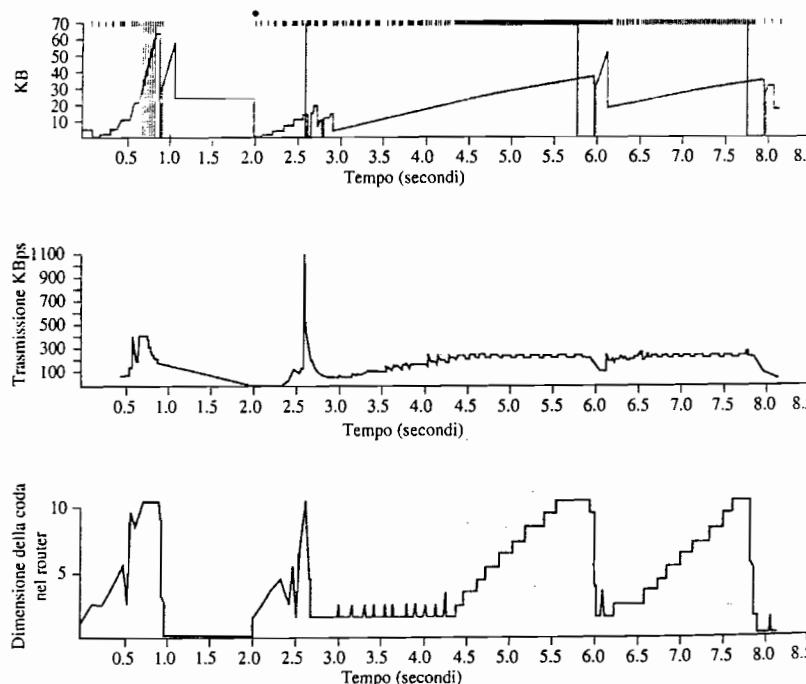


Figura 6.18 Finestra di congestione in rapporto alla velocità di throughput osservata (i tre grafici sono sincronizzati). In alto la finestra di congestione, al centro il throughput osservato e in basso lo spazio occupato nei buffer del router. La linea di spessore maggiore raffigura l'andamento del valore di CongestionWindow nel tempo; i pallini nella parte alta del grafico indicano gli istanti in cui scadono le temporizzazioni; i trattini verticali nella parte alta del grafico indicano le trasmissioni di pacchetti; le linee verticali che attraversano il grafico segnalano gli istanti in cui furono trasmessi per la prima volta i pacchetti che subiscono ritrasmissioni.

6.4 Strategie per evitare la congestione

che funge da collo di bottiglia. I tre grafici sono sincronizzati nel tempo. Nel periodo che va dai 4.5 ai 6.0 secondi (la zona ombreggiata), la finestra di congestione aumenta (come si vede nel grafico superiore). Ci aspetteremmo di veder aumentare anche il grafico del throughput, che invece rimane costante (grafico centrale), perché il throughput non può aumentare oltre l'ampiezza di banda disponibile: oltre tale punto, ogni aumento della dimensione della finestra va solamente ad aumentare lo spazio occupato nei buffer del router che funge da collo di bottiglia (grafico inferiore).

Un'utile metafora che descrive il fenomeno illustrato in Figura 6.18 è la guida di un'automobile sul ghiaccio. Il tachimetro (la finestra di congestione) può dire che state viaggiando a 30 miglia orarie, ma guardando fuori dal finestrino dell'automobile e vedendo persone a piedi che vi superano (velocità di trasmissione misurata) potete dedurre che non state viaggiando a più di 5 miglia orarie. L'energia supplementare viene assorbita dai pneumatici dell'automobile (i buffer del router).

Il meccanismo TCP Vegas sfrutta questa idea per misurare e controllare la quantità di dati aggiuntivi che la connessione mantiene in transito, dove per "dati aggiuntivi" intendiamo dati che la sorgente non avrebbe trasmesso se avesse cercato una piena corrispondenza con l'ampiezza di banda disponibile della rete. Ovviamente, se una sorgente sta inviando troppi dati aggiuntivi, provocherà grandi ritardi e forse porterà alla congestione. Meno ovviamente, se una connessione sta inviando una quantità troppo limitata di dati aggiuntivi, non potrà reagire abbastanza rapidamente a momentanei aumenti dell'ampiezza di banda disponibile nella rete. Le azioni intraprese dalla strategia TCP Vegas per impedire la congestione sono basate su modifiche nella quantità stimata di dati aggiuntivi presenti nella rete, e non solo sui pacchetti eliminati. Descriviamo ora l'algoritmo in dettaglio.

Innanzitutto, definiamo il valore BaseRTT di un flusso come il valore di RTT di un pacchetto in un momento in cui il flusso non è sottoposto a congestione. In pratica, TCP Vegas imposta il valore di BaseRTT al minimo di tutti i valori misurati per il tempo di round trip, che molto spesso è il valore di RTT del primo pacchetto inviato dalla connessione, perché la coda del router aumenta in seguito al traffico generato dal flusso stesso.

Se ipotizziamo che la connessione non abbia un flusso troppo elevato, allora ci possiamo aspettare che il throughput sia:

$$\text{ExpectedRate} = \text{CongestionWindow}/\text{BaseRTT}$$

dove CongestionWindow è la finestra di congestione di TCP, che assumiamo (in questa discussione) essere uguale al numero di byte in transito.

Successivamente, TCP Vegas calcola la velocità di trasmissione attuale per la sorgente, ActualRate , memorizzando l'istante di trasmissione di un particolare pacchetto e calcolando il numero di byte che vengono trasmessi tra quell'istante e il momento in cui viene ricevuta la conferma di quel pacchetto, calcolando contemporaneamente un campione di RTT per quello stesso pacchetto nel momento in cui ne arriva la conferma e dividendo il numero di byte trasmessi per quel campione di RTT. Questo calcolo viene eseguito una volta per ogni intervallo di tempo di durata eguale al tempo di round trip.

In una terza fase, TCP Vegas confronta i valori di ActualRate e ExpectedRate e modifica di conseguenza la finestra. Definiamo una variabile $\text{Diff} = \text{ExpectedRate} - \text{ActualRate}$, notando che è, per definizione, non negativa, dal momento che, se fosse $\text{ActualRate} > \text{ExpectedRate}$, dovremmo aggiornare BaseRTT al valore dell'ultimo campione di RTT. Definiamo anche due soglie, α e β , sostanzialmente corrispondenti alla situazione in cui si hanno,

rispettivamente, troppo pochi o troppi dati aggiuntivi in rete. Quando $\text{Diff} < \alpha$, TCP Vegas aumenta linearmente la finestra di congestione nel corso del successivo intervallo di tempo di durata RTT, mentre quando $\text{Diff} > \beta$ la finestra viene diminuita linearmente nello stesso intervallo di tempo. La finestra rimane immutata quando $\alpha < \text{Diff} < \beta$.

Dal punto di vista intuitivo, si può vedere che quanto più il throughput reale si allontana dal throughput atteso, tanto maggiore è la congestione nella rete, per cui è necessario ridurre la velocità di trasmissione della sorgente. La soglia β innesta questa diminuzione. D'altra parte, quando il throughput reale si avvicina troppo al throughput atteso, c'è il rischio che la connessione non stia utilizzando tutta l'ampiezza di banda disponibile: la soglia α innesta il corrispondente aumento della finestra. L'obiettivo è quello di avere sempre, nella rete, una quantità di byte aggiuntivi compresa tra α e β .

La Figura 6.19 mostra l'andamento dell'algoritmo TCP Vegas che impedisce la congestione. Il grafico superiore mostra l'andamento della finestra di congestione, la medesima informazione degli altri grafici visti nel capitolo. Il grafico inferiore mostra il throughput atteso e reale che provocano i cambiamenti della dimensione della finestra di congestione: è questo il grafico che meglio illustra il funzionamento dell'algoritmo. La linea superiore, di spessore maggiore, rappresenta il valore di `ExpectedRate`, mentre quella inferiore corrisponde ad `ActualRate`. L'ampia fascia grigia indica la regione situata tra le soglie α e β : il limite superiore della fascia dista α KBps da `ExpectedRate`, mentre il limite inferiore dista, sempre da `ExpectedRate`, β KBps. L'obiettivo è quello di mantenere `ActualRate` fra queste due soglie, cioè all'interno della regione grigia. Ogni volta che `ActualRate` scende al di sotto di tale regione (cioè si allontana troppo da `ExpectedRate`), TCP Vegas diminuisce la dimensione della finestra di congestione, nel timore che nella rete vengano tenuti troppi pacchetti nei buffer. Analogamente, ogni volta che `ActualRate` sale al di sopra della regione grigia (cioè si

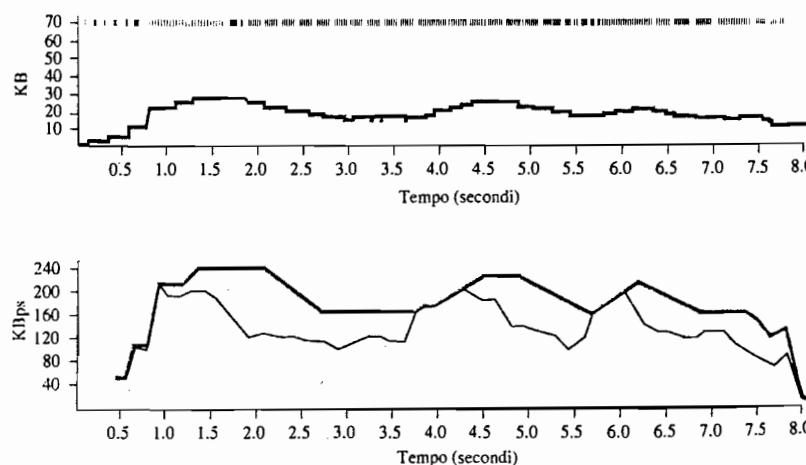


Figura 6.19 Andamento del meccanismo di TCP Vegas che impedisce la congestione. In alto la finestra di congestione, in basso il throughput atteso (linea di spessore maggiore) e il throughput reale. L'area in grigio è la regione che si trova tra le soglie α e β .

Valutare un nuovo meccanismo di controllo della congestione

Supponete di aver sviluppato un nuovo meccanismo di controllo della congestione e di volerne valutare le prestazioni. Ad esempio, potreste volerlo confrontare con il meccanismo attualmente utilizzato in Internet. Come si può procedere per effettuare le misure necessarie alla valutazione della nuova strategia? Anche se un tempo lo scopo primario di Internet era quello di agevolare le ricerche sulle reti di calcolatori, oggi Internet è un'enorme rete di produzione e, quindi, assolutamente inadatta all'esecuzione di un esperimento controllato.

Se il vostro approccio coinvolge soltanto i nodi terminali, ipotizzando soltanto che Internet abbia dei router con code FIFO, allora è possibile eseguire il meccanismo di controllo della congestione su un piccolo insieme di host e misurare il throughput che le vostre connessioni sono in grado di ottenere, anche se è d'uopo aggiungere alcune parole di cautela. È incredibilmente semplice ideare un meccanismo di controllo della congestione che fornisca un throughput che sia cinque volte superiore alla norma di quanto avviene per le connessioni TCP in Internet: basta semplicemente inviare a raffica i pacchetti in rete ad alta velocità, provocando così un fenomeno di congestione. Tutti gli altri host, che eseguono il protocollo TCP, rilevano la congestione e riducono la velocità a cui inviano i propri pacchetti, per cui il vostro meccanismo può utilizzare tranquillamente tutta la banda disponibile. Una strategia veloce, ma difficilmente la si potrà ritenere equa.

Eseguendo esperimenti direttamente in Internet, anche se si opera con cura, non sarà possibile valutare meccanismi di controllo della congestione che richiedano modifiche nei router. Semplicemente, non è possibile modificare il software in esecuzione su migliaia di router al solo scopo di valutare un nuovo meccanismo di controllo della congestione. In questo caso, i progettisti di reti sono costretti a collaudare i propri sistemi in reti simulate o in reti private. Ad esempio, i grafici del protocollo TCP presentati in questo capitolo sono stati generati mediante un'implementazione di TCP eseguita in un simulatore di rete. Ciò che è difficile, tanto in una simulazione quanto in una rete usata per il collaudo, è ideare una topologia e un traffico di carico che siano rappresentativi della vera Internet.

avvicina troppo a `ExpectedRate`), TCP Vegas aumenta la finestra di congestione, nel timore che la rete sia sottoutilizzata.

Poiché, come abbiamo appena spiegato, l'algoritmo confronta la differenza fra i valori di throughput atteso e reale con le soglie α e β , queste due soglie sono definite in KBps, anche se è forse più appropriato concepirle in termini della quantità di buffer aggiuntivi occupati nella rete dalla connessione. Ad esempio, in una connessione con `BaseRTT` uguale a 100 ms e pacchetti di 1 KB, se $\alpha = 30$ KBps e $\beta = 60$ KBps, possiamo immaginare che il valore di α esprima il fatto che la connessione deve impegnare, nella rete, almeno tre buffer aggiuntivi e che β stia a significare che la connessione non dovrebbe mai impegnare più di sei buffer aggiuntivi. In pratica, impostando α al valore corrispondente ad un buffer e β al valore di tre buffer si ottiene un buon funzionamento.

Infine, avrete notato che TCP Vegas diminuisce linearmente la finestra di congestione, apparentemente in conflitto con la regola enunciata in precedenza, che impone una diminuzione moltiplicativa per garantire la stabilità. Il motivo sta nel fatto che TCP Vegas non usa una diminuzione moltiplicativa quando si assiste alla scadenza di una temporizzazione, ma la

diminuzione lineare appena descritta è una diminuzione *anticipata* della finestra di congestione che, sperabilmente, accade prima che si verifichi la congestione e che i pacchetti iniziano ad essere eliminati.

6.5 Qualità di servizio

Per molti anni le reti a commutazione di pacchetto hanno promesso di fornire supporto alle applicazioni multimediali, che combinano dati con informazioni audio e video. Dopo tutto, una volta che siano state rese digitali, le informazioni audio e video diventano identiche ad ogni altra forma di dati: un flusso di bit da trasmettere. Un primo ostacolo al mantenimento di questa promessa è stato il bisogno di linee di collegamento con elevata ampiezza di banda. Tuttavia, di recente i miglioramenti delle codifiche hanno ridotto le ampiezze di banda necessarie per le applicazioni audio e video, mentre, al tempo stesso, la velocità delle linee è aumentata.

Va detto, però, che trasmettere audio e video in una rete richiede altro che fornire una sufficiente ampiezza di banda. Chi partecipa ad una conversazione telefonica, ad esempio, si aspetta di essere in grado di conversare in modo che una persona possa rispondere a qualcosa detto dall'altra persona e che quest'ultima senta immediatamente la risposta: di conseguenza, la puntualità di consegna può essere decisiva. Quando parliamo di applicazioni particolarmente sensibili alla puntualità dei dati usiamo l'espressione *applicazioni in tempo reale (real-time applications)*. Le applicazioni video e vocali tendono ad essere esempi canonici di applicazioni in tempo reale, ma ne esistono altre, come il controllo industriale: vorreste che un comando inviato ad un braccio meccanico gli sia consegnato prima che il braccio vada a sbattere contro un ostacolo. Anche le applicazioni di trasferimento di file possono avere vincoli di puntualità, ad esempio richiedendo che l'aggiornamento di una base di dati venga completata durante la notte, prima che, il giorno successivo, riprendano le attività lavorative che hanno bisogno di quei dati.

La caratteristica distintiva delle applicazioni in tempo reale è che necessitano di una qualche forma di garanzia dalla rete che i dati arriveranno in tempo con elevata probabilità (in base ad una qualche definizione di "arriveranno in tempo"). Mentre le applicazioni non in tempo reale possono usare una strategia di ritrasmissione fra nodi terminali per assicurarsi che i dati arrivino in modo corretto, tali strategie non garantiscono la puntualità, anzi, il tempo di ritrasmissione si aggiunge alla latenza totale, nel caso in cui i dati arrivino in ritardo. L'arrivo puntuale deve essere fornito dalla rete stessa (i router), non da ciò che si trova ai confini della rete (gli host). Possiamo quindi concludere che il modello best-effort, in cui la rete cerca di consegnare i dati ma non fa promesse e scarica sui confini della rete stessa il compito di sistemare le cose che non vanno a buon fine, non è sufficiente per le applicazioni in tempo reale. Ciò di cui abbiamo bisogno è un nuovo modello di servizio, nel quale le applicazioni che necessitano di maggiori garanzie possano chiederle alla rete. La rete può, poi, rispondere garantendo che farà quanto richiesto, oppure potrà dire che in quel determinato momento non è in grado di fornire garanzie adeguate. Notate che tale modello di servizio è comprensivo del modello attuale: quelle applicazioni che sono soddisfatte del modello best-effort potrebbero comunque usare il nuovo modello di servizio, con requisiti semplicemente meno restrittivi. Ciò implica che la rete tratterà alcuni pacchetti in modo diverso dagli altri, cosa che non viene fatta nel modello best-effort. Si dice spesso che una rete che sia in grado di fornire questi diversi livelli di servizio fornisce il supporto per la *qualità di servizio (QoS, quality of service)*.

6.5.1 Requisiti delle applicazioni

Prima di prendere in esame i vari protocolli e meccanismi che si possono usare per fornire alle applicazioni un'adeguata qualità di servizio, dovremmo cercare di capire quali siano le necessità di tali applicazioni. Per iniziare, possiamo catalogare le applicazioni in due tipi: in tempo reale e non in tempo reale. Le ultime vengono anche chiamate applicazioni "per dati tradizionali", perché sono state, tradizionalmente, le principali applicazioni sulle reti per dati. Fra queste, troviamo le applicazioni più comuni, come Telnet, FTP, posta elettronica, navigazione Web, e così via. Tutte queste applicazioni possono funzionare senza garanzie sulla puntualità di consegna dei dati. Un altro termine usato per caratterizzare questa classe di applicazioni non in tempo reale è *elastiche*, perché sono in grado di subire un rilassamento elastico, senza problemi particolari, in caso di aumentato ritardo. Notate che queste applicazioni possono comunque trarre beneficio da ritardi di più breve durata, ma non diventano inutilizzabili quando i ritardi aumentano. Notate anche che i requisiti di ritardo sono variabili per applicazioni interattive, come Telnet, e per applicazioni più asincrone, come la posta elettronica, con le applicazioni per ingenti trasferimenti di dati, come FTP, che si collocano nel mezzo.

Esempio di audio in tempo reale

Come esempio concreto di applicazione in tempo reale, considerate un'applicazione audio simile a quella illustrata in Figura 6.20. I dati vengono generati raccogliendo campioni da un microfono e resi digitali usando un convertitore analogico-digitale ($A \rightarrow D$). I campioni digitali vengono inseriti in pacchetti, che vengono trasmessi attraverso la rete e ricevuti all'altro capo. Nell'host ricevente i dati devono essere riprodotti (*play back*) ad una velocità adeguata. Ad esempio, se i campioni vocali sono stati raccolti alla velocità di un campione ogni 125 μs , dovrebbero essere riprodotti alla stessa velocità. Di conseguenza, possiamo immaginare che ogni campione sia caratterizzato da un proprio *istante di riproduzione (playback time)*, l'istante di tempo nel quale l'host ricevente necessita di avere il campione: nell'esempio vocale, ogni campione ha un istante di riproduzione ritardato rispetto al campione precedente di 125 μs . Se i dati arrivano dopo il proprio istante di riproduzione, perché ritardati nella rete o perché eliminati e successivamente ritrasmessi, sono essenzialmente inutili: è proprio questa totale inutilità dei dati in ritardo a caratterizzare le applicazioni in tempo reale. Nelle applicazioni elastiche, sarebbe meglio che i dati arrivassero in tempo, ma li possiamo usare anche in caso contrario.

Un modo per far funzionare l'applicazione vocale sarebbe quello di accertarsi che tutti i campioni richiedano esattamente lo stesso tempo per attraversare la rete: dato che i campioni vengono inseriti nella rete al ritmo di uno ogni 125 μs , essi arriveranno di conseguenza al ricevitore con il medesimo ritmo, pronti per essere riprodotti. Tuttavia, in generale è complicato garantire che tutti i dati che attraversano una rete a commutazione di pacchetto siano



Figura 6.20 Un'applicazione audio.

sottoposti esattamente al medesimo ritardo: i pacchetti attraversano le code di router e switch e le lunghezze di queste code variano nel tempo, dando luogo a ritardi che tendono a variare nel tempo essi stessi e, di conseguenza, sono potenzialmente diversi per ogni pacchetto del flusso audio. Il modo di gestire tutto ciò all'estremo ricevente consiste nella memorizzazione di una certa quantità di dati di riserva, in modo da avere sempre una scorta di pacchetti in attesa di essere riprodotti nel momento opportuno. Se un pacchetto arriva con un piccolo ritardo, viene inserito nel buffer finché non giunge il momento giusto per la sua riproduzione; se arriva, invece, con un ritardo più elevato, non ci sarà bisogno di memorizzarlo a lungo nel buffer del ricevitore prima di doverlo riprodurre. In questo modo abbiamo, a tutti gli effetti, aggiunto un valore costante all'istante di riproduzione di tutti i pacchetti, per avere una qualche garanzia di successo. Questo valore costante (*offset*) che viene aggiunto si chiama *punto di riproduzione (playback point)*. L'unico caso in cui incontriamo problemi si ha quando i pacchetti vengono ritardati dalla rete per un tempo così lungo da arrivare dopo il proprio istante di riproduzione, provocando lo svuotamento del buffer di riproduzione.

Il funzionamento di un buffer di riproduzione è illustrato in Figura 6.21. La diagonale più a sinistra mostra i pacchetti che vengono generati ad un ritmo costante; la linea curva mostra gli istanti di arrivo dei pacchetti, con un ritardo variabile rispetto all'istante in cui sono stati spediti, in relazione a ciò che hanno incontrato nella rete. La diagonale più a destra mostra i pacchetti che vengono riprodotti ad un ritmo costante, dopo aver atteso nel buffer di riproduzione per un certo periodo di tempo. A condizione che la linea relativa alla riproduzione sia sufficientemente a destra, la variazione del ritardo dovuto alla rete non verrà mai percepita dall'applicazione. Tuttavia, se spostiamo un po' a sinistra tale linea di riproduzione, alcuni pacchetti inizieranno ad arrivare troppo in ritardo per poter essere utilizzati.

Per la nostra applicazione audio esistono dei limiti oltre i quali non possiamo ritardare la riproduzione dei dati. È difficile sostenere una conversazione se il tempo che intercorre tra il momento in cui parlate e il momento in cui il vostro interlocutore vi sente è maggiore di 300 ms, per cui ciò che vogliamo dalla rete, in questo caso, è la garanzia che tutti i nostri dati arriveranno entro 300 ms. Se i dati arrivano in anticipo, li inseriamo nel buffer fino all'appropriato istante di riproduzione; se arrivano in ritardo, non ci servono e li ignoriamo.

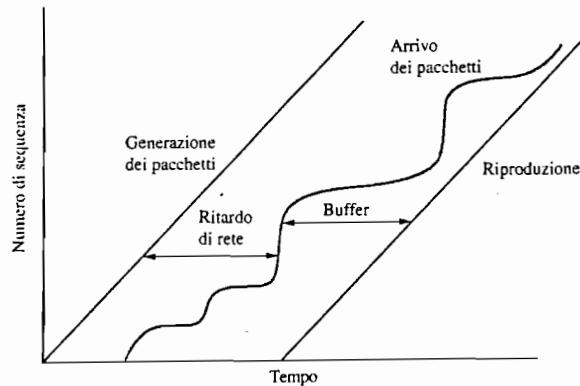


Figura 6.21 Un buffer di riproduzione.

Per apprezzare meglio quanto possa essere variabile il ritardo dovuto alla rete, la Figura 6.22 mostra il ritardo di sola andata misurato lungo un certo percorso attraverso Internet durante una certa giornata. Nonostante i numeri precisi possano variare in relazione al percorso e al giorno, il fattore chiave qui è la *variabilità* del ritardo, che si riscontra in maniera coerente per quasi tutti i percorsi e in qualsiasi momento. Come indicato dalle percentuali cumulative presenti nella parte alta del grafico, il 97% dei pacchetti, in questo caso, ha avuto una latenza di 100 ms o inferiore. Ciò significa che, se l'applicazione audio che abbiamo usato come esempio fosse stata impostata con un punto di riproduzione di 100 ms, allora, in media, 3 pacchetti ogni 100 sarebbero arrivati troppo tardi per essere utilizzabili. Una cosa importante da notare in merito a questo grafico è che la parte terminale della curva, che si estende verso destra, è molto lunga: per avere la garanzia che tutti i pacchetti arrivino in tempo, dovremmo impostare il valore del punto di riproduzione intorno ai 200 ms.

Tassonomia delle applicazioni in tempo reale

Ora che abbiamo un'idea concreta di come funzionino le applicazioni in tempo reale, possiamo prendere in esame alcune diverse categorie di applicazioni, allo scopo di motivare il nostro modello di servizio. La tassonomia che segue deve molto al lavoro di Clark, Braden, Shenker e Zhang, le cui pubblicazioni su questo argomento si possono trovare nella sezione "Ulteriori Letture" al termine di questo capitolo. La tassonomia delle applicazioni è riassunta nella Figura 6.23.

La prima caratteristica in base alla quale è possibile caratterizzare le applicazioni è la loro tolleranza rispetto alla perdita di dati, dove la "perdita" potrebbe avvenire perché un pacchetto arriva troppo tardi per essere riprodotto così come per le normali vicende della rete. Da un lato, un campione audio andato perduto può essere ricostruito per interpolazione dei campioni circostanti, con un'incidenza relativamente piccola sulla qualità dell'audio che viene percepita: la qualità degrada fino al punto in cui un discorso diviene incomprensibile soltanto quando vengono persi campioni in continuazione. D'altra parte, il programma di controllo di un robot è un classico esempio di applicazione in tempo reale che non può tollerare perdite di dati.

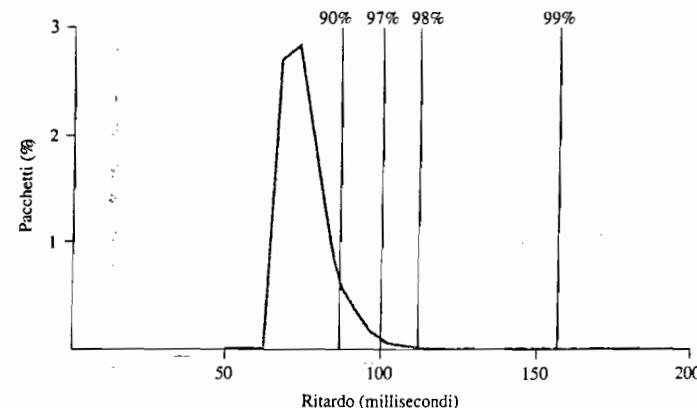


Figura 6.22 Esempio di distribuzione dei ritardi per una connessione Internet.

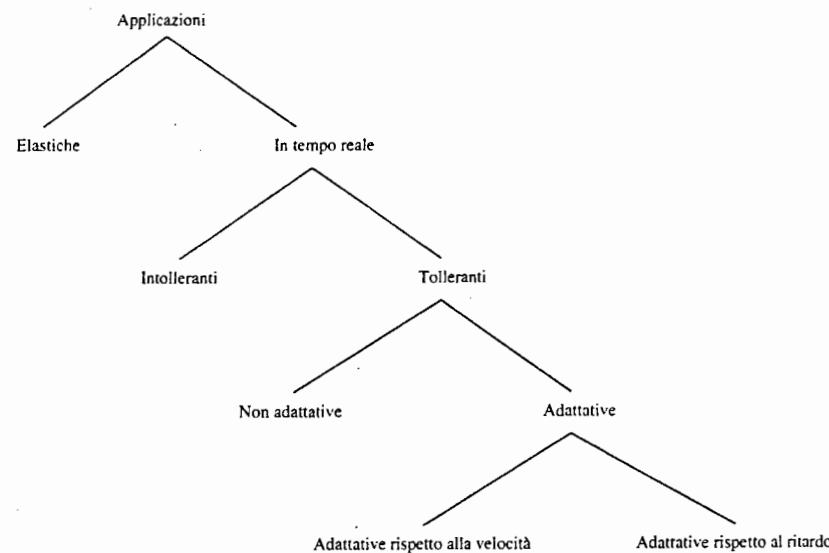


Figura 6.23 Tassonomia delle applicazioni.

rare perdite: non si può accettare la perdita di un pacchetto contenente il comando che provoca l'arresto di un braccio meccanico. Di conseguenza, possiamo catalogare le applicazioni in tempo reale come *tolleranti* e *intolleranti*, in relazione al fatto che possano, o meno, tollerare occasionali perdite di pacchetti (vale la pena di notare come molte applicazioni in tempo reale siano assai più tolleranti rispetto alle perdite occasionali di pacchetti rispetto ad applicazioni non in tempo reale: ad esempio, confrontate un'applicazione audio e l'applicazione FTP, dove la perdita di un solo bit, se non corretta, può rendere un file completamente inutilizzabile).

Un secondo modo per catalogare le applicazioni in tempo reale riguarda la loro capacità di adattamento. Ad esempio, un'applicazione audio potrebbe essere in grado di adattarsi al ritardo che affligge i pacchetti durante il loro attraversamento della rete. Se notiamo che i pacchetti arrivano quasi sempre entro 300 ms dal momento in cui sono stati spediti, allora possiamo impostare in modo adeguato il punto di riproduzione, memorizzando in un buffer tutti i pacchetti che arrivano in un tempo inferiore a 300 ms. Supponete che, in seguito, si osservi che tutti i pacchetti arrivano, in realtà, entro 100 ms: se spostiamo all'indietro il punto di riproduzione, portandolo a 100 ms, probabilmente gli utenti dell'applicazione percepiranno un miglioramento. Il processo di traslazione del punto di riproduzione richiede, in pratica, di riprodurre campioni ad una velocità superiore per un certo periodo di tempo, cosa che, in un'applicazione vocale, può essere fatta in modo appena percettibile, accorciando semplicemente le pause fra le parole. In questo caso, quindi, l'aggiustamento del punto di riproduzione è abbastanza semplice ed è stato implementato in modo efficace in molte applicazioni vocali come il programma di teleconferenza audio che prende il nome di vat. Notate che l'aggiustamento del punto di riproduzione può avvenire in entrambe le direzioni, ma tali

aggiustamenti provocano una distorsione del segnale riprodotto durante il periodo di aggiustamento e che gli effetti di questa distorsione dipendono molto dall'uso che l'utente terminale fa dei dati.

Osservate che, se calcoliamo il punto di riproduzione nell'ipotesi che tutti i pacchetti arrivino entro 100 ms e poi scopriamo che alcuni pacchetti arrivano, in realtà, con un po' di ritardo, saremo costretti ad eliminarli, cosa che non avremmo dovuto fare se avessimo mantenuto il punto di riproduzione a 300 ms. Dovremmo, quindi, modificare il punto di riproduzione soltanto quando questa procedura fornisce un vantaggio significativo e soltanto quando saremo ragionevolmente certi che il numero di pacchetti ritardatari rimarrà modesto, certezza che può derivare dall'osservazione di quanto avvenuto recentemente oppure da garanzie ricevute dalla rete.

Le applicazioni che possono modificare il proprio punto di riproduzione vengono chiamate *adattative rispetto al ritardo* (*delay-adaptive*). Un'altra categoria di applicazioni adattative sono quelle *adattative rispetto alla velocità* (*rate-adaptive*); ad esempio, molti algoritmi di codifica video sono in grado di seguire un compromesso tra la velocità dei bit e la qualità, per cui, se scopriamo che la rete è in grado di garantire una certa ampiezza di banda, possiamo impostare di conseguenza i parametri di codifica, per poi modificarli, migliorando la qualità, nel caso in cui si renda disponibile una maggiore ampiezza di banda.

Approcci per il supporto della qualità di servizio (QoS)

Considerando l'ampio spettro dei requisiti delle applicazioni, ciò di cui abbiamo bisogno è un modello di servizio assai ricco, che soddisfi le esigenze di ogni applicazione. Ciò conduce ad un modello di servizio che non abbia un'unica categoria (*best-effort*), ma fornisca il supporto a diverse classi, ciascuna disponibile a soddisfare i bisogni di alcuni insiemi di applicazioni. Andando verso questo obiettivo, siamo ora pronti ad esaminare alcuni approcci che sono stati sviluppati per fornire uno spettro di qualità di servizio, che possono essere suddivisi in due ampie categorie:

- approcci a grana fine (*fine-grained*), che forniscono QoS a singoli flussi o a singole applicazioni;
- approcci a grana grossa (*coarse-grained*), che forniscono QoS ad ampie categorie di dati o aggregazioni di traffico

Nella prima categoria troviamo i "Servizi Integrati" (*Integrated Services*), un'architettura di qualità di servizio sviluppata dalla IETF e spesso associata al protocollo RSVP (Resource Reservation Protocol). Nella seconda categoria si trovano i "Servizi Differenziati" (*Differentiated Services*), che nel momento in cui scriviamo sono in fase di standardizzazione in IETF. Nelle prossime due sottosezioni discuteremo, uno dopo l'altro, questi due approcci.

È ben noto che la tecnologia ATM fornisce un ricco insieme di capacità in merito alla qualità di servizio e viene normalmente considerata come appartenente alla categoria degli approcci a grana fine, poiché le risorse vengono associate ai singoli circuiti virtuali. Tuttavia, un uso molto diffuso di ATM riguarda l'interconnessione di router, i quali possono scegliere di inviare lungo un singolo circuito virtuale un'ampia aggregazione di traffico, per cui è possibile usare la tecnologia ATM anche per la qualità di servizio a grana grossa. Nella Sezione 6.5.4 discuteremo i dettagli della qualità di servizio in ATM.

Infine, l'inserimento del supporto per la qualità di servizio in una rete non rappresenta necessariamente tutto ciò che serve per il supporto delle applicazioni in tempo reale. Termi-

niamo la nostra discussione riassumendo ciò che gli host terminali potrebbero fare per fornire un miglior supporto ai flussi in tempo reale, indipendentemente da quanto si diffonderanno i meccanismi di QoS citati, come Integrated Services o Differentiated Services.

6.5.2 Integrated Services (RSVP)

Il termine "Integrated Services" (spesso indicato IntServ, per brevità) si riferisce ad uno studio prodotto da IETF intorno agli anni 1995-97. Il gruppo di lavoro IntServ ha sviluppato le specifiche per un certo numero di *classi di servizio*, progettate per soddisfare le necessità di alcuni dei tipi di applicazioni descritte precedentemente. Ha anche definito come poter usare il protocollo RSVP per effettuare prenotazioni usando tali classi di servizio e i paragrafi che seguono danno una panoramica di tali specifiche e dei meccanismi usati per implementarle.

Classi di servizio

Una delle classi di servizio è stata progettata per le applicazioni intolleranti, che richiedono l'arrivo puntuale di ogni pacchetto. La rete deve garantire che il massimo ritardo di un pacchetto abbia un certo valore e l'applicazione può, di conseguenza, impostare il proprio punto di riproduzione in modo che nessun pacchetto arriverà mai dopo il proprio istante di riproduzione (si fa l'ipotesi che i pacchetti arrivati in anticipo possano sempre essere gestiti da buffer). Si parla di questo servizio come di un *servizio garantito*.

Oltre al servizio garantito, IETF prese in considerazione diversi altri servizi, ma alla fine si concentrò su uno di essi per soddisfare le esigenze delle applicazioni tolleranti e adattative, un servizio noto con il nome di *carico controllato* e motivato dall'osservazione che le applicazioni esistenti in tale categoria venivano eseguite abbastanza bene in reti che non avessero un carico pesante. L'applicazione audio vat, ad esempio, modifica il proprio punto di riproduzione al variare del ritardo della rete e fornisce un segnale audio di qualità ragionevole fintantoché la frequenza di perdita di pacchetti rimane nell'ordine del 10%.

Lo scopo del carico controllato consiste nell'emulazione di una rete poco caricata per quelle applicazioni che richiedono il servizio, anche se la rete nella sua globalità può, in realtà, avere un carico pesante. Il trucco consiste nell'utilizzo di un meccanismo di gestione delle code come WFQ (si veda la Sezione 6.2) per isolare il traffico a carico controllato dal traffico rimanente, con qualche forma di controllo di ammissibilità, per limitare la quantità totale di traffico a carico controllato presente in una certa linea di collegamento in modo che il carico rimanga ragionevolmente basso. In seguito tratteremo con maggiore dettaglio il problema del controllo di ammissibilità.

Queste due classi di servizio sono, ovviamente, un sottoinsieme di tutte le classi che si potrebbero fornire: rimane da verificare, nel momento in cui Integrated Services diverrà diffuso, se queste due classi siano adeguate per soddisfare le necessità di tutti i tipi di applicazioni descritte in precedenza.

Panoramica dei meccanismi

Ora che abbiamo integrato il nostro modello di servizio best-effort con alcune nuove classi di servizio, il problema successivo è come implementare una rete che fornisca questi servizi alle applicazioni; questa sezione fornisce una panoramica dei meccanismi principali, ma ricordate, mentre leggete questa sezione, che i meccanismi che vi sono descritti stanno ancora per essere messi a punto dalla comunità dei progettisti di Internet. La cosa principale che dovete astrarre dalla discussione è una comprensione generale dei componenti coinvolti nel fornire il supporto al modello di servizio prima delineato.

6.5 Qualità di servizio

Innanzitutto, mentre con un servizio best-effort possiamo semplicemente dire alla rete che vogliamo inviare i nostri pacchetti e abbandonarli alla rete stessa, un servizio in tempo reale richiede di comunicare alla rete qualcosa di più rispetto al tipo di servizio che chiediamo. Possiamo fornire informazioni qualitative, come "usa un servizio a carico controllato", o quantitative, come "Necessito di un ritardo massimo di 100 ms". Oltre a descrivere ciò che vogliamo, abbiamo bisogno di comunicare alla rete che cosa stiamo per trasmettere, perché un'applicazione con piccola ampiezza di banda richiederà una minor quantità di risorse di rete rispetto ad un'applicazione con elevata ampiezza di banda. L'insieme di informazioni che vogliamo fornire alla rete prende il nome di *specifica di flusso (flowspec)*, un nome che prende spunto dall'idea che un insieme di pacchetti associati ad un'unica applicazione e che condividano requisiti comuni venga chiamato *flusso*, in modo coerente a quanto visto nella Sezione 6.1.

Secondariamente, quando chiediamo alla rete di fornirci un particolare servizio, la rete deve decidere se è realmente in grado di fornire quel servizio. Ad esempio, se 10 utenti chiedono un servizio in cui ciascuno di loro userà costantemente una capacità di linea di 2 Mbps, mentre condividono una linea di connessione con capacità di 10 Mbps, la rete dovrà dire di no a qualcuno. Il processo in base al quale si decide a chi dire di no viene detto *controllo di ammissibilità*.

Ci serve, poi, un meccanismo mediante il quale gli utenti della rete e i componenti della rete stessa si scambino informazioni, come le richieste di servizio, le specifiche di flusso e le decisioni sul controllo di ammissibilità. Nel mondo ATM si parla, a questo proposito, di *segnalazione*, ma questa parola ha diversi significati, per cui parleremo di *prenotazione di risorse*, che verrà attuata mediante un protocollo di prenotazione di risorse.

Infine, dopo che sono stati descritti i flussi e i relativi requisiti e sono state prese le decisioni sull'ammissibilità, gli switch e i router della rete devono soddisfare i requisiti dei flussi. Un aspetto chiave nel soddisfacimento di questi requisiti sta nella gestione del modo in cui, all'interno degli switch e dei router, i pacchetti vengono inseriti nelle code e se ne pianifica la trasmissione. Quest'ultimo meccanismo è la *pianificazione dei pacchetti (packet scheduling)*.

Specifiche di flusso

La specifica di un flusso si può scomporre in due parti: la descrizione delle caratteristiche di traffico del flusso (detta *TSpec*) e la descrizione del servizio richiesto alla rete (detta *RSpec*). La RSpec è molto specifica di ciascun servizio e relativamente semplice da descrivere. Ad esempio, per un servizio a carico controllato, la RSpec è banale: l'applicazione richiede, semplicemente, il servizio a carico controllato, senza alcun parametro aggiuntivo. Per un servizio garantito si dovrà specificare un obiettivo di ritardo o un suo limite superiore (in realtà, nella specifica del servizio garantito di IETF, non si indica un ritardo ma un'altra grandezza dalla quale può essere calcolato il ritardo).

La descrizione TSpec è un po' più complicata. Come è stato mostrato nel nostro esempio precedente, dobbiamo fornire alla rete sufficienti informazioni in merito all'ampiezza di banda usata dal flusso, per consentire che vengano prese decisioni di ammissibilità sensate. Tuttavia, per la maggior parte delle applicazioni l'ampiezza di banda non è un singolo numero, ma qualcosa che varia continuamente: un'applicazione video, ad esempio, genererà più bit al secondo quando la scena muta rapidamente piuttosto che quando è stabile. Supponete di avere 10 flussi che arrivano ad uno switch da porte d'ingresso diverse, ma che debbano uscire tutti verso la stessa linea a 10 Mbps; supponete anche che, in un intervallo di tempo sufficientemente lungo, ciascun flusso non invii più di 1 Mbps. Potreste pensare che questa

situazione non presenti problemi, ma, se si tratta di applicazioni con velocità di bit variabile, come i flussi video compressi, allora esse invieranno occasionalmente dati con velocità maggiore della velocità media. Se un numero sufficiente di sorgenti trasmette al di sopra della propria velocità media, la velocità totale con la quale i dati arrivano allo switch potrà essere maggiore di 10 Mbps, un eccesso di dati che verrà inserito nella coda prima di essere inviato lungo la linea di uscita. Più tempo durerà questa condizione, più lunga diventerà la coda, fino al punto che potrebbe divenire necessaria l'eliminazione di pacchetti e se anche ciò non dovesse accadere, i dati presenti in coda verrebbero ritardati. Se i pacchetti vengono ritardati per un tempo sufficiente, il servizio che è stato richiesto non verrà fornito.

In seguito discuteremo con precisione come gestire le code per controllare il ritardo ed evitare l'eliminazione dei pacchetti, ma qui notiamo che abbiamo bisogno di sapere come varia nel tempo l'ampiezza di banda delle nostre sorgenti. Una modalità per descrivere le caratteristiche di ampiezza di banda delle sorgenti viene chiamata filtro *token bucket*, che è descritto da due parametri: una velocità di token, r , e una profondità di bucket, B . Tale filtro funziona nel modo seguente: per poter inviare un byte, devo possedere un token; per poter inviare un pacchetto di lunghezza n , mi servono n token; inizio senza alcun token e li accumulo alla velocità di r token al secondo, senza poter accumulare più di B token. Ciò significa che posso inviare nella rete una raffica di B byte quanto velocemente voglio, ma in un intervallo sufficientemente lungo non posso inviare più di r byte al secondo. Si può facilmente capire come questa informazione sia molto utile per l'algoritmo che effettua il controllo di ammissibilità, quando si cerca di capire se la nuova richiesta di servizio possa essere accolta.

La Figura 6.24 mostra come si possa usare un filtro *token bucket* per caratterizzare le richieste di banda di un flusso. Per semplicità, ipotizzate che ogni flusso possa inviare dati sotto forma di singoli byte, invece che di pacchetti. Il flusso A genera dati alla velocità costante di 1 Mbps e può, quindi, essere descritto da un filtro token bucket con $r = 1$ Mbps e $B = 1$ byte. Ciò significa che riceve token alla velocità di 1 Mbps ma non può memorizzare più di un solo token, che spende immediatamente. Anche il flusso B può trasmettere ad una velocità media di 1 Mbps nel lungo termine, ma lo fa inviando alla velocità di 0.5 Mbps per 2 secondi e di 2 Mbps per 1 secondo. Dal momento che la velocità r del filtro token bucket è, in un certo senso, una velocità media nel lungo periodo, il flusso B può essere descritto da un token bucket con velocità uguale a 1 Mbps. Diversamente dal flusso A, però, il flusso B ha bisogno di un bucket con profondità B di almeno 1 MB, in modo da poter risparmiare token mentre sta inviando con velocità inferiore a 1 Mbps, per poterli usare quando deve trasmet-

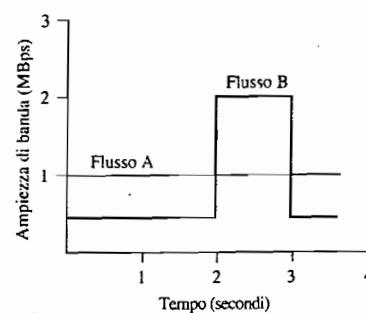


Figura 6.24 Due flussi con la stessa velocità media ma diverse descrizioni di tipo *token bucket*.

6.5 Qualità di servizio

tere a 2 Mbps. In questo esempio, per i primi 2 secondi riceve token alla velocità di 1 Mbps ma li spende soltanto alla velocità di 0.5 Mbps, per cui mette da parte $2 \times 0.5 = 1$ MB di token, da spendere nel terzo secondo (insieme ai nuovi token che continuerà a ricevere in quel secondo) per inviare dati alla velocità di 2 Mbps. Alla fine del terzo secondo, avendo speso i token in eccesso, ricomincerà a risparmiarne di nuovo, trasmettendo nuovamente alla velocità di 0.5 Mbps.

È interessante notare come un singolo flusso si possa descrivere con molte diverse combinazioni di token bucket. Come banale esempio, il flusso A potrebbe essere descritto dallo stesso filtro token bucket usato per il flusso B, con una velocità di 1 Mbps e un bucket di profondità ugualé a 1 MB. Il fatto che il flusso A non abbia mai bisogno di accumulare token non rende meno accurata questa descrizione, ma significa che non siamo riusciti a trasmettere alla rete questa utile informazione: il fatto che il flusso A ha, in realtà, requisiti di banda molto costanti nel tempo. In generale, è bene essere quanto più esplicativi possibile in merito alle necessità di banda di un'applicazione, per evitare l'allocazione di risorse di rete in eccesso.

Controllo di ammissibilità

L'idea del controllo di ammissibilità è semplice: quando un flusso vuole ricevere un particolare livello di servizio, il controllo di ammissibilità esamina le TSpec e RSpec del flusso e cerca di decidere se il servizio desiderato possa essere fornito per quella quantità di traffico, in base alle risorse attualmente disponibili, senza provocare un peggioramento di servizio per un flusso precedentemente ritenuto ammissibile. Se il servizio può essere fornito, il flusso viene ammesso, altrimenti viene rifiutato. La parte difficile è capire quando dire sì e quando dire no.

Il controllo di ammissibilità è molto dipendente dal tipo di servizio richiesto e dalla disciplina di gestione delle code usata nei router: parleremo di quest'ultimo argomento più avanti. Per un servizio garantito, abbiamo bisogno di un algoritmo preciso, per prendere decisioni ben definite: sì o no. La decisione è abbastanza semplice se ogni router usa la strategia di gestione equa e pesata delle code, come si è visto nella Sezione 6.2. Per un servizio a carico controllato la decisione si può basare su un approccio euristico, del tipo "l'ultima volta che ho ritenuto ammissibile un flusso con questa TSpec in questa classe, i ritardi per la classe sono andati oltre il limite accettabile, quindi è meglio dire no", oppure "i miei ritardi attuali sono talmente inferiori ai limiti che dovrei essere in grado di ammettere un altro flusso senza difficoltà".

Il controllo di ammissibilità non deve essere confuso con la *politica di gestione (policing)*. Il primo è la decisione, che viene presa per ogni flusso, di ammettere il flusso oppure no. La seconda è una funzione applicata ad ogni pacchetto per assicurarsi che un flusso sia conforme alla TSpec che aveva usato per effettuare la prenotazione. Se un flusso non risulta essere conforme alla propria TSpec, ad esempio perché sta inviando pacchetti ad una velocità doppia di quanto aveva dichiarato di voler fare, allora è probabile che interferisca con il servizio fornito ad altri flussi ed occorre prendere azioni correttive. A questo proposito esistono diverse opzioni praticabili, la più ovvia delle quali prevede di eliminare i pacchetti in eccesso. Un'altra opzione, però, potrebbe verificare se i pacchetti in eccesso interferiscono davvero con il servizio erogato agli altri flussi: se non interferiscono, i pacchetti potrebbero essere inoltrati lo stesso, dopo aver aggiunto un contrassegno che dice, in sostanza, "Questo pacchetto non rispetta le regole, se c'è bisogno di eliminare pacchetti questo va eliminato prima di altri".

Il controllo di ammissibilità è strettamente correlato all'importante argomento della *policy* (politica di gestione). Ad esempio, un gestore di rete potrebbe voler consentire l'ammissibilità delle prenotazioni effettuate dall'amministratore delegato dell'azienda, rifiutando le prenotazioni richieste da dipendenti di livello inferiore, anche se la richiesta dell'amministratore delegato può comunque fallire se non ci sono risorse disponibili: si vede, quindi, che quando si prendono decisioni in merito all'ammissibilità delle prenotazioni occorre tenere in considerazione sia la disponibilità di risorse sia la politica di gestione. Nel momento in cui scriviamo questo libro, l'applicazione delle politiche di gestione nelle reti è un'area che sta ricevendo molta attenzione.

Protocollo di prenotazione

Mentre le reti orientate alla connessione hanno sempre avuto bisogno di una qualche forma di protocollo per instaurare nei commutatori le informazioni di stato necessarie al funzionamento dei circuiti virtuali, le reti prive di connessione come Internet non hanno mai avuto tali protocolli. Come abbiamo notato in questa sezione, però, abbiamo la necessità di fornire parecchie informazioni alla rete quando vogliamo richiedere un servizio in tempo reale. Anche se per Internet sono stati proposti parecchi protocolli con questo scopo, quello su cui gravitano le maggiori attenzioni è il **Resource Reservation Protocol (RSVP)**, che è particolarmente interessante perché differisce in modo veramente sostanziale dai protocolli di segnalazione convenzionali usati nelle reti orientate alla connessione.

Una delle ipotesi chiave, sottostanti al protocollo RSVP, consiste nell'obiettivo di non togliere nulla alla robustezza attuale delle reti prive di connessione. Dato che le reti prive di connessione si basano sul fatto di non avere informazioni di stato (o averne pochissime) memorizzate all'interno della rete stessa, è possibile che i router interrompano bruscamente il proprio funzionamento e si riavvino, e che le linee di collegamento si guastino e poi riprendano la propria operatività, mentre la connettività tra nodi terminali rimane attiva. Il protocollo RSVP cerca di mantenere viva questa robustezza, usando il concetto di *stato soft nei router*. Lo stato soft, al contrario dello stato *hard* che si usa nelle reti orientate alla connessione, non ha bisogno di essere rimosso esplicitamente quando non è più necessario, perché scade dopo un periodo di tempo opportuno (ad esempio, un minuto) se non viene periodicamente aggiornato. In seguito vedremo come questo aiuti la robustezza.

Un'altra importante caratteristica di RSVP è l'obiettivo di fornire supporto ai flussi multicast con la stessa efficacia di quello fornito ai flussi unicast. Ciò non sorprende, dato che le applicazioni multicast attive su MBone, come vat e vic, sono state le prime candidate per beneficiare di servizi in tempo reale. Una delle scoperte dei progettisti di RSVP è che molte applicazioni multicast hanno molti più destinatari che mittenti, come esemplificato dal grande numero di ascoltatori e dall'unica persona che parla in una lezione tenuta sulla rete MBone. Ancora, i ricevitori possono avere requisiti diversi. Ad esempio, un ricevitore potrebbe voler ricevere dati da un'unica sorgente, mentre altri potrebbero voler ricevere dati da tutte le sorgenti. Piuttosto che lasciare al mittente il compito di tener traccia di un numero di destinatari potenzialmente elevato, è più sensato che siano i destinatari a gestire le proprie necessità. Questa osservazione suggerisce il nome di approccio *orientato ai destinatari (receiver-oriented)* che è stato adottato per RSVP. Al contrario, le reti orientate alla connessione lasciano solitamente alla sorgente il compito di prenotare le risorse, come avviene comunemente per chi effettua una chiamata telefonica, che provoca l'allocazione di risorse all'interno della rete telefonica. Lo stato soft e la natura orientata ai destinatari di RSVP genera un certo numero di proprietà interessanti. Una di queste proprietà consiste nel fatto che è molto semplice aumentare o

6.5 Qualità di servizio

diminuire il livello di allocazione di risorse fornito ad un ricevitore. Dato che ogni ricevitore invia periodicamente messaggi di aggiornamento per mantenere attivo lo stato soft, è facile inviare anche una nuova prenotazione che chieda un nuovo livello di risorse. Nel caso in cui un host interrompa bruscamente il proprio funzionamento, le risorse prenotate da quell'host per un determinato flusso andranno incontro ad una naturale scadenza e verranno liberate. Per capire cosa accade, invece, nel caso di guasto di un router o di una linea, dobbiamo guardare più da vicino al meccanismo di prenotazione.

Considerate inizialmente il caso di una sorgente e di un destinatario che cercano di ottenere una prenotazione per traffico che fluisce fra loro. Ci sono due cose che devono accadere prima che un destinatario possa fare la prenotazione. Prima di tutto, il destinatario ha bisogno di sapere che tipo di traffico verrà inviato dalla sorgente, in modo da poter fare una prenotazione appropriata; deve, cioè, conoscere la TSpec della sorgente. Secondariamente, deve sapere quale percorso seguiranno i pacchetti dalla sorgente al destinatario, in modo da effettuare una prenotazione in ogni router del percorso. Entrambi questi requisiti si possono soddisfare con l'invio, dalla sorgente al destinatario, di un pacchetto contenente la TSpec. In questo modo, ovviamente, il destinatario ottiene una copia della TSpec; l'altra cosa che accade è che ogni router esamina il messaggio (chiamato messaggio di tipo PATH, percorso) mentre lo inoltra e identifica il percorso a ritroso (reverse path) che verrà usato dal destinatario per inviare le prenotazioni verso la sorgente, nel tentativo di ottenere una prenotazione in ciascun router che si trova sul percorso. La costruzione dell'albero multicast avviene con meccanismi come quelli descritti nella Sezione 4.4.

Dopo aver ricevuto un messaggio PATH, il destinatario invia una prenotazione all'indietro contenuta in un messaggio RESV, "risalendo" l'albero multicast. Questo messaggio contiene la TSpec della sorgente e una RSpec che descrive le richieste di questo destinatario. Ogni router lungo il percorso esamina la richiesta di prenotazione e cerca di allocare le risorse necessarie per soddisfarla. Se la prenotazione può essere effettuata, la richiesta RESV viene trasferita al router successivo, altrimenti si restituisce un messaggio d'errore al destinatario che aveva effettuato la richiesta. Quando tutto va a buon fine, la prenotazione è correttamente memorizzata in ogni router che si trova lungo il percorso che va dalla sorgente al destinatario. Fintantoché il destinatario vuole mantenere attiva la prenotazione, invia il medesimo messaggio RESV ogni 30 secondi circa.

Possiamo ora vedere cosa accade quando si guasta un router o una linea. I protocolli di instradamento si adatteranno al guasto e creeranno un nuovo percorso dalla sorgente al destinatario. I messaggi PATH vengono inviati ogni 30 secondi circa e possono essere inviati anche prima di tale scadenza se un router identifica un cambiamento nella propria tabella di inoltro, per cui il primo di tali messaggi che verrà inviato dopo che il nuovo percorso si sarà stabilizzato raggiungerà il destinatario lungo il nuovo percorso. Il successivo messaggio RESV inviato dal destinatario seguirà il nuovo percorso e (auspicabilmente) stabilirà una nuova prenotazione lungo tale nuovo percorso. Nel frattempo i router che non si trovano più sul percorso utilizzato non riceveranno più i messaggi RESV periodici, liberando le risorse corrispondenti. In questo modo RSVP gestisce piuttosto bene i cambiamenti di topologia, almeno se le modifiche di instradamento non sono eccessivamente frequenti.

Il successivo problema che dobbiamo affrontare riguarda la gestione del multicast, dove possono esistere più sorgenti che inviano ad un gruppo composto da più destinatari, una situazione illustrata dalla Figura 6.25. Per prima cosa affrontiamo il problema di più destinatari e una sola sorgente. Mentre un messaggio RESV risale l'albero multicast, può darsi che incontri un settore dell'albero dove è già stata effettuata la prenotazione di qualche altro

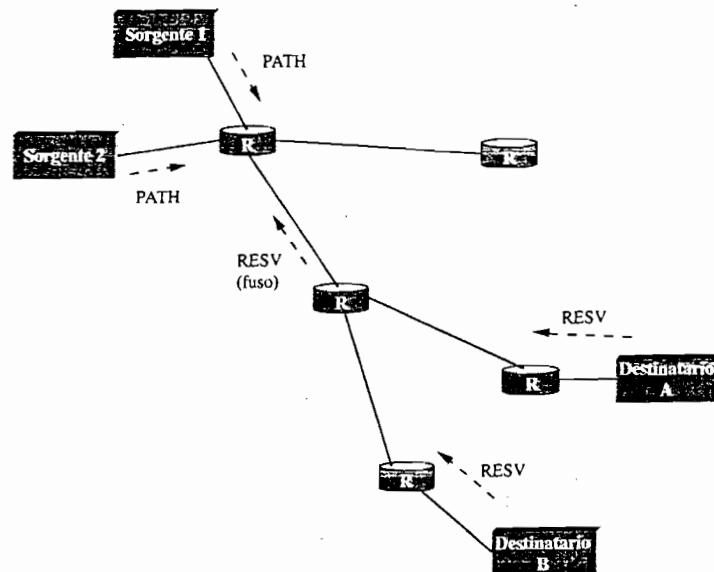


Figura 6.25 Effettuare prenotazioni lungo un albero multicast.

destinatario e potrebbe accadere che le risorse prenotate da quel punto in poi siano sufficienti a servire entrambi i destinatari. Ad esempio, se il destinatario A ha già effettuato la prenotazione che gli fornisce un ritardo garantito inferiore a 100 ms e la nuova richiesta che proviene dal destinatario B riguarda un ritardo inferiore a 200 ms, allora non è necessaria alcuna nuova prenotazione. D'altra parte, se la nuova richiesta riguardasse un ritardo inferiore a 50 ms, allora per prima cosa il router dovrebbe verificare se si tratta di una richiesta ammissibile e, in tal caso, dovrebbe inviare la richiesta verso l'alto. La volta successiva in cui A dovesse chiedere un ritardo minimo di 100 ms, non ci sarà bisogno di inoltrare la richiesta verso l'alto. In generale, le richieste si possono fondere in questo modo per soddisfare le richieste di tutti i destinatari che si trovano al di sotto del punto di fusione.

Se nell'albero esistono anche sorgenti multiple, i destinatari devono raccogliere le TSpec da tutte le sorgenti ed effettuare una prenotazione che sia sufficiente ad accogliere il traffico proveniente da tutte le sorgenti, anche se può darsi che questo non richieda la somma di tutte le TSpec. Ad esempio, in un'audioconferenza di 10 oratori, non ha molto senso allocare risorse a sufficienza per trasportare 10 flussi audio, perché il risultato di 10 persone che parlano contemporaneamente sarebbe incomprensibile. Di conseguenza, potremmo immaginare una prenotazione che sia sufficiente ad ospitare due oratori e non di più. Il calcolo della TSpec complessiva che corrisponde alle TSpec di tutte le sorgenti è, ovviamente, un problema specifico per ciascuna applicazione. Ancora, potremmo essere interessati a ricevere dati soltanto da un sottoinsieme di tutti i possibili oratori: RSVP consente diversi "stili" di prenotazione, per gestire opzioni del tipo "Prenota risorse per tutte le sorgenti", "Prenota risorse per n sorgenti qualsiasi" oppure "Prenota risorse soltanto per le sorgenti A e B".

6.5 Qualità di servizio

"Integrated Services" e le tecnologie di rete

Uno dei problemi da affrontare nell'estensione del modello di servizio best-effort di IP consiste nel fatto che IP deve potersi eseguire in reti di qualsiasi tecnologia. Il modello best-effort di IP fu adottato proprio perché costituisce il minimo comun denominatore dei servizi offerti da tutte le reti. Poiché IntServ va oltre questo modello, occorre capire come questo nuovo modello di servizio possa operare nelle più varie tecnologie di rete. Per risolvere questo problema, IETF ha creato un gruppo di lavoro chiamato *Integrated Services over Specific Link Layers* (ISSLL).

Lo strato di linea di collegamento più semplice su cui operare è la linea punto-punto, perché una linea di questo tipo ha sempre delle caratteristiche di qualità di servizio fisse e prevedibili. Due tra le tecnologie di rete più complesse (e importanti) sono, invece, Ethernet e ATM. Nel caso di ATM, il gruppo ISSLL ha specificato come i vari parametri usati nelle specifiche di flusso di IntServ vadano trasformati nei parametri specifici per la qualità di servizio dei circuiti virtuali di ATM, una trasformazione non troppo complessa (nonostante i numerosi dettagli da prendere in esame) perché IntServ e la qualità di servizio in ATM presentano parecchie similitudini (come visto nella Sezione 6.5.4).

Il caso di Ethernet è complicato dal fatto che l'accesso alla banda della linea di collegamento è deciso in modo completamente decentralizzato. Ciò significa, ad esempio, che nessun router può determinare se sia sicuro ritenere ammissibile una nuova prenotazione di flusso in un segmento Ethernet, perché qualche altro router può aver già fatto la stessa cosa per un flusso che richiede l'intera banda della linea. Per risolvere questo problema, il gruppo ISSLL definisce un'entità centralizzata con la responsabilità di decidere quali dispositivi, fra i tanti, possano richiedere prenotazioni. Tale entità è chiamata SBM (subnet bandwidth manager). Una parte della specifica del SBM considera il processo necessario per eleggere automaticamente un dispositivo che funga da SBM designato (DSBM) per una rete Ethernet, perché più nodi potrebbero essere in grado di svolgere tale compito ma è importante che vi sia un unico gestore in carica in un certo istante. Notate che esiste un limite alla gestione della banda effettuata da un SBM in una Ethernet, perché c'è sempre il rischio che nodi che stiano semplicemente inviando pacchetti in modalità best effort (e che, quindi, non abbiano alcun bisogno di colloquiare con l'SBM) possano occupare una frazione di banda sufficiente ad impedire che i flussi che hanno effettuato prenotazioni ricevano la qualità di servizio promessa loro dall'SBM. Questo problema può essere parzialmente attenuato tenendo sotto osservazione l'utilizzo della rete Ethernet da parte di traffico di tipo best effort e facendo previsioni ragionevoli sul suo andamento nel tempo.

Classificazione e pianificazione dei pacchetti

Dopo aver descritto il traffico e il servizio di rete desiderato e aver effettuato adeguate prenotazioni in tutti i router che si trovano lungo il percorso, l'unica cosa che rimane da fare è che i router forniscono effettivamente il servizio richiesto ai pacchetti di dati. Per questo, occorre fare due cose:

- Associare a ciascun pacchetto la prenotazione corretta, in modo che possa essere gestito in modo adeguato: un procedimento noto come *classificazione* dei pacchetti.

- Gestire i pacchetti nelle code in modo che ricevano il servizio che è stato richiesto, un procedimento che prende il nome di *pianificazione* dei pacchetti.

La prima parte viene implementata esaminando cinque campi del pacchetto: l'indirizzo della sorgente, l'indirizzo del destinatario, il numero di protocollo, la porta di sorgente e la porta di destinazione (in IPv6 si può usare il campo FlowLabel dell'intestazione per consentire la ricerca basata su una singola, più breve chiave). In base a queste informazioni, il pacchetto può essere classificato nel modo corretto. Ad esempio, si può decidere che appartenga alla classe del carico controllato, oppure che faccia parte di un flusso garantito che deve essere gestito separatamente da tutti gli altri flussi garantiti. In breve, esiste una corrispondenza tra le informazioni relative al flusso che sono contenute nell'intestazione del pacchetto e un identificativo univoco di classe che determina come gestire il pacchetto in coda. Per i flussi garantiti, questa corrispondenza può essere di tipo uno a uno, mentre per altri servizi potrebbe essere di tipo molti a uno. I dettagli della classificazione sono strettamente correlati ai dettagli della gestione delle code.

Dovrebbe risultare evidente che una semplice coda FIFO in un router non sarà adeguata a fornire molti diversi servizi e a fornire diversi livelli di ritardo nell'ambito di ciascun servizio. Nella Sezione 6.2 sono state presentate discipline di gestione delle code molto più sofisticate ed è probabile che in un router si usi qualche combinazione di esse.

I dettagli della pianificazione dei pacchetti non dovrebbero essere specificati nel modello di servizio, perché si tratta di un'area dove i realizzatori possono cercare di essere creativi per realizzare in modo efficiente il modello di servizio. Nel caso del servizio garantito, si è stabilito che una gestione della coda equa e pesata, in cui ciascun flusso abbia la propria coda con una certa frazione della capacità della linea, è in grado di fornire un limite al ritardo tra nodi terminali che può essere calcolato facilmente. Per il carico controllato si possono usare schemi più semplici. Una possibilità prevede di trattare tutto il traffico a carico controllato come un singolo flusso aggregato (per quanto riguarda il solo meccanismo di pianificazione), determinando il peso per tale flusso in base alla quantità totale di traffico ammessa nella classe a carico controllato. Il problema si complica se si considera che in un singolo router si possono fornire diversi servizi contemporaneamente e che ciascuno di tali servizi può richiedere un diverso algoritmo di pianificazione. Di conseguenza, serve un algoritmo di gestione globale delle code per gestire le risorse nell'ambito di servizi diversi.

Problemi di scalabilità

Anche se l'architettura Integrated Services e il protocollo RSVP hanno rappresentato un significativo miglioramento del modello di servizio best-effort di IP, molti fornitori di servizio Internet hanno ritenuto che non si trattasse del modello giusto da installare. Il motivo di questa reticenza è relativo ad uno degli obiettivi di progetto fondamentali del protocollo IP: la scalabilità. Nel modello di servizio best-effort, i router della rete memorizzano poche informazioni di stato (o nessuna informazione, addirittura) al riguardo dei singoli flussi che li attraversano, per cui, al crescere di Internet, l'unica cosa che devono fare i router per stare al passo con tale crescita è movimentare una maggior quantità di bit al secondo e gestire tabelle di instradamento di dimensioni maggiori. Il protocollo RSVP, invece, ammette la possibilità che ogni flusso che attraversa un router richieda una corrispondente prenotazione. Per capire quanto sia grave questo problema, supponete che ogni flusso di una linea OC-48 (a 2.5 Gbps) rappresenti un flusso audio a 64 Kbps. Il numero di tali flussi è

$$2.5 \times 10^9 / (64 \times 10^3) = 39000$$

Ognuna delle relative prenotazioni richiede una certa quantità di informazioni di stato che devono essere memorizzate e periodicamente aggiornate, il router deve catalogare ciascuno di questi flussi, applicare la relativa politica di gestione e manipolare di conseguenza la coda, le decisioni relative al controllo di ammissibilità vanno prese ogni volta che uno di tali flussi invia una richiesta di prenotazione e serve qualche meccanismo che "faccia pressione" sugli utenti in modo che non effettuino prenotazioni troppo ampie per periodi di tempo troppo lunghi⁵.

Questi problemi di scalabilità hanno impedito, nel momento in cui scriviamo, un'ampia diffusione di IntServ e sono stati sviluppati altri approcci che non richiedono una quantità così elevata di informazioni di stato per ogni flusso, approcci che verranno discussi nella sezione successiva.

6.5.3 Differentiated Services (EF, AF)

Mentre l'architettura Integrated Services assegna risorse ai singoli flussi, il modello Differentiated Services (spesso chiamato DiffServ, per brevità) assegna risorse ad un numero limitato di classi di traffico. In realtà, alcuni degli approcci proposti per DiffServ dividono semplicemente il traffico in due classi. Si tratta di un approccio molto sensato: se considerate la difficoltà che hanno gli operatori di rete a garantire il semplice servizio best-effort in una internetwork, è ragionevole aggiungere opzioni al modello di servizio operando per piccoli incrementi.

Supponete di aver deciso di migliorare il modello di servizio best-effort aggiungendo una nuova classe, che chiameremo "premium": ovviamente, ci sarà bisogno di un modo per identificare quali pacchetti sono premium e quali sono normali pacchetti di tipo best-effort. Invece di usare un protocollo come RSVP per comunicare ai router che determinati flussi stanno per inviare pacchetti premium, sarebbe molto più semplice se i pacchetti stessi lo comunicassero al router al momento del loro arrivo, cosa che, ovviamente, si potrebbe realizzare usando un bit dell'intestazione del pacchetto: se tale bit vale 1 il pacchetto è di tipo premium, altrimenti è di tipo best-effort. Tenendo a mente tutto ciò, dobbiamo risolvere due problemi:

- Chi imposta il bit di tipo premium, e in quali circostanze?
- Cosa fa di diverso un router quando vede un pacchetto di tipo premium?

Per la prima domanda esistono molte risposte possibili, ma un approccio comune consiste nell'impostare il bit in corrispondenza di un confine di gestione della rete: ad esempio, un router che si trova al confine della rete di un fornitore di servizi Internet potrebbe impostare il bit premium per i pacchetti in arrivo ad una certa interfaccia che lo connette alla rete di una particolare azienda, magari perché quell'azienda paga per un livello di servizio migliore del servizio best-effort. È anche possibile che non tutti i pacchetti vengano contrassegnati come premium: ad esempio, si potrebbe configurare il router in modo che contrassegni pacchetti come premium fino ad una certa velocità massima, lasciando che tutti i pacchetti ulteriori rimangano di tipo best-effort.

⁵ Far pagare le prenotazioni potrebbe essere un modo per "fare pressione". coerentemente con il modello di tariffazione telefonica che fa pagare ogni singola chiamata, ma non è l'unico modo per fare pressione, anche perché la tariffazione per singola chiamata è ritenuto uno dei principali costi operativi della rete telefonica.

Ipotizzando che i pacchetti siano stati contrassegnati in qualche modo, come si comportano i router che elaborano pacchetti premium? Anche qui ci sono diverse possibilità. In effetti, il gruppo di lavoro Differentiated Services di IETF sta standardizzando un insieme di comportamenti che i router possono tenere nei confronti di pacchetti dotati di tale contrassegno, che vengono chiamati "comportamenti per hop" (PHB, *per-hop behavior*), un termine che indica la definizione di un comportamento tenuto dai singoli router piuttosto che un servizio svolto fra nodi terminali. Essendoci più nuovi comportamenti di questo tipo, c'è anche bisogno di più di un bit nell'intestazione del pacchetto, che dica ai router quale comportamento tenere. La scelta di IETF è caduta sul vecchio byte *TOS* dell'intestazione IP, che non è mai stato usato, ridefinendolo. Sei bit di tale byte sono stati assegnati ai *code point* di DiffServ (DSCP, *Differentiated Services code point*), ciascuno dei quali identifica un particolare PHB da applicare al pacchetto.

Uno dei PHB più semplici da spiegare è noto come "inoltro accelerato" (EF, *expedited forwarding*). I pacchetti contrassegnati per il comportamento EF dovrebbero essere inoltrati dai router con il minimo ritardo e la minima probabilità di eliminazione. L'unica condizione che consente ai router di garantire ciò a tutti i pacchetti EF è che la loro velocità di arrivo presso il router sia rigidamente limitata ad un valore inferiore alla velocità con la quale il router può inoltrare i pacchetti EF. Ad esempio, un router con un'interfaccia a 100 Mbps deve avere la certezza che i pacchetti destinati a tale interfaccia non superino mai la velocità di 100 Mbps, e potrebbe anche voler esser certo che tale velocità sia un po' inferiore a quel valore, in modo da poter, di tanto in tanto, avere il tempo di inviare altri pacchetti (ad esempio, aggiornamenti di instradamento).

La limitazione in velocità dei pacchetti EF si ottiene configurando i router che si trovano ai confini di un dominio di gestione, in modo che impongano una velocità massima di ingresso ai pacchetti EF che entrano nel dominio. Un approccio semplice, ancorché conservativo, sarebbe quello di garantire che la somma delle velocità di tutti i pacchetti EF che entrano nel dominio sia inferiore all'ampiezza di banda della linea di collegamento più lenta del dominio stesso. Ciò garantirebbe che, anche nel caso peggiore in cui tutti i pacchetti EF volessero convergere sulla linea più lenta, essa non verrebbe sovraccaricata e potrebbe comportarsi correttamente.

Esistono diverse possibili strategie di implementazione per il comportamento EF. Una di queste consiste nell'assegnare ai pacchetti EF una priorità rigidamente superiore a quella di tutti gli altri pacchetti. Un'altra strategia prevede di suddividere in code eque e pesate i pacchetti EF e gli altri pacchetti, con il peso della coda EF sufficientemente elevato da fare in modo che tutti i pacchetti EF vengano consegnati velocemente. Questo ha un vantaggio rispetto alla priorità rigida: anche ai pacchetti che non siano di tipo EF si garantisce una certa probabilità di accesso alla linea, anche nel caso in cui il traffico di tipo EF sia eccessivo. In questo caso può darsi che non si riesca a rispettare il comportamento previsto per tutti i pacchetti EF, ma si evita anche che il traffico essenziale all'instradamento venga escluso dalla rete qualora il traffico di tipo EF sia eccessivo.

Un'altra strategia per-hop viene chiamata "inoltro garantito" (AF, *assured forwarding*). Un comportamento che ha origine da un approccio noto con il nome di "RED dentro e fuori" (RIO, *RED with In and Out*) oppure "RED pesato", due miglioramenti all'algoritmo RED di base visto nella Sezione 6.4.2. La Figura 6.26 mostra il funzionamento di RIO; in analogia con la Figura 6.17, vediamo la probabilità di eliminazione lungo l'asse y, che aumenta all'aumentare della lunghezza media della coda, rappresentata lungo l'asse x. Ora, però, abbiamo due diverse curve di probabilità di eliminazione per le due classi di traffico, che, in RIO, vengono denominate classi "interna" ("in") ed "esterna" ("out") per motivi che verranno

6.5 Qualità di servizio

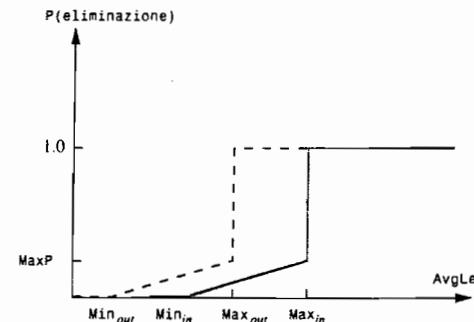


Figura 6.26 Strategia RIO: RED con probabilità di eliminazione di "in" e di "out".

chiari a breve. Dato che la curva "out" ha un valore di *MinThreshold* minore di quello della curva "in", è chiaro che, con bassi livelli di congestione, verranno eliminati dall'algoritmo RED solamente i pacchetti contrassegnati "out". Se la congestione diventa più significativa, viene eliminata una percentuale più elevata di pacchetti "out" e, se la lunghezza media della coda inizia a superare *Min_{in}*, RED inizia ad eliminare anche pacchetti "in".

Il motivo per cui le due classi di pacchetti vengono chiamate "in" e "out" deriva dal modo in cui avviene la marcatura dei pacchetti stessi. Abbiamo già fatto notare che sono i router che si trovano ai confini di un dominio amministrativo a contrassegnare i pacchetti: possiamo immaginare che un router di questo tipo si trovi al confine fra la rete di un fornitore di servizi di rete e un cliente di tale fornitore. Il cliente potrebbe essere un'altra qualsiasi rete, ad esempio la rete di un'azienda o di un altro fornitore di servizi di rete. Il cliente e il fornitore di servizi di rete si accordano su un certo profilo di inoltro garantito (e forse il cliente pagherà il fornitore per questo specifico profilo), che potrebbe assomigliare a "Il cliente X può inviare fino a y Mbps di traffico con inoltro garantito", oppure potrebbe essere specificato in modo più complesso. Indipendentemente dal profilo, il router di confine può facilmente identificare i pacchetti che arrivano da quel particolare cliente come appartenenti ("in") al profilo, oppure no ("out"). Nell'esempio appena citato, fino a quando il cliente invia meno di y Mbps tutti i suoi pacchetti saranno contrassegnati come "in"; non appena supera tale velocità, i pacchetti in eccesso verranno contrassegnati come "out".

La combinazione di un "misuratore di profilo" al confine e della strategia RIO in tutti i router della rete del fornitore di servizi può dare al cliente una garanzia elevata (ma non totale) che i pacchetti che rientrano nel profilo saranno consegnati. In particolare, se la maggior parte dei pacchetti, compresi quelli inviati da clienti che non hanno pagato per avere un profilo, sono di tipo "out", allora solitamente la strategia RIO agirà in modo da mantenere la congestione a livelli sufficientemente bassi da dover raramente eliminare pacchetti di tipo "in". Ovviamente ci deve essere, nella rete, ampiezza di banda a sufficienza perché accada raramente che i soli pacchetti di tipo "in" siano in grado di congestionare una linea fino al punto in cui la strategia RIO preveda l'eliminazione di pacchetti di tipo "in".

Proprio come in RED, l'efficacia di un meccanismo come RIO dipende, in qualche misura, da scelte corrette per i parametri, e in RIO ci sono molti più parametri. Al momento in cui scriviamo, c'è ancora un intenso dibattito in merito a come possa esattamente funzionare questo schema in reti reali.

Una proprietà interessante di RIO consiste nel fatto che non viene modificata la sequenzialità dei pacchetti "in" e "out". Ad esempio, se una connessione TCP sta inviando pacchetti attraverso un misuratore di profilo ed alcuni suoi pacchetti vengono contrassegnati come "in", mentre altri vengono contrassegnati come "out", questi ultimi avranno probabilità di eliminazione diverse nelle code dei router, ma arriveranno a destinazione nello stesso ordine in cui sono stati inviati, una cosa importante per la maggior parte delle implementazioni di TCP, che funzionano molto meglio quando i pacchetti arrivano nella sequenza corretta, anche se sono state progettate per gestire l'ordinamento scorretto. Notate anche che meccanismi quali la ritrasmissione veloce possono essere erroneamente innescati da arrivi fuori sequenza.

L'idea di RIO si può generalizzare, usando più di due curve di probabilità di eliminazione: questa idea è alla base dell'approccio che prende il nome di RED pesato (WRED, *weighted RED*). In questo caso il valore del campo DSCP viene usato per scegliere una curva di probabilità, in modo da poter fornire diverse classi di servizio.

Un terzo modo per realizzare Differentiated Services consiste nell'utilizzo del valore di DSCP per determinare in quale coda vada inserito un pacchetto, usando un gestore di code eque e pesate come quello descritto nella Sezione 6.2.2. Come caso semplificato potremmo usare un *code point* per indicare la coda del traffico "best-effort" e un secondo *code point* per selezionare la coda del traffico "premium". Poi, dobbiamo scegliere un peso per la coda premium che garantisca ai pacchetti premium un servizio migliore di quello dei pacchetti best-effort: questo dipende dal carico offerto dai pacchetti premium. Ad esempio, se assegniamo alla coda premium il peso 1 e alla coda best-effort il peso 4, siamo conseguentemente sicuri che l'ampiezza di banda disponibile per i pacchetti premium è

$$B_{\text{premium}} = W_{\text{premium}} / (W_{\text{premium}} + W_{\text{best-effort}}) = 1/(1+4) = 0.2$$

Abbiamo, quindi, riservato il 20% della linea ai pacchetti premium: se il carico offerto dal traffico premium è soltanto il 10% della linea, in media, allora il traffico premium si comporterà come se si trovasse in una rete assai poco carica ed il servizio sarà ottimo. In particolare, si può contenere il ritardo dei pacchetti appartenenti alla classe premium, perché in questa situazione la strategia WFQ cercherà di trasmettere i pacchetti premium non appena arriveranno. D'altra parte, se il carico del traffico premium fosse del 30%, si comporterebbe come se si trovasse in una rete con un carico molto elevato ed il ritardo dei pacchetti premium potrebbe essere molto alto, anche peggiore di quello dei pacchetti di tipo best-effort. Di conseguenza, per questo tipo di servizio è di vitale importanza conoscere il carico offerto e impostare con attenzione i pesi. Tuttavia, notate che essere molto conservativi nell'impostare i pesi per la coda premium è un approccio sicuro: se questo peso viene scelto di valore molto elevato in relazione al carico atteso, fornisce un margine di errore pur non impedendo al traffico di tipo best-effort di sfruttare la banda riservata al traffico premium quando questo non la utilizza. Proprio come in WRED, possiamo generalizzare questo approccio basato su WFQ in modo da avere più di due classi rappresentate da diversi *code point*. Possiamo, inoltre, combinare l'idea di un selettori di code con l'eliminazione preferenziale. Ad esempio, con 12 *code point* possiamo avere quattro code di pesi diversi, ciascuna delle quali ha tre preferenze di eliminazione: esattamente ciò che ha fatto IETF nella definizione di "servizio garantito" (*assured service*).

6.5 Qualità di servizio

6.5.4 Qualità di servizio in ATM

Le caratteristiche della qualità di servizio fornita dalle reti ATM sono, per molti aspetti, simili a quelle fornite da una rete IP che usi Integrated Services. Tuttavia, gli enti di standardizzazione di ATM hanno prodotto un totale di cinque classi di servizio, in confronto alle tre di IETF⁶. Le cinque classi di servizio di ATM sono:

- velocità di bit costante (CBR, *constant bit rate*)
- velocità di bit variabile, in tempo reale (VBR-rt, *variable bit rate – real-time*)
- velocità di bit variabile, non in tempo reale (VBR-nrt, *variable bit rate – non-real-time*)
- velocità di bit disponibile (ABR, *available bit rate*)
- velocità di bit non specificata (UBR, *unspecified bit rate*)

La maggior parte delle classi di servizio di ATM e di IP sono abbastanza simili, tranne una, ABR, che non ha una vera e propria controparte in IP. Di questa classe ci occuperemo in seguito, mentre le classi rimanenti sono di facile comprensione nei termini di ciò che avete già appreso.

Notate che nella tecnologia ATM la qualità di servizio viene definita nel momento in cui si instaura un circuito virtuale, inserendo opportune informazioni nei messaggi di segnalazione che vengono inviati per instaurare, appunto, un circuito virtuale. La classe VBR-rt è molto simile al servizio garantito di IP nella versione Integrated Services. I parametri specifici che vengono usati per impostare un circuito virtuale con VBR-rt sono un po' diversi da quelli usati per effettuare una prenotazione di servizio garantito, ma l'idea di base è la stessa. Il traffico generato dalla sorgente è caratterizzato mediante un filtro *token bucket*, specificando anche il massimo ritardo totale di attraversamento della rete che si richiede.

La classe CBR non è molto diversa da VBR-rt, tranne per il fatto che si presume che le sorgenti di traffico CBR trasmettano a velocità costante. Si tratta, in verità, di un caso particolare di VBR, con la velocità di trasmissione massima e media della sorgente che assumono il medesimo valore. La ragione principale per cui questa costituisce una classe separata in ATM è che questo caso particolare è di estrema importanza per le compagnie telefoniche, dato che la maggior parte dei servizi che oggi offrono (ad esempio, chiamate vocali e linee noleggiate) forniscono all'utente un canale con ampia banda fissa. Si è anche visto che CBR risulta essere un servizio facile da specificare e implementare, per cui molti dei primi commutatori ATM fornivano solamente il supporto per CBR ma non per VBR. La pronta disponibilità di CBR nei prodotti ATM ha certamente aiutato la diffusione della tecnologia nel mercato, soprattutto se si pensa che questi prodotti sono comparsi ben prima che fossero disponibili router P con una qualsiasi caratteristica di QoS di cui si potesse discutere.

La classe VBR-nrt assomiglia vagamente al servizio a carico controllato di IP. Di nuovo, la sorgente di traffico viene specificata mediante un filtro *token bucket*, ma non c'è una rigida garanzia di ritardo che viene fornita da VBR-rt o dal servizio garantito di IP.

La classe UBR è il servizio best-effort di ATM, anche se c'è una piccola differenza fra UBR e il modello standard del servizio best-effort fornito da IP. Dato che ATM richiede

⁶ Tra le classi di servizio, contiamo il servizio best-effort, oltre al carico controllato e al servizio garantito.

RSVP e ATM

Ora che abbiamo visto le caratteristiche salienti della qualità di servizio di RSVP e di ATM, è interessante confrontare i due approcci. Notate che, ragionando ad alto livello, gli obiettivi di RSVP sono identici a quelli di un protocollo di segnalazione orientato alla connessione: instaurare informazioni di stato nei nodi della rete che inoltrano i pacchetti, in modo che la gestione dei pacchetti avvenga in modo corretto. Tuttavia, al di là di tale obiettivo di alto livello, non ci sono molte altre somiglianze.

La Tabella 6.1 confronta il protocollo RSVP con l'attuale protocollo di segnalazione ATM Forum, che è derivato da protocollo Q.2931 di ITU-T (ricordate dalla Sezione 3.3 che Q.2931 definisce come venga instradato un circuito virtuale attraverso la rete, oltre a come riservare le risorse ad esso necessarie). Le differenze derivano principalmente dal fatto che RSVP parte da un modello privo di connessione e cerca di aggiungere funzionalità senza arrivare alle connessioni tradizionali, mentre ATM parte da un modello orientato alla connessione. Anche l'obiettivo di RSVP, di gestire in modo efficiente il multicast, è evidente: l'approccio orientato al destinatario cerca di gestire in modo scalabile i gruppi multicast con un gran numero di destinatari.

Tabella 6.1 Confronto del protocollo RSVP e della segnalazione ATM

RSVP	ATM
Il destinatario effettua la prenotazione	La sorgente genera la richiesta di connessione
Stato soft (aggiornamento/temporizzazione)	Stato hard (cancellazione esplicita)
Non correlato all'instaurazione del percorso	Concomitante all'instaurazione del percorso
La qualità di servizio può modificarsi dinamicamente	La qualità di servizio è immutabile per la durata della connessione (anche se in qualche modo può variare in ABR)
Destinatari diversi possono avere diverse QoS	Destinatari diversi hanno la stessa QoS

sempre una fase di segnalazione prima di poter inviare dati, è possibile veicolare informazioni relative alla sorgente durante l'instaurazione di un circuito virtuale. UBR consente alla sorgente di specificare la velocità massima alla quale trasmetterà, che può essere soltanto un valore inferiore alla velocità della linea. I commutatori possono sfruttare questa informazione per decidere se il nuovo circuito virtuale possa danneggiare circuiti preesistenti e, di conseguenza, rifiutare l'instaurazione del nuovo circuito o tentare di negoziare con la sorgente un'velocity massima inferiore.

Veniamo, infine, alla classe ABR, che è qualcosa in più di una semplice classe di servizio: definisce anche un insieme di meccanismi per il controllo di congestione. Essendo stata progettata da un ente di standardizzazione, è piuttosto complessa, per cui questa sezione ne presenta solamente i punti salienti.

Un circuito virtuale ATM ha, ovviamente, due terminali estremi, che possiamo identificare come sorgente e destinazione, anche se i circuiti virtuali sono solitamente bidirezionali. Per cui un nodo che funge da sorgente in una direzione è la destinazione nell'altra. I meccanismi di ABR operano su un circuito virtuale scambiando fra sorgente e destinazione speciali

6.5 Qualità di servizio

celle ATM chiamate celle di *gestione delle risorse* (RM, resource management). L'obiettivo dell'invio di celle RM consiste nel recupero di informazioni sullo stato della congestione nella rete, che vengono restituite alla sorgente in modo che possa, poi, inviare traffico alla velocità opportuna. Da questo punto di vista, le celle RM sono un meccanismo di controllo della congestione a *feedback esplicito*, simile a DECbit, diversamente da quanto avviene con l'uso del *feedback implicito* nel protocollo TCP, che si accorge della congestione in base alla perdita di pacchetti.

Inizialmente la sorgente invia alla destinazione una cella, inserendovi l'informazione relativa alla velocità a cui vorrebbe spedire celle di dati. I commutatori lungo il percorso esaminano la velocità richiesta e decidono se sono disponibili risorse sufficienti per consentire tale velocità, basandosi sulla quantità di traffico presente negli altri circuiti. Se sono disponibili risorse a sufficienza, la cella RM viene trasmessa senza modifiche, altrimenti la velocità richiesta viene diminuita prima di trasmettere la cella. Giunta a destinazione, la cella viene rispedita al mittente, che quindi viene a conoscenza della velocità a cui può inviare dati.

Lo scopo di ABR è quello di consentire ad una sorgente di aumentare o diminuire la propria velocità consentita, in relazione alle condizioni operative; di conseguenza, vengono inviate periodicamente celle RM, che possono contenere una richiesta di velocità più o meno elevata. Ancora, la velocità a cui una sorgente può trasmettere diminuisce nel tempo se non viene utilizzata, per scoraggiare le richieste del tipo "non si sa mai che possa servire".

Anche se abbiamo ipotizzato che la sorgente e la destinazione delle celle RM siano i due estremi del circuito virtuale, non è necessario che sia così. ABR estende il concetto di sorgente e destinazione, introducendo la nozione di sorgente virtuale (VS, *virtual source*) e destinazione virtuale (VD, *virtual destination*), che sono "virtuali" nel senso che non sono i veri estremi del circuito virtuale. Ciò consente di abbreviare il percorso ciclico di controllo lungo il quale fluiscono le celle RM rispetto al circuito virtuale stesso: in questo modo il tempo di risposta del sistema può essere ridotto. Inoltre, questo expediente può ridurre lo spazio necessario nei buffer dei commutatori, riducendo il tempo che trascorre dal momento in cui questi iniziano ad entrare in congestione al momento in cui la sorgente (virtuale) rallenta. La Figura 6.27 mostra un circuito virtuale che è stato sezionato in due parti, mediante una sorgente e una destinazione virtuali. La vera sorgente invia celle RM al commutatore avente la capacità di agire da VS/VD, il quale, agendo come una destinazione virtuale, restituisce le celle RM alla sorgente, inserendovi la velocità alla quale vuole accettare traffico per il circuito virtuale in oggetto. Assumendo, poi, il ruolo di sorgente virtuale, invia celle RM verso la vera destinazione, la quale a sua volta le restituirà alla sorgente virtuale, indicando la velocità alla quale quest'ultima potrà inviare traffico lungo il circuito virtuale.

C'è molta flessibilità nel modo in cui un commutatore può implementare effettivamente la classe di servizio ABR. Solitamente vengono usati algoritmi proprietari per modificare i valori della velocità nelle celle RM mentre queste attraversano i commutatori, in base ad una

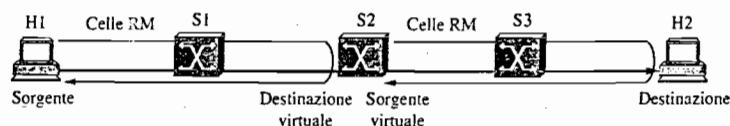


Figura 6.27 Circuito virtuale di classe ABR con percorsi ciclici segmentati per mezzo di una sorgente e una destinazione virtuali

gran varietà di informazioni, tra cui l'attuale occupazione dei buffer, la velocità di arrivo delle celle misurata per tutti i circuiti virtuali e le rispettive velocità consentite (che possono essere diverse da quelle misurate). Questi algoritmi, come è normale per gli algoritmi di controllo della congestione, cercano di rendere massimo il throughput, mantenendo a valori bassi il ritardo e la quantità di pacchetti perduti.

Un aspetto poco chiaro di ABR consiste nella sua interazione con i meccanismi del protocollo TCP che evitano la congestione. Queste strategie sono regolate in modo particolare in base all'esperienza e funzionano in base al presupposto che nella rete non stia accadendo nulla di strano: se vengono inviati troppi pacchetti, alcuni vengono eliminati. Con ABR potete immaginare che la sorgente virtuale debba eliminare pacchetti mentre sta attendendo il ritorno di una cella RM, anche se in realtà potrebbe non esserci alcuna congestione nella rete. Nel momento in cui scriviamo, la maggior parte degli esperimenti con ABR sono stati condotti in condizioni di laboratorio o mediante simulazione, con esperienze assai limitate nel mondo reale.

6.5.5 Controllo di congestione basato su equazioni

Terminiamo la nostra discussione sulla qualità di servizio tornando al controllo di congestione del protocollo TCP, questa volta, però, nel contesto delle applicazioni in tempo reale. Ricordate che TCP modifica la finestra di congestione della sorgente (e, quindi, la velocità a cui questa può trasmettere) in risposta a messaggi di conferma (ACK) e a scadenze di temporizzazioni. Uno dei punti di forza di questo approccio consiste nel fatto che non richiede la cooperazione dei router della rete: è una strategia puramente basata sugli host. Una strategia di questo tipo costituisce un complemento ai meccanismi di qualità di servizio che abbiamo finora considerato, perché oggi le applicazioni possono usare soluzioni basate sugli host, finché i meccanismi relativi alla qualità di servizio non saranno installati in modo diffuso; anche quando DiffServ sarà installato in tutta la rete, sarà ancora possibile che la coda di un router subisca troppe prenotazioni e vorremo che le applicazioni in tempo reale sappiano reagire in modo ragionevole in tale situazione.

Anche se vorremmo sfruttare l'algoritmo di controllo di congestione di TCP, il protocollo TCP non è adeguato per le applicazioni in tempo reale. Un primo motivo di ciò sta nell'affidabilità del protocollo TCP, che richiede ritrasmissioni, i cui ritardi sono spesso incompatibili con le necessità delle applicazioni in tempo reale. Tuttavia, cosa succederebbe se disaccoppiassemo il protocollo TCP dal suo meccanismo di controllo della congestione, cioè aggiungessimo un controllo di congestione simile a quello di TCP ad un protocollo non affidabile come UDP? Potremmo chiamare "UDP con controllo di congestione" (UDP+CC) il risultato. Le applicazioni in tempo reale potrebbero usare UDP-CC?

Da un lato si tratta di un'idea attraente, perché porterebbe le applicazioni in tempo reale a competere in modo equo con i flussi TCP. L'alternativa (come accade oggi) è che le applicazioni video usino UDP senza alcuna forma di controllo di congestione e, di conseguenza, sottraggano banda ai flussi TCP, che si fanno da parte in presenza di congestione. D'altra parte, il comportamento a dente di sega dell'algoritmo di controllo della congestione in TCP (ricordate la Figura 6.9) non è adatto alle applicazioni in tempo reale: significa che la velocità a cui le applicazioni possono trasmettere aumenta e diminuisce in continuazione. Al contrario, le applicazioni in tempo reale funzionano bene quando sono in grado di sostenere una velocità di trasmissione che varia più gradualmente, in un periodo di tempo relativamente lungo.

6.6 Riepilogo

Possiamo ottenere il meglio da entrambe le situazioni, la compatibilità con il controllo di congestione di TCP in quanto ad equità, pur mantenendo per le applicazioni una velocità di trasmissione che varia più gradualmente? Recenti ricerche suggeriscono che la risposta sia affermativa. Nello specifico, sono stati proposti diversi algoritmi di controllo della congestione "TCP-friendly", che possono cooperare con TCP, avendo due obiettivi. Uno è quello di modificare la finestra di congestione lentamente, valutando le situazioni su periodi più lunghi (ad esempio, RTT) piuttosto che sulla base del singolo pacchetto, ottenendo così una velocità di trasmissione che varia più gradualmente. Il secondo obiettivo è quello di essere "TCP-friendly", nel senso di competere in modo equo con i flussi TCP. Questa proprietà viene spesso forzata, garantendo che il comportamento del flusso si adegui in base ad un'equazione che rappresenta un modello del comportamento del protocollo TCP. Di conseguenza, questo approccio viene a volte chiamato *controllo di congestione basato su equazioni*.

Per i nostri scopi la formulazione precisa dell'equazione è meno interessante della sua forma generale:

$$\text{Rate} = \left(\frac{1}{\text{RTT} \times \sqrt{\rho}} \right)$$

secondo la quale, per essere "TCP-friendly", la velocità di trasmissione (Rate) deve essere inversamente proporzionale al tempo di round trip (RTT) e alla radice quadrata del tasso di perdita di pacchetti (ρ). Chi fosse interessato, consulti la sezione "Ulteriori letture" al termine del capitolo per conoscere i dettagli del modello completo. In altre parole, per costruire un meccanismo di controllo della congestione basato su questa relazione, il ricevitore deve informare la sorgente in merito al tasso di perdita di pacchetti che sta osservando (ad esempio, potrebbe dire di non aver ricevuto il 10% degli ultimi 100 pacchetti) e la sorgente deve poi modificare la propria velocità di trasmissione, aumentandola o diminuendola, in modo che la relazione continui ad essere valida. Ovviamente rimane compito dell'applicazione adattarsi a queste modifiche della velocità disponibile, ma, come vedremo nel capitolo successivo, molte applicazioni in tempo reale si adattano piuttosto bene.

6.6 Riepilogo

Come abbiamo appena visto, il problema dell'allocazione delle risorse non solo è centrale nelle reti di calcolatori, ma è anche molto complesso. Questo capitolo ha esaminato due aspetti dell'allocazione di risorse. Il primo, il controllo di congestione, si occupa di prevenire il generale degrado del servizio quando la richiesta di risorse da parte degli host supera la quantità di risorse disponibili nella rete. Il secondo aspetto si occupa di fornire diverse qualità di servizio alle applicazioni che necessitano di maggiori garanzie di quelle fornite dal modello best-effort.

La maggior parte dei meccanismi di controllo della congestione hanno come obiettivo il modello di servizio best-effort della rete Internet dei giorni nostri, dove la responsabilità principale del controllo di congestione ricade sui nodi terminali della rete. Tipicamente, la sorgente usa il *feedback* (cioè l'informazione ricevuta, o implicitamente appresa dalla rete o esplicitamente inviata da un router) per modificare il carico inviato alla rete: si tratta, precisamente, del meccanismo di controllo della congestione usato dal protocollo TCP.

Indipendentemente da ciò che stanno facendo esattamente i nodi terminali, i router implementano una disciplina di gestione delle code che decide quali pacchetti vengono trasmessi e quali vengono eliminati. A volte questi algoritmi di gestione delle code (come, ad esempio, WFQ) sono abbastanza sofisticati da separare rigidamente le varie componenti di traffico, mentre in altri casi (ad esempio, RED e DECBit) il router tenta di tenere sotto controllo la lunghezza della propria coda, per poi inviare messaggi all'host sorgente nel momento in cui sta per insorgere una congestione.

Gli approcci emergenti per la qualità di servizio aspirano a fare sostanzialmente di più del semplice controllo di congestione: il loro obiettivo è quello di consentire ad applicazioni con requisiti assai variabili di ritardo, perdita di pacchetti e throughput, di vedere soddisfatti questi requisiti mediante nuovi meccanismi interni alla rete. L'approccio Integrated Services consente ai flussi delle singole applicazioni di specificare le proprie necessità ai router usando un meccanismo di segnalazione esplicito (RSVP), mentre l'approccio Differentiated Services classifica i pacchetti all'interno di un ristretto numero di classi che ricevono, nei router, un trattamento differenziato. Nonostante la segnalazione usata dalla tecnologia ATM sia molto diversa dal protocollo RSVP, esistono notevoli somiglianze tra le classi di servizio di ATM e quelle di Integrated Services.

Problema aperto All'interno e all'esterno della rete

La domanda più importante che ci dovremmo porre è: quanto ci possiamo aspettare dalla rete e quanta parte di responsabilità dovremo comunque far ricadere sugli host terminali? Le strategie basate sulla prenotazione hanno certamente il vantaggio di fornire qualità di servizio più varie degli schemi basati sul feedback in uso oggi giorno e questa è una forte motivazione per inserire maggiori funzionalità nei router. Questo significa che il controllo di congestione tra nodi terminali, come quello di TCP, ha i giorni contati? Sembra molto improbabile. Il protocollo TCP e le applicazioni che lo usano sono ben collaudati e in molti casi non hanno bisogno di molto di più dalla rete di ciò che hanno già. Inoltre, è assai poco probabile che tutti i router di una rete mondiale ed eterogenea come Internet possano mai implementare il medesimo algoritmo di allocazione delle risorse. Sembra ragionevole che, alla fine, gli host che si trovano ai due estremi di una connessione debbano, almeno per qualche particolare, fare da soli. Dopotutto, non dovremmo mai dimenticare il principio ispiratore del progetto di Internet: fare in modo che i router svolgano il compito più semplice possibile e che le cose difficili vengano fatte nei nodi terminali, sui quali si ha maggior controllo. Sarà veramente interessante vedere come tutto questo si evolverà nei prossimi anni.

In un certo senso, l'approccio Differentiated Services rappresenta una soluzione intermedia fra una rete con un bagaglio di informazioni davvero minimo e la rete molto più dotata (e con molte più informazioni memorizzate al proprio interno) che è richiesta da Integrated Services. Di certo la maggior parte dei fornitori di servizi Internet sono riluttanti a consentire ai propri clienti di effettuare prenotazioni RSVP all'interno della rete dei fornitori. Una domanda importante è se l'approccio Differentiated Services sarà in grado di soddisfare le richieste delle applicazioni più esigenti. Ad esempio, se un fornitore di servizio sta cercando di offrire un servizio di telefonia su larga scala mediante una rete IP, le tecniche di Differentiated Services saranno adatte a fornire la qualità di servizio attesa dagli utenti del servizio telefonico tradizionale? Sembra probabile che si debbano esplorare ancora altre opzioni di qualità di servizio, con diverse quantità di informazioni all'interno della rete.

Ulteriori letture

L'elenco di letture consigliate per questo capitolo è lungo, in conseguenza della grande mole di lavoro interessante che è stato fatto sul controllo di congestione e sull'allocatione di risorse. L'elenco comprende le pubblicazioni originali che hanno presentato i diversi meccanismi discussi nel capitolo: oltre ad una descrizione più dettagliata di questi meccanismi, comprendendo anche una analisi approfondita della loro efficacia ed equità, queste pubblicazioni sono una lettura necessaria anche per avere una visione più completa delle relazioni esistenti fra i vari problemi legati al controllo di congestione. Inoltre, il primo lavoro dell'elenco presenta una valida panoramica di alcuni dei primi studi di questo argomento, mentre l'ultimo lavoro è considerato tra quelli fondamentali nello sviluppo della capacità di Internet in relazione alla qualità di servizio.

- Gerla, M., e L. Kleinrock "Flow control: A comparative survey", *IEEE Transactions on Communications* COM-28(4):533-573, April 1980.
- Demers, A., S. Keshav e S. Shenker "Analysis and simulation of a fair queueing algorithm", *Proceedings of the SIGCOMM '89 Symposium*, pagg. 1-12, September 1989.
- Jacobson, V. "Congestion avoidance and control", *Proceedings of the SIGCOMM '88 Symposium*, pagg. 314-329, August 1988.
- Ramakrishnan, K., e R. Jain "A binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer", *ACM Transactions on Computer Systems* 8(2):158-181, May 1990.
- Floyd, S., e V. Jacobson "Random early detection gateways for congestion avoidance", *IEEE/ACM Transactions on Networking* 1(4):397-413, August 1993.
- Clark, D., S. Shenker e L. Zhang "Supporting real-time applications in an integrated services packet network: Architecture and mechanism", *Proceedings of the SIGCOMM '92 Symposium*, pagg. 14-26, August 1992.

Oltre a questi lavori, che raccomandiamo, esiste molto altro materiale sull'allocatione di risorse. Per chi inizia ad esplorare l'argomento, due pubblicazioni di Kleinrock [Kle79] e Jaffe [Jaf81] gettano le basi per l'uso della potenza come misura dell'efficacia del controllo di congestione. Ancora, Jain [Jai91] discute approfonditamente i vari problemi correlati alla valutazione delle prestazioni, con una descrizione dell'indice di equità di Jain.

In Brakmo e Peterson [BP95] si possono trovare maggiori dettagli su TCP Vegas, con il seguito del lavoro presentato in Low *et al.* [LPW02]. Le tecniche simili per evitare la congestione che sono state presentate nella Sezione 6.4 si trovano in Wang e Crowcroft [WC92, WC91] e in Jain [Jai89]: il primo dei lavori citati presenta una splendida panoramica dei metodi usati per evitare la congestione e basati sulla comprensione di come si modifichi la rete quando si approssima una congestione. Tra le modifiche che sono state proposte per l'algoritmo RED troviamo "Fair RED" (FRED), descritta da Lin e Morris [LM97] e una variante di RED auto-configurente presentata da Feng *et al.* [FKSS99]. La tecnica a coppie di pacchetti brevemente delineata nella Sezione 6.3.2 è descritta con maggior cura in Keshav [Kes91], mentre la tecnica dell'eliminazione parziale di un pacchetto, che abbiamo accennato nella Sezione 6.4.2, è stata descritta da Romanow e Floyd [RF94].

Lo standard proposto per ECN è stato scritto da Ramakrishnan, Floyd e Black in [RFB01]. Gli sforzi per rendere più generale questa idea, nella forma di una gestione attiva delle code, sono stati portati avanti da Stoica *et al.* [SSZ98], Low *et al.* [LPW⁺02] e Katahi *et al.* [KHR02].

I lavori più recenti sulla pianificazione dei pacchetti hanno esteso il lavoro originario sulla gestione equa delle code citato in precedenza. Tra gli esempi migliori, citiamo gli articoli di Stoica e Zhang [SZ97], Bennett e Zhang [BZ96] e Goyal, Vin e Chen [GVC96].

Sono stati anche pubblicati molti altri articoli sull'architettura Integrated Services, tra i quali una panoramica di Braden *et al.* [BCS94] e una descrizione di RSVP di Zhang *et al.* [ZDE+93]. La prima pubblicazione ad affrontare l'argomento Differentiated Services è di Clark [Cla97], che ha presentato il meccanismo RIO, insieme all'architettura generale di Differentiated Services. Il seguito, un lavoro di Clark e Fang [CF98], presenta alcuni risultati di simulazione. [BBC+98] definisce l'architettura Differentiated Services, mentre [DCB-02] definisce il comportamento per-hop EF.

Infine, sono stati recentemente proposti diversi algoritmi di controllo della congestione di tipo "TCP-friendly", progettati per un utilizzo con applicazioni in tempo reale. Tra questi, gli algoritmi di Floyd *et al.* [FHPW00], Sisalem e Schulzrinne [SS98], Rhee *et al.* [ROY00] e Rejase *et al.* [RHE99], tutti basati sul precedente modello basato su equazioni del throughput di TCP di Padhye *et al.* [PFTK98].

Esercizi

1. I flussi possono essere definiti "tra host" o "tra processi".
 - a) Discutete quali implicazioni hanno questi due approcci per i programmi applicativi.
 - b) IPv6 contiene il campo FlowLabel per dare indicazioni ai router in merito ai singoli flussi. L'host sorgente deve inserire in tale campo una funzione di hash pseudocasuale di tutti gli altri campi che servono ad identificare il flusso; il router può, quindi, usare un sottoinsieme qualunque di questi bit come valore di hash per una ricerca rapida del flusso nelle proprie tabelle. Su che cosa dovrebbe basarsi il calcolo di FlowLabel, per ognuno di questi due approcci?
2. Il protocollo TCP usa un modello di allocazione delle risorse basato su finestre, con feedback e con responsabilità agli host. Come avrebbe potuto essere progettato TCP per usare, invece, i seguenti modelli?
 - a) Con responsabilità agli host, con feedback e basato sulla velocità.
 - b) Con responsabilità ai router e con feedback.
- ★ 3. Per ognuna delle reti seguenti, tracciate schematicamente dei grafici, in funzione del carico, per il throughput, il ritardo e la potenza. Il throughput va misurato come percentuale del suo valore massimo. Il carico va misurato (in modo piuttosto innaturale) come numero di stazioni (N) pronte a trasmettere in un certo istante; notate che ciò implica che ci sia sempre una stazione pronta a trasmettere (tranne nel caso in cui sia $N = 0$, caso che potete ignorare). Ipotizzate che ogni stazione abbia ogni volta un solo pacchetto da trasmettere.
 - a) Rete Ethernet. Ipotizzate, come nell'Esercizio 52 del Capitolo 2, che la dimensione media dei pacchetti sia pari a 5 volte l'intervallo di slot e che, quando N stazioni stanno cercando di trasmettere, il tempo medio necessario perché una stazione abbia successo è pari a $N/2$ intervalli di slot.
 - b) Rete token ring, con TRT = 0.
4. Supponete che due host, A e B, siano connessi mediante un router, R. La linea di collegamento A-R ha ampiezza di banda infinita, mentre la linea R-B può inviare un pacchetto al secondo. La coda di R è infinita. Il carico si misura come numero di pacchetti

al secondo inviati da A a B. Tracciate i grafici del throughput e del ritardo in funzione del carico, oppure, se non riuscite a tracciare uno dei grafici, fornite una motivazione. Sarebbe più adatta una diversa modalità di misurazione del carico?

5. È possibile che il protocollo TCP Reno raggiunga uno stato in cui la dimensione della finestra di congestione è molto maggiore (cioè almeno il doppio) del prodotto tra RTT e l'ampiezza di banda? È una cosa probabile?
6. Considerate la disposizione di host, H, e di router, R e R1, della Figura 6.28. Tutte le linee di collegamento sono full-duplex e tutti i router sono più veloci delle rispettive linee. Dimostrate che R1 non può andare incontro a congestione, mentre per ogni altro router R è possibile identificare uno schema di traffico che rende congestionato quell'unico router.
7. Supponete che uno schema di controllo della congestione dia luogo ad un insieme di flussi in competizione che raggiungono i throughput seguenti: 100 KBps, 60 KBps, 110 KBps, 95 KBps e 150 KBps.
 - a) Calcolate l'indice di equità per questo schema.
 - b) Aggiungete ora alla situazione precedente un flusso con throughput di 1000 KBps e ricalcolate l'indice di equità.
8. Nella gestione equa delle code, il valore F_i veniva interpretato come annotazione oraria: l'istante in cui termina la trasmissione del pacchetto i -esimo. Fornite un'interpretazione di F_i per la gestione delle code equa e pesata e identificate una formula per esprimere F_i in funzione di F_{i-1} , del tempo di arrivo A_i , della dimensione del pacchetto P_i e del peso w assegnato al flusso.
9. Descrivete un esempio di come la mancata prevaricazione nell'implementazione delle code con gestione equa porti ad un ordine di trasmissione dei pacchetti diverso da quanto avviene nel servizio di tipo round-robin bit per bit.
10. Supponete che un router abbia un'uscita e tre flussi di ingresso. Riceve i pacchetti elencati nella Tabella 6.2 pressappoco nello stesso istante, nell'ordine elencato, durante un periodo in cui la porta d'uscita è impegnata ma tutte le code sono vuote. Indicate in quale ordine vengono trasmessi i pacchetti, ipotizzando che:
 - a) si usi una gestione equa delle code;
 - b) si usi una gestione equa e pesata delle code, con il flusso 2 di peso 2 e gli altri flussi con peso 1.

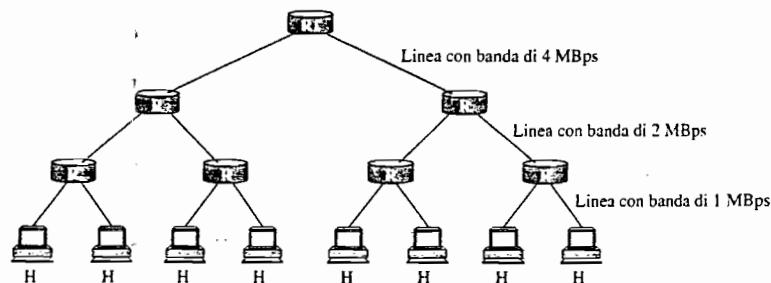


Figura 6.28 Schema per l'Esercizio 6.

Tabella 6.2 Pacchetti per l'Esercizio 10.

Pacchetto	Dimensione	Flusso
1	100	1
2	100	1
3	100	1
4	100	1
5	190	2
6	200	2
7	110	3
8	50	3

Tabella 6.3 Pacchetti per l'Esercizio 11.

Pacchetto	Dimensione	Flusso
1	200	1
2	200	1
3	160	2
4	120	2
5	160	2
6	210	3
7	150	3
8	90	3

- ✓ 11. Supponete che un router abbia un'uscita e tre flussi di ingresso. Riceve i pacchetti elencati nella Tabella 6.3 pressappoco nello stesso istante, nell'ordine elencato, durante un periodo in cui la porta d'uscita è impegnata ma tutte le code sono vuote. Indicate in quale ordine vengono trasmessi i pacchetti, ipotizzando che:
- si usi una gestione equa delle code;
 - si usi una gestione equa e pesata delle code, con il flusso 2 che abbia una frazione di banda uguale al doppio di quella del flusso 1, ed il flusso 3 con una frazione di banda uguale a 1.5 volte la banda del flusso 1. Notate che le situazioni di parità devono risolversi nell'ordine di numerazione di flusso, prima il flusso 1, poi il flusso 2, infine il flusso 3.
12. Supponete che la strategia di eliminazione di un router consista nell'eliminare il pacchetto di costo più elevato ogni volta che le code sono piene, dove definiamo il costo di un pacchetto come il prodotto della sua dimensione per il tempo *rimanente* che trascorrerà in coda (notate che per calcolare il costo, invece del tempo rimanente, è equivalente utilizzare la somma delle dimensioni dei pacchetti precedenti).
- Quali vantaggi e svantaggi potrebbe offrire questa strategia, confrontata con l'eliminazione terminale (*tail drop*)?
 - Fornite un esempio di una sequenza di pacchetti in coda per la quale l'eliminazione del pacchetto di maggior costo differisce dall'eliminazione del pacchetto di dimensioni maggiori.

- c) Fornite un esempio in cui due pacchetti, con il trascorrere del tempo, si scambiano la posizione relativa nella graduatoria di costo.

13. Due utenti, uno che usa Telnet e uno che invia file con FTP, mandano il proprio traffico attraverso il router R. La linea uscente da R è abbastanza lenta perché entrambi gli utenti abbiano propri pacchetti nella coda di R in ogni istante. Discutete le prestazioni relative viste dall'utente di Telnet se la politica di gestione della coda di R per questi due flussi è
- un servizio di tipo round-robin
 - una gestione equa
 - una gestione equa modificata, dove contiamo solamente il costo dei byte di dati e non le intestazioni IP o TCP

Considerate soltanto il traffico uscente. Ipotizzate che i pacchetti Telnet abbiano 1 byte di dati, che i pacchetti FTP abbiano 512 byte di dati e che tutti i pacchetti abbiano intestazioni di 40 byte.

14. Considerate un router che stia gestendo tre flussi, sui quali arrivano pacchetti di dimensione costante ai seguenti istanti

flusso A: 1, 2, 4, 6, 7, 9, 10
 flusso B: 2, 6, 8, 11, 12, 15
 flusso C: 1, 2, 3, 5, 6, 7, 8

I tre flussi condividono la medesima linea di uscita, sulla quale il router può trasmettere un pacchetto per unità di tempo. Ipotizzate che sia disponibile uno spazio infinito nel buffer.

- Supponete che il router usi una gestione equa della coda (FQ). Per ogni pacchetto, indicate l'istante in cui viene trasmesso dal router. Le parità negli istanti di arrivo vanno risolte dando la precedenza ad A, poi a B e infine a C. Notate che all'istante 2 il tempo dell'orologio di FQ è $A_i = 1.5$.
- Supponete che il router usi una gestione equa e pesata della coda, con i flussi A e B che hanno la stessa frazione di capacità della linea, mentre il flusso C ha il doppio della capacità del flusso A. Per ogni pacchetto, indicate l'istante in cui viene trasmesso dal router.

- ✓ 15. Considerate un router che stia gestendo tre flussi, sui quali arrivano pacchetti di dimensione costante ai seguenti istanti

flusso A: 1, 3, 5, 6, 7, 8, 11
 flusso B: 1, 4, 7, 8, 9, 13, 15
 flusso C: 1, 2, 4, 6, 7, 12

I tre flussi condividono la medesima linea di uscita, sulla quale il router può trasmettere un pacchetto per unità di tempo. Ipotizzate che sia disponibile uno spazio infinito nel buffer.

- Supponete che il router usi una gestione equa della coda (FQ). Per ogni pacchetto, indicate l'istante in cui viene trasmesso dal router. Le parità negli istanti di arrivo vanno risolte dando la precedenza ad A, poi a B e infine a C. Notate che all'istante 2 il tempo dell'orologio di FQ è $A_i = 1.333$.
- Supponete che il router usi una gestione equa e pesata della coda, con i flussi A e C che hanno la stessa frazione di capacità della linea, mentre il flusso B ha il doppio

- della capacità del flusso A. Per ogni pacchetto, indicate l'istante in cui viene trasmesso dal router.
16. Nell'ipotesi che TCP implementi un'estensione che consente alla dimensione della finestra di essere maggiore di 64 KB, supponete di usare tale estensione in una linea di collegamento a 1 Gbps con latenza di 100 ms per trasferire un file di 10 MB e che la finestra di ricezione di TCP sia 1 MB. Se il protocollo TCP invia pacchetti di 1 KB, nell'ipotesi che non vi sia congestione e che non si perdano pacchetti:
 - a) Quanti intervalli di durata uguale a RTT trascorrono prima che la partenza lenta porti la finestra alla dimensione di 1 MB?
 - b) Quanti intervalli di durata uguale a RTT servono per trasferire il file?
 - c) Se il tempo necessario per trasferire il file è dato dal numero di RTT necessari moltiplicato per la latenza della linea, qual è il throughput effettivo per il trasferimento? Quale percentuale dell'ampiezza di banda della linea viene utilizzata?
 17. Considerate un semplice algoritmo di controllo della congestione che usa l'aumento lineare e la diminuzione moltiplicativa ma non usa la partenza lenta, la cui unità di misura sia il pacchetto piuttosto che il byte e che inizi ogni connessione con una finestra di congestione uguale ad un pacchetto. Date una descrizione dettagliata dell'algoritmo. Ipotizzate che il ritardo sia dovuto soltanto alla latenza e che quando viene spedito un gruppo di pacchetti venga restituita un'unica conferma ACK. Tracciate un grafico della finestra di congestione in funzione dei tempi di round-trip nella situazione in cui vadano perduti i seguenti pacchetti: 9, 25, 30, 38 e 50. Per semplicità, ipotizzate che il meccanismo delle temporizzazioni sia perfetto e che si accorga della perdita di un pacchetto dopo un tempo uguale a RTT dal momento in cui è stato trasmesso.
 18. Nella situazione illustrata nel problema precedente, calcolate il throughput effettivo ottenuto dalla connessione. Ipotizzate che ogni pacchetto contenga 1 KB di dati e che RTT valga 100 ms.
 19. Durante l'aumento lineare, il protocollo TCP calcola un incremento della finestra di congestione come

$$\text{Increment} = \text{MSS} \times (\text{MSS}/\text{CongestionWindow})$$

Spiegate perché il calcolo di questo incremento ogni volta che arriva un ACK può fornire un risultato non corretto. Fornite una definizione più precisa di questo incremento. (Suggerimento: un messaggio ACK può confermare una quantità di dati maggiore o minore di MSS)

20. In quali circostanze in TCP possono scadere comunque le temporizzazioni anche se si usa il meccanismo di ritrasmissione veloce?
21. Supponete di voler aggiungere il controllo di congestione affidato agli host nei protocolli BLAST e CHAN di RPC. Che forma assumerebbe questo controllo? Sarebbe meglio aggiungerlo a BLAST, a CHAN o ad entrambi?
22. Supponete che fra A e B ci sia un router, R. L'ampiezza di banda della linea A-R è infinita (cioè i pacchetti non subiscono ritardo), ma la linea R-B introduce un ritardo di banda di un secondo per pacchetto (cioè due pacchetti impiegano due secondi, e così via). Le conferme da B a R vengono però inviate istantaneamente. A invia dati a B lungo una connessione TCP, usando la partenza lenta ma con una finestra di dimensione arbitrariamente grande. R ha una coda di dimensione 1, oltre al pacchetto che sta

trasmettendo. Ad ogni secondo, la sorgente per prima cosa elabora tutte le conferme in arrivo, poi risponde alle temporizzazioni che scadono.

- a) Usando un periodo fisso di 2 secondi come valore di TimeOut, cosa viene inviato e ricevuto agli istanti $T = 0, 1, \dots, 6$ secondi? La linea rimane mai inattiva a causa di temporizzazioni che scadono?
- b) Cosa cambia se si usa un valore di TimeOut di 3 secondi?
23. Supponete che A, B e R siano quelli dell'esercizio precedente, tranne per il fatto che la coda di R può ora ospitare tre pacchetti, oltre a quello che sta trasmettendo. A inizia una connessione usando la partenza lenta, con una finestra di ricezione infinita. Al secondo ACK duplicato (cioè al terzo ACK per il medesimo pacchetto) si innesca la ritrasmissione veloce; il valore di TimeOut è infinito. Trascurate il recupero veloce: quando viene perduto un pacchetto, la finestra assume dimensione unitaria. Fornite una tabella che mostri, per i primi 15 secondi, cosa riceve A, cosa invia A, cosa invia R, cosa elimina R e qual è la coda di R.
24. Supponete che la linea R-B dell'esercizio precedente passi da un ritardo di banda ad un ritardo di propagazione, in modo che ora due pacchetti richiedano 1 secondo per essere trasmessi. Elencate cosa viene inviato e ricevuto durante i primi 8 secondi. Ipotizzate un valore costante della temporizzazione uguale a 2 secondi, che allo scadere di una temporizzazione si usa la partenza lenta e che si fondano le conferme ACK spedite pressappoco nello stesso istante. Notate che la dimensione della coda di R è ora ininfluente (perché?).
25. Supponete che l'host A raggiunga l'host B attraverso i router R1 e R2: A-R1-R2-B. Non si usa la ritrasmissione veloce e A calcola il proprio valore di TimeOut come $2 \times \text{EstimatedRTT}$. Ipotizzate che le linee A-R1 e R2-B abbiano ampiezza di banda infinita; la linea R1-R2, però, introduce un ritardo di banda di 1 secondo per pacchetto per i pacchetti di dati (ma non per i messaggi ACK). Descrivete uno scenario in cui la linea R1-R2 non viene utilizzata al 100%, anche se A ha sempre dati pronti per essere spediti. Suggerimento: supponete che il valore di CongestionWindow di A aumenti da N a $N + 1$, essendo N la dimensione della coda di R1.
26. Siete un fornitore di servizi Internet e gli host dei vostri clienti si connettono direttamente ai vostri router. Sapete che alcuni host stanno sperimentando una nuova versione di TCP e avete il sospetto che stiano usando un protocollo TCP "aggressivo", senza alcun controllo di congestione. Quali misurazioni potreste effettuare nei vostri router per stabilire che un cliente non sta affatto usando la partenza lenta? Se un cliente usasse la partenza lenta all'inizio della connessione, ma non dopo la scadenza di una temporizzazione, potreste accorgervene?
27. La violazione dei meccanismi di controllo della congestione di TCP solitamente richiede l'esplicita cooperazione della sorgente. Tuttavia, considerate la parte ricevente di un ingente trasferimento di dati che usa un protocollo TCP modificato in modo da confermare pacchetti non ancora arrivati, perché non tutti i dati sono necessari oppure perché i dati perduti possono essere recuperati successivamente con un trasferimento distinto. Quali effetti ha questo comportamento del ricevitore sulle proprietà del controllo di congestione della sessione? Potete identificare un modo per modificare il protocollo TCP in modo che venga eliminata la possibilità, per una sorgente, di trarre vantaggio da tale comportamento?
28. Considerate il grafico di TCP mostrato in Figura 6.29. Identificate gli intervalli di tempo che rappresentano la partenza lenta all'inizio della connessione e la partenza lenta

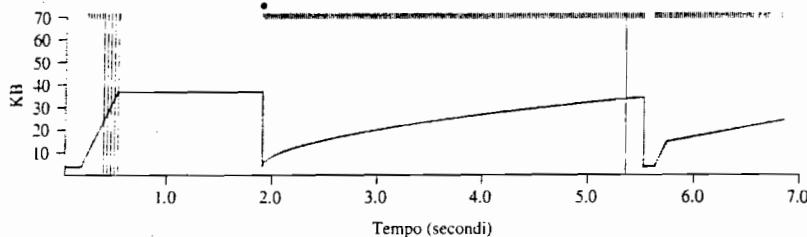


Figura 6.29 Grafico TCP per l'Esercizio 28.

- dopo la scadenza di una temporizzazione, e l'intervallo in cui si impedisce la congestione con un aumento lineare. Spiegate cosa succede fra gli istanti 0.5 e 1.9. La versione di TCP che ha prodotto questo grafico presenta una caratteristica che è assente nella versione di TCP che ha prodotto la Figura 6.11. Qual è questa caratteristica? Sia questo grafico che quello di Figura 6.13 non presentano una particolare caratteristica, quale?
29. Supponete di scaricare un file di grandi dimensioni lungo una linea telefonica a 3 KBps. Il vostro software visualizza un conteggio medio di byte al secondo. Come varia questo conteggio per effetto del controllo di congestione di TCP e delle perdite occasionali di pacchetti? Ipotizzate che soltanto un terzo (ad esempio) del tempo RTT totale sia dovuto alla linea telefonica.
 30. Supponete che TCP venga usato su una linea soggetta a molte perdite, che in media perde un segmento ogni quattro. Ipotizzate che la dimensione della finestra banda × ritardo sia considerevolmente maggiore di quattro segmenti.
 - a) Cosa accade quando inizia una connessione? Si arriva mai alla fase di aumento lineare prevista dalla strategia che evita la congestione?
 - b) Senza l'uso di un meccanismo di feedback esplicito da parte dei router, avrebbe modo TCP di distinguere queste perdite dovute alla linea da perdite dovute a congestione, almeno nel breve periodo?
 - c) Supponete che le sorgenti TCP abbiano ricevuto in modo affidabile dai router indicazioni esplicite di congestione. Nell'ipotesi che linee come quella citata siano comuni, sarebbe utile usare finestre di dimensioni molto maggiori di quattro segmenti? Cosa dovrebbe fare il protocollo TCP?
 31. Supponete che due connessioni TCP condividano un percorso attraverso un router R. La dimensione della coda del router è di sei segmenti e ciascuna connessione ha una finestra di congestione stabile pari a tre segmenti. Queste connessioni non usano alcun controllo di congestione. Si tenta ora di instaurare una terza connessione TCP, ancora attraverso R, anche questa senza alcun controllo di congestione. Descrivete uno scenario in cui, almeno per un certo tempo, la terza connessione non ottiene alcuna frazione della banda disponibile, mentre le prime due connessioni continuano a trasmettere usando il 50% ciascuna. Cambia qualcosa se la terza connessione usa la partenza lenta? Come fa ad essere di aiuto nella soluzione di questo problema il fatto che le prime due connessioni usino un meccanismo che impedisce del tutto la congestione?
 32. Supponete che una connessione TCP abbia una finestra di dimensione pari a otto segmenti, un RTT di 800 ms, che la sorgente spedisca segmenti alla velocità costante di uno ogni 100 ms e che il ricevitore restituisca le conferme ACK alla stessa velocità.

senza ritardi. Viene perso un segmento e la perdita è rilevata dall'algoritmo di ritrasmissione veloce nel momento in cui viene ricevuto il terzo ACK duplicato. Nel momento in cui viene finalmente ricevuto il messaggio di conferma del segmento ritrasmesso, quanto tempo in totale è stato perduto dalla sorgente (in confronto ad una trasmissione priva di perdite di segmenti) se:

- a) la sorgente attende l'ACK del pacchetto perduto e ritrasmesso prima di far avanzare di nuovo la finestra scorrevole?
- b) la sorgente usa l'arrivo continuo di ogni ACK duplicato come indicazione di poter far avanzare la finestra scorrevole di un segmento?
33. Il libro afferma che l'aumento additivo è una condizione necessaria perché un meccanismo di controllo della congestione sia stabile. Delineate una specifica instabilità che potrebbe insorgere qualora tutti gli aumenti fossero esponenziali, cioè se TCP continuasse ad utilizzare la partenza "lenta" dopo che il valore di CongestionWindow ha superato CongestionThreshold.
34. Discutete i vantaggi e gli svantaggi della marcatura di un pacchetto (come nel meccanismo DEChit) rispetto all'eliminazione di un pacchetto (come in RED).
35. Considerate un router RED con $\text{MaxP} = 0.02$ e con una lunghezza media della coda in posizione mediana rispetto alle due soglie.
 - a) Determinate la probabilità di eliminazione P_{count} , per $\text{count} = 1$ e $\text{count} = 50$.
 - b) Calcolate la probabilità che nessuno dei primi 50 pacchetti venga eliminato. Notate che questa probabilità è $(1 - P_1) \times \dots \times (1 - P_{50})$.
- ✓ 36. Considerate un router RED con $\text{MaxP} = p$ e con una lunghezza media della coda in posizione mediana rispetto alle due soglie.
 - a) Determinate la probabilità che nessuno dei primi n pacchetti venga eliminato.
 - b) Determinate p in modo che la probabilità che nessuno dei primi n pacchetti venga eliminato sia α .
37. Spiegate l'idea che sta alla base dell'impostazione di MaxThreshold al valore $2 \times \text{MinThreshold}$ in RED.
38. Spiegate perché, nel meccanismo RED, MaxThreshold è in realtà minore della dimensione corrente del buffer disponibile.
39. Spiegate il conflitto fondamentale esistente fra tollerare le raffiche e controllare la congestione di rete.
40. Secondo voi, perché la probabilità di eliminazione P di RED non aumenta, semplicemente, in modo lineare da $P = 0$ in corrispondenza di MinThreshold, fino a $P = 1$ in corrispondenza di MaxThreshold?
- ★ 41. In TCP Vegas, il calcolo di ActualRate viene eseguito dividendo la quantità di dati trasmessa in un intervallo di durata RTT per il valore di RTT stesso.
 - a) Dimostrate che, per qualsiasi implementazione di TCP, se la dimensione della finestra rimane costante, allora la quantità di dati trasmessi in un intervallo di durata RTT è costante, dopo che è stata inviata una finestra intera. Ipotizzate che la sorgente trasmetta ciascun segmento nell'istante in cui riceve un ACK, che i pacchetti non vengano perduti e che vengano consegnati in ordine, che tutti i segmenti abbiano la stessa dimensione e che la prima linea lungo il percorso non sia la più lenta.
 - b) Tracciate un diagramma temporale che mostri che la quantità di dati per un RTT può essere inferiore al valore di CongestionWindow.
42. Supponete che una connessione TCP Vegas misuri il valore di RTT del suo primo pacchetto e imposti BaseRTT a quel valore, ma che poi avvenga un guasto su una linea della

- rete e tutto il traffico successivo venga instradato verso un percorso alternativo con un valore di RTT doppio. Come reagirà TCP Vegas? Cosa accadrà al valore di CongestionWindow? Ipotizzate che non scada nessuna temporizzazione e che β sia molto minore del valore iniziale di ExpectedRate.
43. Considerate le due seguenti cause di ritardo di rete di 1 secondo (nell'ipotesi che i messaggi ACK vengano restituiti istantaneamente):
- un router intermedio con un ritardo di banda in uscita di 1 secondo per pacchetto, senza traffico concomitante
 - un router intermedio con un ritardo di banda in uscita di 100 ms per pacchetto e con 10 pacchetti in coda provenienti da un'altra sorgente che li ripristina continuamente
- a) Come potrebbe un generico protocollo di trasporto distinguere tra le due situazioni?
- b) Supponete che le connessioni precedenti siano gestite da TCP Vegas, con un valore iniziale di CongestionWindow uguale a 3 pacchetti. Cosa accadrebbe, in ognuno dei due casi, al valore di CongestionWindow? Assumete i valori di $BaseRTT = 1$ secondo e $\beta = 1$ pacchetto al secondo.
44. Motivate il fatto che il problema del controllo di congestione viene gestito meglio al livello internet piuttosto che al livello ATM, almeno quando solo una parte della internetwork è in tecnologia ATM. In una rete puramente IP-over-ATM, la congestione viene gestita meglio a livello di celle o a livello TCP? Perché?
45. Fornite una traccia di come un commutatore ATM potrebbe implementare l'eliminazione parziale di pacchetti e l'eliminazione anticipata di pacchetti. Uno dei due meccanismi è sensibilmente più semplice dell'altro?
46. Considerate la tassonomia di Figura 6.23.
- Fornite un esempio di applicazione in tempo reale che sia intollerante e adattativa in velocità.
 - Spiegate perché vi aspettereste che un'applicazione tollerante alle perdite di pacchetti fosse almeno in qualche modo adattativa in velocità.
 - Ignorando la parte (b), fornite un esempio di applicazione che potrebbe essere considerata tollerante e non adattativa. Suggerimento: un'applicazione viene definita tollerante alle perdite anche se tollera solamente piccole perdite; dovete considerare l'adattabilità in velocità come la capacità di modificare il proprio comportamento in seguito a modifiche rilevanti dell'ampiezza di banda.
47. La pianificazione della trasmissione (Tabella 6.4) per un determinato flusso elenca, per ogni secondo, il numero di pacchetti inviati fra quell'istante e il secondo successivo. Il flusso deve rimanere confinato all'interno di un filtro token bucket. Di quale profondità

Tabella 6.4 Pianificazione di trasmissione per l'Esercizio 47.

Tempo (secondi)	Pacchetti inviati
0	5
1	5
2	1
3	0
4	6
5	1

di bucket ha bisogno il flusso per le seguenti velocità di token, nell'ipotesi che il bucket sia inizialmente vuoto?

- a) 2 pacchetti al secondo
b) 4 pacchetti al secondo

- ✓ 48. La pianificazione della trasmissione (Tabella 6.5) per un determinato flusso elenca, per ogni secondo, il numero di pacchetti inviati fra quell'istante e il secondo successivo. Il flusso deve rimanere confinato all'interno di un filtro token bucket. Determinate la profondità D del bucket necessaria, in funzione della velocità di token, r . Notate che r assume soltanto valori interi positivi. Assumete che il bucket sia inizialmente vuoto.
49. Supponete che un router abbia accettato flussi con le TSpec mostrate in Tabella 6.6, descritte in termini di filtri token bucket con velocità di token r pacchetti al secondo e profondità di bucket di B pacchetti. I flussi viaggiano nella stessa direzione e il router può inoltrare un pacchetto ogni 0.1 secondi.
- Qual è il ritardo massimo a cui può essere sottoposto un pacchetto?
 - Qual è il numero minimo di pacchetti del terzo flusso che verranno inviati dal router in 2.0 secondi, nell'ipotesi che il flusso invii pacchetti in modo uniforme alla propria velocità massima?
50. Supponete che un router RSVP perda improvvisamente il proprio stato di prenotazione, pur rimanendo operativo.
- Cosa accadrà ai flussi con prenotazioni in essere se il router gestisce i flussi con prenotazione e senza prenotazione mediante un'unica coda FIFO?
 - Cosa potrebbe accadere ai flussi con prenotazioni in essere se il router usasse una gestione della coda equa e pesata per separare il traffico con prenotazione da quello senza prenotazione?
 - Prima o poi i ricevitori di questi flussi richiederanno il rinnovo delle proprie prenotazioni. Fornite uno scenario in cui queste prenotazioni vengano rifiutate.
51. Considerate il circuito virtuale ATM in classe ABR di Figura 6.27, segmentato in due percorsi ciclici di controllo in corrispondenza del commutatore S2.

Tabella 6.5 Pianificazione di trasmissione per l'Esercizio 48.

Tempo (secondi)	Pacchetti inviati
0	5
1	5
2	1
3	0
4	6
5	1

Tabella 6.6 TSpec per l'Esercizio 49.

r	B
1	10
2	4
4	1

- a) Supponete che la cella di gestione di risorse RM₁, riparta da S2 verso H1 segnalando un'elevata velocità disponibile, ma subito dopo arrivi in S2 una cella RM proveniente da H2 che segnala una bassa velocità disponibile per la seconda parte del circuito. Quali problemi potrebbe trovarsi a dover affrontare S2?
- b) Quando S2 riceve RM₁ da H1, potrebbe semplicemente trattenerla mentre invia la propria cella RM₂ a H2, in attesa che quest'ultima ritorni. Quando RM₂ torna, S2 potrebbe restituire RM₁, riducendo, se necessario, la velocità in essa specificata. Perché questa strategia potrebbe creare problemi? Suggerimento: ricordate lo scopo della segmentazione in percorsi ciclici di controllo più brevi.

Rappresentazione dei dati per la rete

Problema Come si trattano i dati?

Dal punto di vista della rete, i programmi applicativi si scambiano messaggi, che sono soltanto sequenze di byte senza alcun significato. Dal punto di vista delle applicazioni, però, questi messaggi contengono diversi tipi di *dati*: sequenze di numeri interi, riquadri di immagini video, linee di testo, immagini digitali, e così via. In altre parole, quei byte hanno un significato. Consideriamo ora il problema di come sia meglio codificare in sequenze di byte i diversi tipi di dati che le applicazioni si vogliono scambiare: per molti aspetti, si tratta di un problema simile alla codifica delle sequenze di byte in segnali elettromagnetici che abbiamo affrontato nella Sezione 2.2.

Ripensando alla nostra discussione sulla codifica, presentata nel Capitolo 2, i problemi erano essenzialmente due. Il primo consisteva nel fare in modo che il ricevitore potesse estrarre dal segnale il medesimo messaggio che era stato inviato dal mittente: si trattava del problema della tramatura (*framing*). Il secondo riguardava l'efficienza della codifica. Quando si codificano in messaggi di rete i dati delle applicazioni, ci si trova nuovamente di fronte a questi due problemi.

Per quanto riguarda il fatto che mittente e destinatario vedano gli stessi dati, la soluzione consiste nel fatto che le due parti si accordino su un formato del messaggio, spesso chiamato *formato di presentazione*. Se il mittente vuole inviare al destinatario una sequenza di numeri interi, ad esempio, allora le due parti si devono accordare su come si rappresenti ciascun numero intero (di quanti bit è composto e se il bit più significativo arriva per primo o per ultimo) e su quanti elementi siano contenuti nella sequenza. La Sezione 7.1 descrive varie codifiche per i dati tradizionalmente presenti nei calcolatori, come i numeri interi, i numeri in virgola mobile, le sequenze di caratteri (*stringhe*), gli array e le strutture. Esistono anche ben definiti formati per dati multimediali: le informazioni video, ad esempio, vengono solitamente trasmesse nel formato MPEG (Moving Picture Experts Group), mentre le immagini statiche vengono solitamente trasmesse nel formato JPEG (Joint Photographic Experts Group) oppure GIF (Graphical Interchange Format). Dato che questi formati sono interessanti prin-

cipalmente per l'algoritmo di compressione che utilizzano, li considereremo nel contesto della Sezione 7.2.

Il secondo problema chiave del capitolo, l'efficienza della codifica, è ricco di storia, che si può far risalire al lavoro pionieristico di Shannon sulla teoria dell'informazione degli anni Quaranta del secolo scorso. In effetti, il lavoro in questo ambito è guidato da due forze contrapposte. In una direzione, si vorrebbe inserire nei dati quanta più ridondanza possibile, in modo che il destinatario sia in grado di estrarre i dati corretti anche se nel messaggio sono stati introdotti errori. I codici a rilevazione e a correzione di errori che abbiamo visto nella Sezione 2.4 aggiungono ai messaggi informazioni ridondanti proprio per questo motivo. Nell'altra direzione, vorremmo eliminare dai dati quanto più possibile la ridondanza, in modo da poterli codificare con il minor numero possibile di bit. Questo è l'obiettivo della *compressione dei dati*, di cui parleremo nella Sezione 7.2.

La compressione è importante per i progettisti di reti per un insieme di ragioni, non soltanto perché raramente abbiamo a disposizione banda in abbondanza in tutti i punti della rete. Ad esempio, il modo in cui viene progettato un algoritmo di compressione ha effetto sulla sensibilità nei confronti della perdita di dati o nel loro ritardo, e influenza quindi il progetto dei meccanismi di allocazione delle risorse e i protocolli di trasporto. Di converso, se la rete sottostante non è in grado di garantire una quantità fissa di ampiezza di banda per la durata di una videoconferenza, può darsi che si vogliano progettare algoritmi che possano adattarsi alle modifiche nelle condizioni della rete.

Un aspetto importante, che riguarda tanto il formato di presentazione quanto la compressione dei dati, è la necessità che l'host mittente e l'host destinatario elaborino ogni byte di dati presenti nel messaggio: per questa ragione il formato di presentazione e la compressione vengono a volte chiamate funzioni di *elaborazione dei dati*. Ciò è in contrasto con la maggior parte dei protocolli che abbiamo visto fino a questo punto, che elaborano un messaggio senza prestare attenzione al suo contenuto. A causa di questa necessità di leggere, elaborare e scrivere ogni byte di un messaggio, l'elaborazione dei dati influenza il throughput end-to-end nella rete, per il quale queste elaborazioni possono costituire il fattore limitante.

7.1 Formato di presentazione

Una delle più comuni trasformazioni dei dati in rete consiste nel passaggio dal formato di presentazione usato dai programmi applicativi ad una forma che sia adatta alla trasmissione attraverso la rete, e viceversa: questa trasformazione viene solitamente chiamata *impostazione di presentazione*. Come illustrato nella Figura 7.1, il programma mittente trasforma i dati che vuol trasmettere dalla forma che usa internamente in un messaggio che possa essere trasmesso dalla rete: come si dice, i dati vengono *codificati* in un messaggio. Dal lato ricevente, l'applicazione traduce tale messaggio in arrivo in una rappresentazione che possa essere elaborata dall'applicazione stessa: il messaggio viene *decodificato*. La codifica viene anche, a volte, chiamata *argument marshalling (predisposizione delle informazioni)*, mentre la decodifica viene corrispondentemente detta *unmarshalling*, una terminologia che proviene dal mondo dei protocolli RPC, dove il client crede di invocare una normale procedura con un insieme di parametri ("argomenti"), che però vengono "messi insieme e ordinati un modo opportuno ed efficiente"¹ per dar vita ad un messaggio di rete.

¹ Definizione di *marshalling* presa dal Webster's New Collegiate Dictionary.

7.1 Formato di presentazione

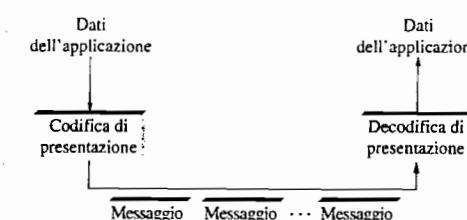


Figura 7.1 La presentazione delle informazioni richiede la codifica e la decodifica dei dati delle applicazioni.

Potreste chiedervi cosa renda questo problema così difficile da rendere ragionevole l'utilizzo di un nome altisonante come "marshalling". Un motivo sta nel fatto che i calcolatori rappresentano i dati in modi diversi. Ad esempio, alcuni calcolatori rappresentano i numeri in virgola mobile nel formato standard IEEE 754, mentre altre macchine usano ancora i propri formati non standard. Anche per dati più semplici, come i numeri interi, architetture diverse usano dimensioni diverse (es: 16 bit, 32 bit, 64 bit). Per complicare ulteriormente le cose, in alcuni elaboratori i numeri interi sono rappresentati in forma *big-endian* (cioè il bit più significativo di una parola composta di due byte si trova nel byte avente indirizzo minore), mentre in altri sono rappresentati in forma *little-endian* (il bit più significativo si trova nel byte con indirizzo maggiore). Il processore Motorola 680x0 è un esempio di architettura big-endian, mentre Intel 80x86 è un esempio di architettura little-endian. Le rappresentazioni big-endian e little-endian del numero intero 34677374 sono indicate in Figura 7.2.

Un altro motivo che rende difficile l'operazione di marshalling consiste nel fatto che i programmi applicativi sono scritti in linguaggi diversi e, anche quando si usa un unico linguaggio, ci sono più compilatori diversi. Ad esempio, i compilatori godono di una certa libertà nel modo in cui dispongono le strutture (*record*) in memoria, per esempio per quanto riguarda la quantità di bit di riempimento (*padding*) fra i campi che compongono la struttura stessa. Di conseguenza, non si può semplicemente trasmettere una struttura da un calcolatore ad un altro, anche se entrambi i calcolatori avessero la stessa architettura e i programmi fossero scritti nello stesso linguaggio, perché il compilatore dell'elaboratore destinatario potrebbe allineare diversamente i campi nella struttura.

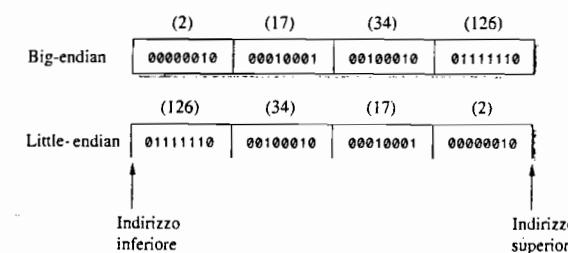


Figura 7.2 Ordinamento tra i byte che rappresentano il numero intero 34677374 in formato big-endian e little-endian.

7.1.1 Tassonomia

Anche se chiunque abbia lavorato nel campo del marshalling vi dirà che non sono argomenti complessi (si tratta, semplicemente, di scambiare dei bit), occorre scegliere tra un numero sorprendente di alternative di progetto. Incominciamo delineando una semplice classificazione dei sistemi di marshalling: quella che segue non è assolutamente l'unica tassonomia possibile, ma è sufficiente a prendere in esame la maggior parte delle alternative interessanti.

Tipi di dati

Come prima cosa ci chiediamo quali tipi di dati devono essere considerati dal sistema. In generale, possiamo classificare i tipi di dati a cui un meccanismo di marshalling fornisce supporto su tre livelli. Ogni livello pone un compito sempre più complesso al sistema di marshalling.

Al livello più basso, un sistema di marshalling opera su un insieme di *tipi di base*, che tipicamente includono i numeri interi, i numeri in virgola mobile e i caratteri. In aggiunta, il sistema potrebbe fornire anche il supporto per i tipi ordinali e i valori logici (*booleani*). Come descritto precedentemente, la scelta su cui si fonda la definizione dell'insieme dei tipi di base consiste nel fatto che il processo di codifica deve poter convertire ciascun tipo di base da una rappresentazione ad un'altra: ad esempio, convertire un numero intero big-endian in un numero intero little-endian.

Al livello successivo troviamo i tipi *flat* (cioè "non gerarchici"): le strutture e gli array. Anche se i tipi non gerarchici potrebbero, a prima vista, non sembrare molto complicati in riferimento al problema del marshalling, in realtà lo sono. Il problema è che i compilatori usati per compilare i programmi applicativi inseriscono, a volte, bit privi di informazione (*padding*) ma utili per l'allineamento dei campi che compongono la struttura, per ottenere un allineamento su confini di parole. Il sistema di marshalling tipicamente *compatta* le strutture in modo che non contengano tali bit.

Al livello più alto, il sistema di marshalling deve gestire *tipi complessi*, che sono costruiti mediante puntatori. In sintesi, la struttura dati che un programma vuole inviare ad un altro potrebbe non essere contenuta in un'unica struttura, ma potrebbe invece coinvolgere puntatori da una struttura ad un'altra: un albero costituisce un valido esempio di tipo complesso che usa i puntatori. Naturalmente, il processo di codifica dei dati deve predisporre la struttura dati in modo opportuno per la trasmissione attraverso la rete, perché i puntatori sono implementati come indirizzi di memoria e il solo fatto che una struttura si trovi in un determinato indirizzo di memoria su un elaboratore non autorizza a ritenere che essa si trovi allo stesso indirizzo su un altro elaboratore. In altre parole, il sistema di marshalling deve *sequenzializzare* (rendere non gerarchiche) le strutture di dati complesse, un processo che prende il nome inglese di "*serialization*".

Riassumendo, in relazione alla complicazione dell'insieme di tipi di dati, il compito di eseguire il marshalling richiede solitamente conversioni fra tipi di base, compattazione di strutture e disposizione in sequenza lineare di strutture di dati complesse, tutto per dare origine ad un messaggio composto da informazioni contigue che possa essere trasmesso attraverso la rete. La Figura 7.3 illustra questo compito.

7.1 Formato di presentazione

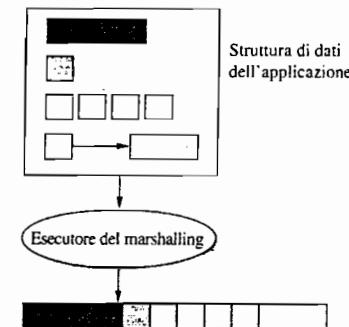


Figura 7.3 Operazione di marshalling: conversione, compattazione e sequenzializzazione.

Strategia di conversione

Una volta che sia stato stabilito il sistema dei tipi, il passo successivo riguarda la decisione in merito a quale strategia di conversione verrà utilizzata dall'esecutore dell'operazione di marshalling. Si pongono, in generale, due opzioni, che prenderemo in esame una dopo l'altra: *forma canonica intermedia* e *fa tutto il ricevitore*.

L'idea della forma canonica intermedia prevede di decidere una rappresentazione esterna per ciascun tipo: l'host mittente, prima di inviare i dati, li traduce dalla propria rappresentazione interna a questa rappresentazione esterna e il destinatario traduce i dati ricevuti da questa rappresentazione esterna alla propria rappresentazione locale. Per rendere l'idea, considerate come dati i numeri interi: gli altri tipi saranno trattati in modo simile. Potreste dichiarare che per la rappresentazione esterna dei numeri interi si usa il formato big-endian: l'host mittente dovrà quindi tradurre ciascun numero intero che vuole inviare nel formato big-endian e l'host destinatario dovrà tradurre gli interi dal formato big-endian a qualsivoglia rappresentazione usu (proprio come avviene in Internet per le intestazioni dei protocolli). Ovviamente, un particolare host potrebbe usare già il formato big-endian, per cui non sarebbe necessaria alcuna conversione.

L'alternativa, che a volte viene chiamata *fa tutto il ricevitore*, prevede che il mittente trasmetta i dati nel proprio formato interno, senza convertire i tipi di base, ma compattando e rendendo sequenziali le strutture dati più complesse. Di conseguenza, il destinatario ha la responsabilità di tradurre i dati dal formato usato dal mittente al proprio formato locale. Il problema di questa strategia è che ogni host deve esser pronto a convertire i dati relativi all'architettura di qualsiasi altro elaboratore. Nel mondo delle reti di calcolatori, in questi casi si parla di *soluzione N per N*: ognuna delle *N* architetture deve essere in grado di gestire tutte le *N* architetture. Al contrario, in un sistema che usa una forma canonica intermedia, ogni host ha bisogno di conoscere solamente come effettuare la conversione tra la propria rappresentazione ed una sola altra rappresentazione, quella esterna.

Usare un formato esterno comune è chiaramente la cosa giusta, vero? Questo è certamente stato il sentimento comune nella comunità delle reti di calcolatori negli ultimi 25 anni, ma la risposta non può essere così netta. Si può notare come non vi siano poi così tante diverse rappresentazioni per i vari tipi di base o, detto in altro modo, *N* non è poi così grande. Inoltre, il caso più comune è che due macchine dello stesso tipo comunichino tra loro: in

questa situazione, sembra assurdo tradurre i dati dalla rappresentazione di quella architettura ad una qualche diversa rappresentazione esterna, soltanto per dover tradurre di nuovo i dati giunti a destinazione nella medesima architettura.

Una terza opzione, anche se non conosciamo alcun sistema che la usi, consiste nell'usare la strategia *fa tutto il ricevitore* se il mittente sa che la destinazione ha la stessa architettura, usando invece una forma canonica intermedia se le due macchine hanno architettura diversa. Come fa il mittente a conoscere l'architettura del destinatario? Potrebbe apprendere questa informazione da un server di nomi oppure eseguendo prima una semplice verifica per vedere se si ottiene il risultato corretto.

Marcatori (*tag*)

Il terzo problema che riguarda il marshalling consiste nel modo in cui il destinatario possa capire quale tipo di dati siano contenuti nel messaggio ricevuto. Esistono due approcci molto utilizzati: i dati *marcati (tagged)* e *non marcati (untagged)*. L'approccio con marcatori è più intuitivo, per cui lo descriveremo per primo.

Un marcatore è una qualsiasi informazione aggiuntiva contenuta in un messaggio, oltre alla concreta rappresentazione dei tipi di base, che aiuti il destinatario a decodificare il messaggio. Esistono diversi possibili marcatori che si possono inserire all'interno di un messaggio. Ad esempio, si potrebbe aggiungere ad ogni dato un contrassegno di *tipo*, che indichi che il valore seguente è un numero intero, oppure un numero in virgola mobile, o altro. Un altro esempio è un contrassegno di *lunghezza*, che viene usato per indicare il numero di elementi di un array o la dimensione di un numero intero. Un terzo esempio è un contrassegno di *architettura*, che potrebbe essere usato in una strategia *fa tutto il ricevitore* per specificare l'architettura in cui sono stati generati i dati contenuti nel messaggio. La Figura 7.4 mostra come si potrebbe codificare in un messaggio marcato un semplice numero intero a 32 bit.

L'alternativa, ovviamente, prevede di non utilizzare i marcatori. In questo caso, come può il destinatario sapere come decodificare i dati? Lo sa perché è stato programmato per saperlo. In altre parole, se invocate una procedura remota che riceve come parametri due numeri interi e un numero in virgola mobile, non c'è ragione che la procedura remota analizzi dei marcatori per scoprire cosa ha appena ricevuto: semplicemente, ipotizza che il messaggio contenga due numeri interi e un numero in virgola mobile, e li decodifica di conseguenza. Notate che questa strategia funziona nella maggior parte delle situazioni, ma non quando si inviano array di lunghezza variabile: in tal caso, si usa comunemente un marcatore di lunghezza per indicare quanto è lungo l'array.

Vale anche la pena di notare che l'approccio senza marcatori implica che il formato di presentazione sia veramente *end-to-end*, perché non è possibile, per un agente intermedio, interpretare il messaggio, a meno che i dati non siano marcati. Vi potreste chiedere: come mai un agente intermedio potrebbe aver bisogno di interpretare un messaggio? Potrebbero essere accadute cose strane, molto probabilmente risultanti da soluzioni particolari a problemi inattesi, per la cui gestione il sistema non era stato progettato. Questo libro non si occupa delle cattive pratiche di progettazione di rete.

<code>tipo = INT</code>	<code>lungh = 4</code>	<code>valore = 417892</code>
-------------------------	------------------------	------------------------------

Figura 7.4 Un numero intero a 32 bit codificato in un messaggio marcato (*tagged*).

7.1 Formato di presentazione

Stub

Uno *stub* ("adattatore") è una sezione di codice che implementa il marshalling e viene solitamente usato per fornire il supporto a RPC. Sul lato client lo stub predisponde i parametri per la procedura all'interno di un messaggio che possa essere trasmesso mediante il protocollo RPC. Sul lato server lo stub converte nuovamente il messaggio in un insieme di variabili che possano essere utilizzate come parametri per invocare la procedura remota. Gli stub possono essere interpretati o compilati.

In un approccio basato sulla compilazione, ogni procedura ha uno stub per il client e uno per il server, entrambi "personalizzati". Anche se è possibile scrivere gli stub a mano, solitamente vengono generati da un compilatore di stub, a partire da una descrizione dell'interfaccia della procedura, come illustrato in Figura 7.5. Poiché lo stub è compilato, è solitamente molto efficiente. In un approccio basato su un interprete, il sistema fornisce stub "generici" per il client e per il server, nei quali i parametri vengono impostati da una descrizione dell'interfaccia della procedura. Dato che questa descrizione è facilmente modificabile, gli stub interpretati hanno il vantaggio di essere flessibili. In pratica, gli stub compilati sono più comuni.

7.1.2 Esempi (XDR, ASN.1, NDR)

Descriviamo ora, brevemente, tre comuni rappresentazioni di dati per la rete, all'interno di questa tassonomia. Per illustrare come funziona ciascun sistema, usiamo il tipo di base per i numeri interi.

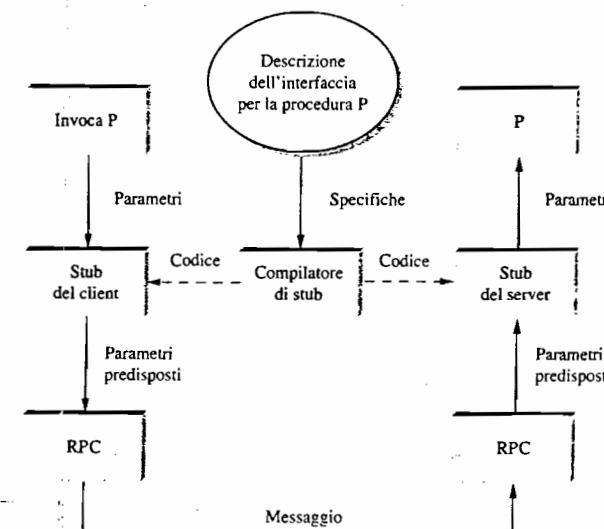


Figura 7.5 Il compilatore di stub riceve in ingresso la descrizione dell'interfaccia e genera in uscita gli stub per il client e per il server.

XDR

La External Data Representation (XDR) è il formato per i dati in rete usato da SunRPC. Nella tassonomia che abbiamo appena presentato, XDR

- fornisce supporto al sistema completo dei tipi del linguaggio C, con l'eccezione dei puntatori a funzione
- definisce una forma canonica intermedia
- non usa marcatori (tranne che per indicare le lunghezze degli array)
- usa stub compilati

Un numero intero XDR è un elemento di dati a 32 bit che codifica un numero intero in C; viene rappresentato in notazione in complemento a due, con il byte più significativo dell'intero in C che si colloca nel primo byte dell'intero in XDR, mentre il byte meno significativo dell'intero in C si trova nel quarto byte dell'intero in XDR: XDR usa, quindi, per i numeri interi un formato big-endian. Esattamente come il linguaggio C, XDR fornisce il supporto per numeri interi con segno e privi di segno.

Il formato XDR rappresenta gli array di lunghezza variabile specificando per prima cosa un numero intero privo di segno (4 byte) che indica il numero di elementi dell'array, seguito da tanti elementi del tipo appropriato quanti ne servono. XDR codifica i campi componenti di una struttura nell'ordine in cui sono dichiarati all'interno della struttura stessa. Per gli array, come per le strutture, la dimensione di ciascun elemento/campo componente viene rappresentata come multiplo di 4 byte: tipi di dati di dimensioni minori vengono estesi fino ad occupare 4 byte con bit di valore 0, con l'unica eccezione per i caratteri, che vengono codificati in un solo byte.

Il frammento di codice seguente fornisce un esempio di una struttura C (`item`) e la funzione XDR che codifica/decodifica tale struttura (`xdr_item`). La Figura 7.6 rappresenta schematicamente la rappresentazione di rete nel formato XDR di questa struttura, nel caso in cui il campo `name` sia lungo sette caratteri e l'array `list` contenga tre valori.

In questo esempio, `xdr_array`, `xdr_int` e `xdr_string` sono tre funzioni primitive messe a disposizione da XDR per codificare e decodificare, rispettivamente, array, numeri interi e stringhe di caratteri. Il parametro `xdrs` è una variabile di "contesto" usata da XDR per tenere traccia della posizione all'interno del messaggio in fase di elaborazione; contiene anche un segnale che indica se si sta usando la funzione per codificare o per decodificare il messaggio. In altre parole, le funzioni come `xdr_item` vengono utilizzate sia nel client sia nel server. Notate che il programmatore dell'applicazione può scrivere la funzione `xdr_item` a mano oppure la può generare usando un compilatore di stub, denominato `rpcgen` (di cui non parla-

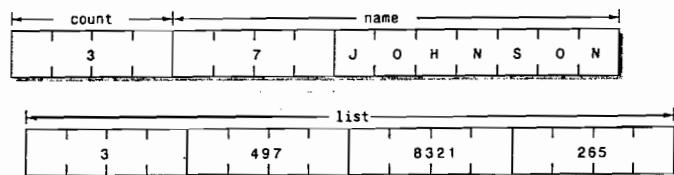


Figura 7.6 Esempio di codifica XDR di una struttura.

mo). Nel secondo caso, `rpcgen` riceve come dato d'ingresso la procedura remota che definisce la struttura `item` e produce in uscita lo stub corrispondente.

```
#define MAXNAME 256;
#define MAXLIST 100;

struct item {
    int count;
    char name[MAXNAME];
    int list[MAXLIST];
};

bool_t
xdr_item(XDR *xdrs, struct item *ptr)
{
    return (xdr_int(xdrs, &ptr->count) &&
            xdr_string(xdrs, &ptr->name, MAXNAME) &&
            xdr_array(xdrs, &ptr->list, &ptr->count, MAXLIST,
                      sizeof(int), xdr_int));
}
```

Le prestazioni di XDR dipendono, naturalmente, dalla complessità dei dati. Nel semplice caso di un array di numeri interi, dove ogni numero deve semplicemente essere convertito scambiando l'ordine dei byte, servono mediamente tre istruzioni per ciascun byte, per cui la conversione dell'intero array è un'operazione le cui prestazioni sono limitate principalmente dalla memoria. Su un tipico elaboratore di oggi, ciò porta da un limite superiore dell'ordine di 100 MBps (800 Mbps). Conversioni più complesse, che richiedano più istruzioni per byte, verranno ovviamente eseguite più lentamente.

ASN.1

Abstract Syntax Notation One (ASN.1) è uno standard ISO che definisce, fra le altre cose, una rappresentazione per i dati che vengono inviati in una rete. La parte di ASN.1 che si occupa della rappresentazione dei dati prende il nome di BER (Basic Encoding Rules). ASN.1 fornisce supporto al sistema dei tipi del linguaggio C (con l'esclusione dei puntatori a funzione), definisce una forma canonica intermedia e usa marcatori per i tipi; gli stub possono essere interpretati o compilati. Uno dei motivi per cui BER di ASN.1 è famosa consiste nel suo utilizzo all'interno dello standard di Internet SNMP (Simple Network Management Protocol). ASN.1 rappresenta ogni elemento di dato con una tripletta avente la forma

`<tag, length, value>`

Il campo `tag` ha tipicamente una dimensione pari a 8 bit, anche se ASN.1 consente la definizione di marcatori che si estendono su più byte. Il campo `length` specifica quanti byte compongono il campo `value` e ne parleremo meglio in seguito. I tipi di dati composti, come le strutture, si possono comporre annidando tipi primitivi, come illustrato in Figura 7.7.

Se il campo `value` è lungo 127 byte o meno, allora il campo `length` viene specificato mediante un unico byte, per cui, ad esempio, un numero intero a 32 bit viene codificato con

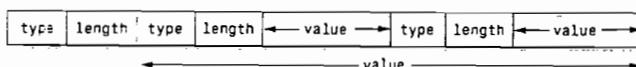


Figura 7.7 Tipi composti creati mediante annidamento con il formato ASN.1 BER.

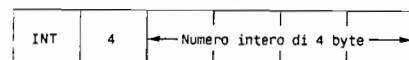


Figura 7.8 Rappresentazione di un numero intero di 4 byte nel formato ASN.1 BER.

un campo *type* di un byte, un campo *length* di un byte e un campo *value* di 4 byte che codifica il numero vero e proprio, come illustrato in Figura 7.8. Il campo *value*, nel caso di un numero intero, viene rappresentato con la notazione in complemento a due e con il formato big-endian, proprio come in XDR. Ricordate, però, che, nonostante il campo *value* del numero intero sia rappresentato esattamente nello stesso modo in ASN.1 e in XDR, la rappresentazione XDR non prevede, in associazione al numero stesso, né il campo *type* né il campo *length*. Questi due marcatori occupano, entrambi, spazio nel messaggio e, ciò che è forse più importante, richiedono un'elaborazione durante le fasi di marshalling e unmarshalling. Questo è uno dei motivi per cui ASN.1 non è efficiente quanto XDR; un altro motivo consiste nel fatto che, essendo ciascun dato preceduto da un campo *length*, è molto improbabile che il valore del dato sia allineato con i confini naturali delle parole (secondo i quali, ad esempio, un numero intero dovrebbe iniziare quando inizia una parola): ciò complica la procedura di codifica e decodifica.

Se il campo *value* è lungo 128 byte o più, allora il campo *length* viene specificato con più byte. A questo punto vi potreste chiedere come mai un campo *length* di un byte possa specificare una lunghezza massima di 127 byte invece che di 256: il motivo risiede nel fatto che un bit del campo *length* viene usato per indicare quanto sia lungo il campo *length* stesso. Uno 0 nell'ottavo bit indica che il campo *length* è di un solo byte. Per specificare che il campo *length* è più lungo, l'ottavo bit viene posto al valore 1, mentre gli altri 7 bit indicano quanti byte ulteriori compongono il campo *length*. La Figura 7.9 illustra un campo *length* semplice, di un solo byte, e un campo *length* multibyte.

NDR

La rappresentazione Network Data Representation (NDR) è lo standard per la codifica dei dati che viene usato nell'ambiente Distributed Computing Environment (DCE). Diversamente da XDR e ASN.1, NDR usa la strategia *fa tutto il ricevitore*, inserendo all'inizio di

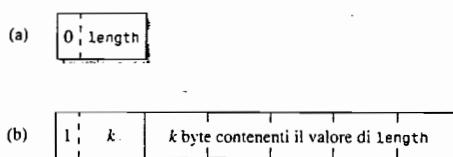


Figura 7.9 Rappresentazione del campo *length* nel formato ASN.1 BER:
(a) un solo byte; (b) più byte.

7.1 Formato di presentazione

ogni messaggio un marcatore che identifica l'architettura, lasciando privi di marcatori i singoli dati e usando un compilatore per generare gli stub. Tale compilatore analizza la descrizione di un programma scritta nel linguaggio IDL (Interface Definition Language) e genera gli stub necessari; IDL assomiglia molto al linguaggio C, per cui sostanzialmente fornisce supporto al sistema di tipi del linguaggio C.

La Figura 7.10 illustra il marcatore che definisce l'architettura, a 4 byte, che viene inserito all'inizio di ogni messaggio codificato mediante NDR. Il primo byte contiene due campi di 4 bit. Il primo campo, *IntegrRep*, definisce il formato per tutti i numeri interi contenuti nel messaggio: un valore 0 in questo campo significa che i numeri avranno il formato big-endian, mentre un valore 1 significa little-endian. Il campo *CharRep* specifica il formato che viene usato per i caratteri: 0 significa ASCII (American Standard Code for Information Interchange), mentre 1 significa EBCDIC (una vecchia alternativa alla codifica ASCII, definita da IBM). Il byte successivo, *FloatRep*, definisce la rappresentazione che viene usata per i numeri in virgola mobile: 0 significa IEEE 754, 1 significa VAX, 2 significa Cray e 3 significa IBM. Gli ultimi due byte sono riservati per sviluppi futuri. Notate che nei casi più semplici, come un array di numeri interi, NDR effettua le medesime elaborazioni di XDR, per cui raggiunge le medesime prestazioni.

7.1.3 Linguaggi di marcatura (XML)

Nonostante abbiamo discusso del problema del formato di presentazione dal punto di vista di RPC (cioè di come codificare i tipi di dati primitivi e le strutture di dati complesse in modo da poterli inviare da un programma client ad un programma server), si ripresenta il medesimo problema in altre situazioni. Ad esempio, come fa un server Web a descrivere una pagina Web in modo che qualsiasi navigatore sappia come visualizzarla sullo schermo? In questo caso specifico, la risposta è il linguaggio HTML (HyperText Markup Language), che indica quali caratteri vanno resi in grassetto o in corsivo, quali dimensioni e tipi di caratteri si devono usare e dove vanno posizionate le immagini. In generale, molte delle applicazioni descritte nel Capitolo 9 hanno una corrispondente rappresentazione standard dei dati.

Nel caso dei server e browser Web, considerare il linguaggio HTML come uno standard per il formato di presentazione è un po' una forzatura, dato che ha più caratteristiche in comune con i linguaggi di impaginazione del testo che con ASN.1 o XDR. Tuttavia, esiste una generalizzazione di HTML che può essere legittimamente considerata una rappresentazione standard dei dati: si tratta di XML, Extensible Markup Language. Diversamente da HTML, che descrive come visualizzare i dati, XML descrive semplicemente i dati, indipendentemente dal fatto che questi vengano visualizzati, elaborati o inviati ad un'altra entità su Internet.

XML è basato su una rappresentazione testuale e assomiglia molto a HTML. Ad esempio, quanto segue è la descrizione, nel linguaggio XML, di un dato che rappresenta un dipendente: viene chiamata *documento XML* e potrebbe essere memorizzata in un file denominato *employee.xml*. La prima riga specifica la versione di XML che viene usata, mentre le righe rimanenti specificano i quattro campi che compongono la struttura di dati di un dipendente;

0	4	8	16	24	31
IntegrRep	CharRep	FloatRep	Extension 1	Extension 2	

Figura 7.10 Il marcatore di architettura usato in NDR.

l'ultimo di tali dati (*hiredate*) contiene tre sottocampi. In altre parole, XML consente all'utente di specificare una struttura annidata di coppie marcatore/valore. Si tratta di qualcosa di simile alla possibilità di rappresentare tipi composti in XDR, ANSI e NDR, ma in un formato che possa essere elaborato da programmi e letto da persone.

```
<?xml version="1.0"?>
<employee>
  <name>John Doe</name>
  <title>Head Bottle Washer</title>
  <id>123456789</id>
  <hiredate>
    <day>5</day>
    <month>June</month>
    <year>1986</year>
  </hiredate>
</employee>
```

Ciò che manca in questo esempio sono le istruzioni su come vada interpretato il documento. Ad esempio, l'anno (*year*) "1986" è una stringa o un numero intero, e se è un numero intero quale rappresentazione si usa? La prima parte della risposta è che in XML ogni cosa è basata sul testo, per cui ciò che deve fare un'entità per interpretare una struttura XML ricevuta da un'altra entità è soltanto analisi del testo. In altre parole, il codice ASCII viene usato in XML come rappresentazione intermedia comune. Dopo che l'entità ricevente ha effettuato l'analisi del testo, però, rimane ancora da sapere quali operazioni siano consentite sui singoli campi dei dati. Ad esempio, è consentito sottrarre gli anni fra loro? E quali operazioni si possono compiere con gli identificativi, *id*? Come si possa visualizzare tale struttura di dati è un'altra domanda che ci possiamo porre, e proprio per questo motivo è corretto vedere XML come una generalizzazione di HTML.

Per risolvere questo problema, XML consente agli utenti di definire uno *schema*, che è il termine usato nell'ambito delle basi di dati per specificare come si deve interpretare un insieme di dati. Quanto segue è uno schema relativo all'esempio precedente. Notate che lo schema è, anch'esso, un documento XML valido, solitamente memorizzato nel file *employee.xsd*.

```
<?xml version="1.0"?>
<xss:schema xmlns:xss="http://www.cs.princeton.edu/XMLSchema"
  targetNamespace="http://www.cs.princeton.edu"
  xmlns="http://www.cs.princeton.edu"
  elementFormDefault="qualified">

  <xss:element name="employee">
    <xss:complexType>
      <xss:sequence>
        <xss:element name="name" type="xss:string"/>
        <xss:element name="title" type="xss:string"/>
        <xss:element name="id" type="xss:string"/>
        <xss:element name="hiredate">
          <xss:complexType>
```

```
<xss:sequence>
  <xss:element name="day" type="xss:integer"/>
  <xss:element name="month" type="xss:string"/>
  <xss:element name="year" type="xss:integer"/>
</xss:sequence>
</xss:complexType>
</xss:element>
</xss:sequence>
</xss:complexType>
</xss:element>
</xss:schema>
```

Le righe iniziali del documento contengono informazioni relative al contesto nel quale viene definito lo schema, dettagli che non sono importanti per la nostra trattazione. Per quanto riguarda le righe successive, esiste una relazione evidente con il documento *employee.xml* definito in precedenza. Ad esempio:

```
<xss:element name="title" type="xss:string"/>
```

afferma che il valore del campo *title* è una stringa. XML contiene parecchi tipi predefiniti, tra i quali:

```
xss:string
xss:decimal
xss:integer
xss:boolean
xss:date
xss:time
```

Va detto, ed è importante, che un documento XML può fare riferimento allo schema che definisce come vanno interpretati i dati: lo si fa sostituendo, nel nostro esempio *employee.xml*, la seconda riga con la seguente:

```
<employee xmlns="http://www.cs.princeton.edu/XMLSchema"
  xmlns:xsi="http://www.cs.princeton.edu/XMLSchema-instance"
  xsi:schemaLocation="http://www.cs.princeton.edu/schema/employee.xsd">
```

Anche in questo caso non ci interessano i dettagli: l'informazione essenziale è che lo schema relativo a questo documento si trova nel file *employee.xsd*, che si può trovare in Internet all'indirizzo

<http://www.cs.princeton.edu/schema/employee.xsd>

Non vi sorprenderà il fatto che XML fornisca il supporto per un insieme di funzionalità ben più vasto di quanto illustrato in questo semplice esempio, ma, in fondo, l'idea su cui si basa XML è elementare: definire una sintassi per descrivere i dati che le applicazioni possono condividere tramite Internet.

7.2 Compressione dei dati

A volte i programmi applicativi hanno bisogno di spedire, con particolari vincoli temporali, più dati di quanti possano essere accettati dall'ampiezza di banda della rete. Ad esempio, un'applicazione video potrebbe voler trasmettere un flusso video di 10 Mbps, ma avere a disposizione soltanto una rete a 10 Mbps: come ben sa chiunque abbia usato Internet, è raro riuscire a trasferire dati tra due punti di Internet ad una velocità che si avvicini a 1 Mbps. Inoltre, nel momento in cui scriviamo il modello di allocazione delle risorse di Internet è fortemente condizionato dal fatto che le singole applicazioni non utilizzino, su una linea congestionata, una quantità di banda maggiore della loro "frazione equa". Per tutti questi motivi è spesso importante *comprimere* i dati alla sorgente, per poi trasmetterli attraverso la rete ed, infine, *decomprimere* nel ricevitore.

Per molti aspetti la compressione è strettamente connessa alla codifica dei dati: nel momento in cui pensiamo a come codificare un insieme di dati in un insieme di bit, potremmo pensare anche a come codificare quei dati nel più piccolo insieme di bit possibile. Ad esempio, se avete un blocco di dati composto dai 26 simboli dell'alfabeto, da A a Z, e tutti questi simboli hanno la stessa probabilità di essere presenti nel blocco di dati che state codificando, allora la cosa migliore che potete fare è codificare ogni simbolo con 5 bit (perché $2^5 = 32$ è la più piccola potenza di 2 superiore a 26). Se, invece, il simbolo R è presente il 50% delle volte, allora sarebbe una buona idea usare un numero minore di bit per codificare R di quanti se ne usano per gli altri simboli. In generale, se è nota la probabilità relativa con cui ciascun simbolo ricorre nei dati, si può assegnare un diverso numero di bit a ciascun simbolo possibile, in modo da minimizzare il numero di bit richiesto per codificare un blocco di dati assegnato. Si tratta dell'idea che sta alla base dei *codici di Huffman*, uno dei primi e più importanti sviluppi nel campo della compressione dei dati.

Gli algoritmi di compressione si suddividono in due classi. La prima, detta *compressione priva di perdite* ("lossless compression"), garantisce che i dati ripristinati dopo il processo di compressione e decompressione siano esattamente uguali ai dati originali. Un algoritmo di compressione privo di perdita di informazione viene utilizzato per comprimere file di dati, come codice eseguibile, file di testo e dati numerici, perché i programmi che elaborano questi file di dati non possono tollerare errori nei dati stessi. Al contrario, la *compressione con perdite* ("lossy compression") non garantisce che i dati ripristinati siano assolutamente identici ai dati inviati, perché un algoritmo con perdite elimina alcune informazioni che non possono più essere ricostruite. Auspicabilmente, però, l'informazione perduta non sarà essenziale per il ricevitore. Gli algoritmi con perdite di informazione vengono utilizzati per comprimere immagini statiche, video e audio: una cosa sensata, perché tali dati contengono spesso più informazioni di quante l'occhio o l'orecchio umano ne possano percepire, ed è accettabile che contengano errori e imperfezioni che il cervello umano è in grado di compensare. Ancora, gli algoritmi con perdite raggiungono solitamente rapporti di compressione molto più elevati delle loro controparti prive di perdite: possono essere migliori anche per un ordine di grandezza.

Si potrebbe pensare che comprimere i dati prima di spedirli possa essere sempre una buona idea, perché la rete consegnerebbe i dati compressi in un tempo minore di quanto può fare con i dati non compressi. Tuttavia, ciò non è sempre vero. Gli algoritmi di compressione e decompressione sono spesso onerosi dal punto di vista del tempo di calcolo. La domanda che vi dovete porre è se valga la pena perdere tempo per la compressione e la decompressione, in considerazione di fattori come la velocità del processore nell'host e l'ampiezza di banda

7.2 Compressione dei dati

della rete. In particolare, se B_c è il valore medio dell'ampiezza di banda con cui i dati possono essere elaborati dal compressore e dal decompressore (posti in serie), B_n è l'ampiezza di banda della rete (che comprende anche i tempi di elaborazione della rete stessa) per i dati non compressi e r è il rapporto di compressione medio, e se ipotizziamo che tutti i dati vengano compressi prima che inizi la trasmissione dei dati stessi, allora il tempo necessario per inviare x byte di dati non compressi è

$$x/B_n$$

mentre il tempo richiesto per comprimerli e spedirli in forma compressa è

$$x/B_c + x/(r B_n)$$

Di conseguenza, la compressione è utile se

$$x/B_c + x/(r B_n) < x/B_n$$

che è equivalente alla relazione

$$B_c > r/(r - 1) \times B_n$$

Ad esempio, per un rapporto di compressione uguale a 2, B_c dovrebbe essere maggiore di $2 \times B_n$ perché la compressione avesse senso.

Per molti algoritmi di compressione è possibile che non si debba comprimere *l'intero* insieme di dati prima di iniziare la trasmissione (altrimenti i sistemi di videoconferenza non potrebbero funzionare), ma che sia sufficiente memorizzare una certa quantità di dati (ad esempio, pochi frame video). In questo caso, nell'equazione precedente come valore di x si userebbe la quantità di dati necessaria per "riempire la condutture" (*fill the pipe*).

Ovviamente, quando si parla di algoritmi di compressione con perdita di informazione, le risorse di elaborazione non sono l'unico fattore. In relazione all'applicazione in esame, gli utenti possono voler esplorare compromessi assai diversi fra l'ampiezza di banda (o il ritardo) e la quantità di informazione che viene perduta per effetto della compressione. Ad esempio, è difficile che un radiologo che esamina una mammografia possa tollerare una perdita significativa nella qualità dell'immagine, mentre potrebbe ben tollerare un ritardo di alcune ore per ottenere l'immagine attraverso una rete. Al contrario, è ormai ben evidente che molte persone sarebbero disposte a sopportare una discutibile qualità nell'audio pur di effettuare conversazioni telefoniche gratuite (per non parlare della possibilità di parlare al telefono mentre si guida).

7.2.1 Algoritmi di compressione senza perdita di informazione

Iniziamo presentando tre algoritmi di compressione senza perdita di informazione. Non li descriviamo in grande dettaglio, vediamo solamente l'idea fondamentale, perché la maggiore utilità nelle reti di oggi proviene dagli algoritmi con perdita di informazione usati per comprimere dati relativi ad immagini e a flussi video. Ciò nonostante, faremo qualche commento in merito a come funzionino sulle immagini digitali questi algoritmi senza perdite. Alcune delle idee che stanno alla base di queste tecniche di compressione senza perdite si

ritroveranno nelle sezioni successive, quando prenderemo in esame gli algoritmi con perdite che vengono utilizzati per comprimere le immagini.

Run Length Encoding

RLE (Run Length Encoding) è una tecnica di compressione veramente semplice, che usa un approccio brutale: sostituire occorrenze consecutive di un certo simbolo con una sola copia del simbolo stesso e un conteggio del numero di occorrenze (da cui il nome *run length*, "lunghezza della serie"). Ad esempio, la stringa AAABBCDDDD verrebbe codificata con 3A2B1C4D.

L'algoritmo RLE può essere usato per comprimere immagini digitali confrontando i valori di pixel adiacenti e, quindi, codificando soltanto le modifiche: una tecnica piuttosto efficace per quelle immagini che abbiano ampie regioni omogenee. Ad esempio, non è insolito che, per immagini di testo digitalizzato, RLE riesca ad ottenere rapporti di compressione dell'ordine di 8 a 1: RLE funziona bene con questo tipo di file perché spesso contengono una grande quantità di spazio bianco, che può essere rimosso. Non per niente RLE è l'algoritmo di compressione utilizzato per la trasmissione dei fax. Nonostante ciò, nel caso di immagini con un grado di variabilità locale anche modesto, non è insolito che la compressione aumenti, in realtà, la dimensione dell'immagine in byte, perché servono 2 byte per rappresentare un singolo simbolo, anche quando quel simbolo non viene ripetuto.

Differential Pulse Code Modulation

Un altro semplice algoritmo di compressione privo di perdite è DPCM (Differential Pulse Code Modulation). In questo caso, l'idea consiste nell'emettere, per prima cosa, un simbolo di riferimento e, in seguito, per ogni simbolo presente nei dati, emettere la differenza tra tale simbolo e il simbolo di riferimento. Ad esempio, usando il simbolo A come simbolo di riferimento, la stringa AAABBCDDDD verrebbe codificata con A0001123333, perché A è uguale al simbolo di riferimento, B differisce di 1 rispetto al simbolo di riferimento, e così via. Notate che questo semplice esempio non rende evidenti i veri vantaggi di DPCM, che consistono nel fatto che, quando le differenze sono piccole, possono essere codificate con un numero di bit minore di quello necessario per codificare il simbolo stesso. In questo esempio, le differenze, nell'intervallo da 0 a 3, possono essere rappresentate con 2 bit ciascuna, invece che con i 7 o 8 bit richiesti per il carattere completo. Non appena la differenza diventa troppo elevata, viene selezionato un nuovo simbolo di riferimento.

La tecnica DPCM funziona meglio di RLE per la maggior parte delle immagini digitali, perché si avvantaggia del fatto che pixel adiacenti sono solitamente simili. A causa di tale correlazione, la dinamica delle differenze tra i valori di pixel adiacenti può essere assai inferiore alla dinamica dell'immagine originale e tale dinamica può quindi essere rappresentata usando un numero minore di bit. Usando DPCM per comprimere immagini digitali, abbiamo misurato fattori di compressione di 1.5 a 1.

Un approccio poco diverso, denominato *codifica delta*, codifica semplicemente un simbolo come la sua differenza dal simbolo precedente: ad esempio, AAABBCDDDD sarebbe rappresentato come A001011000. Notate che la codifica delta funziona probabilmente bene per codificare immagini dove i pixel adiacenti sono simili tra loro. È anche possibile comprimere usando RLE dopo aver eseguito la codifica delta, poiché, nel caso in cui ci fossero molti simboli simili in successione, potremmo avere lunghe sequenze di zeri.

Metodi basati su dizionario

L'ultimo metodo di compressione senza perdite che prendiamo in considerazione è l'appuccio basato su dizionario, di cui l'algoritmo di compressione Lempel-Ziv (LZ) è il più noto. Il comando `compress` di Unix usa una variante dell'algoritmo LZ.

Un algoritmo di compressione basato su dizionario costruisce un dizionario (una tabella) di stringhe di lunghezza variabile (immaginatele come se fossero frasi comuni) che vi aspettate di trovare all'interno dei dati, poi sostituisce ciascuna di queste stringhe, quando compare nei dati, con il corrispondente indice nel dizionario. Ad esempio, invece di lavorare con singoli caratteri nei dati di tipo testo, potreste considerare ciascuna parola come una stringa ed usare l'indice di tale parola all'interno del dizionario. Per rimanere su questo esempio, la parola "compression" corrisponde all'indice 4978 in un particolare dizionario, quello contenuto nel file di Unix `/usr/share/dict/words`. Per comprimere un testo, ogni volta che compare la stringa "compression", questa verrebbe sostituita con 4978. Poiché questo particolare dizionario contiene solamente 25000 parole, servirebbero 15 bit per codificare questo indice, per cui la stringa "compression" potrebbe essere rappresentata con 15 bit piuttosto che con i 77 bit necessari nella rappresentazione ASCII a 7 bit. Un fattore di compressione 5 a 1!

Ovviamente, tutto questo lascia irrisolto il problema di stabilire quale sia il dizionario. Una possibilità consiste nella definizione di un dizionario statico, preferibilmente uno che sia adatto ai dati che si stanno comprimendo. Una soluzione più generale, che è quella usata nella compressione LZ, consiste in una definizione adattativa del dizionario, in base al contenuto dei dati che si stanno comprimendo. In questo caso, tuttavia, il dizionario costruito durante la compressione deve essere inviato insieme ai dati, in modo che la parte di algoritmo che effettua la decompressione possa svolgere il proprio compito. Determinare in modo preciso come si possa costruire un dizionario adattativo è stato oggetto di molte ricerche e alla fine del capitolo presenteremo le pubblicazioni più significative sull'argomento.

Una variante dell'algoritmo LZ viene usata per comprimere immagini digitali nel formato GIF (Graphic Interchange Format). Prima di attuare la compressione, GIF trasforma le immagini con colori a 24 bit in immagini con colori a 8 bit, identificando i colori usati nell'immagine, che sono solitamente in numero molto inferiore a 2^{24} , scegliendo poi i 256 colori che approssimano meglio i colori usati nell'immagine. Tali colori vengono memorizzati in una tabella, per la quale si può usare un indice a 8 bit, ed il valore di ciascun pixel viene sostituito dal valore dell'indice corrispondente. Notate che si tratta di una compressione con perdite per ogni immagine che abbia più di 256 colori. Successivamente, GIF esegue sul risultato una compressione con una variante dell'algoritmo LZ, considerando le sequenze di pixel più frequenti come se fossero stringhe che compongono il dizionario. Usando questo approccio, in alcuni casi con il formato GIF si possono ottenere rapporti di compressione di 10 a 1, soltanto, però, quando l'immagine è composta da un numero relativamente basso di colori ben diversi tra loro. Immagini di scene naturali, che spesso contengono una gamma di colori più continua, non vengono compresse con tali rapporti dallo standard GIF. In un altro esperimento siamo riusciti ad ottenere un rapporto di compressione 2 a 1 applicando il comando `compress` di Unix, che è basato sull'algoritmo LZ, al codice sorgente per i protocolli descritti in questo libro.

7.2.2 Compressione di immagini (JPEG)

L'uso, in continuo aumento negli ultimi anni, delle immagini digitali (uso che ha ricevuto impulso dall'invenzione di terminali grafici, piuttosto che di reti ad alta velocità), ha accre-

sciuto in modo sempre più critico il bisogno di algoritmi di compressione progettati per dati relativi ad immagini digitali. In risposta a questa esigenza, ISO ha definito un formato per immagini digitali denominato JPEG, dal nome del gruppo che l'ha progettato, Joint Photographic Experts Group ("Joint", *congiunto*, nella sigla JPEG sta a significare che si è trattato di uno sforzo *congiunto* degli enti di standardizzazione ISO e ITU). Questa sezione descrive l'algoritmo di compressione che costituisce il cuore di JPEG, mentre la sezione successiva descrive un formato correlato, MPEG, usato per i dati di tipo video.

Prima di descrivere la compressione JPEG, è necessario puntualizzare che JPEG, GIF e MPEG sono più di semplici algoritmi di compressione: definiscono anche il *formato* per i dati relativi ad immagini o a video, in modo molto simile a ciò che fanno XDR, NDR e ASN.1 nel definire il formato per dati di tipo numerico e per le stringhe. Tuttavia, questa sezione si concentra sugli aspetti di questi standard che sono in relazione alla compressione.

La compressione JPEG ha luogo in tre fasi, come illustrato in Figura 7.11. Per effettuare la compressione, l'immagine viene elaborata da queste tre fasi, un blocco di dimensioni 8×8 per volta. La prima fase applica la trasformata coseno discreta (DCT, *discrete cosine transform*) al blocco. Se pensate all'immagine come ad un segnale nel dominio spaziale, allora DCT trasforma questo segnale in un equivalente segnale nel dominio della *frequenza spaziale*. Si tratta di un'operazione priva di perdita di informazione, ma predispone i dati per la fase successiva, con perdite. Dopo aver eseguito la trasformazione mediante DCT, la seconda fase effettua una quantizzazione del segnale risultante e, nel compiere questa operazione, si vengono a perdere le informazioni meno significative contenute nel segnale. La terza fase codifica il risultato finale, ma, nel farlo, aggiunge una compressione priva di perdite alla compressione con perdite ottenuta nelle prime due fasi. La decompressione segue le tre medesime fasi, ovviamente in ordine inverso.

La discussione che segue descrive ciascuna fase in maggiore dettaglio, con la semplificazione di considerare solamente immagini in toni di grigio; alla fine della sezione parleremo anche delle immagini a colori. Nel caso delle immagini in toni di grigio, ogni pixel dell'immagine è caratterizzato da un valore a 8 bit che indica la luminosità del pixel, dove 0 significa bianco e 255 significa nero.

Fase DCT

La trasformazione DCT è strettamente correlata alla trasformata veloce di Fourier (FFT, *fast Fourier transform*): considera come valore di ingresso una matrice 8×8 di valori di pixel e produce in uscita una matrice 8×8 di coefficienti di frequenza. Potete pensare alla matrice di ingresso come ad un segnale di 64 punti, definito in due dimensioni spaziali (x e y): la trasformazione DCT scomponete questo segnale in 64 frequenze spaziali. Per comprendere, dal punto di vista intuitivo, cosa sia una frequenza spaziale, immaginate di spostarvi all'interno di un'immagine, ad esempio nella direzione x : vedrete il valore di ciascun pixel variare secondo una qualche funzione di x . Se questo valore cambia lentamente al variare di x , allora ha una frequenza spaziale bassa, mentre se cambia valore rapidamente ha una frequenza spaziale

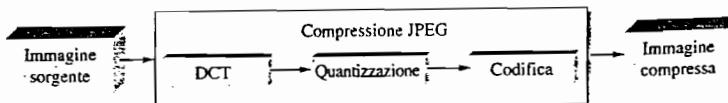


Figura 7.11 Schema a blocchi della compressione JPEG.

7.2 Compressione dei dati

elevata. Di conseguenza, le basse frequenze corrispondono alla caratteristica della figura nelle sue linee principali, mentre le altre frequenze corrispondono ai dettagli. L'idea che sta alla base della trasformazione DCT è quella di separare le caratteristiche essenziali, che sono di grande importanza per la visualizzazione dell'immagine, dai dettagli, che sono meno importanti e, in alcuni casi, possono essere percepiti a fatica dall'occhio umano.

La trasformazione DCT, insieme alla sua inversa che viene eseguita durante la decompressione, è definita dalle formule seguenti:

$$DCT(i,j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} pixel(x,y) \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

$$pixel(x,y) = \frac{1}{\sqrt{2N}} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} C(i)C(j) DCT(i,j) \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}} & \text{se } x=0 \\ 1 & \text{se } x>0 \end{cases}$$

dove $pixel(x,y)$ è il valore, nella scala dei toni di grigio, del pixel che si trova nella posizione (x,y) all'interno del blocco 8×8 in fase di compressione; in questo caso, $N = 8$.

Il primo coefficiente di frequenza, nel punto $(0,0)$ della matrice di uscita, viene chiamato *componente continua* (*DC coefficient*): dal punto di vista intuitivo, si può vedere come si tratti di una misura del valore medio dei 64 pixel nella matrice di ingresso. Gli altri 63 elementi della matrice di uscita vengono chiamati *coefficienti AC* e aggiungono a questo valore medio l'informazione relativa alle frequenze spaziali sempre più elevate. Di conseguenza, procedendo dal primo coefficiente di frequenza verso il sessantaquattresimo coefficiente, ci si sposta dalle informazioni in bassa frequenza alle informazioni in alta frequenza, cioè dalle caratteristiche più grossolane dell'immagine ai dettagli sempre più piccoli. Questi coefficienti di frequenza sempre più elevate sono sempre meno importanti per la qualità percepita dell'immagine. La seconda fase della compressione JPEG decide quale frazione di questi coefficienti vada eliminata.

Fase di quantizzazione

La seconda fase di JPEG è quella in cui avviene la perdita di informazione in seguito alla compressione. La fase DCT, in sé, non provoca perdita di informazione: trasforma solamente l'immagine in un formato che rende più semplice l'individuazione di quale informazione vada rimossa (anche se, ovviamente, esiste una qualche forma di perdita di informazione anche durante la fase DCT per effetto dell'aritmetica in virgola fissa). La quantizzazione è semplice da capire: si tratta di eliminare i bit non significativi dei coefficienti di frequenza. Per vedere come funziona la fase di quantizzazione, immaginate di voler comprimere dei numeri interi minori di 100, ad esempio 45, 98, 23, 66 e 7. Se avete deciso che la conoscenza di questi numeri troncati al più prossimo multiplo di 10 è sufficiente per i vostri scopi, allora potete dividere ciascun numero per il valore di quantizzazione (*quantum*) 10 usando l'aritmetica dei numeri interi, ottenendo 4, 9, 2, 6 e 0. Questi numeri possono poi essere codificati con 4 bit anziché usare i 7 bit necessari per codificare i numeri originali.

Invece di usare lo stesso intervallo di quantizzazione per tutti i 64 coefficienti, JPEG usa una tabella di quantizzazione che fornisce questo valore per ogni coefficiente, come specificato dalla formula seguente. Potete pensare a questa tabella (Quantum) come ad un parametro da poter impostare per controllare la quantità di informazione che viene perduta e, corrispondentemente, quale rapporto di compressione si ottenga. In pratica, lo standard JPEG specifica un insieme di tabelle di quantizzazione che si sono dimostrate efficaci nella compressione di immagini digitali, un esempio delle quali è fornito nella Tabella 7.1. In tabelle come questa, i coefficienti relativi alle frequenze inferiori hanno un intervallo di quantizzazione prossimo a 1 (per cui l'informazione delle basse frequenze non viene perduta), mentre i coefficienti delle frequenze più elevate hanno valori maggiori (provocando, quindi, una maggiore perdita di informazione per tali frequenze). Notate che, come risultato di queste tabelle di quantizzazione, molti dei coefficienti ad alta frequenza risultano azzerati dopo la quantizzazione, rendendoli soggetti a compressione durante la terza fase.

L'equazione fondamentale della quantizzazione è

$$\text{QuantizedValue}(i, j) = \text{IntegerRound}(DCT(i, j)/\text{Quantum}(i, j))$$

dove

$$\text{IntegerRound}(x) = \begin{cases} \lfloor x + 0,5 \rfloor & \text{se } x \geq 0 \\ \lfloor x - 0,5 \rfloor & \text{se } x < 0 \end{cases}$$

La decompressione viene quindi definita, semplicemente, come

$$DCT(i, j) = \text{QuantizedValue}(i, j) \times \text{Quantum}(i, j)$$

Ad esempio, se il coefficiente DC, cioè $DCT(0, 0)$, per un certo blocco era uguale a 25, allora la quantizzazione di questo valore, usando la Tabella 7.1, risulterebbe essere

$$\lfloor 25/3 + 0,5 \rfloor = 8$$

Durante la decompressione questo coefficiente verrebbe ripristinato come

$$8 \times 3 = 24$$

Tabella 7.1 Esempio di tabella di quantizzazione per JPEG.

Quantum =	3 5 7 9 11 13 15 17
	5 7 9 11 13 15 17 19
	7 9 11 13 15 17 19 21
	9 11 13 15 17 19 21 23
	11 13 15 17 19 21 23 25
	13 15 17 19 21 23 25 27
	15 17 19 21 23 25 27 29
	17 19 21 23 25 27 29 31

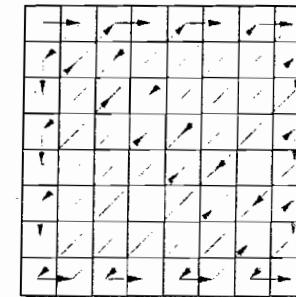


Figura 7.12 Elaborazione dei coefficienti di frequenza quantizzati secondo uno schema a zigzag.

Fase di codifica

La fase finale della compressione JPEG codifica i coefficienti di frequenza quantizzati in forma compatta, producendo un'ulteriore compressione, che risulta essere priva di perdita di informazione. Iniziando dal coefficiente DC in posizione (0, 0), vengono elaborati tutti i coefficienti, procedendo a zigzag secondo la sequenza mostrata in Figura 7.12. Lungo questo percorso a zigzag viene usato una forma di codifica di tipo RLE (si applica RLE ai soli coefficienti di valore 0, ma si tratta di una situazione rilevante perché molti degli ultimi coefficienti hanno valore 0). I singoli valori dei coefficienti vengono poi codificati usando un codice di Huffman (lo standard JPEG consente anche l'utilizzo di una codifica aritmetica al posto del codice di Huffman).

Dato che, poi, il coefficiente DC contiene una frazione elevata dell'informazione globale contenuta nel blocco 8×8 dell'immagine sorgente e tipicamente le immagini cambiano poco da un blocco all'altro, ogni coefficiente DC viene codificato come differenza rispetto al coefficiente DC precedente: la codifica delta descritta nella Sezione 7.2.1.

Immagini a colori

La discussione precedente si è basata sull'ipotesi che ogni pixel fosse costituito da un singolo valore in una scala di toni di grigio; nel caso di un'immagine a colori, si possono scegliere molte diverse rappresentazioni per ciascun pixel. Una rappresentazione, detta RGB (*red*, *green*, *blue*), rappresenta ogni pixel con tre componenti di colore: rosso, verde e blu. La rappresentazione RGB è quella usata solitamente nei dispositivi grafici di ingresso e uscita. Un'altra rappresentazione, chiamata YUV, ha anch'essa tre componenti: una componente di luminanza (Y) e due componenti di crominanza (U e V). YUV, esattamente come RGB, è un sistema di coordinate tridimensionale, ma, se confrontato con RGB, le coordinate appaiono ruotate per determinare una corrispondenza migliore con l'apparato visivo umano: una condizione di vantaggio, in quanto l'apparato visivo umano non è sensibile in modo uniforme ai diversi colori. Ad esempio, siamo in grado di distinguere la luminanza (luminosità) di un pixel molto meglio della sua saturazione (colore).

Una domanda interessante è: come mai queste tre componenti, in ognuna delle due rappresentazioni, si possono combinare per produrre colori accettabili? Una risposta banale è che sono stati anche definiti sistemi di colore con due sole coordinate, ma si è verificato che non erano adeguati per riprodurre fedelmente i colori, così come vengono percepiti dall'uomo. Ciò che è importante per la nostra discussione è che ogni pixel di un'immagine a colori è caratteriz-

zato da tre valori separati. Per comprimere un'immagine di questo tipo, ognuna di queste tre componenti viene elaborata indipendentemente dalle altre, esattamente come abbiamo visto per i singoli valori della scala di grigi. In altre parole, potete pensare ad un'immagine colorata come se fosse costituita da tre immagini separate, sovrapposte l'una all'altra per ricreare l'immagine da visualizzare. Notate che, in generale, la tecnica JPEG non è limitata ad immagini con tre componenti: usando JPEG è possibile comprimere un'immagine multispettrale.

JPEG contiene un certo numero di varianti che controllano la quantità di compressione che si può ottenere in funzione della fedeltà dell'immagine, usando, ad esempio, diverse tabelle di quantizzazione. Queste varianti, aggiunte al fatto che immagini diverse hanno caratteristiche diverse, rendono impossibile dire con precisione quale sia il fattore di compressione ottenibile con JPEG. In senso generale, però, è ampiamente accettato il fatto che la tecnica JPEG sia in grado di comprimere immagini con colori a 24 bit con un fattore di compressione all'incirca di 30 a 1: per prima cosa l'immagine viene compressa di un fattore 3 riducendo i colori a 24 bit in colori a 8 bit (come descritto per il formato GIF), poi si comprime di un ulteriore fattore 10 usando l'algoritmo descritto in questa sezione.

7.2.3 Compressione video (MPEG)

Rivolghiamo ora la nostra attenzione al formato MPEG, che prende il nome dal Moving Picture Experts Group che l'ha definito. In prima approssimazione, un'immagine in movimento (ad esempio, un video) è semplicemente una successione di immagini statiche, chiamate anche *frame* o riquadri, visualizzate ad una determinata velocità. Ciascuno di questi frame può venire compresso usando la stessa tecnica basata sulla trasformazione DCT che abbiamo usato in JPEG, ma fermarsi a questo punto sarebbe un errore, perché non verrebbe eliminata la ridondanza di informazione, dovuta alla correlazione fra due riquadri successivi, che è presente in una sequenza video. Ad esempio, due frame successivi di un video conterranno informazioni quasi identiche se la scena non è molto movimentata, per cui non è necessario inviare due volte le medesime informazioni. Anche quando c'è movimento, ci può essere parecchia ridondanza, perché un oggetto in movimento potrebbe essere sempre identico a se stesso da un frame all'altro: in alcuni casi, è solo la posizione ad essere modificata. MPEG si occupa proprio di questa ridondanza di correlazione fra frame successivi. MPEG definisce anche un meccanismo per codificare un segnale audio insieme al segnale video, ma in questa sezione esaminiamo solamente gli aspetti video di MPEG.

Tipi di frame

MPEG riceve in ingresso una sequenza di frame video e li comprime in tre tipi di frame, chiamati *frame I* (intra-riquadro), *frame P* (riquadro predetto) e *frame B* (riquadro bidirezionale). Ogni frame in ingresso viene compresso per dare origine ad uno di questi tipi di frame. I frame di tipo *I* si possono immaginare come riquadri di riferimento: sono autonomi, non dipendono da frame precedenti, né da frame successivi. In prima approssimazione, un frame *I* è semplicemente la versione compressa, con JPEG, del corrispondente frame nel video sorgente. I frame *P* e *B* non sono autonomi, ma sono specificati mediante differenze relative a qualche frame di riferimento. Più in dettaglio, un frame *P* è specificato come differenza rispetto al frame *I* precedente, mentre un frame *B* fornisce un'interpolazione fra due frame *I* o *P* precedenti e successivi.

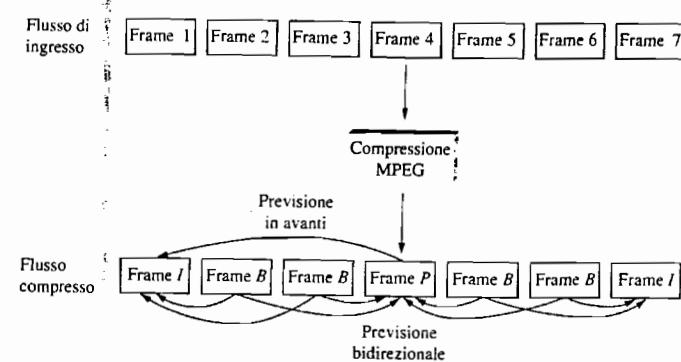


Figura 7.13 Sequenza di frame *I*, *P* e *B* generati da MPEG.

La Figura 7.13 illustra una sequenza di sette frame video che, dopo essere stati compressi mediante MPEG, producono una sequenza di frame *I*, *P* e *B*. I due frame *I* sono indipendenti: ognuno dei due può essere decompresso dal ricevitore indipendentemente da qualsiasi altro frame. Il frame *P* dipende dal precedente frame *I* e può essere decompresso dal ricevitore soltanto se è arrivato anche il precedente frame *I*. Ciascun frame *B* dipende sia dal frame *I* o *P* precedente, sia dal frame *I* o *P* seguente: entrambi questi frame di riferimento devono arrivare al ricevitore prima che MPEG possa decomprimere il frame *B* per riprodurre il frame video originale.

Dal momento che ogni frame *B* dipende da un frame che, nella sequenza temporale, si trova più avanti, i frame compressi non vengono trasmessi in ordine sequenziale, ma, ad esempio, la sequenza *I B B P B B I* mostrata nella Figura 7.13 viene trasmessa nell'ordine *I P B B I B B I*. Ancora, MPEG non definisce il rapporto fra frame di tipo *I* e frame di tipo *P* e *B*: questo rapporto può variare in base alla compressione e alla qualità dell'immagine richieste. Ad esempio, è possibile trasmettere soltanto frame di tipo *I*, situazione che sarebbe simile all'utilizzo di JPEG per comprimere il video.

Diversamente da quanto fatto in precedenza per JPEG, la discussione seguente pone l'attenzione sulla *decodifica* di un flusso MPEG, perché è un'operazione un po' più semplice da descrivere e anche perché è questa l'operazione che viene più spesso implementata oggi nei sistemi di rete, perché la codifica MPEG è così onerosa che viene normalmente eseguita *offline* (cioè non in tempo reale). Ad esempio, in un sistema di *video a richiesta* (*video-on-demand*), i file video verrebbero codificati e memorizzati su disco prima di iniziare il servizio. Quando un utente vuole vedere un video, verrebbe trasmesso alla macchina dell'utente il flusso MPEG, che verrebbe decodificato e visualizzato in tempo reale.

Esaminiamo con maggiore attenzione i tre tipi di frame. Come già detto, i frame *I* sono assai simili alla versione compressa dei frame sorgenti. La differenza principale consiste nel fatto che MPEG funziona con unità, dette *macroblocco*, aventi dimensione 16×16 . Per un video a colori rappresentato nel sistema di coordinate YUV, le componenti U e V in ciascun macroblocco vengono "sotto-campionato" per generare un blocco 8×8 , cioè ad ogni sottoblocco 2×2 nel macroblocco vengono assegnati un valore U e un valore V, la media dei valori assunti dai quattro pixel. Per quanto riguarda la componente Y, il sottoblocco mantiene i quattro valori originali. Si può operare in questo modo perché le componenti U e V possono

essere trasmesse con minore accuratezza senza disturbare l'immagine in modo visibile, in quanto l'occhio umano è meno sensibile ai colori di quanto lo sia alla luminosità. La Figura 7.14 mostra la relazione esistente tra un frame e i macroblocchi corrispondenti.

Anche i frame *P* e *B* vengono elaborati usando i macroblocchi come unità di base. Dal punto di vista intuitivo, possiamo dire che l'informazione relativa ad ogni macroblocco cattura il movimento presente nel video, cioè mostra in quale direzione e per quale distanza si è mosso il macroblocco, relativamente al frame di riferimento. Nel seguito mostriamo come si usa un frame *B* per ricostruire un frame durante la decompressione; i frame *P* vengono elaborati in modo simile, tranne per il fatto che dipendono da un unico frame di riferimento anziché da due.

Prima di entrare nel dettaglio di come venga decompresso un frame *B*, notiamo subito che un macroblocco di un frame *B* non è necessariamente definito in relazione ad un frame precedente e ad uno successivo, come abbiamo detto in precedenza, ma può, invece, essere specificato in relazione ad uno solo di essi. In effetti, un certo macroblocco di un frame *B* può usare la medesima codifica interna usata in un frame *I*. È stata prevista questa flessibilità per quei casi in cui l'immagine in movimento cambia troppo rapidamente: a volte può aver senso inserire una codifica completa (*intracoding*), piuttosto che una codifica predittiva in avanti o all'indietro. Di conseguenza, ogni macroblocco di un frame *B* contiene un campo "tipo" che indica quale codifica è stata usata per il macroblocco stesso. Nella discussione che segue, però, consideriamo solamente il caso generale in cui il macroblocco usa la codifica predittiva bidirezionale.

In tal caso, ogni macroblocco di un frame *B* viene rappresentato da un quaterna: (1) una coordinata che identifica il macroblocco all'interno del frame, (2) un vettore di spostamento relativo al frame di riferimento precedente, (3) un vettore di spostamento relativo al frame di riferimento successivo, e (4) un variazione, δ , per ogni pixel del macroblocco (che indica quanto sia cambiato il pixel in relazione ai due pixel di riferimento). Per ogni pixel del macroblocco, la prima cosa da fare è identificare il pixel di riferimento corrispondente nei frame di riferimento passato e futuro. Questo compito viene assolto usando i due vettori di spostamento associati al macroblocco; successivamente, si aggiunge la variazione δ alla media fra questi due pixel di riferimento. Detto con maggior precisione, se indichiamo con F_p e F_f rispettivamente, i frame di riferimento passato e futuro, e i vettori di riferimento corrispon-

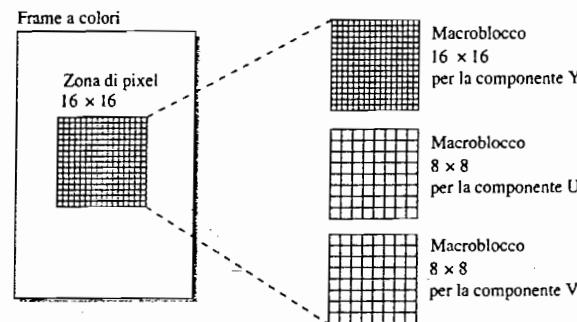


Figura 7.14 Un frame visto come insieme di macroblocchi.

7.2 Compressione dei dati

denti sono indicati con (x_p, y_p) e (x_f, y_f) , allora il pixel avente coordinate (x, y) nel frame attuale (indicato come F_c) viene calcolato con la formula seguente:

$$F_c(x, y) = (F_p(x + x_p, y + y_p) + F_f(x + x_f, y + y_f))/2 + \delta(x, y)$$

dove δ è la variazione relativa al pixel, così come specificato all'interno del frame *B*. Queste variazioni sono codificate nello stesso modo dei pixel nel frame *I*, cioè vengono elaborati da una trasformazione DCT e poi quantizzati. Poiché queste variazioni sono tipicamente piccole, la maggior parte dei coefficienti DCT valgono 0 dopo la quantizzazione, per cui si possono comprimere in modo efficace.

Dovrebbe essere abbastanza chiaro, dalla discussione precedente, come si possa eseguire la codifica, con un'unica eccezione. Quando, durante la compressione, si genera un frame *P* o *B*, MPEG deve decidere come posizionare i macroblocchi. Ricordate che ogni macroblocco di un frame *P*, ad esempio, è definito in relazione ad un macroblocco di un frame *I*, ma che il macroblocco del frame *P* non deve essere necessariamente avere, nel frame, la stessa posizione che ha il macroblocco corrispondente nel frame *I*: la differenza tra le posizioni è data dal vettore di spostamento. Si dovrebbe scegliere un vettore di spostamento tale da rendere il macroblocco del frame *P* il più possibile simile al corrispondente blocco del frame *I*, in modo che i valori di variazione (δ) per tale macroblocco siano minimi. Questo significa che occorre individuare il movimento degli oggetti presenti nell'immagine, da un frame al successivo: si tratta di un problema di *stima del moto* e sono note parecchie tecniche (euristiche) per risolverlo (nella sezione "Ulteriori letture", al termine del capitolo, parleremo di alcuni lavori che affrontano questo problema). La difficoltà di questo problema è uno dei motivi per cui la codifica MPEG richiede tempi più lunghi della decodifica, a parità di hardware. Lo standard MPEG non indica alcuna tecnica particolare: definisce solamente il formato per la codifica di questa informazione nei frame *B* e *P* e l'algoritmo per ricostruire i pixel durante la decompressione, come già descritto.

Efficienza e prestazioni

La codifica MPEG raggiunge solitamente un rapporto di compressione di 90 a 1, anche se a volte si ottengono addirittura rapporti 150 a 1. In termini dei singoli tipi di frame, ci possiamo attendere un rapporto di compressione di circa 30 a 1 per i frame di tipo *I* (coerentemente con i rapporti che si ottengono usando JPEG, dopo aver trasformato i colori a 24 bit in colori a 8 bit), mentre i rapporti di compressione per i frame *P* e *B* sono tipicamente da tre a cinque volte inferiori di quelli relativi ad un frame *I*. Senza la preliminare riduzione dei colori, da 24 a 8 bit, la compressione che si può ottenere con MPEG è tipicamente compresa tra 30 a 1 e 50 a 1.

Lo standard MPEG richiede calcoli molto intensivi. La compressione viene tipicamente eseguita *offline*, fattore non problematico quando di predispongono filmati per un servizio di tipo *video-on-demand*. Oggi si può effettuare la compressione in tempo reale usando hardware specifico, ma anche le implementazioni software stanno rapidamente colmando la lacuna. Per la compressione, sono disponibili schede video MPEG di basso costo, che fanno poco più di una ricerca indicizzata nella tabella dei colori YUV, che, fortunatamente, è la fase più dispendiosa in termini di tempo di elaborazione. La maggior parte della decodifica MPEG viene oggi eseguita tramite software: negli ultimi anni i processori sono diventati sufficientemente veloci da tenere il passo con velocità video di 30 frame al secondo, eseguendo la decodifica di flussi MPEG senza alcun ausilio di hardware specifico. Ad esempio, anche un

modesto processore a 600 MHz è in grado di decomprimere filmati MPEG abbastanza velocemente da visualizzare un flusso video 640×480 (pixel) alla velocità di 30 frame al secondo.

Altri standard per la codifica video

Terminiamo segnalando che MPEG non è l'unico standard disponibile per la codifica video. Ad esempio, ITU-T ha definito la "serie H" per la codifica di dati multimediali in tempo reale, che comprende standard per flussi video, flussi audio, informazioni di controllo e operazioni di multiplazione (coordinando audio, video e dati in un unico flusso di bit). All'interno della serie H, troviamo gli standard H.261 e H.263, la prima e la seconda generazione di standard per la codifica video. Diversamente da MPEG, che ha come obiettivo flussi con velocità dell'ordine di 1.5 Mbps, gli standard H.261 e H.263 sono stati progettati per le velocità tipiche delle linee ISDN, cioè possono gestire flussi video su linee con ampiezza di banda variabile con incrementi di 64 Kbps. In linea di principio, sia H.261 che H.263 assomigliano molto allo standard MPEG: usano la trasformazione DCT, la quantizzazione e la compressione inter-riquadro. Le differenze rispetto a MPEG consistono soltanto in piccoli dettagli.

Si stanno anche diffondendo nuove versioni di MPEG, in particolare MPEG-4. Il modo migliore per descrivere MPEG-4 consiste nel considerarlo una generalizzazione di MPEG per la gestione di video naturali e sintetici (cioè generati dal calcolatore). Il suo funzionamento prevede di considerare ogni scena (frame) come un insieme di oggetti video, piuttosto che elaborare soltanto macroblocchi rettangolari. MPEG-4 ha anche l'obiettivo di fornire supporto a velocità di bit inferiori (ad esempio, adatte a dispositivi palmari wireless), diversamente da MPEG-2 (si veda nel seguito), che ha invece come obiettivo i flussi video a larga banda (adatti, ad esempio, per la televisione ad alta definizione, HDTV). A tutt'oggi, comunque, l'aspetto più importante di MPEG-4 è la sua compatibilità rispetto a MPEG-2, per cui la maggior parte dei video MPEG-4 oggi disponibili sfrutta ancora la stima del moto e i meccanismi di compressione appena descritti.

Infine, in un campo di applicazione non molto lontano dalla compressione video troviamo gli standard per la codifica di animazioni, come quelle che spesso compaiono quando si naviga nel Web. L'esempio più diffuso sembra essere il formato FLASH di Macromedia, che in sostanza è un protocollo per specificare un insieme di poligoni e linee (gli elementi costitutivi dei video generati al calcolatore), con un sequenza di vettori che indicano come questi oggetti si muovano sulla scena al trascorrere del tempo. In questo senso, FLASH non è veramente un algoritmo di compressione, quanto, piuttosto, uno standard di codifica.

7.2.4 Trasmettere MPEG attraverso la rete

Come già detto in precedenza, MPEG non definisce soltanto una modalità di compressione di flussi video, ma specifica anche il formato di un video compresso secondo lo standard MPEG. Analogamente, JPEG e GIF definiscono un formato per immagini statiche. Se ci concentriamo su MPEG, la prima cosa da ricordare è che definisce il formato di un *flusso video* (*video stream*), senza indicare come suddividere tale flusso in pacchetti di rete. Di conseguenza, si può usare MPEG per video memorizzati su un disco, così come per video trasmessi attraverso una connessione di rete orientata al flusso, come quella fornita dal protocollo TCP. A breve daremo maggiori dettagli sul modo di suddividere un flusso MPEG in pacchetti.

Il formato MPEG è uno dei più complicati fra tutti i protocolli presentati in questo libro, complicazione che deriva dal desiderio di lasciare all'algoritmo di codifica tutti i gradi di libertà possibili sul modo di codificare un certo flusso video, oltre che dall'evoluzione dello standard nel tempo (es: MPEG-1 e MPEG-2). Ciò che descriviamo ora è il *profilo principale* di un flusso video MPEG-2, profilo che si può considerare analogo ad una "versione", tranne per il fatto che il profilo non viene specificato esplicitamente in un'intestazione MPEG, ma il ricevitore lo deve desumere dalla combinazione di campi dell'intestazione che vede.

Il profilo principale di un flusso MPEG-2 ha una struttura annidata, come evidenziato in Figura 7.15 (ma tenete presente che questa figura nasconde un *sacco* di complicati dettagli). Al livello più esterno, il video contiene una sequenza di gruppi di immagini (GOP, group of pictures), separati da un campo SeqHdr; la sequenza è terminata da un SeqEndCode (0xb7). Il campo SeqHdr che precede ogni GOP specifica, fra le altre cose, la dimensione di ciascuna immagine (frame) presente nel GOP (misurata sia in pixel sia in macroblocchi), l'intervallo di tempo fra due immagini (misurato in ms) e due matrici di quantizzazione per i macroblocchi che appartengono al GOP: una per i macroblocchi intracodificati (blocchi I) e una per i macroblocchi intercodificati (blocchi P). Dato che queste informazioni vengono fornite per ogni GOP (piuttosto che una sola volta per l'intero flusso video, come vi sareste potuti immaginare), è possibile modificare la tabella di quantizzazione e la velocità di frame ai confini tra due GOP, lungo tutto il filmato: in questo modo il flusso video è adattativo, nel tempo, come diranno in seguito.

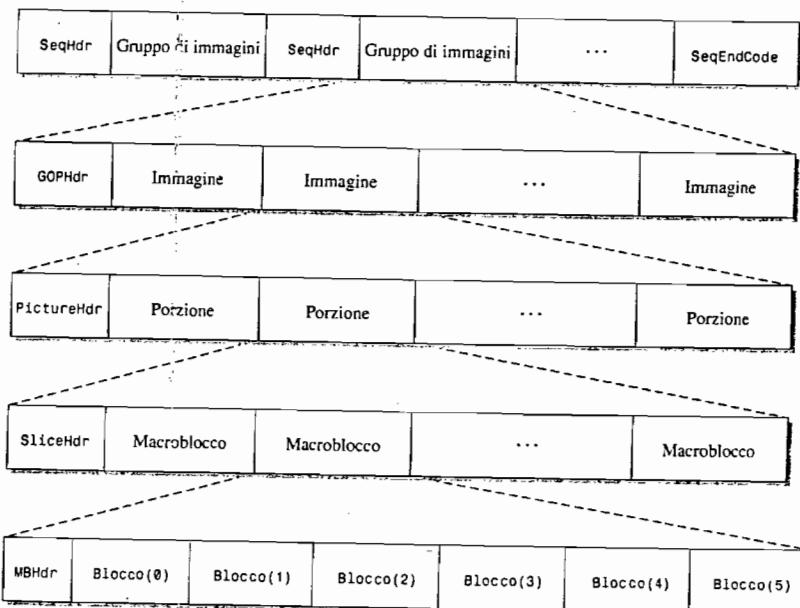


Figura 7.15 Formato di un flusso video compresso con MPEG.

Ogni GOP è composto da un campo GOPHdr, seguito dall'insieme di immagini che compongono il GOP stesso. Il campo GOPHdr specifica il numero di immagini presenti nel GOP, oltre ad informazioni di sincronizzazione (ad esempio, quando deve essere riprodotto il GOP, relativamente all'inizio del filmato). Ogni immagine, a sua volta, è descritta da un campo PictureHdr e da un insieme di *porzioni* (*slice*) che compongono l'immagine (una porzione è una regione dell'immagine, ad esempio una linea orizzontale). Il campo PictureHdr identifica il tipo di immagine (*I*, *B* o *P*), oltre ad una tabella di quantizzazione specifica per l'immagine. Il campo SliceHdr indica la posizione verticale della porzione, oltre a dare un'ulteriore opportunità per modificare la tabella di quantizzazione, questa volta tramite un fattore di scala costante invece che descrivendo un'intera tabella nuova. Successivamente, il campo SliceHdr è seguito da una sequenza di macroblocchi. Infine, ogni macroblocco contiene un'intestazione che specifica l'indirizzo del macroblocco all'interno dell'immagine, insieme ai dati per i sei blocchi che compongono il macroblocco: un valore per la componente U, uno per la componente V e quattro per le componenti Y (ricordate che la componente Y è 16×16 , mentre le componenti U e V sono 8×8).

Dovrebbe essere chiaro che uno dei punti di forza del formato MPEG consiste nel fatto che fornisce al codificatore la possibilità di modificare la codifica stessa al trascorrere del tempo. Si può cambiare la velocità di frame, la risoluzione, la proporzione fra i vari tipi di frame che definiscono un GOP, la tabella di quantizzazione e la codifica usata per singoli macroblocchi. Di conseguenza, è possibile adattare la velocità a cui si trasmette un filmato attraverso una rete con un compromesso fra la qualità delle immagini e l'ampiezza di banda della rete. Come possa, precisamente, un protocollo di rete sfruttare questa proprietà di adattamento è, oggi, oggetto di intense ricerche (si veda il riquadro).

Un altro aspetto interessante dell'invio di un flusso MPEG attraverso una rete è il modo in cui il flusso viene suddiviso in pacchetti. Se lo si invia lungo una connessione TCP, la suddivisione in pacchetti non è un problema: è il protocollo TCP a decidere quando ha memorizzato un numero di byte sufficiente ad inviare il successivo datagramma IP. Tuttavia, quando si usano filmati interattivi, raramente si trasmette con il protocollo TCP, perché la ritrasmissione dei segmenti perduti, tipica di TCP, potrebbe introdurre una latenza inaccettabile. Se stessimo trasmettendo un video usando, ad esempio, UDP, allora avrebbe senso scomporre il flusso in punti accuratamente selezionati, come i confini tra macroblocchi, perché questo limiterebbe gli effetti della perdita di un pacchetto ad un solo macroblocco, piuttosto che provocare danni a più macroblocchi in seguito alla perdita di un solo pacchetto. Questo è proprio un esempio di *framing* al livello di applicazione (Application Level Framing), di cui parleremo più approfonditamente nella Sezione 9.3, dove prenderemo in esame le applicazioni video.

La scomposizione del flusso in pacchetti è soltanto il primo problema da risolvere nella trasmissione in rete di un filmato compresso con MPEG. Il successivo, complesso problema è la gestione della perdita di pacchetti. Da un lato, se la rete elimina un frame di tipo *B*, è possibile riprodurre nuovamente il frame precedente senza compromettere in modo sensibile la qualità del filmato: un frame su 30 non è un grosso problema. D'altra parte, la perdita di un frame di tipo *I* ha conseguenze più gravi, perché nessuno dei successivi frame *B* e *P* potrà essere elaborato. Di conseguenza, la perdita di un frame di tipo *I* avrebbe come risultato la perdita di più frame del filmato. Pur potendo ritrasmettere il frame *I* mancante, il ritardo che ne risulterebbe non sarebbe probabilmente accettabile in una videoconferenza in tempo reale. Una soluzione a questo problema sarebbe l'utilizzo delle tecniche di tipo Differentiated Services descritte nella Sezione 6.5.3, per contrasse-

Codifica adattativa di filmati

Abbiamo già messo in evidenza che la codifica video che usa MPEG consente un compromesso tra l'ampiezza di banda utilizzata e la qualità dell'immagine. Di converso, dovrebbe essere chiaro che l'ampiezza di banda d'uscita di un algoritmo di compressione video operante con un certo livello di qualità non sarà, in generale, costante, ma varierà nel tempo in relazione alla quantità di dettagli e di movimento presenti nel flusso video. Queste caratteristiche sollevano alcuni problemi interessanti in merito a come progettare un sistema di trasporto per filmati compressi attraverso una rete a pacchetti.

Supponiamo di avere un codificatore video che produce un flusso video compresso ad una velocità media di R bps, ma che ogni tanto emette raffiche con velocità fino a $3R$ bps. Potenzialmente, potremmo trasmettere il flusso video lungo un canale con ampiezza di banda costante (ad esempio, una linea noleggiata o un circuito CBR) di capacità R , a patto di far transitare il flusso video attraverso un "buffer di smoothing" che appiani i picchi istantanei della velocità di trasmissione. Ora, potrebbe accadere, ad un certo punto, che il buffer di smoothing si riempia oltre un livello accettabile, magari a causa di una prolungata azione movimentata che da luogo ad un lungo periodo con uscite velocità elevata da parte del codificatore. A questo punto, potremmo aumentare per un po' di tempo il rapporto di compressione, riducendo così la velocità dei dati (e la qualità dell'immagine) e consentendo lo svuotamento del buffer di smoothing. Quando il buffer è prossimo a svuotarsi completamente, potremmo aumentare nuovamente la qualità della codifica.

In una rete a commutazione di pacchetto potremmo fare più o meno la stessa cosa, senza aver bisogno di un buffer di smoothing. Immaginiamo di avere un modo per misurare la capacità ancora disponibile ed il livello di congestione lungo il percorso, usando, ad esempio, un algoritmo di controllo della congestione basato su equazioni, come quelli descritti nella Sezione 6.5.5. Al variare dell'ampiezza di banda disponibile, possiamo trasferire tale informazione al codificatore, in modo che possa modificare i propri parametri di codifica per rallentare durante la congestione e spedire in modo più aggressivo (e con maggiore qualità dell'immagine) quando la rete è scarica. Si tratta di un comportamento analogo a quello del protocollo TCP, tranne per il fatto che, nel caso dei filmati, stiamo realmente modificando la quantità totale dei dati che vengono inviati, piuttosto che il tempo impiegato per inviare una quantità fissa di dati, dato che non vogliamo introdurre ritardi in un'applicazione video.

Si ha un problema interessante nel caso in cui si voglia inviare un flusso video a più ricevitori con una trasmissione *multicast*. Come facciamo a scegliere la velocità corretta per ogni ricevitore, dal momento che vi possono essere livelli di congestione assai diversi lungo i diversi percorsi? Una soluzione ingegnosa a questo problema consiste nel suddividere il video trasmesso in più "strati" (*layer*). Il primo strato contiene il livello basilare di dettaglio, necessario per vedere un'immagine di una qualche utilità, mentre ciascuno strato successivo aggiunge maggiori dettagli, costituiti da informazioni di frequenza sempre più elevata. Ogni strato può essere, poi, inviato ad un diverso indirizzo di gruppo multicast, e ogni ricevitore può decidere a quanti strati unirsi. Se il ricevitore A è sottoposto ad elevata congestione, potrebbe unirsi solamente al gruppo multicast che trasporta lo strato di base, mentre il ricevitore B potrebbe unirsi a tutti i gruppi, per ricevere tutti gli strati. Il ricevitore A potrebbe, poi, tentare periodicamente di unirsi al

(continua)

(seguito)

gruppo che trasporta lo strato successivo, corrispondente al successivo livello di dettaglio, per vedere se sia disponibile una maggiore ampiezza di banda. Questo approccio prende il nome di RLM, *receiver-driven layered multicast* (multicast stratificato e guidato dal ricevitore). Un problema interessante per la ricerca consiste nel creare il corretto insieme di incentivi per fare in modo che un ricevitore si unisca al numero di gruppi appropriato, piuttosto che unirsi semplicemente a tutti, perché unendosi a troppi gruppi si provocherebbe una congestione di rete non necessaria.

gnare i pacchetti contenenti frame di tipo *I* con una probabilità di eliminazione inferiore di quella degli altri pacchetti.

Un'ultima osservazione riguarda il fatto che la scelta di come codificare il filmato non dipende solamente dall'ampiezza di banda disponibile nella rete, ma anche dai vincoli di latenza dell'applicazione. Ancora una volta, un'applicazione interattiva, come la videoconferenza, richiede valori di latenza inferiori. Il fatto critico è il modo in cui i frame di tipo *I*, *P* e *B* si combinano nel GOP. Considerate il seguente GOP:

I B B B P P B B B I

Il problema che questo GOP pone ad un'applicazione di videoconferenza consiste nel fatto che la sorgente deve ritardare la trasmissione dei quattro frame *B* finché non sia disponibile il frame *P* o *I* seguente, perché ciascun frame *B* dipende dal frame *P* o *I* successivo. Se il video viene riprodotto alla velocità di 15 frame al secondo (cioè un frame ogni 67 ms), ciò significa che il primo frame *B* viene ritardato di 4×67 ms, che è più di un quarto di secondo. Questo ritardo va aggiunto ad ogni ritardo di propagazione imposto dalla rete. Un quarto di secondo è molto maggiore della soglia di 100 ms che gli umani sono in grado di percepire: è per questa ragione che molte applicazioni di videoconferenza codificano il video usando JPEG, spesso chiamato anche *JPEG in movimento* (motion-JPEG), che risolve anche il problema dell'eliminazione di un frame di riferimento, perché tutti i frame sono indipendenti l'uno dall'altro. Notate, però, che una codifica interframe che dipenda solamente dai frame precedenti e non dai frame successivi non è un problema. Così, un GOP così fatto

I P P P P I

funzionerebbe bene per videoconferenze interattive.

7.2.5 Compressione audio (MP3)

Lo standard MPEG non definisce solamente come vengano compressi i dati video, ma definisce anche uno standard per la compressione audio. Questo standard può essere usato per comprimere la parte audio di un filmato (nel qual caso lo standard MPEG definisce come l'audio compresso venga intervallato al video compresso in un unico flusso MPEG) oppure per comprimere audio a sé stante (ad esempio, un CD audio).

Per comprendere la compressione audio, occorre iniziare dai dati. L'audio di "qualità CD", che è lo standard di fatto per la rappresentazione digitale dell'audio di elevata qualità.

viene campionato alla velocità di 44.1 KHz (cioè si raccoglie un campione ogni 23 ms circa). Ogni campione è di 16 bit, per cui un flusso audio stereo (cioè a 2 canali) produce una velocità di bit pari a

$$2 \times 44.1 \times 1000 \times 16 = 1.41 \text{ Mbps}$$

Per fare un confronto, la voce di qualità telefonica viene campionata alla velocità di 8 KHz, con campioni di 8 bit, dando luogo ad una velocità di bit di 64 Kbps, che, non a caso, è la velocità di una linea ISDN.

È chiaramente necessaria una qualche forma di compressione per trasmettere un segnale audio di "qualità CD" lungo, diciamo, una coppia di linee voce/dati ISDN con una capacità di 128 Kbps. Per peggiorare le cose, la sincronizzazione e la correzione d'errore richiedono di usare 49 bit per codificare ciascun campione di 16 bit, con una velocità di bit risultante di

$$49/16 \times 1.41 \text{ Mbps} = 4.32 \text{ Mbps}$$

Lo standard MPEG va incontro a questa necessità definendo tre livelli di compressione, elencati nella Tabella 7.2. Di questi, il più usato è il Layer III, più comunemente noto come MP3.

Per ottenere questi rapporti di compressione, MP3 usa tecniche simili a quelle usate da MPEG per comprimere dati video. Dapprima suddivide il flusso audio in un certo numero di sottobande di frequenza, in modo vagamente analogo a come MPEG elabora separatamente le componenti Y, U e V di un flusso video. Come secondo passo, ogni sottobanda viene scomposta in una sequenza di blocchi, che sono simili ai macroblocchi di MPEG, tranne per il fatto che sono di lunghezza variabile, da 64 a 1024 campioni (l'algoritmo di codifica può modificare la dimensione del blocco in relazione a certi effetti di distorsione che vanno ben oltre lo scopo di questo libro). Infine, ogni blocco viene trasformato usando un algoritmo DCT modificato, per poi essere quantizzato e codificato con codice di Huffman, esattamente come avviene per il video in MPEG.

La parte critica di MP3 consiste nel numero di sottobande che vengono usate e nel numero di bit allocati per ciascuna sottobanda, ricordando che si sta cercando di produrre la più elevata qualità audio possibile per una velocità di bit prefissata. Il modo preciso per effettuare questa allocazione dipende da modelli psicoacustici, che non rientrano negli obiettivi di questo libro, ma, per rendere l'idea, pensate che è più sensato assegnare più bit alle sottobande di frequenza più bassa quando si sta comprimendo una voce maschile, mentre è preferibile assegnare più bit alle sottobande di frequenza più elevata quando si comprime una voce femminile. Dal punto di vista operativo, per ottenere l'effetto desiderato, MP3 modifica dinamicamente le tabelle di quantizzazione usate per ogni sottobanda.

Tabella 7.2 Velocità della compressione MP3.

Codifica	Velocità di bit	Fattore di compressione
Layer I	384 Kbps	4
Layer II	192 Kbps	8
Layer III	128 Kbps	12

Dopo aver compresso le sottobande, queste vengono compattate in frame di dimensioni fisse, aggiungendo un'intestazione, che contiene informazioni di sincronizzazione e informazioni sull'allocazione dei bit necessarie al decodificatore per determinare quanti bit sono stati usati per codificare ciascuna sottobanda. Come accennato in precedenza, questi frame audio possono essere intercalati con frame video per dar corpo ad un flusso MPEG completo. Una nota a margine, piuttosto interessante, consiste nell'osservare che, qualora nella rete si verifichi una situazione di congestione, mentre l'eliminazione di frame video di tipo B potrebbe funzionare ragionevolmente bene, l'esperienza insegna che eliminare frame audio non sia una buona idea, perché gli utenti sono in grado di tollerare meglio un video di scarsa qualità piuttosto che un audio scadente.

7.3 Riepilogo

Questo capitolo ha descritto come i dati delle applicazioni vengono codificati in pacchetti per la rete. Diversamente dai protocolli descritti precedentemente nel libro, che potete immaginare che elaborino *messaggi*, queste trasformazioni elaborano *dati*.

Il primo problema è il formato di presentazione, che riguarda il formato dei diversi tipi di dati su cui operano i programmi applicativi: numeri interi, numeri in virgola mobile, sequenze di caratteri (stringhe), array e strutture. Questo richiede la traslazione dei byte, passando dall'ordine usato nei calcolatori all'ordine usato nella rete, e la *serializzazione* delle strutture dati composte. Abbiamo esplorato lo spazio delle alternative di progetto per il formato di presentazione e abbiamo discusso tre meccanismi specifici che occupano punti diversi all'interno di tale spazio: XDR, ASN.1 e NDR.

Il secondo argomento affrontato è la compressione, che si occupa di ridurre l'ampiezza di banda necessaria per trasmettere diversi tipi di dati. Gli algoritmi di compressione possono essere privi di perdita o con perdita di informazione, gli ultimi essendo più adatti per i dati relativi ad immagini e a filmati. JPEG, MPEG e MP3 sono esempi di protocolli di compressione con perdita di informazione per, rispettivamente, immagini statiche, filmati e dati di tipo audio.

Problema aperto

Le reti di calcolatori e l'elettronica di consumo

Abbiamo presentato lo standard MPEG come se fosse stato progettato per comprimere i dati video in modo da poterli trasmettere attraverso reti a commutazione di pacchetto, ma ovviamente non è così: MPEG è un generico formato video che è applicabile allo stesso modo ad un film memorizzato su DVD come ad segnale televisivo trasmesso in HDTV. Tutto questo punta ad una convergenza fra calcolatori, reti ed elettronica di consumo.

In un futuro non troppo lontano ci possiamo attendere che, nelle case, esista un *portale multimediale* (MG, *media gateway*), posto tipicamente accanto al televisore, in sostituzione dei decoder satellitari di oggi. Questo MG sarà connesso ad un fornitore di servizi Internet (ISP, *Internet service provider*), probabilmente mediante lo stesso supporto fisico utilizzato per la TV via cavo, che già arriva nelle case. Il portale multimediale avrà anche un certo numero di porte che consentiranno di collegare diversi dispositivi elettronici di consumo, come una videocamera digitale, un riproduttore di DVD, un videogame, e così via. A tutt'oggi

Ulteriori letture

gi, sembra che la modalità di connessione condivisa tra questi dispositivi possa essere lo standard Firewire, una linea seriale a 400 Mbps sviluppata da Apple, ma esistono altre possibilità, come varie tecnologie wireless.

Cosa sarà richiesto a tale MG? Da una parte veicolerà flussi multimediali fra dispositivi diversi, in modo molto simile a come oggi i router IP inoltrano pacchetti di dati fra porte. Ad esempio, potrebbe essere possibile inviare un filmato dei bambini ripresi con la videocamera digitale verso la linea di collegamento all'ISP e poi attraverso la rete, per i nonni in paziente attesa. Un'altra cosa che dovrà fare sarà la traduzione tra i protocolli di TCP/IP di Internet e i formati caratteristici dei vari dispositivi: è ovviamente possibile che prima o poi anche le videocamere diventino nodi di Internet a tutti gli effetti (e abbiano, ad esempio, un proprio indirizzo IP), ma questi portali multimediali rimanderanno al futuro la necessità di connettere tutto ad Internet.

La prospettiva di un'ampia disponibilità di "elettrodomestici in Internet" solleva parecchi problemi interessanti, il primo dei quali è l'utilizzo dello spazio di indirizzamento. Il protocollo IP, nella sua versione 6, è stato progettato con l'obiettivo di espandere lo spazio di indirizzamento IP in modo così vasto che l'assegnazione di indirizzi IP a qualunque oggetto immaginabile (tostapane, contatori dell'acqua, ecc.) non possa provocare l'esaurimento degli indirizzi. Nonostante ciò, oggi i fornitori di servizi Internet sono riluttanti ad installare IPv6 e continuano ad assegnare con grande cautela indirizzi IPv4 ai propri clienti, una strategia che potrà creare gravi problemi nel futuro.

Un altro problema riguarda la facilità di configurazione dei dispositivi IP. Mentre oggi molti utenti di Internet non hanno problemi ad impostare, nel proprio PC, l'indirizzo IP, la maschera di sottorete e il gateway di default, è improbabile che l'acquirente medio di una videocamera voglia imparare a configurare cose più complesse del pulsante "registra". La configurazione "plug-and-play" (*connetti e usa*) dei dispositivi IP rimane un obiettivo primario.

Ulteriori letture

Il nostro elenco di letture consigliate per questo capitolo contiene due pubblicazioni che presentano una panoramica, rispettivamente, degli standard JPEG e MPEG: la loro valenza principale consiste nella spiegazione dei vari fattori che hanno dato vita agli standard. Consigliamo anche il lavoro sul *receiver-driven layered multicast* come esempio illuminante di approccio di sistema alla progettazione, che prende in esame i problemi del multicast, del controllo di congestione e della codifica video.

- Wallace, G.K. "The JPEG still picture compression standard", *Communications of the ACM* 34(1):30-44, April 1991.
- Le Gall, D. "MPEG: A video compression standard for multimedia applications", *Communications of the ACM* 34(1):46-58, April 1991.
- McCanne, S., V. Jacobson e M. Vetterli "Receiver-driven layered multicast", *Proceedings of the SIGCOMM '96 Symposium*, pagg. 117-130, September 1996.

Sfortunatamente non esiste un lavoro unico con una trattazione completa del formato di presentazione. Oltre alle specifiche di XDR, ASN.1 BER e NDR (si vedano Srinivasan [Sri95b], le raccomandazioni CCITT [CCITT92a, CCITT92b] e la Open Software Foundation [OSF94]),

altre tre pubblicazioni parlano di argomenti correlati al formato di presentazione, discutendo argomenti correlati alle prestazioni: O'Malley *et al.* [OPM94], Lin [Lin93] e Chen *et al.* [CLNZ89].

Sull'argomento della compressione, un buon punto di inizio è la codifica Huffman, originariamente definita in [Huf52]. L'algoritmo LZ originale è stato presentato in Ziv e Lempel [ZL77], mentre in [ZL78] si può trovare una versione dell'algoritmo migliorato dagli stessi autori: si tratta di due lavori di natura squisitamente teorica. La pubblicazione che più ha contribuito alla diffusione concreta dell'approccio LZ è di Welch [Wel84]. Per una panoramica più completa sul tema della compressione, raccomandiamo l'articolo di Nelson [Nel92], ma potete anche leggere parecchi recenti libri sull'argomento multimedialità, tra i quali suggeriamo Written *et al.* [WBM99], che presenta un rapporto estremamente elevato fra informazione scientifica e informazione accattivante, e Buford [Buf94], che è una raccolta di interventi che coprono l'intero mondo della multimedialità. Per una descrizione esaustiva dello standard MPEG, si veda Mitchell *et al.* [MPFL96], mentre per una descrizione di MP3 si veda Noll [Nol97].

Infine, raccomandiamo il seguente riferimento attivo:

- <http://bmrc.berkeley.edu/projects/mpeg/index.html>: una raccolta di programmi relativi a MPEG, alcuni dei quali sono utilizzati negli esercizi seguenti

Esercizi

1. Considerate il codice C seguente:

```
#define MAXSTR 100

struct date {
    char month[MAXSTR];
    int day;
    int year;
};

struct employee {
    char name[MAXSTR];
    int ssn;
    struct date *hireday;
    int salary_history[5];
    int num.raises;
};

static struct date date0 = {"MAY", 5, 1996};
static struct date date1 = {"JANUARY", 7, 2002};

static struct employee employee0 = {"RICHARD", 4376, &date0,
                                    {14000, 35000, 47000, 0, 0}, 2};
```

```
static struct employee employee1 = {"MARY", 4377, &date1,
                                    {90000, 150000, 0, 0, 0}, 1};
```

dove num_raises + 1 corrisponde al numero di dati validi nell'array salary_history. Mostrate la rappresentazione di rete che viene generata da XDR per employee0.

- ✓ 2. Mostrate la rappresentazione di rete che viene generata da XDR per employee1 nelle condizioni dell'esercizio precedente.
3. Scrivete la funzione XDR che codifica e decodifica le strutture dati dell'Esercizio 1. Se avete a disposizione un'implementazione di XDR, eseguite la funzione e misurate il tempo necessario per codificare e decodificare un esemplare della struttura employee.
4. Usando funzioni di libreria come htonl e la funzione bcopy di Unix o la funzione CopyMemory di Windows, realizzate una funzione che generi la stessa rappresentazione di rete per le strutture dell'Esercizio 1 che viene generata da XDR. Se possibile, confrontate le prestazioni delle vostra codifica e decodifica "manuale" con quelle delle corrispondenti funzioni XDR.
5. Usate XDR e htonl per codificare un array di numeri interi con 1000 elementi. Misurate e confrontate le prestazioni di entrambe le soluzioni. Come si raffrontano ad un semplice ciclo che legge e scrive un array di numeri interi con 1000 elementi? Eseguite l'esperimento sia con un calcolatore che abbia l'ordinamento dei byte uguale all'ordinamento dei byte previsto per la rete, sia con un altro calcolatore che usi l'ordinamento inverso.
6. Scrivete una vostra implementazione di htonl. Usando la vostra realizzazione di htonl e la versione della libreria standard (se è disponibile hardware di tipo little-endian), eseguite esperimenti atti a determinare quanto tempo sia necessario per scambiare i byte di un numero intero rispetto a copiarli semplicemente.
7. Indicate la codifica ASN.1 per i tre numeri interi seguenti. Notate che, in ASN.1, come in XDR, i numeri interi sono lunghi 32 bit.
 - a) 101
 - b) 10120
 - c) 16909060
- ✓ 8. Indicate la codifica ASN.1 per i tre numeri interi seguenti. Notate che, in ASN.1, come in XDR, i numeri interi sono lunghi 32 bit.
 - a) 15
 - b) 29496729
 - c) 58993458
9. Indicate la rappresentazione big-endian e little-endian per i numeri interi dell'Esercizio 7.
- ✓ 10. Indicate la rappresentazione big-endian e little-endian per i numeri interi dell'Esercizio 8.
11. XDR viene usato per codificare/decodificare l'intestazione del protocollo SunRPC illustrato in Figura 5.20 e la versione di XDR viene determinata mediante il campo **RPCVersion**. Quali potenziali difficoltà sono insite in questo approccio? Sarebbe possibile, per una nuova versione di XDR, usare il formato little-endian per i numeri interi?
12. Il processo di gestione del formato di presentazione viene a volte considerato come un autonomo strato di protocollo, separato dall'applicazione. In questo caso, perché sarebbe una cattiva idea inserire in tale strato di presentazione la compressione dei dati?
13. Supponete di avere un calcolatore con parole di 36 bit: Le stringhe sono rappresentate con cinque caratteri di 7 bit, compattati in ciascuna parola. Quali problemi di presentazione si dovrebbero risolvere perché questo calcolatore possa scambiare numeri interi e stringhe con il resto del mondo?

14. Usando il linguaggio di programmazione che preferite, tra quelli che forniscono supporto alla conversione automatica tra tipi definiti dall'utente, definite un tipo `netint` e dotatelo di conversioni che consentano l'assegnazione e il confronto per uguaglianza tra variabili di tipo `int` e di tipo `netint`. Si può, con una generalizzazione di questo approccio, risolvere il problema del marshalling?
15. Architetture diverse hanno convenzioni diverse sull'ordinamento tra i bit, oltre che sull'ordinamento tra i byte: ad esempio, in merito al fatto che il bit meno significativo di un byte sia il bit 0 o il bit 7. Il lavoro [Pos81] definisce (nella sua Appendice B) l'ordinamento per i bit che sia lo standard per la rete. Perché, allora, l'ordinamento tra i bit non è importante per il formato di presentazione dei dati?
- ★ 16. Sia $p \leq 1$ la frazione di macchine in una rete che hanno architettura big-endian, mentre la frazione rimanente, $1 - p$, sono little-endian. Supponete di scegliere due macchine a caso e di inviare un numero intero dall'una all'altra. Indicate il numero medio di conversioni necessarie nel caso di ordinamento di byte big-endian per la rete, con l'approssimazione *fa tutto il ricevitore*, nei casi $p = 0.1$, $p = 0.5$ e $p = 0.9$. Suggerimento: la probabilità che entrambe le entità coinvolte siano big-endian è p^2 ; la probabilità che le due entità usino un diverso ordinamento di byte è $2p(1 - p)$.
17. Fate esperimenti con un programma di compressione (ad esempio, `compress`, `gzip` o `pkzip`). Quali fattori di compressione riuscite ad ottenere? Osservate se è possibile generare file di dati per i quali si possano ottenere fattori di compressione 5 a 1 oppure 10 a 1.
- ★ 18. Supponete che un file contenga le lettere *a*, *b*, *c* e *d*. Nominalmente, quindi, servono 2 bit per ogni lettera memorizzata nel file.
 - a) Ipotizzate che la lettera *a* ricorra il 50% delle volte, *b* il 30% e *c* e *d* il 10% ciascuna. Indicate una codifica di ogni lettera sotto forma di stringa di bit che dia luogo ad una compressione ottimale. Suggerimento: Usate un solo bit per la lettera *a*.
 - b) Quale percentuale di compressione avete ottenuto nel punto precedente? (Si tratta della media delle percentuali di compressione ottenute per ciascuna lettera, pesate mediante la frequenza della lettera)
 - c) Rispondete di nuovo ai due punti precedenti, ipotizzando che *a* e *b* ricorrono il 40% delle volte ciascuna, *c* il 15% e *d* il 5%.
- ★ 19. Supponete di avere a disposizione una funzione di compressione, *c*, che trasforma una stringa di bit, *s*, in una stringa compressa, *c(s)*.
 - a) Dimostrate che per qualsiasi numero intero *N* deve esistere una stringa *s* di lunghezza *N* per cui la lunghezza di *c(s)* sia maggiore o uguale a *N*, cioè tale da non effettuare, in realtà, alcuna compressione.
 - b) Comprimete alcuni file già compressi (provate a comprimerli più volte, in sequenza, con il medesimo programma di compressione). Cosa accade alla dimensione del file?
 - c) Data una funzione di compressione *c* come al punto (a), individuate una funzione *c'* tale che per qualsiasi stringa di bit *s* la lunghezza di *c'(s)* sia minore o uguale a uno più il minimo fra la lunghezza di *c(s)* e la lunghezza di *s*, cioè, nel caso peggiore, la compressione con la funzione *c'* aumenta la dimensione di una sola unità.
20. Individuate un algoritmo per la codifica RLE che richieda un solo byte per rappresentare simboli non ripetuti.
21. Scrivete un programma per costruire un dizionario di tutte le "parole" (definite come sequenze di caratteri consecutivi privi di spazi) presenti in un file di testo. Potremo poi comprimerne il file (trascurando la perdita di informazione relativa alle spaziature)

rappresentando ogni parola mediante un indice nel dizionario. Scaricate da Internet il file `rfc791.txt` contenente [Pos81] ed eseguite il vostro programma su di esso. Indicate la dimensione del file compresso, nell'ipotesi che ogni parola venga codificata con 12 bit (un valore che dovrebbe essere sufficiente) e, successivamente, nell'ipotesi che le 128 parole più comuni vengano codificate con 8 bit, mentre le altre con 13 bit. Fate l'ipotesi che il dizionario stesso venga memorizzato usando, per ogni parola, un numero di byte uguale a uno più la lunghezza della parola.

- ★ 22. La trasformata coseno-discreta unidimensionale è simile alla trasformata bidimensionale, tranne per il fatto che viene eliminata la seconda variabile (*j* o *y*) e il secondo fattore coseno. Inoltre, viene eliminato, dalla sola trasformazione DCT inversa, il coefficiente iniziale $1/(2N)^{1/2}$. Implementate questa trasformazione e la sua inversa per $N = 8$ (si può fare con un foglio elettronico, anche se sarebbe meglio usare un linguaggio di programmazione con cui poter usare le matrici) e rispondete alle seguenti domande:
 - a) Se il dato in ingresso è $\langle 1, 2, 3, 5, 5, 3, 2, 1 \rangle$, quali coefficienti di DCT sono prossimi a zero?
 - b) Se il dato è $\langle 1, 2, 3, 4, 5, 6, 7, 8 \rangle$, quanti coefficienti di DCT dobbiamo preservare per fare in modo che, dopo la trasformazione DCT inversa, nessun valore differisca dal valore originale per più di 1%? E per 10%? Ipotizzate che i coefficienti DCT eliminati vengano sostituiti da zeri.
 - c) Sia s_i , con $1 \leq i \leq 8$, la sequenza di ingresso costituita da un valore 1 in posizione *i* e valori 0 in tutte le altre posizioni. Supponete di applicare la trasformazione DCT a s_i , azzerare gli ultimi tre coefficienti e, quindi, applicare la trasformazione DCT inversa. Quale valore di *i*, con $1 \leq i \leq 8$, produce l'errore minore nella posizione *i*-esima del risultato? E l'errore maggiore?
 23. Confrontate la dimensione di un'immagine completamente bianca nel formato JPEG con una "tipica" immagine fotografica della stessa dimensione. In quale fase (o fasi) del processo di compressione JPEG l'immagine bianca assume dimensioni inferiori a quella dell'immagine fotografica?
- Nei prossimi tre esercizi possono essere utili i programmi `cjpeg` e `djpeg`, che si possono trovare all'indirizzo `ftp://uu.net/graphics/jpeg`; in alternativa si possono usare altri programmi di conversione JPEG. Per creare ed ispezionare manualmente file grafici, si raccomanda il formato portatile a toni di grigio, `pgm`, dopo aver consultato la documentazione in formato `man` di Unix per `pgm(5)/ppm(5)`.
24. Create un'immagine in toni di grigio consistente in una griglia 8×8 con una linea nera verticale nella prima colonna. Comprimetela nel formato JPEG e decomprimetela. Quanto sono diversi i byte risultanti con l'impostazione di qualità di default? Come descrivereste, visivamente, le imprecisioni introdotte? Quale valore di qualità è sufficiente impostare per ripristinare con precisione il file?
 25. Create un'immagine in toni di grigio 8×8 composta da una stringa di testo ASCII di 64 caratteri. Usate soltanto lettere minuscole, senza spazi o caratteri di punteggiatura. Comprimetela nel formato JPEG e decomprimetela. Quanto è riconoscibile, come testo, l'immagine risultante? Perché l'aggiunta di caratteri di spaziatura potrebbe peggiorare le cose? Con la qualità impostata al valore 100, si otterebbe un modo sensato di comprimere testi?

26. Scrivete un programma che implementi la DCT diretta e inversa, usando l'aritmetica in virgola mobile. Eseguite il programma su un'immagine campionata in toni di grigio. Dato che DCT è priva di perdite, l'immagine in uscita dal programma dovrebbe corrispondere all'immagine in ingresso. Modificate ora il programma in modo che azzeri alcune delle componenti di frequenza più elevata e osservate come diventa l'immagine risultante. Quanto è diverso il risultato rispetto a quanto avviene con JPEG?
27. Esprimete DCT(0, 0) in termini della media dei valori di $pixel(x, y)$.
28. Ragionate sulle funzioni che ci si può aspettare da uno standard per filmati: avanzamento veloce, possibilità di apportare modifiche (*editing*), accesso casuale, e così via (per ispirarvi, leggete il lavoro di Le Gall, "MPEG: A video compression standard for multimedia applications", citato nella sezione "Ulteriori Letture" di questo capitolo). Illustrate il progetto di MPEG in termini di queste caratteristiche.
29. Supponete di voler implementare l'avanzamento veloce (e la funzione inversa) per flussi MPEG. A quali problemi andate incontro se limitate il vostro meccanismo alla visualizzazione dei soli frame I ? Se non lo fate, allora qual è il numero massimo di frame della sequenza originale che dovete decodificare per visualizzare un certo frame nella sequenza di avanzamento veloce?
30. Usate il programma `mpeg_play` per riprodurre un filmato codificato con MPEG. Fate esperimenti con le varie opzioni, in particolare con `-nob` e `-nop`, che vengono usate per omettere dal flusso, rispettivamente, i frame B e i frame P . Quali sono gli effetti visibili di tali omissioni?
31. Il programma `mpeg_stat` viene usato per visualizzare statistiche per flussi video. Usatelo per determinare, per alcuni flussi:
 - il numero dei frame I , B e P e la loro sequenza
 - il rapporto medio di compressione per l'intero filmato
 - il rapporto medio di compressione per ciascun tipo di frame
32. Supponete di avere un filmato con due punti bianchi che si muovono uno verso l'altro a velocità costante su uno sfondo nero. Codificate lo con MPEG. In un frame I i due punti sono distanti 100 pixel, mentre nel successivo frame I sono sovrapposti. Il punto di sovrapposizione finale si trova al centro di un macroblocco 16×16 .
 - Descrivete come si potrebbe codificare in modo ottimo la componente Y dei frame B (o P) intermedi.
 - Supponete ora che i punti siano colorati e che il colore cambi lentamente mentre i punti si spostano. Descrivete quale potrebbe essere la codifica dei valori U e V.

Sicurezza delle reti

Problema Rendere sicuri i dati

Le reti di calcolatori sono tipicamente una risorsa condivisa usata da molte applicazioni per molti scopi diversi. A volte i dati trasmessi fra due processi applicativi sono confidenziali e le applicazioni preferirebbero che altri non fossero in grado di leggerle. Ad esempio, quando si acquista un prodotto sul World Wide Web, a volte gli utenti trasmettono i loro numeri di carta di credito attraverso la rete, un'operazione pericolosa perché è facile per alcuni porsi in ascolto nella rete e leggere i pacchetti che vi transitano. Quindi, a volte gli utenti desiderano *cifrare* i messaggi che inviano, con l'obiettivo di impedire che qualcuno, ponendosi in ascolto lungo il canale di trasmissione, sia in grado di leggere i contenuti del messaggio.

L'idea della codifica cifrata è abbastanza semplice: il mittente applica una funzione *cifrante* al messaggio *di testo* originario (*plaintext*), il messaggio *di testo* cifrato risultante (*ciphertext*) viene inviato attraverso la rete e il destinatario applica una funzione inversa (denominata funzione *decifrante*) per ricostruire il messaggio di testo originario. Il procedimento di cifratura/decifratura è, in generale, dipendente da una *chiave* (*key*) segreta, condivisa dal mittente e dal destinatario. Qualora venga usata un'adatta combinazione di chiave e di algoritmo cifrante, il compito di chi si ponga in ascolto per decifrare il testo cifrato diviene sufficientemente arduo e il mittente e il destinatario possono essere certi che la loro comunicazione sia sicura.

Questo comune utilizzo della crittografia è progettato per assicurare la *privacy* (*privacy*), cioè per prevenire il rilascio di informazioni non autorizzato. La privacy, però, non è l'unico servizio fornito dalla crittografia, che può anche essere usata per realizzare altri servizi, ugualmente significativi, tra i quali l'*autenticazione* (cioè la verifica dell'identità delle entità remote coinvolte in una comunicazione) e l'*integrità* (che garantisce che un messaggio non sia stato alterato). Questo capitolo inizia presentando i concetti di base della crittografia, tra i quali una descrizione dei tre principali algoritmi crittografici: DES (Data Encryption Standard), RSA (l'algoritmo di Rivest, Shamir e Adleman) e MD5 (Message Digest 5). In seguito illustriamo come questi algoritmi vengano usati per fornire i servizi di autenticazione e di integrità, per poi passare a

discutere il problema di come facciano gli utenti ad ottenere, la prima volta, le chiavi di cui hanno bisogno (il problema della *distribuzione delle chiavi*). Il capitolo si conclude descrivendo un insieme di sistemi di sicurezza e dei relativi protocolli che sono stati progettati per la rete Internet e ivi installati.

Una cosa da ricordare durante la lettura di questo capitolo è che i diversi algoritmi e protocolli per la privacy, l'autenticazione e l'integrità vengono descritti soltanto in una modalità autonoma. In pratica, la progettazione di un sistema sicuro richiede una complessa combinazione del corretto insieme di protocolli e algoritmi: un compito difficile, perché ogni protocollo è vulnerabile ad una serie di attacchi diversi. Per peggiorare le cose, determinare se un protocollo sia "buono a sufficienza" è più un'arte e una questione politica che una scienza. Un'analisi approfondita di questi diversi attacchi, e come poter costruire un sistema completo che minimizzi il rischio di danni, va al di là degli scopi di questo libro.

8.1 Algoritmi crittografici

Esistono tre grandi categorie di algoritmi crittografici: algoritmi *a chiave segreta*, algoritmi *a chiave pubblica* e algoritmi *hashing*. Gli algoritmi a chiave segreta sono simmetrici, nel senso che entrambi i partecipanti¹ alla comunicazione condividono un'unica chiave. La Figura 8.1 illustra l'utilizzo della cifratura a chiave segreta per trasmettere dati lungo un canale altrimenti insicuro. DES (Data Encryption Standard) è l'esempio più noto di funzione cifrante a chiave segreta, mentre IDEA (International Data Encryption Algorithm) è un altro esempio.

Diversamente dal caso in cui una coppia di entità condivide una chiave segreta, la crittografia *a chiave pubblica* richiede che ogni entità partecipante abbia una chiave *privata* che non è condivisa con nessuno e una chiave *pubblica* che viene resa pubblica in modo che tutti la conoscano. Per inviare un messaggio sicuro ad un'entità, si cifra il messaggio usando la sua ben nota chiave pubblica, messaggio che verrà decifrato dal destinatario mediante la sua chiave privata, secondo lo scenario rappresentato in Figura 8.2. Il più noto algoritmo di cifratura a chiave pubblica è RSA, che prende il nome dai suoi ideatori, Rivest, Shamir e Adleman.

Il terzo tipo di algoritmo di crittografia è una funzione di *hash* o di *message digest* ("riassunto di messaggio"). Diversamente dai due precedenti tipi di algoritmi, le funzioni crittografiche



Figura 8.1 Cifratura a chiave segreta.

¹ Usiamo, per le parti coinvolte in una comunicazione sicura, il termine *partecipanti*, perché questo è il termine che abbiamo usato in tutto il libro per identificare le due entità terminali di un canale. Nel mondo della sicurezza, queste due entità vengono solitamente chiamate *principal*.

8.1 Algoritmi crittografici



Figura 8.2 Cifratura a chiave pubblica.

di hash non richiedono tipicamente l'uso di chiavi²: l'idea consiste nel creare una corrispondenza tra un messaggio potenzialmente di grandi dimensioni e un numero piccolo di dimensioni fisse, in analogia con il modo in cui le normali funzioni di hash identificano una corrispondenza tra valori appartenenti ad un insieme di grandi dimensioni e valori appartenenti ad un insieme di dimensioni inferiori.

Il modo migliore di immaginare una funzione crittografica di hash è pensare che calcoli una *somma di controllo crittografico* (*cryptographic checksum*) per un messaggio, che protegge il destinatario da subdole modifiche del messaggio, esattamente come una normale somma di controllo protegge il destinatario da modifiche accidentali del messaggio stesso. Ciò avviene perché tutti gli algoritmi di hash crittografico sono scelti in modo molto attento per essere funzioni a senso unico: data la somma di controllo di un messaggio, è virtualmente impossibile ricostruire quale messaggio abbia prodotto quella particolare somma. Detto in altro modo, il costo di elaborazione necessario per trovare due messaggi che generino, mediante la funzione di hash, la medesima somma di controllo è troppo elevato per essere plausibile. L'importanza di questa proprietà consiste nel fatto che se conoscete la somma di controllo per un messaggio (insieme al messaggio stesso) e siete in grado di calcolare in modo esatto la medesima somma di controllo per quel messaggio, allora è altamente probabile che sia stato proprio quel messaggio a generare la somma di controllo che state esaminando.

L'algoritmo per il calcolo di somma di controllo crittografica più ampiamente diffuso è Message Digest, nella sua versione 5 (MD5). Una proprietà importante di MD5, oltre alle proprietà delineate nel paragrafo precedente, è la sua migliore efficienza computazionale rispetto a DES o RSA. Più avanti, in questa stessa sezione, vedremo l'importanza di questo fatto.

Vogliamo porre nuovamente l'accento sul fatto che gli algoritmi crittografici, come DES, RSA e MD5, sono solamente gli elementi costitutivi per un sistema sicuro. La Figura 8.3 presenta una semplice tassonomia che illustra questo punto. Esaminando questi servizi e questi blocchi elementari dovremmo rispondere a questa domanda: come hanno fatto i partecipanti ad ottenere, la prima volta, le varie chiavi necessarie? Si tratta del problema della *distribuzione delle chiavi*, uno dei problemi centrali quando si parla di sicurezza, come vedremo nelle sezioni successive.

Prima di mostrare come si usino gli algoritmi crittografici per costruire sistemi sicuri, descriviamo come funzionano i tre algoritmi più noti: DES, RSA e MD5. Daremo anche qualche dettaglio per spiegare *perché* funzionano, ma non possiamo fare molto su questo

² Anche se gli algoritmi di *hashing* non richiedono come argomento una chiave (cioè non sono parametrici in funzione di una chiave), il messaggio che calcolano contiene spesso una chiave, generando termini confusi come "MD5 con chiave" (*keyed MD5*).

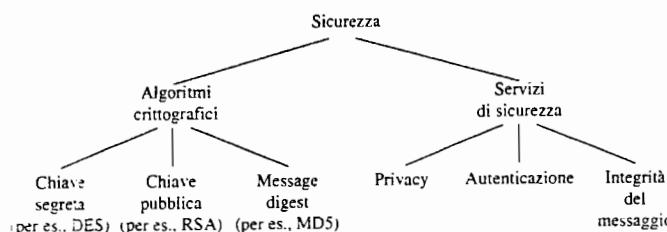


Figura 8.3 Tassonomia della sicurezza delle reti.

fronte perché i principi di progetto su cui si basa DES non sono di dominio pubblico. Nel caso di RSA, una spiegazione accurata del meccanismo di funzionamento richiederebbe delle basi di conoscenza di teoria dei numeri che vanno al di là degli scopi di questo libro, ma potremo dare qualche spunto per la comprensione dei meccanismi basilari. Tuttavia, prima di esaminare i dettagli degli algoritmi, facciamo un passo indietro e chiediamoci cosa vogliamo da un algoritmo crittografico.

8.1.1 Requisiti

Il requisito di base per un algoritmo di crittografia è che sia in grado di trasformare testo leggibile in testo cifrato in modo che soltanto il destinatario previsto (che possiede la chiave di decifrazione) possa ricostruire il testo leggibile. Ciò significa che il metodo di cifratura dovrebbe resistere agli attacchi di chi non possiede la chiave. Per iniziare, ipotizzeremo che l'algoritmo di cifratura sia, per se stesso, noto a tutti e che venga tenuta segreta la sola chiave: un'ipotesi motivata dal fatto che, se la sicurezza dipendesse dal fatto che venga tenuto segreto l'algoritmo, sarebbe necessario sostituirlo ogni volta che si avesse la sensazione che non sia più segreto, il che significherebbe modifiche potenzialmente frequenti dell'algoritmo, una cosa problematica perché sviluppare un nuovo algoritmo richiede molto lavoro. Ancora, uno dei modi migliori per scoprire se un algoritmo è efficace è utilizzarlo per lungo tempo: se nessuno lo viola, probabilmente è sicuro (fortunatamente, ci sono un sacco di persone che tenteranno di violare l'algoritmo e che renderanno ampiamente pubblico il fatto di esserci riusciti, per cui, in generale, non avere notizie è una buona notizia). Di conseguenza, nell'installazione di un nuovo algoritmo esiste un rischio considerevole. Per concludere, il nostro primo requisito è che l'unica cosa necessaria per garantire la sicurezza della privacy dei dati sia la segretezza della chiave, e non l'algoritmo di cifratura.

È importante capire che quando qualcuno intercetta un brano di testo cifrato potrebbe avere a propria disposizione maggiori informazioni di quante non siano contenute nel testo cifrato stesso. Per esempio, può sapere che il testo originario era scritto in inglese, per cui la lettera e ricorre più spesso di tutte le altre nel testo originario, così come è facile prevedere la frequenza di molte altre lettere e di combinazioni comuni di lettere. Questa informazione può semplificare significativamente il compito di scoprire la chiave. Analogamente, l'intercettatore potrebbe avere informazioni relative al probabile contenuto del messaggio: ad esempio, è probabile che all'inizio di una sessione di collegamento ad un sistema remoto (*remote login*) compaia la parola "login". Ciò può consentire un attacco di tipo "known plaintext" (testo originario conosciuto), che ha una probabilità di successo molto più elevata di un

attacco di tipo "ciphertext only" (testo cifrato soltanto). Ancora migliore è l'attacco di tipo "chosen plaintext" (testo originario prescelto), che può essere consentito nel caso in cui l'attaccante possa inviare al mittente informazioni che si ritengono probabili e plausibili per il mittente stesso, come accade spesso in situazioni di guerra, ad esempio.

I migliori algoritmi di crittografia, quindi, sono in grado di impedire che l'attaccante deduca la chiave anche quando è in possesso sia del testo originario sia del testo cifrato. Un approccio (quello di DES) consiste nel rendere l'algoritmo talmente complesso che nel testo cifrato non rimanga, teoricamente, nessuna informazione relativa alla struttura del testo originario. Questo non lascia all'attaccante altra scelta che quella di una ricerca esaustiva nello spazio delle chiavi possibili, attività che può essere resa impraticabile scegliendo uno spazio delle chiavi sufficientemente ampio e rendendo ragionevolmente costosa (dal punto di vista delle risorse di calcolo) l'operazione di verifica di una chiave. Come vedrete, in base a queste proprietà DES è oggi divenuto solo marginalmente sicuro. Altri algoritmi crittografici sfruttano la potenza della matematica: ad esempio, RSA può essere violato solamente se l'attaccante è in grado di trovare i fattori di un numero che sia il prodotto di due numeri primi molto grandi, un problema che è noto (o almeno è ampiamente ritenuto) essere molto costoso.

I requisiti per un algoritmo di tipo *message digest* sono un po' diversi. Questi algoritmi devono essere funzioni *a senso unico*, nel senso che, noto il risultato prodotto dalla funzione, sia impraticabile, dal punto di vista delle risorse di calcolo, determinare un dato di ingresso che lo abbia generato. Poiché questi algoritmi producono generalmente un "riassunto" (*digest*) più breve del messaggio originario, esisteranno certamente più messaggi diversi che producono lo stesso riassunto, ma l'identificazione di due di tali messaggi deve essere impraticabile. Sono richieste queste proprietà per garantire che, dato un messaggio m e il suo riassunto $MD(m)$, non sia possibile trovare un nuovo messaggio m' $\neq m$ che generi il medesimo riassunto: in questo modo sarà impossibile modificare il messaggio m e fare in modo che la funzione che genera il riassunto produca lo stesso risultato per il messaggio modificato.

Perché una funzione generatrice di riassunti soddisfi questi requisiti, i suoi risultati devono essere distribuiti in modo uniformemente casuale; ad esempio, se un riassunto è lungo 128 bit, ci devono essere 2^{128} risultati possibili. Ciò significa che, se i risultati prodotti dalla funzione sono distribuiti uniformemente, prima di trovare due messaggi che generino lo stesso riassunto sarà necessario calcolare, mediamente, il riassunto di 2^{64} messaggi diversi (un fenomeno simile al "problema del compleanno", si vedano gli esercizi per maggiori dettagli). Se, invece, i risultati prodotti dalla funzione non sono distribuiti casualmente in modo uniforme (cioè alcuni risultati sono più probabili di altri), allora può accadere che l'identificazione di due messaggi che producono lo stesso riassunto sia molto più semplice, cosa che violerebbe la sicurezza dell'algoritmo.

L'altro requisito per i *message digest* è che la loro generazione abbia un'efficienza computazionale ragionevole. Se l'invocazione della funzione che genera il riassunto del messaggio riduce di un ordine di grandezza il throughput di un'applicazione, è improbabile che molti utenti ritengano che i benefici dell'integrità del messaggio e dell'autenticazione del mittente che essa fornisce valgano il suo costo.

8.1.2 Cifratura a chiave segreta (DES)

L'algoritmo DES cifra un blocco di 64 bit di testo usando una chiave a 64 bit. La chiave, in realtà, contiene soltanto 56 bit utilizzabili: l'ultimo bit di ognuno degli 8 byte della chiave è

un bit di parità per quel byte. I messaggi più lunghi di 64 bit si possono cifrare con DES nel modo descritto in seguito.

L'applicazione di DES prevede tre fasi distinte:

1. I 64 bit del messaggio vengono permutati (mischiati).
2. Ai dati risultanti e alla chiave si applica sedici volte una stessa operazione.
3. Al risultato viene applicata la permutazione inversa di quella del passo 1.

La Figura 8.4 illustra questa panoramica ad alto livello di DES.

La Tabella 8.1 rappresenta una parte della permutazione iniziale; la permutazione finale è quella inversa (cioè, ad esempio, il bit 40 andrebbe a finire nella posizione 1). Si ritiene che queste due permutazioni non aggiungano nulla alla sicurezza di DES: alcuni esperti di sicurezza ritengono che siano state previste per allungare il tempo di calcolo, ma ciò è altrettanto probabile quanto il fatto che siano una conseguenza dell'iniziale implementazione hardware, che, ad esempio, richiedeva alcune restrizioni sulla disposizione dei pin.

Durante ciascuna fase, il blocco a 64 bit viene scomposto in due metà a 32 bit e si scelgono 48 bit dalla chiave a 56 bit. Se indichiamo le metà sinistra e destra del blocco nella fase i

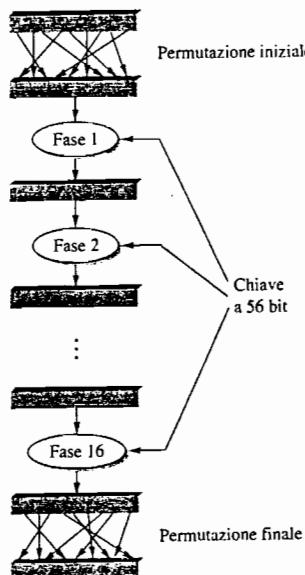


Figura 8.4 Panoramica ad alto livello di DES.

Tabella 8.1 Permutazione iniziale (e finale) di DES.

Posizione iniziale	1	2	3	4	5	...	60	61	62	63	64
Posizione finale	40	8	48	16	56	...	-9	49	17	57	25

8.1 Algoritmi crittografici

511

come, rispettivamente, L_i e R_i , e la chiave a 48 bit nella fase i come K_i , allora queste tre parti vengono combinate durante la fase i secondo la regola seguente:

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

dove F è una funzione combinatoria descritta in seguito e \oplus è l'operatore di OR esclusivo (XOR). La Figura 8.5 illustra l'operazione fondamentale che viene svolta ad ogni fase. Notate che L_0 e R_0 corrispondono alla metà sinistra e destra del blocco di 64 bit che viene prodotto dalla permutazione iniziale e che L_{16} e R_{16} vengono ricombinate insieme per formare il blocco di 64 bit a cui viene, poi, applicata la permutazione finale.

Dobbiamo ora definire la funzione F e mostrare come ogni chiave K_i venga estratta dalla chiave di 56 bit. Partiamo dalla chiave: per prima cosa i 56 bit vengono permutati come indicato in Tabella 8.2. Notate che i bit in ottava posizione (e posizioni multiple) vengono ignorati (ad esempio, il bit 64 non è presente nella tabella), riducendo la dimensione della chiave da 64 a 56 bit. Successivamente, in ogni fase, i 56 bit vengono divisi in due metà di 28 bit e ciascuna metà viene ruotata verso sinistra di una posizione o di due posizioni, dipendentemente dalla fase in cui ci si trova: l'estensione della rotazione, in numero di bit, è indicata in Tabella 8.3 per ciascuna fase. I 56 bit che sono prodotti da questa rotazione a scorrimento vengono usati sia come dato di ingresso per la fase successiva sia per selezionare i 48 bit che compongono la chiave della fase successiva. La Tabella 8.4 mostra come vengono selezionati i 48 bit fra i 56 della chiave: notate che vengono contemporaneamente selezionati e permutati. Ad esempio, il bit in posizione 9 non viene selezionato, perché non è presente nella tabella.

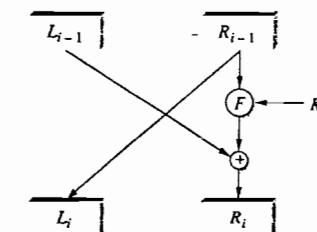


Figura 8.5 Elaborazione svolta da ogni fase di DES.

Tabella 8.2 Permutazione della chiave di DES.

Posizione iniziale	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Posizione finale	8	16	24	56	52	...	59	60	61	62	63	17	25	45	37	29

Tabella 8.3 Rotazione della chiave in DES, fase per fase.

Fase	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Rotazione	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Tabella 8.4 Permutazione per la compressione DES.

Posizione iniziale	1	2	3	4	5	6	7	8	10	11	12	13	14	15	16	17
Posizione finale	5	24	7	16	6	10	20	18	12	3	15	23	1	9	19	2
Posizione iniziale	19	20	21	23	24	26	27	28	29	30	31	32	33	34	36	37
Posizione finale	14	22	11	13	4	17	21	8	47	31	27	48	35	41	46	28
Posizione iniziale	39	40	41	42	44	45	46	47	48	49	50	51	52	53	55	56
Posizione finale	39	32	25	44	37	34	43	29	36	38	45	33	26	42	30	40

La funzione F combina la chiave a 48 bit risultante per la fase i (K_i) con la metà di destra del blocco di dati risultante dalla fase $i-1$ (R_{i-1}), nel modo seguente. Per semplificare la notazione, indicheremo rispettivamente K_i e R_{i-1} con K e R . Per prima cosa la funzione F espande R da 32 a 48 bit, in modo da poterla combinare con i 48 bit di K : per farlo, suddivide R in otto porzioni di 4 bit ed espande ciascuna porzione portandola a 6 bit, copiando il bit più a destra e il bit più a sinistra delle porzioni di 4 bit adiacenti a sinistra e a destra, rispettivamente. Questa espansione è illustrata in Figura 8.6, dove R viene considerata circolare, nel senso che la prima e l'ultima porzione sono ritenute contigue ai fini dell'algoritmo.

Successivamente, i 48 bit di K vengono suddivisi in otto porzioni di 6 bit e ad ogni porzione viene applicata l'operazione XOR con la porzione corrispondente generata dalla precedente espansione di R . Infine, ogni valore di 6 bit risultante viene elaborato da una *funzione di sostituzione* (*substitution box*, S box), che riduce ogni porzione di 6 bit in una

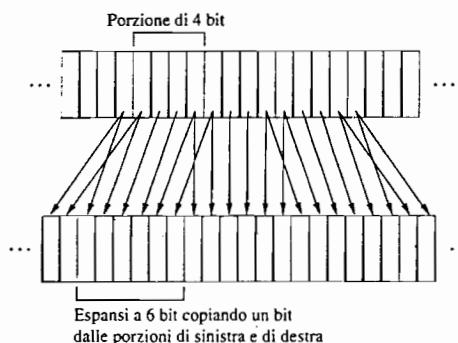


Figura 8.6 Fase di espansione di DES.

8.1 Algoritmi critografici

porzione di 4 bit. In realtà esistono otto diverse S box, una per ognuna delle porzioni di 6 bit. Potete immaginare che una S box realizzzi semplicemente una corrispondenza multi-a-uno da numeri a 6 bit a numeri a 4 bit. La Tabella 8.5 mostra parte della funzione della S box per la prima porzione. E, con questo, abbiamo terminato la fase i .

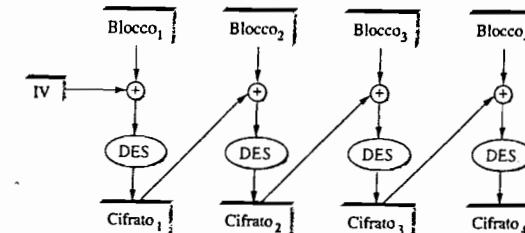
Notate che la descrizione precedente non fa distinzione fra la cifratura e la decifrazione. Una delle caratteristiche interessanti di DES è che le due sezioni dell'algoritmo funzionano allo stesso identico modo. L'unica differenza sta nel fatto che le chiavi vengono applicate in ordine inverso, $K_{16}, K_{15}, \dots, K_1$.

Ricordate anche che la discussione precedente è limitata ad un singolo blocco di dati di 64 bit. Per cifrare un messaggio più lungo usando DES, si usa tipicamente una tecnica che prende il nome di CBC (*cipher block chaining*, concatenazione di blocchi cifrati). L'idea di CBC è semplice: al testo cifrato per il blocco i viene applicata l'operazione XOR con il testo originario del blocco $i+1$, prima di sottoporlo a DES. Al posto del blocco cifrato di indice 0, che non esiste, si usa un *vettore di inizializzazione* (*IV, initialization vector*). Questo vettore IV, che è un numero casuale generato dal mittente, viene trasmesso insieme al messaggio, in modo che si possa ricostruire il primo blocco del testo originario. La Figura 8.7 mostra il funzionamento di CBC per la parte di cifratura di un messaggio di 256 bit (quattro blocchi). La decifrazione funziona come ci si potrebbe aspettare, perché la funzione XOR è l'inversa di se stessa, per cui è sufficiente iniziare il procedimento dall'ultimo blocco e procedere verso la parte iniziale del messaggio.

Terminiamo notando che non esiste prova matematica della sicurezza di DES. Qualunque sia la sicurezza che si ottiene, il mezzo con cui la si ottiene è l'applicazione di due tecniche: confusione e dispersione dell'informazione (avendo appena analizzato l'algoritmo, dovreste essere in grado di apprezzare appieno queste due tecniche). Ciò che si può dire è che l'unico modo conosciuto per violare DES è la ricerca esaustiva nello spazio di tutte le 2^{56}

Tabella 8.5 Esempio di S box di DES (bit da 1 a 6).

Ingresso	000000	000001	000010	000011	000100	000101	...
Uscita	1110	0100	1101	0001	0010	1111	...
Ingresso	...	111010	111011	111100	111101	111110	111111
Uscita	...	0011	1110	1010	0000	0110	1101

Figura 8.7 CBC (*cipher block chaining*) per messaggi di grandi dimensioni.

chiavi possibili, anche se in media ci si può aspettare di dover esplorare solamente metà dello spazio delle chiavi, cioè $2^{55} = 3.6 \times 10^{16}$ chiavi. Con moderne stazioni di lavoro è possibile eseguire la cifratura in $1.5 \mu\text{s}$, per cui si può violare una chiave in $5.0 \times 10^{16} \mu\text{s}$ (circa 1500 anni). Anche se questo tempo può sembrare molto lungo, ricordate che l'esplorazione dello spazio delle chiavi è un compito che si può *molto* facilmente risolvere in parallelo, per cui impiegando 3000 PC si potrebbe violare una chiave in sei mesi.

Questo periodo di tempo viene considerato al limite della sicurezza ragionevole, soprattutto in considerazione del fatto che la velocità dei processori raddoppia ogni 18 mesi. Per questa ragione, oggi molte applicazioni usano 3DES (*triple-DES*), cioè cifrano i dati tre volte, un'operazione che può essere fatta usando tre chiavi diverse, oppure due chiavi soltanto: per prima cosa si usa la prima chiave, poi la seconda, poi di nuovo la prima.

8.1.3 Cifratura a chiave pubblica (RSA)

RSA è un algoritmo molto diverso, non soltanto perché richiede chiavi diverse per la cifratura (chiave pubblica) e la decifrazione (chiave privata), ma anche perché si basa sulla teoria dei numeri: l'aspetto più interessante di RSA sta proprio nel modo in cui vengono scelte queste due chiavi. L'azione di cifratura o decifrazione di un messaggio viene espressa da una funzione semplice, anche se questa funzione richiede una potenza di calcolo enorme. In particolare, RSA usa solitamente una chiave di 1024 bit, rendendo la sua elaborazione molto più onerosa di quella di DES; ne parleremo in seguito.

La prima cosa da fare è la generazione di una chiave pubblica e di una chiave privata. Per farlo, si scelgono due numeri primi molto grandi, p e q , e li si moltiplicano per ottenere n . Sia p sia q devono essere lunghi circa 256 bit. Successivamente, si sceglie la chiave di cifratura e , in modo che e e $(p - 1) \times (q - 1)$ siano primi fra loro (due numeri sono primi fra loro se non hanno in comune fattori diversi da 1). Infine, si calcola la chiave di decifrazione d con la formula seguente

$$d = e^{-1} \pmod{((p - 1) \times (q - 1))}$$

La chiave pubblica è costituita dalla coppia $\langle e, n \rangle$, mentre la chiave privata è data dalla coppia $\langle d, n \rangle$. I numeri primi di partenza p e q non servono più e possono essere eliminati, ma non devono essere rivelati.

Date queste due chiavi, la cifratura è definita dalla formula seguente:

$$c = m^e \pmod{n}$$

mentre la decifrazione è definita da:

$$m = c^d \pmod{n}$$

dove m è il messaggio originario e c è il messaggio cifrato risultante. Notate che m deve essere inferiore ad n , per cui non può essere più lungo di 1024 bit. Un messaggio più lungo viene semplicemente elaborato come concatenazione di più blocchi di 1024 bit.

Per vedere come funziona tutto ciò, considerate il seguente esempio, che usa valori molto piccoli di p e q . Supponete di scegliere $p = 7$ e $q = 11$. Ciò significa che

$$n = 7 \times 11 = 77$$

8.1 Algoritmi crittografici

e

$$(p - 1) \times (q - 1) = 60$$

per cui dobbiamo scegliere un valore di e che sia primo rispetto a 60. Scegliamo $e = 7$, perché 7 e 60 non hanno fattori comuni (oltre, ovviamente, a 1). Ora dobbiamo calcolare d come segue:

$$d = 7^{-1} \pmod{((7 - 1) \times (11 - 1))}$$

che è come dire:

$$7 \times d = 1 \pmod{60}$$

Si ottiene, quindi, $d = 43$, perché

$$7 \times 43 = 301 = 1 \pmod{60}$$

Quindi, abbiamo ora la chiave pubblica $\langle e, n \rangle = \langle 7, 77 \rangle$ e la chiave privata $\langle d, n \rangle = \langle 43, 77 \rangle$. Notate che in questo esempio sarebbe molto semplice determinare i valori di p e q conoscendo n , da cui poi determinare e da d , ma se n fosse il prodotto di due numeri, ognuno dei quali fosse lungo 256 bit, sarebbe impraticabile, in termini di risorse di calcolo, determinare p e q . Dovrebbe anche essere evidente il motivo per cui p e q non devono essere rivelati: noti questi valori, è facile calcolare la chiave privata a partire dalla chiave pubblica.

Considerate ora una semplice operazione di cifratura. Supponete di voler cifrare un messaggio contenente il valore 9. Seguendo l'algoritmo di cifratura sopra delineato:

$$c = m^e \pmod{n} = 9^7 \pmod{77} = 37$$

Quindi, 37 è il messaggio cifrato che invieremmo (potete facilmente verificare questi calcoli con una calcolatrice).

Ricevendo il messaggio, il testo cifrato verrebbe decifrato nel modo seguente:

$$m = c^d \pmod{n} = 37^{43} \pmod{77} = 9$$

Di conseguenza, come richiesto, il messaggio originale viene ricostruito (la verifica di questo calcolo con una calcolatrice è un po' più difficile; occorre eseguire l'elevazione a potenza in più fasi e calcolare il resto della divisione modulo 77 dopo ogni fase per evitare di dover gestire numeri interi troppo elevati per una calcolatrice).

Notate che, quando due entità vogliono cifrare i dati che si stanno scambiando usando un algoritmo a chiave pubblica come RSA, sono necessarie due copie di chiavi, pubblica e privata. Non si può cifrare con la chiave privata e chiedere alla controparte di decifrare con la chiave pubblica, perché tutti hanno accesso alla chiave pubblica e tutti potrebbero, quindi, decifrare il messaggio. In altre parole, l'entità A cifra i dati che invia all'entità B usando la chiave pubblica di B e B usa la propria chiave privata per decifrare tali dati, mentre B cifra i dati che invia ad A usando la chiave pubblica di A, il quale decifra tale messaggio usando la propria chiave privata. Osservate che A non è in grado di decifrare un messaggio che ha inviato a B: soltanto B possiede la chiave privata necessaria.

Violazione di RSA

Nel 1977 venne dichiarata una sfida per la violazione di un messaggio di 129 cifre (430 bit) che era stato cifrato con RSA. Si riteneva che il codice fosse impenetrabile, essendo necessario un calcolo di 40 milioni di miliardi ("quadrilioni") di anni con gli algoritmi allora noti per la scomposizione in fattori di grandi numeri. Nell'aprile del 1994, soltanto 17 anni dopo, quattro scienziati affermarono di aver violato il codice. Il messaggio crittografato era

THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE

Il problema era stato risolto usando un metodo di scomposizione in fattori che richiese circa 5000 anni di calcolo su una macchina "MIP" ("million instructions per second", cioè un calcolatore che esegue un milione di istruzioni al secondo). Si giunse al risultato in otto mesi, suddividendo il problema in problemi più semplici ed inviando tali sottoproblemi, usando la posta elettronica, a diversi calcolatori in tutto il mondo.

Ricordate che non è sempre necessario spendere 5000 anni-MIP per violare una chiave, specialmente se la chiave non viene scelta con attenzione. Ad esempio, venne scoperto un problema di sicurezza in un programma di navigazione Web che usava RSA per cifrare i numeri di carta di credito inviati attraverso Internet. Il problema consisteva nel fatto che il sistema usava un metodo altamente prevedibile (una combinazione dell'identificativo del processo e della data) per generare un numero casuale che, a sua volta, veniva usato per generare una chiave pubblica e la chiave privata corrispondente. Chiavi di questo tipo sono facilmente violabili.

La sicurezza di RSA deriva dalla premessa che la scomposizione in fattori di numeri di grandi dimensioni sia un problema molto dispendioso in termini di risorse di calcolo. In particolare, scomponendo n in fattori si otterebbero p e q , la cui conoscenza compromette la sicurezza della chiave privata, perché si può calcolare d . La velocità con cui si possono scomporre in fattori numeri molto grandi è funzione sia della velocità del processore sia dell'algoritmo di scomposizione utilizzato. Si stima che numeri di 512 bit saranno scomponibili in fattori primi nei prossimi anni, ed è per questo motivo che oggi si stanno usando chiavi di 1024, o anche di 2048, bit. Ricordate che, mentre noi ci stiamo concentrando sulla sicurezza dei dati che viaggiano attraverso la rete (dati che, quindi, sono vulnerabili per un periodo di tempo assai breve), in generale chi si occupa di sicurezza deve tenere in considerazione la vulnerabilità dei dati che vengono conservati in archivi per decenni.

8.1.4 Algoritmi di Message Digest (MD5)

Esistono parecchi algoritmi per la generazione di *message digest* (riassunto di messaggio), assai diffusi e noti con la sigla MD n , con vari valori di n : MD5 è quello più usato nel momento in cui scriviamo. Un'altra funzione di generazione di riassunti di messaggi è l'algoritmo SHA (*secure hash algorithm*). Tutte queste funzioni fanno la stessa cosa: calcolano una somma di controllo crittografica di lunghezza prefissata a partire da un messaggio di ingresso arbitrariamente lungo.

Dal punto di vista matematico, gli algoritmi che calcolano questi riassunti tendono ad avere più elementi in comune con DES piuttosto che con RSA: non hanno, cioè, un fonda-

mento matematico formale, ma si affidano alla complessità dell'algoritmo per generare un'uscita casuale in modo da soddisfare i requisiti delineati in precedenza. Presentiamo qui soltanto una breve descrizione dell'algoritmo MD5, che sembra costituito da un insieme di trasformazioni casuali, per cui non sorprende che generi uscite abbastanza casuali³.

Il funzionamento di base di MD4, MD5 e SHA è raffigurato in Figura 8.8. Questi algoritmi operano su un messaggio di 512 bit per volta, per cui il primo passo consiste nel portare la dimensione del messaggio ad un multiplo di 512 bit, accodando al messaggio un numero di bit compreso fra 1 e 512, il primo dei quali assume il valore 1, i rimanenti assumono il valore 0 e sono seguiti da un numero intero a 64 bit che contiene la lunghezza, in bit, del messaggio originale. Notate che ciò consente di usare messaggi di qualsiasi lunghezza, fino ad un massimo di 264 bit.

Il calcolo del riassunto inizia con il valore del riassunto inizializzato ad una costante; questo valore viene combinato con i primi 512 bit del messaggio, per generare un nuovo valore di riassunto, usando una trasformazione complessa che viene descritta in seguito; il nuovo valore viene combinato con i successivi 512 bit del messaggio, usando la medesima trasformazione, e così via, finché non viene prodotto il valore finale del riassunto.

La componente principale dell'algoritmo MD5 è, quindi, la trasformazione che riceve in ingresso il valore attuale del riassunto, di 128 bit, più 512 bit di messaggio, e genera in uscita un nuovo riassunto di 128 bit. MD5, come altri moderni algoritmi che calcolano *message digest* (e diversamente da alcuni dei primi di questi algoritmi, come MD2), operano con quantità a 32 bit, perché queste vengono gestite in modo più efficiente dai moderni processori. Di conseguenza, possiamo immaginare che il valore attuale del riassunto sia composto da 4 parole di 32 bit (d_0, d_1, d_2, d_3) e che la porzione di messaggio che si sta riassumendo sia composta da 16 parole di 32 bit (da m_0 a m_{15}).

La trasformazione elementare eseguita da MD5 può essere scomposta in quattro fasi. Nella prima fase viene prodotto un nuovo valore del riassunto a partire dal vecchio valore

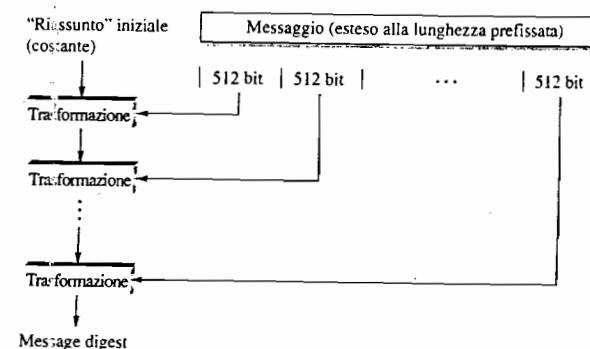


Figura 8.8 Panoramica del funzionamento del riassunto di messaggi (*message digest*).

³ Ciò non significa che qualsiasi insieme di trasformazioni casuali sarebbe adeguato: è necessario verificare che un algoritmo di questo tipo produca veramente uscite casuali.

e dalle 16 parole del messaggio, usando 16 passi, i primi 6 dei quali sono qui indicati:

$$\begin{aligned}d_0 &= (d_0 + F(d_1, d_2, d_3) + m_0 + T_1) \leftarrow 7 \\d_3 &= (d_3 + F(d_0, d_1, d_2) + m_1 + T_2) \leftarrow 12 \\d_2 &= (d_2 + F(d_3, d_0, d_1) + m_2 + T_3) \leftarrow 17 \\d_1 &= (d_1 + F(d_2, d_3, d_0) + m_3 + T_4) \leftarrow 22 \\d_0 &= (d_0 + F(d_1, d_2, d_3) + m_4 + T_5) \leftarrow 7 \\d_1 &= (d_3 + F(d_0, d_1, d_2) + m_5 + T_6) \leftarrow 12\end{aligned}$$

Questo procedimento continua finché tutte le 16 parole sono state riassunte. Ogni passo provoca la riscrittura di una delle parole del riassunto, con un nuovo valore che dipende dal suo vecchio valore, dal valore attuale delle altre tre parole del riassunto e da una delle parole del messaggio che si sta riassumendo. La funzione $F(a, b, c)$ è una combinazione di operazioni sui bit (OR, AND, NOT) dei suoi argomenti. I valori T_i sono costanti. L'operatore " $\leftarrow n$ " ruota di n bit a sinistra l'operando che si trova sua sinistra.

La seconda fase sembra molto simile alla prima (specialmente se i vostri occhi hanno assunto un aspetto vitreo). Le differenze sono queste:

- La funzione F è sostituita da un'altra funzione G un po' diversa.
- Le costanti, da T_1 a T_{16} , sono sostituite da un altro insieme di costanti diverse (da T_{17} a T_{32}).
- La quantità di rotazione verso sinistra ad ogni passo è $\{5, 9, 14, 20, 5, 9, \dots\}$.
- Invece di considerare i byte del messaggio nell'ordine che va da m_0 a m_{15} , il byte che viene usato al passo i è $m_{(5i+1) \bmod 16}$.

Nella terza fase:

- La funzione G è sostituita da un'altra funzione ancora, H , che è semplicemente la funzione XOR applicata ai propri parametri.
- Viene usato un altro insieme di costanti (da T_{33} a T_{48}).
- La quantità di rotazione verso sinistra ad ogni passo è $\{4, 11, 16, 23, 4, 11, \dots\}$.
- Il byte del messaggio che viene usato al passo i è $m_{(3i+5) \bmod 16}$.

A questo punto un lettore attento sarebbe in grado di proporre una quarta fase che sarebbe tanto sicura quanto quella che viene effettivamente utilizzata, ma, per completezza, la quarta fase ha le seguenti proprietà:

- La funzione H è sostituita dalla funzione I , che è una combinazione di operazioni sui bit (XOR, OR, NOT) dei suoi argomenti.
- Viene usato un altro insieme di costanti (da T_{49} a T_{64}).
- La quantità di rotazione verso sinistra ad ogni passo è $\{6, 10, 16, 21, 6, 10, \dots\}$.
- Il byte del messaggio che viene usato al passo i è $m_{(7i) \bmod 16}$.

Dopo tutto questo lavoro, i valori originari di (d_0, d_1, d_2, d_3) sono stati pesantemente rimesscolati, in modo che, pur essendo totalmente correlati ai byte del messaggio, non esiste un modo algoritmico per scoprire quali fossero i byte del messaggio stesso. Il riassunto, così rimesscolato, viene poi aggiunto al valore del riassunto esistente prima della fase attuale, per divenire

8.2 Strategie di sicurezza

il nuovo valore del riassunto. L'algoritmo procede, poi, a calcolare il riassunto dei 16 byte successivi del messaggio, finché non c'è più nulla da riassumere: l'uscita dell'ultimo stadio è il riassunto del messaggio.

Anche se non ha l'efficienza computazionale delle prime forme di riassunti crittografici, MD5 è ancora abbastanza valido da questo punto di vista. Notate che tutte le operazioni sono di facile implementazione nei moderni processori: rotazioni, addizioni, OR, AND, NOT e XOR bit per bit.

8.1.5 Implementazione e prestazioni

DES e MD5 sono parecchi ordini di grandezza più veloci di RSA, quando vengono realizzati mediante software. Con processori allo stato dell'arte, ad esempio, l'algoritmo DES elabora i dati ad una velocità di circa 100 Mbps e MD5 a circa 600 Mbps, mentre RSA arriva solamente a 100 Kbps. In pratica, le prestazioni di RSA non vengono valutate in termini di velocità di throughput, perché non ha senso eseguire RSA su un messaggio di dati. Tipicamente si parla, invece, di RSA in termini del numero di valori che può firmare e verificare in un secondo. Con i processori di oggi, RSA può firmare circa 100 valori a 1024 bit in un secondo e verificarne circa 2000.

Se implementati in hardware, cioè in un chip VLSI dedicato, si è visto che DES e MD5 possono raggiungere velocità misurate in gigabit al secondo, mentre le velocità di RSA rimangono dell'ordine di kilobit al secondo. Per questo motivo DES e MD5 sono, fra i tre algoritmi, quelli che con maggiore probabilità vengono implementati in hardware: RSA, anche quando venisse implementato in hardware, sarebbe ancora troppo lento per essere di utilizzo pratico nella cifratura di messaggi di dati. Al contrario, RSA viene tipicamente usato per cifrare piccole quantità di dati, come una chiave segreta. I protocolli di sicurezza usano poi queste chiavi segrete, protette con RSA, come chiavi per DES e MD5, magari implementati in hardware, per fornire la privacy e l'integrità dei messaggi. Nelle sezioni seguenti vedremo con precisione come si realizza tutto ciò.

8.2 Strategie di sicurezza

Gli algoritmi crittografici sono solamente uno degli elementi del quadro che è necessario comporre quando si vuole garantire la sicurezza all'interno di una rete. La prossima cosa di cui abbiamo bisogno è un insieme di strategie e protocolli per risolvere alcuni problemi. In questa sezione prenderemo in esame le strategie che vengono utilizzate per autenticare i partecipanti ad una comunicazione, le tecniche per garantire l'integrità dei messaggi e alcuni approcci per risolvere il problema della distribuzione delle chiavi pubbliche.

8.2.1 Protocolli di autenticazione

Prima che due partecipanti vogliano stabilire un canale sicuro per le reciproche comunicazioni (usando, cioè, un algoritmo come DES per cifrare i messaggi che si scambiano), vorranno in generale anche determinare che l'altro partecipante sia quello che dice di essere: un problema di autenticazione. Se pensate all'autenticazione nel contesto di una relazione client/server, ad esempio un file system remoto, allora è facilmente comprensibile che il server voglia stabilire l'identità del client: se il client sta per modificare o cancellare un file di John, il server ha l'obbligo di accertarsi che il client sia, a tutti gli effetti, John. Allo stesso tempo,

però, anche il client vuole spesso verificare l'identità del server: dopo tutto, non volete iniziare a scrivere dati sensibili in quello che ritenete essere un file server, per scoprire, soltanto più tardi, che si trattava di un altro processo.

Questa sezione descrive tre comuni protocolli per implementare l'autenticazione. I primi due usano la crittografia a chiave segreta (ad esempio, DES), mentre il terzo usa la crittografia a chiave pubblica (ad esempio, RSA). Notate che è spesso durante il processo di autenticazione che i due partecipanti stabiliscono quale sarà la chiave di sessione che verrà usata per garantire la privacy durante la comunicazione seguente. Nel seguito discutiamo come prende le mosse questo processo.

Semplice accordo in tre fasi (*three-way handshake*)

Quando due entità che vogliono autenticarsi reciprocamente (immaginate un client e un server) condividono già una chiave segreta, è possibile usare un protocollo di autenticazione molto semplice. Si tratta di una situazione analoga a quella di un utente (il client) che sia registrato per l'accesso ad un calcolatore (il server), con il client e il server che conoscono la parola d'accesso (*password*).

Il client e il server si autenticano reciprocamente usando un semplice protocollo per l'accordo in tre fasi (*three-way handshake*), simile a quello già descritto nella Sezione 5.2.3. Nel seguito usiamo $E(m, k)$ per indicare la cifratura del messaggio m con la chiave k e $D(m, k)$ per indicare la decifratura del messaggio m con la chiave k .

Come si può vedere in Figura 8.9, il client per prima cosa seleziona un numero casuale x e lo cifra usando la sua chiave segreta, che indichiamo con CHK (*client handshake key*). Il client, poi, invia al server $E(x, CHK)$, insieme ad un identificativo per se stesso (*ClientId*). Il server usa la chiave che, in base alle sue conoscenze, corrisponde al client *ClientId* (chiamiamola SHK , *server handshake key*) per decifrare il numero casuale. Il server aggiunge 1 al numero che ha ricostruito e restituisce il risultato al client, inviando anche un numero casuale y cifrato con SHK . Successivamente, il client decifra la prima metà di questo messaggio e , se ottiene come risultato il numero casuale x che aveva inviato al server aumentato di uno, sa che il server possiede la sua chiave segreta. A questo punto, il client ha autenticato il server. Il client decifra anche il numero casuale che il server ha spedito (e il risultato dovrebbe essere y), cifra questo numero dopo averlo aumentato di uno e invia il risultato al server. Se il server è in grado di ricostruire $y + 1$, allora sa che il client è quello vero.

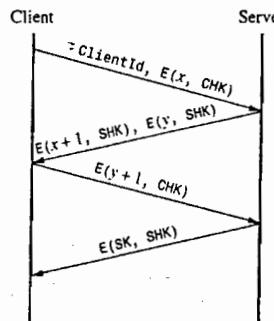


Figura 8.9 Protocollo di accordo in tre fasi (*three-way handshake*) per l'autenticazione.

8.2 Strategie di sicurezza

Dopo il terzo messaggio, ciascuna entità ha autenticato se stessa presso l'altra. Il quarto messaggio della Figura 8.9 corrisponde al fatto che il server invia al client una chiave di sessione (SK , *session key*), cifrata usando SHK (che è uguale a CHK). Tipicamente il client e il server usano poi SK per cifrare i successivi dati che si scambiano. Il vantaggio di usare una chiave di sessione sta nel fatto che, in questo modo, la chiave segreta (che è duratura) viene usata soltanto per un numero limitato di messaggi, rendendo difficile ad un attaccante la raccolta di dati che potrebbero essere usati per determinare la chiave.

Rimane soltanto da rispondere alla domanda: da dove vengono le chiavi di *handshake* che il client e il server usano all'inizio? Una possibilità è che queste corrispondano ad una parola d'accesso (*password*) introdotta dall'utente: in questa situazione il valore di *ClientId* potrebbe essere l'identificativo dell'utente. Dato che una parola d'accesso scelta dall'utente potrebbe non essere adatta come chiave segreta, spesso viene eseguita una trasformazione per ottenere, ad esempio, una chiave DES di 56 bit valida.

Entità terza e fidata (*trusted third party*)

Una situazione più probabile è quella che due entità non sappiano nulla l'una dell'altra, ma si fidino entrambe di una terza entità. Questa terza entità viene a volte chiamata *server di autenticazione* (*authentication server*) e usa un protocollo per aiutare le due entità ad autenticarsi reciprocamente. Esistono, in realtà, molte varianti di questo protocollo; descriviamo quella usata in Kerberos, un sistema di sicurezza sviluppato al MIT e basato sull'architettura TCP/IP.

Nel seguito, indichiamo con A e B le due entità che vogliono autenticarsi reciprocamente e chiamiamo S il server di autenticazione fidato (di cui entrambe si fidano). Il protocollo Kerberos parte dall'ipotesi che sia A che B condividano con S una chiave segreta; indichiamo con K_A e K_B , rispettivamente, queste due chiavi. Come prima, $E(m, k)$ indica la cifratura del messaggio m con la chiave k .

Come mostrato in Figura 8.10, l'entità A invia per prima cosa un messaggio al server S , per identificare se stessa e B . Il server genera un'indicazione oraria (*timestamp*) T , un tempo di vita (*lifetime*) L e una nuova chiave di sessione K . L'indicazione oraria T avrà all'incirca la medesima funzione del numero casuale usato nel semplice protocollo di accordo in tre fasi descritto prima, oltre al fatto che viene usata insieme a L per limitare il tempo di validità della chiave di sessione K . Quando terminerà questo tempo, le entità A e B dovranno rivolgersi

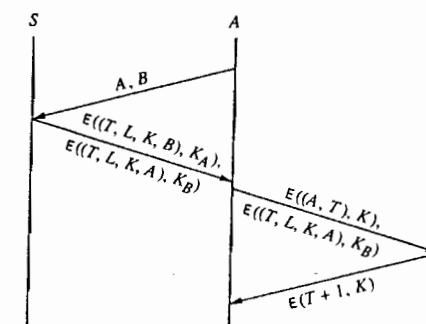


Figura 8.10 Autenticazione con una entità terza in Kerberos.

nuovamente al server S per ottenere una nuova chiave di sessione. In questo caso, l'obiettivo è quello di limitare la vulnerabilità di tutte le chiavi di sessione.

In seguito, il server S risponde ad A con un messaggio composto da due parti. La prima parte contiene i tre valori, T , L e K , insieme all'identificativo per l'entità B , cifrati con la chiave condivisa tra il server e A . La seconda parte contiene i tre valori, T , L e K , insieme all'identificativo per l'entità A , cifrati questa volta con la chiave condivisa tra il server e B . Chiaramente, quando A riceve questo messaggio, sarà in grado di decifrarne la prima parte, ma non la seconda: questa seconda parte viene semplicemente inviata da A a B , insieme ai valori di A e T cifrati usando la nuova chiave di sessione K (A è riuscito a ricostruire T e K decifrando la prima parte del messaggio ricevuto da S). Infine, B decifra la parte di messaggio ricevuta da A che era stata originariamente cifrata da S : nel fare ciò, ricostruisce T , K e A . Usando K , decifra la parte di messaggio cifrata da A e, dopo aver osservato che i valori di A e T sono uguali nelle due parti del messaggio, risponde con un messaggio che contiene il valore $T + 1$ cifrato usando la nuova chiave di sessione K .

A e B possono ora comunicare reciprocamente usando la chiave di sessione segreta e condivisa, K , per avere la privacy garantita.

Autenticazione con chiave pubblica

L'ultimo protocollo di autenticazione che presentiamo usa la crittografia a chiave pubblica (ad esempio, RSA). Il protocollo a chiave pubblica è utile perché le due parti non hanno bisogno di condividere una chiave segreta: hanno solamente bisogno di conoscere la chiave pubblica dell'altra entità. Come mostrato in Figura 8.11, l'entità A cifra un numero casuale x usando la chiave pubblica di B , e B dimostra di conoscere la corrispondente chiave privata decifrando il messaggio e restituendo x ad A . A questo punto, A può autenticare se stesso nei confronti di B esattamente nello stesso modo.

8.2.2 Protocolli per l'integrità dei messaggi

A volte due entità che comunicano non si curano del fatto che qualcuno sia abusivamente in grado di leggere i messaggi che si scambiano, ma vogliono evitare che un impostore possa inviare messaggi asserendo di essere una delle due entità: le due entità vogliono che sia garantita l'integrità dei loro messaggi.

Uno dei modi per garantire l'integrità di un messaggio consiste nel cifrarlo usando DES con la tecnica CBC (*cipher block chaining*), per poi usare il *residuo* di CBC (l'ultimo blocco

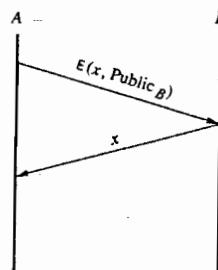


Figura 8.11 Autenticazione con chiave pubblica.

8.2 Strategie di sicurezza

Scambio di chiavi di Diffie-Hellman

La nostra presentazione delinea una distinzione tra gli algoritmi crittografici (come RSA) e i protocolli di sicurezza che usano gli algoritmi crittografici per fornire i servizi di autenticazione e di integrità dei messaggi (ad esempio, il protocollo di firma digitale che usa RSA). In realtà, questa distinzione non è sempre così evidente, dal momento che esistono protocolli di sicurezza strettamente interconnessi con algoritmi crittografici. Lo scambio di chiavi di Diffie-Hellman, che permette a due utenti di scambiarsi una chiave segreta lungo un canale insicuro in un modo che ricorda la costruzione della chiavi pubblica e privata in RSA, è un esempio di questo tipo ampiamente utilizzato. Diffie-Hellman è un meccanismo potente perché consente a due entità di concordare sull'utilizzo di una chiave segreta (che in seguito può essere utilizzata da un algoritmo a cifratura simmetrica, come DES) senza doversi affidare ad una terza entità di cui si fidino entrambe.

Il protocollo ha due parametri, p e g , entrambi pubblici e che possono essere utilizzati da tutti gli utenti di un sistema. Il parametro p è un numero primo ed il parametro g (solitamente chiamato generatore) è un numero intero minore di p , con la proprietà che per ogni numero n , compreso fra 1 e $p - 1$, esiste una potenza k di g tale che

$$n = g^k \bmod p$$

Supponete che Alice e Bob vogliano accordarsi su una chiave segreta da condividere. Dapprima Alice genera un valore casuale privato, a , e Bob ne genera un altro, b . Sia a che b vengono estratti dall'insieme di numeri interi $\{1, \dots, p - 2\}$. Alice e Bob derivano, nel modo seguente, i propri valori pubblici. Il valore pubblico di Alice è

$$g^a \bmod p$$

e il valore pubblico di Bob è

$$g^b \bmod p$$

Si scambiano poi questi valori e, infine, Alice calcola

$$g^{ab} = (g^b)^a \bmod p$$

e Bob calcola

$$g^{ba} = (g^a)^b \bmod p$$

Dato che

$$g^{ab} = g^{ba} = k$$

Alice e Bob condividono ora la chiave segreta k .

uscito dal processo CBC) come *codice di integrità del messaggio* (MIC, *message integrity code*). Nell'esempio di CBC della Figura 8.7, il residuo di CBC è cipher₄. Il messaggio di testo non cifrato viene trasmesso al destinatario insieme al MIC, che agisce più o meno come una somma di controllo: se il destinatario non è in grado di riprodurre il MIC ricevuto usando la chiave segreta che condivide con il mittente, allora il messaggio non è stato inviato da quel mittente, oppure è stato modificato rispetto a quando era stato trasmesso. Notate che non ha senso usare DES con CBC sia per cifrare il messaggio (per scopi di privacy) sia per generare il MIC (per scopi di integrità), perché si finirebbe per trasmettere semplicemente il messaggio cifrato con CBC con l'ultimo blocco ripetuto. Di conseguenza, chiunque volesse illecitamente modificare il messaggio cifrato con CBC potrebbe prendere il valore del blocco finale di ciò che vuole inviare e inviarlo due volte.

Questa sezione esamina tra alternative per garantire l'integrità dei messaggi. La prima usa RSA per generare una firma digitale (*digital signature*): la cifratura RSA usata da sola tende ad essere lenta, ma la si può usare unitamente a MD5 per ottenere una tecnica molto più efficiente. Il secondo ed il terzo approccio usano MD5 (eventualmente insieme a RSA) per garantire l'integrità dei messaggi.

Firma digitale con RSA (*digital signature*)

Una *firma digitale* (*digital signature*) è un caso speciale di codice per l'integrità dei messaggi: il codice può essere stato generato da uno soltanto dei partecipanti alla comunicazione. L'algoritmo di firma digitale più semplice da capire è la firma RSA, che funziona in modo ovvio: dato che un partecipante è l'unico che conosce la propria chiave privata, usa tale chiave per generare la firma. Qualsiasi altro partecipante può verificare tale firma usando la corrispondente chiave pubblica. In altre parole, per firmare un messaggio, lo si cifra usando la propria chiave privata, mentre per verificare una firma, lo si decifra usando la chiave pubblica del presunto mittente. Ovviamente, ciò significa che generare una firma con RSA è altrettanto lento quanto RSA, che abbiamo già visto essere due o tre ordini di grandezza più lento di DES. Osservate che l'uso delle chiavi è esattamente opposto a come vengono utilizzate per garantire la privacy: il mittente cifra con la propria chiave privata invece che con la chiave pubblica del destinatario, e il destinatario decifra con la chiave pubblica del mittente piuttosto che con la propria chiave privata.

È stato proposto, dal NIST (National Institute for Standards and Technology), uno standard per la firma digitale noto come DSS, che è simile all'approccio appena descritto, tranne per il fatto che usa un diverso algoritmo, chiamato El Gamal, invece di RSA.

MD5 con chiave (*keyed MD5*)

Ricordate che MD5 genera una somma di controllo crittografica per un messaggio, somma che non dipende da una chiave segreta, per cui non è impossibile che un impostore crei un messaggio fingendo che provenga da qualcun altro, calcolando poi la somma di controllo MD5 per tale messaggio. Esistono, però, due metodi che consentono di usare MD5 per realizzare l'integrità dei messaggi ed entrambi gli approcci superano i problemi di prestazioni che insorgono quando si usa RSA.

Il primo metodo, che viene solitamente chiamato *MD5 con chiave* (*keyed MD5*), funziona così. Supponete di poter fare in modo che mittente e destinatario di un messaggio condividano una chiave segreta *k*, per effetto di una qualche preconfigurazione delle chiavi oppure di un meccanismo più dinamico, come Kerberos. Il mittente, quindi, esegue MD5 sul risultato della concatenazione del messaggio (indicato con *m*) e di questa chiave. In pratica, la

8.2 Strategie di sicurezza

chiave *k* viene ad essere un prolungamento del messaggio, ai fini dell'esecuzione di MD5, e viene eliminata dal messaggio dopo tale fase. Il mittente trasmette, quindi:

$$m + \text{MD5}(m + k)$$

dove MD5(*s*) rappresenta il risultato dell'applicazione dell'algoritmo MD5 alla stringa *s*, e *a* + *b* indica il risultato della concatenazione delle stringhe *a* e *b*.

Il destinatario del messaggio applica di nuovo MD5 al corpo del messaggio concatenato con la chiave segreta *k*: se il risultato corrisponde alla somma di controllo inviata insieme al messaggio, allora il messaggio deve essere stato inviato effettivamente dall'entità che detiene quella chiave.

La tecnica MD5 con chiave, per se stessa, non dipende dalla crittografia a chiave pubblica, ma può essere combinata con tale strategia per semplificare il problema di fare in modo che mittente e destinatario condividano una chiave segreta *k*. Il mittente sceglie *k* a caso, la cifra usando RSA con la chiave pubblica del destinatario, poi cifra nuovamente il risultato con la propria chiave privata. Ora il risultato può essere inviato al destinatario insieme al messaggio originario e alla somma di controllo MD5; per riassumere, quindi, il messaggio completo trasmesso dal mittente è:

$$m + \text{MD5}(m + k) + E(E((k, \text{chiave_pubblica_destinatario}), \text{chiave_privata_mittente}))$$

Il destinatario estrae la chiave casuale usando la chiave pubblica RSA del presunto mittente e la propria chiave privata, poi procede eseguendo MD5 sulla concatenazione del messaggio ricevuto e di *k*. Come prima, se il risultato coincide con la somma di controllo inviata insieme al messaggio, allora il messaggio deve essere stato inviato dall'entità che ha generato la chiave casuale. Anche se questo approccio risolve il problema di trasferire la chiave segreta dal mittente al destinatario, lascia aperto il problema di consegnare in modo affidabile al destinatario la chiave pubblica del mittente, un problema di cui parleremo nella Sezione 8.2.3.

MD5 con firma RSA

Il secondo metodo per usare MD5 allo scopo di garantire l'integrità del messaggio funziona in combinazione con RSA, nel modo seguente. Il mittente esegue MD5 sul messaggio originale che vuole proteggere, generando una somma di controllo MD5, poi firma questa somma di controllo con la propria chiave privata RSA: il mittente, cioè, non firma l'intero messaggio, ma soltanto la somma di controllo. Vengono poi trasmessi il messaggio originario, la somma di controllo MD5 e la firma RSA. Usando la medesima notazione introdotta precedentemente, ciò significa che il mittente trasmette

$$m + \text{E}(\text{MD5}(m), \text{chiave_privata_mittente})$$

Il destinatario verifica il messaggio in questo modo:

- esegue l'algoritmo MD5 sul messaggio ricevuto
- decifra la somma di controllo firmata, utilizzando la chiave pubblica del mittente
- confronta le due somme di controllo

Se sono uguali, ciò significa che il messaggio non è stato modificato dal momento in cui il mittente ha calcolato la somma di controllo MD5 e l'ha firmato.

8.2.3 Distribuzione di chiavi pubbliche (X.509)

La crittografia a chiave pubblica è una tecnologia estremamente potente, ma si affida alla distribuzione delle chiavi pubbliche. Il problema di fare in modo che le chiavi pubbliche arrivino alle persone che ne hanno bisogno in modo che possano essere certe della validità delle chiavi stesse (cioè che appartengano all'entità a cui si presume che appartengano) risulta essere un problema arduo. Questa sezione esamina il problema e alcune delle soluzioni generali del problema stesso. Nella Sezione 8.3 vengono descritti alcuni specifici sistemi che hanno tentato di risolvere il problema.

Supponete che l'entità *A* voglia far pervenire a *B* la propria chiave pubblica. Non può usare semplicemente la posta elettronica o una bacheca pubblica per inviarla, perché senza la chiave pubblica di *A*, *B* non ha modo di autenticare la chiave per accertarsi che provenga veramente da *A*: una terza entità potrebbe inviare una chiave pubblica a *B* e dichiarare che il messaggio proviene da *A*. Se *A* e *B* sono individui che si conoscono nella vita reale, possono trovarsi nella stessa stanza e *A* può fornire direttamente la propria chiave pubblica a *B*, magari con un biglietto da visita. Tuttavia, questo approccio ha evidenti svantaggi, vista l'impossibilità di ricevere una chiave da qualcuno con il quale non ci si possa incontrare fisicamente. La soluzione basilare per questo problema si affida all'utilizzo di *certificati digitali*. Le sezioni che seguono spiegano cosa sono i certificati digitali e alcuni dei problemi che sorgono usando per ottenere l'obiettivo della distribuzione delle chiavi.

Certificati

Nella Sezione 8.2.2 abbiamo introdotto la nozione di *firma digitale*, mediante la quale il proprietario di una determinata chiave può firmare una certa quantità di dati, crittografandoli. Una firma digitale attesta che i dati sono stati generati dal proprietario di una certa chiave e che non sono stati modificati dal momento in cui sono stati firmati. Un certificato è, semplicemente, un tipo speciale di documento con firma digitale. Il documento afferma, in sostanza: "Io certifico che la chiave pubblica presente in questo documento appartiene all'entità nominata in questo documento, firmato *X*". In questo caso, *X* può essere chiunque abbia una chiave pubblica. Solitamente *X* è un'autorità di certificazione (CA, *certification authority*)⁴, cioè un'entità amministrativa che si occupa di emettere certificati. Dovrebbe essere evidente che questo certificato ha una qualche utilità solamente per quelle entità che già possiedono la chiave pubblica di *X*, perché tale chiave è necessaria per verificare la firma. Di conseguenza, il certificato, da solo, non risolve il problema della distribuzione delle chiavi, ma ci fornisce una strada per risolverlo. Ovviamente, una volta che avete la chiave pubblica di un'entità *X*, potete iniziare a ricevere chiavi pubbliche di altre entità, a patto che queste entità siano in grado di ottenere certificati emessi da *X*.

L'idea dei certificati consente la costruzione di una "catena fiduciaria". Se *X* certifica che una certa chiave pubblica appartiene a *Y* e poi *Y* certifica che un'altra chiave pubblica appartiene a *Z*, allora esiste una catena di certificati da *X* a *Z*, anche se *X* e *Z* non si sono mai incontrati. Se *Z* vuole fornire la propria chiave pubblica ad *A*, può fornire la catena completa dei certificati: il certificato per la chiave pubblica di *Y* emesso da *X* e il certificato per la chiave pubblica di *Z* emesso da *Y*. Se *A* possiede la chiave pubblica di *X*, può usare la catena per verificare che la chiave pubblica di *Z* sia valida.

⁴ Le entità CA sono anche note con il nome di *certificate authority*.

8.2 Strategie di sicurezza

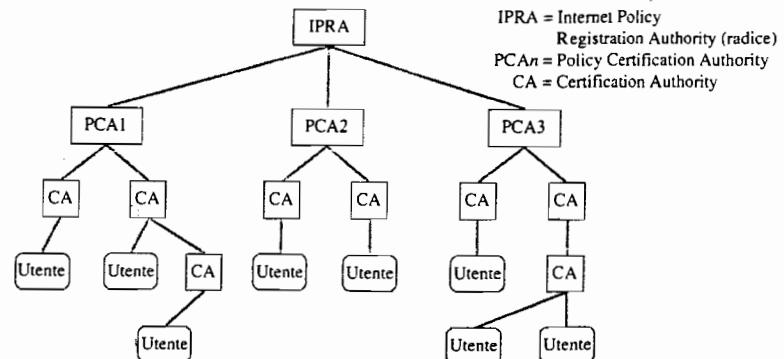


Figura 8.12 Gerarchia di autorità di certificazione strutturata ad albero.

Con questa idea relativa alla costruzione di catene fiduciarie, la distribuzione delle chiavi pubbliche diviene, in qualche modo, più trattabile. Un modo abbastanza diffuso per la costruzione di tali catene consiste nel disporle in una gerarchia strutturata ad albero, come mostrato nella Figura 8.12. Se tutti possiedono la chiave pubblica della CA che funge da radice, allora ogni entità può fornire una catena di certificati ad altre entità, sapendo che sarà sufficiente costruire una catena fiduciaria per tale entità.

Nella costruzione di catene fiduciarie ci sono ancora problemi sostanziali. Prima di tutto, anche se siete sicuri di possedere la chiave pubblica della CA che funge da radice, dovete avere la garanzia che ogni CA dalla radice verso i rami sta svolgendo in modo corretto il proprio compito. Se alcune CA sono disposte ad emettere certificati a individui senza verificarne l'identità, allora ciò che sembra una catena di certificati valida perde, in realtà, di significato. Nella Sezione 8.3.1 viene presentato un diverso approccio a questo problema, in cui le catene fiduciarie costituiscono una maglia arbitraria invece di un rigido albero.

Una domanda che ci dobbiamo porre in merito ai certificati è: che cosa viene certificato? Dato che un certificato crea una corrispondenza tra un'identità e una chiave pubblica, dovremmo capire meglio cosa sia per noi una "identità". Ad esempio, un certificato che afferma "Questa chiave pubblica appartiene a John Smith" non sarebbe molto utile, se non ci dice anche quale delle migliaia di John Smith sta identificando. I certificati, quindi, devono usare un ben definito spazio dei nomi per le identità che certificano. Ad esempio, spesso i certificati vengono emessi per indirizzi di posta elettronica: un certificato potrebbe affermare, in effetti, che "Questa chiave pubblica appartiene a jsmith@acme.com (John Q. Smith)".

Ovviamente i certificati possono essere emessi anche per molte entità che non siano individui. In particolare, è utile poter emettere un certificato per un dominio, all'interno del Domain Name System. Ad esempio, un certificato per il dominio acme.com sarebbe utile per consentire ai clienti che visitano il sito Web acme.com di avere la certezza di essere giunti nel vero sito Web e non in quello di un impostore, prima di inviare le proprie informazioni relative alla carta di credito.

Uno degli standard più diffusi per i certificati è X.509. Questo standard lascia aperti molti dettagli, ma specifica una struttura di base per i certificati. Le componenti di un certificato devono, ovviamente, contenere:

- il nome dell'entità che si sta certificando
- la chiave pubblica dell'entità
- il nome dell'autorità di certificazione
- una firma digitale

I certificati che seguono lo standard X.509 possono usare molti diversi algoritmi di firma digitale, per cui il certificato deve specificare quale algoritmo usa. Un'altra informazione possibile è la data di scadenza del certificato, di cui parleremo in seguito.

Una cosa importante da capire sui certificati è che il possesso di un certificato non dice nulla sulla vostra identità. I certificati possono essere liberamente copiati e distribuiti, e in verità bisogna farlo perché siano utili. Per dimostrare che voi siete l'entità denominata nel certificato, dovete fare qualcosa che dimostri che possedete la chiave *privata* corrispondente alla chiave pubblica contenuta nel certificato: si tratta, ovviamente, del problema dell'autenticazione, descritto nella Sezione 8.2.1.

Revoca di certificati

Un problema che sorge con i certificati è come si possa revocare, o annullare, un certificato. Perché ciò è importante? Supponete di avere il sospetto che qualcuno abbia scoperto la vostra chiave privata. Può darsi che nell'universo esistano molti certificati che affermano che voi siete il proprietario della chiave pubblica corrispondente a quella chiave privata. La persona che ha scoperto la vostra chiave privata ha quindi tutto ciò che serve per impersonare la vostra identità: certificati validi e la vostra chiave privata. Per risolvere questo problema, sarebbe utile poter revocare i certificati che mettono in corrispondenza la vostra vecchia chiave, ormai compromessa, e la vostra identità, per fare in modo che l'impostore non sia più in grado di persuadere altre persone di essere voi.

La soluzione basilare al problema è abbastanza semplice. Un'autorità di certificazione può emettere un *elenco di revoca di certificati* (CRL, *certificate revocation list*), che consiste in un elenco firmato digitalmente di certificati che sono stati revocati. Il CRL viene periodicamente aggiornato e reso pubblico: essendo firmato digitalmente, può essere semplicemente affisso in una bacheca elettronica pubblica. Ora, quando l'entità A riceve un certificato di B e lo vuole verificare, per prima cosa consulterà l'ultimo CRL emesso dall'autorità di certificazione. Un certificato è valido finché non viene revocato; se tutti i certificati avessero una durata illimitata, il CRL diventerebbe sempre più lungo, perché non si potrebbe mai eliminare un certificato dall'elenco, nel timore che possa ancora essere utilizzata una copia del certificato revocato. Tuttavia, assegnando una data di scadenza ad un certificato, nel momento in cui questo viene emesso, possiamo limitare il periodo di tempo durante il quale un certificato revocato deve permanere nel CRL: non appena è trascorso il periodo di validità del certificato, esso può essere rimosso dal CRL.

8.3 Esempi di sistemi

A questo punto abbiamo visto molte delle componenti che servono per costruire un sistema sicuro: algoritmi di crittografia, protocolli di autenticazione e meccanismi di distribuzione delle chiavi. In questa sezione prendiamo in esame alcuni sistemi completi che usano tali componenti.

Questi sistemi si possono sostanzialmente classificare in base allo strato di protocollo nel quale operano. I protocolli IPSEC (IP Security), come si può capire dal nome, operano

nello strato IP (strato di rete). Tra i sistemi che operano nello strato di applicazione troviamo PGP (Pretty Good Privacy), che rende sicura la posta elettronica, e SSH (Secure Shell), un'applicazione per rendere sicura la connessione a sistemi remoti. Un certo numero di protocolli operano a livello intermedio, al livello di trasporto, tra i quali citiamo lo standard TLS (Transport Layer Security) di IETF e il protocollo precedente dal quale deriva, SSL (Secure Socket Layer). Le sezioni seguenti descrivono le caratteristiche salienti di ognuno di questi approcci.

8.3.1 Pretty Good Privacy (PGP)

Pretty Good Privacy (PGP, "privacy molto buona") è una soluzione molto diffusa per aggiungere alla posta elettronica la possibilità di cifrare i messaggi e di autenticare il mittente. L'aspetto più interessante di PGP consiste nella gestione dei certificati. Ricordate che il problema di fondo nella distribuzione delle chiavi pubbliche è la costituzione di una catena fiduciaria: PGP riconosce che ogni utente ha un proprio insieme di criteri in base ai quali può voler dare fiducia a chiavi certificate da altri e fornisce gli strumenti necessari per gestire il livello di confidenza che ognuno ripone in tali certificati.

Per comprendere meglio il problema, supponete che A, una persona che conoscete bene, vi consegni personalmente la sua chiave pubblica. In questo caso, siete ben certi che si tratti proprio della sua chiave pubblica. Se, invece, A vi consegna un certificato relativo a B, firmato da A, dovreste pensare se A sia il tipo di persona che sarebbe capace di firmare un falso certificato in cambio di denaro, oppure se possa essere un po' disattenta nel verificare che sia stato veramente B a chiedere di firmare quel certificato e non un'altra persona. Potreste concedere fiducia ad A nel momento in cui firma certificati per alcune persone (ad esempio, i suoi colleghi di lavoro) ma non per altre (ad esempio, personaggi politici). Le cose ovviamente peggiorano man mano che la catena di "fiducia" (o di sfiducia) si allunga.

Piuttosto che forzare l'utilizzo di una gerarchia di certificazione rigida, come era stato fatto in un precedente sistema di posta elettronica sicura chiamato PEM (Privacy Enhanced Mail), PGP permette che le relazioni di certificazione costituiscano una griglia arbitraria. Inoltre, consente a ciascun utente di decidere autonomamente quanta fiducia assegnare ad un certificato. Ad esempio, supponete di avere un certificato per B fornito da A: potete assegnare a tale certificato un livello di fiducia modesto, ma, se avete altri certificati per B forniti da C e D, ognuno dei quali è altrettanto moderatamente degno di fiducia, il vostro livello di confidenza relativamente al fatto che la chiave pubblica di B che possedete sia valida può aumentare considerevolmente. In breve, PGP riconosce che il problema di concedere fiducia è una cosa abbastanza personale e fornisce all'utente gli strumenti di base che gli consentano di prendere decisioni, piuttosto che costringerli a dare fiducia incondizionata ad una rigida gerarchia di autorità di certificazione. Citando Phil Zimmermann, lo sviluppatore di PGP, "PGP è adatto alle persone che preferiscono ripiegare personalmente il proprio paracadute".

PGP è un sistema divenuto piuttosto popolare nella comunità delle reti di calcolatori e le sessioni di firma di chiavi sono ormai normali negli incontri di IETF. In queste occasioni, ogni persona può:

- raccogliere chiavi pubbliche da altre persone di cui conosce l'identità
- fornire la propria chiave pubblica ad altri
- far firmare la propria chiave pubblica da altri, ottenendo così certificati che saranno persuasivi per un insieme di persone sempre più esteso

- firmare la chiave pubblica di altre persone, aiutandole così a costituire un proprio insieme di certificati da poter usare per distribuire la propria chiave pubblica
- raccogliere certificati da altre persone di cui si ha sufficiente fiducia.

In questo modo, al passare del tempo un utente raccoglierà un insieme di certificati con vario grado di fiducia. PGP memorizza questi certificati in un file denominato *mazzo di chiavi (key ring)*.

Supponete ora che l'utente A voglia inviare un messaggio all'utente B, fornendo a B le prove che il messaggio proviene veramente da A. PGP segue la sequenza di fasi mostrate in Figura 8.13. Per prima cosa, A genera una somma di controllo crittografica per il corpo del messaggio (usando, ad esempio, MD5) e, successivamente, cifra tale somma usando la chiave privata di A (PGP consente l'utilizzo di molti diversi algoritmi crittografici e nel messaggio viene indicato quale sia stato utilizzato).

Dopo aver ricevuto il messaggio, B usa il software di PGP per la gestione delle chiavi per cercare la chiave pubblica di A nel suo mazzo di chiavi. Se non la trova, ovviamente B non è in grado di verificare l'autenticità del messaggio. Se trova la chiave, allora calcola la somma di controllo per il messaggio ricevuto, decifra con la chiave pubblica di A la somma di controllo cifrata che ha ricevuto e confronta le due somme: se sono uguali, B è certo che il messaggio sia stato inviato da A e che non sia stato modificato dopo essere stato firmato. Oltre a segnalare il risultato della verifica della firma, PGP segnala a B il livello di fiducia che aveva precedentemente assegnato alla chiave pubblica in oggetto, basandosi sul numero di certificati che possiede per A e sul livello di fiducia assegnato agli individui che hanno firmato tali certificati.

La cifratura di un messaggio è ugualmente semplice ed è riassunta nella Figura 8.14. A sceglie una chiave a caso, diversa per ogni messaggio, e la usa per cifrare il messaggio usando un algoritmo simmetrico come DES. La chiave specifica del messaggio viene cifrata usando la chiave pubblica del destinatario: PGP recupera questa chiave dal mazzo di chiavi di A e segnala ad A il livello di confidenza che aveva assegnato a tale chiave. Il messaggio viene codificato per prevenire danneggiamenti da parte di gateway di posta elettronica e viene inviato a B. Quando B lo riceve, usa la propria chiave privata per decifrare la chiave specifica del messaggio, e poi usa quest'ultima per decifrare il messaggio con l'algoritmo appropriato. PGP consente l'utilizzo di molti diversi algoritmi crittografici nelle varie funzioni: gli algoritmi

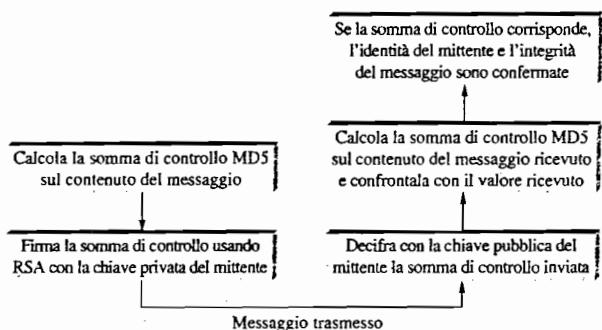


Figura 8.13 Integrità del messaggio e autenticazione con PGP.

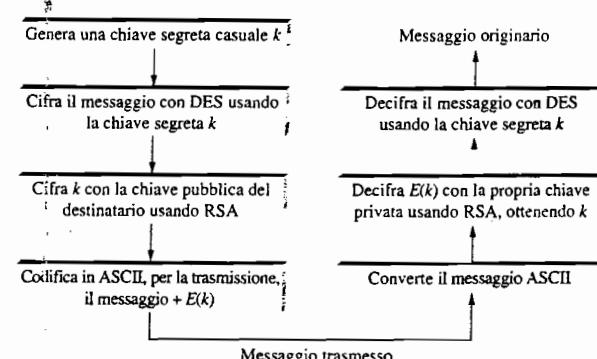


Figura 8.14 Cifratura di un messaggio con PGP.

che vengono usati in un determinato messaggio sono specificati nella sua intestazione. Progettare un sistema di sicurezza indipendente dal protocollo è un'ottima idea, perché non è possibile prevedere quando il vostro algoritmo crittografico preferito si dimostrerà insufficientemente robusto per le vostre esigenze. Sarebbe comodo poter passare rapidamente ad un nuovo algoritmo senza dover modificare la specifica o l'implementazione del protocollo. Oltre ad inserire nel messaggio di posta elettronica queste informazioni, PGP consente all'utente di elencare i suoi algoritmi preferiti nel file che contiene la sua chiave pubblica: in questo modo, chiunque possieda la sua chiave pubblica saprà quali algoritmi si possono usare per spedire un messaggio.

8.3.2 Secure Shell (SSH)

SSH (Secure Shell) fornisce un servizio di accesso remoto a calcolatori e ha lo scopo di sostituire programmi meno sicuri, come Telnet e rlogin, usati fin dai primi giorni di Internet (SSH può anche essere usato per eseguire comandi remotamente e per trasferire file, come, rispettivamente, i comandi Unix rsh e rcp, ma concentreremo la nostra attenzione su come SSH fornisca il servizio di accesso remoto). L'utilizzo principale di SSH consiste in un'autenticazione client/server robusta (con il client SSH che viene eseguito sulla macchina dell'utente, mentre il server SSH è in esecuzione sul calcolatore al quale l'utente vuole accedere remotamente), ma vi è anche la possibilità di garantire l'integrità dei messaggi e la privacy. Telnet e rlogin non hanno alcuna di queste caratteristiche.

Per apprezzare maggiormente l'importanza di SSH nell'Internet dei giorni nostri, considerate che fino a pochi anni fa molte persone usavano modem su linee telefoniche commutate per connettere i propri calcolatori di casa a quelli dell'ufficio (o di scuola). Ciò significa che ogni volta che richiedevano l'accesso le loro *password* venivano inviate in chiaro lungo una linea telefonica e attraverso la LAN del luogo di lavoro. L'invio di una password in chiaro attraverso una LAN non è una buona idea, ma almeno non è rischioso come inviarla attraverso Internet. Oggi, però, molte persone hanno abbonamenti con fornitori di accesso ad Internet (ISP) che offrono un servizio DSL o con modem via cavo ad alta velocità, e si connettono ai calcolatori dell'ufficio passando attraverso questi ISP. Ciò significa che quan-

do richiedono l'accesso, sia le loro password sia i dati che inviano e ricevono attraversano potenzialmente un gran numero di reti di cui non ci si può fidare. SSH fornisce uno strumento per cifrare i dati inviati lungo queste connessioni e migliora la robustezza dei meccanismi di autenticazione utilizzati per richiedere l'accesso.

L'ultima versione di SSH, la versione 2, consiste di tre protocolli:

- SSH-TRANS: un protocollo dello strato di trasporto
- SSH-AUTH: un protocollo di autenticazione
- SSH-CONN: un protocollo di connessione

Ci concentriamo sui primi due, che hanno un ruolo nell'accesso remoto. Parleremo brevemente degli scopi di SSH-CONN alla fine della sezione.

SSH-TRANS mette a disposizione un canale cifrato fra il client e il server e viene eseguito al di sopra di una connessione TCP. Ogni volta che un utente usa SSH per connettersi ad un calcolatore remoto, il primo passo consiste nell'instaurazione di un canale SSH-TRANS fra questi due calcolatori. Perché due calcolatori instaurino questo canale sicuro, la prima azione consiste nell'autenticazione del server presso il client mediante RSA. Successivamente, il client e il server si accordano su una chiave di sessione che useranno per cifrare tutti i dati trasmessi lungo il canale. Questa descrizione di alto livello sorvola su parecchi dettagli, tra i quali il fatto che il protocollo SSH-TRANS contiene una negoziazione dell'algoritmo di cifratura che verrà usato dalle due parti. Ad esempio, spesso viene scelto 3DES. Ancora, SSH-TRANS prevede la verifica dell'integrità del messaggio per tutti i dati che vengono scambiati attraverso il canale.

L'unico dettaglio che non possiamo trascurare è il modo in cui il client viene in possesso della chiave pubblica del server, di cui ha bisogno per autenticare il server stesso. Anche se può sembrare strano, il server comunica al client la propria chiave pubblica al momento della connessione. La prima volta che un client si connette ad un certo server, SSH avverte l'utente che non ha mai effettuato connessioni con tale calcolatore e chiede se vuole procedere. Sebbene sia un'azione rischiosa, dato che SSH non è effettivamente in grado di autenticare il server, spesso gli utenti rispondono "sì" a questa domanda. In seguito, SSH memorizza la chiave pubblica del server e nella successiva occasione in cui l'utente si connette allo stesso calcolatore la chiave memorizzata viene confrontata con quella fornita dal server in quel momento. Se sono uguali, SSH autentica il server. Se sono diverse, però, SSH avverte nuovamente l'utente che c'è qualche problema, e l'utente ha la possibilità di interrompere la connessione. In alternativa, un utente particolarmente prudente può recuperare la chiave pubblica del server con qualche meccanismo fuori linea, memorizzarla nel proprio calcolatore client e non correre mai, in questo modo, il rischio della "prima volta".

Dal momento in cui esiste il canale SSH-TRANS, il passo successivo prevede che l'utente richieda effettivamente l'accesso al calcolatore remoto o, più specificatamente, si autentichi presso il server. SSH mette a disposizione tre diversi meccanismi per eseguire questa operazione. Il primo prevede che, dal momento che i due calcolatori stanno comunicando tramite un canale sicuro, sia sufficiente per l'utente inviare semplicemente la propria password al server: questa operazione non è sicura quando si usa Telnet, perché la password verrebbe inviata in chiaro, ma nel caso di SSH la password viene cifrata dal canale SSH-TRANS. Il secondo meccanismo usa la cifratura a chiave pubblica: ciò richiede che l'utente abbia comunicato preventivamente al server la propria chiave pubblica. Il terzo meccanismo, chiamato "autenticazione basata sull'host" (*host-based authentication*), prevede che qualsiasi utente

che dichiari di essere "chiunque" e provenga da un certo insieme di host fidati viene automaticamente ritenuto lo stesso utente sul server. L'autenticazione basata sull'host richiede che l'*host* client autentichi se stesso presso il server nel momento della prima connessione, mentre il protocollo SSH-TRANS standard prevede la sola autenticazione del server.

L'elemento principale che dovreste ricavare da questa discussione è che SSH è un'applicazione abbastanza lineare dei protocolli e degli algoritmi che avete visto in questo capitolo. Tuttavia, ciò che rende SSH difficile da capire sono tutte le chiavi che un utente deve creare e gestire, per le quali l'interfaccia esatta dipende dal sistema operativo. Ad esempio, il pacchetto OpenSSH che viene eseguito sulla maggior parte dei calcolatori Unix mette a disposizione un comando `ssh-keygen` per creare copie di chiavi pubblica e privata, che vengono memorizzate in diversi file all'interno della directory `.ssh` nella *home directory* dell'utente. Ad esempio, il file `~/.ssh/known_hosts` memorizza le chiavi di tutti gli host ai quali l'utente si è connesso, il file `~/.ssh/authorized_keys` contiene le chiavi pubbliche necessarie per autenticare l'utente quando si connette al calcolatore su cui si trova il file stesso (cioè è un file usato sul server) e il file `~/.ssh/identity` contiene le chiavi private necessarie per autenticare l'utente sui calcolatori remoti (cioè è un file usato sul client).

Infine, SSH si è dimostrato così utile come sistema per rendere sicuri gli accessi remoti che è stato esteso per fornire supporto anche ad altre applicazioni insicure basate su TCP, come X Windows e i lettori di posta elettronica con protocollo IMAP. L'idea è quella di eseguire queste applicazioni all'interno di un "tunnel SSH" sicuro, una caratteristica che viene chiamata *inoltro di porta (port forwarding)* e che usa il protocollo SSH-CONN. In Figura 8.15 è rappresentato lo schema di funzionamento, con un client sull'host A che comunica indirettamente con un server sull'host B, inoltrando il proprio traffico attraverso una connessione SSH. Il meccanismo viene chiamato inoltro di porta perché quando i messaggi giungono alla ben nota porta SSH del server, per prima cosa SSH decifra i contenuti, poi "inoltra" i dati alla vera porta sulla quale il server si trova in ascolto.

8.3.3 Sicurezza nello strato di trasporto (TLS, SSL, HTTPS)

Per capire gli obiettivi di progetto e i requisiti dello standard TLS (Transport Layer Security) che è in corso di sviluppo presso IETF, è utile prendere in esame uno dei problemi principali

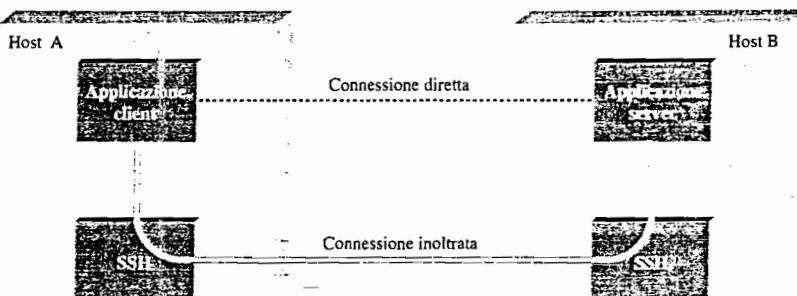


Figura 8.15 Utilizzo dell'inoltro di porta in SSH per mettere in sicurezza altre applicazioni basate su TCP.

che si propone di risolvere. Con il passare del tempo, il World Wide Web è divenuto sempre più popolare e le imprese commerciali hanno iniziato ad interessarsi ad esso, rendendo evidente come per le transazioni commerciali sul Web sia necessario un certo livello di sicurezza. L'esempio canonico di questo è l'acquisto mediante carta di credito, che pone parecchi problemi nel momento in cui è necessario inviare ad un calcolatore sul Web le informazioni relative alla carta di credito stessa. Prima di tutto, vi potreste preoccupare del fatto che le informazioni vengano intercettate durante l'attraversamento della rete e usate in seguito per fare acquisti non autorizzati; ma vi potreste anche preoccupare del fatto che i dettagli della transazione possano essere modificati, ad esempio, per cambiare la quantità di materiale acquistato. Ed è molto probabile che vogliate essere certi che il calcolatore al quale state inviando le informazioni relative alla vostra carta di credito sia effettivamente quello che appartiene al venditore in questione e non a qualche altra entità. Di conseguenza, si nota immediatamente l'esigenza di privacy, integrità e autenticazione nelle transazioni Web. La prima soluzione di questo problema che è stata ampiamente utilizzata era nota con il nome di SSL (Secure Socket Layer) e ha costituito la base dello standard TLS di IETF.

I progettisti di SSL e di TLS si sono resi conto che questi problemi non erano peculiari delle transazioni Web (cioè di quelle che usano HTTP) e misero a punto, invece, un protocollo di utilizzo generale che si situa tra il protocollo applicativo (ad esempio, HTTP) e il protocollo di trasporto (ad esempio, TCP). Il motivo per cui viene chiamato "sicurezza a livello di trasporto" è che, dal punto di vista dell'applicazione, questo strato di protocollo sembra un normale protocollo di trasporto, tranne per il fatto che è sicuro: il mittente può aprire una connessione e consegnarle byte da trasmettere, e lo strato di trasporto sicuro li farà pervenire al destinatario con la necessaria privacy, integrità e autenticazione. Eseguendo lo strato di trasporto sicuro al di sopra di TCP, all'applicazione vengono fornite anche tutte le normali funzionalità di TCP (affidabilità, controllo di flusso, controllo di congestione, ecc.); una configurazione degli strati di protocolli che è raffigurata in Figura 8.16.

Quando il protocollo HTTP viene utilizzato con questa modalità, prende il nome di HTTPS (HTTP sicuro). In realtà, il protocollo HTTP non viene modificato: semplicemente, consegna e riceve i dati attraverso lo strato TLS anziché tramite il protocollo TCP. Per comodità, ad "HTTPS" è stata assegnata una porta TCP (443), cioè, se vi connettete ad un server sulla porta TCP 443, è probabile che vi troverete a colloquiare con il protocollo TLS, che trasmetterà i vostri dati al protocollo HTTP dopo aver verificato che la decifrazione e l'autenticazione siano andate a buon fine.

Una delle differenze interessanti tra un protocollo TLS e uno progettato per la posta elettronica è che esiste la possibilità per una negoziazione in tempo reale. Come abbiamo visto in precedenza, esistono molti algoritmi crittografici che possono essere utilizzati per diverse operazioni e non è ragionevole ipotizzare che la controparte della comunicazione li

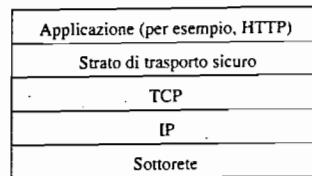


Figura 8.16 Lo strato di trasporto sicuro interposto tra gli strati di applicazione e di trasporto.

implementi tutti: di conseguenza, può darsi che vogliate negoziare finché non trovate una configurazione che vada bene ad entrambi. Potrete anche voler cambiare gli algoritmi nel bel mezzo di una conversazione, se, ad esempio, avete alcuni dati molto importanti, per i quali valga la pena usare una cifratura con un costo computazionale più elevato. Per questo motivo, TLS è suddiviso in due parti:

- un protocollo di *handshake*, usato per negoziare i parametri della comunicazione
- un protocollo "a record", usato per il vero trasferimento di dati

Si può pensare al protocollo di *handshake* come allo strumento per instaurare uno stato condiviso fra le due entità comunicanti che sia sufficiente a consentire il procedere di una comunicazione sicura. Le componenti di questo stato condiviso sono l'insieme degli algoritmi crittografici concordati e i relativi parametri, come chiavi di sessione, vettori di inizializzazione e così via. Questo stato condiviso è rappresentato da un identificativo di sessione (ID) che viene memorizzato sia dal client che dal server, per scopi che verranno discussi in seguito. È interessante notare come il protocollo di handshake possa anche negoziare l'utilizzo di un algoritmo di compressione, non perché questo offra qualche beneficio per la sicurezza, ma perché è facile farlo mentre si stanno negoziando tutte le altre cose e si è già deciso di eseguire alcune costose operazioni su tutti i byte di dati. L'insieme dei messaggi usati durante la fase di handshake è mostrato nella Figura 8.17. Alcuni messaggi vengono inviati soltanto in alcune situazioni, e sono indicati fra parentesi quadre. Notate che la fase di handshake richiede un tempo almeno uguale a due RTT e fino ad una dozzina di messaggi.

Il protocollo di handshake ha anche il compito di consentire lo scambio di certificati fra le due entità coinvolte, se questo è richiesto. Ad esempio, durante un acquisto con carta di credito, il client ha bisogno di sapere che sta colloquiando con il vero server, ma non è necessario che il client si autentichi presso il server. In questo caso, il server dovrebbe fornire al client un certificato, o una catena di certificati, se necessario, come parte dell'iniziale handshake, consegnando così al client una copia affidabile della propria chiave pubblica. Il

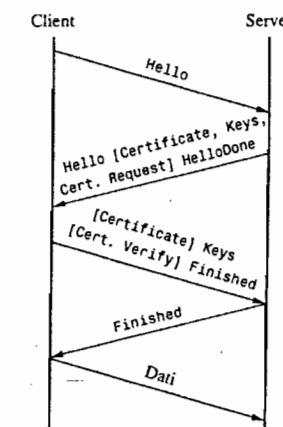


Figura 8.17 Protocollo di *handshake* per instaurare una sessione TLS.

server è poi in grado di autenticare i messaggi successivi firmandoli con la propria chiave privata. Il client può cifrare i messaggi con la chiave pubblica del server e una delle prime cose che farà con tale chiave sarà cifrare e inviare al server un "pre-master secret" (*segreto preliminare*). Successive informazioni confidenziali ("segreti"), come le chiavi di sessione, i vettori di inizializzazione e così via, vengono derivate da questo segreto preliminare.

Il protocollo "a record" definisce un insieme di formati e procedure mediante le quali i messaggi che arrivano dallo strato di applicazione vengono:

- frammentati o accorpati in blocchi di dimensione opportuna per i passi successivi
- eventualmente compressi
- protetti nella loro integrità usando una funzione di hash, come MD5
- cifrati
- consegnati allo strato inferiore (ad esempio, TCP) per la trasmissione

La possibilità di negoziare gli algoritmi crittografici, ancorché utile, espone questo approccio ad una forma di attacco che appartiene alla categoria generale degli attacchi "per interposizione" (*man-in-the-middle attack*). Questa categoria di attacchi presuppone l'esistenza di un intermediario che modifica i messaggi in transito fra due legittimi partecipanti di una comunicazione. Dal momento che la negoziazione iniziale degli algoritmi deve avvenire senza protezione crittografica, un intermediario potrebbe modificare la scelta degli algoritmi in modo che vengano utilizzati algoritmi più deboli di quelli che potrebbero in realtà essere utilizzati dai due partecipanti, algoritmi che, quindi, l'intermediario potrebbe violare più facilmente. Un'applicazione progettata in modo superficiale potrebbe semplicemente accettare qualsiasi algoritmo scelto dal protocollo TLS, senza verificare che sia sufficientemente robusto per i propri scopi. Un'applicazione ben progettata, invece, interromperebbe la transazione, trasformando a tutti gli effetti questo attacco in un attacco di tipo *denial-of-service* (impedimento del servizio), cosa che un intermediario può sempre porre in atto, eliminando semplicemente i pacchetti in transito.

Un'altra interessante caratteristica del protocollo TLS, che è piuttosto utile per le transazioni Web, è la capacità di "ripristinare" una sessione. Per capire il vantaggio di questo comportamento, è utile capire come la versione 1 del protocollo HTTP faccia uso delle connessioni TCP (i dettagli di HTTP sono esposti nella Sezione 9.2.2). Ogni operazione HTTP, come il recupero di una pagina di testo o di un'immagine da un server, richiede l'apertura di una nuova connessione TCP. Recuperare una singola pagina con un certo numero di oggetti grafici al suo interno potrebbe richiedere molte connessioni TCP. Ricordate dalla Sezione 5.2 che l'apertura di una connessione TCP richiede un handshake in tre fasi prima che possa iniziare la trasmissione dei dati; una volta che la connessione TCP sia pronta ad accettare dati, il client dovrebbe iniziare il protocollo di handshake di TLS, per il quale servono almeno altri due RTT (consumando nuovamente risorse di calcolo e ampiezza di banda nella rete) prima che si possano inviare i veri dati dell'applicazione. La capacità di TLS di ripristinare le sessioni può alleviare questo problema.

Il ripristino di una sessione è un'ottimizzazione della fase di handshake che si può usare in quei casi in cui il client e il server abbiano già instaurato nel passato una qualche forma di stato condiviso. Il client inserisce semplicemente nel proprio messaggio iniziale di handshake l'identificativo di sessione di una sessione già stabilita in precedenza: se il server si accorge di avere ancora memorizzato lo stato di tale sessione e nel momento in cui tale sessione era stata instaurata si era negoziata l'opzione di ripristino, allora il server può rispondere al client

con un risultato che indichi il successo e la trasmissione dei dati può iniziare usando gli algoritmi e i relativi parametri negoziati in precedenza. Se l'identificativo di sessione non corrisponde ad alcuno stato di sessione memorizzato nel server oppure se il ripristino della sessione non era consentito per tale sessione, allora il server ricadrà nel normale procedimento di handshake.

Diversamente da PGP, TLS non specifica alcuna particolare infrastruttura per le chiavi. In pratica, TLS ha avuto molto successo, in quanto ha consentito il commercio nel Web usando un'unica autorità di certificazione, la cui chiave pubblica è inserita all'interno dei più diffusi programmi di navigazione Web; le aziende che vogliono accettare pagamenti con carta di credito nei propri siti Web devono ottenere un certificato da tale CA. Ciò rende molto semplice l'autenticazione del server e consente al server stesso di rendere la propria chiave pubblica nota in modo affidabile a qualsiasi client che conceda fiducia alla CA. Come menzionato in precedenza, la chiave pubblica del server è tutto ciò che serve al client per disporre di tutti i "segreti" necessari per cifrare i dati da e per il server.

8.3.4 Sicurezza per il protocollo IP (IPSEC)

Come è facile capire, lo sforzo più ambizioso per integrare caratteristiche di sicurezza in Internet avviene al livello più basso, quello del protocollo IP. Tale architettura prende il nome di IPSEC (IP Security) e in realtà, diversamente da sistemi o singoli protocolli per la sicurezza, è una cornice (*framework*) che consente di fornire tutti i servizi relativi alla sicurezza di cui abbiamo parlato in questo capitolo. Innanzitutto, è altamente modulare, consentendo agli utenti (o, più probabilmente, ai gestori di sistemi) di effettuare le proprie scelte all'interno di un vasto insieme di algoritmi di cifratura e di protocolli specializzati per la sicurezza. Secondariamente, IPSEC consente agli utenti di scegliere all'interno di un ampio ventaglio di servizi di sicurezza, tra i quali troviamo il controllo d'accesso, l'integrità, l'autenticazione, la protezione dalla replica e la privacy. Ancora, IPSEC consente agli utenti di controllare il livello di dettaglio con cui vengono applicati i servizi di sicurezza. Ad esempio, l'architettura IPSEC può essere utilizzata per proteggere sia flussi "focalizzati" (ad esempio, pacchetti che appartengano ad una particolare connessione TCP che vengono scambiati fra una coppia di host) sia flussi "ampi" (ad esempio, tutti i pacchetti in transito tra due gateway).

In una visione di alto livello, IPSEC è composta da due parti. La prima parte è costituita da due protocolli che implementano i servizi di sicurezza disponibili: si tratta di AH (Authentication Header), che fornisce i servizi di controllo di accesso, integrità di messaggi senza connessione, autenticazione e protezione dalla replica, e di ESP (Encapsulating Security Payload), che si occupa degli stessi servizi, con l'aggiunta della privacy. Questi due protocolli si possono usare da soli o insieme, fornendo all'utente la combinazione esatta di servizi richiesta. La seconda parte si occupa della gestione delle chiavi, che trova posto in un protocollo più ampio denominato ISAKMP: Internet Security Association and Key Management Protocol.

L'astrazione che collega queste due parti fra loro è l'*associazione di sicurezza* (SA, security association). Una SA è una "connessione" *simplex* (cioè in un'unica direzione) che viene protetta da uno o più dei servizi di sicurezza disponibili. Le associazioni di sicurezza si possono instaurare fra una coppia di host, fra un host e un gateway di sicurezza (un router che esegue IPSEC), oppure ancora fra due gateway di questo tipo. Ad esempio, si potrebbe instaurare una SA per garantire l'integrità di ogni pacchetto inviato da un gateway di sicurezza ad un altro: tali pacchetti vengono in effetti inviati da un gateway all'altro attraverso un

tunnel (ricordate la presentazione del tunneling nella Sezione 4.1.8). Per mettere in sicurezza una comunicazione bidirezionale fra una coppia di host (corrispondente, ad esempio, ad una connessione TCP), servono due associazioni di sicurezza, una in ciascuna direzione.

Dal punto di vista dell'host locale, una data SA contiene tutte le informazioni necessarie per eseguire i servizi di sicurezza di AH e di ESP. Quando viene creata, ad una SA viene assegnato, dal calcolatore ricevente, un *indice dei parametri di sicurezza* (SPI, security parameters index): la combinazione di questo SPI e dell'indirizzo IP di destinazione identifica univocamente un'associazione di sicurezza. Sia AH che ESP inseriscono il valore di SPI nella propria intestazione; l'host ricevente usa, quindi, questa informazione per determinare a quale SA appartenga un pacchetto in arrivo e, di conseguenza, quali algoritmi applicare al pacchetto stesso. Il ruolo di ISAKMP consiste nella definizione di procedure e formati di pacchetto per instaurare, negoziare, modificare ed eliminare associazioni di sicurezza. Definisce anche formati di pacchetto per lo scambio di dati relativi alla generazione di chiavi e all'autenticazione, ma questi formati non sono molto interessanti, perché stabiliscono soltanto una cornice entro la quale devono rientrare i veri formati, la cui forma esatta dipende dalla tecnica usata per la generazione delle chiavi, dall'algoritmo di cifratura e dal meccanismo di autenticazione utilizzato. Inoltre, ISAKMP non specifica un particolare protocollo per lo scambio delle chiavi, anche se suggerisce IKE (Internet Key Exchange) come uno di quelli possibili.

Authentication Header (AH)

Il protocollo Authentication Header fornisce ai datagrammi IP il servizio di integrità e di autenticazione dell'origine dei dati in un ambito privo di connessione; fornisce, inoltre, una protezione opzionale contro la replica di trasmissione. L'intestazione di AH è mostrata in Figura 8.18 e viene posta di seguito all'intestazione IPv4 oppure è un'intestazione di estensione per IPv6, in relazione a quale sia la versione di IP con cui viene utilizzata.

Il campo *NextHdr* identifica il tipo di carico utile che seguirà l'intestazione. Il campo *PayloadLength* specifica la lunghezza di AH in termini di parole di 32 bit (in unità di 4 byte), a cui vengono sottratte due parole⁵. Il campo *Reserved* è riservato a sviluppi futuri e per ora viene impostato al valore 0. Il campo *SPI* è un qualsiasi valore di 32 bit che, in combinazione con l'indirizzo IP del destinatario, identifica univocamente l'associazione di sicurezza per questo datagramma.

Il campo *SeqNum* contiene un contatore sempre crescente, o numero di sequenza, e viene usato per fornire protezione da repliche di trasmissione, ma è presente anche se il destinatario ha scelto di non abilitare il servizio anti-replica per una specifica SA. Quando si instaura una SA, il contatore del mittente e il contatore del destinatario vengono inizializzati a 0. Se l'anti-replica è abilitata, come avviene se non vi è negoziazione contraria, il numero di sequenza trasmesso non può mai ripetersi; di conseguenza, il contatore del mittente e quello del destinatario devono essere riportati a zero, instaurando una nuova SA (e, quindi, una nuova chiave), prima che in una certa SA venga trasmesso il pacchetto di indice 2^{32} .

⁵ Tutte le intestazioni di estensione di IPv6 codificano il campo "Hdr Ext Len" sottraendo per prima cosa 1 (cioè una parola di 64 bit) dalla lunghezza dell'intestazione (misurata in parole di 64 bit). AH è un'intestazione di estensione di IPv6, ma, dal momento che la sua lunghezza viene misurata in parole di 32 bit, la lunghezza del carico utile viene calcolata sottraendo due parole (di 32 bit).

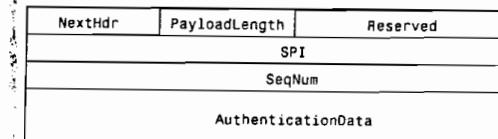


Figura 8.18 Authentication Header di IPSEC.

Infine, *AuthenticationData* è un campo di lunghezza variabile che contiene il codice di integrità del messaggio per quel pacchetto. Il campo deve avere una lunghezza che sia un multiplo intero di 32 bit. AH non specifica nessun particolare algoritmo per calcolare il *message digest* e, fra gli altri, si possono usare DES e MD5: l'unico requisito è che l'algoritmo deve specificare la lunghezza del MIC e le regole di confronto, nonché i passi di elaborazione necessari per la validazione.

Encapsulating Security Payload (ESP)

L'intestazione del protocollo Encapsulation Security Payload è stata progettata per fornire una combinazione di servizi di sicurezza in IPv4 e IPv6. ESP può essere usato da solo oppure insieme a AH. L'intestazione di ESP viene inserita dopo l'intestazione IP e prima dell'intestazione del protocollo di livello superiore (quando viene usata fra due host), o prima di un'intestazione IP incapsulata quando viene usata per realizzare un tunnel fra due gateway di sicurezza.

ESP mette a disposizione i servizi di privacy, di autenticazione dell'origine dei dati, di integrità in ambiente privo di connessione e di anti-replica. L'insieme dei servizi forniti dipende da opzioni selezionate nel momento in cui viene instaurata la SA. La privacy può essere selezionata indipendentemente da tutti gli altri servizi, ma ci si aspetta che sia fornita insieme ad un servizio di integrità e di autenticazione, realizzato con ESP o, separatamente, con AH. L'autenticazione dell'origine dei dati e l'integrità in ambiente privo di connessione sono servizi interconnessi e vengono offerti come opzione insieme alla (opzionale) privacy. Il servizio anti-replica può essere selezionato soltanto se è stata selezionata anche l'autenticazione dell'origine dei dati e la scelta in merito al suo utilizzo è demandata esclusivamente al destinatario. Notate che, sebbene sia la privacy sia l'autenticazione siano opzionali, almeno una delle due deve essere selezionata.

Come in AH, anche l'intestazione di ESP segue l'intestazione di IPv4 oppure costituisce un'intestazione di estensione in IPv6; il suo formato è mostrato in Figura 8.19. Il campo *SPI* ha la stessa funzione di quello visto in AH: aiuta il destinatario a identificare l'associazione di sicurezza a cui appartiene il pacchetto. Analogamente, il campo *SeqNum* protegge dagli attacchi di replica di trasmissione. Il campo *PayloadData* del pacchetto contiene i dati descritti nel campo *NextHdr*. Se è stata selezionata la privacy, i dati vengono cifrati con l'algoritmo di cifratura associato alla SA. A volte si rende necessario inserire zeri (*padding*) per raggiungere una dimensione prefissata, ad esempio perché l'algoritmo di cifratura richiede che il testo in chiaro abbia una dimensione multipla di un certo numero di byte, o per garantire che il testo cifrato risultante abbia una dimensione multipla di 4 byte: il campo *PadLength* memorizza quanti di questi dati sono stati aggiunti ai dati veri. Infine, il campo *AuthenticationData* veicola il MIC, esattamente come in AH: è un campo presente perché ESP è sufficientemente generale da fornire supporto all'integrità dei messaggi e all'autenticazione, oltre che alla privacy.

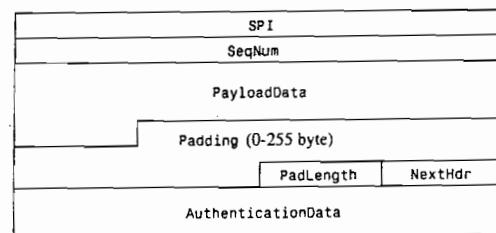


Figura 8.19 L'intestazione di ESP in IPSEC.

8.4 Firewall

Concludiamo la nostra discussione sulla sicurezza delle reti discutendo di un argomento che la maggior parte dei puristi considera un abominio: i *firewall* (letteralmente, *muro di fuoco*, nel senso di *impenetrabile*). Un firewall è un router programmato in modo speciale, che viene posto fra un sito e il resto della rete, come illustrato in Figura 8.20. È un router nel senso che è connesso a due o più reti fisiche e inoltra pacchetti da una rete ad un'altra, ma esegue anche un'operazione di filtraggio dei pacchetti che lo attraversano. Ad esempio, potrebbe eliminare (invece di inoltrare) tutti i pacchetti in arrivo che siano indirizzati verso un particolare indirizzo IP o verso un certo numero di porta TCP, una cosa utile se non volete che utenti esterni possano accedere ad un particolare host o ad un particolare servizio interno al vostro sito. Il firewall potrebbe anche filtrare i pacchetti in base all'indirizzo IP del mittente, una cosa utile per proteggere gli host del vostro sito da una indesiderata inondazione di pacchetti proveniente da un host esterno, inondazione che viene a volte chiamata "attacco di tipo *impedimento di servizio*" (*denial-of-service attack*).

Se tutti i meccanismi di sicurezza descritti in questo capitolo venissero diffusamente attuati, non ci sarebbe bisogno di firewall: quando cercate di stabilire una connessione verso un nodo usando, ad esempio, IPSEC, dovete autenticarvi come pari entità valida. Allora, come mai i firewall sono così utilizzati? Per due motivi. Il primo è che i meccanismi di sicurezza descritti in questo capitolo non sono ancora molto diffusi. Progettare bene algoritmi e protocolli di sicurezza è un compito molto difficile, per cui i firewall sono stati visti come una misura temporanea, in attesa di IPSEC. Anche nel lungo termine, almeno fino a quando ogni singolo sistema eseguirà IPSEC o un altro meccanismo di sicurezza end-to-end simile, sembra probabile che continueremo a dipendere dai firewall. Il secondo motivo è più di principio. Un firewall consente ai gestori di sistema di realizzare una politica di sicurezza in una sede centralizzata, mentre, al contrario, i meccanismi di sicurezza di tipo end-to-end

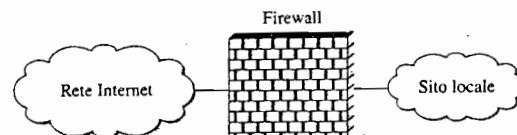


Figura 8.20 Un firewall filtra i pacchetti che fluiscono fra un sito e il resto di Internet.

richiedono che la politica di gestione della sicurezza sia distribuita, con una possibile diversa configurazione di sicurezza in ciascun nodo.

Detto ciò, i firewall sono dispositivi concettualmente molto semplici, che possono essere classificati in una di due ampie categorie: *basati su filtri* e *basati su proxy*. Descriveremo ora brevemente entrambe le categorie e, per limitare la discussione, ci limiteremo ai firewall per Internet.

8.4.1 Firewall basati su filtri

I firewall basati su filtri appartengono al tipo di firewall più semplice e più diffusamente installato. Vengono configurati mediante una tabella di indirizzi che caratterizzano i pacchetti che vogliono, o non vogliono, inoltrare. Parlando di "indirizzi" intendiamo qualcosa di più del semplice indirizzo di destinazione IP, anche se quest'ultima è una delle possibilità. In generale, ogni riga della tabella è una quaterna di informazioni: l'indirizzo IP e la porta TCP (o UDP) del mittente e del destinatario.

Ad esempio, un firewall potrebbe essere configurato per filtrare (cioè per non inoltrare) tutti i pacchetti che soddisfano la seguente condizione:

⟨ 192.12.13.14, 1234, 128.7.6.5, 80 ⟩

Questa maschera dice di filtrare tutti i pacchetti che provengono dalla porta 1234 dell'host 192.12.13.14 e che sono indirizzati alla porta 80 dell'host 128.7.6.5 (la porta 80 è la ben nota porta TCP per HTTP). Ovviamente, spesso non è pratico nominare tutti gli host mittenti di cui si vogliono eliminare i pacchetti, per cui le maschere possono contenere caratteri jolly (*wildcard*). Ad esempio

⟨ *, *, 128.7.6.5, 80 ⟩

impone di filtrare tutti i pacchetti indirizzati alla porta 80 dell'host 128.7.6.5, indipendentemente dall'host mittente e dalla porta da cui è stato spedito il pacchetto. Notate che schemi di indirizzi come questi richiedono che il firewall prenda decisioni di eliminazione/inoltro basandosi anche su numeri di porta di livello 4, oltre agli indirizzi di host di livello 3: per questo motivo i firewall basati su filtri vengono anche spesso chiamati *switch di livello 4*.

Esistono due importanti varianti di questa semplice idea appena esposta. Per prima cosa, c'è il problema di decidere se il firewall può inoltrare tutto ciò che non sia stato esplicitamente inserito in una maschera di filtraggio (come abbiamo ipotizzato nell'esempio precedente) oppure se deve filtrare (cioè eliminare) tutti i pacchetti per i quali non abbia ricevuto esplicite istruzioni di inoltrare. Si tratta di una domanda fondamentale da porsi nel progetto di qualsiasi sistema sicuro: occorre identificare esplicitamente ciò che è ammesso, oppure ciò che non è ammesso? Ad esempio, invece di bloccare l'accesso alla porta 80 dell'host 128.7.6.5, il firewall potrebbe aver ricevuto istruzioni per consentire solamente l'accesso alla porta 25 (la porta del servizio di posta SMTP) di un particolare server di posta elettronica, ad esempio

⟨ *, *, 128.19.20.21, 25 ⟩

ma di bloccare tutto il traffico diverso.

Il secondo problema consiste nel decidere se i filtri vadano specificati nel momento in cui il sistema viene acceso, oppure se si possano installare nuovi filtri mentre il firewall è in

funzione. Gli esempi presentati finora possono essere tutti noti con anticipo: il motivo per cui potreste aver bisogno della seconda possibilità è che stiate utilizzando un firewall configurato in modalità “*drop by default*” (cioè elimina in mancanza di istruzioni) e non sappiate, fino all'ultimo momento, quale porta verrà utilizzata per una particolare connessione valida. Questo accade, ad esempio, con FTP, che instaura una nuova connessione TCP per ogni file che viene trasferito; le due porte usate ai due capi di tale connessione non sono note finché il trasferimento non sta per iniziare, per cui l'insieme di maschere di accesso consentito deve includere, dinamicamente e temporaneamente, tali porte. Si dice che un firewall che abbia questa caratteristica consente la *selezione dinamica delle porte* (*dynamic port selection*).

8.4.2 Firewall basati su proxy

Un *proxy* (“delegato”) è una tecnica di rete generale che compare in diverse situazioni, tra cui i firewall. In generale, un proxy è un processo interposto tra un processo client e un processo server. Per il client, il proxy svolge le funzioni del server e, in un certo senso, prende proprio il posto del server. Per il server, il proxy appare come se fosse il client. Dato che un proxy imita sia il client sia il server, è necessario che contenga al proprio interno tutti i concetti relativi all'applicazione che deve emulare. Una delle cose che potrebbe essere implementata da un proxy è una forma di *cache*: questo consente al proxy di rispondere alle richieste del client senza doverle trasmettere al server, tranne in quei casi in cui nella cache non sia presente l'oggetto richiesto. I proxy sono anche un'opportunità per implementare una politica di sicurezza, ed è proprio questo ruolo che prenderemo in esame ora.

Per capire come funziona un firewall basato su proxy, e anche per quale motivo potremmo volerne implementare uno, considerate un server Web aziendale, tramite il quale l'azienda voglia rendere accessibili ad utenti esterni alcune pagine (e, quindi, non voglia semplicemente programmare il firewall in modo che blocchi tutti gli accessi esterni alla ben nota porta 80 di HTTP), ma voglia limitare l'accesso ad alcune altre pagine ai soli utenti dell'azienda che si trovano in uno o più siti remoti: una situazione rappresentata nella Figura 8.21. Non c'è modo di esprimere questa politica come un filtro, poiché dipende dallo URL contenuto in ogni richiesta HTTP⁶.

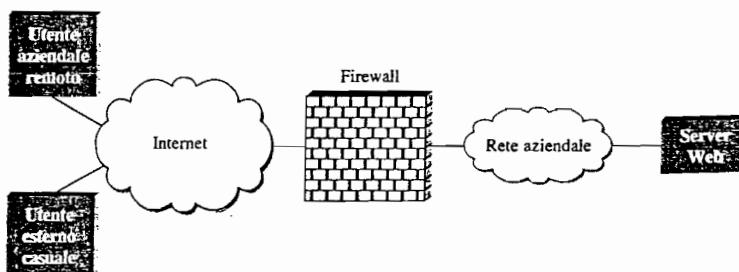


Figura 8.21 Un firewall che protegge il server Web di un'azienda da alcuni accessi esterni.

⁶ Se non sapete bene come funziona il Web, potrebbe valer la pena di leggere prima la Sezione 9.2.2.

La soluzione consiste nell'inserire nel firewall un *proxy HTTP*. Gli utenti remoti instaurano una connessione HTTP/TCP con il proxy, che esamina lo URL contenuto nel messaggio di richiesta. Se la pagina richiesta è accessibile per l'host sorgente, il proxy instaura una seconda connessione HTTP/TCP con il server e inoltra la richiesta al server stesso. Successivamente, il proxy inoltra la risposta nella direzione opposta fra le due connessioni TCP: una situazione rappresentata in Figura 8.22. Se la richiesta non è ammessa, il proxy non crea la seconda connessione e, invece, restituisce un errore al mittente. In un certo senso, il firewall decide dinamicamente quali pacchetti inoltrare e quali pacchetti eliminare, secondo una politica insita nel proxy specifico per l'applicazione.

In questo esempio sono presenti molte cose interessanti da notare. Innanzitutto, il proxy deve poter comprendere il protocollo HTTP, per poter rispondere al client. Secondariamente, una volta installato un proxy HTTP per motivi di sicurezza, si potrebbe estendere la sua funzionalità per decidere verso quale server inoltrare la richiesta, scegliendo fra molti server Web locali, magari nel tentativo di bilanciare il carico fra i server stessi: potrebbe, inoltre, fungere da cache per le pagine Web più richieste, come già detto. Come terza possibile estensione, si possono definire proxy per applicazioni diverse da HTTP: ad esempio, proxy FTP e Telnet sono abbastanza comuni.

Infine, i firewall basati su proxy possono essere caratterizzati dal fatto di essere *trasparenti* oppure *classici*. Un proxy trasparente, come suggerisce il nome, non è visibile esplicitamente né per il mittente né per il destinatario: intercetta semplicemente i messaggi che lo attraversano. Al contrario, un proxy classico riceve messaggi ad esso esplicitamente indirizzati dal mittente, per poi inoltrarli alla loro destinazione finale. Considerate la semplice rete di Figura 8.23, nella quale la sorgente S invia un messaggio al ricevitore R tramite il proxy P. Se P è trasparente, allora S indirizza il messaggio a R, anche se il messaggio attraversa P lungo il suo percorso verso R: P lo inoltra verso R oppure no. Con un firewall classico, S non conosce R, ma indirizza il messaggio a P: in altre parole, P agisce come una porta d'ingresso indirizzabile del sito. Quando il messaggio arriva a P, questo seleziona un nodo “nel retro” a cui inoltrare il messaggio.

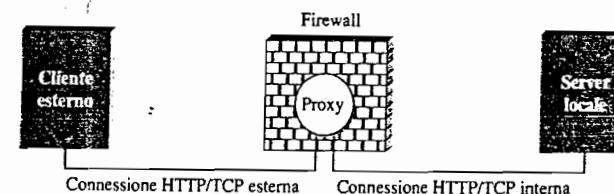


Figura 8.22 Un proxy HTTP che funge da intermediario per l'accesso ad un server Web aziendale.



Figura 8.23 Una semplice internetwork, dove la sorgente S invia un messaggio al ricevitore R attraverso il proxy P.

8.4.3 Limitazioni

Terminiamo questa discussione osservando che, mentre un firewall tradizionale protegge gli utenti interni dagli utenti esterni, non fa niente per proteggere o isolare gli utenti interni fra loro. Anche se pressioni sociali e legali possono fare in modo che gli utenti non violino le politiche di sicurezza locali, queste pressioni sono totalmente inefficaci nei confronti di codice mobile importato dagli utenti nel proprio sito. In particolare, i firewall tradizionali non possono tenere lontano dalla rete locale il codice mobile (ad esempio, i programmi potrebbero entrare sotto forma di messaggi di posta elettronica apparentemente innocenti) e, una volta eseguiti localmente, tali programmi avrebbero un accesso virtualmente illimitato a tutti gli host locali.

Il codice mobile non è l'unico problema. Ad esempio, spesso grandi aziende desiderano isolare parti dell'infrastruttura di calcolo aziendale, cosa che potrebbe accadere se l'azienda avesse un collaboratore esterno che lavora ad un progetto (e, quindi, deve avere accesso ad un sottoinsieme di calcolatori), ma volesse impedire l'accesso all'intera infrastruttura da parte di tale collaboratore. Come ulteriore esempio, considerate il caso dei calcolatori portatili, che rendono semplice per gli utenti la connessione anonima dei propri calcolatori in punti arbitrari della rete.

Un altro punto vulnerabile dei firewall è la crescente attenzione per le comunicazioni wireless, che stanno diffondendosi enormemente. Considerate un lavoratore pendolare che si connette ad un sito aziendale attraverso un firewall: il firewall è stato programmato per consentire al computer dell'utente di inviare pacchetti al sito, poiché l'utente ne ha il permesso. Supponete ora che tale lavoratore abbia una rete wireless che connette più calcolatori all'interno della propria abitazione: non c'è niente che impedisca ad un vicino, o a un concorrente aziendale parcheggiato sotto casa, di diventare parte della "rete domestica" e, quindi, ottenere accesso alla rete aziendale attraverso il firewall.

In generale, è impossibile per i firewall attuali sapere chi sta accedendo alla rete e, quindi, chi possa connettersi ad altri calcolatori della rete. Ciò che serve è la possibilità di spostare dalla periferia all'interno della rete la protezione d'accesso, più vicina agli host che tentano l'accesso stesso. Alla fine, probabilmente sarà necessario che meccanismi di sicurezza come IPSEC forniscano il supporto per un livello di sicurezza di questo tipo.

8.5 Riepilogo

Gli algoritmi di cifratura sono la spina dorsale della sicurezza delle reti. Gli algoritmi a chiave pubblica, come RSA, vengono spesso usati dai protocolli di autenticazione per determinare l'identità dei due processi comunicanti. Una volta autenticati, i due partecipanti alla comunicazione posso concordare su una chiave condivisa che viene utilizzata da algoritmi a chiave segreta, come DES, per cifrare qualsiasi messaggio che viene scambiato, garantendo così la privacy della comunicazione. In alternativa, si può usare un algoritmo di generazione di riassunti di messaggi (*message digest*), come MD5, per proteggere l'integrità dei messaggi. L'intero sistema è reso possibile da un meccanismo di distribuzione delle chiavi, usato per venire in possesso della chiave pubblica necessaria da RSA per iniziare il processo.

Alcuni sistemi recenti usano questi algoritmi e protocolli per offrire servizi di sicurezza in Internet. Al livello di applicazione, si può usare PGP per proteggere i messaggi di posta elettronica e SSH per effettuare connessioni remote in sicurezza. Al livello di trasporto, si può usare TLS per proteggere le transazioni commerciali sul World Wide Web. Al livello di

rete, si può usare l'architettura IPSEC per rendere sicure le comunicazioni fra qualsiasi insieme di host o di gateway in Internet. Mentre questi sistemi stanno divenendo sempre più diffusi, i gestori di reti continuano ad usare i firewall per proteggere il loro sito da pericoli esterni.

Problema aperto Attacchi di tipo "Denial of Service"

Sistemi come IPSEC e TLS sono potenzialmente in grado di dare a Internet il livello di sicurezza richiesto da molte emergenti applicazioni commerciali e da applicazioni governative. La chiave del successo di questi sistemi si basa sul successo del procedimento di distribuzione delle chiavi: senza accesso fiduciario alle chiavi pubbliche, l'intero sistema si blocca. La distribuzione delle chiavi è, comunque, soltanto uno dei problemi relativi alla sicurezza con cui si confrontano le reti di calcolatori. Un problema ugualmente grave è la ricerca di un metodo atto a proteggere i calcolatori connessi ad una rete geografica da attacchi di tipo *impedimento del servizio* (DoS, denial of service). Diversamente dagli attacchi alla privacy delle comunicazioni, dove qualcuno cerca di ottenere l'accesso ad informazioni per le quali non ha la necessaria autorizzazione, un attacco di tipo DoS prevede che qualcuno cerchi di impedire, a coloro che ne hanno legittimo diritto, l'accesso alle informazioni o alle risorse. Un ben noto attacco di tipo DoS, chiamato "attacco SYN", prende il nome dal pacchetto di instaurazione della connessione TCP. In un attacco SYN, un attaccante remoto inonda la vostra macchina di pacchetti SYN, facendole così esaurire tutto il proprio tempo di CPU nel tentativo di instaurare inutili connessioni TCP. La chiave di volta di questo attacco consiste nel fatto che, diversamente da quanto accade inondando una macchina con pacchetti di dati inutili, ogni pacchetto SYN richiede un'elaborazione non banale perché si possa determinare che l'eliminazione del pacchetto sia l'operazione corretta da compiere. I firewall offrono un certo livello di protezione, perché possono essere programmati per eliminare tutti i pacchetti che provengono da un host attaccante noto, ma per l'attaccante è molto semplice inserire in ogni pacchetto SYN un diverso indirizzo IP per il mittente, dato che non ha bisogno di ricevere una risposta.

Un altro ben noto attacco DoS consiste nell'invio ad un router di un flusso di "pacchetti ad albero di Natale", pacchetti aventi tutte le "luci" accese (cioè tutte le opzioni IP abilitate). Il router impegna una frazione così significativa del proprio tempo per elaborare tutte queste opzioni che non è più in grado di elaborare gli aggiornamenti BGP. Un esempio meno noto mette in evidenza quanto possa essere subdolo un attacco di tipo DoS. Un attaccante inonda il router di un ISP con pacchetti IP aventi indirizzi IP in sequenza: questa sequenza esaurì la cache di primo livello del router, che alla fine fu costretto a perdere tutto il proprio tempo nella costruzione di nuove tabelle di inoltro, alle spese delle risposte (mancate) del router ai messaggi di "probe" del router vicini, che così pensarono che il router in questione fosse guasto.

La protezione da attacchi di tipo *denial of service* avviene in tre passi. Il primo consiste nel tener conto di tutte le risorse impegnate da ciascun utente (o flusso). Il secondo prevede l'identificazione dei casi in cui le risorse utilizzate da un utente eccedono quelle consentite da una particolare politica del sistema. Una volta che è stato identificato, in questo modo, un attacco, il passo finale consiste nel revocare la concessione delle risorse impegnate, usando la minore quantità possibile di ulteriori risorse (altrimenti la rimozione di un attaccante diventa a sua volta un attacco DoS). Sfortunatamente, oggi ben pochi sistemi, sia host che router, tengono conto con adeguata precisione delle risorse impegnate nel sistema, senza parlare della definizione di una politica che identifichi una situazione di attacco DoS.

In generale, comunque, è difficile identificare la violazione di una politica di utilizzo delle risorse, perché non necessariamente l'attaccante invia un imponente flusso di pacchetti dalla medesima sorgente. Al contrario, l'attaccante vi potrebbe bombardare con flussi di pacchetti, apparentemente innocenti, che provengono da molti mittenti diversi. Questa situazione viene solitamente definita "attacco DDoS" (*distributed denial-of-service attack*) e prevede che l'attaccante per prima cosa prenda possesso di un vasto insieme di calcolatori (denominati *zombie*), per poi rivolgere verso di voi l'attenzione di tutti questi zombie contemporaneamente. Ad esempio, siti ad elevata visibilità, come CNN, Yahoo, eBay e Amazon, furono messi in ginocchio da un attacco DDoS nel mese di febbraio del 2000. Alla fin fine, gli attacchi DDoS sono problematici perché è quasi impossibile fare una distinzione tra un carico pesante, ma legittimo, proveniente da molteplici sorgenti (ad esempio, la coda per i saldi) e un attacco DDoS.

Ulteriori letture

I primi due lavori relativi alla sicurezza, presi insieme, danno una buona panoramica dell'argomento. L'articolo di Lampson *et al.* contiene una trattazione formale della sicurezza, mentre il lavoro di Satyanarayanan presenta un'interessante descrizione di come venga progettato, in pratica, un sistema sicuro. Il terzo lavoro presenta una panoramica sull'architettura di sicurezza IPSEC ed è il punto giusto da cui prendere le mosse per comprendere appieno lo stato della sicurezza in Internet oggi.

- Lampson, B., *et al.* "Authentication in distributed systems: Theory and practice", *ACM Transactions on Computer Systems* 10(4):265-310, November 1992.
- Satyanarayanan, M. "Integrating security in a large distributed system", *ACM Transactions on Computer Systems* 7(3):247-280, August 1989.
- Kent, S., e R. Atkinson "Security architecture for the Internet Protocol", *Request for Comments* 2401, November 1998.

Ci sono diversi buoni libri che coprono tutti gli aspetti della sicurezza delle reti: ci sentiamo di poter raccomandare Schneier [Sch95] e Kaufman *et al.* [KPS02]. Il primo presenta una trattazione completa dell'argomento, con codice d'esempio, mentre l'ultimo è una panoramica di facile lettura. L'intera architettura IPSEC viene definita in una serie di RFC: [KA98a, MG98a, MG98b, MD98, KA98b, Pip98, MSST98, HC98]. Un libro di Barrett e Silverman [BS01] contiene una descrizione completa di SSH.

Un'iscrizione in merito a come si possano identificare attacchi di tipo *denial of service* e difendersi, si può trovare in Moore *et al.* [MVS01], Spatscheck e Peterson [SP99] e Qiexh *et al.* [QPP02]. Nei lavori di Bellovin [Bel00], Savage *et al.* [SWKA00] e Snoeren *et al.* [SPS⁺01] si possono trovare le più recenti tecniche usate per identificare le sorgenti di attacchi. Il crescente timore di attacchi DDoS è discusso da Garber [Gar00] e Harrison [Har00], mentre un lavoro di Park e Lee [PL01] riferisce in merito ad un approccio preliminare per la difesa da tali attacchi.

Infine, raccomandiamo il seguente riferimento attivo:

- <ftp://cert.org/pub>: una raccolta di annunci relativi alla sicurezza emessi dal CERT (Computer Emergency Response Team)

Esercizi

1. Trovate sul vostro sistema un programma di cifratura (ad esempio, il comando `des` di Unix o `pgp`). Studiate la sua documentazione e fate esperimenti con esso. Misurate la velocità di cifratura e di decifrare. Sono uguali? Cercate di confrontare questi risultati, in termini di tempo, usando chiavi di dimensioni diverse; ad esempio, confrontate DES con 3DES.
2. La Sezione 8.1.2 descrive la trasformazione di cifratura DES da $\langle L_{i-1}, R_{i-1} \rangle$ al passo $i - 1$ a $\langle L_i, R_i \rangle$ al passo i . Indicate la trasformazione inversa, cioè esprimete $\langle L_{i-1}, R_{i-1} \rangle$ in funzione di $\langle L_i, R_i \rangle$.
3. Supponete che nella fase i in DES, L_{i-1} sia composto da tutti zeri, R_{i-1} sia `deadbeef` (in esadecimale) e K_i sia `a5bd96 860841`. Esprimete R_i , nell'ipotesi che si usi una S box semplificata che riduce ogni porzione di 6 bit in una porzione di 4 bit eliminando il primo e l'ultimo bit.
- ✓ 4. Eseguite la fase $i + 1$ della cifratura DES, usando il risultato dell'esercizio precedente per assegnare un valore a L_i e a R_i , e assumete che K_{i+1} sia `5af310 7a3fff`. Esprimete R_{i+1} , nell'ipotesi che si usi una S box semplificata che riduce ogni porzione di 6 bit in una porzione di 4 bit eliminando il primo e l'ultimo bit.
5. Supponete nuovamente che DES usi la S box semplificata dei due esercizi precedenti e ipotizzate anche di eseguire una sola fase di cifratura.
 - a) Supponete che un attaccante abbia sia il testo in chiaro $\langle L_0, R_0 \rangle$ sia il testo cifrato $\langle L_1, R_1 \rangle$. Quante informazioni fornisce tutto ciò all'attaccante in merito alla chiave K_1 ? E quali informazioni su K ? (non abbiamo intenzione di suggerire una debolezza di DES, quanto piuttosto di fornire una giustificazione per la forma di S box veramente utilizzata da DES)
 - b) Riuscire a ricostruire la chiave a partire da un testo in chiaro e un testo cifrato sarebbe sufficientemente problematico per qualsiasi strategia di cifratura; spiegate perché ciò sarebbe particolarmente critico per un sistema critografico a chiave pubblica.
6. Supponete di cifrare con RSA, con $p = 101$, $q = 113$ e $e = 3$.
 - a) Calcolate l'esponente di decifrazione d . (Suggerimento: anche se esistono algoritmi per eseguire questo calcolo, nel caso di $e = 3$ è più efficiente procedere per tentativi)
 - b) Cifrate il messaggio $m = 9876$. Notate che la valutazione di m^3 con aritmetica a 32 bit produce un *overflow*.
- ✓ 7. Supponete di cifrare con RSA, con $p = 13$, $q = 7$ e $e = 5$.
 - a) Calcolate l'esponente di decifrazione d . (Suggerimento: usate l'algoritmo di divisione di Euclide)
 - b) Cifrate il messaggio $m = 7$.
 - c) Decifrate il messaggio cifrato $c = 2$.
- ★ 8. Dimostrate che l'algoritmo di decifrazione RSA ripristina il messaggio originario, cioè che $med \equiv m \pmod{pq}$. Suggerimento: potete ipotizzare che sia sufficiente dimostrare la congruenza di $m \pmod{p}$ e di $m \pmod{q}$, dato che p e q sono primi fra loro.
- ★ 9. Se n è un numero primo e $b < n$, allora $b^{n-1} \equiv 1 \pmod{n}$. Esistono pochi valori composti (cioè non primi) di n (ad esempio, 561) per i quali vale questa relazione di congruenza per tutti i valori di $b < n$, ma con qualche sforzo ulteriore si può usare il *test di Miller*, che se n è primo ha successo per qualsiasi $b < n$ e se n è composto fallisce per almeno

- tre quarti dei valori di $b < n$ (e qui servono gli sforzi di gestione). Se tentiamo il test con un gran numero di valori di $b < n$ scelti a caso e non fallisce con nessuno di essi, allora n è "probabilmente" primo.
- Dimostrate che il calcolo di $b^{n-1} \bmod n$ può essere effettuato con $O(\log n)$ moltiplicazioni. Suggerimento: $b^{13} = b^8 b^4 b$.
 - Dimostrate, usando questo metodo, che $n = 50621$ è un numero composto. Usate $b = 2$. Non avrete bisogno di alcuno sforzo ulteriore: semplicemente, dimostrate che $b^{n-1} \not\equiv 1 \bmod n$.
 - Dimostrate che $2^{280} \equiv 1 \bmod 561$ (e, quindi, automaticamente $2^{560} \equiv (2^{280})^2 \equiv 1$), ma che $2^{140} \not\equiv 1 \bmod 561$. Questo ultimo fatto fa fallire il test di Miller, dimostrandone che $561 (= 3 \times 11 \times 17)$ è composto, anche se il più semplice test $b^{n-1} \equiv 1 \bmod n$ ha successo.
10. Nell'handshake di autenticazione in tre fasi di Figura 8.9, perché il server non è certo dell'identità del client finché non ha ricevuto il terzo messaggio? A quale attacco potrebbe essere esposto il server se ritenesse certa l'identità del client prima di ricevere il terzo messaggio?
11. Supponete che i valori x e y usati nell'handshake a tre fasi di Figura 8.9 fossero correlati al tempo piuttosto che casuali; ad esempio, x e y potrebbero venire incrementati una volta al secondo, oppure una volta per ogni connessione.
- Dimostrate che la tecnica usata nell'attacco di *IP spoofing* delineato nell'Esercizio 19 del Capitolo 5 fallisce.
 - Supponete, in aggiunta, che un attaccante possa porsi in ascolto lungo la connessione e conoscere le precedenti trasmissioni del client. Sarebbe un aiuto per l'attaccante?
 - Supponete, ancora, che l'attaccante sia in grado di modificare l'orario sul server, magari usando il Network Time Protocol. Dimostrate come un attaccante sia così in grado di autenticarsi presso il server senza conoscere CHK (sebbene non riesca, comunque, a decifrare SK).
12. La Figura 8.7 mostra la cifratura CBC. Descrivete il diagramma corrispondente per la decifrazione.
13. La Figura 8.11 mostra l'autenticazione in un senso, mediante RSA. Mostrate come si possa usare RSA per l'autenticazione in entrambi i sensi.
14. Studiate uno schema di cifratura con chiave "escrow" (ad esempio, Clipper), che consenta cioè la decifrazione da parte di particolari autorità senza la chiave originariamente necessaria per la decifrazione. Quali sono i vantaggi e gli svantaggi di questi sistemi?
15. Un meccanismo che consente di resistere agli attacchi di "replica" nelle fasi di autenticazione di *password* è l'uso di *one-time password* (*password monouso*): viene predisposto un elenco di *password* e, dopo aver accettato la *password* di indice N , il server decrementa N e la volta successiva richiederà la *password* di indice $N - 1$. Quando N vale 0, serve un nuovo elenco di *password*. Delineate una strategia mediante la quale l'utente e il server devono ricordare soltanto una *master password*, mp , e possono disporre di un metodo per calcolare localmente $\text{password}[N] = f(mp, N)$. Suggerimento: sia g una opportuna funzione non invertibile (ad esempio, MD5) e sia $\text{password}[N] = g^N(mp) = g$, applicata N volte a mp . Spiegate perché la conoscenza di $\text{password}[N]$ non aiuta a individuare $\text{password}[N - 1]$.
16. Supponete che un utente usi *password monouso* come quelle dell'esercizio precedente (oppure anche *password riutilizzabili*), ma che la *password* venga trasmessa "con sufficiente lentezza".

- a) Mostrate che un intercettatore può ottenere l'accesso al server remoto con un numero di tentativi relativamente modesto. Suggerimento: l'intercettatore inizia i propri tentativi dopo che l'utente ha digitato tutti i caratteri della *password* tranne l'ultimo.
- b) A quali altri attacchi sarebbe soggetto un utente di *password monouso*?
17. Il protocollo di scambio delle chiavi di Diffie-Hellman è vulnerabile agli attacchi per interposizione ("man-in-the-middle"). Spiegate come un attaccante posto fra due partecipanti possa far credere loro di aver concordato un segreto condiviso, mentre in realtà hanno entrambi condiviso un segreto con l'attaccante. Indicate come sia possibile estendere il protocollo di Diffie-Hellman per fornire protezione da questa eventualità.
- ★ 18. Supponete di usare RSA per inviare un messaggio m a tre destinatari, che hanno moduli di cifratura primi tra loro, n_1 , n_2 e n_3 . Tutti i tre destinatari usano il medesimo esponente di cifratura, $e = 3$, una scelta un tempo assai diffusa perché rende molto veloce la cifratura. Dimostrate che chi sia in grado di intercettare tutti i tre messaggi cifrati $c_1 = m^3 \bmod n_1$, $c_2 = m^3 \bmod n_2$ e $c_3 = m^3 \bmod n_3$, possa anche decifrare m in modo efficiente. Suggerimento: il *teorema cinese del resto* (*Chinese remainder theorem*) implica che si possa trovare in modo efficiente un valore c tale che $c = c_1 \bmod n_1$, $c = c_2 \bmod n_2$ e $c = c_3 \bmod n_3$. Con questa ipotesi, mostrate che questo implica che $c = m^3 \bmod n_1 n_2 n_3$. Quindi, notate che $m^3 < n_1 n_2 n_3$.
19. Supponete di possedere un segreto molto breve s (ad esempio, un singolo bit oppure un numero della Sicurezza Sociale) e di voler inviare a qualcun altro un messaggio m che ora non rivela s ma che possa essere usato più tardi per verificare che in questo momento eravate a conoscenza di s . Spiegate perché $m = \text{MD5}(s)$ oppure $m = E(s)$ con la cifratura RSA non sarebbero scelte sicure, e suggerite una scelta migliore.
20. Supponete che due persone vogliano giocare a poker in rete. Per "distribuire" le carte c'è bisogno di un meccanismo per scegliere in modo imparziale un numero casuale x condiviso; ognuna delle due parti perderebbe se l'altra parte potesse influenzare in modo non equo la scelta di x . Descrivete un meccanismo di questo tipo. Suggerimento: potete ipotizzare che se due stringhe di bit x_1 e x_2 sono entrambe casuali, allora l'OR esclusivo $x = x_1 \oplus x_2$ è casuale.
21. Stimate le probabilità di trovare due messaggi che abbiano la stessa somma di controllo MD5, all'interno di un numero totale di messaggi uguale a 2^{63} , 2^{64} e 2^{65} . Suggerimento: si tratta, di nuovo, del problema del compleanno, come nell'Esercizio 49 del Capitolo 2, e di nuovo la probabilità che il $(k + 1)$ -esimo messaggio abbia una somma di controllo diversa da quella di ciascuno dei k messaggi precedenti è $1 - k/2^{128}$. Tuttavia, l'approssimazione che era stata indicata in quel suggerimento per semplificare il prodotto fallisce, in questo caso, piuttosto malamente. Di conseguenza, invece di quella approssimazione, calcolate questa volta il logaritmo di entrambi i membri della relazione e usate l'approssimazione: $\log(1 - k/2^{128}) \approx -k/2^{128}$.
22. Supponete di voler cifrare una sessione Telnet con DES, ad esempio. Telnet invia pacchetti messaggi di un solo byte, mentre DES cifra a blocchi di 8 byte per volta. Spiegate come si possa usare DES in modo sicuro in questo contesto.
23. Considerate il seguente esempio di protocollo UDP (vagamente basato su TFTP, *Request for Comments* 1350) per scaricare file:
- Il client invia la richiesta di un file.
 - Il server risponde con il primo pacchetto di dati.
 - Il client invia una conferma (ACK), poi i due procedono usando stop-and-wait.

- Supponete che client e server siano in possesso, rispettivamente, delle chiavi K_C e K_S , e che queste chiavi siano note anche alla controparte.
- Estendete il protocollo per lo scaricamento di file, usando queste chiavi e MD5, in modo da fornire i servizi di autenticazione del mittente e di integrità del messaggio. Il protocollo dovrebbe essere resistente agli attacchi di replica.
 - Come può l'informazione aggiuntiva presente nel vostro protocollo migliorato fornire protezione contro l'arrivo di pacchetti ritardatari di precedenti incarnazioni della stessa connessione, e contro il ritorno al valore iniziale del numero di sequenza?
 - Usando il vostro navigatore Web preferito, scoprite quali autorità di certificazione per HTTPS vi sono configurate, in mancanza di impostazioni particolari da parte dell'utente. Avete fiducia di queste agenzie? Scoprite cosa accade se disabilitate la fiducia assegnata ad una o a tutte queste autorità di certificazione.
 - Supponete di voler fare in modo che il vostro firewall basato su filtri blocchi tutte le connessioni Telnet entranti, ma consenta le connessioni Telnet uscenti. Un approccio potrebbe essere quello di bloccare tutti i pacchetti entranti che siano destinati alla porta prevista per il servizio Telnet (23).
 - Potremmo voler bloccare i pacchetti entranti destinati anche ad altre porte, ma quali connessioni TCP entranti *devono* essere consentite per non interferire con le connessioni Telnet uscenti?
 - Supponete ora che il vostro firewall possa usare i bit del campo di intestazione TCP Flags, oltre ai numeri di porta. Spiegate come si possa ottenere l'effetto desiderato sul servizio Telnet, pur senza consentire alcuna connessione TCP entrante.
 - Supponete che un firewall sia configurato per consentire tutte le connessioni TCP uscenti, ma le connessioni entranti soltanto se dirette ad alcune porte specifiche. Il protocollo FTP presenta ora qualche problema: quando un client interno contatta un server esterno, la connessione di controllo uscente può essere aperta normalmente, ma la connessione per i dati è solitamente una connessione TCP entrante.
 - Esaminate il protocollo FTP leggendo, per esempio, *Request for Comments 959*. Scoprite come funziona il comando PORT. Descrivete come si possa scrivere il programma client in modo che si limiti il numero di porte alle quali il firewall deve concedere l'accesso entrante. Si può limitare a uno il numero di tali porte?
 - Scoprite come si possa usare il comando PASV di FTP per risolvere questo problema dei firewall.
 - Supponete che dei router con filtro siano disposti come in Figura 8.24; il firewall principale è R1. Spiegate come vanno configurati R1 e R2 perché dall'esterno si possa effettuare una connessione Telnet verso la rete 2 ma non verso host della rete 1. Per evitare che si possa "saltare" all'interno della rete 1, disabilitate anche le connessioni Telnet dalla rete 2 alla rete 1.
 - Perché un fornitore di servizi Internet (ISP) potrebbe voler bloccare una parte del traffico uscente?

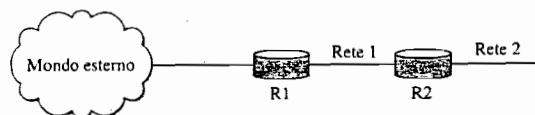


Figura 8.24 Schema per l'Esercizio 27.

- Confrontate un firewall basato su filtri con un firewall basato su proxy in termini di protezione contro attacchi di tipo *spoofing*, descritti nell'Esercizio 19 del Capitolo 5. Ipotizzate la configurazione di Figura 8.21, dove gli host interni, che subiscono l'attacco, ritengono degno di fiducia l'utente dell'azienda remota.
- Si dice che IPSEC possa non funzionare con il protocollo NAT (Network Address Translation, RFC 1631), tuttavia, il fatto che IPSEC funzioni con NAT, oppure no, dipende dalla modalità usata da IPSEC e da NAT. Supponete di utilizzare il vero NAT, nel quale vengono tradotti solamente gli indirizzi IP (senza traduzione di porte). Nei casi seguenti, IPSEC e NAT funzioneranno oppure no? Fornite spiegazioni per tutti i casi.
 - IPSEC usa la modalità di trasporto AH.
 - IPSEC usa la modalità tunnel di AH
 - IPSEC usa la modalità di trasporto di ESP
 - IPSEC usa la modalità tunnel di ESP
 - Cosa succede se usiamo il protocollo PAT (Port Address Translation), che in ambito NAT è anche noto come NAPT (Network Address/Port Translation), nel quale, oltre agli indirizzi IP, vengono tradotti anche i numeri di porta, per condividere un unico indirizzo IP al di fuori della rete privata?

Applicazioni

Problema

Ogni applicazione ha bisogno del proprio protocollo

Abbiamo aperto questo libro parlando di programmi applicativi che le persone vogliono utilizzare sulle reti di calcolatori, dai browser Web agli strumenti di videoconferenza. Nei capitoli successivi abbiamo sviluppato, uno strato per volta, l'infrastruttura di rete necessaria a rendere possibile il funzionamento di queste applicazioni, e ora chiudiamo il cerchio, tornando alle applicazioni di rete, che sono in parte protocolli di rete (nel senso che scambiano messaggi con le pari entità che si trovano su altri calcolatori) e in parte programmi applicativi tradizionali (nel senso che interagiscono con il sistema di gestione a finestre, con il file system e, non ultimo, con l'utente). Questo capitolo prende in esame alcune delle applicazioni di rete oggi più popolari, ponendo l'attenzione ai relativi protocolli. Ciò che vedrete subito è che un protocollo è un protocollo, indipendentemente dallo strato in cui opera: detto in altro modo, il modo migliore per prepararvi alla progettazione di applicazioni di rete è capire, prima di tutto, come si progettano buoni protocolli di rete.

Il primo esempio che prendiamo in esame, un servizio distribuito per i nomi, è anche la prima applicazione che è stata realizzata in una rete. Sebbene si tratti, tecnicamente, di un'applicazione di rete, essendo, a tutti gli effetti, una base di dati distribuita che si basa sui protocolli di trasporto sottostanti, non è un'applicazione che, di norma, gli utenti invocano esplicitamente. Nonostante questo, è un'applicazione da cui dipendono tutte le altre applicazioni, perché un server per i nomi viene usato per tradurre nomi di host in indirizzi di host: l'esistenza di una tale applicazione consente agli utenti delle altre applicazioni di fare riferimento ad host remoti per nome anziché per indirizzo. In altre parole, un server per i nomi viene solitamente utilizzato da altre applicazioni, piuttosto che da persone umane.

Successivamente, passeremo a descrivere un certo numero di applicazioni di rete, dalle più popolari e diffuse alle meno comuni, dallo scambio di messaggi di posta elettronica alla navigazione del Web, dalla gestione di un insieme di elementi di rete ad applicazioni multimediali

come vic e vat, alle tecniche emergenti, come le reti *peer-to-peer* e le reti per la distribuzione di contenuti. Questo elenco non è per nulla esaustivo, ma serve ad illustrare i trucchi della progettazione di protocolli al livello di applicazione, che hanno l'obiettivo di integrare i servizi di trasporto sottostanti (TCP e UDP) in modo che vengano forniti alle applicazioni i servizi di comunicazione che richiedono.

9.1 Servizio per i nomi (DNS)

Fino a questo punto abbiamo identificato gli host mediante i loro indirizzi, un metodo perfettamente adatto all'elaborazione svolta dai router, ma non molto vicino agli utenti umani: proprio per questa ragione ad ogni host di una rete viene tipicamente assegnato un *nome* univoco. Questa sezione descrive come si possa sviluppare un servizio per i nomi (*name service*), che metta in corrispondenza nomi comodi per gli utenti umani con indirizzi comodi per i router. Spesso un servizio di questo tipo è la prima applicazione che viene implementata in una rete, perché elimina la necessità che le altre applicazioni identifichino gli host per mezzo dei loro indirizzi, consentendo l'uso di nomi. A volte i servizi per i nomi vengono chiamati *middleware* ("strumenti intermedi"), perché trovano la loro collocazione fra le applicazioni e la rete sottostante.

I nomi degli host sono diversi dagli indirizzi per due aspetti fondamentali. Innanzitutto, sono solitamente di lunghezza variabile e facilmente memorizzabili per le persone (al contrario, gli indirizzi numerici di lunghezza fissa sono più di più semplice elaborazione per i router). Secondariamente, solitamente i nomi non contengono informazioni che aiutino la rete a localizzare gli host (cioè ad inoltrare pacchetti nella direzione giusta): al contrario, gli indirizzi contengono a volte al proprio interno informazioni di instradamento, con l'eccezione degli indirizzi non decomponibili in più parti (*flat*, non gerarchici).

Prima di entrare nel dettaglio di come si assegnino i nomi agli host di una rete, presentiamo un po' di terminologia di base. Innanzitutto, uno *spazio per i nomi* (*name space*) definisce l'insieme dei nomi possibili. Uno spazio per i nomi può essere *gerarchico* (e i nomi dei file nei sistemi Unix ne sono un classico esempio) oppure *non gerarchico* (*flat*, uno spazio nel quale i nomi non sono decomponibili in più parti). Secondariamente, il sistema dei nomi gestisce un insieme di *corrispondenze* (*binding*) fra nomi e valori: il valore può essere qualsiasi dato che vogliamo venga restituito dal sistema dei nomi quando viene fornito un nome, ma in molti casi è un indirizzo. Infine, un *meccanismo di risoluzione* è una procedura che, quando viene invocata con un nome, restituisce il valore corrispondente. Un *server per i nomi* (*name server*) è una specifica implementazione di un meccanismo di risoluzione che sia disponibile in una rete e che possa essere interrogato inviando un messaggio.

A causa delle sue grandi dimensioni, la rete Internet utilizza un sistema dei nomi particolarmente ben sviluppato, il *sistema dei nomi di dominio* (DNS, *domain name system*): di conseguenza, usiamo il sistema DNS come base per trattare il problema dei nomi di host. Notate che Internet non usa il sistema DNS da sempre. Nelle sue prime fasi di operatività, quando in Internet vi erano soltanto poche centinaia di host, esisteva una tabella non gerarchica di corrispondenze fra nomi e indirizzi, contenuta in un file chiamato *hosts.txt*, che veniva gestita da un'autorità centrale, il NIC (Network Information Center). Ogni volta che un sito voleva aggiungere un nuovo host a Internet, l'amministratore del sito inviava un messaggio di posta elettronica al NIC, indicando la nuova coppia di informazioni di tipo indirizzo/nome dell'host. Questa informazione veniva inserita manualmente nella tabella, la

tabella modificata veniva inviata per posta elettronica ai vari siti nel giro di pochi giorni e l'amministratore di sistema di ogni sito installava la tabella ricevuta in ogni host del proprio sito. La risoluzione dei nomi era quindi realizzata da una semplice procedura che cercava il nome di un host nella copia locale della tabella, restituendo l'indirizzo corrispondente.

Non dovreste restare meravigliati scoprendo che la soluzione al problema di assegnare nomi agli host mediante il file *hosts.txt* non avesse funzionato bene al crescere del numero di host della rete Internet. Di conseguenza, intorno alla metà degli anni Ottanta, venne creato il sistema dei nomi di dominio, DNS, che usa uno spazio dei nomi gerarchico anziché *flat* e che distribuisce per tutta Internet la "tabella" delle corrispondenze per questo spazio dei nomi, dopo averla suddivisa in più parti distinte. Queste sottotabelle sono messe a disposizione da server per i nomi (*name server*) che possono essere interrogati attraverso la rete stessa.

Ciò che avviene in Internet è che un utente fornisce ad un programma applicativo un nome di host (magari all'interno di un nome composto, come un indirizzo di posta elettronica o un URL) e il programma contatta il sistema per i nomi, per tradurre tale nome in un indirizzo di host. L'applicazione, poi, apre una connessione verso tale host, fornendo il relativo indirizzo ad un protocollo di trasporto (ad esempio, TCP). Questa situazione è illustrata (nel caso dell'invio di posta elettronica) nella Figura 9.1.

9.1.1 Gerarchia di domini

Il DNS usa uno spazio dei nomi gerarchico per oggetti di Internet. Diversamente dai nomi di file nel sistema operativo Unix, che vengono elaborati da sinistra a destra e le cui componenti sono separate da barre di divisione, i nomi del DNS vengono elaborati da destra a sinistra e usano il punto come elemento separatore (anche se vengono "elaborati" da destra a sinistra, le persone sono solite "leggere" i nomi di dominio da sinistra a destra). Un esempio di nome di dominio per un host è *cicada.cs.princeton.edu*. Notate che abbiamo affermato che i nomi di dominio sono utilizzati per identificare "oggetti" di Internet: ciò che vogliamo dire è che il DNS non è usato solamente per mettere in corrispondenza nomi e indirizzi di host, mentre è più preciso asserire che il DNS crea corrispondenze tra nomi di dominio e valori. Per il momento ipotizziamo che questi valori siano indirizzi IP, ma più avanti, in questa stessa sezione, torneremo su questo argomento.

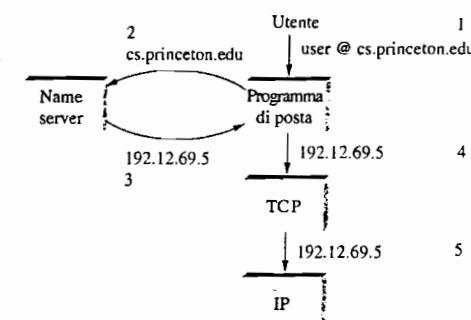


Figura 9.1 Nomi tradotti in indirizzi; i numeri da 1 a 5 mostrano la sequenza di passi del procedimento.

Come la gerarchia dei file nel sistema operativo Unix, la gerarchia del DNS può essere visualizzata mediante un albero, dove ogni nodo dell'albero corrisponde ad un dominio e le foglie dell'albero corrispondono agli host a cui si assegna un nome, come esemplificato nella Figura 9.2. Notate che non è opportuno assegnare alcun significato sémantico al termine "dominio", al di là del fatto che è semplicemente un contesto all'interno del quale si possono definire ulteriori nomi.

Quando venne inizialmente sviluppata la gerarchia dei nomi di dominio, vi furono molte e accese discussioni per decidere a quali convenzioni dovessero sottostare i nomi posti vicino alla cima della gerarchia. Senza entrare in dettaglio in quella discussione, notate che la gerarchia, al primo livello, non è molto estesa. Ci sono domini per ciascun Paese, più i "sei grandi" domini ("big six"): edu, com, gov, mil, org e net. Questi sei domini hanno tutti base negli Stati Uniti d'America e gli unici nomi di dominio che non specificano esplicitamente il nome di un Paese sono quelli degli Stati Uniti. A parte la polarizzazione dedicata agli Stati Uniti, si può ben notare l'impronta militare nella gerarchia: un fenomeno di facile spiegazione, dal momento che lo sviluppo di DNS fu finanziato, in origine, da ARPA, la principale sezione di ricerca del Ministero della Difesa statunitense.

9.1.2 Server per i nomi

La gerarchia completa dei nomi di dominio esiste soltanto in astratto. Rivolgiamo ora la nostra attenzione all'implementazione di questa gerarchia, chiedendoci come venga realizzata in pratica. Il primo passo consiste nel suddividere la gerarchia in sottoalberi, denominati *zone*. Ad esempio, la Figura 9.3 mostra come si possa suddividere in zone la gerarchia di Figura 9.2. Si può pensare a ciascuna zona come se corrispondesse ad un'autorità di gestione che sia responsabile di quella parte della gerarchia completa; ad esempio, la parte superiore della gerarchia è una zona che viene gestita direttamente dal NIC. Al di sotto di questa zona, troviamo una zona che corrisponde alla Princeton University. All'interno di tale zona, alcuni dipartimenti non vogliono avere la responsabilità di gestire la gerarchia (e, quindi, rimangono all'interno della zona dell'università), mentre altri, come il Department of Computer Science, gestiscono la propria zona dipartimentale.

L'importanza delle zone sta nel fatto che corrispondono all'unità fondamentale di implementazione del sistema dei nomi di dominio, il server per i nomi (*name server*). In particolare, l'informazione contenuta in ciascuna zona viene gestita da due o più server per i nomi, ognuno dei quali, a sua volta, è un programma a cui si può accedere tramite Internet. I

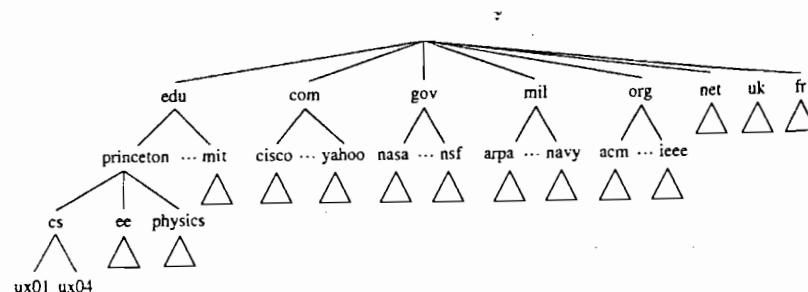


Figura 9.2 Esempio di una gerarchia di domini.

9.1 Servizio per i nomi (DNS)

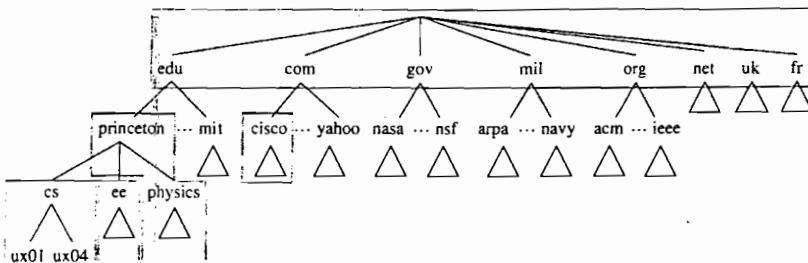


Figura 9.3 Gerarchia di domini suddivisa in zone.

client inviano richieste ai server per i nomi, i quali rispondono fornendo le informazioni richieste. A volte la risposta conterrà l'informazione finale che il client cercava, altre volte contiene un riferimento ad un altro server che il client dovrà interrogare successivamente. Di conseguenza, da un punto di vista implementativo, è più corretto pensare al DNS come se fosse rappresentato da una gerarchia di server per i nomi, come illustrato in Figura 9.4, piuttosto che ad una gerarchia di domini.

Notate che ogni zona è realizzata mediante due o più server per i nomi, in modo ridondante; in questo modo le informazioni sono disponibili anche se uno dei server è guasto. D'altra parte, però, un determinato server per i nomi ha la facoltà di realizzare più di una zona.

Ogni *name server* gestisce le informazioni relative ad una zona come un insieme di *record di risorsa*. In sostanza, un record di risorsa è una corrispondenza fra nome e valore, o, più precisamente, una cinquina che contiene i seguenti campi:

(Name, Value, Type, Class, TTL)

I campi *Name* e *Value* hanno precisamente il significato che si può intuire (rispettivamente, il *nome* e il *valore*), mentre il campo *Type* specifica come debba essere interpretato il campo

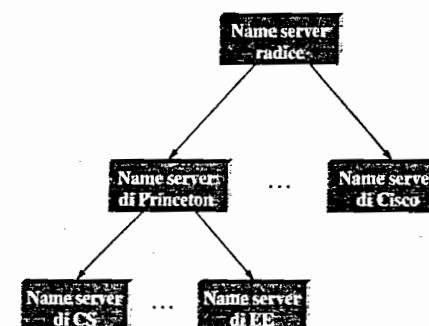


Figura 9.4 Gerarchia di server per i nomi (*name server*).

Value. Ad esempio, **Type = A** indica che **Value** è un indirizzo IP. Di conseguenza, i record A realizzano proprio le corrispondenze tra nomi e valori di cui stiamo parlando. Tra gli altri tipi di record, troviamo:

- **NS:** Il campo **Value** fornisce il nome di dominio di un host che sta svolgendo la funzione di server per i nomi e che sa come risolvere i nomi all'interno del dominio specificato.
- **CNAME:** Il campo **Value** fornisce il nome canonico per un certo host; viene usato per definire dei nomi alternativi (*alias*).
- **MX:** Il campo **Value** fornisce il nome di dominio per un host che sta svolgendo la funzione di server di posta elettronica e che accetta messaggi per il dominio specificato.

Il campo **Class** è stato incluso per consentire ad entità diverse dal NIC di definire tipi di record utili. A tutt'oggi, l'unico valore di **Class** che ha raggiunto un'ampia diffusione è quello usato da Internet, che viene indicato con IN. Infine, il campo **TTL** indica il periodo di validità del record di risorsa. Viene usato dai server che memorizzano in una cache i record di risorsa provenienti da altri server: quando il periodo **TTL** scade, il server deve rimuovere il record dalla propria memoria cache.

Per capire meglio come i record di risorsa rappresentino l'informazione all'interno della gerarchia dei domini, considerate i seguenti esempi, tratti dalla gerarchia di domini presentata in Figura 9.2. Per semplificare gli esempi, ignoriamo il campo **TTL** e descriviamo le informazioni importanti per uno solo dei server per i nomi che implementano ciascuna zona.

Innanzitutto, il server radice (*root name server*) contiene un record di tipo NS per ogni server di secondo livello, oltre ad avere un record di tipo A che ne traduce il nome nell'indirizzo IP corrispondente. Considerati insieme, questi due record, a tutti gli effetti, realizzano un riferimento dal server radice a ciascuno dei server di secondo livello.

```
<princeton.edu, cit.princeton.edu, NS, IN>
<cit.princeton.edu, 128.196.128.233, A, IN>
<cisco.com, ns.cisco.com, NS, IN>
<ns.cisco.com, 128.96.32.30, A, IN>
...

```

Poi, il dominio **princeton.edu** ha un server per i nomi nell'host **cit.princeton.edu**, che contiene i record seguenti. Notate che alcuni di questi record forniscono la risposta finale (ad esempio, l'indirizzo dell'host **saturn.physics.princeton.edu**), mentre altri puntano a *name server* di terzo livello.

```
<cs.princeton.edu, gnat.cit.princeton.edu, NS, IN>
<gnat.cs.princeton.edu, 192.12.69.5, A, IN>
<ee.princeton.edu, helios.ee.princeton.edu, NS, IN>
<helios.ee.princeton.edu, 128.196.28.166, A, IN>
<jupiter.physics.princeton.edu, 128.196.4.1, A, IN>
<saturn.physics.princeton.edu, 128.196.4.2, A, IN>
<mars.physics.princeton.edu, 128.196.4.3, A, IN>
<venus.physics.princeton.edu, 128.196.4.4, A, IN>
...

```

9.1 Servizio per i nomi (DNS)

Infine, un *name server* di terzo livello, come quello gestito dal dominio **cs.princeton.edu**, contiene i record di tipo A per tutti i propri host, e può anche definire un insieme di alias (cioè record di tipo CNAME) per ognuno di tali host. Gli alias sono a volte soltanto nomi di comodo (ad esempio, più brevi) per gli host, ma possono anche essere utilizzati per fornire un livello di redirect. Ad esempio, **www.cs.princeton.edu** è un alias per l'host di nome **cicada.cs.princeton.edu**. Questo permette di spostare su un diverso host il server Web del sito senza creare problemi agli utenti remoti, che continueranno semplicemente ad usare l'alias senza curarsi di quale sia l'host sul quale è in esecuzione, in realtà, il server Web del dominio. I record per lo scambio di posta elettronica (MX, mail exchange) svolgono la medesima funzione per l'applicazione di posta elettronica: permettono all'amministratore di modificare l'host che riceve la posta per conto del dominio senza dover cambiare l'indirizzo di posta elettronica di tutti.

```
<cs.princeton.edu, gnat.cit.princeton.edu, MX, IN>
<cicada.cs.princeton.edu, 192.12.69.60, A, IN>
<cic.cs.princeton.edu, cicada.cs.princeton.edu, CNAME, IN>
<gnat.cs.princeton.edu, 192.12.69.5, A, IN>
<gna.cs.princeton.edu, gnat.cs.princeton.edu, CNAME, IN>
<cicada.cs.princeton.edu, 192.12.69.35, A, IN>
<www.cs.princeton.edu, cicada.cs.princeton.edu, CNAME, IN>
...

```

Note che, anche se teoricamente si possono definire record per risorse di ogni tipo di oggetti, il DNS viene usato tipicamente per assegnare un nome agli host (compresi i server) e ai siti, mentre non viene usato per assegnare nomi alle singole persone o ad altri oggetti come i file e le cartelle: per identificare in modo univoco tali oggetti, si usano solitamente sistemi diversi. Ad esempio, X.500 è un sistema standardizzato da ISO per assegnare i nomi, che rende semplice l'identificazione delle persone mediante un insieme di attributi: il nome, il titolo, il numero di telefono, l'indirizzo postale, e così via. Lo standard X.500 si è dimostrato troppo pesante (e, per certi versi, è stato superato dai potenti motori di ricerca oggi disponibili sul Web), ma alla fine si è evoluto nello standard LDAP (Lightweight Directory Access Protocol). LDAP è un sottoinsieme di X.500 che fu originariamente progettato come programma di accesso a X.500 per i PC e che oggi sta raccogliendo consensi sempre maggiori come sistema per memorizzare informazioni in merito agli utenti, soprattutto al livello di grandi imprese.

9.1.3 Traduzione dei nomi

Data una gerarchia di server per i nomi, consideriamo ora il problema di come possa un client chiedere a tali server la traduzione di un nome di dominio. Per illustrare l'idea di base, supponete che il client voglia risolvere il nome **cicada.cs.princeton.edu** relativamente all'insieme di server presentati nella sottosezione precedente. Per prima cosa, il client invia al server radice una richiesta contenente tale nome. Il server radice, non essendo in grado di trovare una corrispondenza per l'intero nome, restituisce la migliore corrispondenza che trova: il record di tipo NS per **princeton.edu**. Oltre a ciò, il server fornisce anche tutti i record correlati: in questo caso, il solo record di tipo A per **cit.princeton.edu**.

Convenzioni per i nomi

La nostra descrizione del sistema DNS si concentra sui *meccanismi* sottostanti, cioè sul modo in cui la gerarchia viene suddivisa in più server e sulle modalità di traduzione di nomi in indirizzi. Esiste, però, un problema altrettanto interessante, ancorché meno tecnico, che riguarda le *convenzioni* che vengono seguite per decidere i nomi utilizzati nel sistema. Ad esempio, per convenzione tutte le università statunitensi si trovano nel dominio `edu`, mentre le università inglesi si trovano nel sottodominio `ac` ("accademico") del dominio `uk` (United Kingdom, "Regno Unito"). A dire il vero, l'esistenza stessa del dominio `uk`, al posto di un dominio `gb` (Great Britain, "Gran Bretagna"), fu fonte di grandi controversie nei primi tempi di utilizzo del sistema DNS, perché quest'ultima non comprende l'Irlanda del Nord.

La cosa che occorre capire quando si parla di convenzioni è il fatto che a volte vengono definite senza che nessuno prenda esplicitamente una decisione. Ad esempio, è questo il caso della convenzione che vuole che un sito mascheri, con il record `MX`, il nome esatto dell'host che svolge la funzione di server di posta elettronica. Un'alternativa che sarebbe stata possibile sarebbe quella di indirizzare la posta a `user@mail.cs.princeton.edu`, più o meno allo stesso modo in cui siamo abituati a cercare il servizio di FTP pubblico di un sito nell'host di nome `ftp.cs.princeton.edu`, e il suo server Web con il nome `www.cs.princeton.edu`.

Esistono anche convenzioni a livello locale, in quanto spesso un'organizzazione assegna i nomi ai propri calcolatori in base ad un insieme di regole coerenti. Dato che i nomi di host `venus`, `saturn` e `mars` sono fra i più popolari di Internet, non è difficile immaginare quale possa essere una convenzione piuttosto diffusa per assegnare i nomi, anche se alcune di tali convenzioni sono più fantasiose. Ad esempio, un sito ha assegnato alle proprie macchine i nomi `up`, `down`, `crashed`, `rebooting`, e così via, dando luogo a frasi fuorvianti come "`rebooting has crashed`" ("il riavvio ha provocato un guasto") e "`up is down`" ("l'alto è in basso"). Esistono, ovviamente, anche nomi assai meno fantasiosi, come accade quando si danno alle macchine nomi indicizzati con numeri interi.

Il client, non avendo ottenuto la risposta che cercava, invia successivamente la stessa richiesta al server per i nomi che si trova in esecuzione nell'host avente indirizzo IP `128.196.128.233`. Nemmeno questo server è in grado di trovare una corrispondenza per il nome completo, per cui restituisce il record di tipo `NS` e i relativi record di tipo `A` per il dominio `cs.princeton.edu`. Finalmente, il client invia nuovamente la medesima richiesta al server in esecuzione nell'host con indirizzo IP `192.12.69.5`, ottenendo questa volta in risposta il record di tipo `A` per `cicada.cs.princeton.edu`.

Questo esempio lascia ancora senza risposta un paio di domande relative al procedimento di traduzione dei nomi. La prima domanda è: come fa il client a localizzare, inizialmente, il server radice? Oppure, formulando la domanda in altro modo, come si fa a tradurre in un indirizzo IP il nome del server che sa come tradurre i nomi? Si tratta di un problema fondamentale in qualsiasi sistema di assegnazione di nomi e la risposta è che occorre far partire il sistema in qualche modo. In questo caso, la corrispondenza fra nome e indirizzo di uno o più server radice è ben nota, cioè viene resa pubblicamente nota con qualche modalità esterna al sistema stesso che assegna e risolve i nomi.

In pratica, però, non tutti i client conoscono i server radice: molto spesso, invece, in ogni host di Internet è in esecuzione un programma client che viene inizializzato con l'indirizzo di un server *locale* per i nomi. Ad esempio, tutti gli host del Department of Computer Science di Princeton sanno che sull'host `gnat.cs.princeton.edu` si trova un server per i nomi; tale server locale, a sua volta, contiene i record delle risorse relativi a uno o più server radice, ad esempio:

```
<'root', venera.isi.edu, NS, IN>
<venera.isi.edu, 128.9.0.32, A, IN>
```

Di conseguenza, la traduzione di un nome richiede, in pratica, l'invio di una richiesta da parte del client al server locale, il quale a sua volta agisce nel ruolo di client e invia una richiesta a server remoti per conto del client originario. In questo modo si producono le interazioni fra client e server che sono riprodotte in Figura 9.5. Un vantaggio di questo modello è che tutti gli host di Internet non hanno alcuna necessità di essere aggiornati in merito alla localizzazione dei server radice: soltanto i server per i nomi devono avere informazioni relative alla radice. Un secondo vantaggio consiste nel fatto che i server locali ricevono le risposte generate dalle richieste che essi stessi hanno inviato per conto dei client locali: in questo modo il server locale può memorizzare tali risposte e a volte può essere in grado di fornire future risposte senza dover reperire informazioni in rete. Il campo `TTL` dei record di risorsa che vengono restituiti dai server remoti indica il periodo di tempo per il quale ciascun record può essere memorizzato senza problemi.

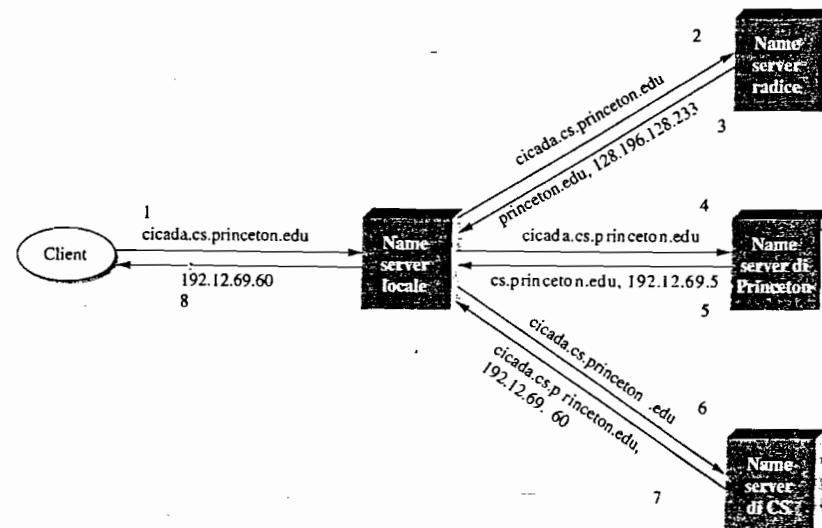


Figura 9.5 Il funzionamento concreto della traduzione dei nomi, dove i numeri da 1 a 8 mostrano la sequenza dei passi coinvolti nel procedimento.

La seconda domanda riguarda la modalità di funzionamento del sistema quando un utente fornisce un nome parziale (ad esempio, cicada) invece di un nome di dominio completo (cioè cicada.cs.princeton.edu). La risposta è che il programma client viene configurato con il dominio locale in cui l'host è situato (in questo caso, cs.princeton.edu) e, prima di inviare una richiesta, aggiunge tale stringa a qualsiasi nome semplice.

Per essere ancora più chiari, abbiamo visto fino ad ora tre livelli diversi di identificativi, i nomi di dominio, gli indirizzi IP e gli indirizzi di rete fisica, e la corrispondenza tra identificativi di un tipo e identificativi di un altro tipo avviene in punti diversi dell'architettura di rete. Per prima cosa, un utente specifica un nome di dominio interagendo con l'applicazione. Poi, l'applicazione usa il sistema DNS per tradurre tale nome in un indirizzo IP: è questo indirizzo IP che viene inserito nei datagrammi, non il nome di dominio (come effetto collaterale, questo processo di traduzione richiede che vengano inviati sulla rete Internet alcuni datagrammi IP, ma questi sono indirizzati a host che eseguono un server per i nomi, non alla destinazione finale). Come terzo passo, il protocollo IP esegue operazioni di inoltro in ogni router, richiedendo spesso di identificare una corrispondenza tra un indirizzo IP ed un altro: in sostanza, il protocollo IP traduce l'indirizzo della destinazione finale nell'indirizzo del router verso cui è diretto il salto successivo. Infine, il protocollo IP sfrutta il protocollo ARP per tradurre l'indirizzo IP del salto successivo nell'indirizzo fisico di tale macchina (il salto successivo potrebbe essere la destinazione finale o un router intermedio): i frame che vengono trasmessi lungo la rete fisica hanno, nelle proprie intestazioni, questi indirizzi fisici.

9.2 Applicazioni tradizionali

Il sistema dei nomi di dominio è probabilmente un'applicazione essenziale per Internet, ma è un'applicazione con cui l'utente non interagisce direttamente: ci occupiamo ora, invece, di quelle applicazioni che vengono direttamente utilizzate dagli utenti, concentrando la nostra attenzione su due tra le più popolari, la posta elettronica e il World Wide Web. Esaminiamo anche la gestione della rete, che, sebbene non sia così familiare all'utente medio, è l'applicazione preferita degli amministratori di sistema. Come il sistema DNS, tutte queste tre applicazioni utilizzano il paradigma richiesta/risposta: gli utenti inviano richieste ai server, i quali rispondono conseguentemente. Parliamo di queste come di applicazioni "tradizionali" perché sono esempi di quelle applicazioni che esistono fin dagli albori delle reti di calcolatori; al contrario, le due sezioni successive prendono in esame una classe di applicazioni che sono divenute realizzabili soltanto nel passato più recente: le applicazioni di tipo *streaming* (cioè applicazioni multimediali per video e audio) e varie applicazioni basate su *reti overlay* ("reti sovrapposte").

Prima di entrare nel dettaglio di ognuna di queste applicazioni, dobbiamo puntualizzare tre aspetti generali. Il primo riguarda l'importanza di distinguere tra *programmi* applicativi e *protocolli* applicativi. Ad esempio, il protocollo HTTP (HyperText Transmission Protocol) è un protocollo applicativo utilizzato per reperire pagine Web da server remoti. Esistono molti diversi programmi applicativi (cioè client Web come Internet Explorer, Mosaic e Netscape) che presentano all'utente un diverso stile per l'interfaccia grafica, ma tutti usano il medesimo protocollo, HTTP, per comunicare attraverso Internet con i server Web. Questa sezione tratta di tre protocolli applicativi:

9.2 Applicazioni tradizionali

- SMTP: Simple Mail Transfer Protocol, usato per scambiare messaggi di posta elettronica.
- HTTP: HyperText Transmission Protocol, usato per le comunicazioni tra *browser* Web e server Web.
- SNMP: Simple Network Management Protocol, usato per ispezionare (e a volte modificare) lo stato di nodi di rete remoti.

Il secondo punto centrale è che, dal momento che tutti i protocolli applicativi descritti in questa sezione seguono il medesimo schema di comunicazione richiesta/risposta, ci aspetteremmo che fossero tutti costruiti al di sopra di un protocollo di trasporto di tipo RPC. Tuttavia, non è così, dato che sono tutti basati sul protocollo TCP o UDP. In effetti, ciascuno di questi protocolli utilizza un proprio semplice meccanismo, simile a RPC, al di sopra di uno dei protocolli di trasporto esistenti; parliamo di protocolli "semplici" perché ognuno di essi non è stato progettato per fornire il supporto ad invocazioni remote di procedure di qualsiasi tipo, ma è stato, invece, progettato per inviare uno specifico insieme di messaggi di richiesta e rispondere ad essi. Il fatto che due di questi protocolli abbiano la parola "Simple" nel proprio nome non è proprio una coincidenza.

Tutti questi tre protocolli hanno un protocollo ausiliario che specifica il formato dei dati che possono essere scambiati. Questa è una delle ragioni per cui questi protocolli sono relativamente semplici: la maggior parte della complessità è gestita da questi documenti di accompagnamento. Ad esempio, SMTP è un protocollo per lo scambio di messaggi di posta elettronica, mentre RFC 822 (questa specifica non ha altro nome) e MIME (Multipurpose Internet Mail Extensions) definiscono il formato dei messaggi di posta elettronica. Analogamente, HTTP è un protocollo per trasferire pagine Web, mentre HTML (HyperText Markup Language) è una specifica di accompagnamento che definisce il formato di tali pagine. Infine, SNMP è un protocollo per interrogare un nodo di rete, mentre MIB (Management Information Base) definisce le variabili sulle quali si possono richiedere informazioni.

9.2.1 Posta elettronica (SMTP, MIME, IMAP)

La posta elettronica è stata la prima applicazione di rete. Dopo tutto, cosa potrebbe essere più naturale del desiderio di inviare un messaggio ad un utente che si trova all'altro capo di una linea di collegamento che attraversa il Paese e che siete appena riusciti a far funzionare? A dire il vero, i pionieri della rete ARPANET non avevano davvero capito, quando fu creata la rete, che la posta elettronica potesse divenire un'applicazione chiave; il principale obiettivo del progetto era l'accesso remoto a risorse di calcolo, ma la posta elettronica si rivelò un'applicazione dal successo sorprendente. Da quel prototipo prese spunto l'evoluzione del sistema di posta elettronica di Internet, che è oggi usato da milioni di persone ogni giorno.

Come per tutte le applicazioni descritte in questa sezione, il modo migliore per iniziare a capire il funzionamento della posta elettronica consiste nel tenere distinti l'interfaccia utente (ad esempio, il programma che utilizzate per leggere la posta) e il sottostante protocollo per il trasferimento dei messaggi (in questo caso, SMTP), e distinguere ulteriormente fra questo protocollo di trasferimento e i protocolli ausiliari (RFC 822 e MIME) che definiscono il formato dei messaggi che vengono scambiati. Cominciamo proprio dal formato dei messaggi.

Formato dei messaggi

Il documento RFC 822 definisce che i messaggi sono costituiti da due parti: una *intestazione* (*header*) e un *corpo* (*body*), rappresentate entrambe mediante testo ASCII. In origine il corpo

era semplicemente costituito da testo, ed è ancora così, anche se RFC 822 è stato integrato da MIME, che consente al messaggio di veicolare dati di qualsiasi tipo. Tali dati sono ancora rappresentati mediante testo ASCII, che, però, non è necessariamente leggibile da una persona umana, potendo essere, ad esempio, la versione opportunamente codificata di un'immagine JPEG. Fra poco saremo più esaurienti in merito a MIME.

L'intestazione di un messaggio è una serie di righe terminate da <CRLF> (la notazione <CRLF> rappresenta una coppia di caratteri di controllo del codice ASCII, *carriage return e line feed*, che vengono solitamente utilizzati per indicare la fine di una riga di testo). L'intestazione è separata dal corpo del messaggio mediante una riga vuota. Ogni riga dell'intestazione contiene un tipo e un valore, separati da un carattere "due punti": molte di queste righe di intestazione sono ben note agli utenti, poiché viene loro chiesto di compilarle quando scrivono un messaggio di posta elettronica. Ad esempio, l'intestazione *To:* identifica il destinatario del messaggio e l'intestazione *Subject:* fornisce informazioni in merito allo scopo del messaggio. Le altre intestazioni sono compilate dal sottostante sistema di consegna della posta; tra queste, *Date:* (l'orario di trasmissione del messaggio), *From:* (l'utente mittente del messaggio) e *Received:* (un'indicazione oraria inserita da ciascun server di posta che ha gestito il messaggio). Esistono, ovviamente, molte altre righe di intestazione: il lettore interessato è rimandato al documento RFC 822.

RFC 822 venne esteso nel 1993 (e nuovamente aggiornato nel 1996) per consentire ai messaggi di posta elettronica di veicolare molti tipi di dati diversi: audio, video, immagini, documenti Word, e così via. MIME è composto da tre parti fondamentali. La prima parte è un insieme di righe d'intestazione che integra l'insieme originale definito in RFC 822 per descrivere, in varie modi, i dati trasportati nel corpo del messaggio. Tra queste, *MIME-Version:* (la versione di MIME che viene utilizzata), *Content-Description:* (una descrizione leggibile di cosa sia contenuto nel messaggio, analogamente alla riga *Subject:*), *Content-Type:* (il tipo di dati contenuti nel messaggio) e *Content-Transfer-Encoding:* (come siano codificati i dati nel messaggio).

La seconda parte contiene la definizione di un insieme di tipi (e sottotipi) di contenuti. Ad esempio, MIME definisce due diversi tipi di immagini statiche, identificate come *image/gif* e *image/jpeg*, ciascuno dei quali ha un ovvio significato. Come ulteriore esempio, *text/plain* si riferisce al testo in forma semplice che potrebbe costituire il corpo di un normale messaggio nello stile di RFC 822, mentre *text/richText* identifica un messaggio che contiene testo "con marcatori" (ad esempio, testo che usa font speciali, caratteri in corsivo, ecc.). Un terzo esempio: MIME definisce un tipo *application*, i cui sottotipi corrispondono a dati prodotti da diversi programmi applicativi (ad esempio, *application/postscript* e *application/msword*).

MIME definisce anche un tipo *multipart* che specifica come sia strutturato un messaggio che veicola più di un tipo di dati, più o meno come un linguaggio di programmazione che definisce sia i tipi di base (ad esempio, numeri interi e numeri in virgola mobile) sia i tipi composti (strutture e array). Un sottotipo possibile di *multipart* è *mixed*, che identifica un messaggio contenente un insieme di dati indipendenti in un ordine specifico: ogni dato ha la propria riga di intestazione che ne descrive il tipo effettivo.

La terza parte indica un modo per codificare i vari tipi di dati in modo che possano essere trasmessi sotto forma di messaggio di posta elettronica in formato ASCII. Il problema consiste nel fatto che, per alcuni tipi di dati (ad esempio, un'immagine JPEG), qualsiasi byte di 8 bit dell'immagine potrebbe contenere uno qualsiasi dei 256 diversi valori possibili, mentre soltanto un sottoinsieme di questi valori comprende caratteri ASCII validi. È importante che

9.2 Applicazioni tradizionali

i messaggi di posta elettronica contengano solamente caratteri ASCII, perché potrebbero attraversare un certo numero di sistemi intermedi (*gateway*, come descritto nel seguito) che assumono che tutta la posta elettronica sia codificata in ASCII e potrebbero rendere illeggibile un messaggio che contenesse caratteri non appartenenti al codice ASCII. Per risolvere questo problema, MIME usa una elementare codifica dei dati binari, trasformandoli in caratteri appartenenti all'insieme ASCII, una codifica chiamata *base64*. L'idea è quella di creare una corrispondenza fra terne di byte dei dati binari originari e quaterne di caratteri ASCII, raggruppando i dati binari in unità di 24 bit e suddividendo ognuna di tali unità in quattro porzioni di 6 bit. Ogni porzione di 6 bit corrisponde ad un carattere ASCII valido all'interno di un insieme di 64 caratteri: ad esempio, 0 corrisponde al carattere A, 1 a B, e così via. Se osservate un messaggio che sia stato codificato usando lo schema di codifica *base64*, noterete soltanto le 52 lettere maiuscole e minuscole, le 10 cifre da 0 a 9 e i caratteri speciali + e /: sono i primi 64 valori dell'insieme di caratteri ASCII.

Come eccezione, per rendere la lettura della posta più agevole per coloro che insistono a voler utilizzare programmi puramente testuali, un messaggio MIME che consista soltanto di testo può essere codificato usando il codice ASCII a 7 bit, ed esiste una codifica leggibile per la maggior parte dei dati ASCII.

Mettendo tutto insieme, un messaggio che contenga una parte di testo normale, un'immagine JPEG e un file PostScript, assomiglia a questo:

```

MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="-----417CA6E2DE4ABCAFBC5"
From: Alice Smith <Alice@cisco.com>
To: Bob@cs.Princeton.edu
Subject: promised material
Date: Mon, 07 Sep 1998 19:45:19 -0400

-----417CA6E2DE4ABCAFBC5
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Bob,
```

Here's the jpeg image and draft report I promised.

--Alice

```

-----417CA6E2DE4ABCAFBC5
Content-Type: image/jpeg
Content-Transfer-Encoding: base64
```

...codifica illeggibile di un'immagine jpeg

```

-----417CA6E2DE4ABCAFBC5
Content-Type: text/plain; name="draft.ps"
Content-Transfer-Encoding: 7bit
```

...codifica illeggibile di un documento PostScript

In questo esempio, la riga `Content-Type` nell'intestazione del messaggio dice che nel messaggio sono presenti varie parti, ciascuna iniziata da una stringa di caratteri che non compaiono nei dati stessi. Ogni parte ha le proprie righe `Content-Type` e `Content-Transfer-Encoding`.

Trasferimento dei messaggi

Esaminiamo ora SMTP, il protocollo utilizzato per trasferire messaggi da un host ad un altro. Per inserire SMTP nel giusto contesto, dobbiamo identificare i protagonisti principali. Innanzitutto, gli utenti interagiscono con un *lettore di posta* (*mail reader*), con il quale leggono, scrivono e archiviano la propria posta: ci sono veramente moltissimi lettori di posta disponibili, da cui poter scegliere, più o meno come avviene per i browser Web, e in realtà la maggior parte dei browser Web contiene oggi anche un lettore di posta. Poi, in ogni host viene eseguito un processo di gestione della posta (*mail daemon*), al quale si può pensare come se avesse il ruolo di un ufficio postale: i lettori di posta consegnano ad esso i messaggi che vogliono inviare ad altri utenti, il *daemon* usa il protocollo SMTP (al di sopra di TCP) per trasmettere il messaggio ad un analogo *daemon* in esecuzione su un altro host, e tale *daemon* inserisce i messaggi in arrivo all'interno della *casella di posta* (*mailbox*) dell'utente, dove il lettore di posta dell'utente lo potrà trovare più tardi. Poiché SMTP è un protocollo che può essere implementato da chiunque, in teoria vi potrebbero essere molte diverse implementazioni del *daemon* per la posta, ma, in realtà, il *daemon* che viene eseguito sulla maggior parte degli host è derivato dal programma *sendmail* originariamente implementato nella versione di Unix di Berkeley.

Anche se è certamente possibile che il programma *sendmail* della macchina mittente stabilisca una connessione SMTP/TCP direttamente con il programma *sendmail* della macchina destinataria, in molti casi i messaggi di posta attraversano uno o più *portali di posta* (*mail gateway*) lungo il percorso che va dall'host mittente all'host destinatario: come gli host terminali, anche questi gateway eseguono un processo *sendmail*. Non è ovviamente un caso che questi nodi intermedi vengano chiamati "gateway", dal momento che il loro compito è quello di memorizzare e inoltrare messaggi di posta elettronica, in modo molto simile al compito svolto da un "gateway IP" (che noi abbiamo chiamato router), che memorizza e inoltra datagrammi IP. L'unica differenza è che un gateway di posta memorizza tipicamente i messaggi su un disco magnetico e prova a ritrasmetterli alla macchina successiva lungo il percorso anche per parecchi giorni, mentre un router IP memorizza i datagrammi in buffer di memoria e tenta di ritrasmetterli soltanto per un frazione di secondo. La Figura 9.6 illustra un percorso con due salti, dal mittente al destinatario.

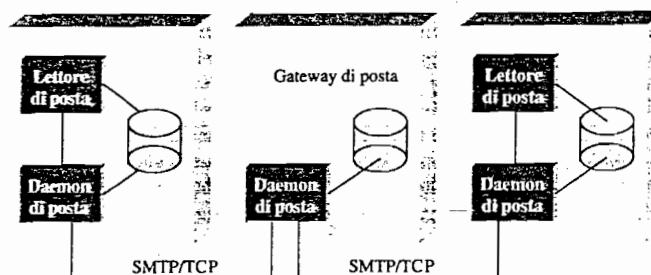


Figura 9.6 Una serie di gateway di posta memorizza e inoltra messaggi di posta elettronica.

9.2 Applicazioni tradizionali

Vi potreste chiedere: perché mai i gateway di posta sono necessari? L'host mittente non potrebbe inviare il messaggio direttamente all'host destinatario? Un motivo è che il destinatario non vuole inserire nel proprio indirizzo il nome dello specifico host sul quale legge la posta. Ad esempio, la posta consegnata a `Bob@cs.princeton.edu` viene dapprima inviata da un gateway di posta che si trova nel CS Department di Princeton (che è l'host di nome `cs.princeton.edu`), e poi inoltrata (mediante una seconda connessione SMTP/TCP) alla specifica macchina sulla quale oggi Bob ha deciso di leggere la propria posta. Il gateway che effettua l'inoltro della posta gestisce una base di dati che mette in corrispondenza gli utenti con la macchina sulla quale desiderano ricevere realmente la propria posta, mentre il mittente non deve preoccuparsi di questi dettagli (l'elenco delle righe `Received`: presenti nell'intestazione del messaggio vi aiuterà a tenere traccia dei gateway di posta che sono stati attraversati dal messaggio stesso). Un altro motivo è che la macchina destinataria potrebbe non essere sempre accessa, nel qual caso il gateway di posta conserva il messaggio finché questo non verrà consegnato.

Indipendentemente da quanti gateway di posta siano presenti lungo il percorso, fra ogni coppia di host viene usata una connessione SMTP autonoma, per far giungere il messaggio più vicino alla sua destinazione. Ogni sessione SMTP è articolata in un dialogo fra i due daemon di posta, uno agente da client e l'altro da server. Dato che RFC 822 stabilisce che i messaggi debbano usare il codice ASCII come formato di rappresentazione, non vi dovreste stupire del fatto che anche il protocollo SMTP sia basato sul codice ASCII: ciò significa che, per una persona umana davanti ad una tastiera, è possibile impersonare il ruolo di client SMTP.

Il modo migliore per capire il protocollo SMTP è con un esempio: nel seguito trovate un dialogo tra l'host mittente `cs.princeton.edu` e l'host destinatario `cisco.com`, corrispondente al caso in cui l'utente Bob di Princeton stia tentando di inviare un messaggio di posta agli utenti Alice e Tom di Cisco. Le righe inviate da `cs.princeton.edu` sono in grassetto, le altre sono inviate da `cisco.com`; sono state aggiunte righe vuote per rendere il dialogo più leggibile.

```

HELO cs.princeton.edu
250 Hello daemon@mail.cs.princeton.edu [128.12.169.24]

MAIL FROM:<Bob@cs.princeton.edu>
250 OK

RCPT TO:<Alice@cisco.com>
250 OK

RCPT TO:<Tom@cisco.com>
550 No such user here

DATA
354 Start mail input; end with <CRLF>.<CRLF>
il testo del messaggio
un'altra riga di testo del messaggio
ecc...
<CRLF>.<CRLF>
250 OK

QUIT
221 Closing connection

```

Come potete vedere, il protocollo SMTP prevede un dialogo fra il client e il server, durante il quale il client invia un comando (ad esempio, HELO, MAIL, RCPT, DATA, QUIT) e il server risponde con un codice (ad esempio, 250, 550, 354, 221) e con una spiegazione del codice comprensibile per un utente umano (ad esempio, No such user here, "utente inesistente"). In questo esempio particolare, per prima cosa il client si identifica con il comando HELO, fornendo come argomento il proprio nome di dominio. Il server verifica che tale nome corrisponda all'indirizzo IP che viene usato per la connessione TCP: noterete, infatti, che il server comunica tale indirizzo IP al client. Il client, poi, chiede al server di accettare posta per due diversi utenti e il server risponde dicendo "sì" per uno e "no" per l'altro. Successivamente il client invia il messaggio, che viene terminato da una riga contenente soltanto un punto ("."); infine, il client termina la connessione.

Ovviamente, esistono molti altri comandi e codici di risposta. Ad esempio, il server potrebbe rispondere al comando RCPT del client con un codice 251, per indicare che l'utente non ha una casella di posta sul server, ma che il server si assume l'incarico di inoltrare il messaggio ad un altro daemon di posta. In altre parole, l'host riveste il ruolo di gateway di posta. Un altro esempio: il client può inviare un comando VRFY per controllare la validità dell'indirizzo di posta di un utente, senza inviare effettivamente un messaggio all'utente stesso.

L'unico altro punto degno di nota è costituito dai parametri dei comandi MAIL e RCPT, ad esempio, rispettivamente, FROM:<Bob@cs.princeton.edu> e TO:<Alice@cisco.com>. Sembrano molto simili a campi di intestazione del formato RFC 822, e in un certo senso lo sono. Ciò che accade, in realtà, è che il daemon di posta analizza il messaggio per estrarre le informazioni che gli servono per eseguire SMTP, informazioni che vanno a costituire ciò che viene chiamato *busta (envelope)* del messaggio, e il client SMTP usa le informazioni di tale busta come parametri per il proprio dialogo con il server SMTP. Una nota storica: il motivo per cui sendmail divenne tanto popolare è che nessuno voleva implementare di nuovo la funzione di analisi dei messaggi. Mentre gli indirizzi di posta elettronica di oggi sembrano abbastanza semplici (ad esempio, Bob@cs.princeton.edu), una volta non era sempre così: prima che tutti fossero connessi ad Internet, non era strano vedere indirizzi di posta simili a user%host@site!neighbor.

Lettore di posta

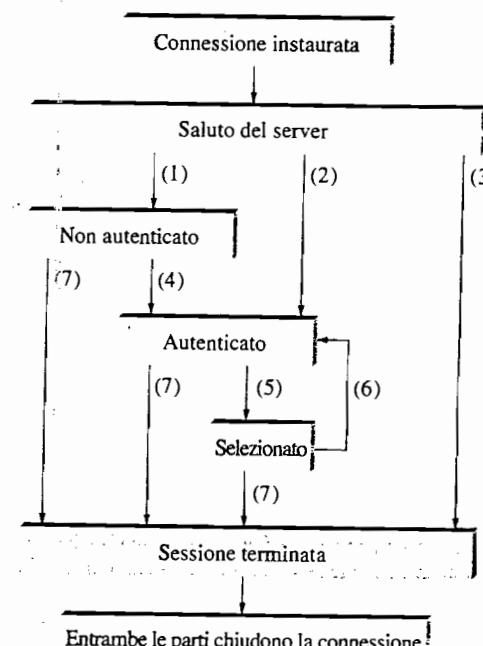
L'ultimo passo è che l'utente recuperi, finalmente, i propri messaggi dalla casella di posta, per leggerli, rispondere ad essi ed eventualmente memorizzarli per usarli in futuro. L'utente esegue tutte queste azioni interagendo con un lettore di posta, che, in molti casi, è semplicemente un programma che viene eseguito sulla macchina dove risiede la casella di posta dell'utente: in questo caso, il lettore di posta legge e scrive, banalmente, il file che realizza la casella di posta. In altri casi l'utente accede alla propria casella di posta da una macchina remota, usando un altro protocollo ancora, come POP (Post Office Protocol) oppure IMAP (Internet Message Access Protocol). Discutere gli aspetti inerenti l'interfaccia utente dei lettori di posta va ben oltre gli scopi di questo libro, mentre invece è assai opportuno che ci occupiamo del protocollo di accesso: in particolare, consideriamo IMAP.

Il protocollo IMAP è simile, per molti aspetti, a SMTP: è un protocollo client/server eseguito al di sopra di TCP, nel quale il client (in esecuzione sulla macchina su cui agisce l'utente) invia comandi aventi la forma di righe di testo ASCII terminate da <CRLF> e il server di posta (in esecuzione sulla macchina che gestisce la casella di posta dell'utente) risponde in modo analogo. Il dialogo inizia con l'autenticazione del client e l'identificazione

9.2 Applicazioni tradizionali

della casella di posta alla quale vuole accedere, e può essere rappresentato dal semplice diagramma di transizione di stato mostrato in Figura 9.7. In questo diagramma, LOGIN, AUTHENTICATE, SELECT, EXAMINE, CLOSE e LOGOUT sono esempi dei comandi che possono essere inviati dal client, mentre OK è una delle possibili risposte del server. Altri comandi sono FETCH, STORE, DELETE e EXPUNGE, con significati abbastanza ovvi, mentre ulteriori possibili risposte del server sono NO (il client non ha il permesso di eseguire tale operazione) e BAD (il comando è sintatticamente scorretto).

Quando l'utente chiede di leggere un messaggio (FETCH), il server lo restituisce in formato MIME e il lettore di posta lo decodifica. Oltre al messaggio in sé, IMAP definisce anche un insieme di *attributi* del messaggio che vengono scambiati come parte di altri comandi,



- (1) connessione priva di pre-autenticazione (saluto OK)
- (2) connessione pre-autenticata (saluto PREAUTH)
- (3) connessione rifiutata (saluto BYE)
- (4) comando LOGIN o AUTHENTICATE andato a buon fine
- (5) comando SELECT o EXAMINE andato a buon fine oppure comando CLOSE
- (6) comando LOGOUT oppure spegnimento del server oppure chiusura della connessione
- (7) comando LOGOUT oppure spegnimento del server oppure chiusura della connessione

Figura 9.7 Diagramma di transizione di stato di IMAP.

indipendentemente dal trasferimento del messaggio stesso; questi attributi contengono informazioni come la dimensione del messaggio e, cosa ancor più interessante, vari *indicatori* (*flag*) associati al messaggio (per esempio: Seen, Answered, Deleted e Recent). Questi indicatori vengono usati per mantenere sincronizzati il client e il server, perché, quando l'utente cancella un messaggio tramite il lettore di posta, il client deve segnalare questo fatto al server di posta. Se, in seguito, l'utente decide di eliminare effettivamente tutti i messaggi "cancellati", il client invia il comando EXPUNGE al server, che conseguentemente elimina davvero dalla casella di posta tutti i messaggi precedentemente "cancellati".

Infine, notate che, quando l'utente risponde ad un messaggio oppure invia un nuovo messaggio, il lettore di posta non inoltra il messaggio dalla macchina client al server di posta usando IMAP, ma usa SMTP. Ciò significa che il server di posta dell'utente è, a tutti gli effetti, il primo gateway di posta che viene attraversato dal messaggio lungo il suo percorso dalla macchina dell'utente alla casella di posta del destinatario.

9.2.2 World Wide Web (HTTP) → Protocollo di tipo testuale

Il World Wide Web (*ragnatela mondiale*) ha avuto talmente successo e ha reso Internet accessibile a talmente tante persone che a volte sembra essere sinonimo di Internet. Un utile modo di pensare al Web è di immaginarlo come un insieme di client e di server che cooperano, parlando tutti il medesimo linguaggio: il protocollo HTTP. La maggior parte delle persone vedono il Web attraverso un programma client dotato di interfaccia grafica, detto *browser* ("navigatore") Web, come Netscape o Internet Explorer: la Figura 9.8 mostra il browser Netscape in azione, mentre visualizza una pagina informativa di un progetto denominato PlanetLab.

Ogni browser Web ha una funzione che consente all'utente di "aprire un URL". Gli URL (Uniform Resource Locator, *localizzatori omogenei di risorse*) forniscono informazioni in merito alla collocazione di oggetti nel Web e assomigliano a questo:

<http://www.cs.princeton.edu/index.html>

Se avete provato ad accedere a quel particolare URL, il vostro browser Web avrà aperto una connessione TCP verso il server Web che si trova in esecuzione su un calcolatore che si chiama www.cs.princeton.edu e avrà immediatamente ricevuto e visualizzato il file di nome `index.html`. La maggior parte dei file presenti nel Web contiene immagini e testo (alcuni anche brani audio e video) e contengono anche URL che puntano ad altri file, e il vostro browser Web dovrà consentirvi, in qualche modo, di identificare tali URL e di richiederne la loro apertura. Questi URL incorporati in altri oggetti vengono chiamati *collegamenti ad ipertesto* (*hypertext link*): quando chiedete al vostro browser Web di aprire uno di questi URL incorporati (ad esempio, selezionandolo e premendo il pulsante del mouse), verrà aperta una nuova connessione per recuperare e visualizzare un nuovo file, e si parla di "seguire un collegamento". In questo modo risulta molto semplice saltare da una macchina all'altra nella rete, seguendo collegamenti ad ogni tipo di informazioni.

Quando scegliete di visualizzare una pagina, il vostro browser (il client) recupera la pagina dal server usando il protocollo HTTP, in esecuzione al di sopra di TCP; come SMTP, anche HTTP è un protocollo di tipo testuale. Essenzialmente, ciascun messaggio HTTP ha la forma generale seguente:

URL = Uniform Resource Locator

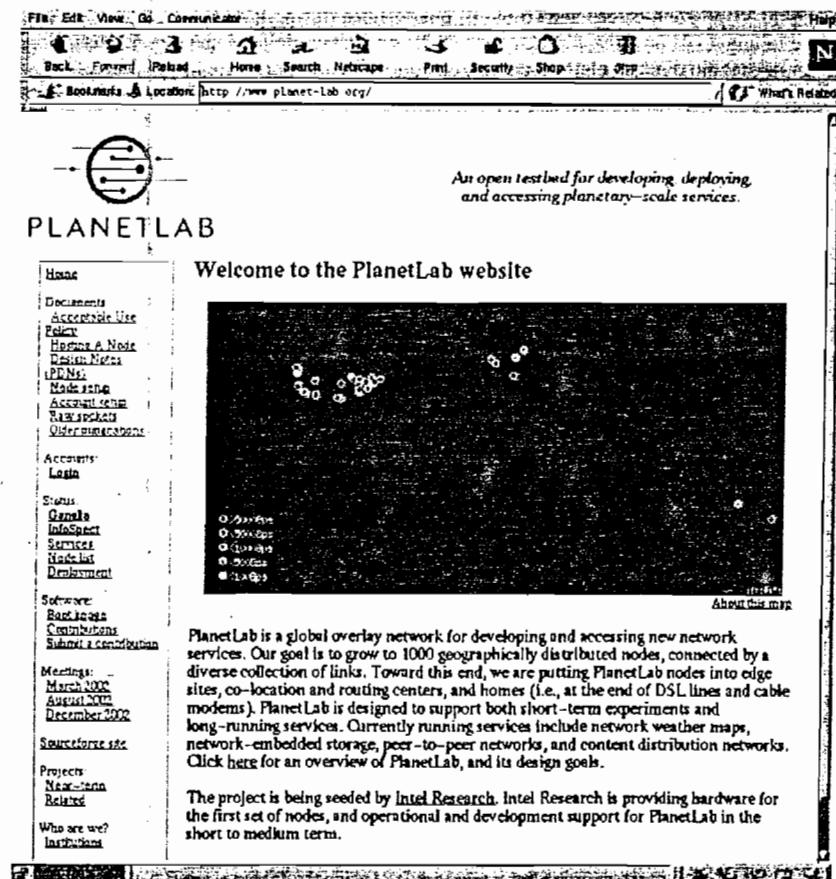
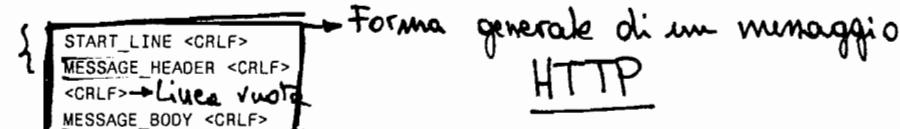


Figura 9.8 Il browser Web Netscape.



dove, come in precedenza, <CRLF> simboleggia l'accoppiata *carriage-return* e *line-feed*. La prima riga (START_LINE) indica se si tratta di un messaggio di richiesta o di risposta. In effetti, identifica la "procedura remota" che deve essere eseguita (nel caso di un messaggio di richiesta) o lo "stato" della richiesta (nel caso di un messaggio di risposta). Il successivo insieme di linee specifica una raccolta di opzioni e parametri che qualificano la richiesta o la

risposta. Vi possono essere zero o più di queste linee di tipo MESSAGE_HEADER (l'insieme viene terminato da una riga vuota), ognuna delle quali assomiglia ad una riga di intestazione in un messaggio di posta elettronica. Il protocollo HTTP definisce molti tipi di intestazioni possibili, alcuni dei quali sono pertinenti in messaggi di richiesta o in messaggi di risposta, mentre altri si riferiscono ai dati veicolati nel corpo del messaggio. Tuttavia, invece di elencare qui tutti i tipi di intestazioni possibili, citeremo soltanto alcuni esempi particolarmente rappresentativi. Infine, dopo la riga vuota troviamo il contenuto del messaggio richiesto (MESSAGE_BODY), una parte che è tipicamente vuota nei messaggi di richiesta.

Messaggi di richiesta

La prima riga di un messaggio di richiesta HTTP specifica tre cose: l'operazione che deve essere eseguita, la pagina Web su cui va eseguita l'operazione e la versione di HTTP che si sta usando. Anche se il protocollo HTTP definisce un vasto assortimento di operazioni che si possono richiedere, tra cui operazioni di "scrittura" che consentono di inserire una pagina Web in un server, le due operazioni più comuni sono GET (recupera la pagina Web specificata) e HEAD (recupera informazioni in merito alla pagina Web specificata). La prima operazione viene utilizzata, ovviamente, quando un browser vuole recuperare e visualizzare una pagina Web, mentre la seconda si usa per verificare la validità di un collegamento ipertestuale. Esistono cinque categorie di codici di risposta, identificate dalla prima cifra del codice stesso, e la Tabella 9.2 le riassume.

Ad esempio, una prima riga siffatta:

```
GET http://www.cs.princeton.edu/index.html HTTP/1.1
```

dice che il client vuole che il server presente nell'host www.cs.princeton.edu fornisca la pagina di nome index.html. Questo particolare esempio usa un URL assoluto, ma è anche possibile usare un identificativo relativo e specificare il nome dell'host in una delle righe del MESSAGE HEADER, ad esempio così:

```
GET index.html HTTP/1.1
Host: www.cs.princeton.edu
```

Tabella 9.1 Operazioni di richiesta HTTP.

+ column

Operazione	Descrizione
OPTIONS	Richiede informazioni sulle opzioni disponibili
GET	Richiede il documento identificato dall'URL
HEAD	Richiede informazioni sul documento identificato dall'URL
POST	Fornisce informazioni (es: annotazioni) al server
PUT	Memorizza il documento nell'URL specificato
DELETE	Cancella l'URL specificato
TRACE	Effettua l'eco del messaggio di richiesta
CONNECT	Utilizzato dai proxy

9.2 Applicazioni tradizionali

Qui, Host è uno dei campi possibili nel MESSAGE_HEADER. Uno dei campi più interessanti, tra questi, è If-Modified-Since, che fornisce al client il modo di effettuare una richiesta condizionata al server: il server restituisce la pagina soltanto se questa è stata modificata dopo l'indicazione oraria specificata in quella riga di intestazione.

Messaggi di risposta

Come i messaggi di richiesta, i messaggi di risposta iniziano con una singola riga, START_LINE. In questo caso la riga specifica la versione di HTTP che si sta usando, un codice a tre cifre che indica se la richiesta è stata soddisfatta oppure no, e una stringa di testo che fornisce le motivazioni della risposta. Ad esempio, questa riga iniziale:

```
HTTP/1.1 202 Accepted
```

indica che il server è stato in grado di soddisfare la richiesta, mentre:

```
HTTP/1.1 404 Not Found
```

indica che non è stato in grado di soddisfare la richiesta perché la pagina non è stata trovata. Esistono cinque categorie di codici di risposta, identificate dalla prima cifra del codice stesso, e la Tabella 9.2 le riassume.

Sempre in analogia con i messaggi di richiesta, i messaggi di risposta possono contenere una o più righe nel MESSAGE_HEADER, per riferire al client informazioni addizionali. Ad esempio, la riga di intestazione Location specifica che l'URL richiesto è disponibile in un luogo diverso, per cui, se la pagina Web del CS Department di Princeton è stata spostata, ad esempio da <http://www.cs.princeton.edu/index.html> a <http://www.princeton.edu/cs/index.html>, allora il server che si trova all'indirizzo originario potrebbe rispondere così:

```
HTTP/1.1 301 Moved Permanently
Location: http://www.princeton.edu/cs/index.html
```

Nel caso più comune, il messaggio di risposta avrà al proprio interno la pagina richiesta, che è un documento HTML; dato che però può avere anche dati non di testo (ad esempio, un'immagine GIF), viene codificata con MIME (si veda la Sezione 9.2.1). Alcune righe di MESSAGE_HEADER forniscono informazioni sul contenuto della pagina, fra cui Content-Length (numero di byte del contenuto), Expires (momento in cui il contenuto sarà da considerarsi

Esempi di 8
* messaggi
di risposta

Tabella 9.2 Le cinque categorie di codici di risposta di HTTP.

Codice	Categoria	Esempio di motivazione
1xx	Informazione	Richiesta ricevuta, l'elaborazione sta continuando
2xx	Successo	L'azione è stata ricevuta con successo, compresa e accettata
3xx	Redirezione	Per portare a termine la richiesta sono richieste azioni ulteriori
4xx	Errore del client	La richiesta contiene errori di sintassi o non può essere soddisfatta
5xx	Errore del server	Il server non è stato in grado di soddisfare una richiesta apparentemente valida

scaduto) e Last-Modified (momento in cui al contenuto sul server è stata apportata la modifica più recente).

Connessioni TCP

La versione originale di HTTP (1.0) richiedeva una diversa connessione TCP per ogni elemento richiesto al server. Non è difficile capire come questo fosse una modalità di funzionamento assai inefficiente: il client e il server dovevano scambiarsi i messaggi di instaurazione e di chiusura della connessione anche soltanto perché il client potesse verificare di essere già in possesso della copia più recente della pagina richiesta. Di conseguenza, scaricare una pagina contenente testo e una dozzina di piccole immagini richiedeva l'instaurazione e la chiusura di 13 diverse connessioni TCP.

Il miglioramento più significativo nella versione più recente di HTTP (1.1) è stato quello di consentire *connessioni persistenti*: il client e il server possono scambiarsi più messaggi di tipo richiesta/risposta usando la stessa connessione TCP. Le connessioni persistenti presentano due vantaggi: il primo è che, ovviamente, viene eliminato il tempo richiesto per instaurare le connessioni, riducendo così il carico per il server, il carico per la rete provocato dai pacchetti TCP addizionali e il ritardo percepito dall'utente; il secondo è che il meccanismo della finestra di congestione di TCP è in grado di operare in modo più efficiente, dal momento che un client invia più messaggi di richiesta lungo un'unica connessione TCP e non è più necessario passare per la fase di "partenza lenta" ad ogni pagina.

Tuttavia, le connessioni persistenti hanno un loro prezzo: il problema è che né il client né il server sono in grado di sapere per quanto tempo una certa connessione TCP debba essere tenuta aperta. Questo aspetto è particolarmente critico per il server, al quale si potrebbe chiedere di mantenere aperte connessioni per migliaia di client. La soluzione è che il server debba chiudere una connessione in seguito alla scadenza di un temporizzatore, qualora non abbia ricevuto richieste da quella connessione per un certo periodo di tempo. Ancora, tanto il client quanto il server devono controllare se la controparte abbia deciso di chiudere la connessione, usando tale informazione come segnale per chiudere la connessione anche dal proprio lato (ricordate che, per porre definitivamente fine ad una connessione TCP, è necessario che venga chiusa da entrambe le parti).

Memorizzazione temporanea (*caching*)

Una delle aree più attive della ricerca (e dell'imprenditoria) nella rete Internet di oggi riguarda la memorizzazione temporanea (*caching*) efficiente di pagine Web, che ha molti vantaggi. Dal punto di vista del client, una pagina che venga trovata in una *cache* molto vicina può essere visualizzata molto più rapidamente di quanto possa avvenire se si deve recuperare la pagina dall'altra parte del globo. Dal punto di vista del server, il suo carico viene ridotto dal fatto che una cache abbia intercettato e soddisfatto una richiesta ad esso diretta.

Questa strategia può essere realizzata in molte diverse condizioni. Ad esempio, il browser di un utente può memorizzare le pagine recentemente visualizzate e usare semplicemente una copia memorizzata quando l'utente visita nuovamente la stessa pagina. Oppure, un sito può utilizzare un'unica *cache*, consentendo a tutti gli utenti di trarre vantaggio dal fatto che può utilizzare un'unica *cache*, consentendo a tutti gli utenti di trarre vantaggio dal fatto che altri utenti abbiano richiesto in precedenza le stesse pagine da loro richieste. Andando verso il cuore di Internet, anche i fornitori di accesso (ISP) possono effettuare il caching. Notate che, nel secondo caso, gli utenti interni al sito sanno quale sia la macchina che effettua il caching per l'intero sito e configurano i propri browser in modo che si connettano direttamente a tale host, che viene a volte chiamato *proxy*. Al contrario, i siti che si connettono ad un

ISP non sono probabilmente consapevoli del fatto che avviene il caching delle pagine Web: semplicemente, tutte le richieste HTTP che provengono dai vari siti connessi al fornitore di accesso si trovano ad attraversare uno stesso router del fornitore stesso, e questo router può esaminare il messaggio per leggerne l'URL della pagina richiesta. Se la pagina è presente nella cache del router, viene restituita come risposta; se non lo è, il router inoltra la richiesta al server e esamina "al volo" la risposta del server nella direzione opposta, memorizzandola nella propria cache, nella speranza di poterla utilizzare per soddisfare una richiesta futura.

Indipendentemente dal luogo in cui vengono memorizzate le pagine, la possibilità di effettuare questo caching è sufficientemente importante da aver influito sul progetto del protocollo HTTP, in modo da rendere più semplice l'implementazione di tale strategia. Il problema è che chi implementa la cache deve essere certo di non rispondere con una versione della pagina che non sia la più recente. Ad esempio, il server assegna ad ogni pagina che invia al client (o ad una cache interposta fra il server e il client) una data di scadenza (il campo di intestazione *Expires*). La cache memorizza questa data e sa di non dover verificare nuovamente la pagina ogni volta che essa viene richiesta finché tale data non sia giunta. Da quel momento in poi (oppure se tale campo d'intestazione non era stato utilizzato), la cache può usare l'operazione *HEAD*, oppure l'operazione *GET* condizionata (cioè con la riga d'intestazione *If-Modified-Since*) per verificare se la copia della pagina in proprio possesso sia la più recente. Più in generale, esiste un insieme di "direttive per la cache" a cui tutti i meccanismi di caching lungo la catena richiesta/risposta devono attenersi, direttive che specificano se un documento possa essere memorizzato in una cache oppure no, per quanto tempo vi possa essere memorizzato, e così via.

9.2.3 Gestione della rete (SNMP)

Una rete è un sistema complesso, sia in termini del numero di nodi che sono coinvolti, sia in termini dell'insieme di protocolli che può essere eseguito in ogni singolo nodo. Anche se vi limitate a prendere in esame soltanto i nodi che appartengono ad un unico dominio amministrativo, come un'università, ci possono essere dozzine di router e centinaia, o anche migliaia, di host di cui tenere traccia. Se pensate alla quantità di informazioni di stato che viene gestita e manipolata da uno qualsiasi di questi nodi (ad esempio, le tabelle di traduzione degli indirizzi, le tabelle di instradamento, lo stato delle connessioni TCP, e così via), è facile cadere in depressione, immaginando di dover gestire tutto.

Non è difficile immaginare di voler conoscere lo stato dei vari protocolli in diversi nodi. Ad esempio, potrete voler tenere sotto controllo il numero di ricostruzioni di datagrammi IP che sono fallite, in modo da determinare se sia il caso di modificare il temporizzatore la cui scadenza innesca l'eliminazione dalla memoria dei datagrammi ricostruiti soltanto in parte. Come ulteriore esempio, potrete voler tenere traccia del traffico in vari nodi (ad esempio, il numero di pacchetti inviati o ricevuti), per determinare se debbano essere aggiunti alla rete nuovi router o nuove linee di connessione. Ovviamente, dovete anche controllare continuamente se avvengano guasti hardware o se il software si comporti in modo errato.

Quello che abbiamo appena descritto è il problema della gestione della rete, un problema che pervade l'intera architettura della rete. Dato che i nodi che vogliamo controllare sono distribuiti nella rete, l'unica vera possibilità che abbiamo è utilizzare la rete stessa per gestirla. Ciò significa che ci serve un protocollo che ci consenta di leggere, e possibilmente di scrivere, le varie informazioni di stato presenti nei diversi nodi della rete. Il protocollo più diffusamente utilizzato per questo scopo è SNMP (Simple Network Management Protocol).

SNMP è sostanzialmente un protocollo di tipo richiesta/risposta specializzato, che consente due tipi di messaggi di richiesta: GET e SET. Il primo è usato per ottenere una determinata informazione da un nodo, mentre il secondo si usa per memorizzare in un nodo una nuova informazione di stato (SNMP mette a disposizione anche una terza operazione, GET-NEXT, di cui parleremo più avanti). La discussione seguente è incentrata sull'operazione GET, perché è quella usata più di frequente.

SNMP si usa nel modo più intuitivo. Un amministratore di sistema interagisce con un programma che funge da client e che visualizza informazioni relative alla rete, solitamente mediante un'interfaccia grafica, che sostanzialmente svolge la stessa funzione di un browser Web. Ogni volta che l'amministratore seleziona una certa informazione che vorrebbe conoscere, il programma client usa SNMP per richiedere tale informazione al nodo in questione (SNMP si appoggia al protocollo UDP sottostante). Un server SNMP, in esecuzione in quel nodo, riceve la richiesta, rintraccia l'informazione appropriata e la restituisce al programma client, che quindi la mostra all'utente.

In questo scenario, altrimenti molto semplice, esiste una sola complicazione: come fa il client a segnalare con precisione quale sia l'informazione che vuole ottenere? E, analogamente, come fa il server a sapere quale variabile leggere dalla memoria per soddisfare la richiesta? La risposta è che il protocollo SNMP è strettamente correlato ad una specifica ausiliaria, detta MIB (management information base), che definisce le informazioni specifiche (le variabili MIB) che si possono richiedere ad un nodo di rete.

La versione attuale di MIB, denominata MIB-II, organizza le variabili in 10 diversi *gruppi*. Come vedrete, la maggior parte dei gruppi corrisponde ad uno dei protocolli descritti in questo libro e quasi tutte le variabili definite in ciascun gruppo vi dovrebbero essere familiari. Ad esempio:

- **System**: parametri generici del sistema (cioè del nodo) considerato nel suo insieme, tra i quali la collocazione fisica del nodo, il tempo trascorso dal suo avvio più recente e il nome del sistema stesso.
- **Interfaces**: informazioni relative alle interfacce di rete (adattatori) connessi al nodo, tra le quali l'indirizzo fisico di ogni interfaccia e il numero di pacchetti che sono stati trasmessi e ricevuti da ogni interfaccia.
- **Address translation**: informazioni relative al protocollo ARP (Address Resolution Protocol) e, in particolare, i contenuti della tabella di traduzione degli indirizzi memorizzata nel nodo.
- **IP**: variabili relative al protocollo IP, tra le quali la tabella di instradamento, il numero di datagrammi inoltrati con successo e un certo numero di statistiche relative alla ricostruzione dei datagrammi, oltre a conteggi relativi all'eliminazione di datagrammi per vari motivi.
- **TCP**: informazioni sulle connessioni TCP, come il numero di aperture di connessione attive e passive, il numero di operazioni di reset, il numero di temporizzazioni scadute, le impostazioni delle temporizzazioni, e così via. Le informazioni relative alle singole connessioni sono disponibili solamente fintantoché la connessione stessa esiste.
- **UDP**: informazioni in merito al traffico UDP, comprendenti il numero totale di datagrammi UDP che sono stati inviati e ricevuti.

Vi sono poi gruppi per ICMP, EGP, e per lo stesso protocollo SNMP. Il decimo gruppo viene usato per *media* diversi.

Tornando al problema di come il client possa indicare con precisione quale informazione vuole ottenere da un nodo, avere a disposizione un elenco di variabili MIB rappresenta solamente metà della soluzione. Rimangono aperti due problemi: per prima cosa, abbiano bisogno di una sintassi precisa, mediante la quale il client possa indicare quali variabili MIB vuole leggere; secondo, è necessaria una precisa rappresentazione dei valori restituiti dal server. Entrambi questi problemi sono stati risolti usando ASN.1.

Considerate prima il secondo problema. Come abbiamo già visto nel Capitolo 7, BER di ASN.1 definisce una rappresentazione per diversi tipi di dati, come i numeri interi. Il documento MIB definisce il tipo di ciascuna variabile, poi usa ASN.1 BER per codificare il valore contenuto in tale variabile prima di trasmetterlo attraverso la rete. Per quanto riguarda il primo problema, ASN.1 definisce anche uno schema per l'identificazione di oggetti, che non è stato descritto nel Capitolo 7. Il documento MIB usa questo sistema di identificazione per assegnare ad ogni variabile MIB un identificativo univoco: questi identificativi vengono espressi con una notazione a "punti" (*dot notation*), in modo non molto diverso dai nomi di dominio. Ad esempio, 1.3.6.1.2.1.4.3 è l'identificativo univoco ASN.1 per la variabile MIB, relativa al protocollo IP, denominata ipInReceives, che conta il numero di datagrammi IP che sono stati ricevuti dal nodo. In questo esempio, il prefisso 1.3.6.1.2.1 identifica la base di dati MIB (ricordate che gli identificativi di oggetti in ASN.1 sono stati pensati per qualsiasi possibile oggetto al mondo), il valore 4 corrisponde al gruppo IP, e il 3 finale indica la terza variabile all'interno di tale gruppo.

Di conseguenza, la gestione della rete funziona così. Il client SNMP inserisce nel messaggio di richiesta l'identificativo ASN.1 corrispondente alla variabile MIB che vuole leggere, e invia il messaggio al server. Il server, poi, mette in corrispondenza tale identificativo con una variabile locale (cioè con una locazione di memoria in cui si trova memorizzato il valore di tale variabile), recupera il valore attuale della variabile e usa il sistema ASN.1 BER per codificarlo prima di inviarlo in risposta al client.

C'è un ultimo dettaglio: molte variabili MIB sono tabelle o strutture di dati. Tali variabili composte forniscono la motivazione per l'operazione SNMP denominata GET-NEXT. Questa operazione, quando viene applicata ad un particolare identificativo di variabile, restituisce il valore di tale variabile insieme all'identificativo della variabile successiva, che può essere, ad esempio, la riga successiva nella tabella o il campo successivo nella struttura: un aiuto per la "navigazione" del client all'interno della tabella o struttura.

9.3 Applicazioni multimediali

Esattamente come le applicazioni tradizionali di cui abbiamo parlato nella sezione precedente, le applicazioni multimediali, come quelle di audio- e video-conferenza, necessitano di protocolli dello strato di applicazione. Gran parte dell'esperienza iniziale con cui sono stati progettati i protocolli per applicazioni multimediali era derivata dagli "strumenti di MBone" (*MBone tools*), applicazioni come vat e vic che erano state sviluppate per MBone, usando il multicast IP per consentire lo svolgimento di conferenze con più partecipanti. Inizialmente ciascuna applicazione implementò il proprio protocollo, ma divenne presto evidente che molte applicazioni multimediali avevano requisiti comuni, favorendo così lo sviluppo di un certo numero di protocolli di utilizzo generale nell'ambito di applicazioni multimediali.

Abbiamo già visto un protocollo ampiamente utilizzato per applicazioni multimediali quando abbiamo parlato di RSVP (Sezione 6.5.2), un protocollo che può essere utilizzato per

richiedere l'allocazione di risorse all'interno della rete in modo che la qualità di servizio (QoS) desiderata possa essere fornita all'applicazione. Oltre ad un protocollo di segnalazione per la qualità di servizio, molte applicazioni multimediali hanno anche bisogno di un protocollo di trasporto, con caratteristiche piuttosto differenti da quelle del protocollo TCP e con maggiori funzionalità di quelle offerte dal protocollo UDP. Il protocollo che è stato sviluppato per soddisfare queste necessità è chiamato RTP (Real-time Transport Protocol), descritto in seguito.

Una terza classe di protocolli di cui hanno bisogno molte applicazioni multimediali riguarda il *controllo di sessione*. Ad esempio, supponete di voler essere in grado di fare chiamate telefoniche attraverso Internet usando il protocollo IP. C'è bisogno di un meccanismo per annunciare al destinatario di tale chiamata che vogliamo parlare con lui, ad esempio inviando un messaggio ad un dispositivo multimediale che faccia squillare un campanello. Vorremo anche poter fornire il supporto per caratteristiche come l'inoltro di chiamata, la chiamata a tre partecipanti, e così via. I protocolli SIP (Session Initiation Protocol) e H.323 sono esempi di protocolli che gestiscono il controllo di sessione, e ne parleremo nella Sezione 9.3.2.

9.3.1 Real-time Transport Protocol (RTP)

Vi potrete chiedere perché un protocollo che, dal nome, sembra essere un "protocollo di trasporto" venga trattato in un capitolo che parla di problemi relativi allo strato di applicazione. Il motivo è che il protocollo RTP contiene una notevole quantità di funzioni che sono specifiche per le applicazioni multimediali. Inoltre, viene tipicamente eseguito al di sopra di uno dei protocolli di trasporto descritti nel Capitolo 5, il protocollo UDP, che fornisce alcune delle funzioni indipendenti dall'applicazione che vengono solitamente associate ad un protocollo di trasporto. Nonostante ciò, RTP viene anche considerato un protocollo di trasporto perché mette a disposizione servizi di tipo end-to-end comuni a diverse applicazioni (la maggior parte dei protocolli dello strato applicativo, come HTTP e SMTP, sono invece specifici per un'unica applicazione). Una cosa da notare, a questo punto, è la difficoltà di far rientrare i protocolli reali in un modello rigidamente a strati.

Prima di entrare nei dettagli di RTP, sarà di aiuto prendere in considerazione alcune delle applicazioni che lo usano. Le applicazioni multimediali vengono, a volte, suddivise in due categorie: applicazioni *per conferenza* e applicazioni *di flusso (streaming)*. Un esempio molto diffuso della prima categoria è vat, lo strumento di audioconferenza che viene spesso utilizzato in reti che supportino il multicast IP. Il pannello di controllo di una tipica conferenza effettuata con vat è mostrato in Figura 9.9. Un'altra applicazione per conferenze è vic, lo strumento di videoconferenza di cui abbiamo parlato nel Capitolo 1 e che abbiamo mostrato in Figura 1.1.

Le applicazioni *streaming* trasferiscono flussi audio o video da un server ad un client, e hanno esempi tipici in prodotti commerciali come Real Audio. Mancando l'interazione con l'utente, queste applicazioni impongono requisiti piuttosto diversi ai protocolli sottostanti. Dovrebbe essere chiaro, a questo punto, che i progettisti di un protocollo di trasporto per applicazioni multimediali si trovano di fronte ad un problema veramente complesso, qualora vogliano definire i requisiti in modo abbastanza generico da soddisfare le necessità di applicazioni molto diverse tra loro. Inoltre, devono fare molta attenzione all'interazione fra applicazioni diverse, come, ad esempio, la sincronizzazione di flussi audio e video. Vedremo in seguito come questi aspetti abbiano influenzato il progetto di RTP.

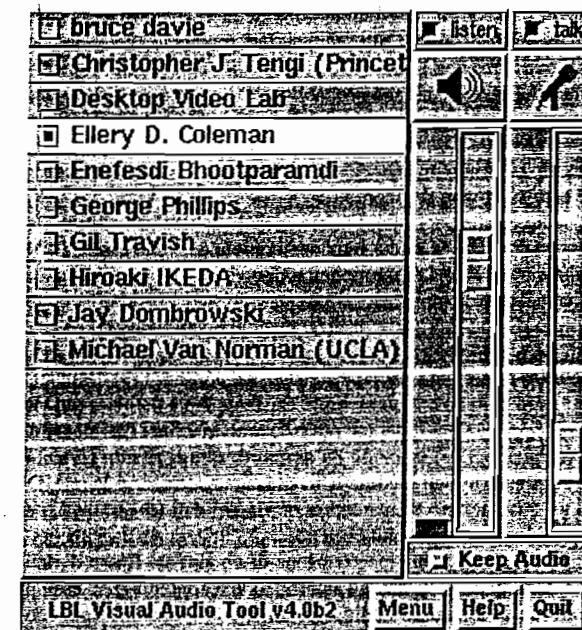


Figura 9.9 Interfaccia utente di un'audioconferenza effettuata con vat.

Il protocollo RTP deriva dal protocollo applicativo che fu originariamente inserito in vat e, ora, le nuove versioni di vat (e di molte altre applicazioni) funzionano usando RTP. Il protocollo RTP può essere eseguito al di sopra di molti protocolli di livello inferiore, ma solitamente si usa UDP, dando luogo alla pila di protocolli rappresentata in Figura 9.10.

Requisiti

Il requisito fondamentale per un protocollo multimediale di utilizzo generale è che consenta ad applicazioni simili di interoperate fra loro. Ad esempio, dovrebbe essere possibile effettuare un'audioconferenza utilizzando due applicazioni realizzate in modo indipendente l'una dall'altra: questo suggerisce immediatamente che le applicazioni farebbero meglio ad utiliz-

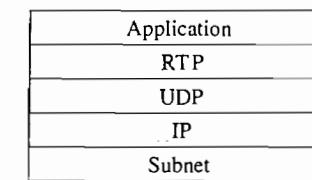


Figura 9.10 Pila di protocolli per applicazioni multimediali che usano il protocollo RTP.

zare lo stesso metodo di codifica e compressione della voce, altrimenti i dati inviati da un utente sarebbero incomprensibili all'altro utente. Dal momento che esistono diversi schemi di codifica vocale, ciascuno con il proprio compromesso tra qualità, requisiti di ampiezza di banda e costi di elaborazione, sarebbe probabilmente una pessima idea stabilire a priori di poter utilizzare un unico schema. Al contrario, il nostro protocollo dovrebbe mettere a disposizione un sistema mediante il quale il mittente possa indicare al destinatario quale schema di codifica vuole usare, gestendo una negoziazione che porti all'identificazione di uno schema sul quale le due parti si trovino d'accordo.

Come per l'audio, anche per il video esistono molti schemi di codifica diversi. Di conseguenza, possiamo vedere che la prima funzione comune che può essere fornita da RTP è la capacità di comunicare la scelta di uno schema di codifica. Notate che ciò serve anche allo scopo di identificare il tipo di applicazione (ad esempio, audio o video): una volta che sappiamo quale algoritmo di codifica viene utilizzato, sappiamo anche quale tipo di dati viene trasmesso.

Un altro requisito importante per RTP consiste nel consentire al destinatario di un flusso di dati di determinare la relazione temporale esistente fra i dati ricevuti. Ricordate, dalla Sezione 6.5, che le applicazioni in tempo reale devono inserire i dati ricevuti in un *buffer di riproduzione (playback buffer)* in modo da ammortizzare il ritardo (*jitter*) che può essere stato introdotto nel flusso di dati durante la trasmissione attraverso la rete. Di conseguenza, sarà necessario un qualche tipo di marcatura oraria dei dati (*timestamp*), per consentire al destinatario di riprodurli nel momento corretto.

Un problema correlato alla temporizzazione di un singolo flusso di dati è quello della sincronizzazione di diversi tipi di dati all'interno di una conferenza. L'esempio più scontato è la sincronizzazione di un flusso audio e di un flusso video che vengono generati dalla medesima sorgente. Come vedremo in seguito, si tratta di un problema un po' più complesso di quello relativo alla determinazione dell'istante di corretta riproduzione per un singolo flusso.

Un'altra importante funzione che deve essere fornita è un'indicazione della perdita di pacchetti. Notate che un'applicazione con vincoli di latenza molto stringenti non può, in generale, utilizzare un protocollo affidabile come TCP, perché la ritrasmissione di dati in caso di perdita di pacchetti farebbe probabilmente arrivare il pacchetto troppo tardi perché possa essere utilizzato. Di conseguenza, l'applicazione deve poter gestire la perdita di pacchetti ed il primo passo perché possa avvenire questa gestione è la notifica del fatto che un pacchetto sia andato perduto. Ad esempio, un'applicazione video che usa la codifica MPEG intraprenderà azioni diverse in caso di perdita di un pacchetto, in relazione al fatto che il pacchetto provenisse da un frame *I*, da un frame *B* o da un frame *P*.

Dal momento che, generalmente, le applicazioni multimediali non vengono eseguite al di sopra di TCP, non hanno a disposizione neppure le caratteristiche di TCP che consentono di impedire l'insorgere di congestione (come descritto nella Sezione 6.3). Tuttavia, molte applicazioni multimediali sono in grado di reagire alla congestione, ad esempio modificando i parametri dell'algoritmo di codifica in modo da ridurre la banda utilizzata. Ovviamente, perché questo meccanismo possa funzionare, il destinatario deve avvertire il mittente che stanno avvenendo perdite di pacchetti, in modo che il mittente possa modificare i propri parametri di codifica.

Un'altra funzione comune a più applicazioni multimediali è la segnalazione del confine fra diversi frame. Un frame, in questo contesto, ha un significato che dipende dall'applicazione. Ad esempio, potrebbe essere utile segnalare ad un'applicazione video che un certo

9.3 Applicazioni multimediali

insieme di pacchetti corrisponde ad un unico frame. In un'applicazione audio, è utile contrassegnare l'inizio di una "raffica di parole" (*talkspurt*), un insieme di suoni o parole seguite da silenzio. Il ricevente può così identificare i silenzi compresi tra raffiche consecutive e usarli per spostare, eventualmente, il punto di riproduzione, in base all'osservazione che piccoli accorciamenti o allungamenti degli spazi tra le parole non sono percepibili dagli utenti, mentre l'accorciamento o l'allungamento delle parole stesse è percepibile e arreca disturbo.

Un'ultima funzione che potremmo voler introdurre nel protocollo è un modo per identificare i mittenti che sia più familiare, per gli utenti, di un indirizzo IP. Programmi come *vat* e *vic* visualizzano nel proprio pannello di controllo stringhe come *Joe User* (*user@domain.com*), per cui il protocollo applicativo deve consentire l'associazione tra tali stringhe e flussi di dati.

Oltre alle funzionalità che abbiamo richiesto al nostro protocollo, notiamo un requisito addizionale: il protocollo dovrebbe fare un uso ragionevolmente efficiente della banda. Detto in altro modo, non vogliamo utilizzare un sacco di bit aggiuntivi da inviare insieme ad ogni pacchetto sotto forma di una lunga intestazione, perché i pacchetti di dati audio, che sono fra i dati multimediali più comuni, tendono ad essere piccoli, in modo da ridurre il tempo necessario per costruirli a partire da campioni di segnale audio. Pacchetti audio di grandi dimensioni significherebbero lunghe latenze, per effetto della generazione dei pacchetti stessi, cosa che ha un effetto negativo sulla qualità percepita per le conversazioni (ricordate che questo fu uno dei fattori determinanti nella scelta della lunghezza delle celle ATM). Dal momento che i pacchetti di dati sono, per se stessi, brevi, l'uso di una grande intestazione starebbe a significare che una quantità rilevante dell'ampiezza di banda della linea di collegamento sarebbe utilizzata dalle intestazioni, riducendo così la capacità disponibile per i dati "utili". Vedremo diversi aspetti del progetto di RTP che sono stati influenzati dalla necessità di usare intestazioni brevi.

Dettagli di RTP

Ora che abbiamo visto l'elenco, relativamente lungo, dei requisiti per il nostro protocollo dedicato alla multimedialità e operante nello strato applicativo, dedichiamoci ai dettagli del protocollo che è stato specificato per soddisfare tali requisiti. Il protocollo RTP fu sviluppato da IETF ed è ampiamente utilizzato; lo standard attuale per RTP definisce una coppia di protocolli, RTP e RTCP (Real-time Transport Control Protocol). Il primo è utilizzato per il trasferimento di dati multimediali, mentre il secondo viene usato per inviare periodicamente informazioni di controllo associate ad un certo flusso di dati. Quando si utilizza UDP come protocollo di trasporto, il flusso di dati RTP e l'associato flusso di controllo RTCP usano, al livello di trasporto, porte consecutive. I dati RTP usano una porta con numero pari e le informazioni di controllo RTCP usano il numero di porta successivo (dispari).

Poiché il protocollo RTP è stato progettato per fornire supporto ad un'ampia gamma di applicazioni, dispone di un meccanismo flessibile mediante il quale si possono sviluppare nuove applicazioni senza dover ripetutamente rivisitare il protocollo stesso. Per ogni classe di applicazioni (ad esempio, audio), RTP definisce un *profilo* e uno o più *formati*. Il profilo contiene informazioni che garantiscono una base comune di comprensione per i campi dell'intestazione RTP per la relativa classe di applicazioni, come diverrà evidente quando esamineremo l'intestazione in dettaglio. La specifica del formato spiega come vanno interpretati i dati che seguono l'intestazione RTP. Ad esempio, l'intestazione RTP potrebbe essere seguita, semplicemente, da una sequenza di byte, ciascuno dei quali rappresenta un singolo campione audio relativo ad un istante che segue il precedente di un intervallo predefinito. In alternativa, il formato dei dati potrebbe essere molto più complesso; un flusso video codifica-

to mediante MPEG, ad esempio, avrà bisogno di una struttura piuttosto articolata per poter rappresentare tutti i tipi di informazioni necessarie.

Il progetto del protocollo RTP si basa su un principio architettonale noto come ALF (Application Level Framing, *struttura dei frame al livello applicativo*), che venne ideato da Clark e Tennenhouse nel 1990 come nuova modalità per progettare protocolli per le applicazioni multimediali emergenti, dopo aver capito che tali nuove applicazioni sarebbero difficilmente state servite da protocolli esistenti, come TCP, e che quindi non sarebbero nemmeno state servite adeguatamente da un qualsiasi tipo di protocollo "che andasse bene per tutti". Il nucleo di questo principio consiste nel ritenere che un'applicazione sia la migliore candidata a comprendere le proprie necessità. Ad esempio, un'applicazione video MPEG sa bene come reagire alla perdita di frame, e reagisce diversamente alla perdita di un frame *I* o di un frame *B*. La stessa applicazione è anche in grado di segmentare al meglio i dati per la trasmissione: ad esempio, è meglio inviare i dati di frame diversi in datagrammi diversi, in modo che la perdita di un pacchetto danneggi un unico frame, e non due. È per questo motivo che RTP affida così tanti dettagli del protocollo al profilo e ai documenti di formato che sono specifici di una particolare applicazione.

Formato dell'intestazione

La Figura 9.11 mostra il formato dell'intestazione usata dal protocollo RTP. I primi 12 byte sono sempre presenti, mentre gli identificativi delle sorgenti che forniscono dati (*contributing source identifiers*) vengono usati soltanto in alcune circostanze. Dopo questa intestazione ci possono essere estensioni dell'intestazione opzionali, come descritto in seguito. Infine, l'intestazione è seguita dal carico utile di RTP, il cui formato è determinato dall'applicazione. Lo spirito di questa intestazione è che debba contenere soltanto quei campi che saranno utilizzati con buona probabilità da molte diverse applicazioni, perché qualsiasi cosa che sia molto specifica di una singola applicazione sarà veicolata in modo più efficiente dal carico utile di RTP relativo a quella sola applicazione.

I primi due bit sono un identificativo della versione, che contiene il valore 2 nella versione di RTP attualmente in uso. Potreste essere indotti a pensare che i progettisti di questo protocollo siano stati un po' grossolani a ritenere che 2 soli bit siano sufficienti a rappresentare tutte le future versioni di RTP, ma ricordate che i bit sono considerati molto costosi nell'intestazione di RTP. Inoltre, l'uso dei profili per diverse applicazioni rende meno proba-

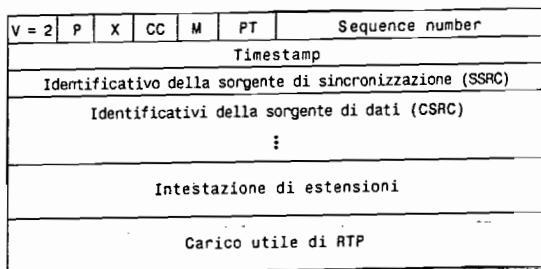


Figura 9.11 Formato dell'intestazione di RTP.

bile che siano necessarie molte revisioni al protocollo RTP di base. In ogni caso, se si dovesse ritenere necessaria un'ulteriore versione di RTP successiva alla versione 2, sarebbe possibile modificare il formato dell'intestazione in modo da rendere possibili, in futuro, altre versioni. Ad esempio, una nuova intestazione RTP con il valore 3 nel campo della versione potrebbe avere un campo di "sottoversione" da qualche parte, all'interno dell'intestazione.

Il bit successivo (P) è il bit di "padding", che viene impostato al valore 1 quando al carico utile di RTP sono stati per qualche motivo aggiunti byte di *padding*, cosa che può avvenire per riempire un blocco di una determinata dimensione, come richiesto, ad esempio, da un algoritmo di cifratura. In tal caso, all'intestazione del protocollo di livello inferiore (ad esempio, UDP) viene comunicata la lunghezza totale dell'intestazione RTP, dei relativi dati e dei byte di padding, e l'ultimo byte di padding contiene il conteggio di quanti siano i byte da ignorare, come illustrato in Figura 9.12. Notate che questa soluzione del problema del padding non richiede, nell'intestazione di RTP, un campo che indichi la lunghezza del pacchetto (sempre nell'ottica di fare in modo che l'intestazione sia di piccole dimensioni); nel caso, molto comune, in cui non ci sia bisogno di padding, la lunghezza viene determinata dal protocollo di livello inferiore.

Il bit di estensione (X) viene utilizzato per indicare la presenza di un'intestazione di estensione, che viene definita da una specifica applicazione e segue l'intestazione principale. Tali intestazioni sono usate raramente, perché generalmente è possibile specificare, per una particolare applicazione, un'intestazione relativa al carico utile come parte della definizione del formato del carico utile stesso.

Il bit X è seguito da un campo a 4 bit (CC) che conta il numero di "sorgenti di dati" (*contributing sources*), se ve ne sono all'interno dell'intestazione. Di tali sorgenti di dati parleremo in seguito.

Abbiamo già messo in evidenza come vi sia spesso bisogno di una qualche forma di identificazione dei frame: a questo provvede il bit di marcatura (*marker bit*, M), che, ad esempio, potrebbe venire impostato al valore 1 all'inizio di una "raffica di parole". Di seguito, troviamo il campo per il tipo di carico utile (*payload type*, PT), a 7 bit, che indica quale tipo di dati multimediali siano contenuti nel pacchetto. Un possibile utilizzo di questo campo potrebbe essere quello di consentire ad un'applicazione di commutare da uno schema di codifica ad un altro in base ad informazioni relative alla disponibilità di risorse nella rete o ad informazioni di *feedback* in merito alla qualità dell'applicazione stessa. L'utilizzo specifico del bit di marcatura e del campo che indica il tipo di carico utile viene determinato con precisione dal profilo dell'applicazione.

Notate che il tipo del carico utile non viene generalmente utilizzato come chiave di demultiplexing per smistare i dati ad applicazioni diverse (o a diversi flussi all'interno della stessa applicazione, ad esempio i flussi audio e video di una videoconferenza), perché tale funzione viene solitamente svolta ad un livello inferiore (ad esempio, il protocollo UDP lo fa nel modo che abbiamo visto nella Sezione 5.1). Di conseguenza, due flussi multimediali che usano RTP utilizzano tipicamente due diverse porte UDP.

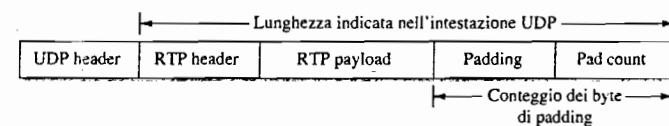


Figura 9.12 Aggiunta di byte di *padding* in un pacchetto RTP.

Il numero di sequenza (*sequence number*) viene utilizzato per consentire al ricevitore di un flusso RTP di rilevare pacchetti mancanti o fuori ordine: è sufficiente che la sorgente incrementi di uno il valore del campo per ogni pacchetto trasmesso. Notate che il protocollo RTP non intraprende alcuna azione nel momento in cui rileva la perdita di un pacchetto, diversamente dal protocollo TCP, che pone in atto azioni correttive per la perdita (innescando una ritrasmissione) e interpreta la perdita del pacchetto come un segnale di congestione (provocando, potenzialmente, la riduzione della finestra). Piuttosto, viene demandato all'applicazione il compito di decidere cosa fare quando viene perso un pacchetto, perché è molto probabile che tale decisione sia strettamente correlata all'applicazione stessa. Ad esempio, un'applicazione video potrebbe decidere che la cosa migliore da fare quando viene perduto un pacchetto sia una seconda riproduzione dell'ultimo frame ricevuto correttamente. Reagendo alla perdita di un pacchetto, altre applicazioni potrebbero, invece, decidere di modificare i propri algoritmi di codifica per ridurre l'ampiezza di banda richiesta, ma questa non è una funzione che venga svolta da RTP: non avrebbe senso fare in modo che il protocollo RTP possa decidere di ridurre la velocità di trasmissione della sorgente, perché ciò potrebbe rendere l'applicazione inservibile.

La funzione del campo *timestamp* ("indicazione oraria") è quella di consentire al ricevitore di riprodurre i campioni nel momento opportuno e di consentire la sincronizzazione di diversi flussi multimediali. Poiché diverse applicazioni possono aver bisogno di una diversa precisione nella misura del tempo, il protocollo RTP non specifica l'unità con la quale vada misurato il tempo stesso: al contrario, il campo *timestamp* è semplicemente il conteggio di "eventi equidistanti nel tempo" (*ticks*), e il tempo che intercorre tra due di tali eventi dipende dalla codifica che viene utilizzata. Ad esempio, un'applicazione audio che campiona i dati una volta ogni 125 µs potrebbe usare tale valore come intervallo fra due tick. La precisione della misura del tempo è uno dei dettagli che viene specificato nel profilo RTP o nel formato del carico utile per una determinata applicazione.

Il valore dell'indicazione oraria nel pacchetto è un numero che rappresenta l'istante in cui è stato generato il *primo* campione presente nel pacchetto; non rappresenta in alcun modo l'orario nel senso comune del termine, hanno significato solamente le differenze tra diverse indicazioni orarie. Ad esempio, se l'intervallo di campionamento è 125 µs e il primo campione del pacchetto $n + 1$ è stato generato 10 ms dopo il primo campione del pacchetto n , allora il numero di intervalli di campionamento che separano i due campioni è

$$\begin{aligned} \text{TimeBetweenPackets/TimePerSample} &= (10 \times 10^{-3}) / (125 \times 10^{-6}) \\ &= 80 \end{aligned}$$

Ipotizzando che la precisione delle indicazioni orarie sia uguale all'intervallo di campionamento, il valore del *timestamp* nel pacchetto $n + 1$ sarebbe uguale a 80 più il valore del *timestamp* presente nel pacchetto n . Notate che può accadere che siano stati inviati meno di 80 campioni, per effetto di tecniche di compressione quali la rilevazione dei silenzi, ciò nonostante l'indicazione oraria consente al ricevitore di riprodurre i campioni con la relazione temporale corretta.

L'identificativo della sorgente di sincronizzazione (SSRC, *synchronization source*) è un numero a 32 bit che identifica univocamente una singola sorgente di un flusso RTP. In una conferenza multimediale, ogni fonte di flussi sceglie un valore di SSRC casuale, con il vincolo di dover risolvere eventuali conflitti nell'improbabile caso che due fonti scelgano il medesimo valore. Rendendo l'identificativo della sorgente un'entità diversa dall'indirizzo di rete

o di trasporto della sorgente stessa, il protocollo RTP si garantisce l'indipendenza dal protocollo di livello inferiore, oltre che consentire ad un singolo nodo avente diverse sorgenti di flusso (ad esempio, più telecamere) di distinguere tra di esse. Quando un singolo nodo genera più flussi di tipo diverso (ad esempio, audio e video), non è necessario che usi il medesimo valore di SSRC in ogni flusso, perché nel protocollo RTCP (descritto in seguito) esistono meccanismi che consentono la sincronizzazione fra più flussi.

Il campo per le sorgenti di dati (CSRC, *contributing source*) viene usato soltanto quando un certo numero di flussi RTP attraversa un "mixer" (*miscelatore*), che viene utilizzato per ridurre i requisiti di banda di una conferenza ricevendo dati da più sorgenti e trasmettendoli come un singolo flusso. Ad esempio, i flussi audio di più persone che parlano contemporaneamente possono essere decodificati e ricodificati come un unico flusso audio: in questo caso, il mixer indica se stesso come la sorgente di sincronizzazione, ma elenca anche le sorgenti di dati, che sono i valori del campo SSRC degli oratori che hanno contribuito al pacchetto in esame.

Protocollo di controllo

Il protocollo RTCP mette a disposizione un flusso di controllo associato ad un flusso di dati per applicazioni multimediali. Tale flusso di controllo fornisce tre principali funzionalità:

- Segnalazioni (*feedback*) sulle prestazioni dell'applicazione e sulla rete.
- Correlazione e sincronizzazione di diversi flussi di dati multimediali che provengono dalla stessa sorgente.
- Indicazioni dell'identità di una sorgente, da visualizzare su una interfaccia utente (ad esempio, l'interfaccia del programma vat mostrata in Figura 9.9).

La prima funzionalità può essere utile per applicazioni a velocità variabile, che possono usare i dati relativi alle prestazioni per decidere di usare uno schema di compressione più aggressivo per ridurre la congestione, oppure di inviare un flusso di qualità più elevata quando c'è poca congestione. Inoltre, può essere utile per diagnosticare problemi di rete.

Potreste pensare che la seconda funzionalità fosse già fornita dall'identificativo della sorgente di sincronizzazione del protocollo RTP, ma in realtà non è così. Come abbiamo già notato, diverse telecamere che si trovano in un singolo nodo possono avere valori di SSRC diversi; inoltre, non esiste il vincolo di utilizzare il medesimo valore di SSRC per due flussi, audio e video, provenienti dallo stesso nodo. Dal momento che possono accadere collisioni nella scelta dei valori di SSRC, può essere necessario modificare il valore di SSRC associato ad un flusso. Per gestire questi problemi, il protocollo RTCP usa il concetto di "nome canonico" (CNAME), che viene assegnato ad una fonte di informazioni, la quale a sua volta, usando i meccanismi tipici di RTCP, viene associata ai vari valori di SSRC che possono essere usati da quella fonte.

La semplice correlazione di due flussi è soltanto una parte del problema della sincronizzazione fra flussi. Poiché flussi diversi possono avere una cognizione del "tempo" completamente diversa (con diversi intervalli fra gli "istanti di tempo" e anche diversi valori per l'accuratezza della loro misura), ci deve essere un modo per sincronizzare accuratamente i flussi fra loro. Il protocollo RTCP risolve questi problemi.

Il protocollo RTCP definisce un certo numero di tipi di pacchetto diversi, fra i quali:

- notifiche (*report*) della sorgente, che consente alle sorgenti che sono attive in una sessione di segnalare statistiche di trasmissione e di ricezione;

- notifiche del ricevente, usate dai riceventi che non siano anche sorgenti per segnalare statistiche di ricezione;
- descrizioni delle sorgenti, che veicolano i CNAME e le altre informazioni di descrizione della sorgente;
- pacchetti di controllo specifici per un'applicazione.

Questi diversi tipi di pacchetti RTCP vengono trasmessi mediante il protocollo di livello inferiore, che, come abbiamo già fatto notare, è tipicamente il protocollo UDP. Più pacchetti RTCP possono essere aggregati in un unico PDU del protocollo di livello inferiore, ma in ogni PDU devono essere presenti almeno due pacchetti: un pacchetto di notifica (*report*) e un pacchetto di descrizione di sorgente. Possono, poi, essere inclusi nel PDU anche altri pacchetti, fino al limite di dimensione imposto dai protocolli sottostanti.

Prima di osservare con maggiore dettaglio i contenuti di un pacchetto RTCP, notiamo che esiste un potenziale problema ogni qualvolta un membro di un gruppo multicast trasmette periodicamente traffico di controllo: se non prendiamo qualche misura per limitarlo, questo traffico di controllo è in grado di impegnare una banda significativa. Ad esempio, in un'audioconferenza è molto probabile che non vi siano più di due o tre sorgenti che inviano dati audio contemporaneamente, perché non c'è motivo per cui tutti debbano parlare nello stesso istante, ma per quanto riguarda il traffico di controllo non esiste questa limitazione "sociale", per cui potrebbero insorgere seri problemi in una conferenza con migliaia di partecipanti. Per gestire questi problemi, il protocollo RTCP prevede un insieme di meccanismi mediante i quali i partecipanti riducono proporzionalmente la frequenza delle proprie notifiche all'aumentare del numero dei partecipanti. Queste regole sono piuttosto complesse, ma l'obiettivo di fondo è questo: limitare la quantità complessiva del traffico RTCP ad una piccola percentuale (tipicamente il 5%) del traffico relativo ai dati RTP. Per realizzare questo obiettivo, i partecipanti dovrebbero conoscere quanta ampiezza di banda viene probabilmente utilizzata in un dato istante (ad esempio, la quantità necessaria ad inviare tre flussi audio) ed il numero di partecipanti. La prima informazione viene appresa mediante fonti esterne al protocollo RTP (che prendono il nome di "gestione della sessione", di cui parleremo alla fine della sezione), mentre la seconda informazione proviene dalle notifiche RTCP degli altri partecipanti. Dal momento che le notifiche RTCP possono essere inviate con frequenze molto basse, è possibile avere solamente una valutazione approssimativa del numero di partecipanti in un dato istante, ma ciò è tipicamente sufficiente. Ancora, è preferibile assegnare una maggior ampiezza di banda, relativamente al protocollo RTCP, alle sorgenti attive, nell'ipotesi che la maggior parte dei partecipanti vogliano ricevere notifiche da esse, per sapere, ad esempio, chi stia parlando.

Dopo aver determinato quale ampiezza di banda può destinare al traffico RTCP, ciascun partecipante predispone la trasmissione delle proprie notifiche periodiche alla velocità opportuna. Le notifiche provenienti da una sorgente e quelle provenienti da un ricevente differiscono solamente per il fatto che le prime contengono alcune informazioni aggiuntive relative alla sorgente, ma entrambi i tipi di notifica forniscono informazioni in merito ai dati che sono stati ricevuti da tutte le sorgenti nel più recente intervallo di notifica.

Le informazioni aggiuntive presenti in una notifica di sorgente sono

- un'indicazione oraria contenente l'istante di tempo (effettivo) nel quale è stata generata la notifica;
- l'indicazione oraria (*timestamp*) RTP corrispondente all'istante in cui è stata generata la notifica;

- il conteggio cumulativo dei pacchetti e dei byte inviati da questa sorgente dal momento in cui ha iniziato a trasmettere.

Noteate che le prime due informazioni possono essere utilizzate per consentire la sincronizzazione di diversi flussi multimediali provenienti dalla medesima sorgente, anche se tali flussi usano diversi intervalli per misurare il tempo nei dati RTP, perché forniscono il mezzo per convertire il tempo effettivo in indicazioni orarie RTP.

Sia le notifiche di sorgente sia quelle di ricevente contengono un blocco di dati per ogni sorgente da cui hanno ricevuto dati nell'intervallo di tempo relativo alla notifica stessa. Ciascun blocco contiene, per ognuna di tali sorgenti, le seguenti informazioni:

- il valore di SSRC;
- la frazione di pacchetti di dati provenienti da tale sorgente che sono stati perduti dal momento in cui è stata inviata la notifica precedente (calcolata confrontando il numero di pacchetti ricevuti con il numero di pacchetti che ci si aspettava di ricevere; quest'ultimo valore può essere determinato dai numeri di sequenza di RTP);
- il numero totale di pacchetti provenienti da tale sorgente che sono stati perduti a partire dalla prima volta in cui si ha avuto notizia della sorgente stessa;
- il più elevato numero di sequenza ricevuto da tale sorgente (esteso a 32 bit per tenere conto del ritorno al valore iniziale del numero di sequenza);
- il valore stimato per il jitter fra due arrivi (calcolato confrontando la spaziatura temporale fra gli istanti di arrivo dei pacchetti ricevuti con la spaziatura prevista al momento della trasmissione);
- l'ultimo valore di *timestamp* ricevuto da tale sorgente mediante il protocollo RTCP;
- tempo trascorso dall'ultima notifica di sorgente ricevuta da tale sorgente mediante il protocollo RTCP.

Come potete facilmente immaginare, i destinatari di queste informazioni possono dedurre tutto ciò che serve sapere in merito allo stato della sessione. In particolare, possono capire se altri destinatari stanno ricevendo i dati di una certa sorgente con una qualità superiore a quella che stanno sperimentando, cosa che potrebbe essere indice della necessità di una prenotazione di risorse oppure di un problema di rete che richiede attenzione. Inoltre, se una sorgente si accorge che molti ricevitori stanno osservando un'elevata perdita dei propri pacchetti, potrebbe decidere di ridurre la propria velocità di trasmissione oppure di usare uno schema di codifica più resistente alle perdite di pacchetti.

L'ultimo aspetto del protocollo RTCP che ci preme prendere in esame è il pacchetto di descrizione di una sorgente. Tale pacchetto contiene, come minimo, l'identificativo SSRC della sorgente e il nome canonico (CNAME) della sorgente stessa. Il nome canonico viene generato in modo che tutte le applicazioni che generano flussi che potrebbero venir sincronizzati (ad esempio, flussi audio e video provenienti dallo stesso utente ma generati separatamente) sceglieranno il medesimo CNAME, pur potendo scegliere diversi valori di SSRC. Ciò consente ad un ricevitore di identificare i flussi di dati che provengono dalla stessa sorgente. Il formato più comune di CNAME è *user@host*, dove *host* è il nome della macchina sorgente, completamente specificato con il proprio nome di dominio. Di conseguenza, un'applicazione eseguita dall'utente di nome *jdoe* sulla macchina *cicada.cs.princeton.edu* userà come CNAME la stringa *jdoe@cicada.cs.princeton.edu*. Il numero di byte usato da questa rappresentazione, che è elevato e fortemente variabile, la renderebbe una pessima scelta come

formato per l'identificativo SSRC, perché quest'ultimo viene trasmesso all'interno di ogni pacchetto e deve essere elaborato in tempo reale. Consentire che i nomi canonici (CNAME) vengano associati ai valori di SSRC in messaggi RTCP periodici permette di usare un formato efficiente e compatto per SSRC.

Nel pacchetto di descrizione della sorgente possono essere contenute anche altre informazioni, come il vero nome dell'utente e il suo indirizzo di posta elettronica: queste informazioni vengono utilizzate nelle interfacce utente delle applicazioni per visualizzare i partecipanti e consentire un contatto con essi, ma sono meno essenziali per l'operatività di RTP di quanto lo sia il valore di CNAME.

9.3.2 Controllo di sessione e controllo di chiamata (SDP, SIP, H.323)

Per comprendere alcuni dei problemi relativi al controllo di sessione, supponete di voler effettuare una videoconferenza ad una certa ora, rendendola disponibile ad un ampio numero di partecipanti. Magari avete deciso di codificare il flusso video usando lo standard MPEG-2, di usare l'indirizzo IP multicast 224.1.1.1 per la trasmissione dei dati e di inviarli usando il protocollo RTP al di sopra di UDP verso la porta numero 4000. Come fate a rendere disponibili tali informazioni a tutti i partecipanti previsti? Un modo sarebbe quello di spedire tutte le informazioni con un messaggio di posta elettronica, ma la cosa ideale sarebbe l'esistenza di un formato e di un protocollo standard per diffondere questo tipo di informazioni. IETF ha un gruppo di lavoro (il gruppo Multiparty Multimedia Session Control) che ha definito protocolli proprio a questo scopo, fra i quali troviamo:

- SDP (Session Description Protocol)
- SAP (Session Announcement Protocol)
- SIP (Session Initiation Protocol)
- SCCP (Simple Conference Control Protocol)

Potreste pensare che, per un problema così semplice, questi protocolli siano troppi, ma il problema ha molte sfaccettature e deve essere risolto in molte situazioni diverse. Ad esempio, annunciare che una certa sessione di conferenza sta per essere resa disponibile sulla rete MBone (annuncio che si potrebbe fare utilizzando SDP e SAP) è molto diverso che tentare di effettuare una chiamata telefonica in una internetwork verso un certo utente in un particolare istante (cosa che potrebbe essere realizzata usando SDP e SIP). Nel primo caso potete pensare di aver raggiunto il vostro scopo dopo aver inviato ad un indirizzo multicast ben noto tutte le informazioni relative alla sessione, in un formato standard. Nel secondo caso dovete localizzare un utente (o più utenti), fargli pervenire un messaggio che annuncia la vostra volontà di effettuare una conversazione (l'analogo dello squillo del telefono) e forse negoziare fra le parti una codifica audio adeguata. Per prima cosa esamineremo il protocollo SDP, che è comune a molte applicazioni, poi il protocollo SIP, che sta diffondendosi per un certo numero di applicazioni interattive, come la telefonia in Internet.

Session Description Protocol (SDP)

Il protocollo SDP è un protocollo piuttosto generico che può essere utilizzato in molte situazioni diverse, e veicola le seguenti informazioni:

- il nome e lo scopo della sessione;

- l'istante di inizio e di termine della sessione;
- i tipi di *mediq* che compongono la sessione (ad esempio, audio, video, ...);
- informazioni dettagliate necessarie per ricevere la sessione (ad esempio, l'indirizzo multicast al quale verranno inviati i dati, il protocollo di trasporto utilizzato, i numeri di porta, gli schemi di codifica).

Il protocollo SDP fornisce queste informazioni in codifica ASCII, usando una serie di righe di testo, ciascuna nella forma "*<tipo>=<valore>*". Un esempio di messaggio SDP renderà chiari molti aspetti.

```
v=0
o=larry 2890844526 2890842807 IN IP4 10.0.1.5
s=Networking 101
i=A class on computer networking
u=http://www.cs.princeton.edu/
e=larry@cs.princeton.edu
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
m=audio 49170 RTP/AVP 0
m=video 51372 RTP/AVP 31
m=application 32416 udp wb
```

Notate che SDP, come HTML è abbastanza semplice da leggere per una persona umana, ma ha regole di impaginazione molto rigide che rendono possibile, per un calcolatore, l'interpretazione non ambigua. Ad esempio, la specifica di SDP definisce tutti i "tipi" possibili di informazioni che possono comparire nel messaggio, l'ordine in cui devono essere presenti e il formato e le parole riservate relativi ad ogni tipo che viene definito.

La prima cosa da notare è che ogni "tipo" di informazione è identificato da un singolo carattere. Ad esempio, la riga *v=0* indica che la "versione" ha il valore zero, cioè che il messaggio è conforme alla versione zero di SDP. La riga successiva identifica l'"origine" della sessione e contiene informazioni sufficienti ad identificare univocamente la sessione stessa: *larry* è il nome dell'utente che ha creato la sessione e *10.0.1.5* è l'indirizzo IP del suo calcolatore; il numero che segue *larry* è un identificativo di sessione che viene scelto in modo da essere unico all'interno di quel calcolatore ed è seguito da un numero di "versione" per l'annuncio SDP (se l'informazione relativa alla sessione venisse aggiornata da un messaggio successivo, il numero di versione verrebbe incrementato).

Le tre righe successive (*s*, *i* e *u*) forniscono il nome della sessione, una descrizione della sessione e un *identificativo omogeneo di risorsa* (URI, *uniform resource identifier*) per la sessione, tutte informazioni utili perché un utente possa decidere se partecipare o meno alla sessione stessa. Tali informazioni possono essere visualizzate nell'interfaccia utente di uno strumento che gestisca un "elenco di sessioni" (*session directory*), che mostri gli eventi in corso e futuri che siano stati annunciati mediante il protocollo SDP. La riga successiva (*e =...*) contiene l'indirizzo di posta elettronica della persona da contattare in riferimento alla sessione. La Figura 9.13 mostra una schermata di uno strumento di gestione delle sessioni, chiamato *sdr*, con la descrizione di alcune sessioni che erano state annunciate nel momento in cui è stata catturata l'immagine.

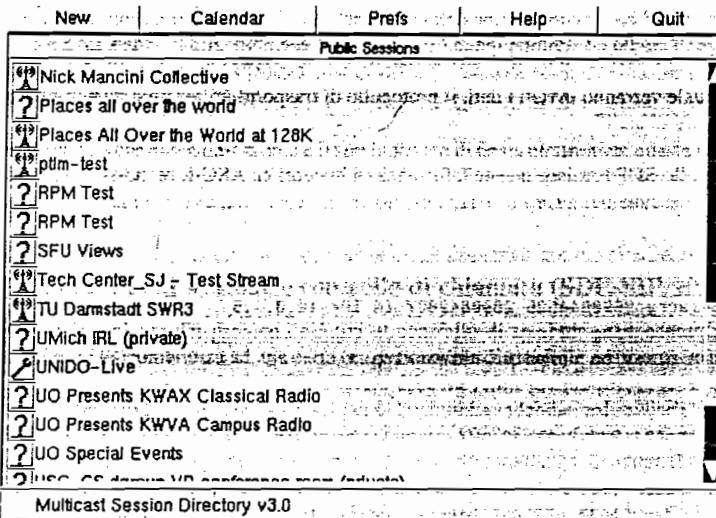


Figura 9.13 Uno strumento di gestione delle sessioni che mostra informazioni desunte da messaggi SDP.

Successivamente, troviamo i dettagli tecnici che consentono ad un programma applicativo di partecipare alla sessione. La riga che inizia con `c=...` fornisce l'indirizzo IP di tipo multicast al quale verranno inviati i dati della sessione: un utente che voglia ricevere la sessione deve unirsi a tale gruppo multicast. Poi, vediamo gli istanti iniziale e finale della sessione (codificati come numeri interi nel formato del protocollo NTP, *Network Time Protocol*). Infine, possiamo ottenere informazioni in merito ai *media* utilizzati nella sessione. Questa sessione rende disponibili tre tipi di *media*: audio, video e un'applicazione di lavagna condivisa denominata *wb* (*whiteboard*, "lavagna"). Per ognuno di questi *media* è presente una riga di informazione con il seguente formato:

```
m=<media> <port> <transport> <format>
```

I tipi di *media* sono auto-esplicativi e i numeri di porta sono, in ciascuno dei tre casi, dei numeri di porta UDP. Esaminando il campo "transport", notiamo che l'applicazione *wb* viene eseguita direttamente al di sopra di UDP, mentre i flussi audio e video vengono trasmessi mediante "RTP/AVP". Ciò significa che usano il protocollo RTP e il *profilo di applicazione* (come definito nella Sezione 9.3.1) denominato AVP. Tale profilo di applicazione definisce un certo numero di diversi schemi di codifica audio e video: in questo caso l'audio viene codificato usando lo schema 0 (che prevede la codifica con una velocità di campionamento di 8 KHz e campioni di 8 bit), mentre il video viene codificato usando lo schema 31, che rappresenta lo schema di codifica H.261. Questi "numeri magici" per gli schemi di codifica sono definiti nel documento RFC che definisce il profilo AVP, ma in SDP è anche possibile descrivere schemi di codifica non standard.

Infine, troviamo la descrizione del tipo di *media* relativo all'applicazione *wb*. L'intero schema di codifica delle informazioni relative a questi dati è specifico dell'applicazione *wb*, per cui è sufficiente fornire soltanto il nome dell'applicazione nel campo "format", in modo analogo all'inserimento del tipo *application/wb* in un messaggio MIME. Ora che sappiamo come descrivere le sessioni, possiamo vedere come queste vengono iniziata. Un modo di utilizzo del protocollo SDP è l'annuncio di conferenze multimediali inviando messaggi SDP ad un indirizzo multicast ben noto. Lo strumento di visualizzazione delle sessioni, mostrato in Figura 9.13, funziona aggregandosi a tale gruppo multicast e visualizzando le informazioni desunte dai messaggi SDP ricevuti.

Il protocollo SDP gioca anche un importante ruolo congiuntamente al protocollo SIP (Session Initiation Protocol). Con l'aumento dell'importanza delle applicazioni di tipo "voice over IP" (VOIP, cioè il supporto di applicazioni di telefonia all'interno di reti IP), il protocollo SIP ha attirato su di sé molta attenzione e ora c'è un gruppo di lavoro di IETF che se ne occupa. Anche se SIP può essere usato per molte altre cose, diverse dalla telefonia IP, è certo che tale applicazione ne costituisce un forte traino.

SIP

Il protocollo SIP è un protocollo del livello applicazione che assomiglia, in qualche modo, al protocollo HTTP, essendo basato su un modello richiesta/risposta simile, pur essendo stato progettato per applicazioni di tipo piuttosto diverso, per cui ha potenzialità abbastanza diverse da quelle di HTTP. Le funzionalità messe a disposizione da SIP si possono raggruppare in cinque categorie:

- Raggiungibilità dell'utente: determinare il dispositivo corretto con cui comunicare per raggiungere un certo utente.
- Disponibilità dell'utente: determinare se l'utente vuole prendere parte ad una particolare sessione di comunicazione oppure se è in grado di farlo.
- Potenzialità dell'utente: determinare i *media* utilizzabili e i relativi schemi di codifica.
- Instaurazione della sessione: stabilire i parametri della sessione, come i numeri di porta, che devono essere utilizzati da entrambe le parti coinvolte nella comunicazione.
- Gestione della sessione: un ampio spettro di funzioni, che comprendono il trasferimento di sessioni (per realizzare, ad esempio, il "trasferimento di chiamata", *call forwarding*) e la modifica dei parametri di sessione.

La maggior parte di queste funzionalità sono di facile comprensione, ma il problema della raggiungibilità dell'utente merita qualche discussione ulteriore. Un'importante differenza tra SIP e, ad esempio, HTTP sta nel fatto che SIP è utilizzato prevalentemente per comunicazioni fra persone umane, per cui è importante essere in grado di individuare singoli *utenti*, non soltanto calcolatori. E, diversamente dalla posta elettronica, non è sufficiente individuare un server al quale prima o poi l'utente accederà per consultare i messaggi là pervenuti: se vogliamo comunicare con un utente in tempo reale, dobbiamo sapere dove si trova in questo momento. Questo problema è ulteriormente complicato dal fatto che un utente potrebbe voler comunicare con molti dispositivi diversi, usando, ad esempio, il PC sulla scrivania del proprio ufficio oppure un dispositivo palmare quando è in viaggio; questi diversi dispositivi possono essere tutti attivi nello stesso istante e potrebbero avere funzionalità assai diverse fra loro (pensate, ad esempio, ad un "cercapersone", *pager*, alfanumerico e ad un "video-telefono" costituito da un PC). Idealmente, dovrebbe essere possibile, per gli altri utenti, localizza-

re un utente e comunicare con il dispositivo corretto in ogni momento. Inoltre, l'utente dovrebbe avere il controllo sulla modalità di ricezione delle chiamate: quando, dove e da chi.

Per consentire ad un utente di esercitare il livello di controllo appropriato sulle proprie chiamate, il protocollo SIP usa il concetto di *proxy*. Si può pensare ad un proxy SIP come ad un punto di contatto con l'utente, a cui vengono inviate le richieste iniziali per qualsiasi comunicazione; i proxy svolgono anche alcune funzioni per conto dei chiamanti e ne esamineremo il funzionamento con un esempio.

Considerate i due utenti di Figura 9.14. La prima cosa da notare è che ogni utente ha un nome, nel formato *user@domain*, molto simile ad un indirizzo di posta elettronica. Quando l'utente Bruce vuole iniziare una sessione con l'utente Larry, invia il proprio messaggio iniziale SIP al proxy locale del proprio dominio, *cisco.com*. Insieme ad altre informazioni, questo messaggio iniziale contiene un URI di SIP, che è una forma di identificativo omogeneo di risorsa di questo tipo:

SIP:larry@princeton.edu

Abbiamo visto un diverso esempio di URI nella Sezione 9.2.2. Gli URL (*uniform resource locator*), come <http://www.cs.princeton.edu>, sono un particolare tipo di URI che contiene informazioni complete per localizzare una risorsa (ad esempio, una pagina Web). Un URI di SIP fornisce l'identificazione completa di un utente, ma non dice come raggiungerlo, perché questa informazione può mutare nel tempo. Vedremo fra breve come si possa determinare la collocazione di un utente.

Dopo aver ricevuto il messaggio iniziale da Bruce, il proxy di *cisco.com* esamina lo URI di SIP e deduce che tale messaggio deve essere inviato al proxy di *princeton.edu*. Per ora, ipotizziamo che il proxy di *princeton.edu* abbia accesso ad una base di dati che consente di mettere in corrispondenza il nome *larry@princeton.edu* con l'indirizzo IP di uno o di più dispositivi mediante i quali Larry desidera ricevere messaggi in questo momento. Il proxy è, quindi, in grado di inoltrare il messaggio al dispositivo prescelto da Larry. L'invio del messaggio a più di un dispositivo viene detto *forking* ("diramazione") e può essere effettuato in parallelo o in serie (ad esempio, inviando il messaggio al telefono cellulare se l'utente non ha risposto al telefono dell'ufficio).

Il messaggio iniziale inviato da Bruce a Larry è molto probabilmente un messaggio SIP di tipo *invite*, che assomiglia a quello che segue:

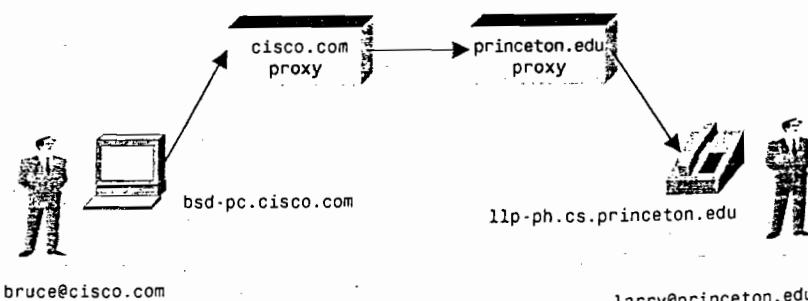


Figura 9.14 Instaurare una connessione attraverso due proxy SIP.

```

INVITE sip:larry@princeton.edu SIP/2.0
Via: SIP/2.0/UDP bsd-pc.cisco.com;branch=z9hG4bK433yte4
To: Larry <sip:larry@princeton.edu>
From: Bruce <sip:bruce@cisco.com>;tag=55123
Call-ID: xy745jj210re3@bsd-pc.cisco.com
CSeq: 271828 INVITE
Contact: <sip:bruce@bsd-pc.cisco.com>
Content-Type : application/sdp
Content-Length : 142
  
```

La prima riga identifica il tipo di funzione che viene eseguita (*invite*), la risorsa sulla quale eseguirla (l'entità chiamata, *sip:larry@princeton.edu*) e la versione del protocollo (2.0). Le righe d'intestazione successive sembrano probabilmente familiari, a causa della loro somiglianza con le righe di intestazione di un messaggio di posta elettronica. Il protocollo SIP definisce un gran numero di campi d'intestazione, anche se qui ne descriviamo solamente alcuni. Notate che l'intestazione *Via*, in questo esempio, identifica il dispositivo da cui ha preso origine il messaggio. Le intestazioni *Content-Type* e *Content-Length*: descrivono i contenuti del messaggio che segue l'intestazione, proprio come nei messaggi di posta elettronica codificati secondo lo standard MIME: in questo caso, il contenuto è un messaggio SDP (Session Description Protocol), che descrive il tipo di *media* (audio, video, ecc.) che Bruce vorrebbe scambiare con Larry ed altre proprietà della sessione, quali i tipi di CODEC (codificatori) di cui si fornisce il supporto. Notate che il campo *Content-Type*: di SIP dà la possibilità di usare, per questo scopo, un protocollo qualsiasi, anche se SDP è quello più comunemente utilizzato.

Tornando all'esempio, quando il messaggio *invite* arriva al proxy di *cisco.com*, tale proxy non solo irroltra il messaggio verso *princeton.edu*, ma risponde anche al mittente del messaggio stesso. Esattamente come nel protocollo HTTP, tutte le risposte hanno un codice, la cui organizzazione è simile a quella del protocollo HTTP, come evidenziato nella Tabella 9.2. Nella Figura 9.15 potete vedere una sequenza di messaggi SIP e delle relative risposte.

Il primo messaggio di risposta presente in questa figura è la risposta provvisoria 100 trying, che indica che il messaggio è stato ricevuto senza errori dal proxy del chiamante. Una volta che il messaggio *invite* è stato consegnato al telefono di Larry, il dispositivo avverte Larry e risponde con un messaggio 180 ringing. L'arrivo di questo messaggio al calcolatore di Bruce genera il "segnale di chiamata" (ring tone). Ipotizzando che Larry voglia e possa comunicare con Bruce, alzerà la cornetta del proprio telefono, provocando l'invio del messaggio 200 OK. Il calcolatore di Bruce risponde con un ACK e a questo punto può iniziare il flusso di dati fra le due parti coinvolte (ad esempio, un flusso audio incapsulato mediante RTP). Notate che a questo punto le due parti conoscono i rispettivi indirizzi, per cui i messaggi ACK possono essere inviati direttamente, senza passare per i proxy. A questo punto i proxy non sono più coinvolti nella comunicazione. Notate che, quindi, i flussi di dati seguiranno tipicamente, all'interno della rete, un percorso diverso da quello seguito dai messaggi di segnalazione iniziali. Inoltre, anche se a questo punto uno dei proxy (o anche entrambi) dovesse guastarsi, la conversazione proseguirebbe normalmente. Infine, quando una delle due parti intende terminare la sessione, invia un messaggio BYE, che in condizioni normali provoca la risposta 200 OK.

C'è ancora qualche dettaglio su cui abbiano sorvolato. Uno di questi riguarda la negoziazione delle caratteristiche della sessione: forse Bruce vorrebbe comunicare usando sia l'audio che

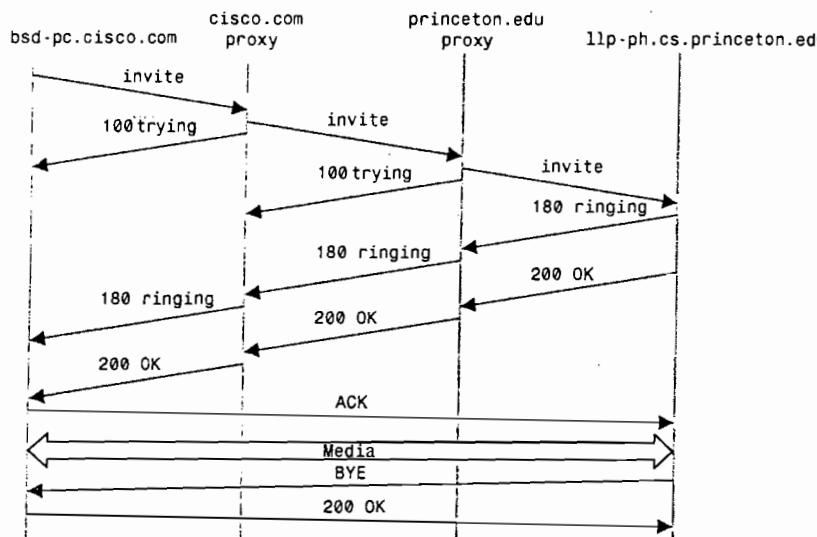


Figura 9.15 Flusso di messaggi per una semplice sessione SIP.

il video, mentre il telefono di Larry ha soltanto capacità audio. In questo caso il telefono di Larry invierebbe un messaggio SDP, insieme alla propria risposta 200 OK, descrivendo le proprietà della sessione che sono accettabili per Larry e per il dispositivo, considerando le opzioni proposte nel messaggio invite di Bruce. In questo modo, prima che inizino i flussi multimediali, viene raggiunto un accordo sui parametri della sessione.

L'altro grande problema sul quale non abbiamo fornito dettagli riguarda la collocazione del dispositivo corretto mediante il quale viene raggiunto Larry. Per prima cosa, il calcolatore di Bruce ha inviato il suo messaggio invite al proxy di cisco.com: questa informazione potrebbe essere stata configurata all'interno del calcolatore stesso, oppure appresa mediante il protocollo DHCP. Successivamente, il proxy di cisco.com ha localizzato il proxy di princeton.edu, operazione che si può compiere usando un tipo speciale di ricerca DNS che restituisce l'indirizzo IP del proxy SIP per il dominio princeton.edu. Infine, il proxy di princeton.edu ha dovuto trovare un dispositivo mediante il quale contattare Larry: solitamente un server proxy ha accesso ad una base di dati relativa alla collocazione degli utenti, che può essere popolata in vari modi, con una configurazione manuale o sfruttando le più flessibili capacità della registrazione di SIP.

Un utente si può registrare presso un servizio di rintracciabilità inviando un messaggio SIP di tipo register all'anagrafe (register) del proprio dominio: tale messaggio crea un'associazione fra un indirizzo di record e un indirizzo per il contatto. Un indirizzo di record è, solitamente, un URI di SIP che sia l'indirizzo "ben noto" dell'utente (ad esempio, `sip:larry@princeton.edu`), mentre l'indirizzo per il contatto sarà l'indirizzo al quale si può reperire l'utente in quel momento (ad esempio, `sip:larry@llp-ph.cs.princeton.edu`): precisamente, è la corrispondenza che serviva al proxy di princeton.edu nel nostro esempio.

9.3 Applicazioni multimediali

Note che un utente si può registrare in più siti e che più utenti si possono registrare per essere raggiungibili al medesimo dispositivo. Ad esempio, si può pensare ad un gruppo di persone che entrano in una sala riunioni dotata di un telefono IP e che tutte si registrino in modo da ricevere le proprie telefonate a quell'apparecchio.

Il protocollo SIP è molto flessibile e ricco di funzioni, potendo fornire supporto per un'ampia scelta di chiamate in situazioni complesse, oltre a poter gestire applicazioni che poco o nulla hanno a che fare con la telefonia. Ad esempio, il protocollo SIP fornisce il supporto ad operazioni che consentono l'instradamento di una chiamata verso un server di posta vocale o un server che pone l'utente in attesa facendogli ascoltare brani musicali. Si può anche facilmente vedere come lo si possa usare per applicazioni di messaggeria istantanea (*instant messaging*), per le quali il gruppo di lavoro SIMPLE di IETF sta attualmente definendo uno standard.

H.323

Anche ITU è stato molto attivo nell'area del controllo delle chiamate, con particolare attenzione al settore della telefonia, che costituisce l'ambito di attività tradizionale di tale organismo. Fortunatamente, in questo caso c'è stato un considerevole coordinamento fra IETF e ITU, per cui i diversi protocolli possono in qualche modo interoperare. La più importante raccomandazione ITU per le comunicazioni multimediali in reti a pacchetto prende il nome di H.323 e raccoglie molte altre raccomandazioni, tra le quali H.225 per il controllo delle chiamate. L'intero insieme di raccomandazioni di H.323 si estende per molte centinaia di pagine ed il protocollo è famoso per la sua complessità, per cui possiamo qui darne soltanto una breve panoramica.

H.323 è molto diffuso come protocollo per la telefonia in Internet e per tale applicazione lo prendiamo qui in considerazione. Un dispositivo che sia sorgente o destinazione di una chiamata viene detto "terminale H.323" e potrebbe trattarsi di una stazione di lavoro che esegue un'applicazione di telefonia per Internet, oppure potrebbe essere un "apparecchio" appositamente progettato, ad esempio un dispositivo simile ad un telefono con software di rete e una porta Ethernet. I terminali H.323 possono colloquiare fra loro direttamente, ma frequentemente le chiamate sono mediate da un dispositivo che prende il nome di *gatekeeper* ("custode" o "portinaio"), che svolge diverse funzioni, come la traduzione fra diversi formati di indirizzi usati per le chiamate telefoniche e il controllo di quante chiamate contemporanee possano essere effettuate, per limitare la banda utilizzata dalle applicazioni H.323. Lo standard H.323 prefigura anche il concetto di *gateway*, che connette una rete H.323 ad altri tipi di reti; l'utilizzo più comune di un gateway è quello di connettere una rete H.323 alla rete telefonica pubblica commutata (PSTN, *public-switched telephone network*), come illustrato in Figura 9.16, consentendo così ad un utente che esegue un'applicazione H.323 sul proprio calcolatore di colloquiare con una persona che usa un telefono convenzionale della rete telefonica pubblica. Un'utile funzione svolta da un *gatekeeper* consiste nell'aiutare un terminale a trovare un gateway, magari scegliendo fra diverse opzioni per trovarne uno che sia più prossimo alla destinazione finale della chiamata: una cosa chiaramente utile in un mondo dove i telefoni convenzionali sono in numero molto più elevato dei telefoni basati su PC. Quando un terminale H.323 effettua una chiamata verso un telefono convenzionale, il gateway diviene l'estremità effettiva per la chiamata H.323 ed è responsabile della traduzione appropriata sia delle informazioni di segnalazione sia del flusso di dati che devono essere trasportati dalla rete telefonica.

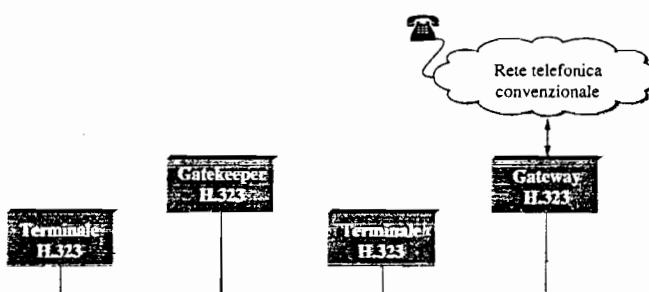


Figura 9.16 Dispositivi presenti in una rete H.323.

Una parte importante dello standard H.323 è costituita dal protocollo H.245, che viene usato per negoziare le proprietà della chiamata, in modo analogo a quanto avviene con il protocollo SDP descritto in precedenza. Un messaggio H.245 può elencare un certo numero di diversi standard di codifica audio per i quali si fornisce il supporto e l'altra entità terminale della conversazione può replicare con un elenco degli standard per i quali fornisce il supporto, per fare in modo che le due entità scelgano uno standard comprensibile ad entrambe. Lo standard H.245 può anche essere usato per indicare quale sia la porta UDP che verrà usata da RTP e RTCP per il flusso di dati (o per i flussi, dato che, ad esempio, una conversazione potrebbe coinvolgere sia flussi audio sia flussi video) della chiamata. Raggiunto questo accordo, la conversazione può procedere, usando RTP per veicolare i flussi di dati e RTCP per trasportare le relative informazioni di controllo.

9.4 Reti sovrapposte (overlay networks)

Fin dal proprio concepimento, la rete Internet ha adottato un modello molto chiaro, nel quale i router interni alla rete hanno il compito di inoltrare pacchetti dalla sorgente alla destinazione, e i programmi applicativi vengono eseguiti sugli host connessi ai confini della rete. Il paradigma client/server, illustrato in riferimento alle applicazioni discusse nelle prime due sezioni di questo capitolo, aderisce senza dubbio a questo modello.

Negli ultimi anni, tuttavia, la distinzione fra *inoltro di pacchetti* e *elaborazione applicativa* è diventata meno chiara. All'interno di Internet agiscono nuove applicazioni distribuite e, in molti casi, tali applicazioni prendono le proprie decisioni relativamente all'inoltro di pacchetti. Queste nuove applicazioni ibride possono a volte essere implementate estendendo i router e gli switch tradizionali, in modo che possano fornire una modesta quantità di elaborazione specifica per singole applicazioni. Ad esempio, i cosiddetti *switch di livello 7* vengono anteposti a *cluster* di server e inoltrano le richieste HTTP ad uno specifico server in base alla risorsa richiesta, identificata dall'URL. Tuttavia, le *reti sovrapposte* (*overlay networks*) si stanno rapidamente affermando come la strategia preferita per introdurre nuove funzionalità in Internet.

Potete immaginare una rete sovrapposta come se fosse una rete logica realizzata al di sopra di una rete fisica. In base a questa definizione, la stessa Internet è una rete sovrapposta, cosa che, in effetti, corrisponde a verità. La Figura 9.17 schematizza una rete sovrapposta

realizzata al di sopra di una rete sottostante. Ogni nodo della rete sovrapposta esiste anche nella rete sottostante, ed elabora e inoltra pacchetti con modalità specifiche dell'applicazione. Le linee che connettono i nodi della rete sovrapposta sono realizzate mediante tunnel attraverso la rete sottostante. Al di sopra di una stessa rete possono esistere più reti sovrapposte, ciascuna con il comportamento specifico di una determinata applicazione, e le reti sovrapposte possono anche essere annidate, una al di sopra dell'altra. Ad esempio, tutte le reti sovrapposte di cui parleremo in questa sezione considerano la rete Internet di oggi come se fosse la rete sottostante.

Abbiamo già visto il tunneling, per realizzare, ad esempio, reti private virtuali (VPN). Riassumendo brevemente, i nodi a ciascun capo del tunnel considerano il percorso a salti multipli (*multihop path*) che li collega come se fosse una singola linea di connessione, e i nodi attraverso cui passa il tunnel inoltrano i pacchetti basandosi sulla loro intestazione più esterna, senza rendersi conto del fatto che i nodi terminali hanno allegato un'intestazione più interna. Ad esempio, la Figura 9.18 mostra tre nodi di una rete sovrapposta (A, B e C) connessi da due tunnel. In questo esempio, il nodo B potrebbe prendere una decisione relativa

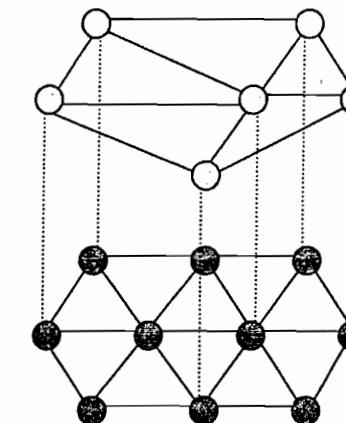


Figura 9.17 Rete sovrapposta (*overlay network*), stratificata al di sopra di una rete fisica.

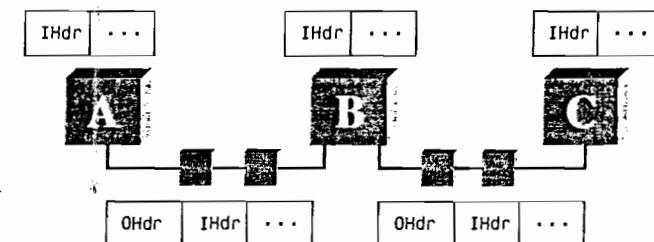


Figura 9.18 Nodi di una rete sovrapposta connessi da tunnel che attraversano nodi fisici.

all'inoltro di pacchetti da A a C basandosi sull'intestazione interna (*IHdr, inner header*), per poi allegare un'intestazione esterna (*OHdr, outer header*) che identifica come destinazione il nodo C all'interno della rete sottostante. I nodi A, B e C sono in grado di interpretare sia l'intestazione interna sia quella esterna, mentre i router intermedi elaborano solamente l'intestazione esterna. Analogamente, A, B e C possiedono un indirizzo sia nella rete sovrapposta sia nella rete sottostante, che non devono necessariamente coincidere; ad esempio, l'indirizzo nella rete sottostante potrebbe essere un indirizzo IP a 32 bit, mentre l'indirizzo nella rete sovrapposta potrebbe essere un indirizzo sperimentale a 128 bit. In effetti, la rete sovrapposta potrebbe anche non utilizzare affatto indirizzi convenzionali, ma prendere decisioni di instradamento basate su URL, su nomi di dominio, su un'interrogazione XML o anche sul contenuto del pacchetto.

9.4.1 Reti sovrapposte per l'instradamento

Il tipo più semplice di rete sovrapposta è una rete che esiste soltanto per fornire il supporto ad una strategia di instradamento alternativa, senza che nei nodi della rete sovrapposta venga svolta alcuna elaborazione addizionale al livello di applicazione. Una rete privata virtuale (si veda la Sezione 4.1.8) può, in un certo senso, essere vista come un esempio di sovrapposizione per l'instradamento, senza però che venga definita una vera strategia alternativa, dal momento che vengono semplicemente definite informazioni alternative da inserire nelle tabelle di instradamento, che poi vengono elaborate dal normale algoritmo di inoltro del protocollo IP. In questo caso particolare, si parla di rete sovrapposta che utilizza "tunnel IP" e la capacità di utilizzare tali VPN è integrata in molti router disponibili in commercio.

Supponete, invece, di voler usare un algoritmo di instradamento che i costruttori di router commerciali non vogliono inserire nei propri prodotti. Come potreste farlo? Potreste semplicemente eseguire il vostro algoritmo in un insieme di host terminali ed effettuare dei tunnel attraverso i router di Internet. Questi host si comporterebbero come router nella rete sovrapposta: in qualità di host sono probabilmente connessi ad Internet mediante un'unica linea fisica di collegamento, ma come nodi della rete sovrapposta sono probabilmente connessi a più nodi vicini, tramite tunnel.

Dal momento che le reti sovrapposte sono, quasi per definizione, uno strumento per introdurre nuove tecnologie indipendentemente da processi di standardizzazione, non esistono esempi di standard per reti sovrapposte da presentare ed esaminare. Al contrario, presenteremo l'idea generale delle reti sovrapposte per l'instradamento descrivendo alcuni sistemi sperimentali che sono stati recentemente proposti da ricercatori nell'ambito delle reti di calcolatori.

Versioni sperimentali di IP

Le reti sovrapposte sono ideali per installare versioni sperimentali del protocollo IP che si spera possano diffondersi nel mondo. Ad esempio, il multicast IP è un'estensione di IP che interpreta gli indirizzi di classe D (quelli che hanno il prefisso 1110) come indirizzi multicast e viene utilizzato insieme ad uno dei protocolli di instradamento multicast, come DVMRP, descritto nella Sezione 4.4.

La rete MBone (*multicast backbone*) è una rete sovrapposta che realizza il multicast IP. Una delle applicazioni più popolari che viene eseguita nella rete MBone è vic, uno strumento che fornisce il supporto per la videoconferenza con molti partecipanti e viene utilizzato per diffondere in Internet seminari e convegni. Ad esempio, le riunioni di IETF, che durano una

Reti sovrapposte e l'irrigidimento di Internet

Data la sua popolarità e la sua straordinaria diffusione, è facile dimenticare che una volta Internet sia stata un laboratorio nel quale i ricercatori potevano fare esperimenti con reti a commutazione di pacchetto. Tuttavia, quanto più Internet è divenuta un successo commerciale, tanto più è meno utile come piattaforma per sperimentare liberamente nuove idee. Oggi, gli interessi commerciali danno forma al continuo sviluppo di Internet.

In effetti, una recente ricerca del National Research Council mette in evidenza l'irrigidimento di Internet, sia dal punto di vista intellettuale (la pressione per la compatibilità con gli standard attuali contrasta l'innovazione) sia in termini dell'infrastruttura stessa (è quasi impossibile che i ricercatori possano apportare modifiche al cuore dell'infrastruttura). La ricerca procede osservando che, al tempo stesso, sta emergendo un intero nuovo insieme di problemi aperti, che potrebbero richiedere un approccio rivoluzionario. Il dilemma, secondo quella ricerca è che

...le tecnologie di successo e ampiamente adottate tendono ad irrigidirsi, rendendo difficile l'introduzione di nuove funzionalità o la sostituzione con tecnologie migliori, anche qualora quelle attuali avessero terminato la propria vita utile. Gli attori industriali esistenti sul mercato non sono, generalmente, motivati per sviluppare o installare nuove tecnologie dirompenti...

Trovare il modo giusto per introdurre tecnologie profondamente innovative è un argomento interessante. Tali innovazioni sono probabilmente in grado di fare molto bene alcune cose, ma saranno sostanzialmente peggiori della tecnologia attuale sotto molti altri punti di vista. Ad esempio, per introdurre in Internet una nuova strategia di instradamento, dovreste costruire un router che non fornisca solamente il supporto per questa nuova strategia, ma che sia anche competitivo con quelli già affermati sul mercato dal punto di vista delle prestazioni, dell'affidabilità, degli strumenti di gestione, e così via: un compito estremamente difficile. Ciò che serve ai ricercatori è un modo che consente agli utenti di sperimentare i vantaggi delle nuove idee senza dover scrivere le centinaia di migliaia di righe di codice necessarie a fornire solamente il supporto del sistema di base.

settimana e coinvolgono migliaia di partecipanti, vengono solitamente diffuse nella rete MBone.

Come le reti private virtuali, la rete MBone usa sia i tunnel IP sia gli indirizzi IP, ma, diversamente dalle VPN, la rete MBone implementa un diverso algoritmo di inoltro: i pacchetti vengono inoltrati a tutti i vicini che si trovano più in basso nell'albero del più breve percorso multicast. Come in tutte le reti sovrapposte, i router con capacità multicast trasmettono mediante tunnel attraverso i router che non hanno tale capacità (*legacy router*), nella speranza che questi un giorno sparcano.

La rete 6-Bone è una rete sovrapposta simile che viene usata per installare, in modo graduale, la versione 6 del protocollo IP (IPv6). Come MBone, la rete 6-Bone usa i tunnel per inoltrare i pacchetti attraverso router IPv4. Diversamente dalla rete MBone, però, i nodi della rete 6-Bone non mettono semplicemente in atto una nuova interpretazione degli indirizzi IP a 32 bit, ma inoltrano i pacchetti sulla base dello spazio di indirizzamento di IPv6 a 128 bit. Inoltre, dato che IPv6 fornisce il supporto alla trasmissione multicast, anche 6-Bone lo fa.

Multicast fra sistemi terminali

Sebbene la rete MBone sia una rete sovrapposta piuttosto famosa, la tecnologia di multicast IP non è riuscita a diffondersi nel mondo e, di conseguenza, le applicazioni basate sul multicast, come la videoconferenza, si sono recentemente orientate su una strategia alternativa, chiamata *multicast fra sistemi terminali (end system multicast)*. L'idea che sta alla base di questa strategia consiste nell'accettare che il multicast IP non diverrà mai universalmente diffuso e di lasciare, invece, che siano gli host terminali che utilizzano una particolare applicazione basata sul multicast ad implementare i propri alberi multicast (come annotazione collaterale). c'è una scuola di pensiero secondo la quale il multicast IP non ha mai avuto successo perché semplicemente, non fa parte dello strato di rete, dal momento che deve fornire supporto a funzionalità di livello più elevato, come il controllo d'errore, di flusso e di congestione, oltre che alla gestione dell'appartenenza ai gruppi).

Prima di descrivere come funziona questa strategia, è importante capire, per prima cosa, che, diversamente dalle VPN e dalla rete MBone, il multicast fra sistemi terminali ipotizza che partecipino alla rete sovrapposta solamente host (e non router) della rete Internet. Inoltre, tipicamente questi host si scambiano messaggi usando tunnel UDP invece di tunnel IP, rendendo più semplice la loro realizzazione sotto forma di normali programmi applicativi. Ciò rende possibile vedere la rete sottostante come un grafo completamente connesso, perché ogni host di Internet è in grado di inviare un messaggio ad ogni altro host. In astratto, quindi, il multicast per sistemi terminali risolve il seguente problema: iniziando con un grafo completamente connesso che rappresenta Internet, l'obiettivo è quello di trovare l'albero multicast al suo interno che raggiunga tutti i membri del gruppo.

Poiché ipotizziamo che la rete Internet sottostante sia completamente connessa, una soluzione banale consisterebbe nel fare in modo che ogni sorgente fosse direttamente connessa ad ogni membro del gruppo. In altre parole, il multicast fra sistemi terminali potrebbe essere realizzato facendo in modo che ogni nodo invii messaggi di tipo unicast ad ogni altro membro del gruppo. Per vedere quale sia il problema derivante da questa realizzazione, specialmente se confrontata con la realizzazione del multicast IP mediante i router, considerate la topologia dell'esempio di Figura 9.19. La Figura 9.19(a) rappresenta una rete fisica, dove R1 e R2 sono router connessi ad una linea di collegamento transcontinentale a bassa ampiezza di banda; A, B, C e D sono host terminali e i ritardi delle linee sono indicati come pesi dei rami. Nell'ipotesi che A voglia inviare un messaggio multicast agli altri tre host, la Figura 9.19(b) mostra come funzionerebbe la banale trasmissione unicast: si tratta, ovviamente, di una strategia poco appetibile: dal momento che il medesimo messaggio deve attraversare per tre volte la linea A-R1, e due copie dello stesso messaggio attraversano la linea R1-R2. La Figura 9.19(c) rappresenta l'albero per il multicast IP costruito dal protocollo DVMRP: questo approccio elimina, chiaramente, i messaggi ridondanti. Senza l'aiuto dei router, però, la cosa migliore che potete sperare di ottenere da parte del multicast fra sistemi terminali è un albero simile a quello mostrato in Figura 9.19(d). Il multicast fra sistemi terminali definisce un'architettura per costruire tale albero.

L'approccio generale consiste nel supporto per più livelli di reti sovrapposte, ciascuno dei quali estrae un sottografo dalla rete sovrapposta sottostante, finché non è stato selezionato il sottografo previsto per l'applicazione. In particolare, per il multicast fra sistemi terminali questo avviene in due passi: per prima cosa costruiamo una semplice *griglia (mesh)* sovrapposta alla Internet completamente connessa, poi selezioniamo un albero multicast all'interno di tale griglia. L'idea è presentata in Figura 9.20, ipotizzando nuovamente che i quattro host terminali siano A, B, C e D. Il primo passo è quello critico: una volta che sia stata scelta una griglia sovrapposta adeguata, eseguiamo semplicemente un algoritmo di instradamento multicast

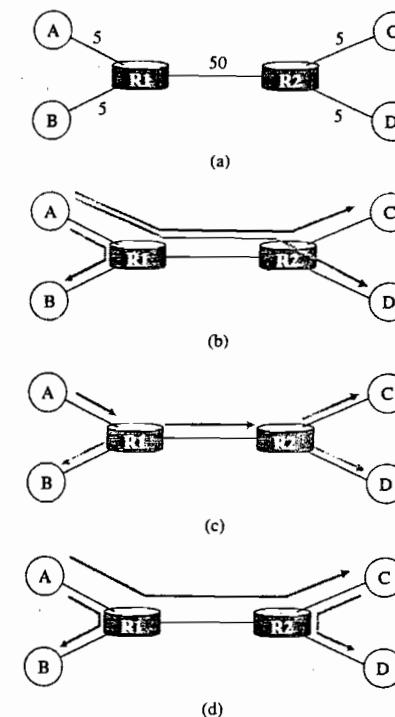


Figura 9.19 Alberi multicast alternativi messi in corrispondenza con una topologia fisica:
 (a) topologia fisica; (b) banale trasmissione unicast; (c) albero multicast costruito al livello di rete (dai router); (d) albero multicast costruito al livello applicativo (dagli host terminali).

standard (come DVMRP) su tale griglia, per costruire l'albero multicast. Possiamo anche concederci il lusso di ignorare il problema della scalabilità, tipico del multicast per Internet, perché la griglia intermedia può essere tracciata in modo da includere solamente quei nodi che vogliono partecipare ad un particolare gruppo multicast.

Il punto chiave nella costruzione della griglia sovrapposta intermedia è la selezione di una topologia che corrisponda, all'incirca, alla topologia fisica della sottostante Internet, ma lo dobbiamo fare senza che nessuno ci dica come sia veramente fatta la topologia di Internet, dal momento che l'algoritmo viene eseguito solamente negli host terminali e non nei router. La strategia generale consiste nel fatto che gli host terminali misurino la latenza di round-trip verso gli altri e che decidano di aggiungere linee alla griglia solamente quando lo ritengono opportuno, nel modo seguente.

Dapprima, ipotizzando che esista già una griglia, ogni nodo scambia con i vicini ad esso direttamente connessi l'elenco di tutti gli altri nodi che ritiene facciano parte della griglia. Quando un nodo riceve da un proprio vicino un tale elenco di appartenenza, aggiunge tali

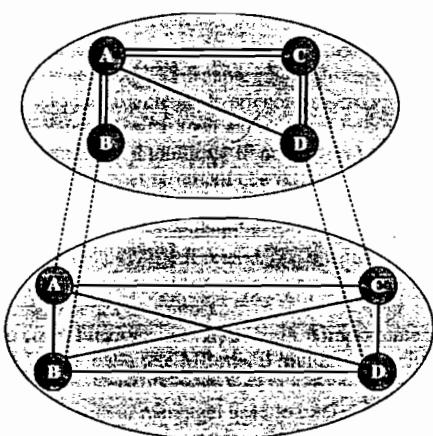


Figura 9.20 Albero multicast all'interno di una griglia sovrapposta.

informazioni al proprio elenco di appartenenza e inoltra l'elenco risultante ai propri vicini. Questa informazione, prima o poi, si propaga attraverso la griglia, in modo molto simile a quanto avviene in un protocollo di instradamento a vettore di distanza.

Quando un host si vuole unire al multicast nella rete sovrapposta, deve conoscere l'indirizzo IP di almeno un altro nodo che faccia già parte di tale rete, a cui inviare un messaggio di "aggregazione alla griglia" (*join mesh*). Il nodo che riceve il messaggio connette il nodo mittente alla griglia, aggiungendo una linea di connessione. In generale, il nuovo nodo potrebbe inviare un messaggio di adesione a più nodi, connettendosi quindi alla rete mediante più linee. Una volta che un nodo sia connesso alla rete tramite un insieme di nodi, invia periodicamente ai propri vicini messaggi di tipo "sono vivo" (*keepalive*), facendo loro sapere di voler continuare a far parte del gruppo.

Quando un nodo esce dal gruppo, invia ai propri vicini a cui è direttamente connesso un messaggio di "abbandono della griglia" (*leave mesh*) e questa informazione viene propagata agli altri nodi della griglia mediante gli elenchi di appartenenza descritti in precedenza. In alternativa, un nodo si può guastare, oppure semplicemente abbandonare silenziosamente il gruppo, nel qual caso i suoi vicini rilevano il fatto che non invia più i messaggi "keepalive". L'abbandono di alcuni nodi ha pochi effetti sulla griglia, ma se un nodo si rende conto che l'abbandono di un nodo ha provocato la suddivisione della griglia in due parti deve creare un nuovo ramo che lo collega ad un nodo che si trova nell'altra parte, inviando un messaggio di tipo "join mesh". Notate che più nodi possono decidere simultaneamente che la griglia è stata sottoposta a suddivisione, provocando l'aggiunta di più linee alla griglia.

Come descritto fino ad ora, finiremo per avere una griglia che è un sottografo della Internet originale, completamente connessa, ma con prestazioni sub-ottime, perché: (1) la selezione iniziale dei vicini aggiunge linee casuali alla topologia; (2) la riparazione dei partizionamenti può aggiungere linee che sono essenziali in quel momento ma che non sono utili a lungo andare; (3) l'appartenenza al gruppo può cambiare per effetto di aggregazioni e abbandoni dinamici; (4) le condizioni della rete sottostante possono cambiare. Ciò che deve

accadere è che il sistema valuti l'importanza di ogni ramo, per aggiungere nuovi rami alla griglia e per eliminare rami esistenti.

Per aggiungere nuovi rami, ogni nodo i contatta periodicamente un membro j , scelto a caso fra quelli ai quali non è direttamente connesso all'interno della griglia, misurando la latenza di round-trip del ramo (i, j) e valutando conseguentemente l'utilità derivante dall'aggiunta del ramo stesso. Se tale utilità si trova al di sopra di una certa soglia, il ramo (i, j) viene aggiunto alla griglia. La valutazione dell'utilità dell'aggiunta del ramo (i, j) potrebbe effettuarsi in questo modo:

```

EvaluateUtility(j)
utility = 0
for each membro m diverso da i
    CL = attuale latenza verso il nodo m lungo il percorso che attraversa la griglia
    NL = nuova latenza verso il nodo m attraverso la griglia aggiungendo il ramo (i, j)
    if (NL < CL) then
        utility += (CL - NL)/CL
return utility

```

La decisione di rimuovere un ramo viene presa in modo simile, tranne per il fatto che ogni nodo i si calcola il costo di ogni linea che lo collega al proprio vicino j nel modo seguente:

```

EvaluateCost(j)
Costij = numero di membri per i quali i usa j come salto successivo
Costji = numero di membri per i quali j usa i come salto successivo
return max(Costij, Costji)

```

Successivamente, si sceglie il vicino avente il costo minore e lo si elimina se tale costo cade al di sotto di una determinata soglia.

Poiché, infine, la griglia viene gestita usando quello che, essenzialmente, è un protocollo a vettore di distanza, è facile eseguire DVMRP per trovare un albero multicast adeguato all'interno della griglia. Notate che, sebbene non sia possibile dimostrare che il protocollo appena descritto genera una griglia ottimale, consentendo quindi a DVMRP di selezionare il miglior albero multicast possibile, sia le simulazioni sia una lunga esperienza sul campo suggeriscono che si comporta molto bene.

Reti sovrapposte elastiche (resilient overlay networks)

Un'altra strategia di instradamento a rete sovrapposta che sta raggiungendo una buona popolarità è quella che cerca percorsi alternativi per applicazioni unicast tradizionali. Tali reti sovrapposte sfruttano l'osservazione che in Internet non è valida la disuguaglianza triangolare, e la Figura 9.21 illustra ciò che vogliamo dire. Non è insolito trovare tre siti Internet (chiamiamoli A, B e C) tali che la latenza fra A e B sia maggiore della somma delle latenze da A a C e da C a B: di conseguenza, a volte sarebbe meglio inviare i pacchetti indirettamente, attraverso un nodo intermedio, piuttosto che inviarli direttamente a destinazione.

Come può accadere ciò? Bene, il protocollo BGP non promette mai di scoprire il percorso più breve fra due siti qualsiasi: cerca soltanto di trovare un percorso. Per rendere le cose ancora peggiori, molto spesso strategie pianificate da amministratori di sistema possono sovvertire il normale funzionamento di BGP: ciò accade, ad esempio, nei punti di *peering* fra i

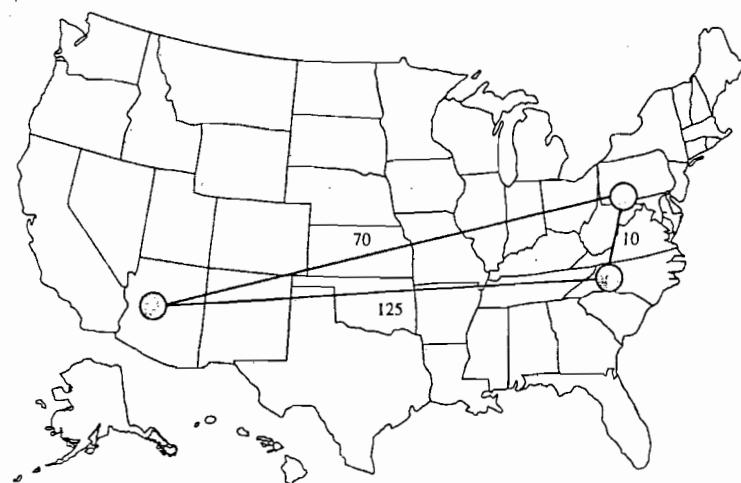


Figura 9.21 All'interno delle reti, la disugualanza triangolare non è necessariamente valida.

maggiori ISP del backbone. In breve, non dovremmo sorprenderci del fatto che, in Internet, la disugualanza triangolare non sia valida.

Come possiamo sfruttare questa osservazione? Il primo passo consiste nel comprendere che, per un algoritmo di instradamento, esiste un compromesso fondamentale fra la scalabilità e il raggiungimento di risultati ottimi. Per un verso, il protocollo BGP scala molto bene per reti molto grandi, ma spesso non identifica il miglior percorso possibile ed è lento ad adattarsi a guasti nella rete; d'altra parte, se il vostro unico obiettivo fosse quello di trovare il percorso migliore fra una manciata di siti, potreste ottenere risultati migliori tenendo sotto controllo la qualità di ogni percorso che potreste usare, avendo così la possibilità di scegliere in ogni momento il percorso migliore.

Una rete sovrapposta sperimentale, denominata RON (*resilient overlay network*, "rete sovrapposta elastica"), fa proprio questo. La rete RON scala fino ad un massimo di poche dozzine di nodi, perché usa la strategia $(N \times N)$ di tenere sotto stretto controllo (mediante l'interrogazione attiva) tre aspetti della qualità dei percorsi fra ogni coppia di nodi: latenza, banda disponibile e probabilità di perdita. Di conseguenza, è in grado di scegliere il percorso ottimale fra due nodi qualsiasi e di modificare rapidamente i percorsi in base a mutamenti nelle condizioni della rete. L'esperienza mostra che RON è in grado di fornire alle applicazioni miglioramenti delle prestazioni abbastanza modesti, ma, ciò che è più importante, reagisce molto più rapidamente ai guasti della rete. Ad esempio, durante un periodo di osservazione di 64 ore, nel 2001, una installazione di RON eseguita su 12 nodi ha rilevato 32 guasti di durata superiore ai 30 minuti ed è stata in grado di reagire a tutti tali guasti in un tempo medio inferiore a 20 secondi. Questo esperimento ha anche suggerito che l'inoltro di dati attraverso un solo nodo intermedio è solitamente sufficiente a gestire i guasti in Internet.

Dato che RON non scala bene, non è possibile utilizzare tale strategia per aiutare un host A scelto a caso a comunicare con l'host B, sempre scelto a caso. A e B devono conoscersi

9.4 Reti sovrapposte (overlay networks)

reciprocamente prima di decidere di comunicare, e devono appartenere al medesimo sistema RON. Ciò nonostante, in determinate situazioni RON sembra essere una buona idea, ad esempio per connettere una dozzina di siti aziendali sparsi nella rete Internet, oppure per consentire a voi e ai vostri 50 amici di allestire la vostra rete sovrapposta privata allo scopo di eseguire alcune applicazioni. La vera domanda che ci dobbiamo porre, però, è: cosa accadrebbe se ognuno volesse eseguire il proprio sistema RON? Il sovraccarico dovuto a milioni di RON che verificano in modo attivo la qualità dei percorsi potrebbe portare la rete al collasso? Quando diversi RON competono per lo stesso percorso, ci sarà qualcuno che potrà osservare un vero miglioramento? Queste domande sono tuttora prive di risposta.

Tutte queste reti sovrapposte illustrano, in generale, un concetto centrale per le reti di calcolatori: la *virtualizzazione*. Ciò significa che è possibile costruire una rete virtuale mediante risorse astratte (logiche) al di sopra di una rete fisica costituita da risorse fisiche. Inoltre, è possibile impilare queste reti virtuali, una sopra all'altra, e far coesistere più reti virtuali allo stesso livello. Ogni rete virtuale, a sua volta, fornisce nuove funzionalità che sono utili per alcuni insiemi di utenti, di applicazioni o di reti di livello superiore.

9.4.2 Reti tra pari (peer-to-peer)

Recenti applicazioni di condivisione di musica, come Napster e KaZaA, hanno introdotto nel linguaggio popolare il termine "peer-to-peer" (da pari a pari). Ma cosa significa esattamente, per un sistema, essere "peer-to-peer"? Di sicuro, nel contesto della condivisione di file MP3, significa non dover scaricare la musica da un sito centralizzato, ma poter, invece, accedere direttamente ai file musicali di chiunque, in Internet, ne abbia una copia memorizzata sul proprio calcolatore. Quindi, più in generale, potremmo dire che una rete tra pari consente ad una comunità di utenti di condividere le proprie risorse (contenuti, spazio di memorizzazione, ampiezza di banda della rete, ampiezza di banda del disco, CPU), fornendo in tal modo l'accesso, rispetto a quanto ciascun singolo utente potrebbe ottenere, ad archivi più grandi, a video/audio-conferenze più ampie, a ricerche e calcoli più complessi, e così via.

Abbastanza spesso, quando si parla di reti peer-to-peer si citano aggettivi quali *decentralizzato* e *auto-organizzante*, intendendo che i singoli nodi si organizzano in una rete senza alcun coordinamento centralizzato. Se ci pensate un po', termini di questo tipo si potrebbero usare anche per descrivere la stessa Internet. Il colmo dell'ironia, però, è che Napster non è un vero e proprio sistema peer-to-peer secondo la definizione che abbiamo dato, perché dipende da un'anagrafe centrale dei file noti e gli utenti devono fare ricerche in tale archivio per scoprire quali calcolatori offrono un particolare file. Soltanto l'ultimo passo, il trasferimento effettivo del file, ha luogo fra due calcolatori che appartengono a due utenti, ma questa è poco più di una tradizionale transazione di tipo client/server: l'unica differenza è che il server è gestito da una persona come voi piuttosto che da una grande azienda.

Siamo quindi tornati alla domanda originaria: cosa c'è di interessante nelle reti peer-to-peer? Una risposta è che sia il processo di localizzazione di un oggetto che vi interessa sia il processo di copiatura di tale oggetto sul vostro calcolatore avvengono senza che dobbiate contattare un'autorità centralizzata, e, al tempo stesso, il sistema è in grado di scalare bene le proprie prestazioni fino ad una dimensione di milioni di nodi. Un sistema peer-to-peer che sia in grado di assolvere a questi due compiti in modo decentralizzato si rivela essere una rete sovrapposta (*overlay network*), nella quale i nodi sono quegli host che desiderano condivide-

re oggetti di interesse per altri (ad esempio, musica e file di vario genere), mentre le linee (tunnel) che connettono tali nodi rappresentano la sequenza di calcolatori che dovete visitare per raggiungere l'oggetto che cercate. Questa descrizione diverrà più chiara dopo che avremo preso in esame due esempi.

Gnutella

Gnutella è una delle prime reti peer-to-peer che abbia tentato di fare una distinzione fra lo scambio di musica (che probabilmente costituisce una violazione del *copyright* di qualcuno) e la generale condivisione di file (che deve essere una cosa buona, dal momento che ci hanno insegnato a condividere le cose fin da quando avevamo due anni). Ciò che è interessante nel sistema Gnutella è che si tratta di uno dei primi sistemi che non dipende da un'anagrafe centralizzata degli oggetti. Al contrario, i partecipanti al sistema Gnutella si organizzano da soli in una rete sovrapposta simile a quella mostrata in Figura 9.22. In sostanza, ogni nodo che esegue il software Gnutella (cioè implementa il protocollo Gnutella) conosce un certo insieme di altri calcolatori che, a loro volta, eseguono il software Gnutella. La relazione "A e B si conoscono" corrisponde ai rami di questo grafo (fra un attimo diremo come si costruisce il grafo stesso).

Ogni volta che l'utente di un certo nodo vuole trovare un oggetto, Gnutella invia ai propri vicini, nel grafo, un messaggio QUERY per l'oggetto stesso (specificando, ad esempio, il nome del file). Se uno dei vicini possiede l'oggetto, risponde al nodo che ha inviato la richiesta con un messaggio QUERY RESPONSE, specificando dove possa essere reperito l'oggetto (ad esempio, un indirizzo IP e un numero di porta TCP). Il nodo che aveva fatto la richiesta potrà, poi, accedere all'oggetto mediante messaggi GET o PUT. Se un nodo non è in grado di soddisfare la richiesta, inoltra il messaggio QUERY ad ognuno dei propri vicini (con l'esclusione di quello che aveva inviato la richiesta), ed il procedimento viene ripetuto. In altre parole, Gnutella inonda la rete sovrapposta per localizzare l'oggetto desiderato, dopo aver impostato un tempo di vita (TTL, *time to live*) per ogni richiesta, in modo che la relativa inondazione non prosegua indefinitamente.

Oltre al tempo di vita e alla stringa di richiesta, ogni messaggio QUERY contiene un identificativo univoco della richiesta stessa (QID, *query identifier*), ma non contiene l'identità della sorgente del messaggio originario. Invece, ogni nodo gestisce un archivio dei messaggi QUERY che ha recentemente elaborato, memorizzandone sia il QID sia il vicino da cui ha ricevuto il messaggio e usando questo archivio in due modi. Innanzitutto, se dovesse mai

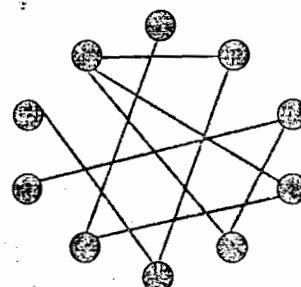


Figura 9.22 Esempio di topologia di una rete peer-to-peer del sistema Gnutella.

9.4 Reti sovrapposte (overlay networks)

ricevere un messaggio QUERY con un QID che corrisponde ad uno di quelli visti recentemente, il nodo non lo inoltrerà: in questo modo si interrompono i cicli di inoltro in modo più veloce di quanto accadrebbe per effetto del solo TTL. Secondariamente, ogni volta che il nodo riceve un messaggio QUERY RESPONSE da un vicino, sa di dover inoltrare la risposta al vicino che aveva inviato originariamente il messaggio QUERY corrispondente: in questo modo la risposta viaggia verso il nodo originario senza che nessuno dei nodi intermedi sappia chi abbia voluto localizzare quel particolare oggetto.

Tornando al problema di come si evolva il grafo, un nodo deve certamente conoscere almeno un altro nodo per aggregarsi alla rete sovrapposta di Gnutella: il nuovo nodo si connette alla rete sovrapposta mediante almeno questa linea di connessione. Dopo questo passo, un nodo apprende dell'esistenza di altri nodi per effetto dei messaggi QUERY RESPONSE, sia per oggetti che ha richiesto, sia per risposte che, semplicemente, lo attraversino. Un nodo è libero di decidere quali, fra i nodi che ha identificato in questo modo, debbano diventare i propri vicini. Il protocollo Gnutella mette a disposizione i messaggi PING e PONG, mediante i quali, rispettivamente, un nodo verifica se un vicino esiste ancora, e il vicino risponde.

Dovrebbe essere chiaro che Gnutella non è un protocollo particolarmente scaltro, per cui i sistemi successivi hanno cercato di migliorarlo. Un aspetto nel quale è possibile apportare miglioramenti è quello del metodo di propagazione delle richieste: l'inondazione ha la interessante proprietà di garantire il reperimento dell'oggetto desiderato con il minor numero possibile di salti, ma non è ben scalabile. Si può, invece, inoltrare le richieste a caso o in base alla probabilità di successo di ricerche precedenti. Un secondo miglioramento deriva dalla replica degli oggetti per iniziativa spontanea, dal momento che più copie vi sono, più facile dovrebbe essere trovarne una. In alternativa, si potrebbe sviluppare una strategia completamente diversa, che è il prossimo argomento che affronteremo.

Reti sovrapposte strutturate

Nel momento stesso in cui i sistemi di condivisione di file combattevano per riempire il vuoto lasciato da Napster, la comunità dei ricercatori stava esplorando un progetto alternativo per reti peer-to-peer. Parlando di queste reti usiamo il termine "reti strutturate", per porre in evidenza la diversità rispetto al modo sostanzialmente casuale (non strutturato) in cui si evolve una rete Gnutella. Le reti sovrapposte non strutturate, come Gnutella, impiegano un meccanismo di costruzione e algoritmi di gestione della rete piuttosto banali: la cosa migliore che possono fare è una ricerca casuale e non affidabile. Al contrario, le reti sovrapposte strutturate sono progettate per adeguarsi ad un grafo avente una struttura particolare che consenta la localizzazione di oggetti in modo affidabile ed efficiente (con ritardo limitato in senso probabilistico), in cambio della complessità introdotta nella fase di costruzione e manutenzione della rete.

Se pensate, ad un elevato livello di astrazione, a cosa stiamo cercando di fare, occorre rispondere a due domande: (1) come si mettono in corrispondenza oggetti e nodi? (2) come si fanno arrivare le richieste al nodo che è responsabile di un determinato oggetto? Cominciamo dalla prima domanda, che si può enunciare molto semplicemente: come si mette in corrispondenza un oggetto di nome x con l'indirizzo del nodo n che è in grado di fungere da server per mettere a disposizione quell'oggetto? Mentre le reti peer-to-peer tradizionali non hanno alcun controllo su quali nodi ospitino l'oggetto x , se potessimo controllare il modo in cui gli oggetti si distribuiscono nella rete, potremmo essere in grado di svolgere meglio il compito di ritrovare tali oggetti quando servono.

Una tecnica ben nota per mettere in corrispondenza nomi e indirizzi prevede l'utilizzo di una *tavella hash*, in modo che

$$\text{hash}(x) \rightarrow n$$

implica che l'oggetto x viene inizialmente inserito nel nodo n ; quando, in seguito, un client cerca di localizzare x , deve soltanto eseguire la funzione di hash di x per determinare che tale oggetto si trova nel nodo n . Un approccio basato su tabella hash ha l'interessante proprietà che tende a distribuire gli oggetti in modo omogeneo nell'insieme di nodi, ma gli algoritmi di hashing più semplici hanno un grave problema: quanti valori di n possiamo avere? (nella terminologia classica dell'hashing, quanti *bucket* ci dovrebbero essere?) Potremmo, semplicemente, decidere che vi siano, ad esempio, 101 valori possibili come risultato della funzione di hash, e usare una funzione di hash che esegua il modulo 101 (cioè il resto della divisione intera per 101), in questo modo:

```
hash(x)
return x % 101
```

Sfortunatamente, se vi sono più di 101 nodi che desiderano ospitare oggetti, non li possiamo sfruttare tutti. D'altra parte, se selezioniamo un numero maggiore del più grande numero di nodi possibile, ci sarà qualche valore di x che produrrà un valore di hash che va ad indirizzare un nodo che non esiste. Infine, c'è anche il problema, non trascurabile, di tradurre il valore restituito dalla funzione di hash in un vero indirizzo IP.

Per risolvere questi problemi, le reti peer-to-peer strutturate usano un algoritmo che prende il nome di *hashing coerente* (*consistent hashing*), che distribuisce uniformemente un insieme di oggetti all'interno di un vasto spazio di identificativi. La Figura 9.23 visualizza uno spazio di identificativi a 128 bit sotto forma di cerchio, usando l'algoritmo sia per piazzare gli oggetti

$$\text{hash}(\text{object_name}) \rightarrow \text{objid}$$

sia per piazzare i nodi

$$\text{hash}(\text{IP_addr}) \rightarrow \text{nodeid}$$

sul cerchio. Dato che uno spazio a 128 bit è veramente enorme, è altamente improbabile che un oggetto produca un valore di hash che sia esattamente uguale al valore di hash generato per l'indirizzo IP di un nodo. In ogni caso, per tener conto di questa remota possibilità, ogni oggetto viene gestito dal nodo il cui identificativo, *nodeid*, è il più vicino, all'interno di questo spazio a 128 bit, all'identificativo dell'oggetto, *objid*. In altre parole, l'idea è quella di usare una funzione di hash di ottima qualità per mettere in corrispondenza sia i nodi sia gli oggetti con i punti del medesimo, vasto e poco occupato, spazio di identificativi; successivamente, gli oggetti vengono messi in corrispondenza con i nodi mediante la prossimità numerica dei rispettivi identificativi. Come nell'hashing ordinario, questa tecnica distribuisce gli oggetti in modo piuttosto equo fra i nodi: diversamente dall'hashing ordinario, quando un nodo (cioè un *bucket* dell'hashing) viene aggiunto o viene eliminato, soltanto pochi oggetti devono essere spostati.

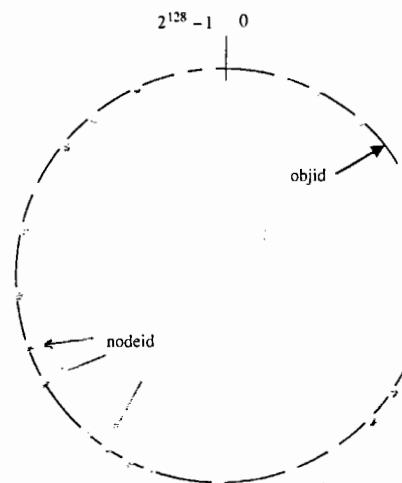


Figura 9.23 Sia i nodi che gli oggetti vengono messi in corrispondenza, tramite una funzione di hash, con gli identificativi dello spazio; gli oggetti vengono gestiti dal nodo più vicino all'interno dello spazio stesso.

Passiamo ora alla seconda domanda: come fa un utente, che voglia accedere all'oggetto x , a sapere qual è il nodo più vicino all'identificativo di x in questo spazio? Una possibile risposta è che ogni nodo memorizza la tabella completa degli identificativi dei nodi e degli indirizzi IP ad essi associati, ma per una rete di grandi dimensioni ciò non sarebbe realizzabile in pratica. L'alternativa, che è l'approccio utilizzato dalle reti peer-to-peer strutturate, consiste nell'*inviare un messaggio a tale nodo!* In altre parole, se costruiamo la rete sovrapposta in modo opportuno (che è come dire che dobbiamo scegliere in modo opportuno i dati nella tabella di instradamento dei nodi), allora possiamo trovare un nodo semplicemente cercando di raggiungerlo con un messaggio. Nel suo complesso, questo approccio viene detto *a tabelle hash distribuite* (DHT, distributed hash tables), perché, dal punto di vista concettuale, la tabella hash è suddivisa (distribuita) fra tutti i nodi della rete.

La Figura 9.24 illustra ciò che accade in un semplice spazio di identificativi a 28 bit. Per rendere più concreta la discussione, consideriamo l'approccio che viene utilizzato da una particolare rete peer-to-peer, chiamata Pastry; altri sistemi funzionano in modo simile (si vedano i lavori citati nella sezione "Ulteriori letture" per ulteriori esempi).

Supponete di trovarvi nel nodo avente identificativo 65a1fc (in esadecimale) e di cercare di localizzare l'oggetto avente identificativo d46a1c. Vi accorgrete che il vostro identificativo non ha nulla in comune con quello dell'oggetto, ma conoscete un nodo che ne condivide almeno il prefisso d: quel nodo, nello spazio degli identificativi, è più vicino all'oggetto di quanto lo siate voi, per cui inoltrate il messaggio verso quel nodo (non specifichiamo il formato del messaggio che viene inoltrato, ma potete immaginare che dica "Voglio localizzare l'oggetto d46a1c"). Ipotizzando che il nodo d13da3 conosca un altro nodo che condivide con l'oggetto un prefisso più lungo, il messaggio viene inoltrato a quest'ultimo nodo. Questo procedimento, che consiste nell'avvicinarsi all'oggetto all'interno dello spazio degli identifi-

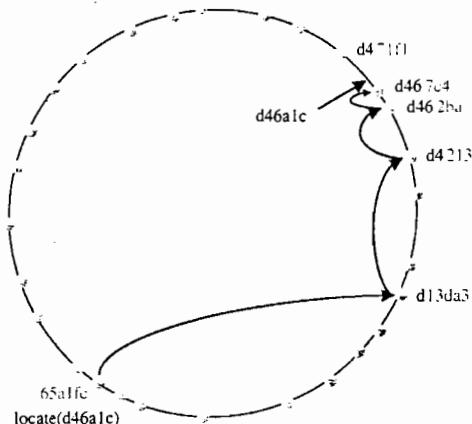


Figura 9.24 Si localizzano gli oggetti mediante l'instradamento attraverso la rete sovrapposta di tipo peer-to-peer.

cativi, continua finché non viene raggiunto un nodo che non conosca un nodo più vicino all'oggetto di quanto non lo sia esso stesso. Questo nodo è, per definizione, quello che ospita l'oggetto. Ricordate che, mentre ci siamo spostati, dal punto di vista logico, attraverso lo "spazio degli identificativi", il messaggio è stato, in realtà, inoltrato di nodo in nodo attraverso la Internet sottostante.

Ogni nodo memorizza sia una tabella di instradamento (di cui parleremo ancora in seguito) sia gli indirizzi IP di un piccolo insieme di nodi aventi identificativi numericamente più piccoli e più grandi del proprio, un insieme che viene chiamato *insieme delle foglie* (*leaf set*) del nodo stesso. L'importanza di questo insieme delle foglie consiste nel fatto che, una volta che un messaggio sia stato instradato verso un qualsiasi nodo che appartenga allo stesso insieme delle foglie a cui appartiene il nodo che ospita l'oggetto, quel nodo può inoltrare il messaggio direttamente verso la sua destinazione finale. Detto in altro modo, l'insieme delle foglie agevola la consegna di un messaggio, in modo corretto ed efficiente, al nodo numericamente più vicino all'oggetto, anche se possono esistere più nodi che condividono con l'oggetto un prefisso di lunghezza massima. Inoltre, l'insieme delle foglie rende più robusto l'instradamento, perché qualsiasi nodo dell'insieme è in grado di instradare un messaggio nello stesso modo di un altro nodo appartenente allo stesso insieme. Di conseguenza, se uno dei nodi non è in grado di far progredire l'instradamento di un messaggio, uno dei suoi vicini all'interno dell'insieme delle foglie potrebbe essere in grado di farlo. Riassumendo, la procedura di instradamento è definita in questo modo:

```

Route(D)
  if D si trova nel mio insieme delle foglie
    inoltra al membro dell'insieme delle foglie numericamente più vicino
  else
    sia l = lunghezza del prefisso condiviso
  
```

9.4 Reti sovrapposte (overlay networks)

```

sia d = valore della cifra di posto l nell'indirizzo di D
if RouteTab[l, d] esiste
  inoltra a RouteTab[l, d]
else
  inoltra al nodo noto numericamente più vicino con un prefisso almeno
  altrettanto lungo
  
```

La tabella di instradamento, denominata *RouteTab*, è un array bidimensionale; ha un riga per ogni cifra esadecimale presente negli identificativi (per identificativi a 128 bit, ci sono 32 cifre) e una colonna per ogni valore esadecimale (ovviamente, questi valori sono 16). Ogni valore della riga *i* condivide con il nodo un prefisso di lunghezza *i* e, all'interno della riga *i*, il valore presente nella colonna *j* ha il valore esadecimale *j* nella posizione *i* + 1. La Figura 9.25 mostra le prime tre righe di un esempio di tabella di instradamento per il nodo 65a1fcx, dove *x* indica un suffisso non specificato. La figura mostra i prefissi degli identificativi con cui sono in corrispondenza i dati presenti nella tabella, ma non mostra i valori presenti nella tabella stessa, che sono gli indirizzi IP del nodo successivo verso cui instradare.

L'inserimento di un nuovo nodo nella rete sovrapposta funziona più o meno come l'instradamento di un messaggio per localizzare un oggetto. Il nuovo nodo deve conoscere almeno un nodo che sia già membro della rete, a cui chiede di instradare un messaggio "aggiungi nodo" verso il nodo numericamente più vicino all'identificativo del nodo che si sta inserendo, come mostrato in Figura 9.26. Mediante questo procedimento di instradamento il nuovo nodo viene a conoscere gli altri nodi con cui condivide un prefisso ed è in grado, in questo modo, di riempire la propria tabella di instradamento. Nel tempo, quando altri nodi entrano nella rete, i nodi esistenti possono aggiungere alla propria tabella di instradamento le informazioni relative a tali nuovi nodi: lo fanno quando un nuovo nodo condivide un prefisso più lungo di quello attualmente memorizzato nella propria tabella. Inoltre, i vicini che appartengono all'insieme delle foglie si scambiano le tabelle reciprocamente, per cui, con il passare del tempo, le informazioni di instradamento di propagano all'interno della rete sovrapposta.

Riga 0	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Riga 1	6	6	6	6		6	6	6	6	6	6	6	6	6	6	6
	0	1	2	3	4		6	7	8	9	a	b	c	d	e	f
Riga 2	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
Riga 3	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
	0	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figura 9.25 Esempio di tabella di instradamento per il nodo con identificativo 65a1fcx.

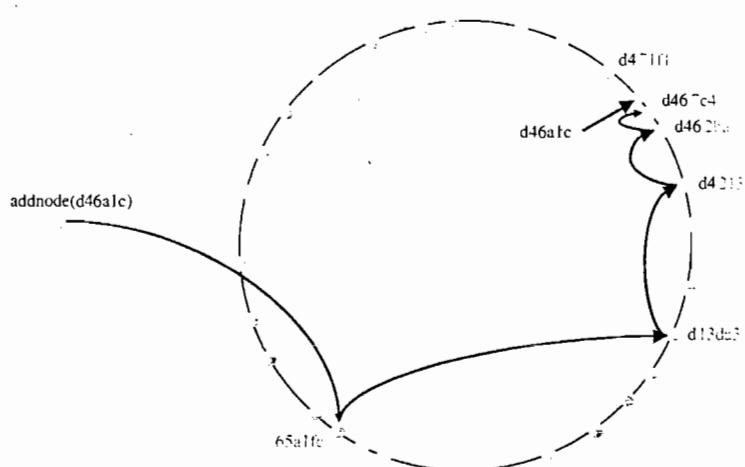


Figura 9.26 Inserimento di un nodo nella rete.

Può darsi che vi siate accorti che, sebbene le reti sovrapposte strutturate forniscano un limite probabilistico al numero di salti di instradamento necessari a localizzare un certo oggetto (il numero di salti in una rete Pastry è limitato da $\log_{16} N$, dove N è il numero di nodi della rete sovrapposta), ogni salto può dare un contributo non trascurabile al ritardo, perché ogni nodo intermedio può trovarsi ovunque, all'interno di Internet (nel caso peggiore, ogni nodo si trova in un diverso continente!). In effetti, per una rete sovrapposta che si estenda sull'intero globo e che usi l'algoritmo appena descritto, il ritardo che ci si può attendere per ogni salto è il ritardo medio calcolato fra tutte le coppie di nodi di Internet! Fortunatamente, in pratica possiamo fare assai meglio. L'idea è quella di scegliere i dati da inserire nella tabella di instradamento in modo che si riferiscono, fra tutti i nodi aventi un identificativo con prefisso adatto a quella casella, a nodi che siano vicini nella rete fisica sottostante. Si osserva, sperimentalmente, che facendo così si ottengono ritardi, per l'instradamento fra nodi terminali, che sono compresi entro un piccolo fattore moltiplicativo rispetto al ritardo previsto fra la sorgente e la destinazione.

Infine, dobbiamo dire che, fino a questo punto, la discussione si è incentrata sul problema generale della localizzazione di oggetti in una rete peer-to-peer. Data tale infrastruttura di instradamento, è possibile costruire diversi servizi. Ad esempio, un servizio di condivisione di file userà i file come nomi degli oggetti: per localizzare un file, per prima cosa si calcola il valore di hash corrispondente al suo nome, che diventa l'identificativo dell'oggetto. poi si instrada verso tale identificativo un messaggio di tipo "trova l'oggetto". Il sistema potrebbe anche replicare ogni file in più nodi, per migliorare la sua disponibilità: un modo per fare ciò potrebbe essere quello di memorizzare una copia del file in ogni nodo dell'insieme delle foglie verso cui vengono normalmente instradati i messaggi di ricerca di quel file. Ricordate che, nonostante tali nodi siano vicini nello spazio degli identificativi, è probabile che siano fisicamente distribuiti nell'intera Internet, per cui, mentre la mancanza di energia elettrica in una città renderebbe inaccessibili tutte le repliche di un file che si trovassero fisicamente

vicine in un file system tradizionale, in una rete peer-to-peer una o più repliche potrebbero tranquillamente sopravvivere ad un guasto di questo tipo.

Usando le tabelle hash distribuite si possono anche immaginare servizi diversi dalla condivisione di file. Considerate, ad esempio, le applicazioni multicast. Invece di costruire un albero multicast a partire da una griglia, si potrebbe costruire l'albero usando rami della rete sovrapposta strutturata, ammortizzando in questo modo il costo della costruzione e della gestione della struttura, suddividendolo fra più applicazioni e più gruppi multicast.

9.4.3 Reti per la distribuzione di contenuti

Abbiamo già visto come il protocollo HTTP, eseguito al di sopra di TCP, consenta ai browser Web di recuperare pagine da server Web. Tuttavia, chiunque abbia atteso un'eternità per ottenere una pagina Web sa bene che il sistema è assai lontano dalla perfezione. Considerando che oggi il backbone di Internet è costituito di linee di collegamento OC-192 (a 10 Gbps), non è ovvio il motivo per cui ciò accada. C'è un generale consenso sul fatto che, quando si tratta di scaricare pagine Web, il sistema presenta tre colli di bottiglia:

- Il primo miglio: Internet può anche avere al proprio interno linee di collegamento ad alta capacità, ma ciò non vi aiuta a scaricare più velocemente una pagina Web se siete connessi con un modem a 56 Kbps.
- L'ultimo miglio: la linea che collega il server a Internet, insieme al server stesso, può essere sovraccaricata da troppe richieste.
- I punti di peering: i pochi fornitori di servizio Internet (ISP) che, insieme, realizzano il backbone di Internet possono avere linee ad alta capacità al proprio interno, ma hanno scarse motivazioni a fornire connessioni di elevata capacità verso i propri pari; se siete connessi all'ISP A e il server è connesso all'ISP B, la pagina che avete richiesto può essere ritardata nel punto di connessione tra A e B.

Per risolvere il primo problema soltanto voi potete fare qualcosa, mentre per il secondo e il terzo la duplicazione può essere utile. Un sistema che agisce in questo modo viene spesso chiamato *rete per la distribuzione di contenuti* (CDN, *content distribution network*), delle quali le due più note sono forse Akamai e Digital Island.

L'idea di una CDN è quella di distribuire in senso geografico un insieme di *server supplenti* (*server surrogates*) che memorizzano in sistemi di cache le pagine normalmente gestite da un insieme di *server di base* (*backend server*). In questo modo, invece di avere milioni di utenti che attendono all'infinito una risposta da www.cnn.com nel momento in cui si diffondono una nuova notizia molto importante (una situazione nota con il nome di *flash crowd*, "folla improvvisa"), è possibile suddividere tale carico fra molti server. Inoltre, invece di dover attraversare le reti di più ISP per raggiungere www.cnn.com, se questi server supplenti si trovano in tutti i fornitori del backbone, sarà possibile raggiungerne uno senza dover attraversare un punto di peering. Ovviamente, la gestione di migliaia di server supplenti nell'intera Internet è troppo costosa per qualsiasi sito che voglia fornire un miglior accesso alle proprie pagine Web: le CDN commerciali forniscono questo servizio a molti siti, ammortizzando così il costo fra molti clienti.

Sebbene questi server vengano chiamati "supplenti", sono in realtà dei sistemi di cache: se non hanno una copia della pagina richiesta dal client, la chiedono al server di *backend*. Nella realtà, però, sono i server di *backend* che duplicano attivamente i propri dati nei server

supplenti, piuttosto che aspettare di ricevere le richieste. Ai server supplenti vengono distribuite, solitamente, soltanto le pagine statiche, e non i contenuti dinamici: per qualsiasi contenuto che cambi frequentemente (come i risultati sportivi e le quotazioni di borsa) o che sia il risultato di una qualche elaborazione (come un'interrogazione a base di dati), i client devono rivolgersi direttamente ad un server di backend.

Avere un vasto insieme di server distribuiti geograficamente non risolve completamente il problema. Per completare il quadro, le CDN devono anche mettere a disposizione un insieme di *redirector* ("re-indirizzatori") che inoltrino le richieste dei client al server più adeguato, come mostrato in Figura 9.27. L'obiettivo principale dei redirector è, per ogni richiesta, la selezione del server che dia luogo al miglior *tempo di risposta* per il client. Un secondo obiettivo è che il sistema, nel suo complesso, sia in grado di elaborare tante richieste al secondo quante l'hardware sottostante (linee di collegamento nella rete e server Web) è in grado di sostenere. Il numero medio di richieste che possono essere soddisfatte in un certo periodo di tempo (noto come *throughput del sistema*) è di interesse soprattutto quando il sistema sta sostenendo un carico molto elevato, ad esempio quando una "folla improvvisa" accede ad un ristretto insieme di pagine oppure un attaccante sta aggredendo un sito mediante DDoS (*distributed denial of service*), come accadde a CNN, Yahoo e parecchi altri siti di elevato profilo nel febbraio 2000.

Le CDN tengono conto di vari fattori per decidere come distribuire le richieste dei client. Ad esempio, per minimizzare il tempo di risposta, un redirector potrebbe selezionare un server in base alla sua vicinanza all'interno della rete (*network proximity*). Al contrario, per migliorare il throughput del sistema globale, è preferibile *bilanciare* in modo omogeneo il carico su un insieme di server. Sia il throughput che il tempo di risposta vengono migliorati se il meccanismo di distribuzione tiene conto del principio di *località*, cioè seleziona un

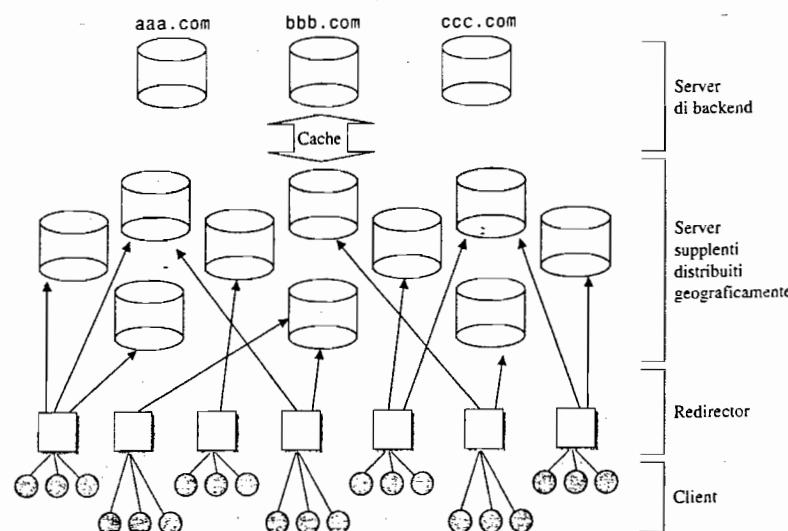


Figura 9.27 Elementi di una rete per la distribuzione di contenuti (CDN).

server che probabilmente ha già nella propria cache una copia della pagina richiesta. L'esatta combinazione di questi fattori che dovrebbe essere impiegata in una CDN è ancora aperta alla discussione. Questa sezione considera alcune delle possibilità.

Meccanismi

Per come lo abbiamo descritto fino a questo momento, un redirector è soltanto una funzione astratta, sebbene sembri assomigliare in qualche modo a quello che si potrebbe chiedere ad un router, dal momento che inoltra dal punto di vista logico un messaggio di richiesta, in modo molto simile ad un router che inoltra pacchetti. In realtà, si possono usare diversi meccanismi per implementare questo re-indirizzamento. Notate che in questa presentazione ipotizzeremo che ogni redirector conosca l'indirizzo di tutti i server disponibili (da questo punto in poi non useremo più l'aggettivo "supplente" e parleremo semplicemente in termini di un insieme di "server"). In sostanza, deve esserci una qualche forma di comunicazione fuori linea che tenga aggiornata questa informazione, dal momento che i server vengono aggiunti ed eliminati.

Innanzitutto, il re-indirizzamento potrebbe essere realizzato estendendo il sistema DNS, in modo che fornisca ai client indirizzi diversi per il server. Ad esempio, quando un client chiede che venga tradotto il nome www.cnn.com, il server DNS potrebbe restituire l'indirizzo IP di un server che ospita le pagine Web di CNN e che in quel momento è sottoposto al minor carico. In alternativa, per un certo insieme di server, potrebbe restituire semplicemente gli indirizzi relativi a rotazione (*round-robin*). Notate che la precisione della suddivisione del carico che si ottiene con un re-indirizzamento basato sul DNS è solitamente a livello di sito (ad esempio, cnn.com) piuttosto che di specifico URL (ad esempio, <http://www.cnn.com/2002/WORLD/europe/06/21/william.birthday/index.html>). Tuttavia, quando restituisce un collegamento ipertestuale interno ad una pagina, un server può sempre riscrivere il relativo URL, rinviando così, in sostanza, il client al server più adeguato per quello specifico oggetto.

Le CDN commerciali usano essenzialmente una combinazione della riscrittura di URL e del re-indirizzamento basato su DNS. Per motivi di scalabilità, il server DNS di alto livello punta ad un server DNS di livello regionale, il quale risponde con il vero indirizzo del server. Per poter reagire rapidamente alle modifiche, i server DNS impostano ad un valore molto breve (ad esempio, 20 secondi) il tempo di vita (TTL) dei record di risorsa che restituiscono: ciò è necessario per fare in modo che i client non tengano traccia dei risultati delle traduzioni del DNS, cosa che impedirebbe loro di rivolgersi nuovamente al server DNS per ottenere la più recente corrispondenza tra URL e server.

Un'altra possibilità è l'utilizzo della caratteristica di re-indirizzamento del protocollo HTTP: il client invia un messaggio di richiesta ad un server, il quale risponde indicando un nuovo (e migliore) server che il client deve contattare per ottenere la pagina richiesta. Sfortunatamente, il re-indirizzamento basato sul server richiede un tempo di round-trip addizionale attraverso Internet e, cosa ancor peggiore, espone i server al rischio di essere sovraccaricati dal servizio di re-indirizzamento stesso. Diversamente, se esiste un nodo vicino al client (ad esempio, un proxy Web locale) che sia a conoscenza dei server disponibili, può intercettare il messaggio di richiesta e istruire il client perché richieda, invece, la pagina al server più appropriato. In questo caso, il redirector si deve trovare in un punto molto congestionato, in modo che tutte le richieste che escono dal sito passino attraverso di esso, oppure il client deve cooperare, indirizzando esplicitamente le richieste al proxy (come succede con i proxy classici, diversamente dai proxy trasparenti).

A questo punto vi potrete chiedere cosa abbiano a che fare le CDN con le reti sovrapposte: anche se ritenere che una CDN sia una rete sovrapposta è in qualche modo una forzatura, queste due categorie di reti hanno un importante caratteristica in comune. Come un nodo di una rete sovrapposta, un redirector basato su proxy prende decisioni di instradamento a livello di applicazione: invece di inoltrare un pacchetto in base ad un indirizzo e in base alla propria conoscenza della topologia della rete, inoltre le richieste HTTP in base all'URL che contengono e alla propria conoscenza della localizzazione e del carico di un insieme di server. L'architettura di Internet di oggi non fornisce un supporto diretto per il re-indirizzamento (dove per "diretto" intendiamo dire che il client potrebbe inviare la richiesta HTTP direttamente al redirector, che la inoltra verso la destinazione), per cui il re-indirizzamento viene tipicamente realizzato indirettamente, facendo in modo che il redirector restituisca l'indirizzo di destinazione più adeguato e che il client contatti tale server per conto proprio.

Strategie

Consideriamo ora alcuni esempi di strategie di gestione che possono essere implementate dai redirector per inoltrare le richieste. In realtà, abbiamo già suggerito una delle strategie più semplici: a rotazione (*round-robin*). Uno schema simile richiederebbe, semplicemente, la selezione casuale di uno dei server disponibili. Entrambi questi approcci sono ben efficaci nella distribuzione omogenea del carico all'interno della CDN, ma non sono altrettanto efficaci nel ridurre il tempo di risposta percepito dal client.

Nessuno di questi due schemi, ovviamente, tiene conto del principio di "prossimità" all'interno della rete fisica, né, cosa ancor più grave, del principio di località: richieste relative al medesimo URL vengono inoltrate a server diversi, rendendo meno probabile il fatto che una pagina venga reperita nella cache presente nella memoria del server selezionato e costringendo il server a recuperare la pagina dal proprio disco o addirittura dal server di backend. Come può, senza un coordinamento globale, un insieme distribuito di redirector fare in modo che richieste relative alla stessa pagina vengano inoltrate allo stesso server (o ad un ristretto insieme di server)? La risposta è sorprendentemente semplice: tutti i redirector usano una qualche forma di hashing per indurre una corrispondenza deterministica fra gli URL e un piccolo insieme di valori. Il principale vantaggio di questo approccio è che non richiede alcuna comunicazione fra i redirector per il coordinamento delle operazioni: indipendentemente da quale sia il redirector che riceve la richiesta per un certo URL, il procedimento di hashing genera sempre lo stesso valore.

A questo punto, qual è uno schema di hashing di buona qualità? Il classico schema di hashing "a modulo", che trasforma ogni URL in un numero intero e poi considera il resto della divisione di tale numero per il numero di server, non è adatto a questa situazione, perché se il numero di server cambia, il calcolo del modulo si riflette in una riduzione della percentuale di pagine che rimangono associate al medesimo server. Anche se non ci si aspetta che il numero di server cambi frequentemente, il fatto che l'aggiunta di nuovi server provochi ingenti modifiche nelle associazioni tra pagine e server è un fenomeno indesiderato.

Un'alternativa consiste nell'utilizzo del medesimo algoritmo di *hashing coerente* (*consistent hashing*) di cui abbiamo parlato nella Sezione 9.4.2. In particolare, ogni redirector assegna per prima cosa ad ogni server un valore di hash all'interno del cerchio unitario. Quindi, per ogni URL che arriva, il redirector genera un valore di hash all'interno dello stesso cerchio unitario e assegna tale URL al server che si trova più vicino a tale valore di hash, lungo la circonferenza. Se, in questo schema, un nodo si guasta, il suo carico viene trasferito automaticamente ai suoi vicini (lungo la circonferenza unitaria), per cui l'aggiunta o la rimozione di

un server provoca modifiche all'assegnazione delle richieste solamente in ambito locale. Notate che, diversamente dal caso delle reti peer-to-peer, nelle quali un messaggio viene instradato da un nodo ad un altro per trovare il server il cui identificativo sia più vicino a quello dell'oggetto cercato, in questo caso ogni redirector conosce l'associazione tra tutti i server e i punti del cerchio unitario, per cui ogni redirector è in grado di selezionare il server "più vicino" indipendentemente dagli altri redirector.

Questa strategia può essere facilmente estesa per tenere in considerazione il carico dei singoli server. Ipotizzate che il redirector sia a conoscenza del carico, nel momento attuale, di ciascuno dei server disponibili. Questa informazione può non essere perfettamente aggiornata: possiamo immaginare che il redirector conti semplicemente quante richieste ha inoltrato a ciascun server negli ultimi secondi, usando questo conteggio come stima del carico, nel momento attuale, di ogni server. Quando riceve un URL, il redirector calcola il valore di hash che corrisponde a tale URL concatenato all'indirizzo di ciascun server, ordinando l'elenco di valori risultanti. Questo elenco ordinato definisce in modo efficace l'ordine in cui il redirector deve prendere in esame i server disponibili, per cui l'elenco viene scandito dall'alto in basso finché non si trova un server che abbia un carico inferiore ad una certa soglia. Il vantaggio di questo approccio, confrontato con il normale hashing coerente, è che l'ordine dei server è diverso per ogni URL, per cui, se un server si guasta, il suo carico viene distribuito in modo uniforme fra tutti gli altri server. Questo approccio è la base per il protocollo CARP (Cache Array Routing Protocol) e il relativo pseudocodice è qui presentato:

```
SelectServer(URL, S)
  for each server  $s_i$  nell'insieme di server  $S$ 
    weight $_i$  = hash(URL, address( $s_i$ ))
    sort( $\text{weight}$ )
    for each server  $s_j$  in ordine decrescente di  $\text{weight}_j$ 
      if Load( $s_j$ ) < soglia then
        return  $s_j$ 
  return server con il più alto weight
```

All'aumentare del carico, questo schema passa dall'utilizzo del solo primo server dell'elenco ordinato alla suddivisione omogenea delle richieste fra tutti i server; alcune pagine, normalmente gestite da server "impegnati", inizieranno ad essere gestite da server meno carichi. Poiché il processo è basato sul carico aggregato di ciascun server piuttosto che sulla popolarità delle singole pagine, i server che ospitano alcune delle pagine più richieste troveranno un numero sempre maggiore di server con cui suddividere il carico, rispetto a quanto accadrà per quei server che ospitano pagine complessivamente poco richieste. Durante il funzionamento del sistema, alcune pagine poco popolari verranno replicate semplicemente perché erano ospitate, in origine, su server che sono divenuti sovraccarichi. Al tempo stesso, se alcune pagine diventano molto popolari, è ipotizzabile che tutti i server del sistema ne abbiano una copia.

Infine, è possibile introdurre nell'equazione anche la prossimità in rete, in almeno due modi diversi. Il primo modo consiste nell'attenuare la distinzione fra carico del server e prossimità in rete, osservando il tempo impiegato da un server per rispondere alle richieste e usando tale misura al posto del parametro "carico del server" nell'algoritmo precedente. Questa strategia tende a preferire server vicini e/o poco carichi rispetto a server lontani e/o molto carichi. Un secondo approccio consiste nell'inserire la prossimità come fattore che influenza

la decisione in uno stadio preliminare, limitando l'insieme dei server candidati che viene preso in considerazione dall'algoritmo (S), includendo soltanto i server vicini. Il problema più difficile consiste nel decidere quali, fra tutti i server potenziali, siano sufficientemente vicini. Una possibilità è quella di selezionare soltanto quei server che sono resi disponibili dallo stesso ISP che serve il client. Un approccio un po' più sofisticato potrebbe prendere in considerazione la mappa dei sistemi autonomi generata da BGP e selezionare soltanto quei server che si trovano ad una distanza inferiore ad un certo numero di salti a partire dal client. Il modo corretto per bilanciare la prossimità in rete e la località della cache dei server è ancora argomento di ricerca.

9.5 Riepilogo

Abbiamo visto quattro protocolli per applicazioni basate sul paradigma client/server: il protocollo DNS usato dal sistema dei nomi di dominio, il protocollo SMTP usato per lo scambio di messaggi di posta elettronica, il protocollo HTTP usato per navigare il World Wide Web e il protocollo SNMP usato per interrogare nodi remoti, con lo scopo di gestire la rete. Abbiamo anche visto un insieme di protocolli del livello di applicazione, fra i quali RTP e SIP, usati per controllare e far funzionare applicazioni a flusso multimediale, come vic e vat, oltre che per l'emergente servizio di telefonia in Internet (voice-over-IP). Infine, abbiamo dato un'occhiata ad applicazioni emergenti (reti sovrapposte, reti peer-to-peer e reti per la distribuzione di contenuti), che mischiano elaborazione applicativa e inoltro di pacchetti con modalità innovative.

I protocolli applicativi sono piuttosto curiosi. Per molti aspetti le applicazioni tradizionali di tipo client/server sono simili ad un ulteriore livello di trasporto, tranne per il fatto che hanno, al proprio interno, conoscenze specifiche per una determinata applicazione. Si potrebbe dire che sono semplicemente protocolli di trasporto specializzati e che si stratificano protocolli di trasporto uno sopra all'altro fino a riprodurre l'esatto servizio richiesto da una particolare applicazione. Analogamente, i protocolli per reti sovrapposte e per reti peer-to-peer possono essere considerati come un modo per fornire un'infrastruttura di instradamento alternativa, ancora una volta progettata per soddisfare le necessità di una particolare applicazione. L'unico insegnamento che possiamo certamente trarre da questa osservazione è che il progetto di protocolli al livello di applicazione non è molto diverso dal progetto di protocolli per il cuore della rete: meglio si sono compresi gli ultimi, meglio si progetteranno i primi.

Problema aperto Una nuova architettura di rete

Non è facile puntare il dito su uno specifico problema aperto nell'ambito dei protocolli applicativi: l'intero settore è aperto a nuove applicazioni che vengono inventate ogni giorno e le necessità di queste applicazioni, dal punto di vista della rete, sono, diciamo così, strettamente dipendenti dall'applicazione stessa. La vera sfida per i progettisti di reti di calcolatori consiste nel riconoscere che ciò che le applicazioni chiedono alla rete si modifica nel tempo e che queste modifiche portano ad inevitabili evoluzioni nei protocolli di trasporto che vengono sviluppati e nelle funzionalità che vengono inserite nei router.

Lo sviluppo di nuovi protocolli di trasporto è un problema di difficoltà ragionevole. Può darsi che non siate in grado di convincere IETF ad imporre il proprio beneliscito sul vostro protocollo di trasporto alla stessa stregua di TCP o di UDP, ma certamente non c'è niente che

Ulteriori letture

vi impedisca di progettare la migliore applicazione multimediale al mondo, basata su un nuovo protocollo end-to-end al di sopra di UDP, in modo simile a quanto avviene per RTP.

È molto più difficile, invece, far penetrare all'interno della rete (nei router) conoscenze specifiche per un'applicazione, perché, per poter essere sfruttato da una particolare applicazione, un nuovo servizio o funzionalità di rete deve essere inserito in molti, se non tutti, router di Internet. Le reti sovrapposte forniscono un modo per introdurre nuove funzionalità nella rete senza la cooperazione di tutti i router (o anche solo di alcuni), ma, nel lungo periodo, è probabile che l'architettura della rete sottostante avrà bisogno di essere modificata per gestire queste reti sovrapposte. Abbiamo visto questo problema a proposito di RON (di come la selezione dei percorsi di RON e di BGP interagiscono fra loro) e ci si può aspettare che questo divenga un problema generale nel momento in cui le reti sovrapposte inizieranno a prendere il sopravvento.

Una possibilità è che si assista all'evoluzione verso una nuova architettura di rete *rigida* oppure che, invece, la prossima architettura di rete sia altamente adattativa. Al limite, invece di definire un'infrastruttura per il trasporto di pacchetti di dati, l'architettura della rete potrebbe consentire ai pacchetti il trasporto sia di dati che di codice (o di puntatori a codice) che dicono ai router come va elaborato il pacchetto. Una rete di questo tipo fa emergere un nuovo insieme di problemi, non ultimo dei quali il modo di garantire la sicurezza in un mondo dove qualunque applicazione possa sostanzialmente programmare i router.

Ulteriori letture

La pubblicazione su cui si basano i protocolli del livello di applicazione è quella di Clark e Tennenhouse, che viene citata dai progettisti di RTP come la loro linea guida. Lo sviluppo del sistema DNS è ben descritto da Mockapetris e Dunlap. Sebbene le reti sovrapposte e le reti peer-to-peer siano ancora un'area emergente, gli ultimi sei lavori di ricerca elencati forniscono un valido punto di partenza per comprendere l'argomento.

- Clark, D., e D. Tennenhouse "Architectural considerations for a new generation of protocols", *Proceedings of the SIGCOMM '90 Symposium*, pag. 200-208, September 1990.
- Mockapetris, P., e K. Dunlap "Development of the domain name system", *Proceedings of the SIGCOMM '88 Symposium*, pag. 123-133, August 1988.
- Karger, D., E. Lehman, F.T. Leighton, R. Panigrahy, M. Levine e D. Lewin "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web", *Proceedings of the ACM Symposium on Theory of Computing*, pag. 654-663, 1997.
- Chu, Y., S. Rao e H. Zhang "A case for end system multicast", *Proceedings of the ACM SIGMETRICS '00 Conference*, pag. 1-12, June 2000.
- Andersed, D., H. Balakrishnan, F. Kaashoek e R. Morris "Resilient overlay networks", *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pag. 131-145, October 2001.
- Rowstron, A., e P. Druschel "Storage management and caching in PAST, a large-scale persistent peer-to-peer storage utility", *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pag. 188-201, October 2001.
- Stoica, I., R. Morris, D. Karger, F. Kaashoek e H. Balakrishnan "Chord: A peer-to-peer lookup service for Internet applications", *Proceedings of the ACM SIGCOMM Conference*, pag. 149-160, August 2001.

- Ratnasamy, S., P. Francis, M. Handley, R. Karp e S. Shenker "A scalable content-addressable network", *Proceedings of the ACM SIGCOMM '01*, pag. 161-172, August 2001.

Ci sono parecchi lavori sul problema dell'assegnazione dei nomi, così come sul problema correlato della ricerca di risorse (trovare, prima di tutto, quali risorse esistono). Studi generali sul problema dei nomi si possono trovare in Terry [Ter86], Comer e Peterson [CP89], Birrell *et al.* [BLNS82], Saltzer [Sal78], Shoch Sho78] e Watson [Wat81]; sistemi di nomi basati su attributi (descrittivi) sono descritti in Peterson [Pet88] e Bowman *et al.* [BPY90], mentre la ricerca di risorse è l'argomento trattato in Bowman *et al.* [BDMS94].

Il protocollo SMTP venne originariamente definito nel documento RFC 821 [Pos82] mentre, ovviamente, le specifiche RFC 822 si trovano nel documento RFC 822 [Cro82]. Lo standard MIME è definito in una serie di documenti RFC; la specifica originaria si trova in RFC 1521 [BF93], mentre la versione più recente è definita in RFC 2045 [FB96].

La versione 1.0 del protocollo HTTP è stata specificata nel documento RFC 1945 [BLFF96], mentre la versione più recente (1.1) è definita in RFC 2068 [FGMBL97]. Ci sono moltissime pubblicazioni che trattano delle prestazioni del Web, in particolare del caching per il Web. Un valido esempio di questi è i lavori di Danzig sul traffico Web e le sue deduzioni sull'efficacia del caching [Dan98].

La gestione della rete è un campo così vasto e importante che IETF vi dedica un'intera area e troviamo ben più di 100 documenti RFC che descrivono i vari aspetti di SNMP e MIB. Le due citazioni fondamentali, però, sono Case *et al.* [CMRW93], che definisce la versione 2 di SNMP (SNMPv2), e McCloghrie e Rose [MR91], che definisce la seconda versione delle variabili MIB obbligatorie (MIB-II). Molti degli altri documenti RFC relativi a SNMP/MIB definiscono estensioni all'insieme fondamentale delle variabili MIB (ad esempio, variabili specifiche per una particolare tecnologia di rete oppure per un prodotto di un particolare produttore). Perkins e McGinnis [PM97] presentano una buona introduzione a SNMP e MIB.

Il protocollo RTP è descritto nel documento RFC 1889 [SCFJ96], anche se molti dei dettagli più interessanti si trovano in documenti preliminari (*draft*) di Internet che non sono ancora stati pubblicati. McCanne e Jacobson [MJ95] descrivono vic, una delle applicazioni che usano RTP.

Il protocollo SIP è definito nel documento RFC 3261 [SCJ+02], che contiene un utile sezione a carattere introduttivo, oltre alla specifica dettagliata del protocollo.

Il rapporto del National Research Council sull'irrigidimento di Internet si può trovare in [NRC01], mentre Peterson *et al.* [PACR02] hanno proposto l'utilizzo delle reti sovrapposte per introdurre tecnologie fortemente innovative. Lo studio originario per individuare percorsi in modo alternativo rispetto a BGP fu presentato da Savage *et al.* [SCH+99]. L'idea di usare il DNS per bilanciare in modo omogeneo un insieme di server venne descritto nel documento RFC 1784 [Bri95]. Il protocollo CARP (Cache Array Routing Protocol) è definito in un Internet Draft [CPVR97]. Una trattazione onnicomprensiva del problema del caching per il Web in confronto a server replicati si può trovare nel libro di Rabinovich e Spatscheck [RS02]. Wang *et al.* esplorano lo spazio di progetto per i redirector [WPP02].

Infine, raccomandiamo il seguente riferimento attivo per tenere il passo con la rapida evoluzione del Web:

- <http://www.w3.org>: il World Wide Web Consortium

Esercizi

- Sia ARP che DNS dipendono da sistemi di cache; i tempi di vita tipici dei dati nella cache di ARP sono di 10 minuti, mentre per il DNS di parla di alcuni giorni. Fornite giustificazioni per questa differenza. Quali indesiderabili conseguenze di potrebbero essere in caso di tempo di vita troppo lungo per i dati presenti nella cache del DNS?
- IPv6 rende praticamente inutile il lavoro di ARP, consentendo l'inserimento degli indirizzi hardware come parte dell'indirizzo IPv6. In che modo questo complica il lavoro del sistema DNS? Ci sono ripercussioni sul problema di identificare il server DNS locale?
- I server DNS consentono anche la ricerca all'inverso (*reverse lookup*): dato un indirizzo IP, ad esempio 128.112.169.4, si genera una stringa di testo che lo contiene in formato inverso, 4.169.112.128.in-addr.arpa e si effettua la ricerca usando i record PTR di DNS (che costituiscono una gerarchia di domini analoga a quella dei nomi di dominio). Supponete di voler autenticare il mittente di un pacchetto basandovi sul nome del suo host, essendo certi che l'*indirizzo IP* del mittente sia vero. Spiegate quanto sia poco sicuro convertire l'*indirizzo* del mittente in un nome con il sistema appena descritto, per poi confrontare il nome ottenuto con un elenco assegnato di host fidati. Suggerimento: Di quali server DNS vi fidereste?
- Qual è la relazione fra un nome di dominio (ad esempio, cs.princeton.edu) ed un numero di sottorete IP (per esempio, 192.12.69.0)? Tutti gli host della sottorete devono essere identificati dallo stesso *name server*? Cosa si può dire della ricerca all'inverso (*reverse lookup*) di cui abbiamo parlato nell'esercizio precedente?
- Supponete che un host decida di usare, per la traduzione degli indirizzi, un server per i nomi che non sia interno alla propria organizzazione. In relazione a richieste che non trovino risposta in alcuna cache DNS, quali sono le situazioni in cui questa scelta non genera un traffico totale maggiore di ciò che avverrebbe usando un server per i nomi locali? Quando questa scelta può produrre una più elevata frequenza di risposte trovate nella cache DNS (*cache hit rate*) ed eventualmente anche un traffico totale inferiore?
- La Figura 9.4 mostra la gerarchia dei server per la traduzione dei nomi. Come rappresentereste questa gerarchia se un server servisse più zone? In tale situazione, in che relazione sarebbe la gerarchia dei server per i nomi rispetto alla gerarchia delle zone? Come pensate di gestire il fatto che ogni zona può avere più server per i nomi?
- Usate il programma di servizio/utility whois per scoprire il responsabile del vostro sito, almeno per quanto concerne InterNIC. Cercate il vostro sito sia mediante il suo nome registrato nel sistema DNS, sia mediante il numero di rete IP; per quest'ultima informazione può darsi che dobbiate usare un server whois alternativo (ad esempio, con il comando whois -h whois.arin.net ...). Fate la stessa ricerca anche per princeton.edu e cisco.com.
- Molte piccole organizzazioni fanno gestire i propri siti Web da altri. Come pensate di poter utilizzare il programma whois per scoprire se si tratta di un caso come questo e, in tal caso, scoprire l'identità del vero gestore?
- Una caratteristica dell'attuale gerarchia .com nel sistema DNS è quella di essere enormemente "empia".
 - Proponete una riorganizzazione della gerarchia .com in senso maggiormente gerarchico. Quali obiezioni intravedete all'adozione della vostra proposta?
 - Quali potrebbero essere alcune delle conseguenze del fatto di avere la maggior

- parte dei nomi di dominio nel sistema DNS con quattro o più livelli, rispetto alla situazione attuale in cui sono solitamente due?
10. Supponete, al contrario, di abbandonare qualsiasi pretesa di avere un sistema DNS gerarchico, spostando semplicemente tutti i domini di tipo .com nel server radice: www.cisco.com diventerebbe www.cisco, o forse soltanto cisco. Quali riflessi avrebbe questa scelta, in generale, sul traffico che coinvolge il server radice? Quali riflessi ci sarebbero su tale traffico nel caso particolare della traduzione di un nome come cisco nell'indirizzo di un server Web?
 11. Quali problemi insorgono, nella cache del sistema DNS, cambiando l'indirizzo IP corrispondente al nome dell'host di un server Web? Come possono essere minimizzati tali problemi?
 12. Usate un opportuno programma che effettui ricerche nel sistema DNS (ad esempio, nslookup) e disabilitate la possibilità di ricerche ricorsive (ad esempio, con il comando set norecurse), in modo che, quando il programma invia una richiesta ad un server DNS e tale server non è in grado di fornire una risposta completa in base alle proprie informazioni, il server restituisca l'indirizzo del successivo server DNS che va interrogato piuttosto che inoltrare automaticamente la richiesta a tale server. Poi, portate a termine manualmente una traduzione di nome come quella di Figura 9.5; provate con il nome di host www.cs.princeton.edu. Elencate ogni server intermedio che avete contattato. Può darsi che dobbiate specificare che alcune richieste sono relative a record di tipo NS piuttosto che ai normali record di tipo A.
 13. Discutete come si potrebbe riscrivere il protocollo SMTP o il protocollo HTTP utilizzando un ipotetico protocollo generico di tipo richiesta/risposta (magari qualcosa di simile a CHAN RPC). Sarebbe possibile spostare dal livello applicativo al livello di trasporto qualcosa di analogo alle connessioni persistenti? Quali altri compiti specifici delle applicazioni si potrebbero trasferire all'interno di questo protocollo?
 14. La maggior parte dei client Telnet possono essere usati per connettersi alla porta 25, la porta SMTP, invece che alla porta Telnet. Usando uno di questi strumenti, connettetevi ad un server SMTP ed inviate a voi stessi (o a qualcun altro, dopo aver ottenuto il permesso) qualche messaggio di posta contraffatto. Quindi, esaminate le intestazioni che sono state prodotte per mettere in evidenza come il messaggio non sia originale.
 15. Quali caratteristiche di SMTP e/o di un server di posta come sendmail potrebbero essere utilizzate (o aggiunte) per fornire qualche forma di robustezza rispetto alle contraffazioni di posta elettronica come quelle viste nell'esercizio precedente?
 16. Scoprite come i server SMTP gestiscono i comandi sconosciuti provenienti dall'altra entità e come, in particolare, questo meccanismo consenta l'evoluzione del protocollo (ad esempio, verso un "SMTP esteso"). Potete leggere il documento RFC che descrive il protocollo SMTP, oppure contattare un server SMTP come nell'Esercizio 14 e verificare le sue risposte a fronte di comandi inesistenti.
 17. Come abbiamo detto nel testo, il protocollo SMTP prevede lo scambio di parecchi piccoli messaggi. Nella maggior parte dei casi, le risposte del server non hanno effetto su ciò che il client invia successivamente, per cui il client potrebbe utilizzare la tecnica del *command pipelining*: l'invio di più comandi all'interno di un unico messaggio.
 - a) Per quali comandi SMTP è *necessario* che il client faccia attenzione alle risposte del server?
 - b) Ipotizzate che il server legga ciascun messaggio proveniente dal client con la funzione gets() o una funzione equivalente, che legge una stringa fino al carattere

<LF>. Cosa dovrebbe fare per accorgersi del fatto che un client ha usato il *command pipelining*?

- c) È risaputo che la tecnica del *pipelining* crea problemi ad alcuni server; cercate di scoprire come un client possa negoziare l'utilizzo di questa tecnica.
 18. Scoprite quali altre caratteristiche sono consentite dai record di tipo MX di DNS, oltre a fornire un alias per un server di posta: quest'ultima funzione potrebbe, in fin dei conti, essere fornita anche da un record CNAME. I record MX sono stati progettati per la posta elettronica; sarebbe utile un analogo record di tipo "WEB" per il protocollo HTTP?
 19. Uno dei problemi più difficili che deve gestire un protocollo come MIME è la grande varietà dei formati di dati. Consultate il documento RFC che definisce MIME per scoprire come faccia MIME a gestire formati di immagini o di testo nuovi o specifici di alcuni sistemi.
 20. Lo standard MIME fornisce supporto per più rappresentazioni dello stesso contenuto usando la sintassi multipart/alternative; ad esempio, un testo potrebbe essere spedito come text/plain, textrichtext e application/postscript. Perché pensate che il testo semplice debba essere il *primo* formato, anche se l'implementazione potrebbe ritenere più semplice posizionarlo dopo gli altri formati nativi?
 21. Consultate il documento RFC che definisce MIME e scoprite come la codifica base64 gestisce i dati binari aventi una lunghezza non divisibile per tre byte.
 22. Nel protocollo HTTP versione 1.0, un server contrassegna la fine di un trasferimento chiudendo la connessione. Spiegate perché, relativamente allo strato TCP, questo costituiva un problema per il server. Scoprite come la versione 1.1 di HTTP abbia eliminato questo problema. Come si potrebbe risolvere questo problema in un più generale protocollo di tipo richiesta/risposta?
 23. Scoprite come configurare un server HTTP in modo da eliminare il messaggio 404 not found e restituire invece un messaggio predefinito (e auspicabilmente più utile per l'utente). Decidete se tale caratteristica debba far parte del protocollo o di una sua implementazione, oppure se sia tecnicamente permessa dal protocollo (nel sito www.apache.org potete trovare la documentazione relativa al server HTTP apache).
 24. Perché il comando HTTP GET di pagina 654
- ```
GET http://www.cs.princeton.edu/index.html HTTP/1.1,
```
- contiene il nome del server che viene contattato? Il server non conosce già il proprio nome? Usate Telnet, come nell'esercizio 14, per connettervi alla porta 80 di un server HTTP e scoprite cosa accade se non indicate il nome dell'host.
25. Quando un server HTTP invoca il comando close() alla propria estremità di una connessione, deve attendere nello stato FIN\_WAIT\_2 del protocollo TCP finché il client non chiude la sua estremità. Quale meccanismo del protocollo TCP può essere d'aiuto ad un server HTTP per gestire i client che non cooperano o che sono realizzati male e non chiudono la propria estremità della connessione? Se possibile, esaminate l'interfaccia di programmazione relativa a questo meccanismo e descrivete come un server HTTP la potrebbe utilizzare.
  26. Il protocollo POP3 (Post Office Protocol 3) consente ad un client di recuperare solamente la posta, usando una password per l'autenticazione. Tradizionalmente, per inviare posta un client dovrebbe semplicemente inviarla al proprio server ed attendersi che questa venga consegnata.

- a) Spiegate perché i server di posta spesso non permettono più questa consegna per affidamento da parte di un client qualsiasi.
- b) Proponete un'opzione di SMTP per l'autenticazione dei client remoti.
- c) Scoprite quali metodi esistenti sono disponibili per risolvere questo problema.
27. Supponete che un sito Web molto grande desideri implementare un meccanismo mediante il quale i client accedano, fra più server HTTP, a quello che sia "più vicino" in base a qualche metrica adeguata.
- a) Discutete lo sviluppo di un meccanismo, all'interno del protocollo HTTP, per fare ciò.
- b) Discutete lo sviluppo di un meccanismo, all'interno del sistema DNS, per fare ciò. Confrontate i due meccanismi. È possibile adottare uno dei due approcci senza modificare il browser?
28. Scoprite se avete accesso ad un nodo SNMP che risponda alle interrogazioni che gli inviate. Se lo trovate, usate qualche programma di utilità per SNMP (ad esempio, la suite *ucd-snmp*) e fate queste prove:
- a) Scaricate l'intero gruppo *system*, usando qualcosa di questo tipo
- ```
snmpwalk nomeDelNodo public system
```
- Provate il comando precedente un'altra volta, indicando 1 al posto di *system*.
- b) Navigate manualmente nel gruppo *system*, usando più operazioni SNMP GET-NEXT (usando cioè *snmpgetnext* o un comando equivalente), recuperando un dato per volta.
29. Usando il dispositivo SNMP e i programmi descritti nell'esercizio precedente, scaricate il gruppo *tcp* (numericamente, il gruppo 6) o un altro gruppo. Poi, eseguite qualche azione che provochi un cambiamento in alcuni contatori del gruppo, e scaricate nuovamente il gruppo, per evidenziare la modifica. Cercate di farlo in un modo che vi garantisca che siano le vostre azioni a provocare la modifica dei dati.
30. Quali informazioni, fra quelle fornite da SNMP, potrebbero essere utili perché qualcuno possa pianificare l'attacco di *IP spoofing* dell'Esercizio 19 del Capitolo 5? Quali altre informazioni di SNMP potrebbero essere considerate sensibili?
31. Cercate di identificare situazioni nelle quali un'applicazione RTP potrebbe ragionevolmente comportarsi nel modo seguente:
- inviare, praticamente nello stesso istante, più pacchetti che devono avere un'indicazione oraria diversa
 - inviare, in momenti diversi, pacchetti che devono avere la stessa indicazione oraria
- Motivate, di conseguenza, perché le indicazioni orarie (*timestamp*) di RTP devono, almeno in alcuni casi, essere fornite (almeno indirettamente) dall'applicazione. Suggerimento: pensate a casi in cui la velocità di trasmissione e la velocità di riproduzione non coincidono.
32. Per garantire una riproduzione accurata, la minima risoluzione temporale necessaria per le indicazioni orarie (*timestamp*) sarebbe quella di un tempo di frame o di un tempo di campionamento vocale, ma solitamente l'intervallo di tempo che viene usato è considerevolmente inferiore. Come mai?
33. Supponete di voler fare in modo che le notifiche di RTCP restituite dai ricevitori ammontino a non più del 5% del flusso primario uscente di RTP. Se ogni notifica è di 84 byte, il traffico RTP è di 20 Kbps e ci sono 1000 ricevitori, quanto spesso viene inviata una notifica da un singolo ricevitore? E quanto spesso nel caso in cui ci fossero 10000 ricevitori?

34. Il documento RFC 1889 specifica che l'intervallo di tempo che intercorre tra due notifiche RTCP inviate da un ricevitore contiene un fattore di casualità, per evitare che tutti i ricevitori invino una notifica nello stesso istante. Se tutti i ricevitori inviasero le proprie notifiche nel medesimo sottointervallo ampio il 5% dell'intervallo di replica, il flusso RTCP in arrivo presso la sorgente sarebbe di proporzioni paragonabili al flusso di dati RTP uscente dalla sorgente.
- a) I ricevitori video possono ragionevolmente attendere prima di inviare le proprie notifiche finché il processo a più elevata priorità che elabora e visualizza un frame non ha portato a termine il proprio compito; questo potrebbe significare che le loro trasmissioni RTCP sono sincronizzate sui confini dei frame. Potrebbe trattarsi di un problema serio?
- b) Con 10 ricevitori, qual è la probabilità che tutti spediscano le proprie notifiche in un particolare sottointervallo ampio il 5%?
- c) Con 10 ricevitori, qual è la probabilità che metà di essi spediscano in un particolare sottointervallo ampio il 5%? Moltiplicate questo valore per 20 per ottenere una stima della probabilità che metà di essi effettui la trasmissione nel medesimo sottointervallo arbitrario ampio il 5%. Suggerimento: in quanti modi si possono scegliere 5 ricevitori in un insieme di 10?
35. A cosa servono, dal punto di vista pratico, ad un server i dati relativi alla frequenza di perdita di pacchetti e al jitter contenuti in una notifica inviata da un ricevitore?
36. Le applicazioni video vengono solitamente eseguite al di sopra di UDP piuttosto che di TCP perché non sono in grado di tollerare i ritardi dovuti alle ritrasmissioni. Tuttavia, ciò significa che le applicazioni video non sono soggette ai vincoli dell'algoritmo di controllo della congestione del protocollo TCP. Che impatto ha tutto ciò sul traffico TCP? State dettagliati sulle possibili conseguenze. Fortunatamente queste applicazioni video solitamente usano il protocollo RTP, che produce le "notifiche del ricevitore" di RTCP che vengono inviate alla sorgente del flusso periodicamente (ad esempio, una volta al secondo) e contengono la percentuale di pacchetti ricevuti con successo nell'ultimo intervallo di notifica. Descrivete come la sorgente possa utilizzare queste informazioni per modificare la propria velocità di trasmissione in un modo che sia compatibile con il protocollo TCP.
37. Supponete che alcuni ricevitori di un'ampia conferenza possano ricevere dati con un'ampiezza di banda significativamente più elevata di altri. Che cosa si potrebbe implementare per questo caso? Suggerimento: considerate sia il protocollo SAP (Session Announcement Protocol) sia la possibilità di utilizzare "miscelatori" (*mixer*) di terzi.
38. Proponete un meccanismo per decidere quando sia il caso di segnalare la perdita di un pacchetto RTP. Come si confronta il vostro meccanismo con il meccanismo di ritrasmissione adattativa del protocollo TCP presentato nella Sezione 5.2.6?
39. Come potreste codificare dati audio (o video) in due pacchetti in modo che se un pacchetto va perduto, la risoluzione viene semplicemente ridotta come se si avesse a disposizione un'ampiezza di banda pari alla metà di quella disponibile? Spiegate perché questo è molto più difficile se si usa una codifica di tipo JPEG.
40. Spiegate la relazione esistente fra i localizzatori omogenei di risorse (URL, *uniform resource locator*) e gli identificativi omogenei di risorse (URI, *uniform resource identifier*). Date un esempio di URI che non sia un URL.
41. A quali problemi va incontro un meccanismo di re-indirizzamento basato su DNS se si vuole selezionare il server più adeguato sulla base delle informazioni di carico attuale?

Soluzioni di esercizi scelti

Soluzione degli esercizi indicati con un segno di spunta nei vari capitoli del libro.

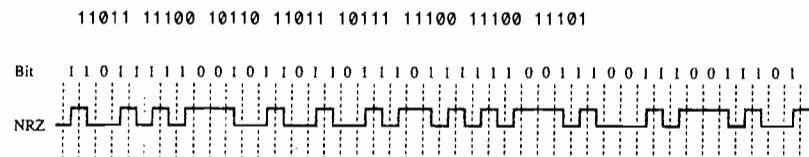
Capitolo 1

6. Considereremo completo un trasferimento quando l'ultimo bit di dati è giunto a destinazione.
 - a) $1.5 \text{ MB} = 12582912 \text{ bit}$. 2 RTT iniziali (160 ms) + $12582912/10000000 \text{ bps}$ (transmissione) + $\text{RTT}/2$ (propagazione) $\approx 1.458 \text{ secondi}$.
 - b) A quanto sopra aggiungiamo il tempo corrispondente a 1499 RTT (il numero di RTT che trascorrono dal momento in cui arriva il pacchetto numero 1 al momento in cui arriva il pacchetto numero 1000), per un totale di $1.46 + 119.92 = 121.38$.
 - c) Questo è 74.5 RTT più i 2 iniziali, per un totale di 6.12 secondi.
 - d) Subito dopo che è terminato l'*handshake* inviamo un pacchetto. Trascorso un intervallo pari a RTT dopo che è terminato l'*handshake* inviamo due pacchetti. Dopo n RTT dall'*handshake* iniziale abbiamo inviato $1 + 2 + 4 + \dots + 2^n = 2^{n+1} - 1$ pacchetti. Di conseguenza, per $n = 10$ siamo già stati in grado di inviare tutti i 1000 pacchetti; l'ultimo gruppo arriva dopo 0.5 RTT. Il tempo totale è $2 + 10.5 \text{ RTT}$, cioè 1 secondo.
8. Il ritardo di propagazione è $50 \times 10^3 \text{ m} / (2 \times 10^3 \text{ m/s}) = 250 \mu\text{s}$. Con 800 bit/250 μs si ottiene 3.2 Mbps. Per pacchetti di 512 byte, ciò porta a 16.4 Mbps.
16. a) Il ritardo di propagazione della linea è $(55 \times 10^9)/(3 \times 10^8) = 184 \text{ secondi}$. Di conseguenza, il valore di RTT è 268 secondi.
b) Il prodotto ritardo \times ampiezza di banda per la linea è pari a RTT \times l'ampiezza di banda = 23.5 Mb.
c) Dopo aver catturato un'immagine, occorre trasmetterla lungo la linea di collegamento e prima che il Controllo Missione la possa interpretare deve arrivare completamente. Il ritardo di trasmissione per 5 Mb di dati è 39 secondi. Di conseguenza, il tempo totale richiesto è il ritardo di trasmissione + il ritardo di propagazione = 223 secondi.

19. a) Per ogni linea, servono $1 \text{ Gbps}/5 \text{ Kb} = 5 \mu\text{s}$ per trasmettere il pacchetto lungo la linea stessa, dopo di che servono altri $10 \mu\text{s}$ perché l'ultimo bit si propaghi. Di conseguenza, per una LAN con un unico switch che inizi ad inoltrare un pacchetto soltanto dopo averlo ricevuto completamente, il ritardo totale per il trasferimento è uguale a due ritardi di trasmissione + due ritardi di propagazione = $30 \mu\text{s}$.
 b) Con tre switch e, di conseguenza, quattro linee di collegamento, il ritardo totale è pari a quattro ritardi di trasmissione + quattro ritardi di propagazione = $60 \mu\text{s}$.
 c) Con la tecnica "cut-through", uno switch deve soltanto decodificare i primi 128 bit prima di poter iniziare l'inoltro del pacchetto. Questo richiede un tempo pari a 128 ns. Questo ritardo sostituisce i ritardi di trasmissione dello switch nella risposta precedente, per un ritardo totale pari a un ritardo di trasmissione + tre ritardi di decodifica per il cut-through + quattro ritardi di propagazione = $45.384 \mu\text{s}$.
 29. a) $1920 \times 1080 \times 24 \times 30 = 1492992000 \approx 1.5 \text{ Gbps}$.
 b) $8 \times 8000 = 64 \text{ Kbps}$.
 c) $260 \times 50 = 13 \text{ Kbps}$.
 d) $24 \times 88200 = 216800 \approx 2.1 \text{ Mbps}$.

Capitolo 2

3. La codifica 4B/5B della sequenza di bit assegnata è la seguente:



7. Contrassegniamo con \wedge una posizione in cui è stato rimosso un bit 0 che era stato aggiunto. Si verifica un errore quando vengono rilevati sette valori 1 consecutivi (*err*). Alla fine della sequenza di bit, viene rilevata la fine del frame (*eof*).

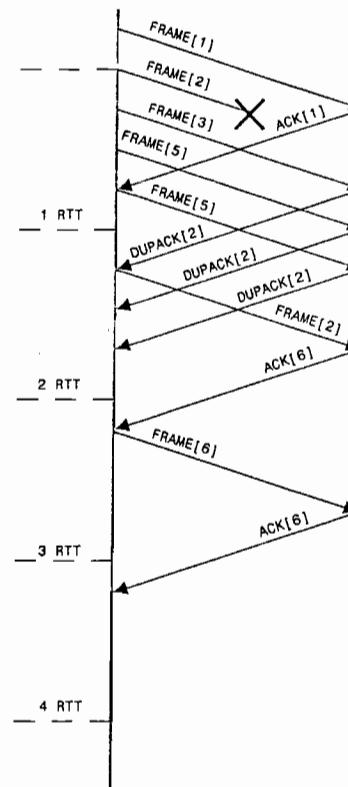
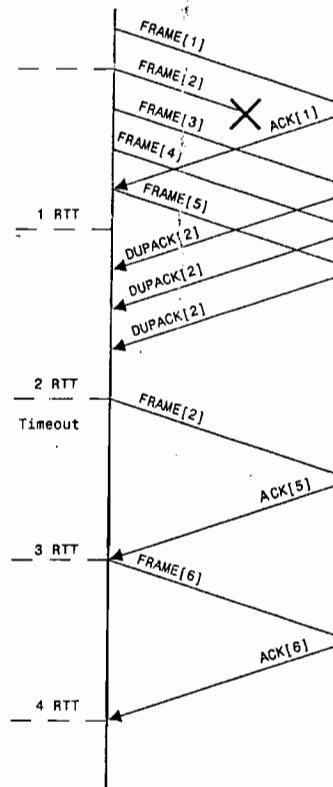
~~01101011111, 101001111111 *err* 0 110 01111110 *eof*~~

19. a) Prendiamo il messaggio 1011001010001011, aggiungiamo otto zeri alla fine e dividiamo per 1000001111 ($x^8 + x^2 + x^1 + 1$). Il resto è 10010011. Trasmettiamo il messaggio originale, aggiungendo tale resto alla fine, in questo modo:

101100100100001110010011

- b) Invertendo il primo bit si ottiene 001100100100101110010011. Dividendo per 1000001111 ($x^8 + x^2 + x^1 + 1$) si ottiene il resto 10110110.
 25. La latenza di sola andata della linea è 100 ms. Il prodotto ampiezza di banda \times ritardo di round trip è pari a circa 125 pacchetti/secondo \times 0.2 secondi, cioè 25 pacchetti. Il valore di SWS dovrebbe essere questo.

- a) Se RWS = 1, sono necessari 26 numeri di sequenza diversi, per cui servono almeno 5 bit.
 b) Se RWS = SWS, lo spazio dei numeri di sequenza deve essere almeno il doppio di SWS, cioè fino a 50, per cui servono 6 bit.
 32. La figura illustra la sequenza temporale degli eventi nel primo caso. Il secondo caso porta ad una riduzione del tempo totale di transazione di circa 1 RTT.



Capitolo 3

2. La tabella di pagina seguente è cumulativa; per ogni sezione dell'esercizio le tabelle dei VCI sono costituite dalle righe relative alla sezione e da tutte le righe delle sezioni precedenti.

Sezione dell'esercizio	Switch	Input		Output	
		Porta	VCI	Porta	VCI
(a)	1	0	0	1	0
	2	3	0	1	0
	4	3	0	0	0
(b)	2	0	0	1	1
	3	3	0	0	0
	4	3	1	1	0
(c)	1	1	1	2	0
	2	1	2	3	1
	4	2	0	3	2
(d)	1	1	2	3	0
	2	1	3	3	2
	4	0	1	3	3
(e)	2	0	1	2	0
	3	2	0	0	1
(f)	2	1	4	0	2
	3	0	2	1	0
	4	0	2	3	4

14. L'elenco seguente mostra le associazioni fra le LAN e i rispettivi bridge designati:

B1 A-interfaccia: A	B2-interfaccia: D (non C)	
B2 B1-interfaccia: A	B3-interfaccia: C	B4-interfaccia: D
B3 C-interfaccia: C	B2-interfaccia: A, D	
B4 D-interfaccia: D	B2-interfaccia: C (non A)	
B1 fuori uso		
B2 A, B, D		
B3 E, F, G, H		
B4 I		
B5 inattivo		
B6 J		
B7 C		

16. Tutti i bridge vedono i pacchetti che provengono da D e vanno verso C. Soltanto B3 e B4 vedono i pacchetti che provengono da C e vanno verso D. Soltanto B1, B2 e B3 vedono i pacchetti che provengono da A e vanno verso C.

33. Il collo di bottiglia è la velocità del bus di I/O, dato che è inferiore all'ampiezza di banda della memoria. L'ampiezza di banda che il bus di I/O è effettivamente in grado di fornire è $1000/2$ Mbps, perché ogni pacchetto attraversa il bus due volte. Di conseguenza, il numero di interfacce è $\lfloor 500/45 \rfloor = 11$.

Capitolo 4

5. Per definizione, il valore di MTU per il percorso è 512 byte. La massima dimensione del carico utile di IP è $512 - 20 = 492$ byte. Dobbiamo trasferire $2048 + 20 = 2068$ byte nel carico utile di IP, che verranno suddivisi in 4 frammenti di 492 byte ciascuno e 1 frammento di 100 byte. Se usiamo il valore di MTU del percorso ci sono, quindi, 5 pacchetti in totale. Nella configurazione precedente avevamo bisogno di 7 pacchetti.

16. a)

Informazioni memorizzate nel nodo	Distanza per raggiungere il nodo					
	A	B	C	D	E	F
A	0	2	∞	5	∞	∞
B	2	0	2	∞	1	∞
C	∞	2	0	2	∞	3
D	5	∞	2	0	∞	∞
E	∞	1	∞	∞	0	3
F	∞	∞	3	∞	3	0

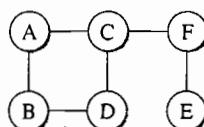
b)

Informazioni memorizzate nel nodo	Distanza per raggiungere il nodo					
	A	B	C	D	E	F
A	0	2	4	5	3	∞
B	2	0	2	4	1	4
C	4	2	0	2	3	3
D	5	4	2	0	∞	5
E	3	1	3	∞	0	3
F	∞	4	3	5	3	0

c)

Informazioni memorizzate nel nodo	Distanza per raggiungere il nodo					
	A	B	C	D	E	F
A	0	2	4	5	3	6
B	2	0	2	4	1	4
C	4	2	0	2	3	3
D	5	4	2	0	5	5
E	3	1	3	5	0	3
F	6	4	3	5	3	0

19. Ecco un esempio di topologia di rete:



22. Applicate ciascuna maschera di sottorete e, se il relativo numero di sottorete corrisponde alla colonna "Numero di sottorete", usate il dato presente in "Salto successivo".

- Applicando la maschera di sottorete 255.255.254.0, otteniamo 128.96.170.0. Per il salto successivo si usa l'interfaccia 0.
- Applicando la maschera di sottorete 255.255.254.0, otteniamo 128.96.166.0 (il salto successivo è verso R2). Applicando la maschera di sottorete 255.255.252.0, otteniamo 128.96.164.0 (il salto successivo è verso R3). Tuttavia, 255.255.254.0 è un prefisso di lunghezza maggiore, per cui usiamo R2 per il salto successivo.
- Nessuno dei numeri di sottorete corrisponde alla maschera, per cui usiamo il router di default, R4.
- Applicando la maschera di sottorete 255.255.254.0, otteniamo 128.96.168.0. Per il salto successivo si usa l'interfaccia 1.
- Applicando la maschera di sottorete 255.255.252.0, otteniamo 128.96.164.0. Il salto successivo è verso R3.

29.

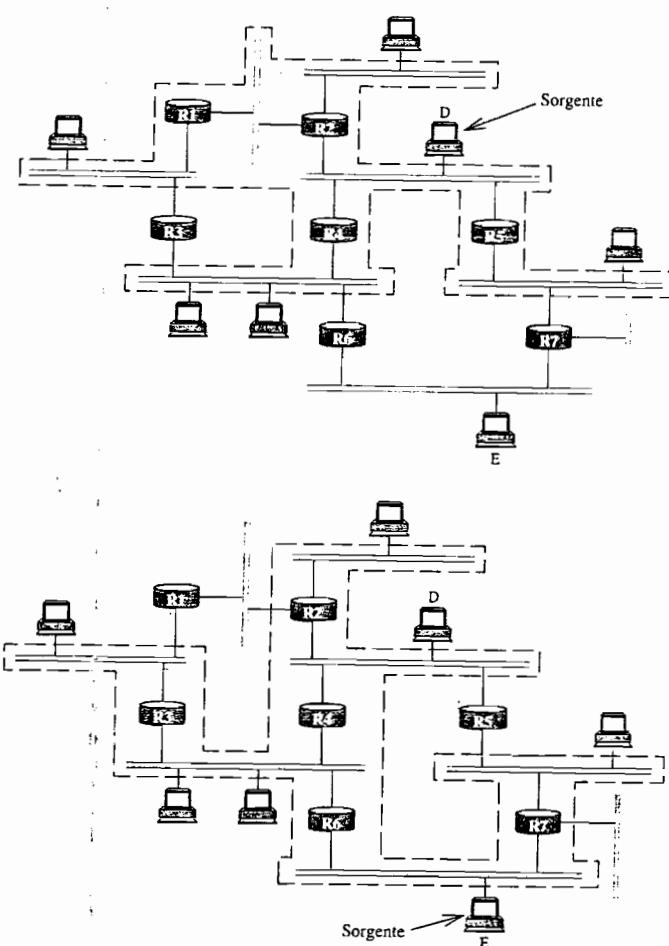
Passo	Confermato	Tentativo
1	(A,0,-)	
2	(A,0,-)	(B,1,B) (D,5,D)
3	(A,0,-) (B,1,B)	(D,4,B) (C,7,B)
4	(A,0,-) (B,1,B) (D,4,B)	(C,5,B) (E,7,B)
5	(A,0,-) (B,1,B) (D,4,B) (C,5,B)	(E,6,B)
6	(A,0,-) (B,1,B) (D,4,B) (C,5,B) (E,6,B)	

46. (a) F (b) B (c) E (d) A (e) D (f) C

55. La figura riportata a pagina seguente indica le porte attive.

Capitolo 5

10. La finestra pubblicizzata dovrebbe essere grande a sufficienza da mantenere piena la linea; in questo caso il prodotto ritardo (RTT) × ampiezza di banda è pari a $140 \text{ ms} \times 1 \text{ Gbps} = 140 \text{ Mb} = 17.5 \text{ MB}$ di dati. È quindi necessario un campo AdvertisedWindow di 25 bit ($2^{25} = 33554432$). Il campo del numero di sequenza non deve tornare al valore iniziale durante il tempo di vita del segmento di dimensione massima. In 60 secondi si possono trasmettere 7.5 GB; 33 bit consentono uno spazio di numeri di



sequenza pari a 8.6 GB, per cui questi ultimi non torneranno al proprio valore iniziale in 60 secondi.

- $2^{32} \text{ byte}/(5 \text{ GB}) = 859 \mu\text{s}$.
 - 1000 istanti di tempo in 859 ms significa un istante ogni 859 μs , per cui si torna al valore iniziale dopo 3.7 Ms, circa 43 giorni.
27. Usando il valore iniziale Deviation = 50, servono 20 iterazioni perché TimeOut scenda al di sotto di 300.0.

Iterazione	SampleRTT	EstRTT	Dev	diff	TimeOut
0	200.0	90.0	50.0		
1	200.0	103.7	57.5	110.0	333.7
2	200.0	115.7	62.3	96.3	364.9
3	200.0	126.2	65.0	84.3	386.2
4	200.0	135.4	66.1	73.8	399.8
5	200.0	143.4	66.0	64.6	407.4
6	200.0	150.4	64.9	56.6	410.0
7	200.0	156.6	63.0	49.6	408.6
8	200.0	162.0	60.6	43.4	404.4
9	200.0	166.7	57.8	38.0	397.9
10	200.0	170.8	54.8	33.3	390.0
11	200.0	174.4	51.6	29.2	380.8
12	200.0	177.6	48.4	25.6	371.2
13	200.0	180.4	45.2	22.4	361.2
14	200.0	182.8	42.0	19.6	350.8
15	200.0	184.9	38.9	17.2	340.5
16	200.0	186.7	36.0	15.1	330.7
17	200.0	188.3	33.2	13.3	321.1
18	200.0	189.7	30.6	11.7	312.1
19	200.0	190.9	28.1	10.3	303.3
20	200.0	192.0	25.8	9.1	295.2

Capitolo 6

11. a) Per prima cosa calcoliamo il tempo F_i . Non serve prendere in considerazione la velocità del clock, perché possiamo assumere $A_i = 0$ per tutti i pacchetti. F_i diventa quindi, semplicemente, la dimensione cumulativa per flusso, cioè $F_i = F_{i-1} + P_i$.

Pacchetto	Dimensione	Flusso	F_i
1	200	1	200
2	200	1	400
3	160	2	160
4	120	2	280
5	160	2	440
6	210	3	210
7	150	3	360
8	90	3	450

Ora trasmettiamo in ordine di F_i crescente: pacchetto 3, pacchetto 1, pacchetto 6, pacchetto 4, pacchetto 7, pacchetto 2, pacchetto 5, pacchetto 8.

- b) Per dare al flusso 1 il peso 2 dividiamo ognuno dei suoi F_i per 2, cioè $F_i = F_{i-1} + P/2$. Per dare al flusso 2 il peso 4 dividiamo ognuno dei suoi F_i per 4, cioè $F_i = F_{i-1} + P/4$. Per dare al flusso 3 il peso 3 dividiamo ognuno dei suoi F_i per 3, cioè $F_i = F_{i-1} + P/3$. Possiamo nuovamente sfruttare il fatto che non c'è attesa.

Pacchetto	Dimensione	Flusso	F_i pesato
1	200	1	100
2	200	1	200
3	160	2	40
4	120	2	70
5	160	2	110
6	210	3	70
7	150	3	120
8	90	3	150

Trasmettendo secondo l'ordine crescente dei valori pesati di F_i inviamo i pacchetti in questo modo: pacchetto 3, pacchetto 4, pacchetto 6, pacchetto 1, pacchetto 5, pacchetto 7, pacchetto 8, pacchetto 2.

15. a) Per il pacchetto i -esimo in arrivo per un determinato flusso, calcoliamo il relativo tempo di terminazione stimato F_i mediante la formula $F_i = \max\{A_i, F_{i-1}\} + 1$, mentre il clock usato per misurare i tempi di arrivo A_i è più lento di un fattore uguale al numero di code attive. Il clock A_i è globale; la sequenza dei valori di F_i calcolati come detto è locale a ciascun flusso.

La tabella seguente elenca tutti gli eventi in base al tempo reale. I pacchetti sono identificati mediante il proprio flusso e tempo di arrivo: quindi, il pacchetto A4 è quello che arriva dal flusso A all'istante 4, cioè il terzo pacchetto. Le ultime tre colonne sono le code di ciascun flusso nell'intervallo di tempo successivo, compreso il pacchetto attualmente in trasmissione. Il numero di tali code attive determina l'incremento di A_i sulla riga successiva. Si hanno più pacchetti sulla stessa riga se hanno lo stesso valore di F_i ; i valori di F_i sono in corsivo quando $F_i = F_{i-1} + 1$ (invece di essere $F_i = A_i + 1$).

Tempo	A_i	Arrivi	F_i	Inviati	Coda di A	Coda di B	Coda di C
1	1.0	A1, B1, C1	2.0	A1	A1	B1	C1
2	1.333	C2	3.0	B1	B1	C1,	C2
3	1.833	A3	3.0	C1	A3	C1,	C2
4	2.333	B4	3.333	A3	A3	B4	C2, C4
		C4	4.0				
5	2.666	A5	4.0	C2	A5	B4	C2, C4

(continua)

(segue)

Tempo	A_i	Arrivi	F_i	Inviai	Coda di A	Coda di B	Coda di C
6	3.0	A6	5.0	B4	A5, A6	B4	C4, C6
		C6	5.0				
7	3.333	B7	4.333	A5	A5, A6	B7	C4, C6, C7
		C7	6.0				
8	3.666	A8	6.0	C4	A6, A8	B7, B8	C4, C6, C7
		B8	5.333				
9	4	A9	7.0	B7	A6, A8, A9	B7, B8, B9	C6, C7
		B9	6.333				
10	4.333			A6	A6, A8, A9	B8, B9	C6, C7
11	4.666	A11	8.0	C6	A8, A9, A11	B8, B9	C7
12	5	C12	7.0	B8	A8, A9, A11	B8, B9	C7, C12
13	5.333	B13	7.333	A8	A8, A9, A11	B9, B13	C7, C12
14	5.666			C7	A9, A11	B9, B13	C7, C12
15	6.0	B15	8.333	B9	A9, A11	B9, B13, B15	C12
16	6.333			A9	A9, A11	B13, B15	C12
17	6.666			C12	A11	B13, B15	C12
18	7			B13	A11	B13, B15	
19	7.5			A11	A11	B15	
20	8			B15		B15	

b) Con la strategia di code eque e pesate abbiamo, per il flusso B:

$$F_i = \max\{A_i, F_{i-1}\} + 0.5$$

Per i flussi A e C, F_i è uguale a prima. Ecco la tabella corrispondente a quella precedente.

Tempo	A_i	Arrivi	F_i	Inviai	Coda di A	Coda di B	Coda di C
1	1.0	A1, C1	2.0	B1	A1	B1	C1
		B1	1.5				
2	1.333	C2	3.0	A1			C1, C2
3	1.833	A3	3.0	C1	A1		C1, C2
4	2.333	B4	2.833	B4	A3	B4	C2, C4
		C4	4.0				

Tempo	A_i	Arrivi	F_i	Inviai	Coda di A	Coda di B	Coda di C
5	2.666	A5	4.0	A3	A3, A5		C2, C4
6	3.166	A6	5.0	C2	A5, A6		C2, C4, C6
		C6	5.0				
7	3.666	B7	4.167	A5	A5, A6	B7	C4, C6, C7
		C7	6.0				
8	4.0	A8	6.0	C4	A6, A8	B7, B8	C6, C7
		B8	4.666				
9	4.333	A9	7.0	B7	A6, A8, A9	B7, B8, B9	C6, C7
		B9	5.166				
10	4.666			B8	A6, A8, A9	B8, B9	C6, C7
11	5.0	A11	8.0	A6	A6, A8, A9, A11	B9	C6, C7
12	5.333	C12	7.0	C6	A8, A9, A11	B9	C6, C7, C12
13	5.666	B13	6.166	B9	A8, A9, A11	B9, B13	C7, C12
14	6.0			A8	A9, A11	B13	C7, C12
15	6.333	B15	6.833	C7	A9, A11	B13, B15	C12
16	6.666			B13	A9, A11	B13, B15	C12
17	7.0			B15	A11	B15	C12
18	7.333			A9	A11		C12
19	7.833			C12	A11		C12
20	8.333			A11	A11		

36. a) Abbiamo

$$\text{TempP} = \frac{\text{AvgLen} - \text{MinThreshold}}{\text{MaxThreshold} - \text{MinThreshold}}$$

AvgLen è il valore medio di MinThreshold e MaxThreshold, per cui questa frazione vale $1/2$ e, quindi, $\text{TempP} = \text{MaxP}/2 = p/2$. Ora, abbiamo $P_{\text{count}} = \text{TempP}/(1 - \text{count} \times \text{TempP}) = 1/(x - \text{count})$, dove $x = 2/p$. Quindi, $1 - P_{\text{count}} = \frac{x - (\text{count} + 1)}{x - \text{count}}$. Valutando il prodotto $(1 - P_1) \times \dots \times (1 - P_n)$, si ottiene $\frac{x-2}{x-1} \cdot \frac{x-3}{x-2} \cdots \frac{x-(n+1)}{x-n} = \frac{x-(n+1)}{x-1}$, dove $x = 2/p$.

b) Dal risultato del punto precedente, $\alpha = \frac{x-(n+1)}{x-1}$. Quindi $x = \frac{(n+1)-\alpha}{1-\alpha}$. Di conseguenza, $p = \frac{2(1-\alpha)-\alpha}{(n+1)-\alpha}$.

48. In ogni secondo il volume del bucket non deve essere negativo. Per una assegnata profondità del bucket, D , e velocità di token, r , possiamo calcolare il volume del bucket $v(t)$ al tempo t (misurato in secondi) e imporre che $v(t)$ non sia negativo.

$$\begin{aligned}
 v(0) &= D - 5 + r = D - (5 - r) \geq 0 \\
 v(1) &= D - 5 - 5 + 2r = D - 2(5 - r) \geq 0 \\
 v(2) &= D - 5 - 5 - 1 + 3r = D - (11 - 3r) \geq 0 \\
 v(3) &= D - 5 - 5 - 1 + 4r = D - (11 - 4r) \geq 0 \\
 v(4) &= D - 5 - 5 - 1 - 6 + 5r = D - (17 - 5r) \geq 0 \\
 v(5) &= D - 5 - 5 - 1 - 6 - 1 + 6r = D - 6(3 - r) \geq 0
 \end{aligned}$$

Definiamo le funzioni $f_1(r), f_2(r), \dots, f_6(r)$ in questo modo:

$$\begin{aligned}
 f_1(r) &= 5 - r \\
 f_2(r) &= 2(5 - r) = 2f_1(r) \geq f_1(r) \quad (\text{per } 1 \leq r \leq 5) \\
 f_3(r) &= 11 - 3r \leq f_2(r) \quad (\text{per } r \geq 1) \\
 f_4(r) &= 11 - 4r < f_3(r) \quad (\text{per } r \geq 1) \\
 f_5(r) &= 17 - 5r \\
 f_6(r) &= 6(3 - r) \leq f_5(r) \quad (\text{per } r \geq 1)
 \end{aligned}$$

Prima di tutto, per $r \geq 5$, $f_i(r) \leq 0$ per qualsiasi valore di i . Ciò significa che se la velocità del token è maggiore di 5 pacchetti al secondo, qualsiasi profondità positiva del bucket (cioè $D \geq 0$) sarà sufficiente. Per $1 \leq r \leq 5$, dobbiamo considerare solamente $f_2(r)$ e $f_5(r)$, perché le altre funzioni hanno valori minori di queste. Si può facilmente verificare che $f_2(r) - f_5(r) = 3r - 7$. Quindi, alla profondità del bucket, D , occorre impostare i valori dati dalla formula seguente

$$D \geq \begin{cases} f_5(r) = 17 - 5r & (r=1,2) \\ f_2(r) = 2(5 - r) & (r=3,4,5) \\ 0 & (r \geq 6) \end{cases}$$

Capitolo 7

2. 4 M A R Y 4377 7 J A N U A R Y 7 2002 2 90000 150000 1
8.

INT	4	15
INT	4	29496729
INT	4	58993458

10. 15 be 00000000 00000000 00000000 00001111
15 le 00001111 00000000 00000000 00000000

29496729 be 00000001 11000010 00010101 10011001
29496729 le 10011001 00010101 11000010 00000001

58993458 be 00000011 10000100 00101011 00110010
58993458 le 00110010 00101011 10000100 00000011

Capitolo 8

4. Per prima cosa notiamo che la S box che è stata fornita implica che per ottenere $R_{i+1} = F(R_i, K_{i+1}) \oplus L_i$ dobbiamo semplicemente eseguire l'operazione XOR fra i 4 bit centrali di ogni spezzone di 6 bit di K_{i+1} con il corrispondente frammento di 4 bit di R_i . I singoli bit di K_{i+1} (5af310 7a3fff in esadecimale) sono:

0101 1010 1111 0011 0001 0000 0111 1010 0011 1111 1111 1111

oppure, separando i quattro bit centrali di ogni frammento di 6 bit e mostrando, sotto, i soli frammenti di 4 bit:

0 1011 01 0111 10 0110 00 1000 00 1111 01 0001 11 1111 11 1111 1
1011 0111 0110 1000 1111 0001 1111 1111

$L_i = R_{i-1}$ è, in esadecimale, deadbeef. In bit, è rappresentato in questo modo:

1101 1110 1010 1101 1011 1110 1110 1111

Dall'esercizio precedente, sappiamo che R_i vale:

1001 0011 0001 0110 1011 1110 1110 1111

Ora possiamo eseguire l'operazione XOR sulle tre parti, L_i , R_i e K_{i+1} . Il risultato finale, R_{i+1} , è il seguente:

1111 1010 1101 0011 1111 0001 1111 1111

o, in esadecimale, fad3 f1ff.

7. a) Dobbiamo trovare d tale che $5d \equiv 1 \pmod{(p-1)(q-1)}$. Ciò significa che $5d + 72k = 1$ per un qualche valore di k . Usando l'algoritmo di divisione mutua di Euclide, troviamo che $d = 29 + 72r$, dove r è un numero intero arbitrario. Consideriamo il minimo valore positivo, $d = 29$.
- b) $pq = 91$, per cui $m^5 \pmod{pq} = 7^5 \pmod{91} = 63$.
- c) $c^d \pmod{pq} = 2^{29} \pmod{91} = 32$.

Glossario

4B/5B: un tipo di schema per la codifica di bit usato nella tecnologia FDDI, nel quale gruppi di 4 bit di dati vengono trasmessi mediante una sequenza di 5 bit.

802.3: standard IEEE per rete Ethernet.

802.5: standard IEEE per rete token ring.

802.11: standard IEEE per rete wireless.

822: si riferisce al documento RFC 822, che definisce il formato per i messaggi di posta elettronica in Internet. Vedere *SMTP*.

AAL: ATM Adaptation Layer. Uno strato di protocollo che si configura al di sopra di ATM. Per le comunicazioni di dati sono definiti due AAL: AAL3/4 e AAL5. Ciascuno strato di protocollo fornisce al mittente un meccanismo per segmentare grandi pacchetti in celle e al ricevente un meccanismo analogo per ricostruire i pacchetti a partire dalle celle ricevute.

ABR: (1) Available bit rate. Uno schema di controllo della congestione basato sulla velocità, sviluppato per reti ATM; ABR si prefigge di consentire ad una sorgente di aumentare o diminuire la velocità assegnata, sulla base di feedback ricevuti dagli switch interni alla rete. Confrontate *CBR*, *UBR* e *VBR*. (2) Area border router. Router che si trova al confine di un'area in relazione ad un protocollo a stato delle linee.

ACK: abbreviazione di *acknowledgement* ("conferma"). Un acknowledgement viene inviato da chi riceve dati per segnalare al mittente che la trasmissione dei dati stessi ha avuto successo.

ACK duplicato: la ritrasmissione di una conferma (ACK) nel protocollo TCP. L'ACK duplicato non conferma alcun nuovo dato. La ricezione di più ACK duplicati innesca il meccanismo di *ritrasmissione veloce (fast retransmit)* di TCP.

AF: Assured forwarding ("inoltro garantito"). Uno dei comportamenti relativi ai salti che è stato proposto per Differentiated Services.

ALF: Application Level Framing. Un principio per il progetto di protocolli che afferma che i programmi applicativi comprendono le proprie necessità di comunicazione meglio di quanto facciano i protocolli di trasporto di utilizzo generale.

ampiezza di banda (bandwidth): una misura della capacità di una linea di collegamento o di una connessione, solitamente espressa in bit al secondo.

AMPS: Advanced Mobile Phone System. Sistema di telefonia cellulare analogico, attualmente sta per essere sostituito da un sistema digitale, noto come PCS.

ANSI: American National Standards Institute. Ente di standardizzazione privato degli Stati Uniti d'America che solitamente partecipa al processo di standardizzazione di ISO. Ha la responsabilità di SONET.

API: Application programming interface. Interfaccia usata dai programmi applicativi per accedere al sottosistema di rete (solitamente il protocollo di trasporto), è specifica di ciascun sistema operativo. Un esempio assai diffuso è la Socket API del sistema operativo Berkeley Unix.

area: nel contesto dell'instradamento a stato delle linee, un insieme di router adiacenti che condividono totalmente l'uno con l'altro le informazioni di instradamento. Un dominio di instradamento è suddiviso in aree per migliorare la scalabilità.

ARP: Address Resolution Protocol. Protocollo dell'architettura Internet, usato per tradurre indirizzi di un protocollo di alto livello in indirizzi fisici relativi all'hardware. Usato comunemente in Internet per mettere in corrispondenza indirizzi IP e indirizzi Ethernet.

ARPANET: una rete sperimentale a commutazione di pacchetto fondata da ARPA verso la fine degli anni Sessanta, che divenne il backbone della rete Internet quando questa si è poi sviluppata.

ARQ: Automatic repeat request (*richiesta di ripetizione automatica*). Generica strategia per inviare pacchetti in modo affidabile su una linea di collegamento non affidabile. Se il mittente non riceve un ACK per un determinato pacchetto entro un tempo prestabilito, ipotizza che il pacchetto non sia arrivato (o che sia stato consegnato con bit errati) e lo ritrasmette. I protocolli stop-and-wait e sliding window sono due esempi di protocolli che usano la strategia ARQ. Confrontate FEC.

ASN.1: Abstract Syntax Notation One. Insieme a BER, uno standard per il formato di presentazione identificato da ISO come parte dell'architettura OSI.

ATM: Asynchronous transfer mode. Una tecnologia di rete orientata alla connessione che trasporta dati usando piccoli pacchetti di dimensione fissa (detti *celle*).

ATMARP: Address Resolution Protocol, migliorato per essere utilizzato in reti ATM.

ATM Forum: uno dei principali enti di standardizzazione per la tecnologia ATM.

aumento additivo/decremento moltiplicativo: strategia a finestra usata da TCP per il controllo della congestione. TCP apre la finestra di congestione con una velocità lineare, ma la dimezza quando si verificano perdite di pacchetti per effetto di congestione. È stato dimostrato che l'aumento additivo con decremento moltiplicativo è una condizione necessaria perché un meccanismo di controllo della congestione sia stabile.

autenticazione: protocollo di sicurezza mediante il quale due parti che non si fidano reciprocamente danno prova di essere ciò che dichiarano di essere.

AS: Autonomous system. Un gruppo di reti e router che sono soggetti ad un'autorità comune e che usano lo stesso protocollo di instradamento intradominio.

backoff esponenziale: una strategia di ritrasmissione che raddoppia il valore del temporizzatore di scadenza ogni volta che il pacchetto viene ritrasmesso.

bandwidth: vedere *ampiezza di banda*.

Bellman-Ford: algoritmo di instradamento a vettore di distanza, che prende il nome dai suoi inventori.

BER: Basic Encoding Rules. Regole per la codifica dei tipi di dati definite da ASN.1.

best-effort (delivery): il modello di servizio dell'attuale architettura di Internet. La consegna di un messaggio viene tentata, ma senza alcuna garanzia di successo.

BGP: Border Gateway Protocol. Un protocollo di instradamento interdominio mediante il quale sistemi autonomi di scambiano informazioni di raggiungibilità. La versione più recente è BGP-4.

BISYNC: Binary Synchronous Communication. Un protocollo dello strato di linea di collegamento orientato ai byte, sviluppato intorno alla fine degli anni Sessanta da IBM.

bit stuffing: una tecnica usata per distinguere, a livello di bit, i dati dalle sequenze di controllo. Usata dal protocollo HDLC.

BLAST: un protocollo che esegue la frammentazione e la ricostruzione di grandi messaggi, usato per costruire un protocollo RPC.

blocco (block): termine relativo al sistema operativo usato per descrivere una situazione in cui un processo sospende la propria esecuzione in attesa di un evento, quale la modifica dello stato di un *semaforo*.

bridge: un dispositivo che inoltra trame dello strato di linea di collegamento da una rete fisica ad un'altra; a volte viene anche detto *switch per LAN*. Confrontare *ripetitore* e *router*.

broadcast: un metodo per la consegna di un pacchetto ad ogni host di una particolare rete o internetwork. Può essere implementato in hardware (es: Ethernet) o in software (es: broadcast IP).

BUS: Broadcast and Unknown Server. Un dispositivo utilizzato nell'emulazione di LAN (LANE).

CA: Certification Authority (o Certificate Authority). Un'entità che firma certificati relativi alla sicurezza, garantendo in tal modo che la chiave pubblica contenuta nel certificato appartenga effettivamente all'entità menzionata nel certificato stesso.

cambio di contesto (context switch): un'operazione con la quale un sistema operativo sospende l'esecuzione di un processo ed inizia l'esecuzione di un altro processo. Un cambio di contesto richiede la memorizzazione dello stato (ad esempio, il contenuto di tutti i registri) relativo al primo processo e il caricamento dello stato del secondo processo.

canale: un generico termine delle telecomunicazioni, usato in questo libro per indicare una connessione logica fra processi.

canali logici concorrenti (concurrent logical channels): più canali logici di tipo stop-and-wait inseriti mediante multiplexing in una singola linea di collegamento punto-punto. La consegna in ordine non viene garantita. Questo meccanismo venne utilizzato dal protocollo IMP-IMP di ARPANET.

carico controllato (controlled load): una delle classi di servizio disponibili nell'architettura Integrated Services di Internet.

CBR: Constant bit rate. Una classe di servizio di ATM che garantisce la trasmissione di dati ad una velocità di bit costante, emulando in tal modo una linea di trasmissione dedicata. Confrontate *ABR*, *UBR* e *VBR*.

CCITT: l'ormai defunto Comité Consultif International de Telegraphique et Telephonique, una unità di International Telecommunications Union (ITU) delle Nazioni Unite. Ora sostituito da ITU-T.

CDN: Content distribution network. Un insieme di server Web supplenti, distribuiti nella rete Internet, che rispondono alle richieste HTTP al posto di un server Web. L'obiettivo di distribuire il più possibile i server supplenti è quello di avere un supplente vicino al client, rendendo possibile fornire più velocemente la risposta ad una richiesta.

cella: un pacchetto di 53 byte della tecnologia ATM, che può trasportare fino a 48 byte di dati.

certificato: un documento firmato digitalmente da un'entità e contenente il nome e la chiave pubblica di un'altra entità. Usato per distribuire chiavi pubbliche. Vedere anche *CA*.

CHAN: un protocollo che implementa canali di tipo richiesta/risposta.

checksum (somma di controllo): solitamente è la somma in complemento a uno di alcuni o di tutti i byte di un pacchetto, calcolata e aggiunta alla fine del pacchetto dal mittente. Il ricevitore calcola nuovamente la somma e la confronta con quella contenuta nel messaggio. Le somme di controllo vengono utilizzate per rilevare errori in un pacchetto e possono anche essere utilizzate per verificare che il pacchetto sia stato consegnato all'host corretto. Il termine *checksum* viene a volte usato (impropriamente) in riferimento, genericamente, a codici a rivelazione d'errore.

chiave di demultiplexing: un campo presente nell'intestazione di un pacchetto che consente la realizzazione del *demultiplexing* (per esempio: il campo *ProtNum* di IP).

chiave pubblica: Vedere *cifratura a chiave pubblica*.

chipping code: sequenza casuale di bit che viene sottoposta all'operazione XOR insieme ai dati del flusso, per realizzare la tecnica a sequenza diretta di spettro diffuso.

CIDR: Classless interdomain routing. Un metodo per l'aggregazione di percorsi di instradamento che tratta un blocco di indirizzi IP di classe C contigui come se fosse una singola rete.

cifratura (encryption): l'applicazione ai dati di una funzione di trasformazione, con l'obiettivo che soltanto il destinatario dei dati potrà essere in grado di leggerli (dopo aver applicato la funzione inversa, di *decifrazione*). In generale, la cifratura dipende da un "segreto" condiviso dal mittente e dal destinatario, oppure da una coppia di chiavi pubblica/privata.

cifratura a chiave pubblica (public key encryption): algoritmi di cifratura (per esempio: RSA) nei quali ciascun partecipante ha una chiave privata (che non condivide con nessun altro) e una chiave pubblica (nota a tutti). Per inviare un messaggio sicuro ad un utente si cifrano i dati con la chiave pubblica di quell'utente: per decifrare il messaggio è

necessaria la corrispondente chiave privata, per cui solamente il destinatario lo può leggere.

circuit switching: vedere *commutazione di circuito*.

circuito virtuale (virtual circuit): l'astrazione fornita da reti orientate alla connessione, come ATM. Solitamente i partecipanti, prima di poter inviare dati, si devono scambiare messaggi per instaurare un circuito virtuale (e probabilmente riservare risorse al circuito stesso). Confrontate con *datagramma*.

client: chi richiede un servizio in un sistema distribuito.

CLNP: Connectionless Network Protocol. La controparte ISO del protocollo IP di Internet.

clock recovery: vedere *ricostruzione del clock*.

clock virtuale (virtual clock): un modello di servizio che consente alla sorgente di prenotare risorse nei router usando una descrizione delle proprie necessità basata sulla velocità. Il clock virtuale supera il servizio di consegna di tipo *best-effort* dell'attuale Internet.

commutatore (switch): un nodo di rete che inoltra pacchetti dai propri ingressi alle proprie uscite in base alle informazioni presenti nell'intestazione di ciascun pacchetto. Differisce da un *router* principalmente per il fatto che non mette in comunicazione reti di tipi diversi.

commutatore ottico (optical switch): un dispositivo di commutazione che inoltra lunghezze d'onda ottiche da una porta d'ingresso ad una porta d'uscita senza convertirle in segnale elettrico.

commutatore per LAN (LAN switch): sinonimo di *bridge*, solitamente utilizzato per indicare un bridge con molte porte. Viene anche chiamato *Ethernet switch* se la tecnologia di collegamento a cui fornisce supporto è Ethernet.

commutazione di circuito (circuit switching): una generica strategia per la commutazione di dati all'interno di una rete, che richiede l'instaurazione di un percorso dedicato (circuito) fra la sorgente e la destinazione. Confrontate con *commutazione di pacchetto*.

commutazione di pacchetto (packet switching): una generica strategia per la commutazione di dati attraverso una rete. La commutazione di pacchetto usa la commutazione di tipo *store-and-forward* ("memorizza e inoltra") di unità di dati discrete denominate pacchetti e implementa il *multiplexing statistico*.

congestione: stato risultante da troppi pacchetti che si contendono risorse limitate (es: ampiezza di banda di una linea e spazio nei buffer di router o di switch), che può costringere un router (o uno switch) ad eliminare pacchetti.

connectionless: vedere *protocollo privo di connessione*.

connessione: in generale, un canale che deve essere predisposto prima di poter essere utilizzato (ad esempio, mediante la trasmissione di opportune informazioni di predisposizione). Ad esempio, TCP mette a disposizione un'astrazione di connessione per la consegna affidabile e in ordine di un flusso di byte. Spesso si dice che le reti orientate alla connessione, come ATM, forniscono un'astrazione di *circuito virtuale*.

context switch: vedere *cambio di contesto*.

controlled load: vedere *carico controllato*.

controllo di congestione: qualsiasi strategia di gestione delle risorse della rete che abbia l'obiettivo di migliorare o evitare le situazioni di congestione. Un meccanismo di controllo della congestione può essere implementato nei router (o switch) all'interno della rete, dagli host che si trovano ai limiti della rete stessa, o da una combinazione di entrambe le modalità.

controllo di flusso: un meccanismo mediante il quale chi riceve dati può regolare la velocità di trasmissione di chi li invia, in modo che i dati non arrivino troppo velocemente per poter essere elaborati. Confrontate con il *controllo di congestione*.

CRC: Cyclic redundancy check. Un codice a rivelazione d'errore calcolato sui byte che compongono un pacchetto e, quindi, allegato al pacchetto stesso dall'hardware di rete (es: adattatore Ethernet). Il codice CRC ha proprietà di rivelazione d'errore migliori del semplice *checksum*.

crossbar switch: un semplice schema di progetto per commutatori nel quale ogni ingresso è connesso direttamente ad ogni uscita e ciascuna porta di uscita ha il compito di risolvere le contese.

CSMA/CD: Carrier Sense Multiple Access with Collision Detect. CSMA/CD è una funzionalità dell'hardware di rete. "Carrier sense multiple access" significa che più stazioni sono in grado di monitorare la linea di collegamento e di rilevare il suo stato, inattivo o utilizzato; "collision detect" indica che se due o più stazioni stanno trasmettendo simultaneamente sulla linea, si accorgono della collisione dei propri segnali. La più nota tecnologia che usa CSMA/CD è Ethernet.

cut-through: una modalità di commutazione o di inoltro mediante la quale un pacchetto inizia ad esser trasmesso verso un'uscita prima di essere stato ricevuto completamente dal nodo di commutazione, riducendo così la latenza attraverso il nodo stesso.

datagram: l'unità fondamentale per la trasmissione nell'architettura Internet. Un datagramma contiene tutte le informazioni necessarie alla propria consegna fino a destinazione, in analogia con una lettera nel sistema postale. Le reti a datagramma sono prive di connessione.

DCE: Distributed Computing Environment. Un insieme di protocolli e di standard, basati su RPC, che forniscono supporto all'elaborazione distribuita. Definito da OSF.

DDCMP: Digital Data Communication Message Protocol. Un protocollo dello strato di linea di collegamento orientato ai byte, usato nella rete DECNET di Digital Equipment Corporation.

DDoS: Distributed denial of service. Un attacco di tipo DoS che ha origine da un insieme di nodi. Ogni nodo attaccante è in grado di apportare soltanto un carico marginale alla macchina obiettivo, ma il carico complessivo di tutti i nodi attaccanti può portare l'obiettivo al collasso.

DECbit: uno schema di controllo della congestione nel quale i router segnalano ai nodi terminali il sopraggiungere di congestione mediante il valore di un bit nell'intestazione dei pacchetti instradati. Quando una determinata percentuale dei pacchetti ricevuti ha tale bit impostato al valore 1, i nodi terminali diminuiscono le proprie velocità di trasmissione.

decifrazione (decrypting): l'azione inversa del processo di *cifratura (encryption)*, necessario a ricostruire i dati presenti in un messaggio cifrato.

demultiplexing: l'uso di informazioni contenute nell'intestazione di un pacchetto per dirigerlo verso l'alto in una pila di protocolli. Ad esempio, IP usa il campo *ProtNum* dell'intestazione IP per decidere a quale protocollo di livello superiore (es: TCP, UDP) appartiene il pacchetto, mentre TCP usa il numero di porta per smistare un pacchetto TCP al corretto processo applicativo. Confrontate con *multiplexing*.

demultiplexing key: vedere *chiave di demultiplexing*.

dense mode multicast: modalità PIM utilizzata quando la maggior parte dei router o degli host ha bisogno di ricevere pacchetti multicast.

DES: Data Encryption Standard. Un algoritmo per la cifratura dei dati basato su una chiave segreta a 64 bit.

DHCP: Dynamic Host Configuration Protocol. Un protocollo usato dagli host, nel momento in cui iniziano il proprio funzionamento, per apprendere varie informazioni relative alla rete, come il proprio indirizzo IP.

DHT: Distributed hash table. Una tecnica mediante la quale un messaggio viene instradato verso un calcolatore che ospita un particolare oggetto, in base al nome dell'oggetto stesso. L'oggetto viene trasformato, mediante hashing, in un identificativo univoco, in modo che ogni nodo intermedio lungo il percorso inoltri il messaggio ad un altro nodo che sia in grado di interpretare un prefisso sempre più lungo di tale identificativo. Le tabelle di hash distribuite vengono usate nelle reti peer-to-peer.

Differentiated Services: una nuova architettura per fornire, in Internet, un servizio migliore di quello di tipo best-effort, proposta in alternativa all'architettura Integrated Services.

direct sequence: una tecnica di tipo *spread spectrum* che richiede di eseguire l'operazione XOR tra il flusso dei dati e una sequenza casuale di bit nota come *chipping code*.

distance vector (vettore di distanza): un algoritmo usato nell'instradamento e basato sul percorso di costo minimo. Ogni nodo pubblicizza verso i propri vicini informazioni di raggiungibilità e i relativi costi e usa gli analoghi aggiornamenti che riceve per costruire la propria tabella di inoltro. Il protocollo di instradamento RIP usa un algoritmo a vettore di distanza. Confrontate con *link state*.

distribuzione delle chiavi: meccanismo mediante il quale gli utenti vengono in possesso delle chiavi pubbliche degli altri utenti, attraverso lo scambio di certificati con firma digitale.

DMA: Direct memory access. Un approccio per connettere host a dispositivi di I/O, mediante il quale il dispositivo legge e scrive direttamente i dati dalla memoria dell'host. Vedere anche *P/I*.

DNA/DECNET: Digital Network Architecture. Un'architettura basata sul modello OSI che implementa un modello di rete privo di connessione e un protocollo di trasporto orientato alla connessione.

DNS: Domain name system. Il sistema distribuito per i nomi di Internet, usato per tradurre i nomi di host (come *cicada.cs.princeton.edu*) in indirizzi IP (come 192.12.69.35).

Il DNS è realizzato mediante una gerarchia di server per i nomi (*name server*).

dominio: può essere riferito ad un contesto nello spazio dei nomi gerarchico del DNS (ad esempio, il dominio "edu") oppure ad una regione di Internet che viene considerata come se fosse un'unica entità nell'instradamento gerarchico. Nel secondo caso, il *dominio* è equivalente ad un *sistema autonomo*.

DoS: Denial of service. Una situazione in cui un nodo attaccante inonda un nodo obiettivo con un carico (un numero di pacchetti) così elevato che, a tutti gli effetti, impedisce agli utenti legittimi di accedere al nodo; di conseguenza, a questi ultimi viene imposto il servizio.

DS3: un servizio di linea di collegamento a 44.7 Mbps offerto dalle compagnie telefoniche. Chiamato anche T3.

DSL: Digital subscriber line. Una famiglia di standard per la trasmissione di dati con velocità di milioni di bit al secondo lungo linee telefoniche in doppino (*twisted pair*).

DVMRP: Distance Vector Multicast Routing Protocol. Protocollo di instradamento multicast usato dalla maggior parte dei router della rete MBone.

DWDM: Dense wavelength division multiplexing. Realizzazione del multiplexing di più lunghezze d'onda luminose ("colori") su una singola fibra ottica. La tecnica è "densa" nel senso che è in grado di utilizzare un numero molto elevato di lunghezze d'onda ottiche.

ECN: Explicit congestion notification. Una tecnica mediante la quale i router informano gli host terminali di una situazione di congestione impostando uno speciale valore nei pacchetti che inoltrano. Viene usata insieme ad algoritmi di gestione attiva della coda, come RED.

EF: Expedited forwarding. Uno dei comportamenti relativi ai singoli hop che sono stati proposti per Differentiated Services.

EGP: Exterior Gateway Protocol. Uno dei primi protocolli di instradamento interdominio di Internet, che veniva usato dai gateway (router) posti al confine fra sistemi autonomi per scambiare informazioni con altri AS. Sostituito da BGP.

encryption: vedere *cifratura*.

Ethernet: una tecnologia per reti locali molto diffusa, che usa CSMA/CD e ha un'ampiezza di banda di 10 Mbps. Una rete Ethernet, per se stessa, è solamente un cavo di collegamento passivo; tutti gli aspetti relativi alla trasmissione in una rete Ethernet sono completamente realizzati dagli adattatori presenti negli host.

extended LAN: vedere *LAN estesa*.

fabric: vedere *matrice*.

fast retransmit: vedere *ritrasmissione veloce*.

FDDI: Fiber Distributed Data Interface. Una tecnologia di rete di tipo token ring ad alta velocità, progettata per fibre ottiche.

FEC: Forward error correction. Una strategia generale per gestire gli errori di bit introdotti

in pacchetti di dati senza dover ritrasmettere il pacchetto. In ogni pacchetto vengono inserite informazioni ridondanti che possono essere utilizzate dal ricevitore per determinare quali bit del pacchetto siano errati. Confrontate con *ARQ*.

Fiber Channel: un protocollo bidirezionale per lo strato di linea di collegamento che viene usato per connettere calcolatori (solitamente supercomputer) alle periferiche. Fiber Channel ha un'ampiezza di banda di 100 MBps e si può estendere per 30 m. Viene usato in modo analogo a HiPPI.

finestra scorrevole: vedere *sliding window*.

firewall: un router che è stato configurato per filtrare (cioè per non inoltrare) pacchetti provenienti da determinate sorgenti. Usato per mettere in atto una politica di sicurezza.

flow control: vedere *controllo di flusso*.

flowspec: specifica dell'ampiezza di banda e dei requisiti di ritardo di un flusso, che viene presentata alla rete per effettuare una prenotazione. Usata con RSVP.

forwarding: vedere *inoltro*.

forwarding table: vedere *tavella di inoltro*.

FQ (fair queueing): un algoritmo di accodamento basato sulla rotazione (*round-robin*) che impedisce ad un processo che non si comporta correttamente di impegnare una porzione arbitrariamente grande della capacità della rete.

frame (trama): un altro nome per il pacchetto, usato tipicamente con riferimento a pacchetti inviati lungo una singola linea di collegamento piuttosto che un'intera rete. Un problema importante consiste nel modo in cui il ricevitore rileva l'inizio e la fine di un frame, problema noto con il nome di *framing*.

Frame Relay: un servizio pubblico a commutazione di pacchetto e orientato alla connessione offerto dalle compagnie telefoniche.

frammentazione/ricostruzione (fragmentation/reassembly): un metodo per la trasmissione di messaggi di dimensioni maggiori del valore di MTU della rete. I messaggi vengono suddivisi ("frammeati") dal mittente in pezzi più piccoli e ricostruiti, poi, dal ricevitore.

frequency hopping (salto di frequenza): una tecnica di tipo *spread spectrum* che prevede la trasmissione dei dati utilizzando una sequenza di frequenze casuale.

FTP: File Transfer Protocol. Il protocollo standard dell'architettura Internet per il trasferimento di file fra host. Costruito al di sopra di TCP.

GMPLS: Generalized MPLS. Consente al protocollo IP di funzionare in modo nativo su reti ottiche commutate.

gopher: un servizio informatico di Internet.

GSM: Global System for Mobile communication. Sistema di telefonia cellulare installato in tutto il mondo (con l'esclusione degli Stati Uniti e del Canada). Simile a PCS, che sta per essere installato negli Stati Uniti e nel Canada.

H.323: protocollo di controllo della sessione usato spesso per la telefonia in reti Internet.

handle: nella programmazione, un identificativo o puntatore che viene usato per accedere a un oggetto.

HDLC: High-level Data Link Control protocol. Un protocollo dello standard ISO per il livello di linea di collegamento. Usa la tecnica del *bit stuffing* per risolvere il problema del *framing*.

HIPPI: High Performance Parallel Interface. Una tecnologia di rete standardizzata da ANSI con velocità di trasmissione di Gbps, usata tipicamente per connettere supercomputer a dispositivi periferici. Usata in modo simile a *Fiber Channel*.

host: un calcolatore connesso ad una o più reti, che interagisce con utenti ed esegue programmi applicativi.

HTML: HyperText Markup Language. Un linguaggio utilizzato per predisporre pagine per il World Wide Web.

HTTP: HyperText Transport Protocol. Un protocollo del livello applicativo basato sul paradigma richiesta/risposta e usato nel World Wide Web. HTTP usa connessioni TCP per trasferire dati.

IAB: Internet Activities Board. L'ente principale che sovrintende allo sviluppo e alla standardizzazione di protocolli nell'architettura Internet. IRTF e IETF sono gruppi di lavoro (*task force*) di IAB.

IBGP: Interior BGP. Il protocollo usato per lo scambio di informazioni di instradamento interdominio fra router dello stesso dominio.

ICMP: Internet Control Message Protocol. Questo protocollo è parte integrante di IP. Consente ad un router o ad un host destinatario di comunicare con la sorgente, solitamente per segnalare un errore nell'elaborazione di datagrammi IP.

IEEE: Institute for Electrical and Electronics Engineers. Una società professionale per ingegneri che definisce anche standard per le reti, fra i quali la serie 802 di standard per le LAN.

IETF: Internet Engineering Task Force. Un gruppo di lavoro di IAB che ha il compito di individuare soluzioni ingegneristiche di breve termine per Internet.

IMAP: Internet Message Access Protocol. Un protocollo dello strato applicativo che permette ad un utente di recuperare la propria posta da un server.

IMP-IMP: un protocollo dello strato di linea di collegamento, orientato ai byte, usato nella rete ARPANET originaria.

indirizzo hardware: l'indirizzo dello strato di linea di collegamento (*link-level address*) usato per identificare l'adattatore che collega l'host alla rete locale.

inoltro (forwarding): l'operazione eseguita da un router su ogni pacchetto: lo riceve da un ingresso, decide verso quale uscita inviarlo e lo invia verso tale uscita.

instradamento (routing): il processo mediante il quale i nodi si scambiano informazioni topologiche per costruire tabelle di inoltro corrette. Vedere anche *inoltro, stato delle linee e vettore di distanza*.

instradamento dalla sorgente (source routing): decisioni di instradamento prese dalla sorgente della trasmissione prima che il pacchetto venga inviato. Il percorso è composto dall'elenco dei nodi che il pacchetto deve attraversare per arrivare a destinazione.

instradamento gerarchico (hierarchical routing): uno schema di instradamento multilivello che usa la struttura gerarchica dello spazio di indirizzamento come base per prendere le decisioni di inoltro. Ad esempio, i pacchetti potrebbero essere inizialmente instradati verso la rete di destinazione e, in seguito, verso uno specifico host di quella rete.

instradamento interdominio (interdomain routing): il processo di scambio di informazioni di instradamento fra diversi domini di instradamento. BGP è un esempio di protocollo interdominio.

instradamento intradominio (intradomain routing): il processo di scambio di informazioni di instradamento all'interno di uno stesso dominio di instradamento o sistema autonomo. RIP e OSPF sono esempi di protocolli intradominio.

Integrated Services: termine solitamente utilizzato per indicare una rete a commutazione di pacchetto che possa fornire un efficace supporto sia ai tradizionali dati per calcolatori sia a dati audio e video in tempo reale. Dà anche il nome ad un modello di servizio che è stato proposto per Internet, progettato per rimpiazzare l'attuale modello di servizio di tipo *best-effort*.

integrità: nel contesto della sicurezza delle reti, un servizio che garantisce che il messaggio ricevuto sia identico al messaggio inviato.

internet: un insieme di reti a commutazione di pacchetto (eventualmente eterogenee) interconnesse mediante router, detto anche *internetwork*.

Internet: la internet mondiale, basata sull'architettura TCP/IP, che collega milioni di host in tutto il mondo.

interoperabilità: la capacità di hardware eterogeneo e di software di produttori diversi di comunicare mediante un corretto scambio di messaggi.

interruzione (interrupt): un evento (tipicamente generato da un dispositivo hardware) che dice al sistema operativo di terminare la sua attuale attività e di intraprendere un'azione specifica. Ad esempio, si una un interrupt per segnalare al sistema operativo che è appena arrivato un pacchetto dalla rete.

IP: Internet Protocol (noto anche con la sigla IPv4). Un protocollo che fornisce un servizio best-effort e privo di connessione per la consegna di datagrammi attraverso Internet.

IPng: Internet Protocol – Next Generation (noto anche con la sigla IPv6). Nuova versione proposta per IP che fornisce uno spazio di indirizzamento più grande e maggiormente gerarchico, oltre ad altre nuove caratteristiche.

IPSEC: IP Security. Un'architettura per l'autenticazione, la privacy e l'integrità dei messaggi; uno dei servizi di sicurezza alternativi nell'architettura Internet.

IRTF: Internet Research Task Force. Un gruppo di lavoro di IAB, responsabile per le decisioni in merito alla ricerca e sviluppo di Internet.

ISDN: Integrated Services Digital Network. Un servizio di comunicazione digitale offerto

dalle compagnie telefoniche e standardizzato da ITU-T. ISDN combina in un unico mezzo fisico i servizi di connessione vocale e di trasmissione di dati digitali.

IS-IS: un protocollo di instradamento a stato delle linee, simile a OSPF.

ISO: International Standards Organization. L'ente internazionale che ha delineato l'architettura OSI a sette livelli e un insieme di protocolli che, però, non ha raggiunto il successo commerciale.

ITU-T: un sottocomitato di ITU (International Telecommunications Union), un ente mondiale che definisce standard in tutte le aree delle comunicazioni internazionali analogiche e digitali. ITU-T si occupa di standard per le telecomunicazioni, in particolare con ATM.

jitter: variazione della latenza di una rete. Un jitter di valore elevato ha un impatto negativo sulla qualità delle applicazioni video e audio.

JPEG: Joint Photographic Experts Group. Sigla usata tipicamente per fare riferimento ad un algoritmo ampiamente utilizzato nella compressione di immagini statiche, che è stato sviluppato dal gruppo JPEG.

Kerberos: un sistema di autenticazione basato su TCP/IP e sviluppato al MIT, nel quale due host usano un terzo host (*trusted third party*), di cui si fidano entrambi, per autentificarsi reciprocamente.

LAN: Local area network. Una rete basata su una tecnologia di rete qualsiasi e che è progettata per estendersi su distanze che vanno fino a poche migliaia di metri (es: Ethernet e FDDI). Confrontate con SAN, MAN e WAN.

LANE: Local area network emulation. Aggiunta di nuove funzionalità ad una rete ATM per fare in modo che si comporti come una LAN a mezzo fisico condiviso (cioè in modo simile ad una Ethernet).

LAN estesa (extended LAN): un insieme di reti LAN connesse tramite bridge.

LAN switch: vedere *commutatore per LAN*.

latenza (latency): una misura di quanto tempo sia necessario ad un singolo bit per propagarsi da un capo ad un altro di una linea o di un canale. La latenza viene misurata come un tempo.

LDAP: Lightweight Directory Access Protocol. Un sottoinsieme del servizio di elenchi X.500 che ha assunto recente popolarità per gestire informazioni relative agli utenti.

LER: Label edge router. Un router ai confini di una "nuvola" MPLS. Esegue una ricerca completa per l'indirizzo IP dei pacchetti in arrivo, poi assegna loro etichette in base al risultato di tale ricerca.

LES: LAN emulation server. Server per una *LANE*.

linea di collegamento (link): una connessione fisica fra due nodi di una rete. Può essere realizzata da un filo di rame o un cavo in fibra ottica, oppure può essere un collegamento wireless (ad esempio, via satellite).

link-level protocol: vedere *protocollo dello strato della linea di collegamento*.

link state: vedere *stato delle linee*.

LSR: Label switching router. Un router che esegue i protocolli di controllo di IP, ma usa l'algoritmo di inoltro a commutazione di etichette di MPLS.

MAC: Media access control. Algoritmi usati per controllare l'accesso in reti a mezzo fisico condiviso, come Ethernet e FDDI.

MACA: Multiple Access with Collision Avoidance. Algoritmo distribuito utilizzato per mediare l'accesso ad un mezzo fisico condiviso.

MACAW: Multiple Access with Collision Avoidance for Wireless. Miglioramento del più generale algoritmo MACA per fornire un miglior supporto alle reti *wireless*. Usato dallo standard 802.11.

MAN: Metropolitan area network. Una rete basata su una delle nuove tecnologie di rete che operano ad alta velocità (fino a parecchi Gbps) e su distanze abbastanza elevate da coprire un'area metropolitana. Confrontate con SAN, LAN e WAN.

Manchester: uno schema di codifica di bit che trasmette l'OR esclusivo del segnale di *clock* e dei dati codificati mediante NRZ. Usato da Ethernet.

matrice (fabric o switching fabric): quella parte di un commutatore che realizza realmente la commutazione; cioè sposta i pacchetti da un ingresso ad un'uscita. Confrontate con *porta*.

MBone: Multicast Backbone. Una rete logica sovrapposta ad Internet, nella quale router avanzati, in grado di gestire la trasmissione multicast, usano il *tunneling* per inoltrare datagrammi multicast attraverso Internet.

MD5: Message Digest versione 5. Un efficiente algoritmo di crittografia a somma di controllo, comunemente utilizzato per verificare che i contenuti di un messaggio non siano stati alterati.

MIB: Management information base. Definisce l'insieme di variabili relative alla rete che possono essere lette o scritte in un nodo di rete. Lo standard MIB viene usato congiuntamente al protocollo SNMP.

MIME: Multipurpose Internet Mail Extensions. Specifiche per la conversione di dati binari (come file conenenti immagini) in testo ASCII, per consentire la loro trasmissione a mezzo posta elettronica.

modalità promiscua (promiscuous mode): una modalità di funzionamento di un adattatore di rete nella quale esso riceve tutti i frame trasmessi sulla rete, e non soltanto quelli ad esso indirizzati.

Mosaic: un famoso browser grafico gratuito per il World Wide Web sviluppato al National Center for Supercomputing Applications (NCSA) della University of Illinois.

MP3: MPEG Layer 3. Standard per la compressione audio usato in MPEG.

MPEG: Moving Picture Experts Group. Termine tipicamente utilizzato in riferimento ad un algoritmo per la compressione di flussi video sviluppato dal gruppo MPEG.

MPLS: Multiprotocol Label Switching. Un insieme di tecniche usate per migliorare le capacità dei router IP, allegando etichette ai pacchetti.

MSAU: Multistation access unit. Un dispositivo utilizzato nelle reti di tipo *token ring* per connettere più stazioni all'anello e poterle eliminare in caso di guasto.

MTU: Maximum transmission unit. La dimensione del più grande pacchetto che può essere inviato attraverso una rete fisica.

multicast: una speciale forma di trasmissione *broadcast* con la quale i pacchetti vengono consegnati ad uno specifico sottoinsieme di host della rete.

multicast in modalità sparsa (sparse mode multicast): una modalità usata da PIM quando in un certo gruppo multicast sono presenti relativamente pochi host o router che devono ricevere dati.

multiplexing: combinazione di canali distinti all'interno di un unico canale di livello inferiore. Ad esempio, canali TCP e UDP diversi vengono sottoposti a multiplexing lungo un singolo canale IP tra due host. L'operazione inversa, *demultiplexing*, avviene nell'host destinatario.

multiplexing statistico: multiplexing di più sorgenti di dati su un canale o una linea di collegamento condivisi, basato sulle richieste di ciascuna sorgente.

name resolution: vedere *traduzione dei nomi*.

NAT: Network address translation. Una tecnica per l'estensione dello spazio di indirizzamento di IP che prevede la traduzione fra indirizzi IP di utilizzo globale e indirizzi a validità locale, effettuata ai confini di una rete o di un sito.

NDR: Network Data Representation. Lo standard per la codifica dei dati utilizzato nel DCE (Distributed Computing Environment), definito da Open Software Foundation. NDR usa una strategia di tipo "fa tutto il ricevitore" e inserisce all'inizio di ciascun messaggio un marcatore che identifica l'architettura.

Netscape: un diffuso browser per il World Wide Web.

network-level protocol: vedere *protocollo dello strato di rete*.

NFS: Network File System. Un famoso *file system* distribuito sviluppato da Sun Microsystems e basato su SunRPC, un protocollo RPC sviluppato sempre da Sun.

NIST: National Institute for Standards and Technology. L'ente di standardizzazione ufficiale degli Stati Uniti.

nodo: termine generico utilizzato per i singoli calcolatori che costituiscono una rete. Fra i nodi troviamo calcolatori di utilizzo generale, ma anche commutatori (*switch*) e router.

NRZ: Non-return to zero. Uno schema di codifica di bit che codifica un valore 1 come segnale alto e un valore 0 come segnale basso.

NRZI: Non-return to zero inverted. Uno schema di codifica di bit che effettua una transizione a partire dal valore attuale del segnale per codificare un valore 1 e mantiene il segnale costante per codificare un valore 0.

NSF: National Science Foundation. Un'agenzia del governo degli Stati Uniti che finanzia la ricerca scientifica statunitense, compresa la ricerca sulle reti e sull'infrastruttura di Internet.

NV: Network Video. Un'applicazione di videoconferenza che viene eseguita sulla rete MBone.

OC: Optical Carrier. Il prefisso usato per indicare le varie velocità delle trasmissioni ottiche con SONET. Ad esempio, OC-1 si riferisce allo standard SONET per la trasmissione in fibra ottica a 51.84 Mbps. Un segnale OC-n differisce da un segnale STS-n soltanto per il fatto che il segnale OC-n è adattato per la trasmissione ottica.

ONC: Open Network Computing. Una versione di SunRPC in via di standardizzazione per Internet.

optical switch: vedere *commutatore ottico*.

OSF: Open Software Foundation. Un consorzio di costruttori di computer che ha definito standard per l'elaborazione distribuita, fra i quali il formato di presentazione NDR.

OSI: Open Systems Interconnection. Il modello di riferimento per reti a sette strati sviluppato da ISO. Contiene linee guida seguite dagli standard dei protocolli ISO e ITU-T.

OSPF: Open Shortest Path First. Un protocollo di instradamento sviluppato da IETF per l'architettura Internet. OSPF si basa su un algoritmo *a stato delle linee*, nel quale ogni nodo si costruisce una topologia della rete e la usa per prendere decisioni di inoltro. Noto oggi come Open Group.

overlay network: vedere *rete sovrapposta*.

pacchetto (packet): un'unità di dati inviata attraverso una rete a commutazione di pacchetto. Vedere anche *frame* e *segmento*.

packet switching: vedere *commutazione di pacchetto*.

pari: vedere *peer*.

partecipante (participant): termine generico usato per indicare i processi, protocolli o host che si scambiano messaggi.

partenza lenta (slow start): un algoritmo per evitare la congestione nel protocollo TCP tentando di regolare la velocità dei segmenti in uscita. Per ogni ACK che viene ricevuto, vengono inviati due ulteriori pacchetti, dando luogo ad un aumento esponenziale del numero di pacchetti inviati e ancora in attesa di conferma.

PAWS: Protection against wrapped sequence numbers. La progettazione di protocolli di trasporto con un grande spazio a disposizione per i numeri di sequenza, per fornire protezione contro il fenomeno dei numeri di sequenza che tornano al proprio valore iniziale in una rete in cui i pacchetti subiscono forti ritardi.

PCS: Personal Communication Services. Nuovo sistema di telefonia cellulare digitale in via di installazione negli Stati Uniti e nel Canada. Simile al sistema GSM installato nel resto del mondo.

PDU: Protocol Data Unit. Sinonimo per *pacchetto* o *frame*.

peer (pari): la controparte su un altro calcolatore con cui il modulo di un protocollo collabora per realizzare un servizio di comunicazione.

peer-to-peer network: vedere *rete peer-to-peer*.

PEM: Privacy Enhanced Mail. Estensioni alla posta elettronica di Internet che forniscono supporto alla protezione della privacy e dell'integrità dei messaggi. Vedere anche *PGP*.

PGP: Pretty Good Privacy. Un insieme di software di pubblico dominio che fornisce le funzionalità di privacy e di autenticazione usando RSA e che usa una griglia di fiducia per la distribuzione delle chiavi pubbliche.

PHB: Per-hop behavior. Comportamento di singoli router nell'architettura Differentiated Services. Due tra i PHB proposti sono *AH* e *EF*.

physical-level protocol: vedere *protocollo dello strato fisico*.

piconet: rete wireless che si estende su una breve distanza (es: 10 m), usata per connettere calcolatori in ufficio (laptop, stampanti, PDA, workstation, ecc.) senza usare cavi.

PIM: Protocol Independent Multicast. Un protocollo di instradamento multicast che si può basare su protocolli di instradamento unicast diversi.

ping: un programma di utilità di Unix usato per verificare il valore di RTT verso vari host di Internet, inviando un messaggio ECHO_REQUEST di ICMP, a cui l'host remoto replica con un messaggio ECHO_RESPONSE.

PIO: Programmed Input/Output. Un approccio per la connessione di host a dispositivi di I/O, nel quale la CPU legge i dati e scrive i dati nel dispositivo di I/O. Vedere anche *DMA*.

poison reverse: usata insieme a *split horizon*, è una tecnica euristica per evitare i cicli nell'instradamento mediante protocolli a vettore di distanza.

porta (port): termine generico solitamente utilizzato per identificare il punto in cui un utente di rete si collega alla rete stessa. In un commutatore, una porta identifica il punto di ingresso o di uscita in cui i pacchetti vengono, rispettivamente, ricevuti o inviati.

POTS: Plain old telephone service. Sigla usata per indicare il servizio telefonico esistente, in contrapposizione a ISDN, ATM o altre tecnologie che vengono offerte dalle compagnie telefoniche ora o che potranno essere offerte in futuro.

PPP: Point-to-Point Protocol. Protocollo della linea di collegamento tipicamente utilizzato per connettere calcolatori mediante una linea telefonica commutata (*dial-up line*).

problema del nodo esposto: situazione che accade in una rete wireless quando due nodi ricevono segnali da una sorgente comune, ma entrambi sono in grado di raggiungere nodi che non ricevono tale segnale.

problema del nodo nascosto: situazione che accade in una rete wireless quando due nodi trasmettono verso una comune destinazione, ma non sono consapevoli di farlo in contemporanea.

processo (process): un'astrazione fornita dal sistema operativo per consentire l'esecuzione contemporanea di operazioni diverse. Ad esempio, ciascuna applicazione dell'utente viene eseguita all'interno di un proprio processo, mentre le varie funzioni del sistema operativo avvengono all'interno di altri processi.

prodotto ritardo × ampiezza di banda: il prodotto del valore di RTT e dell'ampiezza di banda di una rete. Fornisce una misura di quanti dati possano essere in transito attraverso una rete.

promisuous mode: vedere *modalità promiscua*.

protocollo (protocol): la specifica di un'interfaccia fra moduli eseguiti su calcolatori diversi, insieme al servizio di comunicazione implementato da tali moduli. Il termine viene anche utilizzato per fare riferimento ad un'implementazione del modulo che soddisfi le specifiche. Per distinguere fra questi due diversi utilizzi, l'interfaccia viene spesso chiamata *specificità del protocollo*.

protocollo dello strato della linea di collegamento (link-level protocol o link-layer protocol): un protocollo che ha il compito di portare a destinazione *frame* attraverso una rete a connessione diretta (es: Ethernet, token ring o linea punto-punto).

protocollo dello strato di rete (network-level protocol): un protocollo che viene eseguito in una rete commutata, subito al di sopra dello strato della linea di collegamento.

protocollo dello strato fisico (physical-level protocol): lo strato più basso nella pila dei protocolli OSI. La sua funzione principale è quella di codificare i bit nei segnali che vengono propagati attraverso il mezzo fisico trasmissivo.

protocollo di trasporto (transport protocol): un protocollo end-to-end che consente la comunicazione tra processi di host diversi. Un esempio canonico è il protocollo TCP.

protocollo privo di connessione (connectionless): un protocollo nel quale i dati vengono inviati senza alcuna predisposizione preventiva. IP è un esempio di protocollo di questo tipo.

proxy: un agente interposto fra un client e un server, che intercetta i messaggi e fornisce un qualche tipo di servizio. Ad esempio, un proxy può "prendere il posto" di un server, rispondendo alle richieste del client senza contattare il server, usando dati memorizzati in una propria cache.

pseudointestazione (pseudoheader): un sottoinsieme di campi dell'intestazione IP che vengono trasferiti ai protocolli di trasporto TCP e UDP per essere utilizzati nel calcolo della loro somma di controllo. La pseudointestazione contiene gli indirizzi IP del mittente e del destinatario e la lunghezza del datagramma IP, consentendo così la rilevazione di errori di trasmissione in questi campi o la consegna di un pacchetto ad un indirizzo errato.

public key encryption: vedere *cifratura a chiave pubblica*.

punto di rendezvous: un router usato da PIM per consentire ai ricevitori di avere informazioni sui mittenti.

QoS: Quality of Service. Garanzia di consegna dei pacchetti fornita da un'architettura di rete. Solitamente ci si riferisce a garanzie di prestazioni, come l'ampiezza di banda e il ritardo. Internet offre un servizio di consegna di tipo *best-effort*, nel senso che viene messo in atto ogni sforzo possibile per consegnare un pacchetto, ma la consegna non è garantita.

RED: Random early detection. Una disciplina di gestione della coda per i router che, quando si preannuncia una congestione, elimina casualmente alcuni pacchetti per avvertire i mittenti della necessità di un rallentamento nella velocità di trasmissione.

rendezvous: vedere *punto di rendezvous*.

repeater: vedere *ripetitore*.

rete peer-to-peer: una generica categoria di applicazioni che integra la logica applicativa (ad esempio, la memorizzazione di file) con l'instradamento. Fra gli esempi più famosi troviamo Napster e Gnutella. Prototipi usati per la ricerca utilizzano spesso tabelle hash distribuite (*DHT*).

rete sovrapposta (overlay): una rete virtuale (logica) sovrapposta ad una rete fisica esistente. I nodi della rete sovrapposta comunicano fra loro usando tunnel anziché linee fisiche di collegamento. Le reti sovrapposte vengono spesso utilizzate per installare nuovi servizi di rete, dal momento che non richiedono la cooperazione dell'infrastruttura di rete esistente.

RFC: Request for Comments. Documenti di Internet che contengono, fra le altre cose, specifiche di protocolli come TCP e IP.

ricostruzione del clock (clock recovery): il processo di ricostruzione di un segnale di clock valido a partire da un segnale digitale trasmesso serialmente.

RIO: RED with In and Out. Una strategia di eliminazione di pacchetti basata su RED, che utilizza, però, due curve di eliminazione: una per i pacchetti che sono stati contrassegnati come appartenenti al profilo "in" e una per quelli appartenenti al profilo "out". Progettata per essere usata nell'implementazione di Differentiated Services.

RIP: Routing Information Protocol. Un protocollo di instradamento intradominio presente nel sistema operativo Berkeley Unix. Ogni router che esegue RIP costruisce dinamicamente la propria tabella di inoltro utilizzando un algoritmo a *vettore di distanza*.

ripetitore (repeater): un dispositivo che propaga segnali elettrici da un cavo Ethernet ad un altro. Fra due qualsiasi host di una Ethernet ci possono essere al massimo due ripetitori. I ripetitori inoltrano segnali, mentre i *bridge* inoltrano *frame*, e i *router* e gli *switch* inoltrano *pacchetti*.

ritrasmissione veloce (fast retransmit): una strategia usata da TCP che tenta di evitare la scadenza dei temporizzatori in presenza di perdita di pacchetti. TCP ritrasmette un segmento dopo aver ricevuto tre ACK duplicati consecutivi, confermando i dati fino a quel segmento (escluso).

RPB: Reverse-path broadcast. Una tecnica usata per eliminare pacchetti broadcast duplicati.

router ("intradattore"): un nodo di rete, connesso a due o più reti, che inoltra pacchetti da una rete ad un'altra. Confrontate con *bridge*, *repeater* e *switch*.

routing: vedere *instradamento*.

routing table: vedere *tavella di inoltro*.

RPC: Remote Procedure Call. Protocollo di trasporto sincrono di tipo richiesta/risposta, usato in molte interazioni di tipo client/server.

RSA: un algoritmo di cifratura a chiave pubblica che prende il nome dai suoi inventori: Rivest, Shamir e Adleman.

RSVP: Resource Reservation Protocol. Un protocollo per la prenotazione di risorse in una rete. RSVP usa il concetto di *stato soft* nei router e assegna ai destinatari, piuttosto che ai mittenti, la responsabilità di effettuare le prenotazioni.

RTCP: Real-time Transport Control Protocol. Protocollo di controllo associato a RTP.

RTP: Real-time Transport Protocol. Un protocollo *end-to-end* usato da applicazioni multimediali che abbiano vincoli di tempo reale.

RTT: Round-trip time. Il tempo richiesto alla propagazione di un bit di informazione da un capo all'altro di una linea di collegamento o di un canale, e ritorno. In altre parole, il doppio della latenza del canale.

salto di frequenza: vedere *frequency hopping*.

SAN: System area network. Una rete che spazia sui componenti di un calcolatore (ad esempio, il disco, il monitor, una videocamera). Altre volte si usa la stessa sigla per indicare una *storage area network*, e allora comprende interfacce come HiPPI e Fiber Channel. Confrontate con *LAN*, *MAN* e *WAN*.

schema: una specifica di come strutturare e interpretare un insieme di dati, definita per documenti XML.

scrambling: il processo di applicazione della funzione XOR ad un segnale e ad un flusso pseudocasuale di bit prima della trasmissione, per fare in modo che vi sia un numero sufficiente di transizioni di segnale da consentire la ricostruzione del segnale di *clock*. La tecnica di *scrambling* viene usata in SONET.

SDP: Session Description Protocol. Un protocollo dello strato applicativo usato per avere informazioni in merito ai canali audio/video disponibili. Segnala il nome e lo scopo di ciascuna sessione, il suo tempo di inizio e di termine, i tipi di *media* (es: audio, video) che compongono la sessione ed informazioni dettagliate su come ricevere tale sessione (es: l'indirizzo multicast, il protocollo di trasporto e i numeri di porta da utilizzare)

segmento: un pacchetto TCP. Un segmento contiene una porzione del flusso di byte che viene inviato tramite TCP.

segnalazione (signalling): al livello fisico, indica la trasmissione di un segnale su un qualche mezzo fisico. Nella tecnologia ATM, la segnalazione si riferisce al processo di instaurazione di un circuito virtuale.

SELECT: un protocollo di demultiplexing asincrono usato per costruire un protocollo RPC.

semaforo: una variabile utilizzata per fornire il supporto alla sincronizzazione fra processi. Tipicamente, un processo *si blocca* di fronte ad un semaforo, in attesa che qualche altro processo mandi un segnale per sbloccare il semaforo.

server: il fornitore di un servizio in un sistema distribuito di tipo client/server.

signalling: vedere *segnalazione*.

sindrome della finestra inefficiente (silly window syndrome): una condizione che può accadere nel protocollo TCP se, ogni volta che il ricevitore apre di una piccola quantità la propria finestra di ricezione, il mittente invia un piccolo segmento che riempie la finestra. Il risultato è la trasmissione di molti piccoli segmenti e un utilizzo inefficiente della banda.

SIP: Session Initiation Protocol. Un protocollo dello strato applicativo usato nelle applicazioni multimediali. Determina il dispositivo corretto con il quale comunicare per raggiungere un certo utente, determina se l'utente vuole o è in grado di partecipare ad una particolare comunicazione, determina la scelta dei vari *media* e degli schemi di codifica da utilizzare, e stabilisce i parametri della sessione (per esempio i numeri di porta).

sistema autonomo: vedere *AS*.

sliding window ("finestra scorrevole"): un algoritmo che consente al mittente di trasmettere più pacchetti (fino alla dimensione della finestra) prima di ricevere una conferma. Nel momento in cui vengono ricevute conferme per quei pacchetti, presenti nella finestra, che erano stati spediti per primi, la finestra "scorre" e si possono inviare ulteriori pacchetti. L'algoritmo a finestra scorrevole combina l'affidabilità della consegna con un throughput elevato. Vedere *ARQ*.

slow start: vedere *partenza lenta*.

SMDS: Switched Multimegabit Data Service. Un servizio di supporto alla connessione fra LAN e WAN, offerto da alcune compagnie telefoniche.

SMTP: Simple Mail Transfer Protocol. Il protocollo di Internet per la posta elettronica. Vedere 822.

SNA: System Network Architecture. L'architettura di rete proprietaria di IBM.

SNMP: Simple Network Management Protocol. Un protocollo di Internet per il controllo e la gestione di host, reti e router.

socket: l'estensione di Unix che fornisce una API (*application programming interface*) per l'architettura TCP/IP.

soft state: informazioni relative ad una connessione che sono contenute in un router e che vi rimangono memorizzate per un periodo di tempo limitato, invece di esservi memorizzate permanentemente mediante una procedura di instaurazione della connessione (che richiede una esplicita procedura di chiusura della connessione stessa).

somma di controllo: vedere *checksum*.

SONET: Synchronous Optical Network. Uno standard di *framing* basato sul clock per la trasmissione digitale in fibre ottiche. Definisce come le compagnie telefoniche trasmettono i dati attraverso le reti ottiche.

sottorete (*subnet*): si usa quando un singolo indirizzo di rete IP viene assegnato a più reti fisiche. I router interni ad una sottorete usano una maschera di sottorete (*subnet mask*) per capire a quale rete fisica vada inoltrato un pacchetto. L'uso di sottoreti introduce, a tutti gli effetti, un terzo livello nella gerarchia a due livelli degli indirizzi IP.

source routing: vedere *instradamento dalla sorgente*.

sparse mode multicast: vedere *multicast in modalità sparsa*.

split horizon: un metodo per spezzare i cicli di instradamento in un algoritmo di instradamento a vettore di distanza. Quando un nodo invia un aggiornamento di instradamento ai propri vicini, non invia ad un vicino quei percorsi che ha appreso dal vicino stesso. La tecnica di *split horizon* viene utilizzata insieme a quella di *poison reverse*.

spread spectrum ("spettro diffuso"): tecnica di codifica che richiede la dispersione di un segnale in un intervallo di frequenze più ampio del necessario, per minimizzare gli effetti negativi delle interferenze.

SSL: Secure Socket Layer. Uno strato di protocollo al di sopra di TCP che fornisce l'autenticazione e la cifratura delle connessioni. Noto anche come Transport Layer Security (TLS).

stato delle linee (*link state*): un algoritmo per il percorso di costo minimo usato nell'instradamento. Tutti i router vengono inondati con informazioni relative ai vicini connessi direttamente e agli attuali costi di tali linee di connessione; ogni router usa queste informazioni per costruire una propria visione della rete, sulla quale basare le decisioni di inoltro. Il protocollo di instradamento OSPF usa un algoritmo a stato delle linee. Confrontate con *vettore di distanza*.

stop-and-wait: un algoritmo di trasmissione affidabile nel quale il mittente trasmette un pacchetto e attende una conferma prima di inviare il pacchetto successivo. Confrontate con *sliding window* e *canali logici concorrenti*. Vedere anche *ARQ*.

STS: Synchronous Transport Signal. Il prefisso usato per varie velocità di trasmissione di SONET. Ad esempio, STS-1 si riferisce allo standard SONET di una trasmissione a 51.84 Mbps.

SunRPC: protocollo per l'invocazione di procedure remote sviluppato da Sun Microsystems. SunRPC è utilizzato per realizzare NFS. Vedere anche *ONC*.

switch: vedere *commutatore*.

switching fabric: vedere *matrice*.

T1: un servizio di trasporto telefonico standard equivalente a 24 circuiti ISDN, con velocità di 1.544 Mbps. Chiamato anche DS1.

T3: un servizio di trasporto telefonico standard equivalente a 24 circuiti T1, con velocità di 44.736 Mbps. Chiamato anche DS3.

tabella di inoltro (*tabella di inoltro*): la tabella gestita da un router che gli consente di prendere decisioni su come inoltrare pacchetti. Il processo di costruzione della tabella di inoltro viene detto *instradamento (routing)*, per cui la tabella di inoltro viene a volte chiamata anche *tabella di instradamento* o *tabella di routing*. In alcune implementazioni, però, la tabella di inoltro e la tabella di instradamento sono due strutture separate.

tabella di instradamento (*routing table*): vedere *tabella di inoltro*.

TCP: Transmission Control Protocol. Protocollo di trasporto orientato alla connessione dell'architettura Internet. TCP fornisce un servizio di consegna affidabile a flusso di byte.

Telnet: protocollo per terminali remoti dell'architettura Internet. Telnet vi consente di interagire con un sistema remoto come se il vostro terminale fosse connesso direttamente a quel calcolatore.

throughput: la velocità effettiva a cui i dati vengono inviati attraverso un canale. Il termine è spesso usato come sinonimo di *ampiezza di banda (bandwidth)*.

TLS: Transport Layer Security. Servizio di sicurezza che si può disporre al di sopra di un protocollo di trasporto come TCP. Viene spesso utilizzato da HTTP per eseguire transazioni sicure sul World Wide Web. Derivato da *SSL*.

token bucket: un modo per caratterizzare l'ampiezza di banda utilizzata da un flusso. In astratto, i processi accumulano token al trascorrere del tempo, devono spendere un token per trasmettere un byte di dati e devono smettere di trasmettere quando non hanno più token. Di conseguenza, si limita l'ampiezza di banda complessiva, se si è disposti ad accettare qualche "raffica" di traffico.

token ring: una tecnologia di rete fisica che collega gli host in un anello. Un *token* (sequenza di bit) circola nell'anello e un nodo deve avere il possesso del token per poter trasmettere. Esempi di reti di tipo token ring sono FDDI e 802.5.

TP4: OSI Transport Protocol Class 4. Il più potente protocollo di trasporto di OSI, è l'equivalente OSI di TCP.

traduzione dei nomi (*name resolution*): l'azione di tradurre nomi di host (che sono di facile comprensione per le persone) nei corrispondenti indirizzi (che sono utilizzati dai calcolatori). Vedere *DNS*.

trama: vedere *frame*.

transport protocol: vedere *protocollo di trasporto*.

TTL: Time to live (*tempo di vita*). Solitamente è una misura del numero di hop (o di router) che un datagramma IP può attraversare prima di essere eliminato.

tunneling: encapsulamento di un pacchetto usando un protocollo che opera allo stesso livello in cui si trova il pacchetto stesso. Ad esempio, i pacchetti IP di tipo multicast vengono encapsulati in pacchetti IP di tipo unicast per viaggiare all'interno di un *tunnel* attraverso Internet, per realizzare la rete MBone. I tunnel saranno anche utilizzati durante la transizione da IPv4 a IPv6.

Tymnet: una delle prime reti in cui veniva gestita, all'interno di un insieme di router, l'astrazione di *circuito virtuale*.

UBR: Unspecified bit rate. La categoria di servizio "senza accessori" della tecnologia ATM, che offre la consegna *best-effort* delle celle. Confrontate con *ABR*, *CBR* e *VBR*.

UDP: User Datagram Protocol. Protocollo di trasporto dell'architettura Internet che fornisce ai processi del livello applicativo un servizio privo di connessione per datagrammi.

unicast: invio di un pacchetto ad un singolo host di destinazione. Confrontate con *broadcast* e *multicast*.

URI: Uniform resource identifier. Un URL generalizzato, usato, ad esempio, insieme al protocollo SIP per instaurare sessioni multimediali.

URL: Uniform resource locator. Una stringa di testo usata per identificare la collocazione di risorse in Internet. Un tipico URL assomiglia a <http://www.cisco.com>. In questo URL, http è il protocollo usato per accedere alla risorsa che si trova sull'host www.cisco.com.

vat: strumento per audioconferenze usato in Internet; usa il protocollo RTP.

VBR: Variable bit rate. Una delle classi di servizio di ATM, progettata per applicazioni con

requisiti di ampiezza di banda variabili nel tempo, come quelle per la compressione video. Confrontate con *ABR*, *CBR* e *UBR*.

VCI: Virtual circuit identifier. Un identificativo nell'intestazione di un pacchetto che viene usato per la commutazione a circuito virtuale. Nel caso di ATM, una connessione *end-to-end* viene identificata dalla coppia VPI e VCI.

vettore di distanza: vedere *distance vector*.

vic: strumento per videoconferenze per il sistema operativo Unix; usa il protocollo RTP.

virtual circuit: vedere *circuito virtuale*.

virtual clock: vedere *clock virtuale*.

VPI: Virtual path identifier. Un campo di 8 o 12 bit nell'intestazione ATM. Il valore di VPI può essere utilizzato per nascondere all'interno di un singolo "percorso" virtuale più connessioni virtuali attraverso una rete, diminuendo così la quantità di informazioni di stato che devono essere memorizzate negli switch in merito alla connessione. Vedere anche *VCI*.

VPN: Virtual private network. Una rete logica sovrapposta ad una rete esistente. Ad esempio, un'azienda con filiali in tutto il mondo, piuttosto che noleggiare linee di collegamento per tutti i siti, potrebbe costruire una rete al di sopra di Internet.

WAN: Wide area network. Qualsiasi tecnologia di rete fisica che sia in grado di estendersi per grandi distanze (ad esempio, attraverso una nazione). Confrontate con *SAN*, *LAN* e *MAN*.

well-known port: un numero di porta che, per convenzione, viene dedicato all'utilizzo da parte di un particolare server. Ad esempio, in ogni host un server del servizio DNS riceve messaggi sulla "ben nota" porta 53, con protocollo UDP e TCP.

WFQ: Weighted fair queueing. Una variante di *FQ* in cui ad ogni flusso può essere assegnata una diversa frazione della capacità della rete.

WWW: World Wide Web. Un servizio di informazioni ipermediali sulla rete Internet.

X.25: il protocollo standard per la commutazione di pacchetto di ITU.

X.400: lo standard per la posta elettronica di ITU, la controparte del protocollo SMTP dell'architettura Internet.

X.500: lo standard per i servizi di *directory* di ITU, che definisce un servizio di assegnazione dei nomi basato su attributi.

X.509: uno standard di ITU per i certificati digitali.

XDR: External Data Representation. Lo standard di Sun Microsystems per strutture dati indipendenti dall'architettura. Confrontate con *ASN.1* e *NDR*.

XML: Extensible Markup Language. Definisce una sintassi per descrivere dati che possono essere scambiati fra diverse applicazioni Internet.

zona: una porzione della gerarchia dei nomi di dominio, corrispondente ad un'autorità di amministrazione che ha la responsabilità di tale porzione della gerarchia. Ogni zona deve avere almeno due server per i nomi che rispondano a richieste DNS per la zona stessa.

Bibliografia

- [ABKM01] Andersen, D., H. Balakrishnan, F. Kaashoek e R. Morris. "Resilient overlay networks", *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pag. 131-145, October 2001.
- [Bat68] Batcher, K.E. "Sorting networks and their applications", *Proceedings of the 1968 Spring AFIPS Joint Computer Conference* 32:307-314, 1968.
- [BBC*98] Blake, S., D. Black, M. Carlson, E. Davies, Z. Wang e W. Weiss. "An architecture for differentiated services", *Request for Comments* 2475, December 1998.
- [BCS94] Braden, R., D. Clark e S. Shenker. "Integrated services in the internet architecture: An overview", *Request for Comments* 1633, September 1994.
- [BDMS94] Bowman, C.M., P.B. Danzig, U. Manber e M.F. Schwartz. "Scalable internet resource discovery: Research problems and approaches", *Communications of the ACM* 37(8):98-107, August 1994.
- [Bel00] Bellovin, S. M. "ICMP traceback messages", *Work in progress. Internet draft, draft-bellovin-itrace-00.txt*, March 2000.
- [BF93] Borenstein, N. e N. Freed. "MIME (multipurpose internet mail extensions) part one: Mechanisms for specifying and describing the format of internet message bodies", *Request for Comments* 1521, September 1993.
- [BG92] Bertsekas, D. e R. Gallager. *Data Networks*. Prentice Hall, Englewood Cliffs, NJ, second edition, 1992.
- [BG93] Bjorkman, M. e P. Gunningberg. "Locking effects in multiprocessor implementations of protocols", *Proceedings of the SIGCOMM '93 Symposium*, pag. 74-83, September 1993.
- [Bla87] Blahut, R. E. *Principles and Practice of Information Theory*. Addison-Wesley, Reading, MA, 1987.
- [BLFF96] Berners-Lee, T., R. Fielding e H. Frystyk. "Hypertext transfer protocol – HTTP/1.0", *Request for Comments* 1945, May 1996.
- [BLNS82] Birrell, A., R. Levin, R. Needham e M. Schroeder. "Grapevine: An exercise in distributed computing", *Communications of the ACM* 25:250-273, April 1982.
- [BM95a] Bradner, S. e A. Mankin, editors. "The recommendations for the next generation IP protocol", *Request for Comments* 1752, January 1995.
- [BM95b] Bradner, S. e A. Mankin, editors. *IPng: Internet Protocol Next Generation*. Addison-Wesley, Reading, MA, 1995.
- [BMK88] Boggs, D., J. Mogul e C. Kent. "Measured capacity of an Ethernet". *Proceedings of the SIGCOMM '88 Symposium*, pag. 222-234, August 1998.
- [BN84] Birrell, A. e B. Nelson. "Implementing remote procedure calls", *ACM Transactions on Computer Systems* 2(1):39-59, February 1984.

- [Boo95] Boorsook, P. "How anarchy works", *Wired* 3(10):110-118, October 1995.
- [BP95] Brakmo, L.S. e L.L. Peterson. "TCP Vegas: End-to-end congestion avoidance on a global internet", *IEEE Journal of Selected Areas in Communication (JSAC)* 13(8):1465-1480, October 1995.
- [BPY90] Bowman, M., L.L. Peterson e A. Yeatts. "Univers: An attribute-based name server", *Software – Practice and Experience* 20(4):403-424, April 1990.
- [Bri95] Brisco, T. "DNS support for load balancing", *Request for Comments* 1794, Rutgers University, New Brunswick, NJ, April 1995.
- [BS88] Bic, L. e A.C. Shaw. *The Logical Design of Operating Systems*. Prentice Hall, Englewood Cliffs, NJ, 1988.
- [BS01] Barrett, D. e R. Silverman. *SSH: The Secure Shell*. O'Reilly, Sebastopol, CA, 2001.
- [Buf94] Buford, John F. K. *Multimedia Systems*. ACM Press/Addison-Wesley, Reading, MA, 1994.
- [BZ96] Bennett, T. e H. Zhang. "Hierarchical packet fair queueing algorithms", *Proceedings of the SIGCOMM '96 Symposium*, pag. 143-156, August 1996.
- [CCITT92a] Comité Consultif International de Telegraphique et Telephonique. "Open systems interconnection: Specification of abstract syntax notation one (ASN.1)", CCIT Recommendation X.208, 1992.
- [CCITT92b] Comité Consultif International de Telegraphique et Telephonique. "Open systems interconnection: Specification of basic encoding rules for abstract syntax notation one (ASN.1)", CCIT Recommendation X.209, 1992.
- [CF98] Clark, D. e W. Fang. "Explicit allocation of best-effort packet delivery service", *IEEE/ACM Transactions on Networking* 6(4):362-373, August 1998.
- [CFFD93] Cohen, D., G. Finn, R. Felderman e A. DeSchon. "ATOMIC: A low-cost, very-high-speed, local communications architecture", *Proceedings of the 1993 Conference on Parallel Processing*, August 1993.
- [Cha93] Chapin, A.L. "The billion node Internet", In D.C. Lynch e M.T. Rose, editors, *Internet System Handbook*, chapter 17, pag. 707-716. Addison-Wesley, Reading, MA, 1993.
- [CJRS89] Clark, D.D., V. Jacobson, J. Romkey e H. Salwen. "An analysis of TCP processing overhead", *IEEE Communications* 27(6):23-29, June 1989.
- [CK74] Cerf, V. e R. Kahn. "A protocol for packet network intercommunication", *IEEE Transactions on Communications COM-22(5)*:637-648, May 1974.
- [Cla82] Clark, D.D. "Modularity and efficiency in protocol implementation", *Request for Comments* 817, July 1982.
- [Cla85] Clark, D.D. "The structuring of systems using upcalls", *Proceedings of the 10th ACM Symposium on Operating Systems Principles*, pag. 171-180, December 1985.
- [Cla88] Clark, D. "The design philosophy of the DARPA Internet protocols", *Proceedings of the SIGCOMM '88 Symposium*, pag. 106-114, August 1988.
- [Cla97] Clark, D. "Internet cost allocation and pricing", In L. Knight e J. Bailey, editors. *Internet Economics*, pag. 215-253. MIT Press, Cambridge, MA, 1997.
- [CLNZ89] Chen, S.K., E.D. Lazowska, D. Notkin e J. Zahorjan. "Performance implications of design alternatives for remote procedure call stubs", *Proceedings of the Ninth International Conference on Distributed Computing Systems*, pag. 36-41, June 1989.

- [CMRW93] Case, J., K. McCloghrie, M. Rose e S. Waldbusser. "Structure of management information for version 2 of the Simple Network Management Protocol (SNMPv2)", *Request for Comments* 1442, April 1993.
- [Com00] Comer, D.E. *Internetworking with TCP/IP. Volume I: Principles, Protocols e Architecture*. Prentice Hall, Upper Saddle River, NJ, fourth edition, 2000.
- [CP89] Comer, D.E. e L.L. Peterson. "Understanding naming in distributed systems", *Distributed Computing* 3(2):51-60, May 1989.
- [CPVR97] Cohen, J., N. Phadnis, V. Valloppillil e K.W. Ross. "Cache array routing protocol v1.1", <http://ds1.internic.net/internet-drafts/draftvinod-carp-v1-01.txt>, September 1997.
- [Cro82] Crocker, D. "Standard for the format of ARPA Internet text message", *Request for Comments* 822, August 1982.
- [CRZ00] Chu, Y., S. Rao e H. Zhang. "A case for end system multicast", *Proceedings of the ACM SIGMETRICS '00 Conference*, pag. 1-12, June 2000.
- [CS94] Comer, D.E. e D.L. Stevens. *Internetworking with TCP/IP. Volume III: Client-Server Programming and Applications, AT&T TLI Version*. Prentice Hall, Englewood Cliffs, NJ, 1994.
- [CS97] Comer, D.E. e D.L. Stevens. *Internetworking with TCP/IP. Volume III: Client-Server Programming and Applications, Windows Sockets Version*. Prentice Hall, Englewood Cliffs, NJ, 1997.
- [CS00] Comer, D.E. e D.L. Stevens. *Internetworking with TCP/IP. Volume III: Client-Server Programming and Applications, Linux/Posix Sockets Version*. Prentice Hall, Upper Saddle River, NJ, 2000.
- [CSZ92] Clark, D., S. Shenker e L. Zhang. "Supporting real-time applications in an integrated services packet network: Architecture and mechanism", *Proceedings of the SIGCOMM '92 Symposium*, pag. 14-26, August 1992.
- [CT90] Clark, D. e D. Tennenhouse. "Architectural considerations for a new generation of protocols", *Proceedings of the SIGCOMM '90 Symposium*, pag. 200-208, September 1990.
- [CV95] Chandranmenon, G.P. e G. Varghese. "Trading packet headers for packet processing", *Proceedings of the SIGCOMM '95 Symposium*, pag. 162-173, October 1995.
- [CZ85] Cheriton, D.R. e W. Zwaenepoel. "Distributed process groups in the V kernel", *ACM Transactions on Computer Systems* 3(2):77-107, May 1985.
- [Dan98] Danzig, P. "NetCache architecture and deployment", *Third International WWW Caching Workshop*, June 1998.
- [DAPP93] Druschel, P., M. Abbot, M. Pagels e L.L. Peterson. "Network subsystem design", *IEEE Network (Special Issue on End-System Support for High Speed Networks)* 7(4):8-17, July 1993.
- [DBCP97] Degermark, M., A. Brodnik, S. Carlsson e S. Pink. "Small forwarding tables for fast routing lookups", *Proceedings of the SIGCOMM '97 Symposium*, pag. 3-14, October 1997.
- [DC90] Deering, S. e D. Cheriton. "Multicast routing in datagram internetworks and extended LANs", *ACM Transactions on Computer Systems* 8(2):85-110, May 1990.
- [DCB+02] Davie, B., A. Charny, J.C.R. Bennett, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu e D. Stiliadis. "An expedited forwarding phb (per-hop behavior)", *Request for Comments* 3246, March 2002.

- [DEF*96] Deering, S., D. Estrin, D. Farinacci, V. Jacobson, C. Liu e L. Wei. "The PIM architecture for wide-area multicast routing", *ACM/IEEE Transactions on Networking* 4(2):153-162, April 1996.
- [DH98] Deering, S. e R. Hinden. "Internet Protocol, version 6 (IPv6) specification", *Request for Comments* 2460, December 1998.
- [DKS89a] Demers, A., S. Keshav e S. Shenker. "Analysis and simulation of a fair queueing algorithm", *Proceedings of the SIGCOMM '89 Symposium*, pag. 1-12, September 1989.
- [DKS89b] Demers, A., S. Keshav e S. Shenker. "Analysis and simulation of a fair queueing algorithm", *Proceedings of the SIGCOMM '89 Symposium*, pag. 1-12, September 1989.
- [DP93] Druschel, P. e L.L. Peterson. Fbufs: "A high-bandwidth cross-domain transfer facility", *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pag. 189-202, December 1993.
- [DPD94] Druschel, P., L. L. Peterson e B. S. Davie. "Experience with a high-speed network adaptor: A software perspective", *Proceedings of the SIGCOMM '94 Symposium*, pag. 2-13, August 1994.
- [DR00] Davie, B. e Y. Rekhter. *MPLS: Technology and Applications*. Morgan Kaufmann Publishers, San Francisco, CA, 2000.
- [DY75] Drysdale, R. L. e F. H. Young. "Improved divide/sort/merge sorting networks", *SIAM Journal on Computing* 4(3):264-270, September 1975.
- [EFH*98] Estrin, D., D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma e L. Wei. "Protocol independent multicast-sparse mode (pim-sm): Protocol specification", *Request for Comments* 2362, April 1998.
- [EWL*94] Edwards, A., G. Watson, J. Lumley, D. Banks, C. Calamvokis e C. Dalton. User-space protocols deliver high performance to applications on a low-cost Gb/s LAN. *Proceedings of the SIGCOMM '94 Symposium*, pag. 14-23, August 1994.
- [FB96] Freed, N. e N. Borenstein. "Multipurpose Internet mail extensions (MIME) part one: Format of Internet message bodies", *Request for Comments* 2045, November 1996.
- [FGMBL97] Fielding, R., J. Gettys, J. Mogul e T. Berners-Lee. "HyperText Transfer Protocol - HTTP/1.1", *Request for Comments* 2068, January 1997.
- [FHPW00] Floyd, S., M. Handley, J. Padhye e J. Widmer. "Equation-based congestion control for unicast applications", *Proceedings of the SIGCOMM '00 Symposium*, pag. 43-56, Stockholm, Sweden, 2000.
- [Fin88] Finkel, R.A. *An Operating Systems Vade Mecum*. Prentice Hall, Englewood Cliffs, NJ, 1988.
- [FJ93] Floyd, S. e V. Jacobson. "Random early detection gateways for congestion avoidance", *IEEE/ACM Transactions on Networking* 1(4):397-413, August 1993.
- [FKSS99] Feng, W.-C., D. Kandlur, D. Saha e K. Shin. "A self-configuring RED gateway", *IEEE INFOCOM*, New York, pag. 1320-1328, March 1999.
- [FLYV93] Fuller, V., T. Li, J. Yu e K. Varadhan. "Classless interdomain routing (CIDR): An address assignment and aggregation strategy", *Request for Comments* 1519, September 1993.
- [Gar00] Garber, L. "Technology news: Denial-of-service attacks rip the Internet", *Computer* 33(4):12-17, April 2000.

- [GG94] Gopal, I. e R. Guerin. "Network transparency: The plaNET approach", *IEEE/ACM Transactions on Networking* 2(3):226-239, June 1994.
- [Gia*91] Giacopelli, J.N., et al. "Sunshine: A high-performance self-routing broadband packet-switched architecture", *IEEE Journal of Selected Areas in Communications (JSAC)* 9(8):1289-1298, October 1991.
- [Gin99] D. Ginsburg, *ATM: Solutions for Enterprise Internetworking*. Addison-Wesley, Reading, MA, second edition, 1999.
- [GK80] Gerla, M. e L. Kleinrock. "Flow control: A comparative survey", *IEEE Transactions on Communications COM-28(4):553-573*, April 1980.
- [GVC96] Goyal, P., H. Vin e H. Chen. "Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks", *Proceedings of the SIGCOMM '96 Symposium*, pag. 157-168, August 1996.
- [Har00] Harrison, A. "Cyber assaults hit Buy.com, eBay, CNN and Amazon", *Computerworld*, February 2000.
- [HC98] Harkins, D. e D. Carrel. "The Internet Key Exchange (IKE)", *Request for Comments* 2409, November 1998.
- [Hed88] Hedrick, C. "Routing information protocol", *Request for Comments* 1058, June 1988.
- [HMPT89] Hutchinson, N., S. Mishra, L. Peterson e V. Thomas. "Tools for implementing network protocols", *Software - Practice and Experience* 19(9):895-916, September 1989.
- [HP91] Hutchinson, N. e L. Peterson. "The x-kernel: An architecture for implementing network protocols", *IEEE Transactions on Software Engineering* 17(1):64-76, January 1991.
- [HP95] Holzmann, G.J. e B. Pehrson. *The Early History of Data Networks*. IEEE Computer Society Press, Los Alamitos, CA, 1995.
- [HP02] Hennessy, J.L. e D.A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, San Francisco, CA, third edition, 2002.
- [Huf52] Huffman, D.A. "A method for the construction of minimal-redundancy codes", *Proceedings of the IRE* 40(9):1098-1101, September 1952.
- [Jac88a] Jacobson, V. "Congestion avoidance and control", *Proceedings of the SIGCOMM '88 Symposium*, pag. 314-329, August 1988.
- [Jac88b] Jacobson, V. "Congestion avoidance and control", *Proceedings of the SIGCOMM '88 Symposium*, pag. 314-329, August 1988.
- [Jaf81] Jaffe, J.M. "Flow control power is nondecentralizable", *IEEE Transactions on Communications COM-29(9):1301-1306*, September 1981.
- [Jai89] Jain, R. "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks", *ACM Computer Communication Review* 19(5):56-71, October 1989.
- [Jai91] Jain, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, New York, 1991.
- [Jai94] Jain, R. *FDDI Handbook: High-Speed Networking Using Fiber and Other Media*. Addison-Wesley, Reading, MA, 1994.
- [JBB92] Jacobson, V., R. Braden e D. Borman. "TCP extensions for high performance", *Request for Comments* 1323, May 1992.
- [KA98a] Kent, S. e R. Atkinson. "IP authentication header", *Request for Comments* 2402, November 1998.

- [KA98b] Kent, S. e R. Atkinson. "IP encapsulating security payload (ESP)", *Request for Comments* 2406, November 1998.
- [KA98c] Kent, S. e R. Atkinson. "Security architecture for the Internet Protocol", *Request for Comments* 2401, November 1998.
- [KC88] Kanakia, H. e D.R. Cheriton. "The VMP network adaptor board (NAB): High-performance network communication for multiprocessors", *Proceedings of the SIGCOMM '88 Symposium*, pag. 175-187, August 1988.
- [Kes91] Keshav, S. "A control-theoretic approach to flow control". *Proceedings of the SIGCOMM '91 Symposium*, pag. 3-15, September 1991.
- [KHR02] Katabi, D., M. Handley e C. Rohrs. "Congestion control for high bandwidth-delay product networks", *Proceedings of the ACM SIGCOMM '02*, pag. 89-102, August 2002.
- [Kle75] Kleinrock, L. *Queueing Systems. Volume I: Theory*. John Wiley & Sons. New York, 1975.
- [Kle79] Kleinrock, L. "Power and deterministic rules of thumb for probabilistic problems in computer communications". *Proceedings of the International Conference on Communications*, pag. 43.1.1-43.1.10, June 1979.
- [KLL'97] Karger, D., E. Lehman, F.T. Leighton, R. Panigrahy, M. Levine e D. Lewin. "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web", *Proceedings of the ACM Symposium on Theory of Computing*, pag. 654-663, 1997.
- [KM87] Kent, C. e J. Mogul. "Fragmentation considered harmful", *Proceedings of the SIGCOMM '87 Symposium*, pag. 390-401, August 1987.
- [KP91] Karn, P. e C. Partridge. "Improving round-trip time estimates in reliable transport protocols", *ACM Transactions on Computer Systems* 9(4):364-373, November 1991.
- [KPS02] Kaufman, C., R. Perlman e M. Speciner. *Network Security: Private Communication in a PublicWorld*. Prentice Hall, Englewood Cliffs, NJ, 2002.
- [LAAJ00] Labovitz, C., A. Ahuja, A. Abose e F. Jahanian. "Delayed internet routing convergence", *Proceedings of the SIGCOMM 2000 Symposium*, Stockholm, Sweden, pag. 293-306, August 2000.
- [Lam'92] Lampson, B., et al. "Authentication in distributed systems: Theory and practice", *ACM Transactions on Computer Systems* 10(4):265-310, November 1992.
- [Le91] Le Gall, D. "MPEG: A video compression standard for multimedia applications", *Communications of the ACM* 34(1):46-58, April 1991.
- [Lin93] Lin, H.-A. "Estimation of the optimal performance of ASN.1/BER transfer syntax", *Computer Communications Review* 23(3):45-58, July 1993.
- [LM97] Lin, D. e R. Morris. "Dynamics of random early detection", *Proceedings of the SIGCOMM '97 Symposium*, pag. 127-136, Cannes, France, October 1997.
- [LMKQ89] Leffler, S.J., M.K. McKusick, M.J. Karels e J.S. Quarterman. *The Design and Implementation of the 4.3BSD UNIX Operating System*. Addison-Wesley, Reading, MA, 1989.
- [LPW02] Low, S., L. Peterson e L. Wang. "Understanding TCP Vegas: A duality model", *Journal of the ACM* 49(2):207-235, March 2002.
- [LPW'02] Low, S., F. Paganini, J. Wang, S. Adlakha e J. Doyle. "Dynamics of TCP/RED and a scalable control", *IEEE INFOCOM*, pag. 239-248, June 2002.
- [LS98] Lakshman, T.V. e D. Stiliadis. "High speed policy-based packet forwarding

- using efficient multi-dimensional range matching", *Proceedings of the SIGCOMM '98 Symposium*, pag. 203-214, September 1998.
- [LTWW94] Leland, W., M. Taqqu, W. Willinger e D. Wilson. "On the self-similar nature of Ethernet traffic", *IEEE/ACM Transactions on Networking* 2:1-15, February 1994.
- [Mal93] Malkin, G. "RIP version 2 carrying additional information", *Request for Comments* 1388, January 1993.
- [Mas86] Mashey, J. "RISC, MIPS, and the motion of complexity". *UniForum 1986 Conference Proceedings*, pag. 116-124, 1986.
- [MB76] Metcalfe, R. e D. Boggs. "Ethernet: Distributed packet switching for local computer networks", *Communications of the ACM* 19(7):395-403, July 1976.
- [McK99] McKeown, N. "The iSLIP scheduling algorithm for input-queued switches", *IEEE Transactions on Networking* 7(2):188-201, April 1999.
- [MD88] Mockapetris, P. e K. Dunlap. "Development of the domain name system", *Proceedings of the SIGCOMM '88 Symposium*, pag. 123-133, August 1988.
- [MD90] Mogul, J. e S. Deering. "Path MTU discovery", *Request for Comments* 1191, November 1990.
- [MD93] McKenney, P.E. e K.F. Dove. "Efficient demultiplexing of incoming TCP packets", *Proceedings of the SIGCOMM '92 Symposium*, pages 269-280, August 1993.
- [MD98] Madson, C. e N. Doraswamy. "The ESP DES-CBC cipher algorithm with explicit IV", *Request for Comments* 2405, November 1998.
- [Met93] Metcalf, R. "Computer/network interface design lessons from Arpanet and Ethernet", *IEEE Journal of Selected Areas in Communications (JSAC)* 11(2):173-180, February 1993.
- [MG98a] Madson, C. e R. Glenn. "The use of HMAC-MD5-96 within ESP and AH", *Request for Comments* 2403, November 1998.
- [MG98b] Madson, C. e R. Glenn. "The use of HMAC-SHA-1-96 within ESP and AH", *Request for Comments* 2404, November 1998.
- [Min93] Minoli, D. *Enterprise Networking: Fractional T1 to SONET, Frame Relay to BISDN*. Artech House, Norwood, MA, 1993.
- [MJ95] McCanne, S. e V. Jacobson. "vic: A flexible framework for packet video", *ACM Multimedia '95*, pag. 511-522, 1995.
- [MJV96] McCanne, S., V. Jacobson e M. Vetterli. "Receiver-driven layered multicast", *Proceedings of the SIGCOMM '96 Symposium*, pag. 117-130, September 1996.
- [Mor68] Morrison, D. "PATRICIA - A practical algorithm to retrieve information coded in alphanumeric", *Journal of the ACM* 15(4):514-534, October 1968.
- [Moy98] Moy, J. "OSPF version 2", *Request for Comments* 2328, April 1998.
- [MP85] Mogul, J. e J. Postel. "Internet standard subnetting procedure", *Request for Comments* 950, August 1985.
- [MPBO96] Mosberger, D., L. Peterson, P. Bridges e S. O'Malley. "Analysis of techniques to improve protocol latency", *Proceedings of the SIGCOMM '96 Symposium*, pag. 73-84, August 1996.
- [MPFL96] Mitchell, J.L., W.B. Pennebaker, C.E. Fogg e D.J. LeGall. *MPEG Video: Compression Standard*. Chapman Hall, New York, 1996.
- [MR91] McCloghrie, K. e M. Rose. "Management information base for network management of TCP/IP-based internets: MIB-II", *Request for Comments* 1213, March 1991.

- [MSST98] Maughan, D., M. Schertler, M. Schneider e J. Turner. "Internet security association and key management protocol (ISAKMP)", *Request for Comments* 2408, November 1998.
- [Mul90] Mullender, S. "Amoeba: A distributed operating system for the 1990s", *IEEE Computer* 23(5):44-53, May 1990.
- [MVS01] Moore, D., G. Voelker e S. Savage. "Inferring Internet denial of service activity", *Proceedings of 2001 USENIX Security Symposium*, pag. 9-22, August 2001.
- [Nel92] Nelson, M. *The Data Compression Book*. M&T Books, San Mateo, CA, 1992.
- [Nol97] Noll, P. "MPEG digital audio coding", *IEEE Signal Processing Magazine*, pag. 59-81, September 1997.
- [NRC94] National Research Council, Computer Science and Telecommunications Board. *Realizing the Information Future: The Internet and Beyond*. National Academy Press, Washington, DC, 1994.
- [NRC01] National Research Council. *Looking Over the Fence at Networks*. National Academy Press, Washington, DC, 2001.
- [NYKT94] Nahum, E.M., D.J. Yates, J.F. Kurose e D. Towsley. "Performance issues in parallelized network protocols", *Proceedings of the First USENIX Symposium on Operating System Design and Implementation (OSDI)*, pag. 125-137, November 1994.
- [OCD+88] Ousterhout, J.K., A.R. Cherenson, F. Douglis, M.N. Nelson e B.B. Welch. "The Sprite network operating system", *IEEE Computer* 21(2):23-36, February 1988.
- [OP91] O'Malley, S. e L. Peterson. "TCP extensions considered harmful", *Request for Comments* 1263, October 1991.
- [OP92] O'Malley, S. e L. Peterson. "A dynamic network architecture", *ACM Transactions on Computer Systems* 10(2):110-143, May 1992.
- [OPM94] O'Malley, S.W., T.A. Proebsting e A.B. Montz. "Universal stub compiler", *Proceedings of the SIGCOMM '94 Symposium*, pag. 295-306, August 1994.
- [OSF94] Open Software Foundation. *OSF DCE Application Environment Specification*. Prentice Hall, Englewood Cliffs, NJ, 1994.
- [PACR02] Peterson, L., T. Anderson, D. Culler e T. Roscoe. "A blueprint for introducing disruptive technology into the Internet", *Proceedings of HotNets-I*, October 2002.
- [Pad85] Padlipsky, M.A. *The Elements of Networking Style and Other Essays and Animadversions on the Art of Intercomputer Networking*. Prentice Hall, Englewood Cliffs, NJ, 1985.
- [Par94] Partridge, C. *Gigabit Networking*. Addison-Wesley, Reading, MA, 1994.
- [Par98] Partridge, C., et al. "A 50 Gb/s IP router", *IEEE/ACM Transactions on Networking* 6(3):237-247, June 1998.
- [Pax96] Paxson, V. "End-to-end routing behavior in the Internet", *SIGCOMM '96*, pag. 25-38, August 1996.
- [PB61] Peterson, W. W. e D. T. Brown. "Cyclic codes for error detection", *Proceedings of the IRE*, 49:228-235, January 1961.
- [Per85] Perlman, R. "An algorithm for distributed computation of spanning trees in an extended LAN", *Proceedings of the Ninth Data Communications Symposium*, pag. 44-53, September 1985.
- [Per00] Perlman, R. *Interconnections: Bridges, Routers, Switches and Internetworking Protocols*. Addison-Wesley, Reading, MA, second edition, 2000.

- [Pet88] Peterson, L.L. "The Profile naming service", *ACM Transactions on Computer Systems* 6(4):341-364, November 1988.
- [PF94] Paxson, V. e S. Floyd. "Wide-area traffic: The failure of Poisson modeling", *Proceedings of the SIGCOMM '94 Symposium*, pag. 257-268, London, UK, August 1994.
- [PFTK98] Padhye, J., V. Firoiu, D. Towsley e J. Kursoe. "Modeling TCP throughput: A simple model and its empirical validation", *ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pag. 303-314, Vancouver, Canada, 1998.
- [Pip98] Piper, D. "The Internet IP security domain of interpretation for ISAKMP", *Request for Comments* 2407, November 1998.
- [Pie84] Pierce, J. "Telephony - a personal view", *IEEE Communications* 22(5):116-120, May 1984.
- [PL01] Park, K. e H. Lee. "On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets", *Proceedings of ACM SIGCOMM '01*, pag. 15-26, August 2001.
- [PM97] Perkins, D. e E. McGinnis. *Understanding SNMP MIBS*. Prentice Hall, Upper Saddle River, NJ, 1997.
- [Pos81] Postel, J. "Internet Protocol", *Request for Comments* 791, September 1981.
- [Pos82] Postel, J. "Simple Mail Transfer Protocol", *Request for Comments* 821, August 1982.
- [QPP02] Qie, X., R. Pang e L. Peterson. "Defensive programming: Using an annotation toolkit to build DoS-resistant software", *Proceedings of OSDI '02*, pag. 45-60, December 2002.
- [Ram93] Ramakrishnan, K.K. "Performance considerations in designing network interfaces", *IEEE Journal of Selected Areas in Communication (JSAC)* 11(2):203-219, February 1993.
- [RD01] Rowstron, A. e P. Druschel. "Storage management and caching in PAST, a large-scale persistent peer-to-peer storage utility", *Proceedings of the 18th ACM Symposium on Operating Systems Principle (SOSP)*, pag. 188-201, October 2001.
- [RDR+97] Rekhter, Y., B. Davie, E. Rosen, G. Swallow, D. Farinacci e D. Katz. "Tag switching architecture overview", *Proceedings of the IEEE* 82(12):1973-1983, December 1997.
- [RF89] Rao, T.R.N. e E. Fujiwara. *Error-Control Coding for Computer Systems*. Prentice Hall, Englewood Cliffs, NJ, 1989.
- [RF94] Romanow, A. e S. Floyd. "Dynamics of TCP traffic over ATM networks", *Proceedings of the SIGCOMM '94 Symposium*, pag. 79-88, October 1994.
- [RFB01] Ramakrishnan, K., S. Floyd e D. Black. "The addition of explicit congestion notification (ECN) to IP", *Request for Comments* 3168, September 2001.
- [RFHKS01] Ratnasamy, S., P. Francis, M. Handley, R. Karp e S. Shenker. "A scalable content-addressable network", *Proceedings of the ACM SIGCOMM '01*, pag. 161-172, August 2001.
- [RHE99] Rejaie, R., M. Handley e D. Estrin. "RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet", *INFOCOM* (3), pag. 1337-1345, 1999.
- [Rit84] Ritchie, D. "A stream input-output system", *AT&T Bell Laboratories Technical Journal* 63(8):311-324, October 1984.

- [RJ90] Ramakrishnan, K. e R. Jain. "A binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer", *ACM Transactions on Computer Systems* 8(2):158-181, May 1990.
- [RL95] Rekhter, Y. e T. Li. "A Border Gateway Protocol 4 (BGP-4)", *Request for Comments* 1771, March 1995.
- [Rob93] Robertazzi, T.G., editor. *Performance Evaluation of High Speed Switching Fabrics and Networks: ATM, Broadband ISDN, and MAN Technology*. IEEE Press, Piscataway, NJ, 1993.
- [Ros86] Ross, F.E. "FDDI – A tutorial", *IEEE Communications* 24(5):10-17, May 1986.
- [ROY00] Rhee, I., V. Ozdemir e Y. Yi. "Tear: TCP emulation at receivers – Flow control for multimedia streaming", NCSU technical report, April 2000.
- [RR99] Rosen, E. e Y. Rekhter. "BGP/MPLS VPNs", *Request for Comments* 2547, March 1999.
- [RS01] Ramaswami, R. e K. Sivarajan. *Optical Networks: A Practical Perspective, Second Edition*. Morgan Kaufmann Publishers, San Francisco, second edition, 2001.
- [RS02] Rabinovich, M. e O. Spatscheck. *Web Caching and Replication*. Addison-Wesley, Reading, MA, 2002.
- [Sal78] Saltzer, J. "Naming and binding of objects", *Lecture Notes on Computer Science* 60:99-208, 1978.
- [Sat89] Satyanarayanan, M. "Integrating security in a large distributed system", *ACM Transactions on Computer Systems* 7(3):247-280, August 1989.
- [SB89] Schroeder, M.D. e M. Burrows. "Performance of Firefly RPC", *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pag. 83-90, December 1989.
- [SCFJ96] Schulzrinne, H., S. Casner, R. Frederick e V. Jacobson. "RTP: A transport protocol for real-time applications", *Request for Comments* 1889, January 1996.
- [Sch95] Schneier, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, New York, 1995.
- [SCH*99] Savage, S., A. Collins, E. Hoffman, J. Snell e T. Anderson. "The end-to-end effects of Internet path selection", *Proceedings of the ACM SIGCOMM Conference*, Cambridge, MA, pag. 289-300, September 1999.
- [SCJ*02] Schulzrinne, H., G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley e E. Schooler. "SIP: Session initiation protocol", *Request for Comments* 3261, June 2002.
- [Sha48] Shannon, C. A mathematical theory of communication. *Bell Systems Technical Journal* 27:379-423, 623-656, 1948.
- [Sho78] Shoch, J. "Inter-network naming, addressing e routing", *Seventeenth IEEE Computer Society International Conference (COMPICON)*, pag. 72-79, September 1978.
- [SHP91] Spragins, J., J. Hammond e K. Pawlikowski. *Telecommunications: Protocols and Design*. Addison-Wesley, Reading, MA, 1991.
- [SKPG01] Spalink, T., S. Karlin, L. Peterson e Y. Gottlieb. "Building a robust software-based router using network processors", *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pag. 216-229, Chateau Lake Louise, Banff, Alberta, Canada, October 2001.
- [SMKKB01] Stoica, I., R. Morris, D. Karger, F. Kaashoek e H. Balakrishnan. "Chord: A peer-to-peer lookup service for Internet applications", *Proceedings of the ACM SIGCOMM Conference*, pag. 149-160, August 2001.

- [SP99] Spatscheck, O. e L. Peterson. "Defending against denial of service attacks in Scout", *Proceedings of OSDI '99*, pag. 59-72, February 1999.
- [SPS*01] Snoeren, A.C., C. Partridge, L.A. Sanchez, C.E. Jones, F. Tchakountio, S.T. Kent e W.T. Strayer. "Hash-based IP traceback", *Proceedings of ACM SIGCOMM '01*, pag. 3-14, August 2001.
- [SRC84] Saltzer, J., D. Reed e D. Clark. "End-to-end arguments in system design", *ACM Transactions on Computer Systems* 2(4):277-288, November 1984.
- [Sri95a] Srinivasan, R. "RPC: Remote procedure call protocol specification version 2", *Request for Comments* 1831, August 1995.
- [Sri95b] Srinivasan, R. "XDR: External data representation standard", *Request for Comments* 1832, August 1995.
- [SS98] Sisalem, D. e H. Schulzrinne. "The loss-delay based adjustment algorithm: A TCP-friendly adaptation scheme", *Proceedings of NOSSDAV*. Cambridge, UK, pag. 215-226, 1998.
- [SSZ98] Stoica, I., S. Shenker e H. Zhang. "Core-stateless fair queuing: A scalable architecture to approximate fair bandwidth allocations in high-speed networks", *ACM SIGCOMM '98*, pag. 118-130, August 1998.
- [Sta00a] Stallings, W. *Data and Computer Communications*. Prentice Hall, Upper Saddle River, NJ, sixth edition, 2000.
- [Sta00b] Stallings, W. *Local and Metropolitan Area Networks*. Prentice Hall, Upper Saddle River, NJ, sixth edition, 2000.
- [Ste94a] Steenkiste, P. A. "A systematic approach to host interface design for high speed networks", *IEEE Computer* 27(3):47-57, March 1994.
- [Ste94b] Stevens, W. R. *TCP/IP Illustrated. Volume 1: The Protocols*. Addison-Wesley, Reading, MA, 1994.
- [SVSW98] Srinivasan, V., G. Varghese, S. Suri e M. Waldvogel. "Fast scalable level four switching", *Proceedings of the SIGCOMM '98 Symposium*, pag. 191-202, September 1998.
- [SW95] Stevens, W.R. e G.R. Wright. *TCP/IP Illustrated. Volume 2: The Implementation*. Addison-Wesley, Reading, MA, 1995.
- [SWKA00] Savage, S., D. Wetherall, A. Karlin e T. Anderson. "Practical network support for IP traceback", *Proceedings of the 2000 ACM SIGCOMM Conference*, pag. 295-306, August 2000.
- [SZ97] Stoica, I. e H. Zhang. "A hierarchical fair service curve algorithm for link-sharing and priority services", *Proceedings of the SIGCOMM '97 Symposium*, pag. 249-262, October 1997.
- [Tan01] Tanenbaum, A. S. *Modern Operating Systems*. Prentice Hall, Upper Saddle River, NJ, second edition, 2001.
- [Tan02] Tanenbaum, A. S. *Computer Networks*. Prentice Hall, Upper Saddle River, NJ, fourth edition, 2002.
- [Ter86] Terry, D. "Structure-free name management for evolving distributed environments", *Sixth International Conference on Distributed Computing Systems*, pag. 502-508, May 1986.
- [TL93] Thekkath, C.A. e H.M. Levy. "Limits to low-latency communication on high-speed networks", *ACM Transactions on Computer Systems* 11(2):179-203, May 1993.
- [TS93] Traw, C.B.S. e J.M. Smith. "Hardware/software organization of a high-

- performance ATM host interface", *IEEE Journal of Selected Areas in Communications (JSAC)* 11(2):240-253, February 1993.
- [Tur85] Turner, J.S. "Design of an integrated services packet network", *Proceedings of the Ninth Data Communications Symposium*, pag. 124-133, September 1985.
- [USC-ISI81] USC-ISI. "Transmission Control Protocol", *Request for Comments* 793, September 1981.
- [VL87] Varghese, G. e T. Lauck. "Hashed and hierarchical timing wheels: Data structures for the efficient implementation of a timer facility", *Proceedings of the 11th ACM Symposium on Operating Systems Principles*, pag. 25-38, November 1987.
- [Wal91] Wallace, G.K. "The JPEG still picture compression standard", *Communications of the ACM* 34(1):30-44, April 1991.
- [Wat81] Watson, R. "Identifiers (naming) in distributed systems", In B. Lampson, M. Paul e H. Siegert, editors, *Distributed System - Architecture and Implementation*, pag. 191-210. Springer-Verlag, New York, 1981.
- [WC91] Wang, Z. e J. Crowcroft. "A new congestion control scheme: Slow start and search (Tri-S)", *ACM Computer Communication Review* 21(1):32-43, January 1991.
- [WC92] Wang, Z. e J. Crowcroft. "Eliminating periodic packet losses in 4.3-Tahoe BSD TCP congestion control algorithm", *ACM Computer Communication Review* 22(2):9-16, April 1992.
- [Wel84] Welch, T. "A technique for high-performance data compression", *IEEE Computer* 17(6):8-19, June 1984.
- [WM87] Watson, R.W. e S.A. Mamrak. "Gaining efficiency in transport services by appropriate design and implementation choices", *ACM Transactions on Computer Systems* 5(2):97-120, May 1987.
- [WMB99] Witten, I.H., A. Moffat e T.C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, San Francisco, 1999.
- [WPP02] Wang, L., V. Pai e L. Peterson. "The effectiveness of request redirection on CDN robustness", *Proceedings of OSDI '02*, pag. 345-360, December 2002.
- [WVTP97] Waldvogel, M., G. Varghese, J. Turner e B. Plattner. "Scalable high speed routing lookups". *Proceedings of the SIGCOMM '97 Symposium*, pag. 25-36, October 1997.
- [YHA87] Yeh, Y.-S., M.B. Hluchyj e A.S. Acampora. "The knockout switch: A simple, modular architecture for high-performance packet switching", *IEEE Journal of Selected Areas in Communication (JSAC)* 5(8):1274-1283, October 1987.
- [ZDE*93] Zhang, L., S. Deering, D. Estrin, S. Schenker e D. Zappala. "RSVP: A new resource reservation protocol", *IEEE Network* 7(9):8-18, September 1993.
- [Zim80] Zimmerman, H. "OSI reference model - the ISO model of architecture for open systems interconnection", *IEEE Transactions on Communications COM-28(4):425-432*, April 1980.
- [ZL77] Ziv, J. e A. Lempel. "A universal algorithm for sequential data compression", *IEEE Transactions on Information Theory* 23(3):337-343, May 1977.
- [ZL78] Ziv, J. e A. Lempel. "Compression of individual sequences via variable-rate coding", *IEEE Transactions on Information Theory* 24(5):530-536, September 1978.

Indice analitico

- 10Base2, 98
 10BaseT, 98
 3DES, 514
 4B/5B, 66-67, 111
 6-Bone, 599
- A**
- AAL (ATM Adaptation Layer), 173-177
 AAL3/4, 174-176
 AAL5, 176-177
A
 Area Border Router, 276-277
 Available Bit Rate, 450-452
 AC, coefficienti, 485
 access point (AP), 116-118
 accesso diretto alla memoria (DMA), 122-124, 184
 accesso multiplo, 5
 accodamento, 401-407
 discipline, 392
 equo (FQ, fair queueing), 403-407
 FCFS, 401
 FIFO, 11, 401-403
 prioritario, 402-403
 ritardi di, 37, 44
 acknowledgement (ACK), 84
 duplicato, 415
 implicito, 362
 negativo (NAK), 88
 parziale, 356
 per rallentare la trasmissione, 407
 selettivo, 88, 356, 376
 active scanning, 118
 ad hoc, reti mobili, 260
 adattativa, codifica video, 495-496
 adattativa, ritrasmissione, 346-349
 adattatori di rete, 119-127
 definizione, 57
 device driver, 125
 DMA (direct memory access), 122-124
 gestione, 27
 interfaccia, 119-120
 interruzioni, 121-122
- PIO (programmed I/O), 122-124
 ADSL (asymmetric DSL), 61
 advertised window, 351
 AF (assured forwarding), 446-448
 affidabile, flusso di byte, 352-353
 affidabile, inondazione, 246-248
 affidabile, protocollo per datagrammi, 265
 affidabilità, 15-16
 agente esterno, 257-260
vedi anche instradamento per host mobili
 agente residenziale, 257-260
vedi anche instradamento per host mobili
 agenti di mobilità, 257-259
 aggiornamenti, 242-243
 BGP, 273-274
 delle tabelle di instradamento, 242-243
 periodici, 241
 triggered, 241
vedi anche instradamento
 aggregazione gerarchica, 222, 228
 AH (authentication header), 537-539
vedi anche IPSEC
 AIMD, 407-410
vedi anche controllo di congestione
 alberi
 aciclici, 161-162
 condivisi, 293-295
 multicast, 289-293
 PATRICIA, 269
 specifici per una sorgente, 293, 296
 ALF (Application Level Framing), 582
 algoritmi crittografici, 506-519
 chiave pubblica, 506
 chiave segreta, 506
 DES, 507-514
 hashing, 506-507
 implementazione, 519
 MD5, 507, 516-519
 PGP, 530-531
 prestazioni, 519
 requisiti, 508-509
 rischi di installazione, 508
 RSA, 507, 514-516

vedi anche sicurezza
algoritmo hash, 506-507
a modulo, 616
coerenti, 608-609, 616-617
algoritmo di corrispondenza esatta, 298
algoritmo di Dijkstra, 248, 289
algoritmo di Jacobson/Karels, 347-348
algoritmo di Karn/Partridge, 347
algoritmo di Nagle, 345-346
algoritmo di ricerca in avanti (forward search), 249
alias, 558
allocazione di risorse, 392-401
aumento del throughput, 398-399
basata sul feedback, 396-397
basata sulla finestra, 397
basata sulla prenotazione, 396-397
basata sulla velocità, 397
 criteri di valutazione, 398-401
efficace, 398-400
equa, 400-401
focalizzata sugli host, 396
focalizzata sui router, 396
implementazione, 392-393
in reti a commutazione di pacchetto, 393-394
modello di servizio, 396
 problemi, 392-401
riassunto, 453-454
tassonomia, 396-398
vedi anche controllo di congestione
Aloha, 96
ampiezza di banda (bandwidth), 35-40
 del bus di I/O, 184-185
della memoria, 125-127, 184-185
di picco, 126
di una linea fisica, 35-36
di una linea noleggiata, 60
di una rete, 126-127
e latenza, 35-40
effettiva, 126-127
gestore di sottorete (SBM), 443
in relazione al throughput, 36
in relazione alla latenza, 42
in reti ad alta velocità, 41-42
requisiti di, 35-36
RTCP, 586
vedi anche prestazioni
amplificatori ottici, 157-158
AMPS (Advanced Mobile Phone System), 63
anycast, 288
AP (access point), 116-118
API (application programming interface), 26-28
applicazioni, 2-3, 553-619
adattative, 434-435
audio in tempo reale, 431-433
di conferenza, 578

di gestione della rete, 575-577
di posta elettronica, 563-570
di trasferimento file, 430
di videoconferenza, 3
di video-on-demand, 2-3
DNS, 554-562
esempio di, 29-31
ibride, 596-597
in tempo reale, 430-435
multimediali, 577-596
necessità di prestazioni delle, 43-44
non in tempo reale, 430
per voce e video, 430
profilo di, 589
protocolli specifici per, 380-381
sdr, 3
streaming, 2-3, 578-579
tradizionali, 562-577
vat, 3, 578, 580
vic, 3
wb, 3
apprendimento, bridge, 158-161
architettura di rete, 16-25
 a strati, 16-22
definizione, 2, 16, 45
Internet, 23-25
OSI, 22-23
area, 275-277
vedi anche instradamento
ARP (Address Resolution Protocol), 181, 561, 225-228
 ATM, 227-228
definizione, 225
messaggi, 258-259
proxy, 258-259
tabella, 225-226
ARPANET, 23-24
instradamento in, 254-255
ARQ (automatic repeat request), 128
algoritmi, 84-95
canali logici concorrenti, 95
definizione, 84
punto-punto, 119
sliding window, 86-95
stop-and-wait, 84-86
AS (autonomous system, sistema autonomo), 262
concreti, 269-270
di transito, 271
multihomed, 271
protocolli di instradamento intradominio, 269-270
router di confine, 270
router di default, 270
stub, 271
ASCII, 563-566
asincroni, protocolli, 365

ASN.1 (Abstract Syntax Notation One), 475-477
assured forwarding (AF), 446-448
ATM (Asynchronous Transfer Mode), 144, 167-184
 ATM-over-SONET, 179
ATMARP, 227-228
caratteristiche, 153
celle, 168-172
come LAN di backbone, 179-180
commutatori, 179-180, 192
commutatori come LSR di MPLS, 300-301
confronto con Ethernet, 179-180
definizione, 167
emulazione di circuito con tunnel, 304-305
EPD (early packet discard), 424
formato delle celle, 172
futuro di, 193
implementazione, 184-192
nelle LAN, 179-184
percorsi virtuali, 178
prestazioni, 184-192
Q.2931, 168
reti, 60, 222
segmentazione e ricostruzione, 173-177
segnalazione, 168, 437
standardizzazione, 171
strati fisici per, 178-179
vedi anche QoS
ATMARP, 227-228
ATM-over-SONET, 179
attacco
di tipo denial-of-service, 545-546
mediante SYN, 545
aumento additivo: *vedi* AIMD
autenticazione, 519-522
PGP, 530
vedi anche sicurezza
autoconfigurazione, 286
autonomo, sistema: *vedi* AS

B

backbone, 261
backoff esponenziale, 102
bandwidth: *vedi* ampiezza di banda
banyane, reti, 190-192
baud rate, 67
BER (Basic Encoding Rules) di ASN.1, 475
best-effort, 326
definizione, 206-207
modello, 398
BGP (Border Gateway Protocol), 270-275, 618
IBGP, 275
vedi anche instradamento interdominio
big-endian, 469
XDR, 474
BISYNC (Binary Synchronous Communication protocol), 69

bit rate, 67
bit stuffing, 71
BLAST, 355-361
in una semplice pila RPC, 371
SRR, 356
vedi anche RPC
BOOTP, 231
border router, 270
bridge, 156-157
ad apprendimento, 158-161
chiusi ad anello, 161
come nodi dello strato di linea, 222
definizione, 156, 222
designato, 162-165
Ethernet, 156
in LAN estese, 161-162
inoltro di frame, 162
multiporta, 222
radice, 162-165
tabella di instradamento, 159
broadcast, 8
indirizzi, 100
percorso più breve, 290-292
reverse path (RPB), 290-292
supporto di, 165
BSD (Berkeley Software Distribution), 244
buffer, 34-35
capacità dei, 43
come fonte di ritardi, 189
di ricezione, 339-342
di riproduzione (playback), 431-432, 580
di trasmissione, 338-339
di uscita, 188
e caratteristiche di QoS, 189
e porte, 188-189
implementazione FIFO, 188
interni, 188
burst, 15, 82
BUS (Broadcast and Unknown Server), 182-184

C

cache
algoritmo di sostituzione della, 161
caching, 574-575
di binding, 260
calcolo di percorsi, 248-250
canali, 12-15
a flusso di messaggi, 14
capacità disponibile, 43
logici, 12
logici concorrenti, 95
richiesta/risposta, 361
care-of, indirizzi, 258-260
carico
bilanciamento, 251, 614
controllato, 436
controllo, 399

impostazione, 399
rapporto throughput/ritardo in funzione del, 399
CARP (Cache Array Routing Protocol), 617
catena di fiducia, 526-527
cavi, 59
vedi anche linee di collegamento
CBC (cipher block chaining), 513, 522-524
CBR (constant bit rate), 449
CDN (content distribution network), 613-618
Akamai, 613
bilanciamento del carico, 614
come reti sovrapposte, 616
commerciali, 613-615
Digital Island, 613
fattori di distribuzione, 614
"flash crowd", 613
meccanismi, 615-616
redirector, 614
server supplenti, 613
throughput di sistema, 614
celle, 168-172
caratteristiche, 168
commutazione, 167-184
confini, 179
dimensione, 169-171
e code, 169-171
e utilizzazione della linea, 169-171
formato (NNI, UNI), 172
inoltro lungo tunnel, 306
lunghezza del carico utile, 171
RM, 451-452
vedi anche ATM
certificati, 526-528
e protocollo di handshake, 535-536
standard X.509, 528
vedi anche sicurezza
CHAN, 361-369
in confronto a SunRPC, 372
in una semplice pila RPC, 371
vedi anche RPC
checksum, 75-76
critografica, 507
di Internet (algoritmo), 77-78
MD5, 524-525
TCP, 334
UDP, 327
varianti, 76
vedi anche rivelazione d'errore
chiave privata, 506
chiave pubblica, 506
algoritmo, 506
autenticazione, 522
cifratura, 506, 514-516
costruzione, 514
critografia, 506
distribuzione, 526-528

chiave segreta, 505
algoritmo, 506
cifratura, 506, 509-514
chiavi di demux, 21, 28
chiavi, mazzo di, 530
chipping, 114
CIDR (classless interdomain routing), 266-269, 308
vedi anche instradamento
cifratura (encryption), 505
chiave pubblica, 506, 514-516
chiave segreta, 505-506, 509-514
PGP, 530-531
vedi anche sicurezza
ciphertext (messaggio cifrato), 505
circuito virtuale (VC), 148-153
commutati (SVC), 149
permanenti (PVC), 149
Classical IP over ATM, 227-228
client, 14
applicazione d'esempio, 29-30
SMTP, 567-568
SNMP, 577
vedi anche server
code
best-effort, 448
controllo del comportamento, 169-170
e celle, 169-171
gestione attiva, 419
premium, 448
tempo di uscita dalle, 170
codice a correzione d'errori (ECC), 81
codifica, 55, 64-67, 127.
4B/5B, 66-67
delta, 482
fase di, 468
Manchester, 66
NRZ, 64-66
NRZI, 66-67
video adattativa, 495-496
coefficiente DC, 485
coefficienti AC, 485
coerente, hashing, 608-609, 616-617
collegamenti ipertestuali, 570
commutatori, 7, 222
4 x 4, 190
a bus condiviso, 189
a memoria condivisa, 189
ATM, 179-180, 192
Batcher-banyani, 192
come nodi del livello di rete, 222
congestionati, 144
crossbar, 189
di circuito, 143
di livello 4, 541
di livello 7, 596
di pacchetto, 143
esempio di utilizzo, 10
Ethernet, 193
Frame Relay, 223
grafo di protocolli, 145
in confronto a router, 222
LAN, 156-157, 222
ottici, 157-158
prestazioni, 185-187
throughput, 186-187
topologia a stella, 145
WAN, 222
commutazione, 144-156
di celle, 167-184
di circuito virtuale, 146, 148-153
di datagrammi, 146-148
di pacchetti, 143-193
instradamento dalla sorgente, 146, 154-156
ottica, 157-158
compilatore per stub, 354, 473
compressione, 480-498
algoritmi, 481-483
audio, 496-498
con perdite, 480
di immagini, 483-488
di Lempel-Ziv (LZ), 483
JPEG, 483-488
metodi basati su dizionario, 483
MP3, 496-498
MPEG, 488-496
rapporto di, 481
senza perdite, 480-483
video, 488-496
condivisione di risorse, 8-12
conferma: *vedi acknowledgement*
configurazione di host con DHCP, 229-231
congestione, 154
collasso per, 400
e modello X.25, 154
e segmenti perduti, 347
in reti a commutazione di pacchetto, 393-394
vedi anche controllo di congestione, evitare la congestione, finestra di congestione
connectionless flows (flussi privi di connessione), 394-395
connessioni persistenti, 574
connessioni TCP, 2, 29-332
diagramma di transizione di stato, 335-338
flussi MPEG, 494
handshake in tre fasi, 334-335
HTTP, 574
instaurazione e terminazione, 334-338
tempORIZZAZIONI (timeout), 415-416
vedi anche TCP
connettività, 3-8, 45
host-to-host, 8
internetwork, 7-8
linee di collegamento, 5
nodi, 5-8
"nuvoie", 7
contesa, 144, 154
controllo di accesso al mezzo: *vedi MAC*
controllo di congestione, 407-416
aumento additivo/decremento moltiplicativo (AIMD), 407-410
basato su equazioni, 452-453
comportamento, 412-414
coppie di pacchetti, 414
definizione, 329, 391-392, 407
e applicazioni in tempo reale, 452
e controllo di flusso, 393
e instradamento, 393-394
equità, 393
finestra di congestione, 407-416
partenza lenta, 410-414
recupero veloce, 416
TCP, 329, 407-416
"TCP-friendly", 453
valutazione, 429
vedi anche allocazione di risorse
controllo di flusso
definizione, 329
e controllo di congestione, 393
e UDP, 328
hop-by-hop, 153
sliding window per il, 94-95
TCP, 330, 340-342
controllo di sessione, 578, 588-596
CORBA (Common Object Request Broker Architecture), 374
correzione d'errori, 75
e rivelazione d'errori, 83
ECC, 81
in avanti (FEC, forward), 83
CRC (cyclic redundancy check), 69, 75-76
aritmetica polinomiale modulo 2, 79
calcolo, 78-82
CRC-32, 76
mediante divisione di polinomi, 79-82
mediante registri a scorrimento, 83
polinomi comuni, 82
polinomio divisore, 78-80
vedi anche rivelazione d'errore
critografici, algoritmi: *vedi* algoritmi critografici
CRL (certificate revocation list), 528
crossbar, commutatori, 189
CSMA/CD (Carrier Sense Multiple Access with Collision Detect), 55, 96
CSPF (constrained shortest path first), 304
CSR (control status register), 120-121
CSRC (contributing source), 582-583
CTS (Clear to Send), frame, 116

D
 daemon di posta elettronica, 566
 DAS (dual attachment station), 110
 datagrammi, 146-148
 e instradamento dalla sorgente, 154-155
 IP, 206-207, 209-211
 IP di lunghezza variabile, 223
 IP prodotti da frammentazione, 210-211
 vedi anche inoltro di datagrammi
 dati end-to-end, 467-498
 compressione, 480-498
 formato di presentazione, 468-479
 DC, coefficiente, 485
 DCE (Distributed Computing Environment), 374
 NDR, 476-477
 DCE-RPC, 374-377
 vedi anche RPC
 DCT (discrete cosine transform), 484-486
 DDCMP (Digital Data Communication Message Protocol), 68-70
 DDoS (distributed denial-of-service), 546, 614
 DECbit, 417-418
 vedi anche evitare la congestione
 decifrazione, 505
 decompressione, 480-481
 demultiplexing, 21-22
 denial-of-service distribuito (DDoS), 546, 614
 denial-of-service, attacco, 545-546
 DES (Data Encryption Standard), 506-514
 vedi anche algoritmi critografici
 designato, bridge, 162-165
 designato, router, 295
 destinazioni virtuali, 451
 DHCP (Dynamic Host Configuration Protocol), 229-231
 DHT (distributed hash table), 609, 612-613
 diagramma di transizione di stato, 335-338
 vedi anche connessioni TCP
 Differentiated Services (DiffServ), 445-448
 vedi anche QoS
 differenziale, Manchester: *vedi* Manchester
 differenziale
 differenziati, servizi: *vedi* Differentiated Services
 Diffie-Hellman, scambio di chiavi, 523
 DiffServ: *vedi* Differentiated Services
 digest: *vedi* MD5
 digitale, firma, 524-526
 Dijkstra, algoritmo di, 248, 289
 diminuzione moltiplicativa: *vedi* AIMD
 direct sequence (sequenza diretta), 114
 Directorate, IPng, 278-279
 dispositivi di livello 2, 222
 distribuita, tabella hash (DHT), 609, 612-613
 distribuito, instradamento, 238
 divisione di tempo, multiplexing sincrono (STDM), 9-11

dizionario, metodi di compressione basati su, 483
 DMA (direct memory access), 122-124, 184
 vedi anche adattatori di rete
 DNA (Digital Network Architecture), 417
 DNS (domain name system), 554-562
 convenzioni per i nomi, 561
 gerarchia dei domini, 555-557
 nomi, 554-555
 server per i nomi, 556-559, 615
 spazio per i nomi gerarchico, 555
 traduzione dei nomi, 559-642
 dominio di collisione, 98
 dominio, gerarchia, 555-558
 dominio, sistema dei nomi di: *vedi* DNS
 DPCM (Differential Pulse Code Modulation), 482
 DR (designated router), 295
 driver di dispositivi, 27, 125
 vedi anche adattatori di rete
 DSBM (designated subnet bandwidth manager), 443
 DSL (digital subscriber line), 61-62
 ad altissima velocità (VDSL), 62
 asimmetrica (ADSL), 61
 vedi anche linee di collegamento dell'ultimo miglio
 duplicate, conferme, 415
 DWDM (dense wavelength division multiplexing), 157

E

ECC (error correcting code), 81
 ECN (Explicit Congestion Notification), 419
 EF (expedited forwarding), 446
 EGP (Exterior Gateway Protocol), 270
 elenco di certificati revocati (CRL), 528
 eliminazione, 122
 anticipata casuale, 419
 anticipata di un pacchetto (EPD), 424
 probabilità di, 419-421
 terminale, 401-402
 terminale con RED, 422
 e-mail (posta elettronica), 563-570
 encryption: *vedi* cifratura
 end system multicast, 600-603
 EPD (early packet discard), 424
 equazioni, controllo di congestione basato su, 452-453
 errori
 a burst, 15, 82
 di bit, 15
 notifica di, 232
 su due bit, 82
 su un singolo bit, 82
 errori, correzione di: *vedi* correzione d'errori
 errori, rivelazione di: *vedi* rivelazione d'errori

ESP (Encapsulating Security Payload), 537-540
 vedi anche IPSEC
 esponenziale, backoff, 102
 estesa LAN: *vedi* LAN estesa
 Ethernet, 96-103
 10Base2, 98
 10BaseT, 98
 a confronto con ATM, 179-180
 algoritmi di trasmissione, 100-102
 backoff esponenziale, 102
 bridge, 156
 bridging, 144
 commutatori, 193
 controllo di accesso, 98-102
 dominio di collisione, 98
 Fast Ethernet, 96
 formato del frame, 96
 Gigabit Ethernet, 96, 180
 host, 103
 hub, 98
 identificativi globali univoci, 146
 in condizioni di poco carico, 103
 in confronto a token ring, 104
 indirizzi, 99-100
 indirizzi, corrispondenza con indirizzi IP, 226
 MAC (media access control), 98
 proprietà fisiche, 96-98
 protocolli con persistenza p, 100-101
 protocollo con persistenza 1, 100
 ripetitori, 97-98
 scenario di caso peggiore, 102
 schede d'interfaccia, 184
 segmenti multipli, 97
 transceiver e adattatori, 96
 evitare la congestione, 416-430
 basandosi sulla sorgente, 424-430
 DECbit, 417-418
 interazione con ABR, 451
 RED, 418-424
 Extensible Markup Language (XML), 477-479
 External Data Representation (XDR), 473-475

F

fabric: *vedi* matrice
 Fast Ethernet, 96
 fast recovery (recupero veloce), 416
 fast retransmit, (ritrasmissione veloce), 414-416
 FCFS (first-come first-served), accodamento, 401
 FDDI (Fiber Distributed Data Interface), 24, 103, 110-113
 vedi anche token ring
 FDM (frequency-division multiplexing), 9
 FEC (forward error correction), 83
 feedback, allocazione di risorse basata sul, 396-397
 Fiber Channel, 11

Ethernet, 99
 FDDI, 113
 HDLC, 70-71
 MPEG, 488-491
 ordinamento di, 94-95
 orfano, 109
 PPP, 69
 ricevuto, 123-124
 RTS, 116
 runt, 71-75
 SONET, 71-75
 STS, 73-75
 token ring, 109-110
 trasmissione, 123
 wireless, 118-119
 Frame Relay, 153
 commutatori, 223
 framing, 55, 67-75, 127
 frammentazione, 209-211
 campi di intestazione, 212
 con conferme selettive, 376
 e ricostruzione, 209-217
 frequenza di hopping, 114
 FTP (File Transfer Protocol), 13, 24-25
 proxy, 543
 full-duplex, 59
 funzione di hash, 506-507

G
 garantito, servizio, 436
 gateway di posta elettronica, 566
 gateway: *vedi* router
 gather-write, 123
 gerarchico, spazio dei nomi, 554-556
 gerarchia di dominio, 555-558
 gerarchica, aggregazione, 222, 228
 gerarchico, indirizzamento, 221
 gestione attiva della coda, 419
 gestione della concorrenza, 371
 gestione della rete, 575-577
 SNMP, 575-577
 gestore di interruzioni, 122
 GIF (Graphical Interchange Format), 467
 Gigabit Ethernet, 96
 linee di collegamento, 180
 vedi anche Ethernet
 globali, indirizzi, 158-159, 217-219
 GMPLS (generalized MPLS), 302
 Gnutella (rete peer-to-peer), 606-607
 grafo di protocolli, 18-19
 griglia sovrapposta, 600-601
 gruppo di lavoro per Mobile IP, 256-257
 GSM (Global System for Mobile Communication), 64

H
 H.323, 595-596

half-duplex, 59
 handshake in tre fasi, 334-335
 per l'autenticazione, 520-521
 handshake, protocollo, 535-537
 hard, stato, 395
 hash, algoritmi: *vedi* algoritmi hash
 hash, funzione, 506-507
 hash, tabella: *vedi* tabella di hash
 hash, tabella distribuita (DHT), 609, 612-613
 hashing coerente, 608-609, 616-617
 HDLC (High-level Data Link Control), 70-71
 HIPERLAN (High Performance European Radio LAN), 64
 HiPPI (High Performance Parallel Interface), 11
 home address (indirizzo residenziale), 257
 home agent (agente residenziale), 257-260
 hop-by-hop, controllo di flusso, 153
 hopping, frequenza di, 114
 HTML (HyperText Markup Language), 477-478
 definizione, 563
 documenti, 573
 HTTP (HyperText Transfer Protocol), 2, 25.
 562-563
 80 (porta ben nota), 452
 caching, 574-575
 connessioni persistenti, 574
 connessioni TCP, 574
 formato dei messaggi, 570-572
 messaggi di richiesta, 572-573
 messaggi di risposta, 573-574
 operazioni, 536, 572-573
 proxy, 542-543
 redirezione, 615-616
 sicuro (HTTPS), 534
 tipi di intestazione, 572
 utilizzo, 562-563
 hub, 98

I
 I/O programmato: *vedi* PIO
 IBGP (interior BGP), 275
 ICMP (Internet Control Message Protocol), 232
 IDEA (International Data Encryption Algorithm), 506
 IEEE (Institute of Electrical and Electronics Engineers)
 802.3, 96-103
 802.5, 55, 103-110
 802.11, 55, 64, 114-119
 IETF (Internet Engineering Task Force), 20
 IGMP (Internet Group Management Protocol), 289
 IGP (interior gateway protocol), 237
 IKE (Internet Key Exchange), 538
 IMAP (Internet Message Access Protocol).
 568-570
 immagini a colori, 487-488
 vedi anche JPEG

immagini, compressione di, 483-488
 implicita, conferma, 362
 inaffidabile, servizio, 207
 encapsulamento, 20-21
 indicazione oraria (timestamp), 350-351
 indice di equità, 400-401
 indirizzamento
 di sottorete, 263-264
 geografico, 282-283
 gerarchico, 221
 problema, 203
 indirizzi
 anycast, 288
 care-of, 258-260
 di record, 594
 di sorgente, 158-159, 165
 efficienza di assegnazione, 266
 flat, 554
 globali, 158-159, 217-219
 hardware, 259
 home, 257
 inefficienza di assegnazione, 262
 LANE, 181
 multicast, 288
 nomi tradotti in, 555
 NSAP, 279
 residenziali, 257
 tradurre nomi in, 555
 two-tier, 372
 unicast, 282-283
 utilizzazione degli, 262
 vedi anche indirizzi IP
 indirizzi IP, 217-219, 499
 blocchi di, 309
 classi di, 218-219, 279
 come numeri interi decimali, 219
 e nomi di dominio Internet, 219
 e struttura delle internetwork, 229
 flessibilità, 218-219
 gerarchici, 217
 globali, 217-219
 in messaggi di richiesta ARP, 225
 IPv4, 308-309
 IPv6, 279-281
 parte di host, 217-218, 225
 parte di rete, 217-218
 partizionamento, 228
 prefissi, 309
 residenziali, 257
 scopo originario, 262
 spazio degli, 266
 traduzione di nomi in, 560-562
 unicità, 228
 inoltro
 accelerato (EF, expedited forwarding), 446
 basato sulla destinazione, 297-302
 centralizzato, 224
 di celle ATM lungo un tunnel, 306
 di datagrammi, 146-148, 219-224
 di pacchetti, 596
 distribuito, 224
 e instradamento, 236
 garantito (AF, assured forwarding), 446-448
 IP, 268
 vedi anche commutazione
 inoltro, tabella di: *vedi* tabella di inoltro
 inondazione affidabile, 246-248
 vedi anche instradamento a stato delle linee
 input, porta, 185-189
 insieme delle foglie, 610-613
 vedi anche nodi
 instradamento, 8, 146, 219
 aggiornamenti, 241-244
 aree, 275-277
 autenticazione di messaggio, 251
 cicli, 241-242
 come problema di teoria dei grafi, 237
 complessità, 257
 direzionato dalla sorgente, 288
 distribuito, 238
 domini, 269
 e controllo di congestione, 33-394
 e inoltro, 236
 esplicito, 302-304
 interdominio, 269-275
 intradominio, 275
 IP, 204
 IPv6, 279
 metriche, 253-256
 monitoraggio del comportamento, 257
 nel backbone di Internet, 271
 per host mobili, 256-260
 problema del percorso di costo minimo, 237
 senza classi, 266-269
 sorgente, 146, 154-156, 302-303
 stato delle linee, 245-253
 triangolo, 259-260
 verso sottoreti, 266
 vettore di distanza, 238-244
 instradatore: *vedi* router
 Integrated Services (IntServ), 436-445
 vedi anche QoS
 integrità di messaggio, 522-525
 definizione, 505
 PGP, 530
 vedi anche sicurezza
 interdominio, instradamento, 269-275
 interfaccia
 di servizio, 18
 socket, 26-28
 tra pari, 18
 tra protocolli, 34
 Internet
 architettura, 23-25

checksum. 77-78
definizione. 204-205
domini di instradamento. 269
in confronto ad una rete ATM. 222
instradamento nel backbone, 271
multibackbone. 271
nomi di dominio. 219
problemi di scalabilità. 262
struttura ad albero. 261
topologia. 261
traffico in transito. 270-271
traffico locale. 270-271
Internet Protocol: *vedi IP*
internet: *vedi internetwork*
internetwork. 203-209
definizione. 7
protocolli di instradamento. 244
tunnel. 234
interruzioni. 121-122
gestore di. 122
intestazioni di estensione. 285-286
intradominio, instradamento. 275
IP. 204-235
aggregazione gerarchica. 228
configurazione di host. 229-231
definizione. 205-206
dispositivi. 499
inoltro di datagrammi. 219-224
instradamento. 204
reti. 204, 307
reti virtuali. 232-233
segnalazione di errori. 232
tunnel. 233-235
versioni sperimentali. 598-599
IP, datagrammi: *vedi datagrammi IP*
IP, indirizzi: *vedi indirizzi IP*
IP, modello di servizio. 206-207
ipertestuali, collegamenti. 570
IPng Directorate. 278-279
IPSEC (IP Security). 537-540
funzionamento. 528-529
vedi anche sicurezza
IPv4, intestazione del pacchetto. 207-209
IPv6. 204, 277-288, 308-309
problemi di installazione 309
supporto al multicast. 288-289
vedi anche IP
IPX, 206
ISAKMP (Internet Security Association and Key Management Protocol). 537-538
ISDN (Integrated Services Digital Network). 61
ISO (International Standards Organization). 20-23
ISSLL (Integrated Services and Specific Link Layers). 443
ITU (International Telecommunications Union). 23

"serie H", 492
H.245, 596
H.255, 596
H.323, 595-596

J
Jacobson/Karels, algoritmo. 347-348
jitter. 43-44
JPEG (Joint Photographic Experts Group). 467
Karn/Partridge, algoritmo. 347

L
LAN (local area network), 11
ATM in, 179-184
bridge, 222
commutazione, 144
estesa, 156, 162, 166-167
interconnessione, 156
LANE, 181-184
supporto al broadcast, 165
supporto al multicast, 165
virtuale (VLAN), 166
LAN estesa, 156, 166-167
bridge, 161-162
con anelli. 161-162
e scalabilità. 166
vedi anche LAN
LANE (LAN emulation). 181-184
latenza, 35-40
e reti ad alta velocità. 41-42
e ritrasmissione. 430
in relazione alla banda, 42
memoria, 57
percepita, in confronto a RTT, 38
protocolli end-to-end, 378
variabilità, 432
vedi anche ritardo; prestazioni
LCP (Link Control Protocol). 70
LDAP (Lightweight Directory Access Protocol). 559
LDP (Label Distribution Protocol). 297
learning bridge, 158-161
LER (label edge router). 298
lettore di posta elettronica. 566-570
linee di collegamento, 5
ad accesso multiplo, 5
attributi, 58-59
cavi, 59
codifica di un flusso di bit. 58-59
dell'ultimo miglio. 61-63
full-duplex, 59
Gigabit Ethernet, 180
guasto fisico, 15-16
half-duplex, 59
noleggiate, 60-61
OC-48, 343
punto-punto, 5

SONET, 157-158
sovrafficate, 253
STS-N, 60
via satellite, 255
wireless, 63-64
LIS (logical IP subnet). 227-228
little-endian, 469
livello 2, dispositivi. 222
livello 4, commutatori, 541
livello 7, commutatori, 596
logici, canali, 12
LSA(link-state advertisement). 252-253
vedi anche OSPF
LSP (link-state packet). 246-248
LSR (label switching router). 300-302
lunghezza della coda, 170
calcolo della media. 417-418
calcolo pesato. 420-421
gestione, 423-424
LZ (Lempel-Ziv). 483
MAC (media access control). 55
Ethernet, 55
token ring, 106-108

M
MACA (Multiple Access with Collision Avoidance). 116
Macromedia FLASH, 492
MAN (metropolitana area network). 11
Manchester (codifica). 66
differenziale, 66, 109
token ring, 109
vedi anche codifica
marcatore per dati, 472-473
marshalling, 470-473
maschere di sottorete, 263-266
matrici, 189-192
a bus condiviso (shared bus). 189
a memoria condivisa (shared-memory). 189
auto-instradanti, 187-189
banyane, 190-192
crossbar, 189
definizione, 184-186
mazzo di chiavi (key ring). 530
MBone, 235, 588, 589-599
MD5 (Message Digest 5). 507-519
checksum, 524-525
con chiave (keyed). 524-525
con firma RSA. 525
vedi anche algoritmi crittografici
memoria
ampiezza di banda. 125-127, 184-185
cache sul chip, 125
collo di bottiglia, 125-127
del nodo, 57
di sistema, 126-127
latenza, 57

messaggi
ARP. 258-259
autenticazione, 251
coda (trailer), 20
corpo (body), 20
DHCP, 230
di alto livello, 20-21
di basso livello, 20-21
elaborazione di, 498
esempio di struttura dati per. 35
formato BLAST, 359-361
formato CHAN, 363-364
instradamento, 251
intestazione (header). 20
OSPF, 251
PATH, 441-442
RESV, 441-442
SRR, 356
testo cifrato (ciphertext). 505
testo in chiaro (plaintext). 505
messaggio, formato di: *vedi* formato di messaggio
messaggio, integrità di: *vedi* integrità di messaggio
MG (media gateway). 498-499
MIB (management information base). 576-577
MIC (message integrity code). 524
microprotocollo. 355
middleware, 554
MIME (Multipurpose Internet Mail Extensions). 563-566
modelli di processo. 32-34
modelli di servizio
basati sulla QoS. 398, 430
best-effort, 398
definizione, 398
IP, 206-207
QoS multiple, 396
modem via cavo. 62-63
modulazione, 58-59
MP3, 496-498
MPEG (Moving Picture Experts Group). 488-496
MPLS (Multiprotocol Label Switching). 194, 204, 296-308, 488-496
MSAU (multistation access unit). 105
MSL (maximum segment lifetime). 330
raccomandato, 342
MSS (maximum segment size). 408-409
MTU (maximum transmission unit). 210
multicast, 288-296
alberi, 600-601
definizione, 8
estensione, 165
flusso video, 495-496
gruppi, 288-289
indirizzi, 100
motivazioni. 288, 290

prenotazione mediante alberi, 441-442
 RLM (receiver-driven layered multicast), 496
 RPM (reverse-path multicast), 292
 stato delle linee, 289
 supporto, 165
 supporto al flusso, 440-442
 supporto in IPv6, 288-289
 tra sistemi terminali, 600-603
 vettore di distanza, 290-293

multihomed, 271
 multiplexing, 8
 FDM (frequency-division multiplexing), 9
 PDU, 174
 statistico, 10-12
 STDM (synchronous time-division multiplexing), 9-11

N
 Nagle, algoritmo, 345-346
 NAK (negative acknowledgement), 88
 name server: *vedi* server per i nomi
 Napster (rete peer-to-peer), 605
 nascosti, nodi, 115
 NAT (network address translation), 287
 NDR (Network Data Representation), 374, 476-477
 next hop, 220
 NFS (Network File System), 372
 NIC (Network Information Center), 554
 NIST (National Institute for Standards and Technology), 524
 NNI (formato di cella ATM), 172
 nodi
 a livello di linea, 222
 a livello internet, 222
 a livello rete, 222
 aggiunti ad una rete sovrapposta, 611-612
 definizione, 5
 esposti, 116
 indirizzi, 8
 insieme delle foglie, 610-613
 memoria, 57
 mobilità, 117
 nascosti, 115
 nell'instradamento a stato delle linee, 245-246
 pesi dei rami, 256
 rilevazione di guasto di linea, 241-242
 tipi di, 56-57
 noleggio, linee, 60-61
 nomi di dominio: *vedi* DNS
 nomi, server per i: *vedi* server per i nomi
 nomi, traduzione di: *vedi* traduzione di nomi
 non in tempo reale, applicazioni, 430
 NRZ (non-return to zero), 64-66
 SONET, 73
 vedi anche codifica
 NRZI (non-return to zero inverted), 66-67
 vedi anche codifica

NSAP (network service access point), 168
 NSFNET, 261

O
 OSF (Open Software Foundation), 374
 OSI (Open Systems Interconnection), 22-23
 OSPF (Open Shortest Path First), 250-253, 308
 vedi anche instradamento a stato delle linee

P
 pacchetti, 10
 classificazione, 443-444
 contesa, 391
 di dimensione minima, 169
 eliminazione, 419-424
 formato AAL3/4, 174
 formato DHCP, 231
 formato IP, 207-209
 formato IPv6, 284-285
 formato RIP, 245
 inoltro, 596
 LSP, 246-248
 lunghezza fissa, 169-171
 lunghezza variabile, 169-170, 222
 marcatura, 418-419
 MSL (maximum segment lifetime), 330, 342
 premium, 448
 ritrasmissione, 254
 sequenza, 10
 velocità in pps, 185-186
 parallelismo, 169
 parità bidimensionale, 75-77
 vedi anche rivelazione d'errori
 parte terza fidata (trusted third party), 521-522
 vedi anche autenticazione
 partenza lenta, 410-414
 vedi anche controllo di congestione
 PATRICIA, albero, 269
 Paxson, Vern, 257
 PCS (Personal Communication Services), 63
 PDU (protocol data unit), 174-176
 peering point, 613
 peer-to-peer, 605-613
 PEM (Privacy Enhanced Mail), 529
 percorsi
 di default, 270
 più brevi, 237
 virtuali, 178
 periodico, aggiornamento, 241
 permanente, circuito virtuale, 149
 persistenti, connessioni, 574
 persistenza, 100-101
 PGP (Pretty Good Privacy), 529-531
 vedi anche sicurezza
 PHB (per-hop behavior), 446
 pianificazione di pacchetti, 443-444
 piconet, 64

pila di protocolli, 19
 PIM (Protocol Independent Multicast), 289, 293-296
 PIO (programmed I/O), 122-124
 vedi anche adattatori di rete
 playback: *vedi* riproduzione
 polinomio divisore, 78-80
 politiche di eliminazione, 401-402
 polling, 122
 POP (Post Office Protocol), 568
 Port Mapper, 373
 porte, 185-189
 ben note (well-known), 327
 definizione, 146
 di input, 185-189
 di output, 185-189
 selezione dinamica, 542
 UDP, 372-373
 posta elettronica, 563-570
 vedi anche applicazioni
 potenza di una rete, 389-399
 POTS (plain old telephone service), 61
 PPD (partial packet discard), 424
 PPP (Point-to-Point Protocol), 69-70
 premium, 448
 prenotazione di risorse, 437
 in alberi multicast, 441-442
 presentazione: *vedi* formato di presentazione
 prestazioni, 35-44
 privata, chiave, 506
 probabilità di eliminazione, 419-421
 processori di rete, 224
 prodotto ritardo x ampiezza di banda, 40-41
 e perdita di dati, 414
 vedi anche prestazioni
 profilo AVP, 590
 propagazione, ritardo di, 35-40, 44
 prioritario, accodamento, 402-403
 protocolli, 18
 affidabili per dati grammari, 365
 a persistenza *p*, 100-101
 asincroni, 365
 di handshake, 535-537
 end-to-end, 24
 implementazione 31-35, 45
 orientati ai bit, 70-71
 sincroni, 365
 specifiche, 20
 strato della linea di collegamento, 72
 protocolli di instradamento, 223
 algoritmi distribuiti, 238
 esecuzione di, 237-238
 in internetwork, 244
 interdominio, 237
 intradominio, 237
 protocolli end-to-end, 325-381
 latenze di round-trip misurate, 378

prestazioni, 377-379
 proprietà, 325
 RPC, 353-377
 TCP, 329-353
 UDP, 326-328
 vedi anche protocolli proxy, 542-543
 locale per il Web, 615
 SIP, 592
 pseudointestazione, 328
 pubblica, chiave: *vedi* chiave pubblica punto-punto, linee di collegamento, 5
 push in TCP, 350
 PVC (permanent virtual circuit), 149

Q
 Q.2931, 450
 QoS (Quality of Service, qualità di servizio), 12, 430-454
 applicazioni in tempo reale, 431-435
 approcci, 435
 architettura, 435
 ATM, 435, 449-452
 Differentiated Services (DiffServ), 445-448
 Integrated Services (IntServ), 436-445
 modello a circuito virtuale, 153
 modello di servizio, 398, 430
 requisiti delle applicazioni, 431-435
 riassunto, 453-454
 qualità di servizio: *vedi* QoS
 quantizzazione, 485-486

R
 radice, bridge, 162-165
 raggiungibilità, 271-272
 record, 349-350
 recupero veloce (fast recovery), 416
 RED (random early detection), 418-424
 con In e Out (RIO), 446-448
 WRED, 448
 vedi anche evitare la congestione
 relay agent, 230
 requisiti
 algoritmi crittografici, 508-509
 ampiezza di banda, 35-36
 condivisione di risorse, 8-12
 connettività, 3-8
 MD5, 509
 RTP, 579-581
 rete locale: *vedi* LAN
 rete, architettura: *vedi* architettura di rete
 rete, gestione, 575-577
 rete, sicurezza: *vedi* sicurezza
 a collegamento diretto, 55-128
 a commutazione, 6
 a mezzo fisico condiviso, 180

"a pesce", 303
 ad alta velocità, 41-42
 allocazione di risorse, 393-394
 ATM, 60
 banyane, 190-192
 congestione, 393
 Ethernet, 96-103
 fornitore di servizi, 270
 implementazione software, 26-35
 IP, 204
 logiche, 205
 mobili ad hoc, 260
 orientate alla connessione, 148-153
 peer-to-peer, 605-613
 per la distribuzione di contenuti, 613-618
 potenza delle, 398-399
 prive di connessione, 146-148
 prossimità, 612-618
 rappresentazione con grafo, 237-238
 scalabili, 145, 222
 sovrapposte (overlay), 300-301, 596-618
 store-and-forward, 6-7
 token ring, 103-113
 virtuali private (VPN), 232-233
 wireless, 114-119
 X.25, 152-153
 RFC 822, 563-566
 ricezione, buffer, 339-342
 richiesta di ripetizione automatica: *vedi* ARQ
 richiesta di ritrasmissione selettiva (SRR), 356
 richiesta/risposta, 14
 ricostruzione, 211-217
 vedi anche segmentazione
 RIO (RED with In and Out), 446-448
 RIP (Routing Information Protocol), 244-245
 vedi anche instradamento a vettore
 di distanza
 ripetitori, 97-98
 vedi anche Ethernet
 ripetizione, richiesta automatica di: *vedi* ARQ
 ripristino di una sessione, 536-537
 riproduzione (playback), 431-435
 buffer di, 580
 risoluzione di nomi: *vedi* traduzione di nomi
 risorse: *vedi* allocazione di risorse
 ritardo
 di accodamento, 37, 44
 di propagazione, 35-40, 44
 esempio di distribuzione del, 433
 in rapporto al throughput, 399
 vedi anche latenza
 ritorno al valore iniziale (wraparound), 342
 ritrasmissione, 254
 adattativa, 346-349
 e latenza, 430
 richiesta selettiva (SRR), 356
 veloce, 414-416

S
 SA (security association), 537-538
 SAN (system area network), 11
 SAR (segmentation and reassembly), 173-177
 SAS (single attachment station), 110
 SBM (subnet bandwidth manager), 443
 scalabilità
 come ottenerla, 222
 compromesso ottimale, 277
 e incapsulamento, 277
 e principi gerarchici, 262
 e sottoreti, 266-267
 Internet, 262
 IntServ, 444-445
 scansione (attiva e passiva), 118
 scatter-read, 123
 scheduling di pacchetti, 443-444
 SDLC (Synchronous Data Link Control), 71
 SDP (Session Description Protocol), 588-591
 segmentazione e ricostruzione, 173-177
 vedi anche ATM
 segmenti, 332-334
 perdita di, 347
 probe, 342
 risposta, 341-342
 vedi anche TCP
 segnalazione, 149
 PVC, 149-150
 SVC, 149-150
 segnali
 codifica di dati int., 58-59
 fra componenti d' segnalazione, 65
 propagazione di, 58
 segreta, chiave: *vedi* chiave segreta
 SELECT, 355, 369-370
 e SunRPC, 372
 vedi anche RPC
 selettiva, conferma, 88, 356, 376
 selezione dinamica della porta, 542
 self-clocking, 407
 self-routing, matrici, 187-189
 sequenza diretta (direct sequence), 114
 server, 14
 di backend, 613
 esempi applicativi, 30-31
 per i nomi, 554-559, 615
 SMTP, 567-568
 SNMP, 567-568
 supplente, 613
 servizi differenziati (Differentiated Services), 445-448
 servizio a carico controllato, 436
 servizio garantito, 436
 servizio inaffidabile, 207
 servizio, interfaccia di, 18
 servizio, modello di: *vedi* modelli di servizio
 servizio, tipo di (TOS), 253

SRR (selective retransmission request), 356
 SSH (Secure Shell), 529-533
vedi anche sicurezza
 SSL (Secure Socket Layer), 533-534
 SSRC (synchronization source), 584-585
 statistico, multiplexing, 10-12
 stato
 hard, 395
 soft, 395, 440-441
 STDM (synchronous time-division multiplexing), 9-11
 stella, topologia a, 144
 stima del moto, 491
 stop-and-wait, 84-86
 vedi anche ARQ
 store-and-forward, 6-7
 stratificazione, 16-22
 strato di adattamento di ATM (AAL), 173-177
 streaming, 2-3, 578-579
 strutturate, reti sovrapposte, 596-618
 STS-N, 60
 stub, 473
 AS di tipo, 271
 compilatore per, 354, 473
 generazione di NDR, 477
 vedi anche marshalling
 stuffing
 di bit, 71
 di caratteri, 262-266
 subnetting, 262-266
 SunRPC, 372-374
 vedi anche RPC
 supernetting, 268
 supplente, server, 613
 surrogate server, 613
 SVC (switched virtual circuit), 149
 switch: *vedi* commutatore
 SYN, attacco mediante, 545

T
 tabella di hash, 608
 distribuita (DHT), 609, 612-613
 tabella di inoltro, 146, 236
 aggiornamento, 160
 all'ingresso di un tunnel, 234
 con sottoreti, 265
 e tabella di instradamento, 236
 esempio, 221
 gestita da un bridge, 159
 informazioni sulle reti direttamente connesse, 221
 router, 221
 scansione delle righe, 160
 tabella di instradamento, 146, 236
 come array bidimensionale, 611
 e tabella di inoltro, 236
 esempi, 236, 298, 611

iniziale, 239
 passi per la costruzione, 250
 TCP (Transmission Control Protocol), 24, 329-353
 algoritmo di Nagle, 345-346
 algoritmo sliding window, 329, 333, 338-343
 campionamento di RTT, 409-410
 controllo di congestione, 329, 407-416
 latenza di round-trip misurata, 378
 MSS (maximum segment size), 408-409
 riassunto, 379-380
 ritrasmissione adattativa, 346-349
 self-clocking, 407
 silly window syndrome, 344-345
 Vegas, 425-430
 vedi anche protocolli end-to-end; connessioni
 TCP/IP, 128
 tempo reale, applicazioni, 430-435
 temporizzatori (timeout), 84, 415-416
 teorema di Shannon, 63
 TFTP (Trivial File Transport Protocol), 24
 third party: *vedi* parte terza
 throughput
 dei commutatori, 186-187
 dei router, 223-224
 di sistema, 614
 e allocazione di risorse, 398-399
 effettivo, 126-127
 in funzione del traffico, 186-187
 in rapporto al ritardo, 399
 vedi anche ampiezza di banda
 THT (token holding time), 106-107
 timeout, 84, 415-416
 timestamp, 350-351
 tipi di dati, 470
 vedi anche marshalling
 TLS (Transport Layer Security), 529, 533-537
 vedi anche sicurezza
 token bucket, 438-439
 token ring, 103-113
 token, tempo di rotazione del: *vedi* TRT
 topologia a stella, 144
 TOS (type of service), 253
 traceroute, 257
 traduzione di nomi, 257, 559-642
 vedi anche DNS
 traffico
 classi DiffServ, 445
 e throughput, 186-187
 in transito, 270-271
 locale, 270-271
 modelli di, 187-189
 premium, 448
 trame: *vedi* frame
 tramatura: *vedi* framing
 transceiver, 96
 trasformata coseno discreta (DCT), 484-486

trasmissione, buffer di, 338-339
 TRT (token rotation time), 107
 TTRT (target TRT), 111-113
 trusted third party: *vedi* parte terza fidata
 TSpec, 438-442
 vedi anche flowspec
 TTTRT (target token rotation time), 111-113
 tunnel, 233-235
 fra agente residenziale e agente esterno, 259
 IPSEC, 539
 IPv6, 281
 MPLS, 304-308

U
 UBR (unspecified bit rate), 449-450
 vedi anche QoS di ATM
 UDP (User Datagram Protocol), 24, 326-328, 578
 latenze di round-trip misurate, 378
 porte, 372-373
 riassunto, 379
 RTP eseguito al di sopra di, 578-579
 throughput misurato usando, 379
 vedi anche protocolli end-to-end
 ultimo miglio, linee di collegamento, 61-63
 UNI ATM, forma o delle celle, 172
 unicast, 8, 100
 unmarshalling, 463-469
 URI (uniform resource identifier), 468-469, 589
 URL (uniform resource locator), 570-572, 592

V
 vat, 3, 578, 580
 VBR-nrt (variable bit rate – non-real-time), 449
 VBR-rt (variable bit rate – real-time), 449
 VCI (virtual circuit identifier), 148-152
 VDSL (very high rate DSL), 305-306
 velocità
 di baud, 62
 di bit, 67
 di linea, 67
 vic, 580
 video, compressione, 488-496

X, Z
 X.25, 152-153
 protocollo sliding window, 331
 X.500, 559
 X.509, 528
 XDR (External Data Representation), 473-475
 xDSL, 61-62
 XML (Extensible Markup Language), 477-479
 zone, 556-558
 vedi anche server per i nomi

