

# **Reti di calcolatori e Internet**

**Un approccio top-down**

Ottava edizione

**James F. Kurose, Keith W. Ross**

A cura di Antonio Capone e Sabrina Gaito

**MyLab** Codice per accedere  
alla piattaforma



Pearson



# **Reti di calcolatori e Internet**

**Un approccio top-down**

Ottava edizione

James F. Kurose, Keith W. Ross

A cura di Antonio Capone e Sabrina Gaito



© 2022 Pearson Italia – Milano, Torino

*Authorized translation from the English language edition, entitled COMPUTER NETWORKING, 8th edition by JAMES KUROSE; KEITH ROSS, published by Pearson Education, Inc, publishing as Pearson, Copyright © 2021.*

*All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.*

*Italian language edition published by Pearson Italia S.p.A., Copyright © 2022.*

Per i passi antologici, per le citazioni, per le riproduzioni grafiche, cartografiche e fotografiche appartenenti alla proprietà di terzi, inseriti in quest'opera, l'editore è a disposizione degli aventi diritto non potuti reperire nonché per eventuali non volute omissioni e/o errori di attribuzione nei riferimenti.

È vietata la riproduzione, anche parziale o ad uso interno didattico, con qualsiasi mezzo, non autorizzata.

Le fotocopie per uso personale del lettore possono essere effettuate nei limiti del 15% di ciascun volume dietro pagamento alla SIAE del compenso previsto dall'art. 68, commi 4 e 5, della legge 22 aprile 1941, n. 633.

Le riproduzioni effettuate per finalità di carattere professionale, economico o commerciale o comunque per uso diverso da quello personale possono essere effettuate a seguito di specifica autorizzazione rilasciata da CLEARedi, Corso di Porta Romana 108, 20122 Milano, e-mail [autorizzazioni@clearedi.org](mailto:autorizzazioni@clearedi.org) e sito web [www.clearedi.org](http://www.clearedi.org).

I nostri libri sono ecosostenibili: la carta è prodotta sostenendo il ciclo naturale e per ogni albero tagliato ne viene piantato un altro; il cellofan è realizzato con plastiche da recupero ambientale o riciclate; gli inchiostri sono naturali e atossici; i libri sono prodotti in Italia e l'impatto del trasporto è ridotto al minimo.

Curatori per l'edizione italiana: Antonio Capone e Sabrina Gaito

Traduzione: Sabrina Gaito

Redazione: Donatella Pepe

Impaginazione: Andrea Astolfi

Grafica di copertina: Simone Tartaglia

Immagine di copertina: Imaginechina Limited/Alamy Stock Photo

Stampa: Arti Grafiche Battaia – Zibido San Giacomo (MI)

Tutti i marchi citati nel testo sono di proprietà dei loro detentori.

9788891916006

Printed in Italy

8<sup>a</sup> edizione: gennaio 2022

Ristampa  
00 01 02 03 04

Anno  
22 23 24 25 26

#### **LIBBI DI TESTO E SUPPORTI DIDATTICI**

**LIBRI DI TESTO E SUPPORTI DIDATTICI**  
Il sistema di gestione per la qualità della Casa Editrice è certificato in conformità alla norma UNI EN ISO 9001:2015 per l'attività di progettazione, realizzazione e commercializzazione di: • prodotti editoriali scolastici, dizionari lessicografici, prodotti per l'editoria di varia ed università • materiali didattici multimediali off-line • corsi di formazione e specializzazioni in aula a distanza e-learning

**Member of CISQ Federation**



# Sommario

|  |             |
|--|-------------|
| <b>Prefazione all'edizione italiana</b>                                  | <b>XI</b>   |
| <b>Prefazione</b>  | <b>XIII</b> |
| <b>Pearson MyLab</b>   | <b>XXII</b> |
| <b>Capitolo 1 Reti di calcolatori e Internet</b>                         | <b>1</b>    |
| 1.1 Che cos'è Internet?  | 2           |
| 1.1.1 Gli "ingranaggi" di Internet                                       | 2           |
| 1.1.2 Descrizione dei servizi  | 5           |
| 1.1.3 Che cos'è un protocollo?   | 6           |
| 1.2 Ai confini della rete  | 8           |
| 1.2.1 Le reti di accesso   | 10          |
| 1.2.2 Mezzi trasmissivi  | 16          |
| 1.3 Il nucleo della rete   | 19          |
| 1.3.1 Comutazione di pacchetto   | 19          |
| 1.3.2 Comutazione di circuito  | 24          |
| 1.3.3 Una rete di reti   | 28          |
| 1.4 Ritardi, perdite e throughput nelle reti a commutazione di pacchetto | 32          |
| 1.4.1 Panoramica del ritardo nelle reti a commutazione di pacchetto      | 32          |
| 1.4.2 Ritardo di accodamento e perdita di pacchetti                      | 35          |
| 1.4.3 Ritardo end-to-end   | 37          |
| 1.4.4 Throughput nelle reti di calcolatori                               | 40          |
| 1.5 Livelli dei protocolli e loro modelli di servizio                    | 43          |
| 1.5.1 Architettura a livelli   | 43          |
| 1.5.2 Incapsulamento   | 48          |
| 1.6 Reti sotto attacco   | 50          |
| 1.7 Storia delle reti di calcolatori e di Internet                       | 53          |
| 1.7.1 Sviluppo della commutazione di pacchetto: 1961-1972                | 54          |
| 1.7.2 Reti proprietarie e internetworking: 1972-1980                     | 55          |
| 1.7.3 La proliferazione delle reti: 1980-1990                            | 56          |
| 1.7.4 Esplosione di Internet: gli anni '90                               | 57          |
| 1.7.5 Il nuovo millennio   | 58          |
| 1.8 Riepilogo  | 59          |
| Linee guida del testo  | 60          |

|  |            |
|--|------------|
| <b>Capitolo 2 Livello di applicazione</b>                            | <b>67</b>  |
| 2.1 Princìpi delle applicazioni di rete                              | 68         |
| 2.1.1 Architetture delle applicazioni di rete                        | 68         |
| 2.1.2 Processi comunicanti   | 71         |
| 2.1.3 Servizi di trasporto disponibili per le applicazioni           | 74         |
| 2.1.4 Servizi di trasporto offerti da Internet                       | 76         |
| 2.1.5 Protocolli a livello di applicazione                           | 79         |
| 2.1.6 Applicazioni di rete trattate in questo libro                  | 79         |
| 2.2 Web e HTTP   | 80         |
| 2.2.1 Panoramica di HTTP   | 80         |
| 2.2.2 Connessioni persistenti e non persistenti                      | 82         |
| 2.2.3 Formato dei messaggi HTTP                                      | 85         |
| 2.2.4 Interazione utente-server: i cookie                            | 89         |
| 2.2.5 Web caching  | 91         |
| 2.2.6 HTTP/2   | 96         |
| 2.3 Posta elettronica in Internet                                    | 99         |
| 2.3.1 SMTP   | 100        |
| 2.3.2 Formati dei messaggi di posta                                  | 103        |
| 2.3.3 Protocolli di accesso alla posta                               | 103        |
| 2.4 DNS: il servizio di directory di Internet                        | 105        |
| 2.4.1 Servizi forniti da DNS   | 105        |
| 2.4.2 Panoramica del funzionamento di DNS                            | 107        |
| 2.4.3 Record e messaggi DNS  | 113        |
| 2.5 Distribuzione di file P2P  | 117        |
| 2.6 Streaming video e reti per la distribuzione di contenuti         | 123        |
| 2.6.1 Video su Internet  | 124        |
| 2.6.2 Streaming HTTP e DASH  | 124        |
| 2.6.3 Reti per la distribuzione di contenuti                         | 125        |
| 2.7 Programmazione delle socket: come creare un'applicazione di rete | 129        |
| 2.7.1 Programmazione delle socket con UDP                            | 130        |
| 2.7.2 Programmazione delle socket con TCP                            | 135        |
| 2.8 Riepilogo  | 139        |
| <b>Capitolo 3 Livello di trasporto</b>                               | <b>147</b> |
| 3.1 Introduzione e servizi a livello di trasporto                    | 148        |
| 3.1.1 Relazione tra i livelli di trasporto e di rete                 | 148        |
| 3.1.2 Panoramica del livello di trasporto di Internet                | 151        |
| 3.2 Multiplexing e demultiplexing                                    | 152        |
| 3.3 Trasporto non orientato alla connessione: UDP                    | 159        |
| 3.3.1 Struttura dei segmenti UDP                                     | 162        |
| 3.3.2 Checksum UDP   | 163        |
| 3.4 Princìpi del trasferimento dati affidabile                       | 164        |
| 3.4.1 Costruzione di un protocollo di trasferimento dati affidabile  | 166        |

---

|                   |   |            |
|-------------------|---|------------|
| 3.4.2             | Protocolli per il trasferimento dati affidabile con pipeline  | 175        |
| 3.4.3             | Go-Back-N (GBN)   | 178        |
| 3.4.4             | Ripetizione selettiva   | 183        |
| 3.5               | Trasporto orientato alla connessione: TCP   | 188        |
| 3.5.1             | Connessione TCP   | 188        |
| 3.5.2             | Struttura dei segmenti TCP  | 190        |
| 3.5.3             | Timeout e stima del tempo di andata e ritorno (RTT)   | 196        |
| 3.5.4             | Trasferimento dati affidabile   | 198        |
| 3.5.5             | Controllo di flusso   | 206        |
| 3.5.6             | Gestione della connessione TCP  | 208        |
| 3.6               | Principi del controllo di congestione   | 214        |
| 3.6.1             | Cause e costi della congestione   | 214        |
| 3.6.2             | Approcci al controllo di congestione  | 220        |
| 3.7               | Controllo di congestione TCP  | 221        |
| 3.7.1             | Controllo di congestione di TCP classico  | 221        |
| 3.7.2             | Notifica esplicita di congestione assistita dalla rete<br>e controllo di congestione basato sul ritardo | 231        |
| 3.7.3             | Fairness  | 234        |
| 3.8               | Evoluzione della funzionalità del livello di trasporto  | 236        |
| 3.9               | Riepilogo   | 239        |
| <b>Capitolo 4</b> | <b>Livello di rete: il piano dei dati</b>   | <b>251</b> |
| 4.1               | Panoramica del livello di rete  | 252        |
| 4.1.1             | Inoltro e instradamento: piano dei dati e piano di controllo  | 253        |
| 4.1.2             | Modelli di servizio del livello di rete   | 256        |
| 4.2               | Che cosa si trova all'interno di un router?   | 258        |
| 4.2.1             | Elaborazione alle porte di ingresso e inoltro basato sull'indirizzo<br>di destinazione                  | 260        |
| 4.2.2             | Struttura di commutazione (switching)   | 263        |
| 4.2.3             | Elaborazione alle porte di uscita   | 265        |
| 4.2.4             | Dove si verifica l'accodamento?   | 265        |
| 4.2.5             | Schedulazione dei pacchetti   | 270        |
| 4.3               | Il Protocollo Internet (IP): IPv4, indirizzamento, IPv6 e altro ancora                                  | 275        |
| 4.3.1             | Formato dei datagrammi IPv4   | 275        |
| 4.3.2             | Indirizzamento IPv4   | 278        |
| 4.3.3             | NAT (Network Address Translation)   | 287        |
| 4.3.4             | IPv6  | 289        |
| 4.4               | Inoltro generalizzato e SDN   | 294        |
| 4.4.1             | Match   | 296        |
| 4.4.2             | Action  | 298        |
| 4.4.3             | Esempi del paradigma match-action in OpenFlow   | 298        |
| 4.5               | I dispositivi middlebox   | 300        |
| 4.6               | Riepilogo   | 303        |

|   |            |
|---|------------|
| <b>Capitolo 5 Livello di rete: il piano di controllo</b>  | <b>309</b> |
| 5.1 Introduzione  | 309        |
| 5.2 Algoritmi di instradamento  | 312        |
| 5.2.1 Instradamento “link-state” (LS)   | 314        |
| 5.2.2 Instradamento “distance-vector” (DV)  | 318        |
| 5.3 Instradamento interno ai sistemi autonomi: OSPF   | 325        |
| 5.4 Instradamento tra ISP: BGP  | 329        |
| 5.4.1 Il ruolo di BGP   | 329        |
| 5.4.2 Distribuzione delle informazioni dei percorsi in BGP  | 329        |
| 5.4.3 Selezione delle rotte migliori  | 331        |
| 5.4.4 Anycast IP  | 334        |
| 5.4.5 Politiche di instradamento  | 335        |
| 5.4.6 Retrospettiva: come essere presenti in Internet   | 336        |
| 5.5 Il piano di controllo SDN   | 338        |
| 5.5.1 Il piano di controllo SDN: controller SDN e applicazioni di controllo                           | 340        |
| 5.5.2 Il protocollo OpenFlow  | 342        |
| 5.5.3 Interazione tra piano dei dati e piano di controllo: un esempio                                 | 344        |
| 5.5.4 SDN: il passato e il futuro   | 345        |
| 5.6 ICMP (Internet Control Message Protocol)  | 348        |
| 5.7 Gestione della rete e SNMP, NETCONF/YANG  | 350        |
| 5.7.1 Infrastruttura di gestione  | 350        |
| 5.7.2 SNMP ( <i>Simple Network Management Protocol</i> ) e MIB ( <i>Management Information Base</i> ) | 353        |
| 5.7.3 NETCONF e YANG  | 356        |
| 5.8 Riepilogo   | 359        |
| <br><b>Capitolo 6 Livello di collegamento e reti locali</b>   | <b>365</b> |
| 6.1 Introduzione al livello di collegamento   | 366        |
| 6.1.1 Servizi offerti dal livello di collegamento   | 367        |
| 6.1.2 Dov’è implementato il livello di collegamento?  | 368        |
| 6.2 Tecniche di rilevazione e correzione degli errori   | 370        |
| 6.2.1 Controllo di parità   | 371        |
| 6.2.2 Checksum  | 373        |
| 6.2.3 Controllo a ridondanza ciclica (CRC)  | 373        |
| 6.3 Collegamenti e protocolli di accesso multiplo   | 375        |
| 6.3.1 Protocolli a suddivisione del canale  | 377        |
| 6.3.2 Protocolli ad accesso casuale   | 379        |
| 6.3.3 Protocolli a rotazione  | 387        |
| 6.3.4 DOCSIS: il protocollo a livello di collegamento per reti di accesso a Internet HFC              | 388        |
| 6.4 Reti locali commutate   | 389        |
| 6.4.1 Indirizzi a livello di collegamento e ARP   | 390        |
| 6.4.2 Ethernet  | 396        |
| 6.4.3 Switch a livello di collegamento  | 401        |
| 6.4.4 LAN virtuali (VLAN)   | 407        |

|       |  |     |
|-------|--|-----|
| 6.5   | Canali virtuali: una rete come livello di collegamento             | 410 |
| 6.5.1 | Multiprotocol Label Switching (MPLS)                               | 411 |
| 6.6   | Le reti dei data center  | 414 |
| 6.6.1 | Architetture dei data center                                       | 414 |
| 6.6.2 | Tendenze nel networking dei data center                            | 418 |
| 6.7   | Retrospettiva: cronaca di una richiesta di una pagina web          | 421 |
| 6.7.1 | Si comincia: DHCP, UDP, IP ed Ethernet                             | 422 |
| 6.7.2 | Siamo ancora all'inizio: DNS e ARP                                 | 423 |
| 6.7.3 | Siamo ancora all'inizio: instradamento intra-dominio al server DNS | 424 |
| 6.7.4 | Interazione client-server: TCP e HTTP                              | 425 |
| 6.8   | Riepilogo  | 426 |

## **Chapter 7 Wireless and Mobile Networks**

|       |   |
|-------|---|
| 7.1   | Introduction  |
| 7.2   | Wireless Links and Network Characteristics                        |
| 7.2.1 | CDMA  |
| 7.3   | WiFi: 802.11 Wireless LANs  |
| 7.3.1 | The 802.11 Wireless LAN Architecture                              |
| 7.3.2 | The 802.11 MAC Protocol   |
| 7.3.3 | The IEEE 802.11 Frame   |
| 7.3.4 | Mobility in the Same IP Subnet                                    |
| 7.3.5 | Advanced Features in 802.11                                       |
| 7.3.6 | Personal Area Networks: Bluetooth                                 |
| 7.4   | Cellular Networks: 4G and 5G                                      |
| 7.4.1 | 4G LTE Cellular Networks: Architecture and Elements               |
| 7.4.2 | LTE Protocols Stacks  |
| 7.4.3 | LTE Radio Access Network  |
| 7.4.4 | Additional LTE Functions: Network Attachment and Power Management |
| 7.4.5 | The Global Cellular Network: A Network of Networks                |
| 7.4.6 | 5G Cellular Networks  |
| 7.5   | Mobility Management: Principles                                   |
| 7.5.1 | Device Mobility: a Network-layer Perspective                      |
| 7.5.2 | Home Networks and Roaming on Visited Networks                     |
| 7.5.3 | Direct and Indirect Routing to/from a Mobile Device               |
| 7.6   | Mobility Management in Practice                                   |
| 7.6.1 | Mobility Management in 4G/5G Networks                             |
| 7.6.2 | Mobile IP   |
| 7.7   | Wireless and Mobility: Impact on Higher-Layer Protocols           |
| 7.8   | Summary   |
|       | Homework Problems and Questions                                   |
|       | Wireshark Lab: WiFi   |
|       | Interview: Deborah Estrin   |





## Chapter 8 Security in Computer Networks

- 8.1 What Is Network Security?
  - 8.2 Principles of Cryptography
    - 8.2.1 Symmetric Key Cryptography
    - 8.2.2 Public Key Encryption
  - 8.3 Message Integrity and Digital Signatures
    - 8.3.1 Cryptographic Hash Functions
    - 8.3.2 Message Authentication Code
    - 8.3.3 Digital Signatures
  - 8.4 End-Point Authentication
  - 8.5 Securing E-Mail
    - 8.5.1 Secure E-Mail
    - 8.5.2 PGP
  - 8.6 Securing TCP Connections: TLS
    - 8.6.1 The Big Picture
    - 8.6.2 A More Complete Picture
  - 8.7 Network-Layer Security: IPsec and Virtual Private Networks
    - 8.7.1 IPsec and Virtual Private Networks (VPNs)
    - 8.7.2 The AH and ESP Protocols
    - 8.7.3 Security Associations
    - 8.7.4 The IPsec Datagram
    - 8.7.5 IKE: Key Management in IPsec
  - 8.8 Securing Wireless LANs and 4G/5G Cellular Networks
    - 8.8.1 Authentication and Key Agreement in 802.11 Wireless LANs
    - 8.8.2 Authentication and Key Agreement in 4G/5G Cellular Networks
  - 8.9 Operational Security: Firewalls and Intrusion Detection Systems
    - 8.9.1 Firewalls
    - 8.9.2 Intrusion Detection Systems
  - 8.10 Summary
- Homework Problems and Questions  
Wireshark Lab: SSL  
IPsec Lab  
Interview: Steven M. Bellovin

## Capitolo 9 Reti multimediali

- 9.1 Applicazioni multimediali di rete
  - 9.2 Streaming di video registrato
  - 9.3 Voice-over-IP
  - 9.4 Protocolli per applicazioni in tempo reale
  - 9.5 Supporto di Internet alle applicazioni multimediali
  - 9.6 Riepilogo
- Domande e problemi  
Esercizi di programmazione  
Intervista a Henning Schulzrinne



# Prefazione all’edizione italiana

Sono ormai passati cinque anni dall’ultima edizione italiana di *Reti di calcolatori e Internet. Un approccio top-down* di James F. Kurose e Keith W. Ross. In questo intervallo temporale il mondo delle reti, e Internet con esso, è cambiato profondamente, portando innovazione a tutti i livelli e diventando pervasivo nella nostra vita quotidiana. Nonostante la completezza e complessità di questo testo, oggi il più adottato nelle università italiane, gli Autori sono riusciti a cogliere tutti questi cambiamenti, aggiornandolo in maniera innovativa, completa e organica. In questa nuova edizione, infatti, gli Autori propongono una profonda riorganizzazione della struttura del testo che riflette il cambio di paradigma che sta affrontando il mondo delle reti e di Internet con l’avvento delle *software-defined networking* (SDN). L’ampliamento della trattazione sul livello di rete permette inoltre una comprensione profonda del tema cardine delle reti di calcolatori e degli importanti cambiamenti globali che stanno trasformando il *networking*.

Inoltre, da un punto di vista didattico, ma come oggi risulta significativo e strategico l’approccio top-down, introdotto dagli Autori nel 1994 e che oramai è diventato il segno distintivo della loro opera. Iniziare dal livello applicativo per poi scendere verso gli strati bassi della gerarchia di protocolli ha l’indubbio pregio di stimolare fortemente gli studenti nelle fasi iniziali, oltre che permettere di motivare in maniera chiara e comprensibile lo studio successivo dei livelli sottostanti. L’edizione italiana, in particolare, cerca di dare un contributo agli studenti affinché siano pronti a inserirsi nella dimensione ormai internazionale assunta dall’argomento. È cosa nota, infatti, che il vocabolario di tecnici e operatori del settore sia un connubio di italiano e inglese a seconda non solo dell’abitudine, ma anche della disponibilità di espressioni sintetiche nell’una o nell’altra lingua. Per questo motivo si è dedicata una cura particolare al linguaggio tecnico, fornendo per ogni termine, ovunque possibile, la doppia dicitura italiana e inglese e cercando di prediligere nelle parti discorsive la versione più comunemente adottata in ambito professionale.

A parere dei curatori con questa Ottava Edizione gli Autori hanno offerto a docenti, studenti e professionisti un testo sulle reti e su Internet prezioso e unico per affrontare i grandi cambiamenti e sfide del *networking* attuale.

*Sabrina Gaito  
Università degli Studi di Milano*

*Antonio Capone  
Politecnico di Milano*



# Prefazione

*A Julie e ai nostri tre preziosi Chris, Charlie, Nina  
JFK*

*Un sentito ringraziamento ai miei professori,  
ai colleghi e agli studenti di tutto il mondo  
KWR*

Benvenuti alla ottava edizione di *Computer Networking: A Top-Down Approach*. Nei vent'anni intercorsi dalla pubblicazione della prima edizione il testo è stato adottato in centinaia di istituti e università, tradotto in 14 lingue e utilizzato da centinaia di migliaia di studenti e professionisti in tutto il mondo. Abbiamo ricevuto commenti estremamente positivi da moltissimi di questi lettori.

## Le novità introdotte nella ottava edizione

Riteniamo che una ragione importante del successo di questo libro risieda nel fatto che esso continua a offrire un approccio nuovo e aggiornato all'insegnamento delle reti di calcolatori. Anche se in questa ottava edizione abbiamo introdotto delle modifiche, sono stati tuttavia mantenuti invariati quelli che crediamo (e docenti e studenti che hanno utilizzato il nostro libro hanno confermato) essere gli aspetti peculiari di questo libro: l'approccio top-down, l'attenzione a Internet, una trattazione moderna delle reti di calcolatori, la considerazione sia degli aspetti teorici sia di quelli pratici, nonché uno stile e un approccio accessibili rivolti all'apprendimento delle reti di calcolatori. L'ottava edizione è stata ampiamente rivista e aggiornata.

I nostri lettori di lungo corso si ricorderanno che, nel passaggio dalla sesta alla settima edizione, abbiamo approfondito il livello di rete, espandendone la trattazione da un singolo capitolo a due capitoli: il Capitolo 4 (che si concentra sulla componente "piano dei dati" del livello di rete) e il Capitolo 5 (che si concentra sul "piano di controllo" del livello di rete). Questo ampliamento della trattazione del livello di rete riflette la cresciuta importanza del *software-defined networking* (SDN), probabilmente la più importante ed emozionante innovazione delle reti negli ultimi decenni. Anche se relativamente recente, SDN è stato rapidamente adottato in pratica, tanto che è già difficile immaginare un'introduzione moderna al computer networking che non lo tratti. Il tema della gestione della rete con SDN viene discusso in dettaglio nella presente edizione. In questa ottava edizione viene inoltre dedicato ampio spazio alle reti 4G/5G, indubbiamente una delle maggiori novità nell'ambito del networking.

I cambiamenti più rilevanti sono elencati qui di seguito.

- Il Capitolo 1 è stato aggiornato per riflettere la crescente diffusione di Internet e delle reti 4G/5G.
- Il Capitolo 2, che tratta il livello di applicazione, è stato significativamente aggiornato e arricchito con la trattazione dei protocolli per il Web HTTP/2 e HTTP/3.
- Il Capitolo 3 è stato aggiornato per riflettere i progressi e l’evoluzione avvenuti negli ultimi cinque anni nell’uso dei protocolli di controllo della congestione e di controllo degli errori a livello di trasporto. Sebbene tale argomento fosse rimasto relativamente stabile per un po’ di tempo, sono stati fatti importanti progressi rispetto alla settima edizione. Sono stati sviluppati e implementati alcuni nuovi algoritmi di controllo della congestione oltre agli algoritmi di TCP “classico”. Viene fornita una trattazione più approfondita di TCP CUBIC, il protocollo TCP predefinito in molti sistemi implementati, e vengono esaminati gli approcci al controllo della congestione basati sul ritardo, incluso il nuovo protocollo BBR, implementato nella rete dorsale di Google. Oggetto di studio è anche il protocollo QUIC, incorporato nello standard HTTP/3. Il protocollo QUIC, sebbene tecnicamente non sia un protocollo a livello di trasporto in quanto fornisce affidabilità, controllo della congestione e servizi di multiplexing al livello di applicazione, utilizza molti dei principi del controllo degli errori e della congestione sviluppati nei primi paragrafi del Capitolo 3.
- Il Capitolo 4, che copre il piano dei dati a livello di rete, contiene aggiornamenti generali in tutto il capitolo. È stato aggiunto un nuovo paragrafo sui cosiddetti middlebox, che svolgono funzioni a livello di rete diverse dal routing e dall’inoltro, come il firewallsing e il bilanciamento del carico. I middlebox si basano naturalmente sull’operazione di inoltro generalizzato “match plus action” dei dispositivi a livello di rete, trattati in precedenza in questo capitolo. Nuovo materiale è stato aggiunto su argomenti come la “giusta” grandezza del buffering nei router di rete, la neutralità della rete e i principi architetturali di Internet.
- Il Capitolo 5, che tratta il piano di controllo del livello di rete, contiene materiale aggiornato su SDN e una trattazione significativamente nuova della gestione della rete. L’uso di SDN si è evoluto oltre la gestione delle tabelle di inoltro dei pacchetti per includere anche la gestione della configurazione dei dispositivi di rete. Vengono presentati due nuovi protocolli, NETCONF e YANG, la cui adozione e utilizzo hanno alimentato questo nuovo approccio alla gestione della rete.
- Il Capitolo 6, che tratta il livello di collegamento, è stato aggiornato per riflettere la continua evoluzione delle tecnologie di livello di collegamento come Ethernet. È stata anche aggiornata e ampliata la trattazione delle reti dei data center, che sono al centro della tecnologia che guida gran parte del commercio sulla Internet di oggi.
- Il Capitolo 7, disponibile in lingua inglese sulla piattaforma MyLab relativa a questo libro, è stato significativamente aggiornato e rivisto per riflettere i numerosi cambiamenti nelle reti wireless dalla settima edizione, dalle piconet

Bluetooth a corto raggio alle reti locali 802.11 wireless (WLAN), alle reti cellulari wireless wide area 4G/5G. La trattazione delle precedenti reti 2G e 3G è stata rimossa a favore di una discussione più ampia e approfondita delle reti 4G LTE di oggi e delle reti 5G di domani. È stata aggiornata la trattazione dei problemi di mobilità, dal problema locale dell'handover dei dispositivi mobili tra base station al problema globale della gestione dell'identità e del roaming dei dispositivi mobili tra diverse reti cellulari globali.

- Il Capitolo 8, disponibile in lingua inglese sulla piattaforma MyLab relativa a questo libro, riguarda la sicurezza della rete. È stato aggiornato per riflettere i cambiamenti avvenuti in particolare nella sicurezza delle reti wireless.
- Il Capitolo 9, sulle reti multimediali, è stato rimosso da questa edizione. È disponibile sulla piattaforma MyLab di questo libro nella versione in italiano della settima edizione.
- I problemi di fine capitolo sono stati aggiornati alla luce dei nuovi argomenti trattati. Inoltre sul sito [http://gaia.cs.umass.edu/kurose\\_ross/interactive](http://gaia.cs.umass.edu/kurose_ross/interactive), curato dagli autori, si possono trovare esercizi interattivi per creare problemi (e le relative soluzioni) simili ad alcuni di fine capitolo. Gli studenti, creando un numero illimitato di problemi simili con le relative soluzioni, possono effettivamente diventare padroni della materia.

## I lettori

Il libro si rivolge a un primo corso di reti di calcolatori e può essere utilizzato sia nei dipartimenti di informatica sia in quelli di ingegneria informatica ed elettronica. In termini di linguaggi di programmazione si presuppone solo che gli studenti abbiano dimestichezza con C, C++, Java o Python (e solo in pochi punti del volume). Inoltre il testo, pur essendo più preciso e analitico di molti altri sull'argomento, utilizza raramente concetti matematici che non siano insegnati nelle scuole superiori. Abbiamo volutamente cercato di evitare calcoli avanzati e concetti di probabilità o processi stocastici, sebbene abbiamo incluso alcuni problemi che richiedono questo tipo di conoscenza. Il libro è pertanto appropriato per i corsi di laurea triennali e specialistici, ma si rivela utile anche per i professionisti dell'industria del networking.

## Che cosa c'è di unico in questo libro?

L'argomento delle reti di calcolatori è decisamente complesso, dato che comprende concetti, protocolli e tecnologie interconnessi tra loro in maniera intricata. Per affrontare tale complessità, l'organizzazione di molti testi sull'argomento segue la struttura stratificata dell'architettura di rete. In tal modo, gli studenti percepiscono da subito la complessità della materia e imparano i differenti concetti e protocolli propri di ciascun livello dell'architettura, senza perdere di vista l'intero quadro in cui tutte le parti s'incastrano l'una con l'altra. Dalla nostra esperienza tale approccio è di certo consigliabile da un punto di vista metodologico; tuttavia, abbiamo riscontrato che il tradizionale insegnamento *bottom-up* (ossia dal livello fisico verso quello di applicazione) non rappresenta la scelta migliore per un corso moderno sulle reti di calcolatori.

## Un approccio dall'alto verso il basso

Venti anni fa questo libro si avventurò in un territorio inesplorato trattando il networking in modo top-down, ossia iniziando dal livello di applicazione per scendere verso quello fisico. Il riscontro avuto da docenti e studenti ha confermato che tale approccio presenta numerosi vantaggi e funziona bene a livello didattico. In primo luogo enfatizza il livello di applicazione, un'area in forte crescita nelle reti, nella quale si sono verificate molte delle recenti rivoluzioni di questi ultimi anni, tra cui il Web e lo streaming multimediale. Dare subito importanza agli argomenti legati a questo livello rappresenta una differenza notevole rispetto all'impostazione dei testi che forniscono solo limitate informazioni sulle applicazioni di rete, sui loro requisiti, sui paradigmi a livello applicativo (come client-server e peer-to-peer) e sulle interfacce di programmazione delle applicazioni. In secondo luogo, la nostra esperienza didattica (e quella di molti docenti che hanno usato il nostro testo) mostra che affrontare le applicazioni di rete all'inizio del corso costituisce un potente strumento motivazionale. Gli studenti sono stimolati dall'idea di imparare il funzionamento di applicazioni quali l'e-mail, lo streaming video e il Web, che utilizzano quotidianamente. Dopo aver compreso le applicazioni è più facile capire i servizi di rete necessari al loro supporto ed è poi possibile esaminare i vari modi in cui tali servizi sono implementati nei livelli inferiori.

Inoltre, l'approccio top-down consente ai docenti di introdurre lo sviluppo delle applicazioni di rete in una fase iniziale. Gli studenti non soltanto vedono le modalità operative di applicazioni e protocolli comuni, ma imparano anche quanto sia facile crearne di nuovi.

Con un approccio top-down gli studenti affrontano presto le nozioni sulla programmazione delle socket, sui modelli di servizio e sui protocolli: concetti importanti che riemergeranno in tutti i livelli successivi. Fornendo esempi di programmazione delle socket in Python si evidenziano le idee centrali senza confondere gli studenti con codice complesso. Gli studenti della laurea triennale non dovrebbero avere difficoltà nell'interpretare codice Python.

## Internet in risalto

In questo libro si continuano a usare l'architettura e i protocolli di Internet come filo conduttore per studiare i concetti fondamentali delle reti di calcolatori.

Ovviamente, sono inclusi anche concetti e protocolli tratti da altre architetture di rete, ma l'attenzione è chiaramente puntata su Internet. Infatti, l'organizzazione del libro ruota intorno all'architettura a cinque livelli propria di Internet: applicazione, trasporto, rete, collegamento e fisico.

Inoltre, la maggior parte degli studenti nutre uno specifico interesse riguardo a Internet e ai suoi protocolli. Sono coscienti che rappresenti una tecnologia rivoluzionaria e dirompente, che sta radicalmente trasformando il nostro mondo. Sono quindi naturalmente curiosi di conoscerne i molteplici aspetti. Pertanto, usando Internet come guida, è facile per il docente suscitare negli studenti l'interesse per i suoi principi di base.

## Insegnare i principi delle reti

Alle due caratteristiche peculiari di questo libro – l'approccio top-down e l'attenzione a Internet – occorre aggiungerne una terza, riassumibile nel termine “principi”. Il campo del networking è ora sufficientemente maturo da identificare un certo numero di argomenti di fondamentale importanza. Per esempio, nel livello di trasporto le questioni fondamentali includono l'affidabilità della comunicazione su un livello di rete non affidabile, l'inizio e la fine delle connessioni e l'handshaking, il controllo di congestione e di flusso e, infine, il multiplexing.

Due argomenti di fondamentale importanza sul livello di rete sono la determinazione di percorsi “buoni” tra due router e l'interconnessione di reti eterogenee. Nel livello di collegamento un problema fondamentale è la condivisione del canale. Per quanto riguarda la sicurezza di rete, le tecniche per fornire riservatezza, autenticazione e integrità dei messaggi sono tutte basate su principi di crittografia. Questo testo identifica gli argomenti fondamentali del networking e ne tratta i diversi approcci. Apprendendo tali principi, lo studente acquisirà conoscenze “a lungo termine” e quando gli attuali standard e protocolli di rete saranno divenuti obsoleti, i principi conserveranno la loro rilevanza.

Noi crediamo che la combinazione dell'uso di Internet con l'enfasi riguardo ad argomenti fondamentali e approcci risolutivi consenta di comprendere rapidamente ogni tecnologia di rete.

## Caratteristiche didattiche

Abbiamo entrambi tenuto corsi di reti di calcolatori per oltre trent'anni e in questo libro abbiamo riversato più di sessant'anni di esperienza condivisa con molte migliaia di studenti. In questo lasso di tempo siamo anche stati ricercatori attivi su questi argomenti (ci siamo infatti incontrati per la prima volta a un master sulle reti tenuto da Mischa Schwartz nel 1979, presso la Columbia University). Riteniamo che tutto ciò ci abbia dato una visione corretta su quelle che sono state le reti e in quale direzione stanno andando. Abbiamo tuttavia resistito alla tentazione di indirizzare il materiale del libro verso i nostri progetti personali, descritti sul nostro sito web. Il testo riguarda le reti moderne e tratta i protocolli e le tecnologie così come i sottostanti principi. Crediamo anche che l'apprendimento (e l'insegnamento) della materia possano essere divertenti. Un certo senso dell'umorismo, unito all'uso di analogie ed esempi tratti dal mondo reale presenti nel testo renderanno, si spera, questo libro più stimolante.

## Dipendenze tra capitoli

Il primo capitolo presenta una panoramica autonoma sulle reti di calcolatori e introduce molti concetti chiave. La nostra raccomandazione è che, dopo il primo, i docenti trattino i Capitoli dal 2 al 6 in sequenza, presentando gli argomenti secondo la filosofia top-down. Ciascun capitolo si basa infatti su argomenti trattati in quelli precedenti. Dopo i primi sei capitoli il docente può procedere con maggiore flessibilità. Non esistono, infatti, interdipendenze tra gli ultimi capitoli, che possono perciò essere affrontati in qualsiasi ordine. In ogni caso, ciascuno di essi dipende dai contenuti presentati nei primi sei capitoli.

## Infine, saremo lieti di avere vostre notizie

Noi incoraggiamo docenti e studenti a inviarci via e-mail i loro commenti. È stato fantastico ricevere così tanti commenti da docenti e studenti di tutto il mondo riguardo le precedenti edizioni; molti dei loro suggerimenti sono stati inclusi in questa edizione. Incoraggiamo anche i docenti a inviarci le loro esercitazioni (con relative soluzioni) che costituiscano un complemento a quelle attualmente proposte. Le pubblicheremo nella parte riservata ai docenti del sito web. Incoraggiamo inoltre docenti e studenti a creare nuove applet Java che illustrino i concetti e i protocolli del libro. Se avete realizzato un'animazione che ritenete appropriata per il testo, per favore inviatela agli autori. Se l'animazione (inclusa la terminologia e la notazione) è adeguata, saremo lieti di includerla nel sito web del testo, con appropriato riferimento ai suoi autori.

Saremmo lieti se i lettori proseguissero questa corrispondenza, mandandoci URL di particolare interesse, segnalandoci errori tipografici, dissentendo da qualsiasi nostra affermazione e comunicandoci che cosa funziona e che cosa no, magari suggerendoci ciò che ritenete dovrebbe essere o non essere incluso nella prossima edizione. Gli indirizzi sono: kurose@cs.umass.edu e keithross@nyu.edu.

## Ringraziamenti

Fin da quando abbiamo cominciato a scrivere questo libro, nel 1996, molte persone ci hanno fornito un prezioso aiuto e ci hanno suggerito come organizzare al meglio e come tenere un corso di reti: vogliamo inviare un particolare ringraziamento a tutti quelli che ci hanno aiutato dalla prima stesura fino alla ottava edizione. Siamo anche molto grati alle migliaia di lettori – studenti, docenti, professionisti – che, da tutto il mondo, ci hanno fatto pervenire considerazioni e commenti. Un ringraziamento particolare va a:

Al Aho (Columbia University)  
Hisham Al-Mubaid (University of Houston-Clear Lake)  
Pratima Akkunoor (Arizona State University)  
Paul Amer (University of Delaware)  
Shamiul Azom (Arizona State University)  
Lichun Bao (University of California at Irvine)  
Paul Barford (University of Wisconsin)  
Bobby Bhattacharjee (University of Maryland)  
Steven Bellovin (Columbia University)  
Pravin Bhagwat (Wibhu)  
Supratik Bhattacharyya (previously at Sprint)  
Ernst Biersack (Eurécom Institute)  
Shahid Bokhari (University of Engineering & Technology, Lahore)  
Jean Bolot (Technicolor Research)  
Daniel Brushteyn (former University of Pennsylvania student)  
Ken Calvert (University of Kentucky)  
Evandro Cantu (Federal University of Santa Catarina)  
Jeff Case (SNMP Research International)  
Jeff Chaltas (Sprint)  
Vinton Cerf (Google)

Byung Kyu Choi (Michigan Technological University)  
Bram Cohen (BitTorrent, Inc.)  
Constantine Coutras (Pace University)  
John Daigle (University of Mississippi)  
Edmundo A. de Souza e Silva (Federal University of Rio de Janeiro)  
Philippe Decuetos (Eurécom Institute)  
Christophe Diot (Technicolor Research)  
Prithula Dhunghel (Akamai)  
Deborah Estrin (University of California, Los Angeles)  
Michalis Faloutsos (University of California at Riverside)  
Wu-chi Feng (Oregon Graduate Institute)  
Sally Floyd (ICIR, University of California at Berkeley)  
Paul Francis (Max Planck Institute)  
David Fullager (Netflix)  
Lixin Gao (University of Massachusetts)  
JJ Garcia-Luna-Aceves (University of California at Santa Cruz)  
Mario Gerla (University of California at Los Angeles)  
David Goodman (NYU-Poly)  
Yang Guo (Alcatel/Lucent Bell Labs)  
Tim Griffin (Cambridge University)  
Max Hailperin (Gustavus Adolphus College)  
Bruce Harvey (Florida A&M University, Florida State University)  
Carl Hauser (Washington State University)  
Rachelle Heller (George Washington University)  
Phillipp Hoschka (INRIA/W3C)  
Wen Hsin (Park University)  
Albert Huang (former University of Pennsylvania student)  
Cheng Huang (Microsoft Research)  
Esther A. Hughes (Virginia Commonwealth University)  
Van Jacobson (Xerox PARC)  
Pinak Jain (former NYU-Poly student)  
Jobin James (University of California at Riverside)  
Sugih Jamin (University of Michigan)  
Shivkumar Kalyanaraman (IBM Research, India)  
Jussi Kangasharju (University of Helsinki)  
Sneha Kasera (University of Utah)  
Parviz Kermani (formerly of IBM Research)  
Hyojin Kim (former University of Pennsylvania student)  
Leonard Kleinrock (University of California at Los Angeles)  
David Kotz (Dartmouth College)  
Beshan Kulapala (Arizona State University)  
Rakesh Kumar (Bloomberg)  
Miguel A. Labrador (University of South Florida)  
Simon Lam (University of Texas)  
Steve Lai (Ohio State University)  
Tom LaPorta (Penn State University)  
Tim-Berners Lee (World Wide Web Consortium)  
Arnaud Legout (INRIA)

Lee Leitner (Drexel University)  
Brian Levine (University of Massachusetts)  
Chunchun Li (former NYU-Poly student)  
Yong Liu (NYU-Poly)  
William Liang (former University of Pennsylvania student)  
Willis Marti (Texas A&M University)  
Nick McKeown (Stanford University)  
Josh McKinzie (Park University)  
Deep Medhi (University of Missouri, Kansas City)  
Bob Metcalfe (International Data Group)  
Sue Moon (KAIST)  
Jenni Moyer (Comcast)  
Erich Nahum (IBM Research)  
Christos Papadopoulos (Colorado State University)  
Craig Partridge (BBN Technologies)  
Radia Perlman (Intel)  
Jitendra Padhye (Microsoft Research)  
Vern Paxson (University of California at Berkeley)  
Kevin Phillips (Sprint)  
George Polyzos (Athens University of Economics and Business)  
Sriram Rajagopalan (Arizona State University)  
Ramachandran Ramjee (Microsoft Research)  
Ken Reek (Rochester Institute of Technology)  
Martin Reisslein (Arizona State University)  
Jennifer Rexford (Princeton University)  
Leon Reznik (Rochester Institute of Technology)  
Pablo Rodriguez (Telefonica)  
Sumit Roy (University of Washington)  
Dan Rubenstein (Columbia University)  
Avi Rubin (Johns Hopkins University)  
Douglas Salane (John Jay College)  
Despina Sarapilla (Cisco Systems)  
John Schanz (Comcast)  
Henning Schulzrinne (Columbia University)  
Mischa Schwartz (Columbia University)  
Ardash Sethi (University of Delaware)  
Harish Sethu (Drexel University)  
K. Sam Shanmugan (University of Kansas)  
Prashant Shenoy (University of Massachusetts)  
Clay Shields (Georgetown University)  
Subin Shrestra (University of Pennsylvania)  
Bojie Shu (former NYU-Poly student)  
Mihail L. Sichitiu (NC State University)  
Peter Steenkiste (Carnegie Mellon University)  
Tatsuya Suda (University of California at Irvine)  
Kin Sun Tam (State University of New York at Albany)  
Don Towsley (University of Massachusetts)  
David Turner (California State University, San Bernardino)

Nitin Vaidya (University of Illinois)  
Michele Weigle (Clemson University)  
David Wetherall (University of Washington)  
Ira Winston (University of Pennsylvania)  
Di Wu (Sun Yat-sen University)  
Shirley Wynn (NYU-Poly)  
Raj Yavatkar (Intel)  
Yechiam Yemini (Columbia University)  
Dian Yu (NYU Shanghai)  
Ming Yu (State University of New York at Binghamton)  
Ellen Zegura (Georgia Institute of Technology)  
Honggang Zhang (Suffolk University)  
Hui Zhang (Carnegie Mellon University)  
Lixia Zhang (University of California at Los Angeles)  
Meng Zhang (former NYU-Poly student)  
Shuchun Zhang (former University of Pennsylvania student)  
Xiaodong Zhang (Ohio State University)  
ZhiLi Zhang (University of Minnesota)  
Phil Zimmermann (independent consultant)  
Mike Zink (University of Massachusetts)  
Cliff C. Zou (University of Central Florida)

Vogliamo inoltre ringraziare l'intero team di Pearson, in particolare Carole Snyder e Tracy Johnson, che hanno svolto un lavoro assolutamente eccezionale in questa ottava edizione (e che hanno sopportato due autori molto pignoli che sembrano congenitamente incapaci di rispettare le scadenze!). Grazie anche agli artisti, Janet Theurer e Patrice Rossi Calkin, per il loro lavoro sulle belle figure nelle precedenti edizioni del nostro libro, e a Manas Roy e il suo team di SPi Global per il loro meraviglioso lavoro di produzione su questa edizione. Infine, un ringraziamento speciale va ai nostri precedenti redattori di Addison-Wesley e Pearson—Matt Goldstein, Michael Hirsch e Susan Hartman. Questo libro non potrebbe essere quello che è (e potrebbe non essere stato affatto) senza la loro gestione, costante incoraggiamento, pazienza quasi infinita, buon umore e perseveranza.

# Pearson MyLab

## UN AMBIENTE PER LO STUDIO

L'attività di apprendimento di questo corso continua in **MyLab**, l'**ambiente digitale per lo studio** che completa il libro offrendo **risorse didattiche** fruibili sia in **modo autonomo** sia per **assegnazione del docente**. Il **codice sulla copertina** di questo libro consente **l'accesso per 18 mesi a MyLab**.

## COME ACCEDERE

- 1. Registrati** come **studente universitario** all'indirizzo [registrazione.pearson.it](http://registrazione.pearson.it) (Se sei già registrato passa al punto successivo);
- 2.** effettua il **login** alla tua MyPearsonPlace all'indirizzo [www.pearson.it/place](http://www.pearson.it/place) e registra il prodotto digitale cliccando su **Attiva prodotto** ed inserendo il codice presente in copertina;
- 3.** entra nella sezione *Prodotti* e clicca sul tasto **AVVIA** presente di fianco all'immagine della copertina del testo;
- 4.** clicca su **classe MyLab studio autonomo** o, in alternativa, su **Iscriviti a una classe** ed inserisci il codice classe indicato dal tuo docente.



## CHE COSA CONTIENE

MyLab offre la possibilità di accedere al Manuale online: l'**edizione digitale del testo** arricchita da funzionalità che permettono di personalizzarne la fruizione, attivare la sintesi vocale, inserire segnalibri.

Inoltre la **piattaforma digitale MyLab** integra e monitora il percorso individuale di studio con **attività formative e valutative specifiche**. La loro descrizione dettagliata è consultabile nella pagina di catalogo dedicata al libro, all'indirizzo [link.pearson.it/DB700B29](http://link.pearson.it/DB700B29) oppure tramite il presente QR code.



# Reti di calcolatori e Internet

Internet è oggi presumibilmente il più grande sistema ingegnerizzato mai stato creato dall'uomo: centinaia di milioni di calcolatori connessi, collegamenti e commutatori, miliardi di utenti che si connettono tramite computer portatili, tablet e smartphone e infine una serie di nuove "cose" (*things*) connesse a Internet, quali console di gioco, sistemi di sorveglianza, orologi, occhiali, termostati e automobili. Data la grandezza, il numero di componenti e la molteplicità di usi di Internet, v'è speranza alcuna di capire come funziona? Esistono principi guida e strutture in grado di fornire un fondamento alla comprensione di un tale sistema straordinariamente vasto e complesso? E se sì, è possibile che apprendere che cosa siano le reti di calcolatori possa risultare interessante e persino divertente? Fortunatamente, la risposta a tutte queste domande è un deciso "Sì"! Infatti, il nostro obiettivo nello scrivere questo libro è offrire un'introduzione moderna al campo, sempre in evoluzione, delle reti di calcolatori, fornendo i principi e gli approfondimenti pratici necessari per comprendere non solo le reti odierne, ma anche quelle future.

Questo primo capitolo presenta un'ampia panoramica del networking e di Internet e costituisce la base del libro, con lo scopo di fornire una visione d'insieme. Introdurrà i concetti di base e prenderà in considerazione vari componenti che costituiscono una rete, senza perderne di vista la visione globale.

La struttura è la seguente. Dopo aver introdotto la terminologia e i concetti fondamentali, prenderemo in esame i componenti di base, hardware e software, che costituiscono una rete. Inizieremo dall'esterno per analizzare i sistemi alla periferia e le applicazioni in esecuzione sulle reti.

Esploreremo poi il nucleo di una rete di calcolatori, esaminando collegamenti e sistemi che trasportano i dati e li smistano, così come le reti di accesso e i supporti fisici che connettono i sistemi periferici al nucleo della rete.

Impareremo che Internet è una rete di reti e vedremo come tali reti si connettono tra loro.

Completata questa rassegna passeremo a una visione più ampia e più astratta. Nella seconda metà del capitolo esamineremo i ritardi, le perdite e il throughput di una rete e forniremo semplici modelli quantitativi che prendono in considerazione i ritardi di trasmissione, di propagazione e di accodamento per il computo del ritardo e del throughput complessivi. Introdurremo poi alcuni principi fondamentali dell'architettura delle reti di calcolatori, ossia la stratificazione dei protocolli e i modelli di servizio. Impareremo anche che le reti di calcolatori sono vulnerabili a vari tipi di attacchi: ne illustreremo alcuni e vedremo come le reti possano essere rese più sicure. Infine, concluderemo il capitolo con un breve excursus storico.

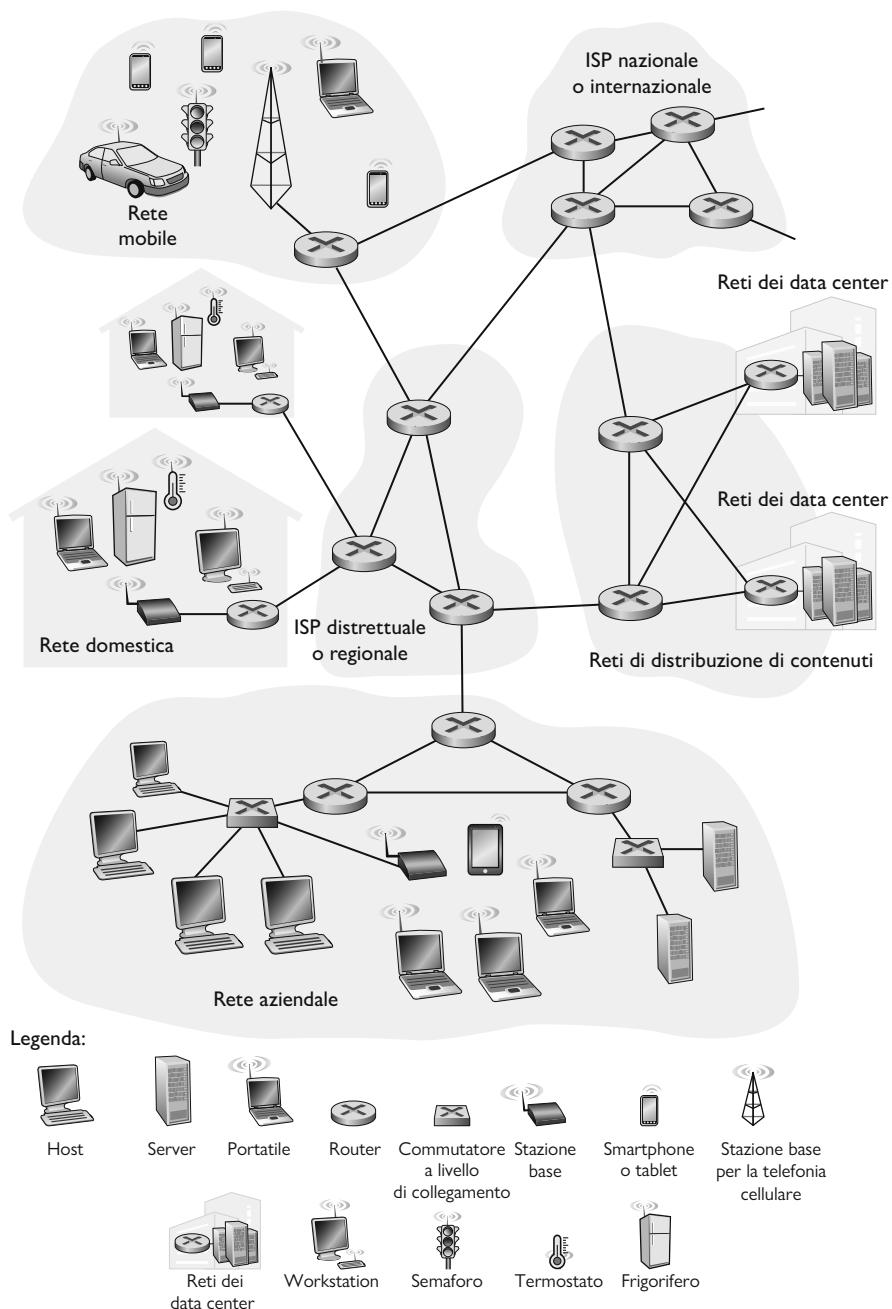
## 1.1 Che cos’è Internet?

In questo libro faremo riferimento a Internet, una specifica rete pubblica di calcolatori, come principale riferimento per affrontare lo studio delle reti e dei loro protocolli. Ma che cos’è Internet? Esistono due modi di rispondere a questa domanda. La prima è descrivere gli “ingranaggi” di Internet, ossia i componenti di base hardware e software che la compongono. Un altro metodo consiste nel descrivere Internet in termini di infrastruttura di rete che fornisce servizi ad applicazioni distribuite. Per cominciare usiamo il primo metodo, rifacendoci, nel corso della trattazione, alla Figura 1.1.

### 1.1.1 Gli “ingranaggi” di Internet

Internet è una rete di calcolatori che interconnette miliardi di dispositivi di calcolo in tutto il mondo. Non molto tempo fa questi dispositivi di elaborazione erano fondamentalmente PC tradizionali, workstation Linux o server che immagazzinavano e trasmettevano informazioni quali pagine web e messaggi di posta elettronica. Sempre più spesso però ci si connette a Internet tramite smartphone e tablet. Attualmente circa la metà della popolazione mondiale è un utente attivo su Internet da dispositivo mobile e si stima che nel 2025 lo sarà il 75% della popolazione mondiale [Statista 2019]. Inoltre “cose” (*things*) finora non connesse a Internet, quali TV, console di gioco, termostati, sistemi di sorveglianza, elettrodomestici, orologi, occhiali, automobili, sistemi di controllo del traffico e molte altre vengono via via connesse. Indubbiamente il termine *rete di calcolatori* comincia a essere datato, visto il grande numero di dispositivi non tradizionali collegati a Internet. In gergo, tutti questi dispositivi sono detti **host (ospite)** o **sistemi periferici (end system)**. Si stima che nel 2017 i sistemi periferici connessi a Internet fossero circa 18 miliardi, che diventeranno 28,5 miliardi nel 2022 [Cisco VNI 2020].

I sistemi periferici sono connessi tra loro tramite una **rete di collegamenti** (*communication link*) e **commutatori di pacchetti** (*packet switch*). Tali collegamenti, come vedremo nel Paragrafo 1.2, possono essere di molti tipi, costituiti da varie tipologie di mezzi fisici, tra cui cavi coassiali, fili di rame, fibre ottiche e onde elettromagnetiche. Collegamenti diversi possono trasmettere dati a velocità differenti, e tale **velocità [o tasso] di trasmissione** (*transmission rate*) viene misurata in bit/secondo (*bps*). Quando un sistema periferico vuole inviare dati a un altro sistema periferico, suddivide i dati in sottoparti e aggiunge un’inte-



**Figura 1.1** Componenti di base di Internet.

stazione a ciascuna di esse: l'insieme delle informazioni risultanti, nel gergo delle reti, viene chiamato **pacchetto (packet)**. I pacchetti sono inviati attraverso la rete alla destinazione, dove vengono riassemblati per ottenere i dati originari.

Un commutatore di pacchetto prende un pacchetto che arriva da uno dei collegamenti in ingresso e lo ritrasmette su uno di quelli in uscita. Esistono commutatori di pacchetto di varia forma e natura, ma i due principali nell'odierna

Internet sono i **router** e i **commutatori a livello di collegamento** (*link-layer switch*). Entrambe le tipologie instradano i pacchetti verso la loro destinazione finale. I commutatori a livello di collegamento sono solitamente usati nelle reti di accesso, mentre i router nel nucleo della rete. Dal sistema di invio a quello di ricezione, la sequenza di collegamenti e di commutatori di pacchetto attraversata dal singolo pacchetto è nota come **percorso** (*route o path*) attraverso la rete. Cisco stima che il traffico IP globale per anno raggiungerà circa i 5 zetta-byte ( $10^{21}$  byte) nel 2022 [Cisco VNI 2020].

Le reti a commutazione di pacchetto (in cui transitano i pacchetti) sono molto simili alle reti stradali e autostradali (in cui transitano veicoli). Si consideri, per esempio, un’azienda che voglia spostare una gran quantità di merci verso un deposito dislocato a migliaia di chilometri di distanza. Nell’azienda le merci sono suddivise in parti più piccole e caricate su una “flotta” di camion, ciascuno dei quali viaggia indipendentemente attraverso la rete di autostrade e strade fino al deposito. Qui le merci vengono scaricate e raggruppate con il resto del carico appartenente alla stessa spedizione. I pacchetti sono simili ai camion, i collegamenti sono analoghi alle autostrade e alle strade, i commutatori di pacchetto agli incroci e i sistemi periferici agli edifici. Come un camion segue un percorso lungo la rete autostradale, così un pacchetto procede lungo un percorso nelle reti di calcolatori.

I sistemi periferici accedono a Internet tramite i cosiddetti **Internet Service Provider (ISP)** che comprendono ISP residenziali quali le compagnie telefoniche, ISP aziendali, ISP universitari, ISP che forniscono accesso WiFi in aeroporti, hotel, bar e altri luoghi pubblici e ISP che forniscono accesso in mobilità a smartphone e altri dispositivi. Un provider è un insieme di commutatori di pacchetto e di collegamenti. Gli ISP forniscono ai sistemi periferici svariati tipi di accesso alla rete, tra cui quello residenziale a larga banda come la DSL, quello in rete locale ad alta velocità e quello wireless (senza fili e in mobilità). Inoltre gli ISP rendono disponibile l’accesso a Internet ai fornitori di contenuti, connettendone i server a Internet. Poiché Internet non è altro che connettere tra loro gli end systems, anche gli ISP che forniscono accesso ai sistemi periferici devono essere interconnessi. Tali provider di livello gerarchico più basso sono interconnessi a quelli nazionali e internazionali di livello più alto, che sono connessi tra di loro direttamente. Un ISP di livello superiore è costituito da router ad alta velocità interconnessi tramite fibra ottica. Ciascuna rete di un ISP, sia di alto sia di basso livello, è gestita in modo indipendente, fa uso del protocollo IP e si conforma a determinate convenzioni riguardo a nomi e indirizzi. Nel Paragrafo 1.3 esamineremo in dettaglio i provider e le loro interconnessioni.

Sistemi periferici, commutatori di pacchetto e altre parti di Internet fanno uso di protocolli che controllano l’invio e la ricezione di informazioni all’interno della rete. Due dei principali protocolli Internet sono il **Transmission Control Protocol (TCP)**, e l’**Internet Protocol (IP)**. Quest’ultimo specifica il formato dei pacchetti scambiati tra router e sistemi periferici. I principali protocolli Internet sono noti con il nome collettivo di **TCP/IP**. Incominceremo a parlare di protocolli in questo capitolo introduttivo, ma è solo l’inizio, la maggior parte del libro sarà dedicata a loro!

Data la loro importanza per Internet, un accordo sulle funzioni svolte da ogni singolo protocollo risulta fondamentale per creare sistemi e prodotti che

interagiscano. Ecco dove entrano in gioco gli standard. Gli **standard di Internet** vengono sviluppati dall'**Internet Engineering Task Force (IETF)** [IETF 2020]. Le pubblicazioni sugli standard di Internet vengono dette **Request For Comment (RFC)**. Inizialmente si trattava di richieste generiche di commenti (da cui il nome) per risolvere problemi architetturali sulle reti precedenti a Internet [Allman 2011]. Le RFC tendono a essere piuttosto tecniche e dettagliate. Esse definiscono vari protocolli tra cui TCP, IP, HTTP (per il Web) e SMTP (per la posta elettronica). Esistono attualmente più di novemila RFC. Anche altri enti specificano standard per i componenti di rete, in particolare per i collegamenti di rete. L'IEEE 802 LAN Standards Committee [IEEE 802 2020], per esempio, specifica gli standard per Ethernet e wireless Wi-Fi.

### 1.1.2 Descrizione dei servizi

A questo punto abbiamo identificato numerosi componenti che concorrono a costituire Internet; possiamo tuttavia descrivere Internet anche da un punto di vista completamente diverso, cioè come un'infrastruttura che fornisce servizi alle applicazioni. Tali applicazioni includono posta elettronica, navigazione sul Web, messaggistica istantanea, sistemi di navigazione con informazioni sul traffico in tempo reale, streaming di musica e video, on-line social media, sistemi di videoconferenza, giochi multiplayer e sistemi di raccomandazione basati sulla posizione. Queste applicazioni sono dette **applicazioni distribuite** (*distributed applications*), in quanto coinvolgono più sistemi periferici che si scambiano reciprocamente dati. L'aspetto più rilevante è che le applicazioni Internet vengono eseguite sui sistemi periferici e non sui commutatori di pacchetto del nucleo della rete. Sebbene i commutatori di pacchetto consentano lo scambio di dati tra sistemi periferici, non hanno a che fare con le applicazioni che sono sorgenti e destinazioni dei dati.

Esploriamo brevemente il concetto di infrastruttura che fornisce servizi alle applicazioni. A questo scopo, immaginate di avere una nuova idea per un'applicazione Internet distribuita che possa elargire un grande beneficio all'umanità o che possa semplicemente rendervi ricchi e famosi. Come fate a trasformare tale idea in una reale applicazione Internet? Dato che le applicazioni sono eseguite sui sistemi periferici, dovete scrivere dei moduli software, in Java, C o Python, che vengano eseguiti sui sistemi periferici. Dal momento che state sviluppando un'applicazione Internet distribuita, i moduli software, eseguiti sui diversi sistemi periferici, avranno bisogno di scambiarsi i dati. Si arriva così al nocciolo della questione: quello che porta a descrivere Internet come una piattaforma per le applicazioni. Come fa una parte di applicazione eseguita su un sistema periferico a istruire Internet affinché recapiti dati a un'altra parte di software eseguita su un altro sistema periferico?

I sistemi periferici collegati a Internet forniscono una **interfaccia socket** (*socket interface*), che specifica come un programma eseguito su un sistema periferico possa chiedere a Internet di recapitare dati a un programma eseguito su un altro sistema periferico. L'interfaccia socket è un insieme di regole che il programma mittente deve seguire in modo che i dati siano recapitati al programma di destinazione.

Descriveremo le interfacce socket di Internet in dettaglio nel Capitolo 2, per ora ricorriamo a una semplice analogia a cui faremo appello frequentemente. Immaginate che Alice voglia inviare una lettera a Bob usando il servizio postale. Ovviamente Alice non può limitarsi a scrivere la lettera (i dati) e a lanciarla fuori dalla finestra. Occorre che Alice inserisca la lettera in una busta sulla quale deve scrivere il nome completo del destinatario, il suo indirizzo e il codice postale. Deve poi sigillare la busta, incollare il francobollo e, infine, imbucarla. Tutto ciò costituisce l’“interfaccia del servizio postale” ovvero un insieme di regole che Alice deve seguire per far sì che il servizio di posta recapiti la sua lettera a Bob. In modo simile Internet dispone di interfaccia socket che il programma mittente deve seguire per far sì che Internet recapiti i dati al programma destinatario.

Il servizio postale fornisce più di un servizio ai suoi clienti: il recapito veloce, la conferma di ricezione, il servizio ordinario e altro ancora. In modo analogo Internet fornisce alle proprie applicazioni molti servizi che verranno descritti nel Capitolo 2.

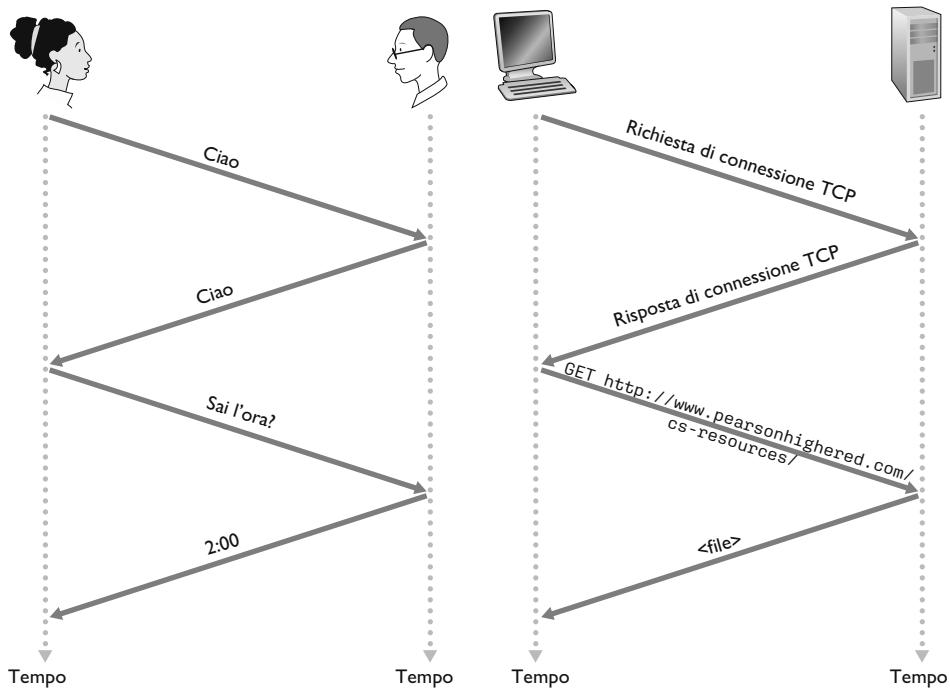
Abbiamo appena fornito due descrizioni di Internet, una nei termini dei suoi componenti hardware e software, l’altra nei termini dei servizi forniti alle applicazioni distribuite. Forse però il lettore è ancora confuso rispetto a che cosa sia Internet. Che cosa sono i commutatori di pacchetto e TCP/IP? Che cosa sono i router? Quali tipi di collegamenti sono presenti in Internet? Che cos’è un’applicazione distribuita? Come può connettersi a Internet un termostato o una bilancia? Se il lettore non ha ancora una risposta a tutte queste domande non c’è da preoccuparsi: lo scopo del testo è proprio quello di introdurlo agli “ingranaggi” di Internet così come ai principi che governano il suo funzionamento. Daremo una spiegazione di questi termini importanti e scioglieremo eventuali dubbi nei seguenti paragrafi e capitoli.

### 1.1.3 Che cos’è un protocollo?

Consideriamo ora un’altra parola chiave delle reti di calcolatori: **protocollo** (*protocol*). Che cos’è un protocollo? Che cosa fa?

#### Un’analogia

Probabilmente è più facile comprendere la nozione di protocollo di rete considerando dapprima alcune analogie con il comportamento umano, dato che noi eseguiamo protocolli in ogni momento. Si consideri che cosa si fa quando si vuole chiedere a qualcuno che ore sono, riferendosi come esempio alla Figura 1.2. Il “protocollo umano” (o quanto meno le buone maniere) impone come prima cosa un saluto (il primo “Ciao” nella Figura 1.2) per iniziare la comunicazione con qualcun altro. La tipica risposta è un messaggio “Ciao” di ritorno. Implicitamente, una persona interpreta una risposta cordiale “Ciao” come l’indicazione di poter procedere e chiedere l’ora. Una risposta diversa (quale un “Non mi scocciare!”, “Non parlo italiano” o qualsiasi altra) potrebbe indicare una scarsa propensione a comunicare o l’incapacità di farlo. In questo caso, si dovrebbe rinunciare a chiedere l’ora. Quando una persona non riceve risposta alla propria domanda, in linea di massima, non ripresenta la richiesta. Si noti che nel “protocollo umano” sono presenti specifici messaggi che inviamo e spe-



**Figura 1.2**  
Un protocollo umano  
e un protocollo di rete.

cifiche azioni che intraprendiamo in risposta ai messaggi ricevuti o ad altri eventi (quali la mancata ricezione di risposta in un tempo ragionevole). Chiaramente, i messaggi trasmessi e ricevuti e le azioni intraprese quando questi messaggi vengono inviati e acquisiti, rivestono un ruolo centrale nel protocollo umano. Se le persone adottano protocolli differenti (per esempio, se una persona è bene educata e l'altra no, o se una delle due comprende il concetto di tempo e l'altra no) i protocolli non interoperano e non è possibile portare a termine una transazione utile. Lo stesso concetto vale per le reti. Lo scambio si appoggia a due (o più) entità che comunicano utilizzando lo stesso protocollo al fine di assolvere un certo compito.

Consideriamo una seconda analogia. Supponiamo di seguire una lezione, per esempio, sulle reti di calcolatori. La voce del docente risuona monotona parlando di protocolli, e noi siamo disorientati. Il docente si interrompe per chiedere: "Ci sono domande?" (messaggio che viene trasmesso a tutti gli studenti e recepito da tutti quelli che non stanno dormendo). Alziamo la mano (trasmettendo un messaggio隐式 al docente). Il docente si rivolge a voi con un sorriso, dicendo "Sì ..." (messaggio trasmesso per incoraggiarvi a porgli una domanda, e i docenti sono felici di ricevere domande). Al che voi fate la vostra domanda (ossia trasmettete il vostro messaggio al docente). Quest'ultimo sente la domanda (riceve il messaggio) e risponde (vi trasmette una risposta). Ancora una volta, osserviamo che la trasmissione e la ricezione dei messaggi e le azioni convenzionali intraprese nel momento in cui tali messaggi vengono inviati e ricevuti rappresentano il cuore di questo protocollo a domanda e risposta.

### Protocolli di rete

Un protocollo di rete è simile a un “protocollo umano”, a eccezione del fatto che le entità che si scambiano messaggi e che intraprendono azioni sono componenti hardware o software di qualche dispositivo (computer, smartphone, tablet, router o altri dispositivi di rete). Qualsiasi attività in Internet che coinvolga due o più entità remote in comunicazione viene governata da un protocollo. Per esempio, i protocolli cablati nelle schede di rete di due calcolatori fisicamente connessi controllano il flusso di bit sul “cavo” tra le due schede; i protocolli di controllo di congestione nei sistemi periferici verificano la velocità alla quale i pacchetti vengono trasmessi tra mittente e destinatario; i protocolli nei router determinano un percorso per il pacchetto dalla sorgente alla destinazione. I protocolli vengono eseguiti in ogni parte di Internet e, di conseguenza, la maggior parte di questo testo riguarda proprio i protocolli di rete.

Come esempio di protocollo di rete con cui probabilmente il lettore ha familiarità si consideri che cosa succede quando si invia una richiesta a un web server, ossia quando si digita l’indirizzo di una pagina web in un browser. La situazione viene mostrata nella parte destra della Figura 1.2. Per prima cosa il vostro calcolatore invierà un messaggio di richiesta di connessione al web server e si metterà in attesa di una risposta. Il web server alla fine riceverà il vostro messaggio di richiesta di connessione e restituirà un messaggio di risposta di connessione. Sapendo che ora è possibile richiedere un documento web, il vostro computer invierà il nome della pagina che vuole prelevare dal server tramite un messaggio GET. Infine, il web server restituirà la pagina web (un file) al vostro calcolatore.

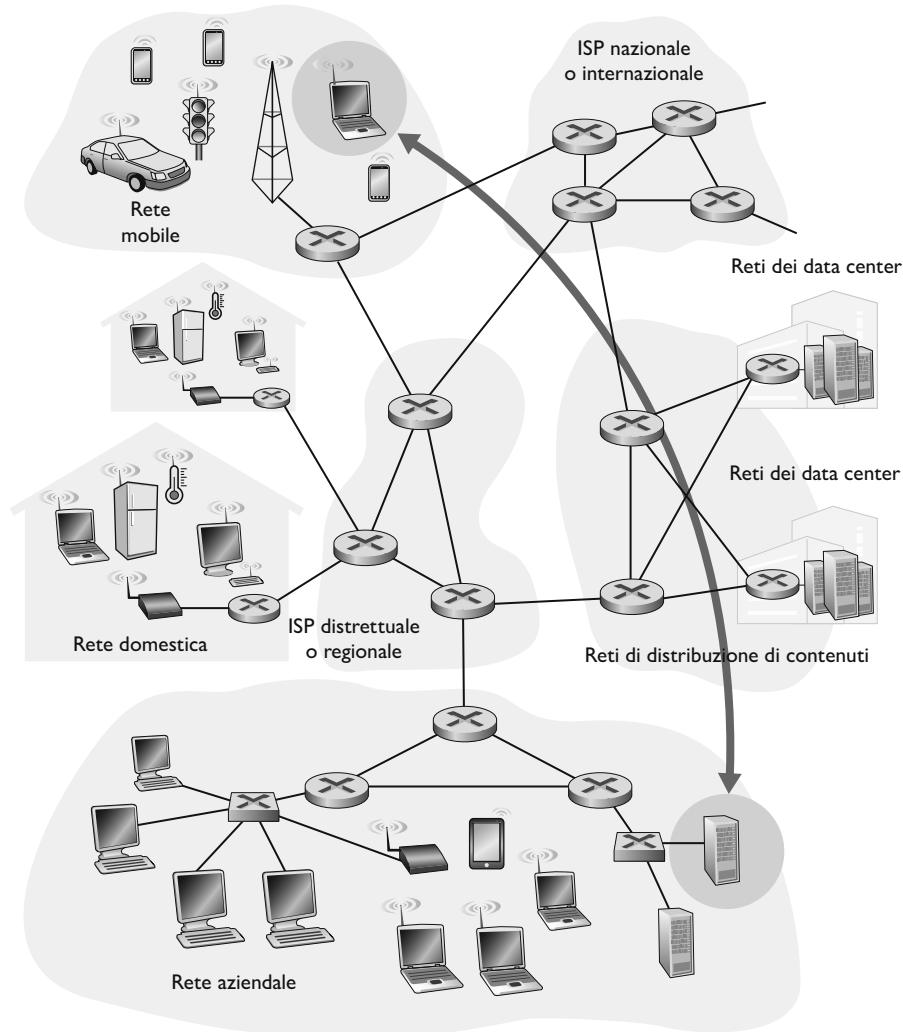
Sulla base degli esempi appena forniti, lo scambio di messaggi e le azioni intraprese quando tali messaggi vengono inviati e ricevuti sono gli elementi chiave che definiscono un protocollo.

*Un protocollo definisce il formato e l’ordine dei messaggi scambiati tra due o più entità in comunicazione, così come le azioni intraprese in fase di trasmissione e/o di ricezione di un messaggio o di un altro evento.*

Internet, e le reti di calcolatori in generale, fanno un uso estensivo di protocolli. Si impiegano protocolli differenti per realizzare compiti diversi. Come vedremo nel corso del libro alcuni protocolli sono semplici e diretti, mentre altri sono complessi e difficili da un punto di vista concettuale. Padroneggiare il campo delle reti di calcolatori equivale a comprendere l’essenza, la finalità e le modalità operative dei protocolli di rete.

## 1.2 Ai confini della rete

Nei precedenti paragrafi abbiamo presentato una panoramica ad alto livello di Internet e dei protocolli di rete. Approfondiamo ora i componenti delle reti di calcolatori e di Internet in particolare. Cominceremo in questo paragrafo dalla periferia della rete (*network edge*) e considereremo i componenti con cui abbiamo più familiarità, ossia i calcolatori, i telefoni cellulari e tutti gli altri dispositivi che utilizziamo quotidianamente. Nel paragrafo successivo ci sposteremo



**Figura 1.3** Interazione tra sistemi periferici.

verso il nucleo per esaminare la commutazione e l'instradamento nelle reti di calcolatori.

Nel paragrafo precedente abbiamo visto che, nel gergo delle reti di calcolatori, i calcolatori e gli altri dispositivi connessi a Internet sono solitamente detti **sistemi periferici** o **end system**, in quanto si trovano ai confini di Internet, come mostrato nella Figura 1.3. I sistemi periferici di Internet includono calcolatori desktop (per esempio: PC, Mac e workstation Linux), server (per esempio, per il Web e per la posta elettronica) e dispositivi mobili (come computer portatili, smartphone e tablet). Inoltre, sempre più spesso vengono connesse a Internet altri tipi di “cose”.

I sistemi periferici vengono anche detti **host** in quanto ospitano (ed eseguono) programmi applicativi quali browser, web server o software di lettura e gestione della posta elettronica. Nel corso del libro useremo i termini host, end system e sistema periferico in modo intercambiabile. Talvolta gli host vengono ulteriormente suddivisi in due categorie: **client** e **server**. In modo informale, i client sono

**BOX 1.1****CASO DI STUDIO****Data center e cloud computing**

Le aziende Internet come Google, Microsoft, Amazon e Alibaba hanno costruito enormi data center, ognuno dei quali ospita da decine a centinaia di migliaia di host. Tali data center non solo sono connessi a Internet, come mostrato nella Figura 1.1, ma contengono al loro interno reti complesse di computer che interconnettono i loro host. I data center sono i motori dietro le applicazioni Internet che utilizziamo quotidianamente. In generale, i data center hanno tre obiettivi che per concretezza descriviamo qui nel contesto di Amazon.

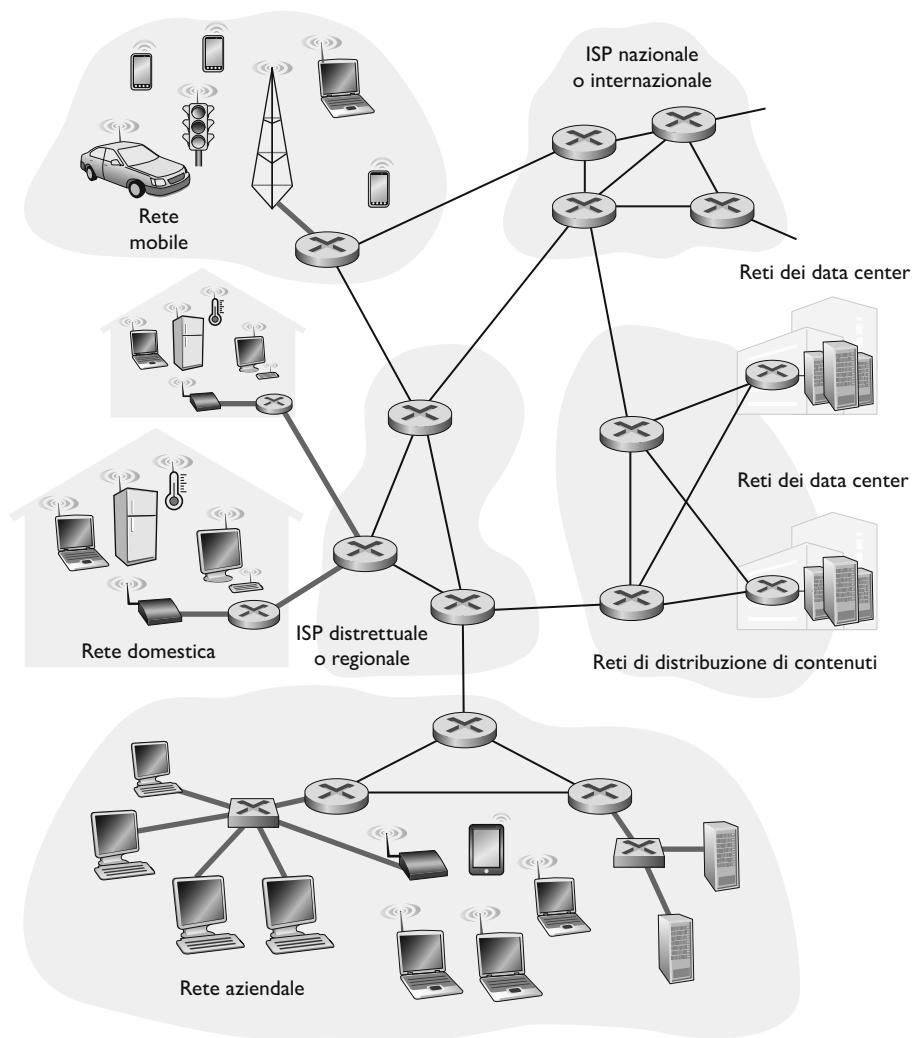
Innanzitutto, servono pagine di e-commerce di Amazon agli utenti quali, per esempio, pagine che descrivono prodotti e informazioni sull'acquisto. Secondo, servono come infrastrutture di elaborazione in parallelo per i dati specifici di Amazon. In terzo luogo, forniscono il servizio di **cloud computing** ad altre società. Infatti, una tendenza importante oggi nell'informatica è l'utilizzo da parte delle aziende di provider di servizi cloud come Amazon per gestire essenzialmente tutte le loro esigenze IT. Per esempio Airbnb e molte altre aziende basate su Internet non possiedono e non gestiscono i propri data center, ma eseguono tutti i loro servizi basati sul Web nel cloud Amazon, chiamato Amazon Service Web (AWS).

Le api operaie dei data center sono gli host. Forniscono contenuti quali pagine web e video, archiviano e-mail e documenti ed eseguono calcoli distribuiti. Gli host nei data center, chiamati "blade" e somiglianti a scatole per pizza, sono generalmente host che includono CPU, memoria e dischi di archiviazione. Gli host sono impilati in rack, ognuno contenente da 20 a 40 blade. I rack vengono quindi interconnessi utilizzando data center sempre più sofisticati e in evoluzione. Le reti dei data center vengono discusse in maggiore dettaglio nel Capitolo 6.

host che richiedono dei servizi e tendono a essere PC, laptop, smartphone e via dicendo, mentre i server si occupano di erogare dei servizi e sono sostanzialmente macchine più potenti che memorizzano e distribuiscono pagine web o flussi video, ritrasmettono la posta elettronica e così via. Oggigiorno, la maggior parte dei server da cui riceviamo i risultati delle ricerche, l'e-mail, le pagine web e i video è collocata in grandi **data center**. Google, per esempio, attualmente dispone di 19 data center, che insieme dispongono di alcuni milioni di server. La Figura 1.3 e il contenuto del Box 1.1 forniscono una descrizione più dettagliata dei data center.

### 1.2.1 Le reti di accesso

Finora abbiamo considerato le applicazioni e i sistemi periferici ai confini della rete; esaminiamo ora le **reti di accesso** (*access network*), cioè la rete che connette fisicamente un sistema al suo **edge router** (router di bordo), che è il primo router sul percorso dal sistema d'origine a un qualsiasi altro sistema di destinazione collocato al di fuori della stessa rete di accesso. La Figura 1.4 mostra svariati tipi di reti di accesso, evidenziate da linee spesse e ombreggiate e gli ambienti (casa, azienda e aree geografiche con accesso in mobilità wireless) in cui vengono usate.



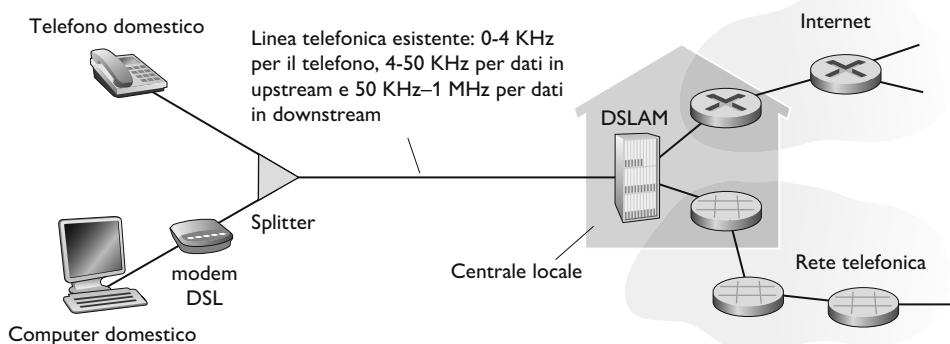
**Figura 1.4** Reti di accesso.

### Accesso residenziale: DSL, via cavo, FTTH e 5G fixed wireless

Nel 2020 più dell'80% delle abitazioni in Europa e negli Stati Uniti ha accesso a Internet [Statista 2019].

Oggi i due accessi residenziali a larga banda più diffusi sono il **digital subscriber line (DSL)** e quello via cavo. Un accesso residenziale a Internet di tipo DSL viene generalmente fornito dalla stessa compagnia telefonica che fornisce anche il servizio di telefonia fissa. In questo modo, in presenza di un accesso DSL, la compagnia telefonica assume anche il ruolo di ISP. Come mostrato nella Figura 1.5, il modem DSL dell'utente usa la linea telefonica esistente per scambiare dati con un *digital subscriber line access multiplex* (DSLAM) che si trova nella **centrale locale** (detta anche CO o *central office*) della compagnia telefonica. Il modem DSL residenziale converte i dati digitali in toni ad alta frequenza per poterli trasmettere alla centrale locale sul cavo telefonico; tutti i se-

**Figura 1.5** Accesso a Internet tramite DSL.



gnali analogici in arrivo dalle abitazioni vengono riconvertiti in formato digitale nel DSLAM.

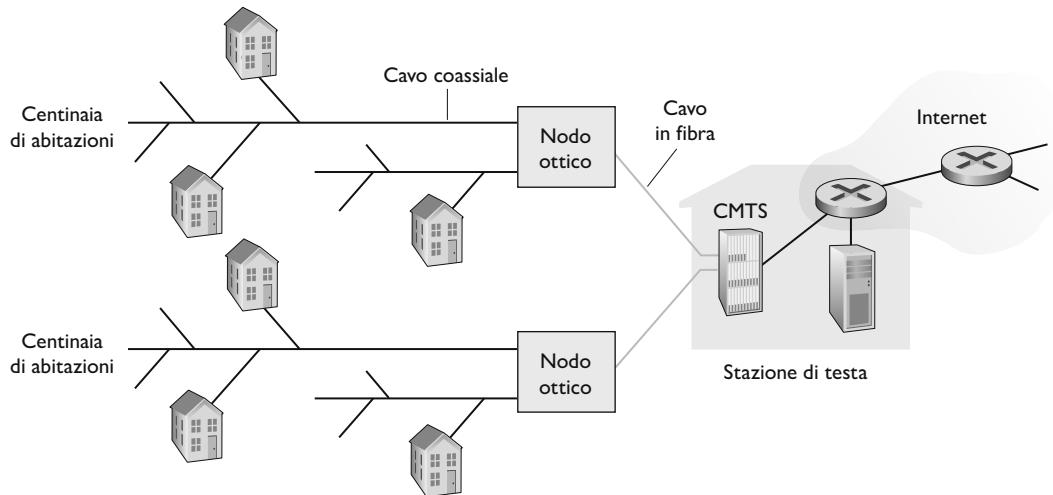
Le linee telefoniche residenziali trasportano contemporaneamente dati e segnali telefonici tradizionali codificandoli in tre bande di frequenza non sovrapposte:

- un canale di downstream (verso l'abitazione) ad alta velocità, nella banda tra 50 kHz e 1 MHz;
- un canale di upstream (verso il DSLAM) a velocità media, nella banda tra 4 e 50 kHz;
- un canale telefonico ordinario a due vie, nella banda tra 0 e 4 kHz.

Tale approccio fa apparire un singolo collegamento DSL come tre collegamenti separati, in modo che una chiamata telefonica e una connessione a Internet possono contemporaneamente condividere lo stesso collegamento DSL. Descriveremo questa tecnica di multiplexing a divisione di frequenza nel Paragrafo 1.3.1. Il DSLAM nella centrale locale separa i segnali dei dati da quelli della telefonia e invia i dati su Internet. Centinaia e anche migliaia di abitazioni sono connesse a un unico DSLAM.

Gli standard DSL definiscono i tassi di trasmissione tra i quali quello in downstream a 24 e 52 Mbps e quello in upstream a 3.5 e 16 Mbps; l'ultimo standard fornisce un tasso aggregato in downstream e upstream di 1 GBPS [ITU 2014]. L'accesso viene detto asimmetrico, perché le velocità di trasmissione in downstream e upstream sono diverse. Le velocità raggiunte effettivamente in downstream e upstream possono però essere inferiori, perché il provider DSL può limitare appositamente il tasso di trasmissione quando offre servizi a più livelli (velocità di trasmissione diverse a costi differenti) o perché il tasso di trasmissione massimo è limitato dalla distanza che intercorre tra l'abitazione e la centrale locale, dalla qualità del materiale con cui è costruito il doppino telefonico e dal grado di interferenza elettrica. La DSL è stata espressamente progettata per distanze piccole tra l'abitazione e la centrale locale; in generale, se l'abitazione dista più di 5 o 10 miglia (8 o 16 chilometri) dalla centrale locale, ci si deve dotare di una forma alternativa di accesso a Internet.

Mentre la DSL usa le infrastrutture già esistenti della compagnia telefonica locale, l'**accesso a Internet via cavo** utilizza le infrastrutture esistenti della tele-



**Figura 1.6** Rete di accesso ibrida a fibra e cavo coassiale.

visione via cavo. Un'abitazione richiede un accesso a Internet via cavo alla stessa azienda che le fornisce il servizio di televisione via cavo. Come illustrato nella Figura 1.6, le fibre ottiche connettono la terminazione del cavo a giunzioni a livello di quartiere, dalle quali viene usato il tradizionale cavo coassiale per la distribuzione televisiva per raggiungere le singole case e appartamenti. Ogni giunzione di quartiere serve generalmente da 500 a 5000 abitazioni. Tale sistema viene spesso chiamato hybrid fiber coax (HFC), in quanto impiega sia la fibra ottica sia il cavo coassiale.

L'accesso a Internet via cavo richiede modem speciali, chiamati **cable modem**. Così come il modem DSL, anche il cable modem è generalmente un dispositivo esterno che si connette al PC di casa attraverso una porta Ethernet (Ethernet verrà trattata in dettaglio nel Capitolo 6). Alla stazione di testa (*cable head end*) il sistema di terminazione del cable modem (*cable modem termination system*) svolge una funzione simile al DSLAM nelle reti DSL: traduce il segnale analogico inviato dai cable modem delle abitazioni a valle in formato digitale. I cable modem dividono la rete HFC in due canali, un canale in downstream e uno in upstream. Come per la DSL, l'accesso è asimmetrico: al canale in downstream vengono allocati tassi di trasmissione più elevati di quello in upstream. Lo standard DOCSIS 2.0 e gli standard 3.0 definiscono tassi di trasmissione in downstream di 40 Mbps e 1,2 Gbps e in upstream di 30 e 100 Mbps. Come nelle reti DSL, la velocità massima potrebbe non venir raggiunta a causa delle condizioni contrattuali o della bassa qualità del mezzo trasmittivo.

Un'importante caratteristica di HFC è il fatto di rappresentare un mezzo di trasmissione condiviso. In particolare, ciascun pacchetto inviato dalla stazione di testa viaggia sul canale di downstream in tutti i collegamenti e verso ogni abitazione; ciascun pacchetto inviato da un'abitazione viaggia sul canale di upstream verso la stazione di testa. Per questa ragione, se diversi utenti stanno contemporaneamente scaricando un file video sul canale di downstream, l'effettiva velocità alla quale ciascun utente riceve il proprio file video sarà significativamente inferiore rispetto a quella totale del canale di downstream. D'altra parte,

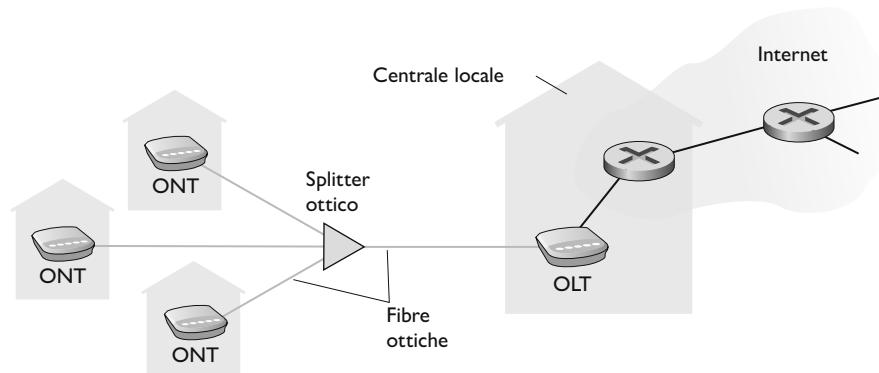
se solo pochi utenti stanno navigando in Internet, allora ciascuno di essi potrà effettivamente ricevere pagine web alla massima velocità di downstream, dato che difficilmente gli utenti richiedono una pagina web esattamente nello stesso istante. Essendo condiviso anche il canale di upstream, è necessario un protocollo di accesso multiplo distribuito per coordinare le trasmissioni ed evitare collisioni (un problema che sarà approfondito quando parleremo di Ethernet nel Capitolo 6).

Sebbene le reti DSL e via cavo rappresentino attualmente la maggioranza degli accessi residenziali a banda larga negli Stati Uniti, una tecnologia promettente che vanta velocità ancora maggiori è detta **fiber to the home (FTTH)** [Fiber Broadband 2020]. Come suggerito dal nome, il concetto di FTTH è semplice: fornire fibra ottica dalla centrale locale direttamente alle abitazioni. La tecnologia FTTH può potenzialmente fornire velocità di accesso a Internet di gigabit al secondo.

Ci sono diverse tecnologie in competizione per la distribuzione su fibra ottica dalle centrali locali alle abitazioni. La rete di distribuzione ottica più semplice è chiamata fibra diretta, in cui una singola fibra collega una centrale locale a un'abitazione. Di solito però una fibra uscente dalla centrale locale è in effetti condivisa da molte abitazioni e solo quando arriva relativamente vicina alle abitazioni viene suddivisa in più fibre, ognuna dedicata a un utente. Vi sono due architetture che eseguono questa suddivisione: le reti ottiche attive (AON, *active optical networks*) e quelle passive (PON, *passive optical networks*). Quelle AON sono essenzialmente Ethernet commutate e verranno discusse nel Capitolo 6.

Descriviamo ora brevemente le reti PON, usate nel servizio FIOS di Verizon. La Figura 1.7 mostra l'FTTH usata nell'architettura di distribuzione PON. Ogni abitazione ha un terminatore ottico chiamato *optical network terminator* (ONT), connesso a un separatore ottico (splitter) di quartiere tramite una fibra ottica dedicata. Lo splitter combina più abitazioni (generalmente meno di 100) in una singola fibra ottica condivisa, che si connette a un altro terminatore che prende il nome di *optical line terminator* (OLT), situato nella centrale locale della compagnia telefonica. L'OLT, fornendo la conversione tra segnali ottici ed elettrici, si connette a Internet tramite un router della compagnia telefonica. Nell'abitazione gli utenti connettono un router residenziale (generalmente wireless) all'ONT e accedono a Internet. Nell'architettura PON,

**Figura 1.7** Accesso a Internet tramite FTTH.



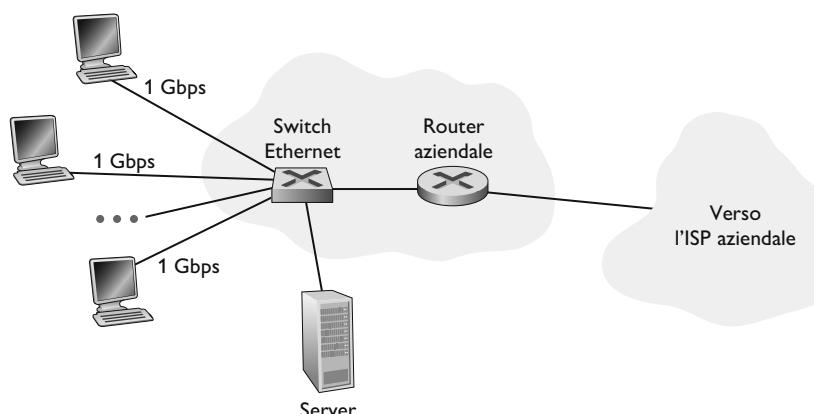
tutti i pacchetti inviati dall'OLT allo splitter sono replicati dallo splitter (similmente a una stazione di testa nel caso di HFC).

Oltre a DSL, cavo e FTTH, sta cominciando a essere utilizzato anche il servizio **5G Fixed Wireless Access** (FWA) che non solo promette un accesso residenziale ad alta velocità, ma lo fa senza installare cavi costosi e soggetti a guasti nel tratto dalla centrale locale (CO) a casa. Con tale tipo di accesso, utilizzando la tecnologia *beam-forming*, i dati vengono inviati in modalità wireless dalla stazione base di un provider al modem di casa. Un router wireless WiFi è collegato al modem (possibilmente in bundle), in modo simile a come un router wireless WiFi è collegato a un modem via cavo o DSL. Le reti cellulari 5G sono trattate nel Capitolo 7, disponibile in inglese sulla piattaforma MyLab.

### **Accesso aziendale (e residenziale): Ethernet e WiFi**

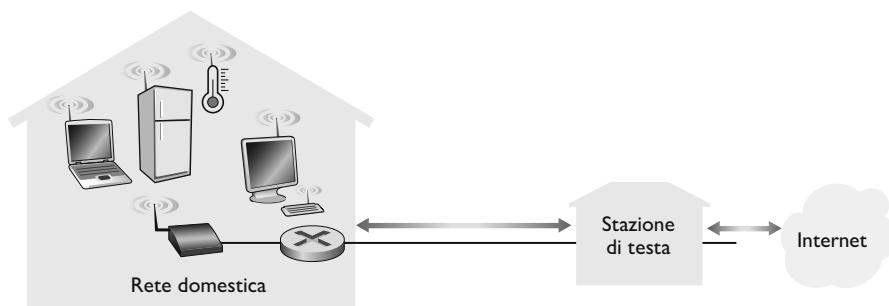
Nelle aziende e nelle università, e sempre di più nelle abitazioni, per collegare i sistemi periferici al router di bordo si utilizza una **rete locale** (LAN, *local area network*). Esistono molti tipi di LAN, ma la tecnologia Ethernet è attualmente la più utilizzata. Come mostrato nella Figura 1.8, Ethernet utilizza un doppino di rame intrecciato per collegare numerosi sistemi periferici tra loro e connetterli a uno switch Ethernet, una tecnologia che verrà discussa nel Capitolo 6. Lo switch, o una rete di apparati simili, viene poi a sua volta connesso a Internet. Con l'accesso tramite Ethernet gli utenti generalmente hanno velocità tra i 100 Mbps e i 10 Gbps, i server tra 1 e 10 Gbps

Sempre più utenti accedono a Internet via wireless da computer portatili, smartphone, tablet e altre “cose”. In una LAN wireless gli utenti trasmettono e ricevono pacchetti entro un raggio di poche decine di metri da e verso un access point wireless (detto anche stazione base o *base station*) connesso a una rete aziendale, che probabilmente include una rete Ethernet cablata, a sua volta connessa a Internet. Le LAN wireless (basate sulla tecnologia IEEE 802.11, nota anche come Wi-Fi) si stanno rapidamente diffondendo in ambienti universitari, uffici, locali pubblici, abitazioni e persino sugli aeroplani. La tecnologia IEEE 802.11 più comunemente installata, che discuteremo in dettaglio nel Capitolo 7, disponibile in inglese sulla piattaforma MyLab, fornisce una velocità di trasmissione condivisa intorno ai 100 Mbps.



**Figura 1.8** Accesso a Internet tramite Ethernet.

**Figura 1.9** Schema di una tipica rete domestica.



Sempre più frequenti sono i casi in cui l'accesso residenziale a larga banda (tramite cable modem o DSL) sia integrato con la poco costosa tecnologia LAN wireless, inizialmente riservata alle reti aziendali, al fine di creare reti domestiche potenti. La Figura 1.9 presenta lo schema di una tipica rete domestica, composta da un computer portatile, da un PC fisso e da alcuni elettrodomestici connessi a Internet, un access point che comunica con il computer portatile e altri dispositivi in modalità wireless e un router che connette l'access point a Internet. Questa rete consente ai residenti dell'abitazione di avere accesso a larga banda a Internet da qualunque punto della casa.

### Accesso wireless su scala geografica: 3G e LTE 4G e 5G

I dispositivi mobili come quelli iPhone e Android vengono sempre più frequentemente impiegati per inviare messaggi, condividere fotografie, effettuare pagamenti, guardare film e ascoltare musica durante i propri spostamenti. Tali dispositivi utilizzano la stessa infrastruttura wireless usata dalla telefonia cellulare per inviare/ricevere pacchetti tramite una stazione base gestita da un fornitore di telefonia cellulare. A differenza del WiFi, l'utente può trovarsi a poche decine di chilometri dalla stazione base, invece che a poche decine di metri.

Le compagnie di telecomunicazioni hanno investito somme enormi nelle cosiddette reti wireless di quarta generazione (4G), che consentono accesso wireless a Internet a velocità fino a 60 Mbps. Tuttavia sono già disponibili tecnologie di accesso su scala geografica a velocità ancora più alta: la quinta generazione (5G). Tratteremo i principi fondamentali delle reti wireless e mobili, così come le tecnologie WiFi, 4G e 5G (e altre ancora) nel Capitolo 7, disponibile in inglese sulla piattaforma MyLab.

### 1.2.2 Mezzi trasmittivi

Nei precedenti paragrafi abbiamo fornito una panoramica di alcune delle più importanti tecnologie di accesso a Internet, indicando il mezzo fisico utilizzato. Per esempio, abbiamo affermato che HFC adotta una combinazione di fibre ottiche e cavi coassiali, che DSL e Ethernet impiegano fili in rame intrecciati, e che le reti di accesso mobili utilizzano lo spettro radio. In questo paragrafo forniremo una breve panoramica dei mezzi trasmittivi comunemente adottati in Internet.

Per definire che cosa si intenda con mezzo fisico facciamo una riflessione sulla breve esistenza di un bit che viaggia da un sistema periferico a un altro, attraversando una serie di collegamenti e router. Il povero bit viene ritrasmesso

più volte. Il sistema di origine è il primo a trasmettere il bit che, poco dopo, sarà ricevuto dal primo router della sequenza; questi lo ritrasmetterà al secondo router che provvederà a inviarlo al successivo router, e così via. Pertanto il nostro bit, quando viaggia dalla sorgente alla destinazione, passa per una serie di coppie trasmettitore-ricevitore, propagandosi dall'uno all'altro sotto forma di onda elettromagnetica o impulso ottico attraverso un **mezzo fisico**. Il mezzo fisico può presentarsi in differenti fogge e dimensioni e non deve necessariamente essere dello stesso tipo per ogni coppia trasmettitore-ricevitore lungo il percorso. Tra gli esempi di mezzi fisici ricordiamo il doppino intrecciato, il cavo coassiale, la fibra ottica multimodale, lo spettro radio terrestre e lo spettro radio satellitare. I mezzi fisici ricadono in due categorie: i **mezzi vincolati** (*guided media*) e quelli **non vincolati** (*unguided media*). Nei primi, le onde vengono contenute in un mezzo fisico, quale un cavo in fibra ottica, un filo di rame o un cavo coassiale. Nei secondi, le onde si propagano nell'atmosfera e nello spazio esterno, come avviene nelle LAN wireless o nei canali digitali satellitari.

Prima di inoltrarci nelle caratteristiche dei vari mezzi spendiamo qualche parola sui loro costi. Il costo effettivo di un collegamento fisico (rame, fibra ottica e così via) è spesso inferiore rispetto ad altri costi delle reti. In particolare, il costo del lavoro associato all'installazione del collegamento fisico può superare di vari ordini di grandezza quello del materiale. Per questo motivo, molti costruttori trovano conveniente installare negli edifici numerosi doppini intrecciati, fibra ottica e cavi coassiali in ogni stanza. Anche se inizialmente viene usato un solo mezzo, ci sono infatti buone probabilità che in futuro ne venga impiegato un altro, risparmiando così il costo di una successiva installazione di cavi.

### Doppino di rame intrecciato

Il mezzo trasmissivo vincolato meno costoso e più utilizzato è il doppino di rame intrecciato, usato da più di un secolo nelle reti telefoniche e comunemente presente nelle case e negli ambienti di lavoro. Infatti, più del 99% delle connessioni cablate impiegano questo mezzo dalla cornetta del telefono al centralino più vicino. Il doppino intrecciato è costituito da due fili di rame distinti, ciascuno spesso meno di 1 mm, disposti a spirale regolare. I fili vengono intrecciati assieme per ridurre l'interferenza elettrica generata da altre coppie presenti nelle vicinanze. Di regola, un certo numero di doppiini viene riunito e avvolto in uno schermo protettivo a formare un cavo. Una coppia di fili costituisce un singolo collegamento di comunicazione. Il **doppino intrecciato non schermato (UTP, unshielded twisted pair)** viene comunemente utilizzato per le reti all'interno di un edificio, cioè per le LAN. Le odierni velocità trasmissive di una rete locale che utilizza il doppino variano tra 10 Mbps e 10 Gbps. Tali velocità dipendono dallo spessore del filo e dalla distanza che separa trasmettitore e ricevitore.

Quando, negli anni '80, emerse la tecnologia a fibra ottica, il doppino intrecciato venne messo in discussione a causa della velocità trasmissiva relativamente bassa: alcuni pensavano addirittura che la tecnologia a fibra ottica avrebbe finito per sostituirlo completamente. Tuttavia il doppino intrecciato non cedette così facilmente. La tecnologia attuale del doppino intrecciato, in particolare la categoria 6a, può raggiungere velocità trasmissive di 10 Gbps per distanze inferiori

a un centinaio di metri. In conclusione, il doppino intrecciato ha rappresentato e tuttora rappresenta la soluzione dominante per le LAN ad alta velocità.

Come abbiamo affermato in precedenza il doppino intrecciato viene comunemente usato per l'accesso a Internet residenziale. Abbiamo visto che la tecnologia dei modem dial-up consente l'accesso a una velocità fino a 56 kbps su doppino e che la tecnologia DSL ha consentito agli utenti residenziali di accedere a Internet a velocità superiori a decine di Mbps (quando gli utenti vivono vicino al CO dell'ISP).

### **Cavo coassiale**

Anche il cavo coassiale è costituito da due conduttori di rame, ma questi sono concentrici anziché paralleli. Con questa struttura, e grazie a uno speciale isolamento e schermatura, il cavo coassiale può raggiungere alte frequenze di trasmissione. Il suo impiego è piuttosto comune nei sistemi televisivi via cavo. Come abbiamo visto precedentemente, questi sistemi sono stati di recente abbinati a modem via cavo per fornire agli utenti residenziali accesso a Internet a velocità di centinaia di Mbps. Nella televisione e nell'accesso a Internet via cavo, il trasmettitore trasla il segnale digitale su una specifica banda di frequenza, e il segnale analogico risultante viene inviato dal trasmettitore a uno o più ricevitori. Il cavo coassiale può essere utilizzato come **mezzo condiviso** vincolato. Più nello specifico, più sistemi periferici possono essere connessi direttamente al cavo e tutti ricevono quanto inviato da altri sistemi periferici.

### **Fibra ottica**

La fibra ottica è un mezzo sottile e flessibile che conduce impulsi di luce, ciascuno dei quali rappresenta un bit. Una singola fibra ottica può supportare enormi velocità trasmissive, fino a decine o centinaia di gigabit al secondo. Tale mezzo è immune all'interferenza elettromagnetica, presenta attenuazione di segnale molto bassa nel raggio di 100 chilometri ed è molto difficile da intercettare. Queste caratteristiche hanno reso la fibra ottica il mezzo trasmissivo vincolato a lungo raggio favorito dagli operatori, in particolare per i collegamenti intercontinentali. La maggior parte delle reti telefoniche a lungo raggio degli Stati Uniti e di altri Paesi impiega esclusivamente fibre ottiche. Le fibre ottiche sono anche il mezzo trasmissivo prevalente delle dorsali Internet. Tuttavia, l'alto costo di trasmettitori, ricevitori e commutatori ottici ha impedito il loro utilizzo per il trasporto a corto raggio, come nelle reti locali o nell'accesso alla rete tipico delle abitazioni. Le velocità dei collegamenti OC (*optical carrier*) standard variano da 51,8 Mbps a 39,8 Gbps; a queste specifiche si fa spesso riferimento come OC-*n* dove la velocità del collegamento è uguale a  $n \times 51,8$  Mbps. Gli standard oggi in uso comprendono OC-1, OC-3, OC-12, OC-24, OC-48, OC-96, OC-192 e OC-768.

### **Canali radio terrestri**

I canali radio trasportano segnali all'interno dello spettro elettromagnetico. Si tratta di un mezzo interessante, dato che non richiede l'installazione fisica di cavi, è in grado di attraversare le pareti, fornisce connettività agli utenti mobili e, potenzialmente, riesce a trasportare un segnale per lunghe distanze. Le caratteristiche dei canali radio dipendono in modo significativo dall'ambiente di

propagazione e dalla distanza alla quale il segnale deve essere inviato. L'ambiente determina infatti la perdita di segnale lungo il percorso (*path*) causata dalla distanza (*path loss*), dall'attraversamento di ostacoli (*shadow fading*), dalla riflessione sulle superfici (*multipath fading*) o dall'interferenza con altri canali radio o segnali elettromagnetici.

A grandi linee, i canali radio terrestri possono essere classificati in tre gruppi: quelli che operano su distanze molto piccole (uno o due metri), quelli che operano in aree locali in un raggio di qualche centinaio di metri, e quelli che operano in aree più vaste, che si estendono per decine di chilometri. I dispositivi a uso personale, come cuffie e tastiere wireless o quelli medicali, operano a corta distanza: le LAN wireless (Paragrafo 1.2.1) utilizzano canali radio su area locale; le tecnologie di accesso cellulare usano invece canali radio per aree più vaste, come vedremo nel Capitolo 7, disponibile in inglese sulla piattaforma MyLab.

### **Canali radio satellitari**

Un satellite per le comunicazioni collega due o più trasmettitori terrestri a microonde, noti come stazioni a terra (*ground station*). Il satellite riceve le trasmissioni su una banda di frequenza, rigenera il segnale utilizzando un ripetitore e trasmette segnali su un'altra frequenza. Nelle comunicazioni si usano due tipi di satellite: quelli **geostazionari (GEO, geostationary earth orbit)** e quelli a **bassa quota (LEO, low-earth orbiting)**.

I satelliti geostazionari vengono posizionati permanentemente in un'orbita a circa 36.000 chilometri dalla superficie terrestre, perfettamente sincronizzata con la rotazione della Terra. La grande distanza tra la stazione a terra e il satellite (e ritorno) introduce un ritardo di 280 millisecondi nella propagazione del segnale. Ciò non di meno, i collegamenti via satellite, in grado di operare alla velocità di centinaia di Mbps, vengono spesso utilizzati dove non è presente accesso a Internet via DSL o cavo.

I satelliti a bassa quota sono posizionati molto più vicino alla Terra e ruotano intorno al nostro pianeta esattamente come la Luna. Possono comunicare sia tra di loro sia con le stazioni a terra. Per fornire la copertura continua di un'area è necessario mandare in orbita molti satelliti. Attualmente si stanno sviluppando svariati sistemi di comunicazione a bassa quota. La tecnologia satellitare a bassa quota potrebbe, fra qualche tempo, essere utilizzata per l'accesso a Internet.

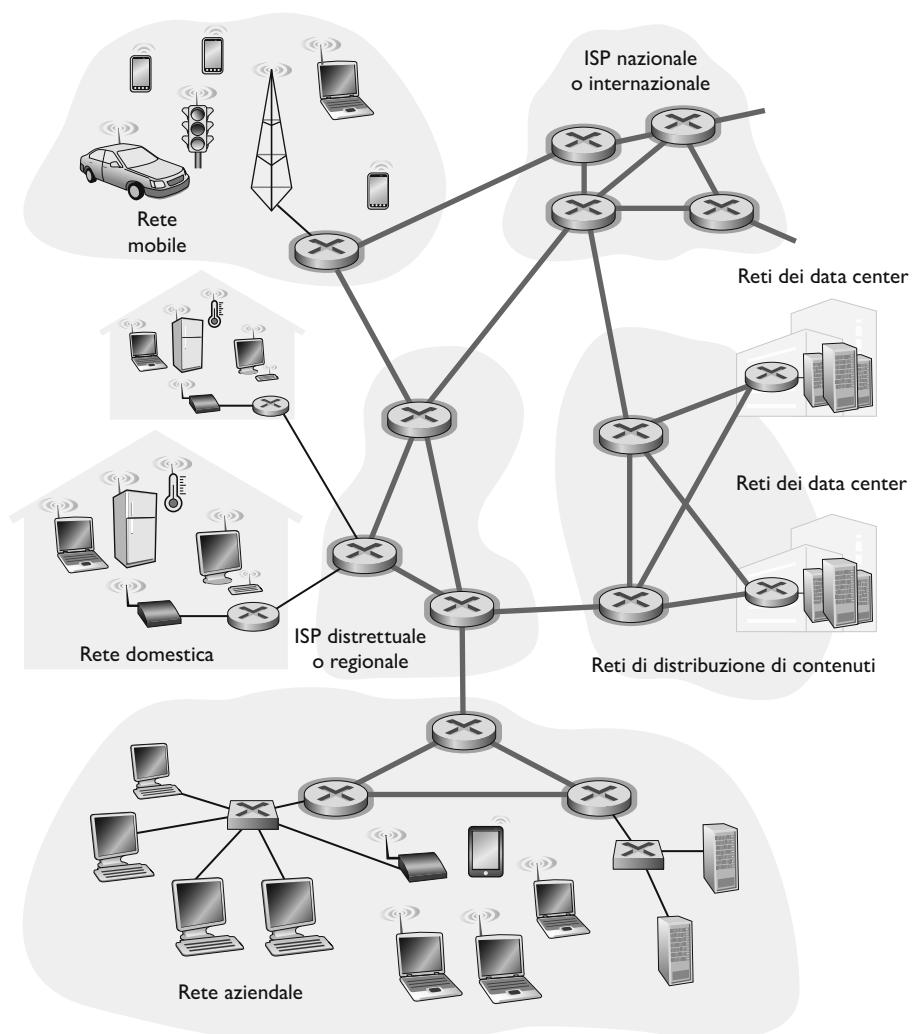
## **1.3 Il nucleo della rete**

Esaminati i confini di Internet, approfondiamo ora in dettaglio il nucleo della rete: una maglia di commutatori di pacchetti e collegamenti che interconnettono i sistemi periferici di Internet. Nella Figura 1.10 il nucleo della rete è evidenziato con linee più spesse e ombreggiate.

### **1.3.1 Compattezza di pacchetto**

Le applicazioni distribuite scambiano **messaggi** che possono contenere qualsiasi cosa il progettista del protocollo desideri. Possono svolgere una funzione di controllo (per esempio, il messaggio "Ciao" nell'esempio della Figura 1.2) o contenere dati come in un messaggio di posta elettronica, un'immagine JPEG

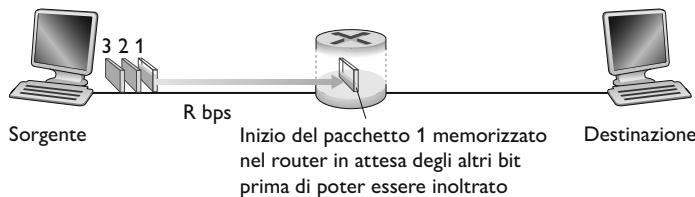
**Figura 1.10** Il nucleo della rete.



o un file audio MP3. La sorgente suddivide i messaggi lunghi in parti più piccole note come **pacchetti**. Tra la sorgente e la destinazione, questi pacchetti viaggiano attraverso collegamenti e **commutatori di pacchetto** (di cui, ricordiamo, esistono due tipi principali: i **router** e i **commutatori a livello di collegamento**). I pacchetti vengono trasmessi su ciascun collegamento a una velocità pari alla velocità totale di trasmissione del collegamento stesso. Quindi, se un sistema periferico o un commutatore invia un pacchetto di  $L$  bit su un canale con velocità di  $R$  bps, il tempo di trasmissione risulta pari a  $L/R$  secondi.

### Trasmissione store-and-forward

La maggior parte dei commutatori di pacchetto utilizza la **trasmissione store-and-forward**. Ciò significa che il commutatore deve ricevere l'intero pacchetto prima di poter cominciare a trasmettere sul collegamento in uscita il primo bit. Per capire meglio la trasmissione store-and-forward si consideri la semplice rete



**Figura 1.11**  
Commutazione  
di pacchetto  
store-and-forward.

mostrata nella Figura 1.11 che consiste di due sistemi periferici collegati attraverso un unico router. Un router è usualmente dotato di molti collegamenti; infatti la sua funzione è quella di instradare un pacchetto in entrata su un collegamento in uscita. In questo semplice esempio il compito del router è piuttosto facile: trasferire un pacchetto dall'unico collegamento in entrata al suo unico collegamento in uscita. In questo esempio la sorgente deve inviare alla destinazione tre pacchetti, ognuno di  $L$  bit. All'istante mostrato nella Figura 1.11 la sorgente ha già trasmesso parte del pacchetto 1 i cui primi bit sono già arrivati al router. Il router non può trasmettere i bit che ha ricevuto in questo momento, perché adotta la modalità store-and-forward; al contrario deve prima immagazzinare nel buffer i bit del pacchetto. Solo dopo aver ricevuto tutti i bit del pacchetto il router può iniziare a trasmettere (inoltrare) il pacchetto sul collegamento in uscita. Per approfondire la trasmissione store-and-forward calcoliamo ora il tempo che intercorre da quando la sorgente inizia a inviare il pacchetto a quando il destinatario lo ha completamente ricevuto. Stiamo qui trascurando il ritardo di propagazione, ossia il tempo che i bit impiegano a percorrere il collegamento a una velocità che si avvicina alla velocità della luce e che verrà discusso nel Paragrafo 1.4. La sorgente inizia la trasmissione al tempo 0; all'istante  $L/R$  secondi ha trasmesso l'intero pacchetto e quest'ultimo è stato ricevuto e memorizzato nel router, in quanto si sono trascurati i ritardi di propagazione. In tale istante il router, avendo appena ricevuto l'intero pacchetto, comincia a trasmetterlo sul collegamento in uscita verso il destinatario; all'istante  $2L/R$  secondi l'intero pacchetto è stato trasmesso dal router e ricevuto dal destinatario. Quindi il ritardo totale è  $2L/R$ . Se invece il router inoltrasse i bit appena arrivano (senza aspettare di ricevere l'intero pacchetto), il ritardo totale sarebbe pari a  $L/R$ , poiché i bit non verrebbero trattenuti nel router. Ma, come vedremo nel Paragrafo 1.4, i router necessitano di ricevere, memorizzare ed elaborare l'intero pacchetto prima di inoltrarlo.

Calcoliamo ora l'intervallo di tempo intercorso da quando la sorgente inizia a inviare il primo pacchetto a quando il destinatario li ha ricevuti tutti e tre. Come prima, al tempo  $L/R$  il router inizia a inoltrare il primo pacchetto. Ma sempre al tempo  $L/R$  la sorgente inizia a inviare il secondo pacchetto, perché ha appena completato l'invio del primo pacchetto. Quindi al tempo  $2L/R$  il destinatario ha ricevuto il primo pacchetto e il router ha ricevuto il secondo. Allo stesso modo, al tempo  $3L/R$ , il destinatario ha ricevuto i primi due pacchetti e il router ha ricevuto il terzo pacchetto. Infine al tempo  $4L/R$  il destinatario ha ricevuto tutti e tre i pacchetti.

Si consideri ora il caso generale della trasmissione di un pacchetto dalla sorgente alla destinazione su un percorso consistente di  $N$  collegamenti ognuno

con velocità di trasmissione  $R$  (quindi vi sono  $N - 1$  router tra la sorgente e il destinatario).

Applicando lo stesso ragionamento si trova che il ritardo da un capo all'altro (end-to-end) è:

$$d_{\text{end-to-end}} = N \frac{L}{R} \quad (1.1)$$

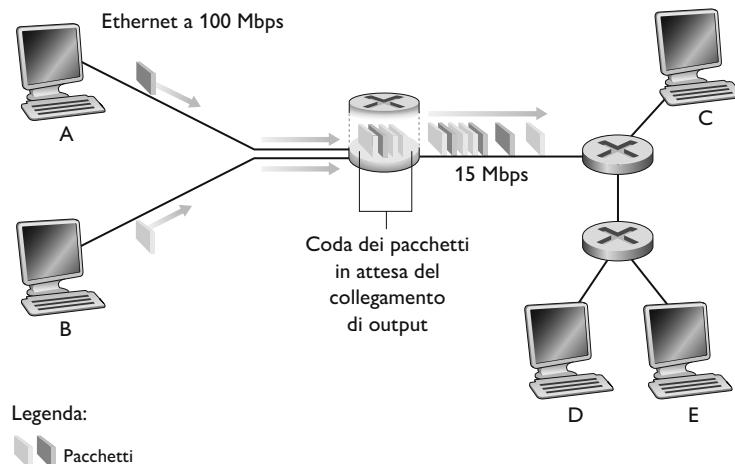
Potete ora tentare di calcolare a quanto ammonterebbe il ritardo per una trasmissione di  $P$  pacchetti su un percorso di  $N$  collegamenti.

### Ritardi di accodamento e perdita di pacchetti

Ogni commutatore di pacchetto connette più collegamenti. Per ciascuno di questi, il commutatore mantiene un **buffer di output** (detto anche **coda di output**) per conservare i pacchetti che sta per inviare su quel collegamento. I buffer di output rivestono un ruolo chiave nella commutazione di pacchetto. Un pacchetto in arrivo che debba essere inviato attraverso un collegamento occupato dalla trasmissione di un altro, deve attendere nella coda di output. Di conseguenza, oltre ai ritardi store-and-forward, i pacchetti subiscono anche **ritardi di accodamento**. Tali ritardi sono variabili e dipendono dal livello di traffico nella rete. Dato che la dimensione del buffer è finita, un pacchetto in arrivo può trovare il buffer completamente riempito da altri pacchetti che attendono la trasmissione. In questo caso si verificherà una **perdita di pacchetto** (*packet loss*): verrà cioè eliminato o il pacchetto in arrivo o uno di quelli che si trova già in coda.

La Figura 1.12 mostra una semplice rete a commutazione di pacchetto. Come nella Figura 1.11, i pacchetti vengono rappresentati da piccole lastre tridimensionali. Il loro spessore rappresenta il numero di bit del pacchetto. In questa figura tutti i pacchetti hanno lo stesso spessore e di conseguenza la stessa lunghezza. Si supponga che gli host A e B stiano inviando pacchetti a E. Dapprima inviano i loro pacchetti lungo i collegamenti Ethernet a 100 Mbps verso il primo commutatore di pacchetto. Quest'ultimo dirige i pacchetti al collegamento da 15 Mbps. Se per un breve lasso di tempo la velocità di arrivo dei

**Figura 1.12**  
Commutazione  
di pacchetto.



pacchetti al commutatore supera la velocità di inoltro sul collegamento in uscita si verifica una congestione, in quanto i pacchetti rimangono in coda nel buffer di output prima di venire trasmessi sul collegamento.

Se per esempio gli host A e B inviano una raffica di cinque pacchetti contemporaneamente, alcuni di questi passeranno del tempo in coda, come accade a noi quando per esempio siamo in fila in banca o al casello autostradale. Esamineremo i ritardi di coda più in dettaglio nel Paragrafo 1.4.

### **Tabelle di inoltro e protocolli di instradamento**

In precedenza abbiamo affermato che un router prende un pacchetto proveniente da uno dei suoi collegamenti e lo inoltra su un altro collegamento. Ma come fa il router a determinare su quale collegamento il pacchetto dovrebbe essere inoltrato? Ciò viene fatto in modi diversi a seconda del tipo di rete. In questo capitolo introduttivo descriveremo l'approccio usato da Internet.

In Internet ogni sistema periferico ha un indirizzo chiamato indirizzo IP. Ogni pacchetto che percorre la rete contiene nella propria intestazione l'indirizzo della sua destinazione che, come gli indirizzi postali, presenta una struttura gerarchica. Quando un pacchetto giunge a un router nella rete, quest'ultimo esamina una parte dell'indirizzo di destinazione e lo inoltra a un router adiacente. Più specificamente, ogni router ha una **tavella di inoltro** (*forwarding table*) che mette in relazione gli indirizzi di destinazione (o loro parti) con i collegamenti in uscita. Quando un pacchetto giunge a un router, questo esamina l'indirizzo e consulta la propria tabella per determinare il collegamento uscente appropriato. Il router quindi dirige il pacchetto verso quel collegamento di uscita.

Il processo di instradamento end-to-end è simile al comportamento di un automobilista che non utilizza cartine stradali, ma preferisce viaggiare chiedendo informazioni. Per esempio, supponiamo che Giovanni voglia andare da Milano a Conegliano, in Via Abate Tommaso, 515. Per prima cosa prende la tangenziale Est di Milano e, fermatosi a un distributore di benzina prima dello svincolo autostradale, chiede come raggiungere la sua destinazione. Il benzinaio, che sa come andare a Venezia, gli suggerisce di imboccare la A4, la cui entrata è molto vicina al distributore. Giovanni si immette sull'autostrada e guida fino a quando giunge nei pressi di Vicenza. Qui chiede informazioni a un altro addetto di una stazione di servizio, che gli dice che per arrivare a Conegliano gli conviene continuare sulla A4 fino alla barriera di Mestre, e quindi chiedere a qualcun altro. Superata la barriera, un altro benzinaio consiglia a Giovanni di imboccare la A27 in direzione di Vittorio Veneto. Giunto a Conegliano, entra in un bar per bere un caffè e chiede alla cassiera come andare in Via Abate Tommaso. Una volta raggiunta la strada, chiede quindi a un bambino in bicicletta come raggiungere il numero civico 515 e, finalmente, il viaggio di Giovanni è completato. Nella precedente analogia, gli addetti alle stazioni di servizio, le cassiere e i bambini in bicicletta sono simili a dei router.

Abbiamo appena appreso che un router usa l'indirizzo di destinazione del pacchetto per consultare una tabella di inoltro e determinare il collegamento di uscita corretto. Ma questa affermazione fa sorgere un'altra domanda: come vengono impostate le tabelle di inoltro? Vengono configurate manualmente in ciascun router o Internet impiega una procedura più automatizzata? Tali questioni verranno discusse approfonditamente nel Capitolo 5. Ma per stuzzicare

la vostra curiosità vi anticipiamo che Internet ha parecchi **protocolli di instradamento** (*routing protocol*) che usa per impostare automaticamente le tabelle di inoltro. Un protocollo di instradamento può, per esempio, determinare il percorso più corto da ciascun router verso ciascuna destinazione e usare questo risultato per configurare le tabelle di inoltro nei router.

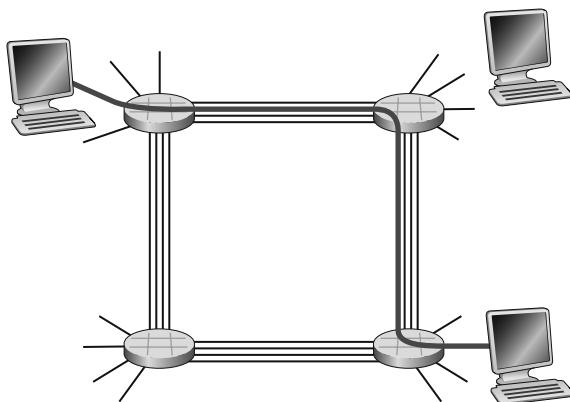
### 1.3.2 Commutazione di circuito

Per spostare i dati in una rete di collegamenti e commutatori esistono due approcci fondamentali: la **commutazione di circuito** e la **commutazione di pacchetto**. Di quest'ultima abbiamo parlato nel paragrafo precedente, esaminiamo ora le reti a commutazione di circuito.

Nelle reti a commutazione di circuito le risorse richieste lungo un percorso (buffer e velocità di trasmissione sui collegamenti) per consentire la comunicazione tra sistemi periferici sono riservate per l'intera durata della sessione di comunicazione. Nelle reti a commutazione di pacchetto, tali risorse non sono riservate; i messaggi di una sessione utilizzano le risorse quando necessario, e di conseguenza potrebbero dover attendere (ossia mettersi in coda) per accedere a un collegamento. Come analogia, si considerino due ristoranti: uno che richiede la prenotazione e l'altro che non la richiede né l'accetta. Nel primo caso, abbiamo l'incombenza di dover telefonare, ma quando arriviamo, in linea di principio, possiamo immediatamente accedere al nostro tavolo e ordinare la cena. Nel secondo non dobbiamo prenotare, ma quando arriviamo al ristorante potremmo dover attendere che si liberi un tavolo prima di poterci accomodare.

Le reti telefoniche sono esempi di reti a commutazione di circuito. Si consideri che cosa avviene quando una persona vuole inviare informazioni a un altro soggetto. Prima che possa iniziare l'invio, la rete deve stabilire una connessione tra mittente e destinatario. Questo è un collegamento “a regola d'arte” in cui i commutatori sul percorso tra mittente e destinatario mantengono lo stato della connessione per tutta la durata della comunicazione. Nel gergo della telefonia questa connessione è detta **circuito**. Quando la rete stabilisce un circuito, riserva anche una velocità di trasmissione costante (pari a una frazione della capacità trasmisiva del canale) nei collegamenti di rete per la durata della connessione. Dato che per questa connessione dal mittente al destinatario è stata riservata una certa larghezza di banda, il mittente può trasferire i dati a una velocità costante *garantita*.

La Figura 1.13 mostra una rete a commutazione di circuito in cui i quattro commutatori sono interconnessi tramite quattro collegamenti. Ciascuno di questi ultimi dispone di quattro circuiti, in modo che ogni collegamento possa supportare quattro connessioni simultanee. Gli host (per esempio PC e workstation) sono tutti direttamente connessi a uno dei commutatori. Quando due host desiderano comunicare la rete stabilisce una **connessione end-to-end** (o **connessione punto a punto**) dedicata a loro. Affinché A invii messaggi a B, la rete deve prima riservare un circuito su ciascuno dei due collegamenti. Nell'esempio, la connessione punto a punto usa il secondo circuito del primo collegamento e il quarto circuito del secondo. Poiché ogni collegamento ospita quattro circuiti, per ogni collegamento utilizzato dalla connessione punto a punto la connessione ottiene un quarto della capacità trasmisiva totale del collegamento per la du-



**Figura 1.13** Semplice rete a commutazione di circuito con quattro commutatori e quattro collegamenti.

rata della connessione stessa. Se, per esempio, ogni connessione tra switch adiacenti ha una velocità di trasmissione pari a 1 Mbps, ogni connessione end-to-end a commutazione di circuito riceve una capacità trasmissiva dedicata pari a 250 kbps.

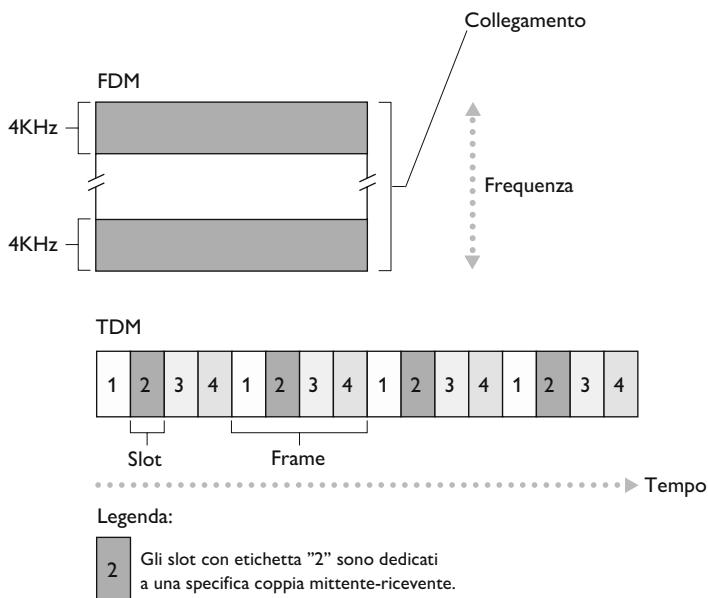
Si consideri invece che cosa accade quando un host invia un pacchetto a un altro host su una rete a commutazione di pacchetto, quale è Internet. Come nella commutazione di circuito, il pacchetto viene trasmesso su di una sequenza di collegamenti ma, a differenza della commutazione di circuito, il pacchetto viene immesso nella rete senza che vengano riservate risorse. Se un collegamento è congestionato perché vi sono altri pacchetti che devono essere trasmessi nello stesso istante, allora dovrà aspettare nel buffer del lato mittente e subire un ritardo. Internet fa del suo meglio per consegnare i pacchetti rispettando i tempi, ma non offre alcuna garanzia.

### Multiplexing nelle reti a commutazione di circuito

Un circuito all'interno di un collegamento è implementato tramite **multiplexing a divisione di frequenza** (**FDM**, *frequency-division multiplexing*) o **multiplexing a divisione di tempo** (**TDM**, *time-division multiplexing*). Con FDM, lo spettro di frequenza di un collegamento viene suddiviso tra le connessioni stabilite tramite il collegamento. Nello specifico, il collegamento dedica una banda di frequenza a ciascuna connessione per la durata della connessione stessa. Nelle reti telefoniche questa banda di frequenza ha normalmente un'ampiezza di 4 kHz (ossia 4000 hertz o 4000 cicli al secondo). La larghezza della banda viene detta **ampiezza di banda** (*bandwidth*). Anche le stazioni radio FM usano FDM per condividere lo spettro di frequenze tra gli 88 MHz e i 108 MHz, all'interno del quale ciascuna stazione ha assegnata una specifica banda di frequenza.

Per un collegamento TDM il tempo viene suddiviso in frame (intervalli) di durata fissa, a loro volta ripartiti in un numero fisso di slot (porzioni) temporali. Quando la rete stabilisce una connessione attraverso un collegamento, le dedica uno slot di tempo in ogni frame. Tali slot sono dedicati unicamente a quella connessione, con uno slot temporale disponibile in ciascun frame alla trasmissione dei dati di connessione.

**Figura 1.14** In FDM ogni circuito occupa con continuità una frazione dell'ampiezza di banda. In TDM, ogni circuito occupa l'intera ampiezza di banda per brevi intervalli di tempo (detti slot).



La Figura 1.14 mostra FDM e TDM per uno specifico collegamento di rete che supporta fino a quattro circuiti. Nel caso di FDM il dominio delle frequenze viene ripartito in quattro bande, ciascuna con ampiezza di 4 kHz. Nel caso di TDM, il dominio del tempo viene suddiviso in frame, con quattro slot di tempo per ciascun intervallo; a ogni circuito viene assegnato lo stesso slot dedicato in tutti i frame. Nel caso di TDM, la velocità di trasmissione di un circuito è uguale alla frequenza di frame moltiplicata per il numero di bit in uno slot. Per esempio, se il collegamento trasmette 8000 frame al secondo e ogni slot è costituito da 8 bit, allora la velocità di trasmissione del circuito è di 64 kbps.

I sostenitori della commutazione di pacchetto hanno sempre ritenuto che la commutazione di circuito fosse dispendiosa, dato che i circuiti dedicati sono inattivi durante i **periodi di silenzio**. Per esempio, durante una chiamata telefonica, quando una persona smette di parlare, le risorse di rete inutilizzate (le bande di frequenza o gli slot nei collegamenti lungo il percorso) non possono essere usate da altre connessioni. Come ulteriore esempio di come tali risorse possano essere sottoutilizzate, si consideri un radiologo che usa una rete a commutazione di circuito per accedere in remoto a una serie di lastre. Il radiologo predisponde una connessione, richiede un'immagine, la osserva e poi ne richiede un'altra. Per tutto il tempo in cui il radiologo guarda le lastre, le risorse di rete vengono spurate, in quanto allocate e non usate. I sostenitori della commutazione di pacchetto tendono anche a sottolineare la complicazione insita nello stabilire circuiti e nel riservare larghezza di banda punto a punto. Infatti ciò richiede un software complesso per coordinare le operazioni dei commutatori lungo il percorso.

Prima di concludere la nostra disquisizione sulla commutazione di circuito presentiamo un esempio numerico che dovrebbe fornire un ulteriore approfondimento. Consideriamo l'invio di un file di 640.000 bit dall'host A al B su una rete a commutazione di circuito. Si supponga che tutti i collegamenti nella rete

utilizzino TDM con 24 slot e presentino una capacità trasmissiva di 1,536 Mbps. Si ipotizzi poi di impiegare 500 ms per stabilire una connessione end-to-end prima che A possa iniziare a trasmettere il file. Quanto tempo richiede l'invio del file? Ogni circuito presenta una velocità di trasmissione di  $1,536 \text{ Mbps} / 24 = 64 \text{ kbps}$ , e pertanto la trasmissione richiede  $640.000 \text{ bit} / 64 \text{ kbps} = 10 \text{ secondi}$ . A questo tempo sommiamo il tempo per stabilire il circuito e ricaviamo che l'invio richiede in tutto 10,5 secondi. Si noti che il tempo di trasmissione è indipendente dal numero di collegamenti: 10 secondi sia nel caso in cui la connessione end-to-end passi tramite un solo collegamento sia nel caso di un centinaio di collegamenti. L'effettivo ritardo end-to-end include anche un ritardo di propagazione; si veda al riguardo il Paragrafo 1.4.

### **Confronto tra commutazione di pacchetto e commutazione di circuito**

Dopo aver descritto la commutazione di circuito e di pacchetto, confrontiamo i due approcci. I denigratori della commutazione di pacchetto hanno sovente sostenuto che il metodo non è adatto ai servizi in tempo reale (come la telefonia e la videoconferenza) a causa dei suoi ritardi end-to-end variabili e non determinabili a priori (dovuti principalmente alla variabilità e imprevedibilità dei ritardi di accodamento). I sostenitori della commutazione di pacchetto rispondono che quest'ultima non soltanto offre una migliore condivisione della larghezza di banda rispetto alla commutazione di circuito, ma è anche più semplice, più efficiente e meno costosa da implementare. Un interessante confronto tra i due approcci è diffusamente trattato in [Molinero-Fernandez 2002].

Perché la commutazione di pacchetto risulta più efficiente? Consideriamo un semplice esempio. Si supponga che gli utenti condividano un collegamento da 1 Mbps e che ciascun utente alterni periodi di attività, in cui genera dati a una velocità costante di 100 kbps, a momenti durante i quali non vengono generati dati. Si ipotizzi poi che l'utente sia attivo solo per il 10% del tempo (e beva caffè per il restante 90%). Con la commutazione di circuito è necessario *riservare* 100 kbps per ciascun utente in ogni istante. Per esempio, nel caso di TDM a commutazione di circuito, se un frame di un secondo viene diviso in 10 slot da 100 ms, ciascun utente si vedrebbe allocato uno slot per frame.

Pertanto il collegamento può supportare simultaneamente solo 10 ( $= 1\text{Mbps}/100 \text{ kbps}$ ) utenti. Con la commutazione di pacchetto la probabilità che un determinato utente sia attivo è pari a 0,1. Se sono presenti 35 utenti, la probabilità di avere 11 o più utenti attivi in contemporanea è approssimativamente 0,0004. Il Problema 8 a fine capitolo spiega come si ottiene questa probabilità. Quando ci sono 10 o meno utenti attivi contemporaneamente (il che avviene con probabilità 0,9996), il tasso di arrivo dei dati è minore o uguale a 1 Mbps, cioè la velocità di output del collegamento. Quindi, nel caso in cui ci siano 10 o meno utenti attivi, il flusso dei pacchetti attraverso il collegamento avviene sostanzialmente senza ritardo, come nel caso della commutazione di circuito. Se, invece, ci sono più di 10 utenti contemporaneamente attivi, la velocità aggregata di arrivo dei dati supera la capacità di output del collegamento, e la coda di output comincerà a crescere. Questa coda continuerà a crescere fino a quando la velocità aggregata di ingresso scenderà sotto 1 Mbps, momento in cui la lunghezza della coda inizierà a diminuire. Dato che la probabilità di

avere più di 10 utenti contemporaneamente attivi è in questo caso assai limitata, la commutazione di pacchetto fornisce sostanzialmente le stesse prestazioni della commutazione di circuito, *ma consente più del triplo degli utenti*.

Consideriamo ora un altro semplice esempio. Si supponga la presenza di 10 utenti, e che un utente improvvisamente generi 1000 pacchetti da 1000 bit, mentre gli altri rimangono inattivi senza generare traffico. Con la commutazione di circuito TDM e 10 slot da 1000 bit per frame, l'utente attivo può utilizzare soltanto il proprio slot temporale per trasmettere dati, mentre i restanti nove slot del frame rimangono inutilizzati. Trascorreranno dieci secondi prima che il milione di bit dell'utente attivo sia stato completamente trasmesso. Nel caso di commutazione di pacchetto, l'utente attivo può continuamente inviare i propri pacchetti alla massima velocità del collegamento (1 Mbps), dato che nessun altro utente genera pacchetti che richiedono di essere trasmessi assieme a quelli dell'utente attivo. In questo caso l'intera quantità di dati sarà trasmessa in un secondo.

Gli esempi riportati mostrano due motivi per cui le prestazioni della commutazione di pacchetto possono essere superiori a quelle della commutazione di circuito. Inoltre sottolineano la cruciale differenza tra le due forme di condivisione della velocità trasmisiva di un collegamento su più flussi di dati. La commutazione di circuito preallocata l'uso del collegamento trasmisivo indipendentemente dalla richiesta, con collegamenti garantiti, ma non utilizzati, che provocano dispendio di tempo. La commutazione di pacchetto d'altro canto alloca l'uso di collegamenti *su richiesta*. Pacchetto per pacchetto, la capacità trasmisiva dei collegamenti sarà condivisa solo tra gli utenti che devono trasmettere.

Sebbene la commutazione di pacchetto e quella di circuito siano entrambe presenti negli odierni sistemi di telecomunicazioni, la tendenza è certamente in direzione della commutazione di pacchetto. Perfino molte reti telefoniche tuttora a commutazione di circuito stanno lentamente migrando verso la commutazione di pacchetto, utilizzata in particolare per le costose chiamate internazionali.

### 1.3.3 Una rete di reti

Abbiamo precedentemente visto che i sistemi periferici (PC, smartphone, server per il Web e la posta elettronica e così via) si collegano a Internet tramite un ISP di accesso che può fornire connettività attraverso una rete cablata o senza fili con svariate tecnologie quali DSL, cavo, FTTH, Wi-Fi e cellulare. È da notare che l'ISP non deve necessariamente essere una compagnia di telecomunicazioni, ma potrebbe anche essere un'università (che eroga servizio a studenti, docenti e tecnici) o un'azienda (che fornisce connettività ai suoi dipendenti). Tuttavia, la connessione degli utenti finali e dei fornitori di contenuti alla rete di un ISP è solo una piccola parte del puzzle da risolvere per connettere i miliardi di utenti che costituiscono Internet. Per completare il puzzle bisogna interconnettere gli stessi ISP. Ciò avviene creando una **rete di reti** (*network of networks*): capire questo concetto è la chiave per capire Internet.

Nel corso degli anni la rete di reti che forma Internet si è evoluta in una struttura altamente complessa. Per la maggior parte questa evoluzione è stata

pilotata da fattori economici e politici più che dalle prestazioni. Per comprendere la struttura di rete dell'Internet odierna costruiamo ora una sequenza incrementale di strutture di rete in cui ogni nuova struttura ne sia un'approssimazione migliore. Ricordiamo che l'obiettivo generale è quello di interconnettere gli ISP di accesso in modo che i sistemi periferici possano scambiarsi pacchetti. Un approccio naïf sarebbe quello di connettere direttamente ogni ISP di accesso con tutti gli altri. Una struttura a maglia completa (*mesh*) è, ovviamente, troppo costosa per gli ISP, in quanto richiederebbe a ognuno di essi di avere un collegamento separato per ciascuna delle centinaia di migliaia degli altri ISP sparsi in tutto il mondo.

La nostra prima struttura di rete, *Struttura di rete 1*, interconnette tutti gli ISP di accesso con un *unico ISP globale di transito*. Il nostro (immaginario) ISP globale di transito è una rete di router e collegamenti che non solo copre l'intero globo, ma ha anche almeno un router prossimo a ognuno delle centinaia di migliaia di ISP di accesso.

Naturalmente sarebbe molto costoso per l'ISP globale costruire una rete così estesa. Per averne un profitto dovrebbe far pagare la connettività a ognuno degli ISP di accesso, a un prezzo che rifletta, anche se non necessariamente in modo direttamente proporzionale, la quantità di traffico che l'ISP di accesso scambia con l'ISP globale. Poiché l'ISP di accesso paga l'ISP globale di transito, l'ISP di accesso è comunemente detto **cliente** (*customer*) e l'ISP globale di transito prende il nome di **fornitore** (*provider*).

Se tuttavia un'azienda costruisse e gestisse un ISP globale che si rivelasse vantaggioso, allora altre aziende si costruirebbero il proprio ISP globale di transito e si metterebbero in competizione con quello originale. Questo ragionamento porta alla *Struttura di rete 2*, che consiste di centinaia di migliaia di ISP di accesso e più ISP globali di transito. Gli ISP di accesso preferirebbero sicuramente la struttura di rete 2 rispetto alla 1, in quanto potrebbero scegliere quale provider globale utilizzare in funzione dei costi e dei servizi che offre. Si noti, comunque, che gli ISP globali di transito devono essere interconnessi tra di loro; in caso contrario un ISP di accesso connesso a uno dei provider globali di transito non potrebbe comunicare con un ISP di accesso connesso a un altro provider globale di transito. La *struttura di rete 2* appena descritta è una gerarchia a due livelli nella quale i provider globali di transito stanno in cima alla gerarchia e gli ISP di accesso alla base. Tale struttura suppone che gli ISP globali di transito non solo siano prossimi a ogni ISP di accesso, ma che gli convenga pure esserlo. Nella realtà, sebbene alcuni ISP abbiano veramente una copertura globale impressionante e siano invece connessi a molti ISP di accesso, nessun ISP è presente in ogni città del mondo.

Al contrario, in ogni regione può esservi un **ISP regionale** al quale tutti gli ISP di accesso della regione si connettono. Ogni ISP regionale si connette all'**ISP di primo livello** (*tier-1 ISP*). Gli ISP di primo livello sono simili al nostro immaginario ISP globale di transito, ma gli ISP di primo livello veramente esistenti non sono presenti in ogni città del mondo. Esistono circa una dozzina di ISP di primo livello tra i quali troviamo Level 3 Communications, AT&T, Sprint e NTT. È interessante notare che non ne esiste alcuno che ufficialmente dichiari lo stato di ISP di primo livello. Come si suol dire: se chiedi se sei membro di un gruppo, probabilmente non lo sei.

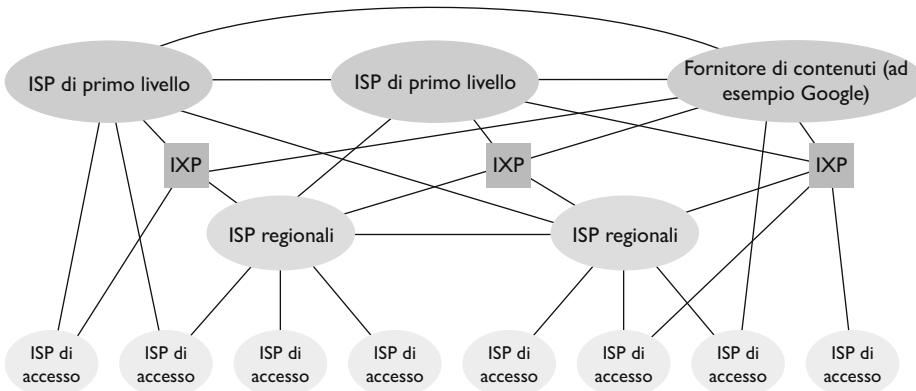
Tornando alla nostra rete di reti, non solo ci sono più ISP di primo livello in competizione, ma anche nelle regioni ci sono più ISP regionali in competizione tra di loro. In questa gerarchia ogni ISP di accesso paga l'ISP regionale a cui si connette, che a sua volta paga il suo ISP di primo livello. Un ISP di accesso può anche connettersi direttamente a un ISP di primo livello e in tal caso lo paga direttamente. Quindi c'è una relazione cliente-fornitore a ogni livello della gerarchia. Poiché gli ISP di livello 1 sono in cima alla gerarchia, essi non pagano nessuno. Per complicare ulteriormente le cose, in alcune regioni ci può essere un ISP regionale più grande, che magari copre l'intera nazione, al quale gli ISP regionali più piccoli della regione si connettono; in questo caso l'ISP regionale più grande si connette all'ISP di primo livello. Per esempio, in Cina, ci sono ISP di accesso in ogni città, che si connettono agli ISP provinciali, che a loro volta si connettono agli ISP nazionali, che infine si connettono agli ISP di primo livello [Tian 2012]. Chiameremo questa gerarchia a molti livelli, che è ancora un'approssimazione grezza dell'Internet odierna, *Struttura di rete 3*.

Per costruire una rete che sia più simile all'Internet odierna dobbiamo aggiungere alla *struttura di rete 3* i PoP, il multi-homing, il peering e gli IXP. I **PoP** (*point of presence*) esistono in tutti i livelli della gerarchia tranne che in quello degli ISP di accesso. Un PoP è semplicemente un gruppo di router vicini tra loro nella rete del provider, tramite il quale gli ISP clienti possono connettersi al fornitore. Una rete cliente si connette al PoP del fornitore affittando un collegamento ad alta velocità da un provider di telecomunicazioni terzo, collegando direttamente uno dei suoi router a un router del PoP. Qualunque ISP, tranne quelli di primo livello, può scegliere la modalità **multi-homing** (o **multi-home**) che consiste nel connettersi a due o più ISP fornitori. Per esempio, un ISP di accesso può effettuare una connessione multi-home con due ISP regionali o con due ISP regionali e un ISP di primo livello. Allo stesso modo un ISP regionale può connettersi con modalità multi-home con più ISP di primo livello. Con questa modalità un ISP può continuare a inviare e ricevere pacchetti in Internet anche se uno dei suoi fornitori è guasto.

Come abbiamo appena visto, per avere una connessione Internet globale gli ISP clienti pagano i loro ISP fornitori. Il costo riflette la quantità di traffico che l'ISP cliente scambia con il fornitore. Per ridurre tali costi, una coppia di ISP vicini e di pari livello gerarchico può fare uso di **peering**,<sup>1</sup> cioè connettere direttamente le loro reti in modo che tutto il traffico tra di esse passi attraverso una connessione diretta piuttosto che transitare da un intermediario. In questa modalità nessun ISP effettua pagamenti all'altro. Come visto prima, anche gli ISP di primo livello fanno peering tra di loro a costo zero. Una lettura piacevole su tale argomento è [Van der Berg 2008]. Utilizzando queste stesse connessioni, un'azienda terza, usando propri apparati e, di solito, un palazzo dedicato, può creare un **IXP** (*Internet exchange point*), un punto d'incontro dove più ISP possono fare peering tra di loro.

---

<sup>1</sup> Da “peer” cioè “paria”, “sullo stesso livello” (N.d.R.).



**Figura 1.15**  
Interconnessioni tra ISP.

Nell'Internet attuale ci sono circa 600 IXP [PeeringDB 2020]. Ci riferiremo a questo ecosistema, consistente in ISP di accesso, ISP regionali, ISP di primo livello, PoP, multi-homing, peering e IXP come alla *Struttura di rete 4*.

Siamo finalmente giunti alla *Struttura di rete 5*, che descrive l'Internet odier- na. La *Struttura di rete 5*, mostrata nella Figura 1.15, è costruita sulla *Struttura di rete 4* aggiungendo le reti di distribuzione di contenuti (*content provider networks*). Google è attualmente uno degli esempi di punta di tali reti. Al momento si stima che Google disponga da 19 data center principali distribuiti tra Nord America, Europa, Asia, Sud America e Australia, ognuno contenente da decine a centinaia di migliaia di server. Inoltre dispone di data center più piccoli, con poche centinaia di server, spesso collocati negli IXP. Tutti i data center di Google sono interconnessi tramite la rete privata di Google che copre l'intero globo, ma è divisa dalla Internet pubblica. La rete privata di Google trasporta traffico solo da e per i server di Google. Come mostrato nella Figura 1.15, la rete privata di Google cerca di aggirare i provider di alto livello facendo peering a costo zero con gli ISP di basso livello connettendosi a loro o direttamente o tramite IXP [Labovitz 2010]. Tuttavia, poiché molti ISP di accesso possono essere raggiunti solo tramite provider di primo livello, la rete di Google si connette anche a questi ultimi e li paga per il traffico scambiato. Creandosi la propria rete, un fornitore di contenuti non solo riduce i costi dovuti agli ISP di livello superiore, ma ha anche maggior controllo su come i suoi servizi sono erogati agli utenti finali. L'infrastruttura di rete di Google sarà descritta in mag- gior dettaglio nel Paragrafo 2.6.

Riassumendo, oggigiorno Internet, una rete di reti, è complessa e consiste di dozzine di ISP di primo livello e centinaia di migliaia di ISP di livello inferiore. Gli ISP si distinguono per la copertura geografica: alcuni di essi si estendono per continenti e oceani mentre altri si limitano a ristrette regioni. Gli ISP di livello più basso si collegano a quelli di livello superiore e questi ultimi si interconnettono tra loro. Gli utenti e i fornitori di contenuto sono clienti degli ISP di livello inferiore, mentre questi ultimi sono a loro volta clienti degli ISP di livello superiore. Ultimamente i più grandi fornitori di contenuti hanno creato le loro reti private e, quando possibile, si connettono direttamente agli ISP di livello inferiore.

## 1.4 Ritardi, perdite e throughput nelle reti a commutazione di pacchetto

Nel Paragrafo 1.1 abbiamo affermato che Internet può essere vista come un'infrastruttura che fornisce servizi alle applicazioni distribuite in esecuzione sui sistemi periferici. Idealmente, vorremmo che i servizi Internet fossero in grado di spostare una quantità di dati qualsiasi tra due sistemi periferici, istantaneamente e senza alcuna perdita di dati. Purtroppo, sebbene questo sia un nobile obiettivo, non è possibile raggiungerlo nella realtà. Le reti di calcolatori limitano necessariamente il throughput, cioè la quantità di dati al secondo che può essere trasferita tra due sistemi periferici, introducono ritardi tra questi ultimi e possono addirittura perdere pacchetti. Da un lato è una sfortuna che le leggi fisiche, oltre a limitare il throughput, introducano ritardi e perdite. Dall'altro, esistono numerosi e affascinanti modi per affrontare tali problemi; più che a sufficienza per colmare un corso sulle reti e dare vita a migliaia di tesi di dottorato. In questo paragrafo inizieremo a esaminare e a quantificare il ritardo, le perdite e il throughput nelle reti di calcolatori.

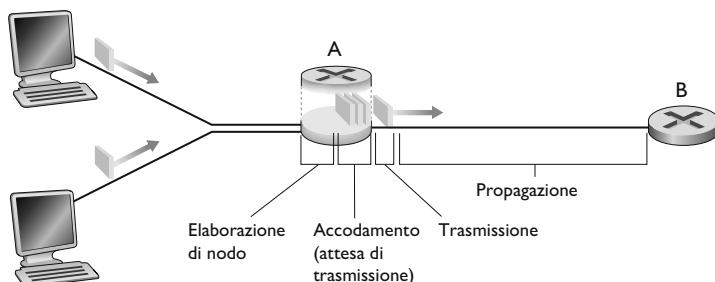
### 1.4.1 Panoramica del ritardo nelle reti a commutazione di pacchetto

Ricordiamo che un pacchetto parte da un host (la sorgente), passa attraverso una serie di router e conclude il viaggio in un altro host (la destinazione). A ogni tappa, il pacchetto subisce vari tipi di ritardo a ciascun nodo (host o router) del tragitto. Di tali ritardi, i principali sono il **ritardo di elaborazione**, il **ritardo di accodamento**, il **ritardo di trasmissione** e il **ritardo di propagazione**, che complessivamente formano il **ritardo totale di nodo** (*nodal delay*). Le prestazioni di molte applicazioni per Internet come la ricerca, la navigazione sul Web, l'e-mail, le mappe, la messaggistica istantanea e il voice-over-IP sono pesantemente influenzate dai ritardi di rete. Al fine di acquisire un'approfondita comprensione sulla commutazione di pacchetto e sulle reti di calcolatori occorre comprendere natura e importanza di questi ritardi.

#### Tipi di ritardo

Esaminiamo i ritardi descritti nel contesto della Figura 1.16. Come parte del proprio percorso dalla sorgente alla destinazione, un pacchetto viene inviato dal nodo a monte (rispetto al flusso dei dati) attraverso il router A verso il router B. Il nostro scopo è caratterizzare il ritardo di nodo presso il router A. Si

**Figura 1.16** Ritardo di nodo al router A.



noti che il collegamento in uscita dal router A verso il router B è preceduto da una coda. Quando il pacchetto arriva al router A dal nodo a monte, il router ne esamina l'intestazione per determinare il collegamento in uscita appropriato e quindi dirige il pacchetto su tale collegamento. In questo esempio, il collegamento in uscita per il pacchetto è quello che porta al router B. Un pacchetto può essere trasmesso su un collegamento solo se non ci sono altri pacchetti in fase di trasmissione e se non esistono pacchetti che lo precedono nella coda; se il collegamento è momentaneamente occupato o se altri pacchetti sono accodati, l'ultimo pacchetto arrivato si metterà in coda.

### Ritardo di elaborazione

Il tempo richiesto per esaminare l'intestazione del pacchetto e per determinare dove dirigerlo fa parte del **ritardo di elaborazione** (*processing delay*). Questo può anche includere altri fattori, tra i quali il tempo richiesto per controllare errori a livello di bit eventualmente occorsi nel pacchetto durante la trasmissione dal nodo a monte al router A. Nei router ad alta velocità questi ritardi sono solitamente dell'ordine dei microsecondi o inferiori. Dopo l'elaborazione, il router dirige il pacchetto verso la coda che precede il collegamento al router B. Nel corso del Capitolo 4 studieremo i dettagli di funzionamento di un router.

### Ritardo di accodamento

Una volta in coda, il pacchetto subisce un **ritardo di accodamento** (*queuing delay*) mentre attende la trasmissione sul collegamento. La lunghezza di tale ritardo per uno specifico pacchetto dipenderà dal numero di pacchetti precedentemente arrivati, accodati e in attesa di trasmissione sullo stesso collegamento. Se la coda è vuota e non è in corso la trasmissione di altri pacchetti, il ritardo di accodamento per il nostro pacchetto è nullo. D'altro canto, se il traffico è pesante e molti altri pacchetti stanno anch'essi aspettando la trasmissione, il ritardo di accodamento è elevato. Vedremo fra breve che il numero di pacchetti che si possono trovare in coda è funzione dell'intensità e della natura del traffico in ingresso alla coda. Nella pratica i ritardi di accodamento possono essere dell'ordine dei microsecondi o dei millisecondi.

### Ritardo di trasmissione

Assumendo che i pacchetti siano trasmessi secondo la politica first-come-first-served (il primo che arriva è il primo a essere servito), come avviene comunemente nelle reti a commutazione di pacchetto, il nostro pacchetto può essere trasmesso solo dopo la trasmissione di tutti quelli che lo hanno preceduto nell'arrivo. Sia  $L$  la lunghezza del pacchetto, in bit, e  $R$  bps la velocità di trasmissione del collegamento dal router A al router B. Il **ritardo di trasmissione** (*transmission delay*) risulta essere  $L/R$ . Questo è il tempo richiesto per trasmettere tutti i bit del pacchetto sul collegamento. Anche i ritardi di trasmissione sono di solito dell'ordine dei microsecondi o dei millisecondi.

### Ritardo di propagazione

Una volta immesso sul collegamento, un bit deve propagarsi fino al router B. Il tempo impiegato è il **ritardo di propagazione** (*propagation delay*). Il bit viaggia alla velocità di propagazione del collegamento, che dipende dal mezzo fisico (fibra ottica, doppino in rame e così via) ed è compresa nell'intervallo che va

dai  $2 \cdot 10^8$  m/s ai  $3 \cdot 10^8$  m/s, corrispondente quest'ultimo alla velocità della luce. Il ritardo di propagazione è dato da  $d/v$ , dove  $d$  è la distanza tra i due router, mentre  $v$  è la velocità di propagazione nel collegamento. Nelle reti molto estese i ritardi di propagazione sono dell'ordine dei millisecondi.

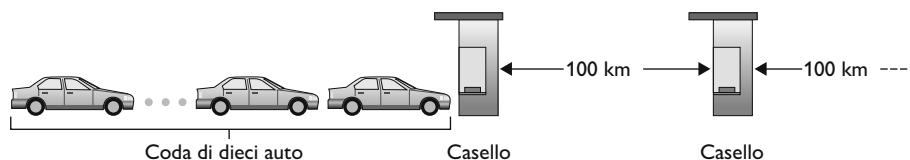
### Confronto tra ritardi di trasmissione e di propagazione

Chi affronta per la prima volta il campo delle reti di calcolatori può avere difficoltà nel comprendere la differenza tra ritardi di trasmissione e di propagazione: una differenza sottile, ma importante. Il ritardo di trasmissione è la quantità di tempo impiegata dal router per trasmettere in uscita il pacchetto, ed è funzione della lunghezza del pacchetto e della velocità di trasmissione del collegamento, ma non ha niente a che fare con la distanza tra i due router. Il ritardo di propagazione, invece, è il tempo richiesto per la propagazione di un bit da un router a quello successivo, ed è funzione della distanza tra i due router, ma non ha niente a che fare con la lunghezza del pacchetto o con la velocità di trasmissione propria del collegamento.

Per meglio chiarire la differenza ricorremo, anche questa volta, all'analogia con un'autostrada, dove i tratti (di 100 km) tra un casello e l'altro corrispondono ai collegamenti e i caselli ai router (Figura 1.17). Si supponga: (1) che le automobili viaggino (ossia si propaghino) a una velocità di 100 km/h (in altre parole, quando un'auto lascia un casello, accelera istantaneamente fino a 100 km/h e mantiene costantemente tale velocità); (2) che dieci automobili, accodate, procedano in ordine fisso (possiamo vedere ogni auto come un bit e l'insieme dei veicoli come un pacchetto); (3) che ciascun casello sia in grado di far transitare (ossia trasmettere) un'auto ogni dodici secondi (ovvero, 5 auto al minuto), e che queste siano le sole a percorrere in quel momento l'autostrada; (4) che la prima auto, una volta raggiunto il casello, prima di superarlo, attenda che gli altri nove veicoli siano allineati dietro di essa. Quindi, tutte le automobili devono raggiungere il casello prima di poter ripartire. Il tempo impiegato dal casello per far passare l'intera fila di macchine, che è di 10 auto / (5 auto/minuto) = 2 minuti, equivale al ritardo di trasmissione in un router. Il tempo richiesto a un'auto per spostarsi dall'uscita di un casello fino al casello successivo, pari a 100 km / (100 km/h) = 1 ora, corrisponde al ritardo di propagazione. Di conseguenza, il tempo che intercorre da quando l'intera coda di vetture si trova di fronte al casello di partenza fino al momento in cui raggiunge quello successivo è la somma del ritardo di trasmissione e del ritardo di propagazione, in questo caso 62 minuti.

Approfondiamo l'analisi dell'analogia. Che cosa succederebbe se il tempo di transito ai caselli fosse superiore al tempo richiesto a un'auto per spostarsi da un casello all'altro? Si supponga, per esempio, che le auto viaggino alla velocità di 1000 km/h e che il casello faccia passare le auto alla velocità di una al minuto. In questo caso il ritardo di viaggio tra due caselli sarebbe di 6 minuti,

**Figura 1.17** Analogia della coda di auto.



mentre il tempo per far defluire l'intera coda di auto sarebbe di 10 minuti, per cui le prime auto della fila arriverebbero al secondo casello prima che le ultime abbiano lasciato il primo. Questa situazione si riscontra anche nelle reti a commutazione di pacchetto: i primi bit di un pacchetto possono pervenire al router successivo mentre molti dei restanti bit del pacchetto sono ancora in attesa di essere trasmessi dal router precedente.

Se un'immagine vale mille parole, allora un'animazione ne vale milioni: la piattaforma MyLab del testo fornisce un'animazione interattiva che spiega e confronta i ritardi di trasmissione e propagazione. Invitiamo i lettori a provarla. Un'utile e piacevole lettura sui ritardi di rete è [Smith 2009].

Siano  $d_{\text{elab}}$ ,  $d_{\text{acc}}$ ,  $d_{\text{trasm}}$  e  $d_{\text{prop}}$  i ritardi di elaborazione, accodamento, trasmissione e propagazione; il ritardo totale di nodo è allora:

$$d_{\text{nodo}} = d_{\text{elab}} + d_{\text{acc}} + d_{\text{trasm}} + d_{\text{prop}}$$

Il contributo di queste componenti del ritardo può variare in modo significativo. Per esempio,  $d_{\text{prop}}$  può essere trascurato (pochi microsecondi) per un collegamento che connette due router nello stesso campus universitario; è, invece, di centinaia di millisecondi per due router interconnessi tramite un satellite geostazionario e può risultare il termine dominante in  $d_{\text{nodo}}$ . Anche il  $d_{\text{trasm}}$  può essere insignificante o molto importante. Il suo contributo è in genere trascurabile per velocità trasmissive di 10 Mbps o superiori (come avviene nelle LAN); invece, può risultare di centinaia di millisecondi per grandi pacchetti Internet inviati su collegamenti a bassa velocità con modem dial-up. Il ritardo di elaborazione,  $d_{\text{elab}}$ , è spesso trascurabile, ma può influenzare pesantemente il throughput massimo di un router, che rappresenta la velocità massima alla quale il router può inoltrare i pacchetti.

#### 1.4.2 Ritardo di accodamento e perdita di pacchetti

La componente più complessa e interessante del ritardo totale di nodo è il ritardo di accodamento,  $d_{\text{acc}}$ , tanto da aver ispirato migliaia di articoli e numerosi libri: [Bertsekas 1991, Kleinrock 1975 e 1976]. In questa sede ne forniamo solo una trattazione ad alto livello e intuitiva, ma il lettore appassionato può andarsi a consultare i libri suggeriti o magari scriverci una tesi di dottorato. A differenza degli altri tre ritardi (elaborazione, trasmissione e propagazione), quello di accodamento può variare da pacchetto a pacchetto. Per esempio, se in una coda vuota arrivano 10 pacchetti contemporaneamente, il primo pacchetto trasmesso non subirà ritardo di accodamento, mentre l'ultimo subirà un ritardo di accodamento piuttosto grande (dovendo attendere la trasmissione dei restanti 9 pacchetti). Pertanto, nel caratterizzare il ritardo di accodamento, si fa uso solitamente di misure statistiche, quali il ritardo di accodamento medio, la varianza del ritardo di accodamento e la probabilità che il ritardo di accodamento superi un valore fissato.

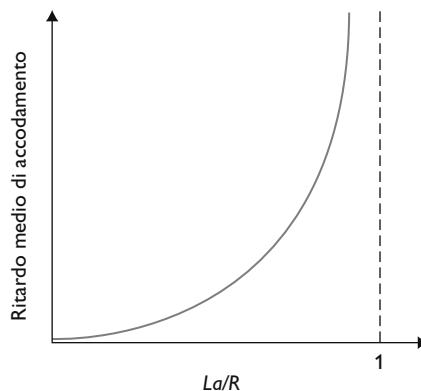
Quando si considera rilevante e quando invece trascurabile il ritardo di accodamento? La risposta dipende dalla velocità di arrivo del traffico alla coda, dalla velocità di trasmissione del collegamento e dalla natura del traffico entrante, ossia se il traffico arriva periodicamente o a raffiche. Per approfondire l'argomento, denotiamo con  $a$  la velocità media di arrivo dei pacchetti nella co-

da, espressa in pacchetti al secondo. Ricordiamo che  $R$  è la velocità di trasmissione, ossia la velocità (in bit al secondo) alla quale i bit vengono trasmessi in uscita dalla coda. Supponiamo poi, per semplicità, che tutti i pacchetti consistano di  $L$  bit. Quindi, la velocità media di arrivo dei bit in coda è di  $La$  bit/s. Infine, assumiamo che la coda possa mantenere un numero illimitato di bit. Il rapporto  $La/R$ , detto **intensità di traffico**, spesso gioca un importante ruolo nella stima dell'entità del ritardo di accodamento. Se  $La/R > 1$ , la velocità media di arrivo dei bit nella coda supera la velocità alla quale i bit vengono ritrasmessi in uscita da essa. In questa situazione sfortunata, la coda tenderà a crescere senza limiti e il ritardo di coda tenderà all'infinito. Pertanto, una delle regole auree nell'ingegneria del traffico è: *progettare il sistema in modo che l'intensità di traffico non superi 1.*

Consideriamo ora il caso  $La/R \leq 1$ . Qui la natura del traffico in arrivo influenza sul ritardo di coda. Per esempio, se i pacchetti arrivano a cadenza periodica (ossia un pacchetto ogni  $L/R$  secondi), allora ciascun pacchetto troverà una coda vuota e non ci saranno ritardi di accodamento. Se invece i pacchetti arrivano a raffiche periodiche, si possono verificare dei significativi ritardi medi di accodamento. Supponiamo, per esempio, che  $N$  pacchetti giungano simultaneamente ogni  $(L/R)N$  secondi. Allora il primo pacchetto trasmesso non subisce ritardo di accodamento; il secondo presenta un ritardo di accodamento di  $L/R$  secondi; più in generale, l' $n$ -esimo pacchetto trasmesso ha un ritardo di accodamento di  $(n - 1)L/R$  secondi. Lasciamo come esercizio il calcolo del ritardo di accodamento medio per questo esempio.

I due casi ora descritti, inerenti ad arrivi periodici, suonano un po' accademici. In genere, il processo di arrivo in coda è *casuale*. In altre parole, gli arrivi non seguono uno schema e i pacchetti sono distanziati da quantità di tempo casuali. In questa situazione più realistica, la quantità  $La/R$  di solito non è sufficiente a caratterizzare in modo completo le statistiche sui ritardi. Ciò nondimeno, essa risulta utile per intuire l'entità del ritardo di accodamento. In particolare, se l'intensità di traffico è vicina a zero, gli arrivi di pacchetti sono pochi e piuttosto distanziati, e risulta poco probabile che un pacchetto in arrivo ne trovi un altro in coda. Di conseguenza, il ritardo di accodamento medio sarà quasi nullo. Al contrario, quando l'intensità di traffico è vicina a 1, si riscontrano intervalli di tempo in cui la velocità di arrivo supera la capacità trasmissiva (il che è dovuto alla variabilità del tasso di arrivo dei pacchetti) e si forma una coda e altri in cui la capacità trasmissiva è inferiore e quindi la coda si riduce. Ciò nonostante, quando l'intensità di traffico si avvicina a 1, la lunghezza media della coda aumenta sempre più. La dipendenza qualitativa del ritardo di accodamento medio dall'intensità di traffico è mostrata nella Figura 1.18.

Un fondamentale aspetto evidenziato nella Figura 1.18 è che quanto più l'intensità di traffico si avvicina a 1, tanto più rapidamente cresce il ritardo medio di accodamento. Ossia, un piccolo incremento percentuale nell'intensità ha come risultato un incremento molto più accentuato nel ritardo. Forse avete sperimentato lo stesso fenomeno in autostrada: una strada che risulta sempre trafficata, ha intensità di traffico vicina a 1. Se qualche evento causa un lieve incremento del traffico rispetto alla norma, il ritardo che subirete può diventare enorme.



**Figura 1.18** Ritardo medio di accodamento in funzione dell'intensità di traffico.

Per capire realmente da che cosa sia determinato il ritardo di accodamento, siete incoraggiati, ancora una volta, a visitare la piattaforma MyLab del volume che fornisce un'animazione interattiva per la simulazione di una coda. Se impostate la velocità di arrivo dei pacchetti in modo che l'intensità di traffico superi 1, vedrete la coda crescere nel tempo.

### Perdita di pacchetti

Nella nostra precedente discussione abbiamo assunto che la coda sia in grado di mantenere un numero infinito di pacchetti. In realtà, le code hanno capacità finita, sebbene le capacità di accodamento dipendano fortemente dalla struttura del router e dal suo costo. Poiché la capacità delle code è finita, in realtà i ritardi dei pacchetti non tendono all'infinito quando l'intensità di traffico si approssima a 1, ma un pacchetto può trovare la coda piena. Non essendo possibile memorizzare tale pacchetto, il router lo eliminerà e il pacchetto andrà perduto. Questo “straripamento” del buffer associato alla coda (detto anche *buffer overflow*) si può vedere nell'animazione di simulazione della coda, quando l'intensità di traffico è maggiore di 1.

Dal punto di vista dei sistemi periferici, la perdita sembrerà come se il pacchetto fosse stato inviato in rete, ma non fosse più riemerso alla destinazione. La frazione di pacchetti perduti aumenta in proporzione all'intensità di traffico. Quindi, le prestazioni di un nodo sono spesso misurate non solo in termini di ritardo, ma anche della probabilità di perdita di pacchetti. Come vedremo nei prossimi capitoli, un pacchetto perduto può essere ritrasmesso per iniziativa locale, in modo da assicurare che tutti i dati vengano alla fine trasferiti dalla sorgente alla destinazione.

### 1.4.3 Ritardo end-to-end

Fin qui, la nostra trattazione si è concentrata sui ritardi di nodo, ossia sui ritardi presso un singolo router. Concludiamo la nostra discussione considerando brevemente il ritardo dalla sorgente alla destinazione (*end-to-end delay*). Per studiare questo concetto supponiamo l'esistenza di  $N - 1$  router tra l'host sorgente e quello di destinazione. Ipotizziamo, inoltre, per il momento, che la rete non sia congestionata (e che quindi i ritardi di accodamento siano trascurabili), il

ritardo di elaborazione a ciascun router e presso il mittente sia  $d_{\text{elab}}$ , la velocità di trasmissione in uscita a ogni router e all'host sorgente sia di  $R$  bps e la propagazione su ciascun collegamento sia  $d_{\text{prop}}$ . I ritardi totali di nodo si accumulano e danno un ritardo complessivo end-to-end pari a

$$d_{\text{end-to-end}} = N (d_{\text{elab}} + d_{\text{trasm}} + d_{\text{prop}}) \quad (1.2)$$

dove, ancora una volta,  $d_{\text{trasm}} = L/R$  e  $L$  è la dimensione del pacchetto. Si noti che l'Equazione (1.2) è una generalizzazione dell'Equazione (1.1) che non considerava i ritardi di elaborazione e propagazione. Lasciamo al lettore il compito di generalizzare questa formula al caso di ritardi eterogenei nei nodi e alla presenza di un ritardo medio di accodamento presso ciascun nodo.

### Traceroute

Per ottenere una misura efficiente dei ritardi in una rete di calcolatori possiamo fare uso del programma diagnostico Traceroute. Si tratta di un semplice programma eseguibile su qualsiasi host di Internet. Quando l'utente specifica il nome di un host di destinazione, il modulo dell'utente invia un certo numero di pacchetti speciali verso tale destinazione. Durante il loro percorso verso la destinazione, questi pacchetti passano attraverso una serie di router. Quando un router riceve uno di questi pacchetti speciali, invia un breve messaggio che torna all'origine. Il messaggio contiene il nome e l'indirizzo del router.

Più nello specifico, si supponga l'esistenza di  $N - 1$  router tra l'origine e la destinazione. L'origine invia  $N$  pacchetti speciali nella rete, ciascuno dei quali ha come indirizzo la destinazione ultima. Tali  $N$  pacchetti speciali sono etichettati da 1 a  $N$ , in sequenza. Quando l' $n$ -esimo router riceve il pacchetto marcato con  $n$ , invece di instradarlo verso la sua destinazione, invia un messaggio che torna verso l'origine. Quando l'host di destinazione riceve il pacchetto speciale  $N$ -esimo, anch'esso restituisce un messaggio all'origine. Esso registra il tempo intercorso tra l'invio di un pacchetto e la ricezione del corrispondente messaggio di ritorno e memorizza anche il nome e l'indirizzo del router (o del destinatario) che restituisce il messaggio. In questo modo, l'origine può ricostruire il percorso intrapreso dai pacchetti ed è inoltre in grado di determinare i ritardi di andata e ritorno per tutte le tratte. In realtà Traceroute ripete l'esperimento appena descritto tre volte: pertanto la sorgente in effetti invia 3 ( $N$  pacchetti alla destinazione). Traceroute è descritto in dettaglio nell'RFC 1393.

Viene ora presentato un esempio di output del programma Traceroute, in cui si segue il percorso dall'origine gaia.cs.umass.edu (presso la University of Massachusetts) a un host del Dipartimento di informatica dell'Università Sorbona di Parigi (precedentemente l'università era conosciuta come UPMC). L'output presenta sei colonne: la prima contiene il valore  $n$  descritto precedentemente, ossia il numero del router lungo il percorso; la seconda è il nome del router; la terza colonna è il suo indirizzo (nella forma xxx.xxx.xxx.xxx); le ultime tre colonne rappresentano i ritardi di andata e ritorno nelle tre prove dell'esperimento. Se l'origine riceve meno di tre messaggi da ogni dato router (per la perdita di pacchetti nella rete), Traceroute pone un asterisco subito dopo il numero del router e riporta meno di tre tempi di andata e ritorno per tale router.

```

1 gw-vlan-2451.cs.umass.edu (128.119.245.1) 1.899 ms 3.266 ms
  3.280 ms
2 j-cs-gw-int-10-240.cs.umass.edu (10.119.240.254) 1.296 ms
  1.276 ms 1.245 ms
3 n5-rt-1-1-xe-2-1-0.gw.umass.edu (128.119.3.33) 2.237 ms
  2.217 ms 2.187 ms
4 core1-rt-et-5-2-0.gw.umass.edu (128.119.0.9) 0.351 ms 0.392
  ms 0.380 ms
5 border1-rt-et-5-0-0.gw.umass.edu (192.80.83.102) 0.345 ms
  0.345 ms 0.344 ms
6 nox300gw1-umass-re.nox.org (192.5.89.101) 3.260 ms 0.416 ms
  3.127 ms
7 nox300gw1-umass-re.nox.org (192.5.89.101) 3.165 ms 7.326 ms
  7.311 ms
8 198.71.45.237 (198.71.45.237) 77.826 ms 77.246 ms 77.744 ms
9 renater-lb1-gw.mx1.par.fr.geant.net (62.40.124.70) 79.357
  ms 77.729 79.152 ms
10 193.51.180.109 (193.51.180.109) 78.379 ms 79.936 80.042 ms
11 * 193.51.180.109 (193.51.180.109) 80.640 ms *
12 * 195.221.127.182 (195.221.127.182) 78.408 ms *
13 195.221.127.182 (195.221.127.182) 80.686 ms 80.796 ms
  78.434 ms
14 r-upmc1.reseau.jussieu.fr (134.157.254.10) 78.399 ms *
  81.353 ms

```

Nel percorso seguito abbiamo 14 router tra la sorgente e la destinazione. La maggior parte di essi ha un nome e tutti hanno un indirizzo. Per esempio, il nome del Router 4 è `core1-rt-at-5-2-0.gw.umass.edu` e il suo indirizzo è `128.119.0.9`. Analizzando i dati forniti per questo stesso router, vediamo che nella prima delle tre prove il ritardo di andata e ritorno tra la sorgente e il router è stato di 0,351 ms. I ritardi di andata e ritorno per le successive due prove sono stati di 0,392 e 0,380 ms. Questi ritardi includono tutte le componenti di ritardo appena trattate, compresi i ritardi di trasmissione, di propagazione, di elaborazione e di accodamento.

Dato che il ritardo di accodamento varia con il tempo, il ritardo di andata e ritorno del pacchetto  $n$  inviato al router  $n$  può in realtà essere maggiore rispetto al ritardo di andata e ritorno del pacchetto  $n + 1$  inviato al router  $n + 1$ . Infatti nell'esempio si può notare che i ritardi nel Router 11 appaiono superiori ai ritardi nel Router 12. Notate anche il grande aumento del ritardo di andata e ritorno quando si passa dal Router 7 al Router 8, dovuto a un collegamento transatlantico in fibra ottica tra i due router, che causa un ritardo di propagazione relativamente grande.

Ci sono parecchi programmi software gratuiti che forniscono un'interfaccia grafica a Traceroute; uno dei nostri preferiti è PingPlotter [PingPlotter 2020].

### Sistemi periferici, applicazioni e altri ritardi

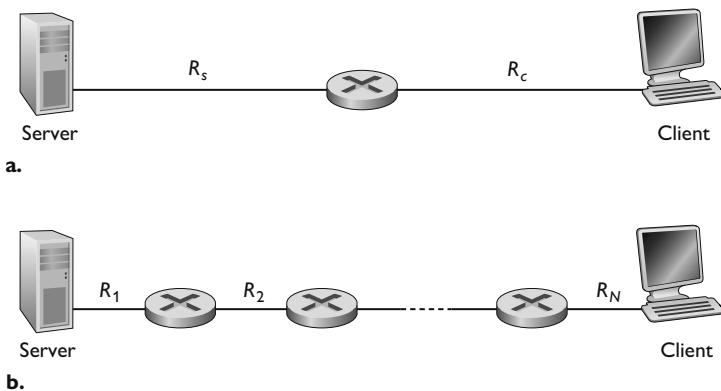
Oltre ai ritardi di elaborazione, trasmissione e propagazione, si potrebbero manifestare ulteriori, significativi, ritardi nei sistemi periferici. Per esempio, un si-

stema periferico che aspetta di trasmettere un pacchetto su un mezzo condiviso (per esempio, in uno scenario Wi-Fi o cable modem) può volontariamente ritardare la sua trasmissione come parte del suo protocollo per condividere il mezzo con altri sistemi periferici. Considereremo in dettaglio questi protocolli nel Capitolo 6. Un altro ritardo importante è quello di trasposizione in pacchetti (“pacchettizzazione”) di un flusso multimediale, presente nelle applicazioni di telefonia su IP (VoIP, *Voice-over-IP*). In VoIP il mittente deve prima di tutto riempire il pacchetto con conversazione digitalizzata e codificata, e poi inviarlo su Internet. Questo tempo per riempire un pacchetto, detto **ritardo di pacchettizzazione**, può essere significativo e avere un impatto sulla qualità della chiamata VoIP percepita dall’utente. Tali questioni verranno ulteriormente esaminate in uno dei problemi alla fine di questo capitolo.

#### 1.4.4 Throughput nelle reti di calcolatori

Oltre al ritardo e alla perdita di pacchetti, un’altra misura critica delle prestazioni in una rete di calcolatori è il throughput end-to-end. Per darne una definizione, considerate il trasferimento di un file voluminoso da A a B, attraverso la rete. Questo file potrebbe essere, per esempio, un grosso videoclip da trasmettere da un computer a un altro. Il **throughput (capacità totale) istantaneo** in ogni istante di tempo è la velocità (in bps) alla quale B sta ricevendo il file; molte applicazioni mostrano il throughput istantaneo durante il download nell’interfaccia utente. Potreste misurare il ritardo end-to-end e il throughput di download tra il vostro server e qualunque altro server nel mondo usando l’applicazione speedtest [Speedtest 2020]. Se il file consiste di  $F$  bit e il trasferimento richiede  $T$  secondi affinché B riceva tutti gli  $F$  bit, allora il **throughput medio** del trasferimento del file è di  $F/T$  bps. Per alcune applicazioni, come la telefonia su Internet, è auspicabile avere un ritardo basso e un throughput istantaneo sopra una certa soglia in modo continuativo (per esempio sopra i 24 kbps per alcune applicazioni di telefonia su IP e oltre i 256 kbps per alcune applicazioni video in tempo reale). Per altre applicazioni, comprese quelle che richiedono il trasferimento di un file, il ritardo non è critico, ma è auspicabile avere il throughput più alto possibile.

Per approfondire ulteriormente il concetto di throughput, consideriamo alcuni esempi. La Figura 1.19(a) mostra due sistemi periferici, un server e un client, connessi da due collegamenti e da un router. Si consideri il throughput per un trasferimento di file dal server al client. Sia  $R_s$  la velocità del collegamento tra il server e il router e  $R_c$  quella del collegamento tra il router e il client. Si supponga che i soli bit inviati sull’intera rete siano quelli tra il server e il client. Ci chiediamo ora, in questo scenario ideale, quale sia il throughput tra server e client. Per rispondere a questa domanda dobbiamo pensare ai bit come a un fluido e ai collegamenti come a delle condotte. Chiaramente il server non può pompare nel suo collegamento a una velocità maggiore di  $R_s$  bps e il router non può inoltrare bit a una velocità più alta di  $R_c$  bps. Se  $R_s < R_c$  i bit immessi dal server scorreranno attraverso il router e arriveranno al client a una velocità di  $R_s$  bps, dando un throughput di  $R_s$  bps. Se, dall’altro lato,  $R_c < R_s$  allora il router non sarà in grado di inoltrare i bit alla stessa velocità alla quale li riceve. In tal caso, i bit lasceranno il router a una velocità di  $R_c$ , dando un throughput end-to-end di  $R_c$ . Si noti poi



**Figura 1.19**  
Throughput per il trasferimento di un file dal server al client.

che se i bit continuano ad arrivare al router a una velocità  $R_s$  e a lasciarlo a una velocità di  $R_c$ , la quantità di bit accumulata al router in attesa di trasmissione al client cresce indefinitamente: una situazione non auspicabile. Quindi, per questa semplice rete con due collegamenti, il throughput è il  $\min(R_c, R_s)$ , cioè la velocità di trasmissione del collegamento che fa da **collo di bottiglia** (*bottleneck link*). Avendo determinato il throughput, possiamo ora stimare il tempo necessario a trasferire un grosso file di  $F$  bit dal server al client come  $F / \min(R_s, R_c)$ . Consideriamo un esempio specifico: supponete di stare scaricando un file MP3 di  $F = 32$  milioni di bit. Il server ha una velocità di trasmissione  $R_s = 2$  Mbps e voi avete un collegamento di accesso di  $R_c = 1$  Mbps. Il tempo necessario a trasferire il file è allora di 32 secondi. Queste espressioni del throughput e del tempo di trasferimento sono solo approssimazioni, in quanto non tengono conto dei ritardi di elaborazione store-and-forward e di quelli legati ai protocolli.

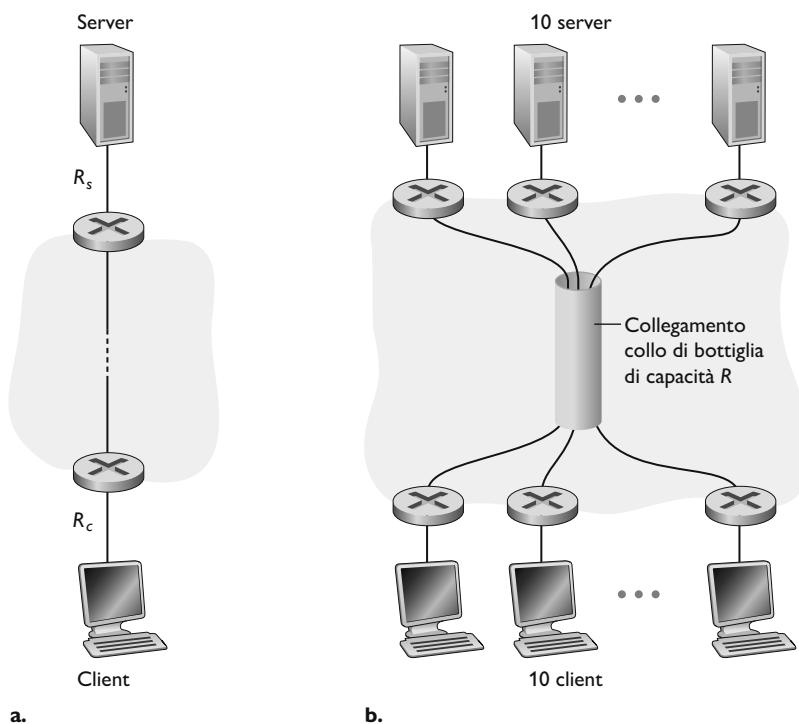
La Figura 1.19(b) mostra una rete con  $N$  collegamenti tra server e client aventi rispettivamente velocità di trasmissione  $R_1, R_2, \dots, R_N$ . Applicando la stessa analisi fatta per la rete con due collegamenti, troviamo che il throughput per un trasferimento di file dal server al client è il  $\min(R_1, R_2, \dots, R_N)$ , che è di nuovo la velocità del collegamento più lento lungo il percorso dal server al client.

Si consideri ora un altro esempio, motivato dall'odierna Internet. La Figura 1.20 (a) mostra due sistemi periferici, un server e un client, collegati a una rete di calcolatori. Si consideri il throughput per un trasferimento di file dal server al client: il server è collegato alla rete con un collegamento alla velocità di  $R_s$ , mentre il client con uno alla velocità di  $R_c$ . Supponete ora che tutti i collegamenti del nucleo della rete abbiano una velocità di trasmissione molto alta, molto più alta di  $R_s$  e  $R_c$ . In effetti, oggi, il nucleo della rete Internet è sovradimensionato con collegamenti ad alta velocità, poco congestionati. Supponete anche che i soli bit inviati globalmente nella rete siano quelli dal server al client. Dato che, in questo esempio, il nucleo della rete è come un condotto largo, la velocità alla quale i bit fluiscono dalla sorgente alla destinazione è di nuovo il minimo tra  $R_s$  e  $R_c$ , cioè il throughput è  $\min(R_s, R_c)$ . Quindi, attualmente, il fattore limitante per il throughput in Internet è tipicamente la rete di accesso.

Come ultimo esempio si consideri la Figura 1.20 (b), nella quale ci sono 10 server e 10 client collegati al nucleo della rete. In questo esempio stanno avvenendo 10 download contemporanei, che coinvolgono 10 coppie client-server.

**Figura 1.20**

Throughput end-to-end:  
 (a) il client scarica un file dal server; (b) 10 client scaricano da 10 server.



Si supponga che questi 10 download siano il solo traffico sulla rete in questo momento. Come mostrato nella figura c'è un collegamento nel nucleo della rete che viene attraversato da tutti i 10 download. Sia  $R$  la velocità di trasmissione di questo collegamento. Supponiamo che tutti i collegamenti di accesso ai server abbiano la stessa velocità  $R_s$ , che tutti i collegamenti di accesso ai client abbiano la stessa velocità  $R_c$  e che le velocità di trasmissione di tutti i collegamenti del nucleo – eccetto l'unico collegamento comune a velocità  $R$  – siano molto più alte di  $R$ ,  $R_s$  ed  $R_c$ . Ci chiediamo ora quali siano i throughput dei download. Chiaramente, se la velocità del collegamento comune  $R$  è grande, diciamo un centinaio di volte più grande di  $R_s$  e  $R_c$ , allora il throughput di ogni download sarà ancora una volta  $\min(R_s, R_c)$ . Ma che cosa accade se la velocità del collegamento comune è dello stesso ordine di grandezza di  $R_s$  e  $R_c$ ? Come sarebbe il throughput in questo caso? Esaminiamo un esempio specifico. Supponete che  $R_s = 2 \text{ Mbps}$ ,  $R_c = 1 \text{ Mbps}$  e  $R = 5 \text{ Mbps}$  e che il collegamento comune suddivida la propria velocità di trasmissione equamente tra i 10 download. Quindi, il collo di bottiglia di ciascun download non è più nella rete di accesso, ma è invece il collegamento condiviso nel nucleo, che fornisce solo 500 kbps di throughput a ciascun download. Pertanto, il throughput end-to-end di ciascun download è ora ridotto a 500 kbps.

Gli esempi nelle Figure 1.19 e 1.20(a) mostrano che il throughput dipende dalla velocità di trasmissione dei collegamenti sui quali passano i dati. Abbiamo visto che, quando non c'è altro traffico che interviene, il throughput può essere semplicemente approssimato alla velocità di trasmissione minima lungo il percorso tra la sorgente e la destinazione. L'esempio della Figura 1.20 (b) mostra

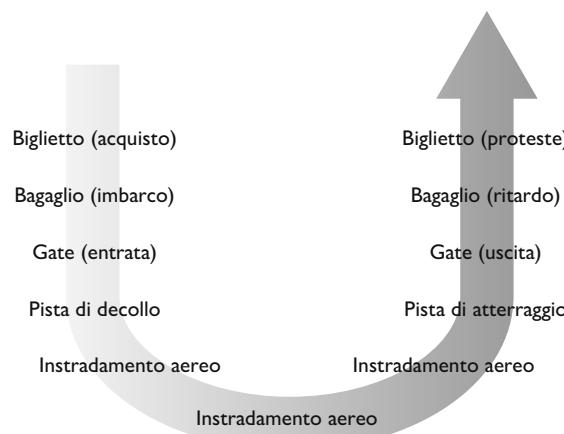
che, più in generale, il throughput dipende non solo dalla velocità di trasmissione dei collegamenti lungo il percorso, ma anche dal traffico sulla rete. In particolare, un collegamento con una velocità di trasmissione buona potrebbe essere il collo di bottiglia per un trasferimento di file, qualora molti altri flussi di dati passassero anch'essi attraverso quel collegamento. Esamineremo il throughput nelle reti di calcolatori più da vicino nei problemi di fine capitolo e nei capitoli successivi.

## 1.5 Livelli dei protocolli e loro modelli di servizio

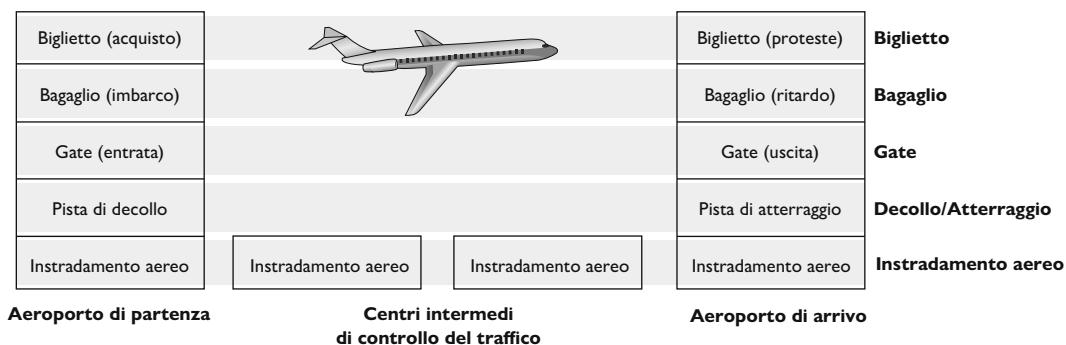
Giunti a questo punto, Internet appare come un sistema estremamente complicato. Abbiamo visto che è costituita da molti pezzi: numerose applicazioni e protocolli, vari tipi di sistemi periferici, router e svariate tipologie di mezzi trasmissivi per i collegamenti. Data questa enorme complessità, esiste una qualche speranza di organizzare l'architettura delle reti, o quanto meno di strutturare una discussione sull'argomento? Fortunatamente, la risposta a entrambe le domande è affermativa.

### 1.5.1 Architettura a livelli

Prima di tentare di organizzare i nostri pensieri sull'architettura di Internet cerchiamo un'analogia con la nostra vita. Nella realtà noi trattiamo sistemi complessi in qualsiasi momento della nostra vita quotidiana. Immaginate, per esempio, che vi chiedano di descrivere il sistema di una linea aerea nel suo complesso, composto da biglietteria, controllo bagagli, personale ai gate, piloti, aeroplani, controllo del traffico aereo e sistemi internazionali di instradamento dei velivoli. Un modo per descrivere tale sistema consiste nel definire la serie di azioni compiute da voi o da altri, quando si vola con una linea aerea: acquistare il biglietto, consegnare i bagagli, andare al gate d'imbarco e infine raggiungere l'aeroplano. Il velivolo decolla e si dirige verso la propria destinazione. Dopo l'atterraggio, scendete a terra e ritirate i bagagli. Se durante il viaggio qualcosa è andato storto, protestate con il personale preposto (non ottenendo nulla nonostante i vostri sforzi). Un simile scenario è mostrato nella Figura 1.21.



**Figura 1.21** Viaggio aereo: azioni.



**Figura 1.22** Stratificazione orizzontale delle funzionalità di una linea aerea.

Possiamo già notare alcune analogie con le reti di calcolatori: la linea aerea vi trasporta da una località di partenza a una di destinazione, così come in Internet un pacchetto viene trasportato da un host sorgente a uno di destinazione. Ma questo non è sufficiente. Nella Figura 1.21 noi andiamo alla ricerca di una struttura. Guardando la figura, notiamo l'esistenza di una funzione di biglietteria a ciascuna estremità; esiste inoltre una funzione inherente i bagagli per i passeggeri già dotati di biglietto, e una funzione al gate per i passeggeri con biglietto e bagaglio già imbarcato. Per i passeggeri che hanno già superato il gate, esistono le funzioni di decollo e atterraggio, e mentre si vola è in funzione l'instrandamento aereo. Ciò suggerisce la possibilità di guardare le funzionalità della Figura 1.21 in modo orizzontale, come mostrato nella Figura 1.22.

La Figura 1.22 presenta le funzionalità di una linea aerea divise in livelli, o strati, fornendo una cornice per la trattazione di un viaggio aereo. Si noti che ciascun livello, combinato con quelli inferiori, implementa delle funzionalità, dei *servizi*. A livello di biglietteria e a livelli inferiori, si ha trasferimento di persone dall'ufficio viaggiatori di una compagnia a un altro. A livello di bagagli e a livelli inferiori, si ha trasferimento di persone e bagagli dall'imbarco al ritiro del bagaglio. Si noti che tale livello fornisce questo servizio solo a persone già provviste di biglietto. A livello di gate, si ha trasferimento di persone e bagagli dal gate di partenza a quello di arrivo. A livello di decollo/atterraggio, si verifica trasferimento di persone e bagagli dalla pista di decollo alla pista d'atterraggio. Ogni livello fornisce il proprio servizio (1) effettuando determinate azioni all'interno del livello (per esempio, a livello di gate, l'entrata e l'uscita dei passeggeri dall'aereo) e (2) utilizzando i servizi del livello immediatamente inferiore (proseguendo nell'esempio, l'utilizzo del servizio di trasferimento dei passeggeri dalla pista di decollo alla pista d'atterraggio proprio del livello di decollo/atterraggio).

Un'architettura a livelli consente di discutere una parte specifica e ben definita di un sistema articolato e complesso. Questa stessa semplificazione ha un valore considerevole grazie all'introduzione della modularità, che rende molto più facile cambiare l'implementazione del servizio fornito da un determinato livello. Fino a quando il livello fornisce lo stesso servizio allo strato superiore e utilizza gli stessi servizi dello strato inferiore, la parte rimanente del sistema rimane invariata al variare dell'implementazione del livello. Si noti la differenza

tra cambiare l'implementazione di un servizio e variare il servizio stesso. Per esempio, se le funzioni di gate fossero variate (imbarcando e sbarcando le persone sulla base dell'altezza), la parte restante del sistema descritto non cambierebbe, dato che il livello di gate fornirebbe la stessa funzionalità (imbarco e sbarco delle persone); si limiterebbe a implementare la funzione in modo differente. Nel caso di sistemi grandi e complessi, che vengono costantemente aggiornati, la capacità di cambiare l'implementazione di un servizio senza coinvolgere altre componenti del sistema costituisce un ulteriore importante vantaggio legato alla stratificazione.

### Stratificazione dei protocolli

Dopo aver parlato di linee aeree, rivolgiamo ora la nostra attenzione ai protocolli di rete. Per dare struttura alla loro progettazione, i progettisti organizzano i protocolli e l'hardware e software che li implementano in **livelli o strati** (*layer*). Ciascun protocollo appartiene a uno dei livelli, così come ogni funzione nell'architettura di una linea aerea della Figura 1.22 apparteneva a un livello. Ancora una volta siamo interessati ai **servizi** offerti da un livello a quello superiore: si tratta del cosiddetto **modello di servizio** (*service model*) di un livello. Proprio come nel caso dell'esempio citato, ogni livello fornisce il suo servizio (1) effettuando determinate azioni all'interno del livello stesso e (2) utilizzando i servizi del livello immediatamente inferiore. Per esempio, i servizi offerti dal livello  $n$  possono includere la consegna affidabile dei messaggi da un lato della rete all'altro. Ciò può essere implementato utilizzando un servizio di consegna dei messaggi non affidabile da lato a lato, fornito dal livello  $n - 1$  e aggiungendo a livello  $n$  la funzionalità di determinare e ritrasmettere i messaggi persi.

Un livello di protocolli può essere implementato via software, hardware o con una combinazione dei due. I protocolli a livello di applicazione, quali HTTP e SMTP, sono quasi sempre implementati via software nei sistemi periferici; e così è anche per i protocolli a livello di trasporto. Dato che i livelli fisico e di data link si occupano della comunicazione su un collegamento specifico, sono di regola implementati nella scheda di rete associata (per esempio schede di rete Ethernet e Wi-Fi). Il livello di rete è spesso un'implementazione mista di hardware e software. Così come le funzioni nell'architettura stratificata della linea aerea erano distribuite tra i vari aeroporti e centri di controllo del volo che costituivano il sistema, anche un protocollo di livello  $n$  è distribuito tra sistemi periferici, commutatori di pacchetto e altre componenti che formano la rete. In pratica, sovente si trova una parte di protocollo di livello  $n$  in ciascuna delle parti che costituiscono la rete.

La stratificazione dei protocolli presenta vantaggi concettuali e strutturali [RFC 3439]. Come abbiamo visto, fornisce un modo strutturato per trattare i componenti dei sistemi. La modularità rende più facile aggiornare la componentistica. Ricordiamo, tuttavia, che alcuni ricercatori e ingegneri operanti nel campo delle reti si oppongono con fermezza alla stratificazione [Wakeman 1992]. Un eventuale svantaggio legato alla stratificazione è la possibilità che un livello duplichli le funzionalità di quello inferiore. Per esempio, molte pile di protocolli forniscono meccanismi di correzione degli errori ai livelli sia di collegamento sia di connessione end-to-end. Un secondo potenziale svantaggio è che la funzionalità a un livello possa richiedere informazioni (per esempio, un

**Figura 1.23** La pila di protocolli Internet.



valore di natura temporale) presenti solo in un altro livello; ciò viola lo scopo insito nella separazione dei livelli.

Considerati assieme, i protocolli dei vari livelli sono detti **pila di protocolli** (*protocol stack*). La pila di protocolli di Internet, mostrata in Figura 1.23 consiste di cinque livelli: fisico, collegamento, rete, trasporto e applicazione. Se esaminate l'indice, vedrete che abbiamo pressappoco organizzato questo libro usando i livelli della pila di protocolli di Internet. Adottiamo un **approccio top-down**, trattando prima il livello di applicazione e poi procedendo verso il basso.

### Livello di applicazione

Il livello di applicazione (*application layer*) è la sede delle applicazioni di rete e dei relativi protocolli. Per quanto riguarda Internet, tale livello include molti protocolli, quali HTTP (che consente la richiesta e il trasferimento dei documenti web), SMTP (che consente il trasferimento dei messaggi di posta elettronica) e FTP (che consente il trasferimento di file tra due sistemi remoti). Vedremo che determinate funzioni di rete, quali la traduzione di nomi di host (per esempio, [www.ietf.org](http://www.ietf.org)) in indirizzi di rete a 32 bit, vengono anch'esse effettuate con l'aiuto di un protocollo a livello di applicazione, il DNS (*domain name system*). Nel prossimo capitolo vedremo quanto sia facile costruire protocolli a livello di applicazione.

Un protocollo a livello di applicazione è distribuito su più sistemi periferici: un'applicazione in un sistema periferico, tramite il protocollo, scambia pacchetti di informazioni con l'applicazione in un altro sistema periferico. Faremo riferimento a questi pacchetti di informazione a livello applicativo come a **messaggi**.

### Livello di trasporto

Il livello di trasporto (*transport layer*) di Internet trasferisce i messaggi del livello di applicazione tra punti periferici gestiti dalle applicazioni. In Internet troviamo due protocolli di trasporto: TCP e UDP. TCP fornisce alle applicazioni un servizio orientato alla connessione,<sup>2</sup> che include la consegna garantita dei messaggi a livello di applicazione alla destinazione e il controllo di flusso

---

<sup>2</sup> Con tale servizio l'utente deve stabilire una connessione, usarla e quindi rilasciarla. Molto spesso, gli approcci orientati alla connessione garantiscono la consegna dei dati e ne preservano anche l'ordine; questo è il caso anche di TCP (*N.d.R.*).

(ossia la corrispondenza tra le velocità di mittente e destinatario). Inoltre, TCP fraziona i messaggi lunghi in segmenti più piccoli e fornisce un meccanismo di controllo della congestione, in modo che una sorgente regoli la propria velocità trasmissiva quando la rete è congestionata. Il protocollo UDP fornisce alle proprie applicazioni un servizio non orientato alla connessione che è davvero un servizio senza fronzoli, senza affidabilità, né controllo di flusso e della congestione. Nel testo chiameremo **segmenti** i pacchetti a livello di trasporto.

### Livello di rete

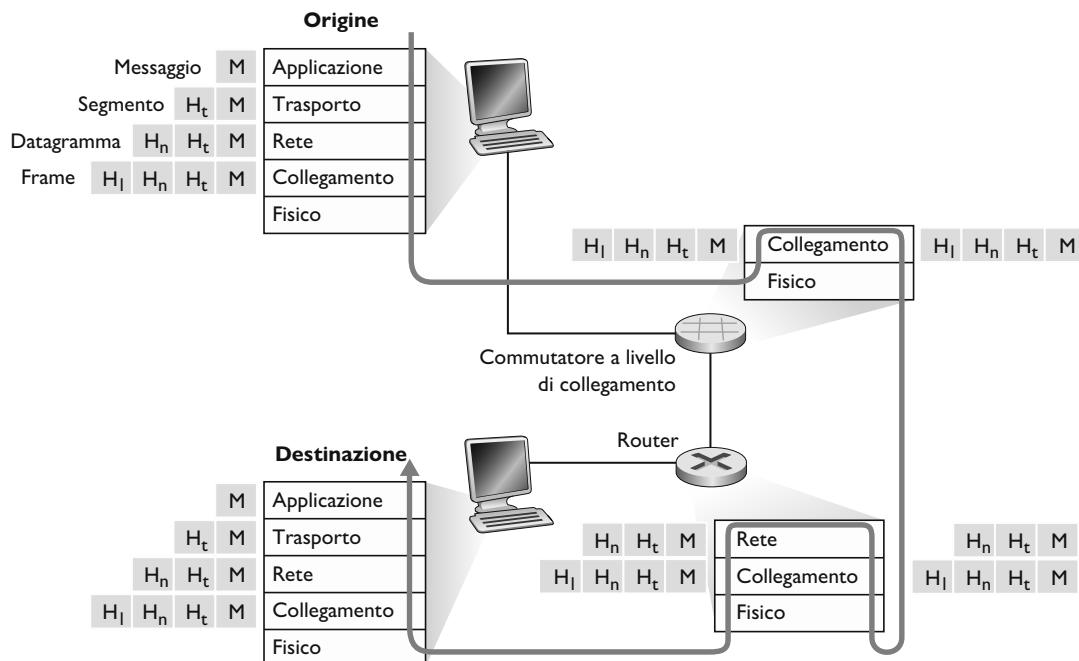
Il livello di rete (*network layer*) di Internet si occupa di trasferire i pacchetti a livello di rete, detti **datagrammi**, da un host a un altro. Il protocollo Internet a livello di trasporto (TCP o UDP) in un host di origine passa al livello sottostante un segmento e un indirizzo di destinazione, esattamente come la consegna di una lettera all'ufficio postale. Il livello di rete mette poi a disposizione il servizio di consegna del segmento al livello di trasporto nell'host di destinazione.

Il livello di rete di Internet comprende il famoso protocollo IP, che definisce i campi dei datagrammi e come i sistemi periferici e i router agiscono su tali campi. Esiste un solo protocollo IP; tutti gli apparati di Internet che presentano un livello di rete lo devono supportare. Il livello di rete di Internet contiene, inoltre, svariati protocolli di instradamento che determinano i percorsi che i datagrammi devono seguire tra la sorgente e la destinazione. Come abbiamo visto nel Paragrafo 1.3, Internet è una rete di reti, e ognuna di esse può scegliere il proprio protocollo di instradamento. Sebbene il livello di rete contenga sia il protocollo IP sia numerosi protocolli di instradamento, esso viene spesso detto semplicemente livello IP, per riflettere il fatto che IP è il collante che tiene unita Internet.

### Livello di collegamento

Il livello di rete di Internet instrada un datagramma attraverso una serie di router tra la sorgente e la destinazione. Per trasferire un pacchetto da un nodo (host o router) a quello successivo sul percorso, il livello di rete si affida ai servizi del livello di collegamento. In particolare, a ogni nodo, il livello di rete passa il datagramma al livello sottostante, che lo trasporta al nodo successivo. In questo nodo, il livello di collegamento passa il datagramma al livello di rete superiore.

I servizi forniti dal livello di collegamento dipendono dallo specifico protocollo utilizzato. Per esempio, alcuni protocolli garantiscono la consegna affidabile, ossia dal nodo che trasmette al nodo che riceve su un collegamento. Si noti che tale servizio di consegna affidabile è diverso da quello omonimo del TCP, che fornisce consegna affidabile da un sistema periferico a un altro. Esempi di livello di collegamento includono Ethernet, Wi-Fi e il protocollo di accesso alla rete DOCSIS. Dato che i datagrammi in genere devono attraversare diversi collegamenti nel loro viaggio dalla sorgente alla destinazione, un datagramma potrebbe essere gestito da differenti protocolli a livello di collegamento lungo le diverse tratte che costituiscono il suo percorso. Per esempio, un datagramma potrebbe essere gestito da Ethernet in un collegamento e da PPP in quello successivo. Il livello di rete riceverà un servizio diverso da ciascuno dei diversi protocolli a livello di collegamento. In questo testo chiameremo **frame** i pacchetti a livello di collegamento.



**Figura 1.24** Host, router e commutatori a livello di collegamento: ciascuno contiene i livelli adeguati alle sue funzionalità.

### Livello fisico

Mentre il compito del livello di collegamento è spostare interi frame da un elemento della rete a quello adiacente, il ruolo del livello fisico (*physical layer*) è trasferire i singoli bit del frame da un nodo a quello successivo. Anche i protocolli di questo livello sono dipendenti dal collegamento e in più dipendono dall'effettivo mezzo trasmittivo (per esempio, doppino o fibra ottica). Per citare un esempio, Ethernet presenta vari protocolli a livello fisico: uno per il doppino intrecciato, uno per il cavo coassiale, uno per la fibra ottica e via dicendo. In ciascuno di tali casi, i bit sono trasferiti lungo il collegamento secondo differenti modalità.

### 1.5.2 Incapsulamento

La Figura 1.24 mostra il percorso seguito dai dati scendendo lungo la pila di protocolli del sistema mittente, risalendo e scendendo lungo le pile di protocolli dei commutatori e dei router che intervengono a livello di collegamento e, infine, risalendo la pila nel sistema ricevente. Come vedremo più avanti nel testo, i router e i commutatori a livello di collegamento sono tutti commutatori di pacchetto. Al pari dei sistemi periferici, organizzano il proprio hardware e software di rete a livelli. In ogni caso non implementano tutti i livelli della pila di protocolli, ma solo quelli inferiori. Come mostrato nella Figura 1.24, i commutatori a livello di collegamento implementano i livelli 1 e 2, mentre i router implementano i livelli da 1 a 3. Ciò significa, per esempio, che i router Internet sono in grado di interpretare il protocollo IP (che è di livello 3), mentre i commutatori a livello di collegamento non possono farlo. Vedremo più avanti che, mentre non riconoscono gli indirizzi IP, sono però in grado di riconoscere gli indirizzi di livello 2, quali gli indirizzi Ethernet. Si osservi come gli host implementano

tutti i cinque livelli; ciò è coerente con l'idea che l'architettura Internet ponga la maggior parte della sua complessità alla periferia della rete.

La Figura 1.24 illustra anche l'importante concetto di **incapsulamento**. Presso un host mittente, un **messaggio a livello di applicazione** (*application-layer message*) M viene passato a livello di trasporto. Nel caso più semplice, questo livello prende il messaggio e gli concatena informazioni aggiuntive (le cosiddette informazioni di intestazione a livello di trasporto,  $H_t$  nella Figura 1.24) che saranno utilizzate dalla parte ricevente del livello di trasporto. Il messaggio a livello di applicazione e le informazioni di intestazione a livello di trasporto costituiscono il **segmento a livello di trasporto** (*transport-layer segment*) che incapsula il messaggio a livello di applicazione. Le informazioni aggiunte potrebbero includere dati che consentono al livello di trasporto lato ricevente di consegnare il messaggio all'applicazione desiderata, o potrebbero inoltre includere bit per il rilevamento degli errori che consentono al ricevente di determinare l'eventuale cambiamento di alcuni bit del messaggio durante il percorso. Il livello di trasporto, quindi, passa il segmento al livello di rete, che aggiunge informazioni di intestazione proprie del livello di rete ( $H_n$ ), quali gli indirizzi dei sistemi periferici di sorgente e di destinazione, andando così a creare un **datagramma a livello di rete** (*network-layer datagram*). A questo punto, il datagramma viene passato al livello di collegamento, il quale aggiunge le proprie informazioni di intestazione creando un **frame a livello di collegamento** (*link-layer frame*). Quindi, a ciascun livello, il pacchetto ha due tipi di campi: quello di intestazione e quello di **payload** (il carico utile trasportato). Il payload è tipicamente un pacchetto proveniente dal livello superiore.

Per comprendere meglio le fasi di tale processo risulta utile fare ricorso a un'analogia con la procedura di invio di una circolare tra uffici tramite il servizio postale ordinario. Supponete che Alice, che si trova in un dato ufficio, voglia mandare una circolare a Bob, che lavora in un altro ufficio. La circolare, che rappresenta il messaggio a livello di applicazione, viene posta in una busta per la comunicazione tra uffici, sulla quale Alice indica il nome del destinatario, Bob, e il reparto in cui lavora. La busta per la comunicazione tra uffici, che riporta le informazioni di intestazione (ossia il nome del destinatario e il suo dipartimento) e contiene il messaggio corrispondente al livello di applicazione (la circolare), è analoga al segmento a livello di trasporto. Lo smistamento della corrispondenza dell'ufficio di Alice prende in consegna la busta, la pone in un'ulteriore busta, conforme ai requisiti richiesti dal servizio postale pubblico, su cui scrive l'indirizzo dell'ufficio mittente e quello dell'ufficio destinatario. In questo senso la busta del servizio postale è analoga al datagramma, in quanto incapsula il segmento a livello di trasporto (la busta per la comunicazione tra uffici), che a sua volta incapsula il messaggio originario (la circolare). L'ufficio postale consegna la busta allo smistamento dell'ufficio destinatario, dove inizia il processo di de-incapsulamento: qui il plico è aperto e viene estratta la circolare imbustata. Questa è inoltrata al destinatario, Bob, che apre la busta per la comunicazione tra uffici ed estrae la circolare.

Il processo di incapsulamento può essere più complesso rispetto a quello descritto sopra. Per esempio, un messaggio grande può essere diviso in più segmenti a livello di trasporto (i quali possono a loro volta essere divisi ciascuno in più datagrammi a livello di rete). Al momento della ricezione, tale segmento deve essere ricostruito a partire dai datagrammi costitutivi.

## 1.6 Reti sotto attacco

Internet è diventata oggi uno strumento importante per molte istituzioni, comprese grandi e piccole imprese, università e pubblica amministrazione. Molte persone, inoltre, fanno affidamento su Internet per alcune delle loro attività professionali, sociali e personali. Miliardi di “cose”, quali dispositivi domestici e quelli indossabili, vengono connessi a Internet. Ma dietro questi servizi di pubblica utilità si nasconde un lato oscuro dove dei “ragazzacci” tentano di devastare la nostra vita quotidiana, danneggiando i nostri computer connessi a Internet, violando la nostra privacy e rendendo inutilizzabili i servizi Internet dai quali dipendiamo.

Il campo della sicurezza di rete si occupa di come i malintenzionati possono attaccare le reti o del modo in cui, noi che siamo prossimi a diventare esperti in reti, possiamo difenderle da questi attacchi o, meglio ancora, progettare nuove architetture che siano, in primo luogo, immuni da attacchi di questo tipo. Data la frequenza e la varietà degli attacchi, tanto di quelli esistenti quanto l'avvento di quelli nuovi e più distruttivi, la sicurezza di rete è diventata negli ultimi anni un argomento centrale nel campo delle reti di calcolatori. Una delle caratteristiche di questo libro di testo è quella di portare in prima linea gli aspetti inerenti la sicurezza di rete.

Dato che non abbiamo ancora esperienza delle reti di calcolatori e protocolli di Internet, inizieremo con una panoramica di quelli che sono attualmente i più frequenti problemi legati alla sicurezza. Questo stuzzicherà la curiosità di un'analisi più approfondita nei capitoli successivi. Iniziamo qui a chiederci che cosa sia andato storto, in che modo le reti di calcolatori siano vulnerabili e quali sono oggi i più frequenti tipi di attacco.

### Malware installati sugli host tramite Internet

Colleghiamo i dispositivi a Internet perché vogliamo ricevere/inviare dati tramite la rete. Ovviamente desideriamo avere a che fare con contenuti “buoni”, come post Instagram, pagine web, messaggi di posta elettronica, chiamate telefoniche, video in tempo reale, risultati dei motori di ricerca e così via. Ma, sfortunatamente, assieme a questi arrivano anche i contenuti “cattivi” – noti come **malware** – che possono penetrare nei nostri dispositivi e infettarli. Questi possono procedere a effettuare le azioni più tremende: cancellare i nostri file, installare spyware che raccolgono le nostre informazioni private, come il nostro numero di carta di credito o le password, e poi inviare tutto ciò (via Internet ovviamente) ai malintenzionati. Il nostro host compromesso può anche essere reclutato in una rete di migliaia di dispositivi compromessi in modo analogo, che complessivamente prende il nome di **botnet** e che i malintenzionati controllano e usano per distribuire e-mail spazzatura (spam) o per sferrare attacchi di negazione del servizio distribuiti (discussi tra breve).

Molti dei malware presenti oggi sono auto replicanti: una volta che hanno infettato un host, da quell'host cercano riferimenti riguardanti altri host su Internet e dai nuovi host infettati cercano riferimenti di ulteriori host. In questo modo un **malware auto-replicante** può diffondersi a velocità esponenziale.

## Attacchi ai server e all'infrastruttura di rete

Un'ampia classe di minacce alla sicurezza può essere classificata come attacchi **di negazione del servizio (DoS, denial-of-service)**. Come suggerisce il nome, un attacco DoS rende inutilizzabile dagli utenti legittimi una rete, un host o un'altra parte di infrastruttura. Web server di posta elettronica, DNS (discussi nel Capitolo 2) e reti istituzionali possono essere tutti soggetti ad attacchi DoS. Gli attacchi DoS Internet sono estremamente comuni e se ne verificano a migliaia ogni anno. Il sito Digital Attack Map [DAM 2020] permette di visualizzare i principali attacchi DoS giornalmente effettuati nel mondo. Molti attacchi DoS su Internet ricadono all'interno di tre categorie.

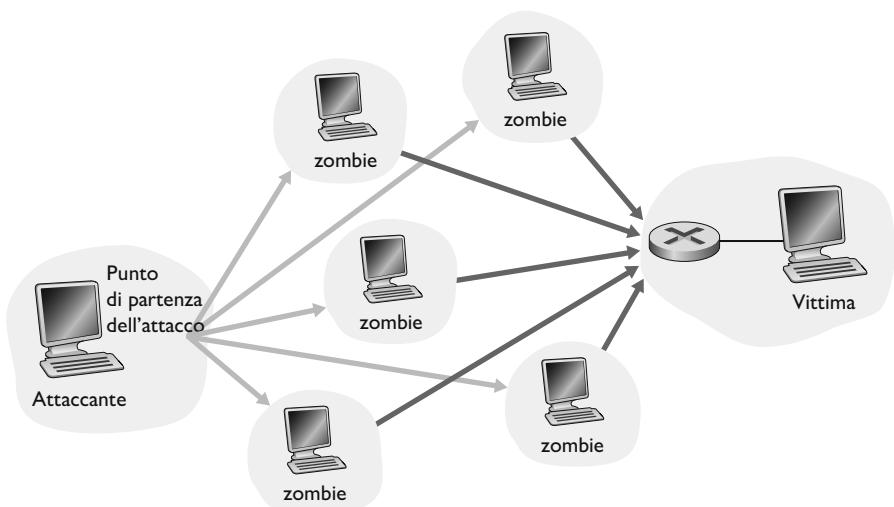
- *Attacchi alla vulnerabilità dei sistemi.* Questo comporta l'invio di pochi messaggi, ben costruiti, a un'applicazione vulnerabile o a un sistema operativo in esecuzione sull'host bersaglio. Se viene inviata la sequenza corretta di pacchetti il servizio può fermarsi o, ancora peggio, l'host può spegnersi.
- *Bandwidth flooding* (inondazione di banda). L'attaccante invia un “diluvio” di pacchetti all'host bersaglio, così tanti che il suo collegamento di accesso viene ostruito, impedendo ai pacchetti legittimi di raggiungere il server.
- *Connection flooding* (inondazione di connessioni). L'attaccante stabilisce un gran numero di connessioni TCP completamente o solo parzialmente aperte (le connessioni TCP verranno discusse al Capitolo 3) all'host bersaglio. L'host può così “ingorgarsi” con queste connessioni senza poter accettare le connessioni valide.

Analizziamo ora l'attacco di bandwidth flooding in maggior dettaglio. Ricordiamo la nostra analisi del ritardo e delle perdite nel Paragrafo 1.4.2: è evidente che se il server ha una velocità di accesso di  $R$  bps, l'attaccante avrà bisogno di mandare traffico a una velocità approssimativamente uguale a  $R$  bps per causare danni. Se  $R$  è molto grande, una singola sorgente di attacco può non essere in grado di generare traffico sufficiente per danneggiare il server. Inoltre, se tutto il traffico provenisse da una singola sorgente, un router a monte potrebbe essere in grado di individuare l'attacco e bloccare tutto il traffico da quella sorgente, prima che arrivi vicino al server. Negli attacchi DoS distribuiti (DDoS), illustrati nella Figura 1.25, l'attaccante controlla più sorgenti, e ciascuna sorgente attacca il bersaglio con del traffico. Con tale approccio, il traffico aggregato di tutte le sorgenti controllate avrà bisogno di essere approssimativamente pari a  $R$  per paralizzare il servizio. Gli attacchi DDoS, che fanno leva sulle botnet con migliaia di host compromessi, sono oggi un evento comune [DAM 2020]. Risulta più difficile individuare e difendersi da attacchi DDoS che da un attacco DoS sferrato da un singolo host. Riflettete sulla seguente questione mentre procedete nel testo: come possono i progettisti di reti difendersi dagli attacchi DoS? Vedremo che sono necessarie difese diverse per i tre tipi di attacchi DoS.

## Analisi del traffico

Molti utenti accedono oggi a Internet attraverso dispositivi wireless, quali computer portatili collegati tramite Wi-Fi o dispositivi portatili con connessioni cellulari (Capitolo 7, disponibile in inglese sulla piattaforma MyLab). Se da un lato l'accesso a Internet da qualsiasi luogo è estremamente comodo e permette

**Figura 1.25** Un attacco DoS distribuito.



nuove e straordinarie applicazioni per gli utenti mobili, dall’altro crea una grave vulnerabilità alla sicurezza. Un ricevitore passivo in prossimità di un trasmettitore wireless può ottenere una copia di ogni pacchetto trasmesso. Questi pacchetti possono contenere ogni tipo di informazione sensibile, compresi password, codici fiscali, strategie commerciali e messaggi personali. Un ricevitore passivo, che memorizza una copia di ciascun pacchetto che transita, è detto anche **packet sniffer**.

I *packet sniffer* possono essere ugualmente utilizzati anche in un ambiente cablato. In un ambiente cablato con distribuzione dei dati in broadcast, come molte LAN Ethernet, lo sniffer può ottenere copie di tutti i pacchetti inviati sulla LAN. Come descritto nel Paragrafo 1.2, anche la tecnologia di accesso via cavo invia in broadcast i pacchetti, che sono quindi vulnerabili all’analisi. Inoltre, un malintenzionato, che riesca ad accedere al router di accesso di un’istituzione o al collegamento di accesso a Internet, può installare uno sniffer che effettua una copia di tutti i pacchetti che entrano ed escono dall’organizzazione. I pacchetti raccolti dallo sniffer possono essere successivamente studiati per estrarne le informazioni sensibili.

I software di analisi dei pacchetti sono disponibili sia gratuitamente, in vari siti web, sia come prodotti commerciali. Sovente i docenti di corsi di reti assegnano esercizi di laboratorio inerenti la scrittura di programmi di analisi dei pacchetti e di ricostruzione dei dati a livello applicativo. In effetti, le esercitazioni che fanno uso di Wireshark [Wireshark 2020] proposte in questo testo (si veda il Paragrafo Esercitazioni Wireshark alla fine del presente capitolo) utilizzano esattamente questo tipo di analizzatori di pacchetti.

Dato che i *packet sniffer* sono passivi, cioè non immettono pacchetti sul canale, sono difficili da individuare; quindi, quando inviamo un pacchetto sul canale wireless, dobbiamo accettare la possibilità che qualche malintenzionato possa effettuarne delle copie. Come avrete indovinato, una tra le migliori difese contro gli analizzatori di pacchetti è costituita dalla crittografia. Nel Capitolo 8, disponibile in inglese sulla piattaforma MyLab, esamineremo come questa venga applicata alla sicurezza di rete.

## Mascheramento

È sorprendentemente facile (avrete le conoscenze su come fare appena procedete nel testo) creare un pacchetto con un indirizzo sorgente, contenuto e indirizzo del destinatario qualsiasi e poi trasmettere questo pacchetto “fatto a mano” su Internet, che doverosamente inoltrerà il pacchetto a destinazione. Immaginate il ricevitore, che non sospetta nulla (per esempio, un router di Internet), che riceve questo tipo di pacchetto, prende l’indirizzo della sorgente (falso) come veritiero e poi esegue alcuni comandi messi all’interno del contenuto del pacchetto (diciamo che modifica la sua tabella di inoltro). La capacità di immettere pacchetti in Internet con un indirizzo sorgente falso è nota come **IP spoofing**, ed è uno dei molti modi attraverso i quali un utente può spacciarsi per un altro.

Per risolvere tale problema abbiamo bisogno di autenticare il punto terminale della comunicazione (*end-point authentication*), cioè di un meccanismo che ci permetta di determinare con certezza se il messaggio ha avuto origine da dove supponiamo l’abbia avuta. Ancora una volta vi invitiamo a pensare man mano che procederete nei capitoli del libro come ciò possa avvenire per le applicazioni di rete e i protocolli.

Illustreremo i meccanismi per *end-point authentication* nel Capitolo 8, disponibile in inglese sulla piattaforma MyLab.

A conclusione di questo paragrafo vale la pena considerare in primo luogo per quale motivo Internet sia diventata così poco sicura. La risposta, essenzialmente, è che Internet è stata originariamente progettata proprio così, basata sul modello “gruppo di utenti mutuamente fidati collegati a una rete trasparente” [Blumenthal 2001]. Un modello in cui, per definizione, la sicurezza è scontata. Molti aspetti dell’architettura originale di Internet riflettono profondamente questa nozione di mutua fiducia. Per esempio, il comportamento abituale prevede che un utente possa inviare un pacchetto a un qualsiasi altro senza chiedere il permesso. E anche l’identità di un utente è abitualmente considerata buona senza bisogno di autenticazione.

Oggi Internet certamente non coinvolge più utenti mutuamente fidati. Tuttavia, gli utenti hanno pur sempre bisogno di comunicare. E anche quando non si fidano l’uno dell’altro, potrebbero voler comunicare anonimamente o attraverso terze parti (per esempio proxy web, che studieremo nel Capitolo 2, o agenti che assistono nella mobilità, che vedremo nel Capitolo 7, disponibile in inglese sulla piattaforma MyLab) e potrebbero non fidarsi dell’hardware e del software o anche dell’etere attraverso cui comunicano. Sappiamo di avere molte sfide legate alla sicurezza che incontreremo man mano che procederemo nel testo: dovremo cercare difese contro analisi, mascheramento, attacchi DDoS, malware e altro ancora. Dovremo tenere a mente che la comunicazione tra utenti mutuamente fidati è un’eccezione piuttosto che una regola. Benvenuti nel modo delle moderne reti di calcolatori!

## 1.7 Storia delle reti di calcolatori e di Internet

I paragrafi precedenti hanno offerto una panoramica della tecnologia delle reti di calcolatori e di Internet. Ora probabilmente siete in grado di impressionare amici e parenti. In ogni caso, se volete fare un figurone alla prossima festa, dovrete arricchire il vostro discorso con qualche curiosità sull’affascinante storia di Internet [Segaller 1998].

### 1.7.1 Sviluppo della commutazione di pacchetto: 1961-1972

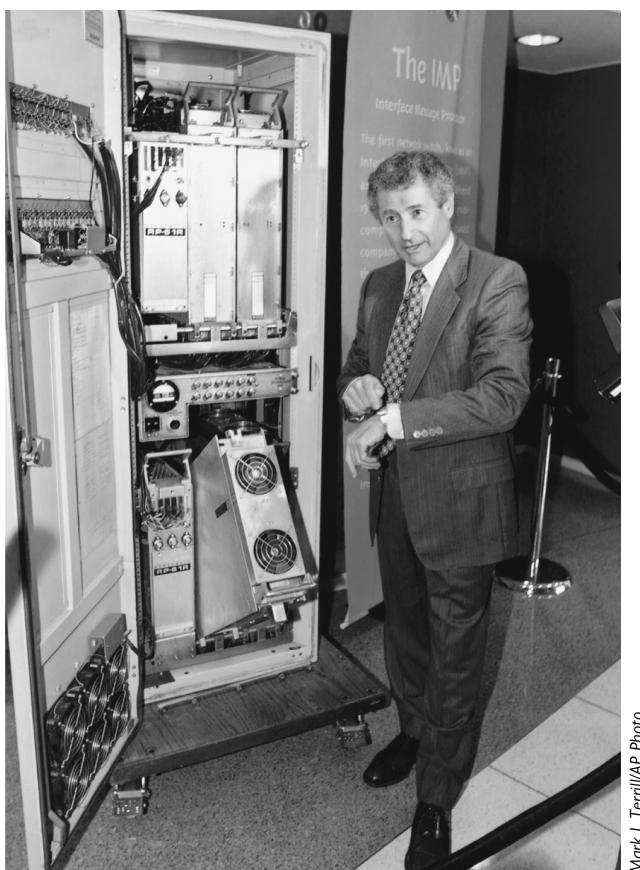
Le reti di calcolatori e l'odierna Internet hanno origine nei primi anni '60, quando la rete telefonica rappresentava la rete di comunicazioni predominante nel mondo. Ricordiamo, dal Paragrafo 1.3, che la rete telefonica utilizza la commutazione di circuito per trasmettere informazioni da un mittente a un destinatario in quanto la voce viene trasmessa a velocità costante. Data la crescente importanza dei calcolatori nei primi anni '60 e l'avvento dei sistemi operativi **time-sharing**, è stato forse naturale chiedersi come collegare i calcolatori in modo che questi potessero essere condivisi da utenti geograficamente distanti. Il traffico generato da tali utenti è facile si componga di una serie di raffiche, ossia intervalli di attività, quali l'invio di un comando a un calcolatore remoto, seguiti da periodi di inattività durante i quali si attende una risposta o si leggono i risultati.

Tre differenti gruppi di ricerca nel mondo, ciascuno dei quali ignaro del lavoro altrui [Leiner 1998], cominciarono a lavorare sulla commutazione di pacchetto considerandola un'efficiente e robusta alternativa alla comunicazione di circuito. Il primo lavoro pubblicato sulle tecniche di commutazione di pacchetto fu quello di Leonard Kleinrock [Kleinrock 1961, Kleinrock 1964], all'epoca studente del MIT. Utilizzando la teoria delle code, il lavoro di Kleinrock dimostrò l'efficacia dell'approccio a commutazione di pacchetto per sorgenti di traffico intermittenti. Nel 1964, Paul Baran [Baran 1964] aveva iniziato a investigare, presso il Rand Institute, l'uso della commutazione di pacchetto per il traffico vocale sicuro nelle reti militari. Intanto presso il National Physical Laboratory (NPL) in Inghilterra, Donald Davies e Roger Scantlebury stavano sviluppando le proprie idee sulla commutazione di pacchetto.

I lavori presso MIT, Rand e NPL gettarono le basi per l'odierna Internet. Ma Internet ha anche una lunga storia di approcci del tipo "prima costruire e poi dimostrare", che risale ai primi anni '60. J. C. R. Licklider [DEC 1990] e Lawrence Roberts, entrambi colleghi di Kleinrock presso il MIT, continuarono a portare avanti un progetto informatico all'Advanced Research Project Agency (ARPA). Roberts pubblicò un progetto complessivo per ARPAnet [Roberts 1967], la prima rete a commutazione di pacchetto, diretto antenato dell'odierna Internet pubblica.

Durante il Labor Day del 1969, venne installato un IMP, uno dei primi commutatori di pacchetto, al campus della University of California di Los Angeles (UCLA), sotto la supervisione di Kleinrock, e poco dopo altri tre commutatori furono installati presso lo Stanford Research Institute (SRI), il campus di Santa Barbara della University of California e alla University of Utah (Figura 1.26). Il primo antenato di Internet, alla fine del 1969, contava quattro nodi. Kleinrock ricorda il primissimo impiego della rete per effettuare un accesso remoto, da UCLA allo SRI, che causò un crash di sistema [Kleinrock 2004].

Nel 1972 ARPAnet aveva già 15 nodi e fu oggetto di una dimostrazione pubblica a opera di Robert Kahn durante la conferenza internazionale sulle comunicazioni via computer di quell'anno. Il primo protocollo tra nodi ARPAnet, noto come NCP (*network-control protocol*), era stato completato [RFC 001]. Una volta reso disponibile un protocollo punto a punto, si potevano cominciare a scrivere le applicazioni. Nello stesso anno Ray Tomlinson realizzò il primo programma di posta elettronica.



**Figura 1.26** Un IMP, uno dei primi commutatori di pacchetto.

### 1.7.2 Reti proprietarie e internetworking: 1972-1980

ARPAnet era inizialmente una rete singola e chiusa. Per comunicare con un calcolatore di ARPAnet era necessario essere collegati a uno dei suoi IMP. Nel primo lustro degli anni '70 nacquero altre reti a commutazione di pacchetto, tra cui: ALOHAnet, una rete a microonde che collegava le università delle isole Hawaii [Abramson 1970], come pure la rete satellitare a pacchetti DARPA [RFC 829] e la rete radio a pacchetti [Kahn 1978]; Telenet, una rete commerciale a commutazione di pacchetto di BBN basata sulla tecnologia ARPAnet; Cyclades, una rete francese a commutazione di pacchetto introdotta da Louis Pouzin [Think 2002]; reti time-sharing quali Tymnet e GE Information Services, alla fine degli anni '60 e nei primi anni '70 [Schwartz 1977]; SNA di IBM (1969-1974), che si poneva in parallelo rispetto ad ARPAnet [Schwartz 1977].

Il numero di reti andava crescendo. In retrospettiva possiamo dire che i tempi erano maturi per lo sviluppo di un'architettura complessiva di connessione delle reti. Il lavoro "primordiale" di interconnessione delle reti (sponsorizzato da DARPA), consistente in sostanza nella creazione di una *rete di reti*, fu svolto da Vinton Cerf e Robert Kahn [Cerf 1974]. E per descrivere tale lavoro venne coniato il termine *internetting*.

Tali principi architetturali furono implementati nel protocollo TCP. Le sue prime versioni, piuttosto diverse rispetto a quella odierna, combinavano una consegna di dati affidabile e in sequenza con ritrasmissione (che fa tuttora parte di TCP) con funzioni di inoltro (che al giorno d'oggi sono effettuate da IP). I primi esperimenti con TCP, combinati con il riconoscimento dell'importanza di un servizio di trasporto punto a punto inaffidabile e senza controllo di flusso per applicazioni quali la telefonia, portarono alla separazione di IP da TCP e allo sviluppo del protocollo UDP. I tre protocolli di Internet fondamentali che conosciamo al giorno d'oggi – TCP, IP e UDP – vennero ideati alla fine degli anni '70.

Oltre alla ricerca relativa a Internet operata da DARPA, erano in corso numerose altre attività riguardo le reti. Nelle Hawaii, Norman Abramson stava sviluppando ALOHAnet, una rete radio a pacchetti che consentiva a vari siti delle isole Hawaii di comunicare tra loro. Il protocollo ALOHA [Abramson 1970] rappresentò il primo protocollo ad accesso multiplo, che consentiva a utenti geograficamente distribuiti di condividere un mezzo di comunicazione (una frequenza radio). Metcalfe e Boggs partirono dal lavoro sul protocollo ad accesso multiplo di Abramson quando svilupparono il protocollo Ethernet [Metcalfe 1976] per reti broadcast cablate e condivise. Venticinque anni fa, ben prima della rivoluzione dei PC e dell'esplosione delle reti, Metcalfe e Boggs stavano gettando le basi per le moderne LAN.

### 1.7.3 La proliferazione delle reti: 1980-1990

Alla fine degli anni '70 ARPAnet collegava circa duecento host. Dieci anni dopo, il numero di host, collegati in una confederazione di reti che assomigliava molto all'odierna Internet, avrebbe raggiunto i 100.000. Gli anni '80 rappresentarono un momento di enorme espansione.

La maggior parte di tale crescita derivò da vari sforzi distinti per creare reti di calcolatori che collegassero assieme le università. BITNET consentì il trasferimento di posta elettronica e di file tra università nel nord-est degli Stati Uniti. CSNET (*computer science network*) venne formata al fine di collegare i ricercatori che non avevano accesso ad ARPAnet. Nel 1986, venne creata NSFNET per fornire accesso ai supercalcolatori sponsorizzati da NSF. Inizialmente con una dorsale da 56 kbps, entro la fine del decennio NSFNET avrebbe raggiunto la velocità di 1,5 Mbps e avrebbe rappresentato la dorsale principale per il collegamento delle reti regionali.

Nella comunità ARPAnet stavano trovando posto molti degli elementi costitutivi dell'odierna architettura Internet. Il primo gennaio 1983 vide il rilascio ufficiale di TCP/IP come il nuovo protocollo standard degli host ARPAnet (andando a sostituire il protocollo NCP). La transizione da NCP a TCP/IP [RFC 801] rappresentò una giornata campale: entro quel giorno, tutti gli host dovevano trasferire dati usando TCP/IP. Verso la fine degli anni '80 furono fatte importanti estensioni a TCP per implementare il controllo di congestione basato sugli host [Jacobson 1988]. Si sviluppò inoltre il DNS, usato per associare i nomi Internet leggibili (quali `gaia.cs.umass.edu`) agli indirizzi a 32 bit di IP [RFC 1034].

In parallelo allo sviluppo di ARPAnet (che fondamentalmente fu un'iniziativa statunitense), nei primi anni '80 la Francia avviò il progetto Minitel, un piano ambizioso per portare la trasmissione dati nelle case di tutti. Sponsorizzato

dal governo francese, il sistema Minitel consisteva in una rete pubblica a commutazione di pacchetto (basata sui protocolli X.25), con server Minitel e terminali economici con modem a bassa velocità. Il Minitel ottenne un vasto successo nel 1984 quando il governo francese assegnò un terminale a ogni famiglia che lo desiderava. I siti Minitel includevano siti gratuiti, quali l'elenco telefonico, e privati, che richiedevano una quota a seconda dell'uso da parte di ciascun utente. Al suo culmine, nella metà degli anni '90, il sistema offriva più di 20.000 servizi che spaziavano dall'home banking ai database per ricerche specialistiche. Il Minitel si trovava in gran parte delle case francesi dieci anni prima che la maggior parte degli americani avesse mai sentito parlare di Internet.

#### **1.7.4 Esplosione di Internet: gli anni '90**

Gli anni '90 furono testimoni di molti eventi che simboleggiavano la continua evoluzione e l'imminente sviluppo commerciale di Internet. ARPAnet, progenitrice di Internet, cessò di esistere. Nel 1991, NSFNET lasciò decadere le proprie restrizioni per usi a scopi commerciali. Nel 1995 NSFNET sarebbe stata fermata e il traffico della dorsale Internet sarebbe stato trasportato da provider commerciali.

L'evento principale degli anni '90, tuttavia, fu la nascita del World Wide Web, che portò Internet nelle case e negli uffici di milioni di persone nel mondo. Il Web svolgeva anche la funzione di piattaforma per sviluppare e diffondere centinaia di nuove applicazioni che noi oggi diamo per scontate tra cui i motori di ricerca (per esempio Google e Bing), il commercio su Internet (per esempio Amazon ed eBay) e i social network (come Facebook).

Il Web fu ideato al CERN da Tim Berners-Lee tra il 1989 e il 1991 [Berners-Lee 1989], e si basava sulle intuizioni scaturite dai primi lavori sugli ipertesti che vanno dagli anni '40, a opera di Vannevar Bush [Bush 1945], fino agli anni '60, a opera di Ted Nelson [Xanadu 2012]. Berners-Lee e i suoi collaboratori svilupparono le prime versioni di HTML, di HTTP, di web server e di browser, ossia i quattro componenti chiave del Web. Alla fine del 1992 c'erano circa 200 web server in funzione, semplici precursori di quello che poi sarebbe avvenuto. Più o meno in quel tempo numerosi ricercatori stavano sviluppando browser web con interfaccia grafica; tra essi ricordiamo Marc Andreessen, che insieme a Jim Clark fondò Mosaic Communications, che sarebbe in seguito diventata Netscape Communications Corporation [Cusumano 1998; Quittner 1998]. Nel 1995, gli studenti universitari usavano quotidianamente Mosaic e Netscape per navigare sul Web. In tale periodo piccole e grandi compagnie iniziarono a installare web server e a gestire il commercio elettronico. Nel 1996, Microsoft cominciò a produrre un suo browser ed ebbe inizio la guerra dei browser tra Netscape e Microsoft, vinta da quest'ultima pochi anni più tardi [Cusumano 1998].

La seconda metà degli anni '90 rappresentò un periodo di enorme crescita e innovazione per Internet, periodo in cui grandi aziende e centinaia di piccole imprese crearono prodotti e servizi Internet. Entro la fine del millennio Internet supportava centinaia di applicazioni, tra cui ricordiamo le quattro più diffuse.

- La posta elettronica, tenendo conto degli allegati e la possibilità di consultarla via Web.
- Il Web, che include navigazione e commercio elettronico su Internet.

- La messaggistica istantanea con liste di contatti.
- Condivisione di file peer-to-peer (in particolare MP3), il cui precursore fu Napster.

Aspetto interessante, le prime due applicazioni provenivano dalla comunità di ricerca, mentre le ultime due vennero create da giovani intraprendenti.

Il periodo che va dal 1995 al 2001 rappresentò un giro sulle montagne russe per Internet nei mercati finanziari. Prima ancora di cominciare a produrre guadagni, centinaia di nuove imprese Internet cominciarono a essere quotate in borsa. Molte compagnie furono valutate miliardi di dollari senza presentare significativi flussi di guadagni. Le azioni Internet crollarono nel 2000-2001, e molte nuove imprese fallirono. Ciò nondimeno, alcune aziende emersero come i grandi vincitori nel settore Internet e tra queste ricordiamo Microsoft, Cisco, Yahoo, eBay, Google e Amazon.

### 1.7.5 Il nuovo millennio

Nei primi due decenni del XXI secolo forse nessun'altra tecnologia ha trasformato la società più di Internet, insieme agli smartphone connessi a Internet. Le innovazioni si susseguono a ritmo frenetico. Si stanno facendo passi avanti su tutti i fronti, inclusi l'installazione di router più veloci, e velocità di trasmissione sempre più alte sia nelle reti di accesso sia nelle dorsali. I seguenti sviluppi meritano tuttavia una particolare attenzione.

- Dall'inizio del millennio abbiamo assistito a una sempre maggiore penetrazione dell'accesso a Internet residenziale a larga banda, non solo tramite modem via cavo e DSL, ma anche attraverso la fibra ottica e ora l'accesso 5G fixed wireless, come discusso nel Paragrafo 1.2. L'accesso a Internet ad alta velocità sta gettando le basi per nuove applicazioni video, inclusa la diffusione di contenuti video generati dagli utenti (YouTube), la televisione e lo streaming video on-demand (Netflix) e la videoconferenza multiutente (Skype, Facetime e Google Hangouts).
- La sempre maggiore presenza di reti wireless ad alta velocità non solo sta rendendo possibile la connessione permanente, ma sta anche consentendo nuovi e interessanti servizi basati sulla posizione quali Yelp, Tinder e Waz. Il numero di dispositivi mobili ha superato nel 2011 quello dei dispositivi fissi. L'accesso veloce a Internet ha portato alla rapida diffusione di computer palmari (iPhone, Android, iPad, e così via) dovuta alla loro possibilità di essere sempre liberamente connessi a Internet.
- Le social network come Facebook, Instagram, Twitter e WeChat, molto popolare in Cina, hanno creato enormi reti di persone che li usano per scambiarsi messaggi o fotografie. Oggigiorno la vita di molti utenti di Internet si basa principalmente su uno o più social network. Attraverso le loro API, le reti sociali on-line creano una piattaforma per nuove applicazioni di rete quali i pagamenti e i giochi distribuiti.
- Come discusso nel Paragrafo 1.3.3 i fornitori di servizi on-line come Google e Microsoft hanno installato le loro reti private estese che non solo connettono i loro data center distribuiti in tutto il mondo, ma sono anche usate per

aggirare il più possibile Internet facendo direttamente peering con gli ISP di basso livello. In questo modo Google fornisce i risultati delle ricerche e l'accesso alle e-mail quasi istantaneamente, come se i data center si trovassero sul computer personale degli utenti.

- Molte aziende commerciali su Internet stanno ora eseguendo le loro applicazioni nel “cloud”, come EC2 di Amazon, Azure di Microsoft o Cloud di Alibaba. Anche molte aziende e università hanno migrato le loro applicazioni, per esempio le e-mail e il Web, sul cloud. Le aziende che forniscono cloud non solo forniscono applicazioni scalabili, ma anche ambienti di memorizzazione e un accesso隐式 alle loro reti private ad alte prestazioni.

## 1.8 Riepilogo

In questo capitolo abbiamo trattato un'enorme quantità di materiale! Abbiamo dato uno sguardo alle varie parti hardware e software che costituiscono Internet in particolare e le reti di calcolatori in generale. Siamo partiti dalla periferia della rete, considerando i sistemi periferici e le applicazioni, nonché il servizio di trasporto fornito alle applicazioni in funzione su di essi. Abbiamo analizzato le tecnologie a livello di collegamento e i mezzi fisici che di solito costituiscono le reti di accesso. Ci siamo poi occupati in modo più approfondito dell'interno e del nucleo della rete, identificando nella commutazione di pacchetto e di circuito i due approcci di base per trasportare i dati attraverso una rete di telecomunicazioni, esaminandone i loro punti di forza e di debolezza. Abbiamo inoltre esaminato la struttura globale di Internet, giungendo alla conclusione che Internet è una rete di reti. Abbiamo visto come la struttura gerarchica di Internet, costituita da ISP di livello superiore e inferiore, le abbia consentito di crescere fino a includere migliaia di reti.

Nella seconda parte del capitolo abbiamo esaminato altri importanti argomenti nel campo delle reti di calcolatori. Per prima cosa abbiamo analizzato le cause di ritardo, di throughput e di perdita di pacchetti nelle reti a commutazione di pacchetto. Abbiamo sviluppato semplici modelli quantitativi per i ritardi di trasmissione, di propagazione e di accodamento, come pure di throughput; faremo un uso intensivo di tali modelli nel paragrafo “Domande e problemi”. Abbiamo poi esaminato la stratificazione dei protocolli e i modelli di servizio, principi chiave dell'architettura delle reti cui faremo riferimento nel corso dell'intero libro. Abbiamo considerato una panoramica dei più diffusi attacchi alla sicurezza dell'odierna Internet. Abbiamo concluso la nostra introduzione con una breve storia delle reti di calcolatori. Il primo capitolo costituisce di per sé un mini-corso di networking.

Pertanto, se vi sentite un po' confusi, non preoccupatevi. Nei capitoli successivi rivisiteremo tutte le idee, trattandole in modo molto più approfondito (è una promessa, non una minaccia!). A questo punto ci auguriamo di avervi fornito una conoscenza generale delle parti che costituiscono una rete, una proprietà di linguaggio sulle reti e un desiderio sempre crescente di imparare di più sulla materia. Questo sarà il nostro compito per il resto del libro.

## Linee guida del testo

Prima di iniziare un viaggio occorrerebbe sempre dare un’occhiata a una cartina stradale per prendere confidenza con le principali strade che si dovranno percorrere. Nel nostro caso la destinazione ultima è una comprensione profonda di che cosa siano, come operino e quali finalità abbiano le reti di calcolatori. Il nostro itinerario segue la sequenza dei capitoli di questo libro:

1. Reti di calcolatori e Internet
2. Livello di applicazione
3. Livello di trasporto
4. Livello di rete: piano dei dati (*data plane*)
5. Livello di rete: piano di controllo (*control plane*)
6. Livello di collegamento e reti locali
7. Reti mobili e wireless (disponibile in inglese sulla piattaforma MyLab)
8. Sicurezza nelle reti di calcolatori (disponibile in inglese sulla piattaforma MyLab)

I Capitoli dal 2 al 6, che rappresentano le cinque parti fondamentali del testo, sono organizzati intorno ai quattro livelli alti della pila a cinque livelli dei protocolli di Internet. Il nostro viaggio comincerà dalla cima della pila di protocolli Internet, ossia dal livello di applicazione, e proseguirà verso il basso. L’idea che sta alla base di questo viaggio top-down è che, una volta comprese le applicazioni, potremo capire i servizi di rete necessari a supportarle. Esamineremo, quindi, uno per volta, i diversi modi in cui i servizi possono essere implementati da un’architettura di rete. Trattare subito le applicazioni fornisce infatti motivazioni e materiale per il resto del libro.

La seconda metà del testo (i Capitoli 7 e 8, disponibili in inglese sulla piattaforma MyLab) analizza in dettaglio due aspetti di fondamentale importanza (e in qualche modo indipendenti) nelle odierne reti di calcolatori. Nel Capitolo 7 esaminiamo le reti wireless e mobili, incluse le LAN wireless (compresi Wi-Fi e Bluetooth), le reti di telefonia cellulare (compreso 4G e 5G). Nel corso del Capitolo 8 tratteremo per prima cosa le basi di cifratura e sicurezza di rete, per poi esaminare come la teoria di base venga applicata in un contesto Internet su ampia scala.

## Domande e problemi

---

### Domande di revisione

#### PARAGRAFO 1.1

- R1.** Qual è la differenza tra host e sistema periferico? Elencate i tipi di sistemi periferici. Un web server è un sistema periferico?
- R2.** Il termine *protocollo* viene spesso utilizzato per descrivere relazioni diplomatiche. Come descrive Wikipedia un protocollo diplomatico?
- R3.** Perché gli standard sono importanti nei protocolli?

#### PARAGRAFO 1.2

- R4.** Elencate quattro tecnologie di accesso classificandole come residenziali, aziendali o geografiche.
- R5.** La velocità trasmissiva di HFC è dedicata o condivisa? Si possono verificare collisioni in un canale HFC di downstream? Per quali motivi?
- R6.** Elencate le possibili tecnologie di accesso nella vostra città. Per ciascun tipo di accesso, fornite la velocità di downstream e upstream pubblicizzate e il prezzo mensile.
- R7.** Qual è la velocità trasmissiva delle LAN Ethernet?

- R8.** Elencate alcuni dei mezzi fisici utilizzati da Ethernet.
- R9.** Per gli accessi residenziali vengono utilizzati modem HFC, DSL e FTTH. Per ciascuna di tali tecnologie, elencate un intervallo di velocità trasmissiva e indicate se si tratta di velocità condivise o dedicate.
- R10.** Descrivete e confrontate le più comuni odierne tecnologie di accesso wireless a Internet.

### PARAGRAFO 1.3

- R11.** Supponete di avere un solo commutatore di pacchetto tra un host di invio e uno di ricezione. La velocità di trasmissione tra l'host d'invio e il commutatore e tra il commutatore e l'host di ricezione sono rispettivamente  $R_1$  e  $R_2$ . Nell'ipotesi che il commutatore adotti la commutazione di pacchetto store-and-forward, qual è il ritardo totale da un capo all'altro per inviare un pacchetto di lunghezza  $L$ ? Si ignorino i ritardi di accodamento, di propagazione e di elaborazione.
- R12.** Quale vantaggio presenta una rete a commutazione di circuito rispetto a una a commutazione di pacchetto? Quali vantaggi ha TDM rispetto a FDM in una rete a commutazione di circuito?
- R13.** Supponete che gli utenti condividano un collegamento a 2 Mbps e che ciascun utente richieda 1 Mbps quando trasmette, ma che ciascuno trasmetta solo il 20% del tempo (si veda il confronto tra commutazione di pacchetto e di circuito nel Paragrafo 1.3).
- Quando si usa la commutazione di circuito, quanti utenti vengono supportati?
  - Per il resto del problema supponete di usare la commutazione di pacchetto. Perché non vi è essenzialmente ritardo di accodamento prima del collegamento, se due utenti o meno trasmettono contemporaneamente? Perché si verifica un ritardo di accodamento se gli utenti che trasmettono contemporaneamente sono tre?
  - Calcolate la probabilità che un dato utente stia trasmettendo.
  - Supponete che vi siano tre utenti: trovate la probabilità che, in ogni istante, tutti i tre utenti trasmettano contemporaneamente e trovate le frazioni di tempo durante le quali la coda cresce.
- R14.** Perché due ISP di pari livello nella gerarchia fanno spesso peering tra loro? Come fa un IXP a guadagnare?
- R15.** Alcuni fornitori di servizi hanno creato reti private. Si descriva la rete di Google. Quali sono le motivazioni che li spingono a crearsi le proprie reti?

### PARAGRAFO 1.4

- R16.** Considerate l'invio di un pacchetto da un host a un altro lungo un percorso fisso ed elencate le componenti di ritardo nel ritardo complessivo. Quali sono costanti e quali variabili?
- R17.** Provate l'animazione interattiva che confronta il ritardo di propagazione e quello di trasmissione sulla piattaforma MyLab del testo. Tra le velocità di trasmissione, il ritardo di propagazione e le dimensioni dei pacchetti disponibili, trovate una combinazione per le quali il mit-

tente finisce di trasmettere prima che il primo bit del pacchetto raggiunga il ricevente. Trovate un'altra combinazione per la quale il primo bit del pacchetto raggiunga il destinatario prima che il mittente abbia terminato di trasmettere.

- R18.** Quanto tempo impiega un pacchetto di lunghezza di 1000 byte per propagarsi su un collegamento lungo 2500 km, con velocità di propagazione di  $2,5 \times 10^8$  m/s e velocità di trasmissione di 2 Mbps? Più in generale, quanto tempo impiega un pacchetto di lunghezza  $L$  a propagarsi su un collegamento lungo  $d$ , con velocità di propagazione  $v$  e velocità di trasmissione di  $R$  bps? Questo ritardo dipende dalla lunghezza del pacchetto? Dalla velocità di trasmissione?
- R19.** Supponete che l'Host A voglia inviare un file voluminoso all'Host B. Il percorso tra l'Host A e l'Host B ha tre collegamenti con frequenze  $R_1 = 500$  kbps,  $R_2 = 2$  Mbps e  $R_3 = 1$  Mbps, rispettivamente.
- Assumete che non vi sia altro traffico nella rete: qual è il throughput per il trasferimento del file?
  - Supponete che il file sia di 4 milioni di byte. Dividendo tale grandezza per il throughput, approssimativamente quanto tempo occorrerà per trasferire il file all'Host B?
  - Ripetete (a) e (b) con  $R_2$  ridotto a 100 kbps.
- R20.** Supponete che un sistema periferico A voglia inviare un file voluminoso al sistema periferico B. Descrivete molto ad alto livello come il sistema periferico A crea i pacchetti dal file. Quando uno di questi pacchetti arriva al commutatore di pacchetti, quali informazioni contenute nel pacchetto vengono utilizzate dal commutatore per determinare il collegamento lungo il quale il pacchetto viene inoltrato? Perché la commutazione di pacchetti in Internet è analoga al guidare da una città a un'altra chiedendo le indicazioni lungo la strada?
- R21.** Provate l'animazione interattiva "Queuing and Loss" sulla piattaforma MyLab del testo. Qual è la frequenza massima di emissione e la velocità minima di trasmissione? Con tali velocità, qual è l'intensità di traffico? Eseguite l'animazione con queste velocità e determinate quanto tempo ci vuole prima che si verifichi la perdita di un pacchetto. Poi ripetete l'esperimento una seconda volta e determinate di nuovo quanto tempo ci vuole prima che si verifichi la perdita di un pacchetto. I valori sono differenti? Spiegate il perché.

### PARAGRAFO 1.5

- R22.** Elencate cinque compiti che un livello può svolgere. È possibile che uno (o più) di tali compiti possa essere eseguito da altri livelli?
- R23.** Quali sono i cinque livelli della pila di protocolli Internet? Quali sono i loro principali compiti?
- R24.** Che cos'è un messaggio a livello di applicazione? Un segmento a livello di trasporto? Un datagramma a livello di rete? Un frame a livello di collegamento?
- R25.** Quali livelli nella pila dei protocolli di Internet vengono elaborati da un router? Quali da un commutatore a livello di collegamento? Quali da un host?

## PARAGRAFO 1.6

- R26.** Che cosa è un malware auto-replicante?
- R27.** Descrivete come si possa creare una botnet e come la si possa usare per un attacco DDoS.

## Problemi

**P1.** Progettate e descrivete un protocollo a livello di applicazione da usare tra una postazione Bancomat e il calcolatore centralizzato di una banca. Il protocollo dovrebbe consentire di verificare la carta dell'utente con relativa password, di monitorare l'andamento del conto (mantenuto nel calcolatore centralizzato) e il ritiro di contante. Le entità del protocollo dovrebbero essere in grado di gestire tutti i casi comuni in cui il conto è sprovvisto di denaro sufficiente a coprire il prelievo. Il protocollo deve essere specificato elencando i messaggi scambiati e le azioni intraprese dalla postazione o dal calcolatore centrale alla trasmissione e ricezione dei messaggi. Descrivete a grandi linee le operazioni del protocollo nel caso di un semplice prelievo senza errori, utilizzando un diagramma simile a quello della Figura 1.2. Esplicitate le assunzioni del protocollo sul sottostante servizio di trasporto end-to-end.

**P2.** L'Equazione 1.1 fornisce una formula per calcolare il ritardo end-to-end quando si invia un pacchetto di grandezza  $L$  su  $N$  collegamenti con velocità  $R$ . Si generalizzi tale formula nel caso si inviano  $P$  pacchetti uno dietro l'altro sugli  $N$  collegamenti.

**P3.** Considerate un'applicazione che trasmetta dati a velocità costante (per esempio, un'unità di dati da  $N$  bit ogni  $k$  unità di tempo, dove  $k$  è piccolo e fissato). Inoltre, una volta avviata, l'applicazione continuerà a funzionare per un periodo di tempo relativamente lungo. Rispondete alle seguenti domande, motivando sinteticamente le vostre affermazioni.

- Per questa applicazione sarebbe più appropriata una rete a commutazione di pacchetto o a commutazione di circuito? Perché?
- Si supponga di utilizzare una rete a commutazione di pacchetto e che il solo traffico in questa rete provenga dall'applicazione descritta sopra. Inoltre, si assuma che la somma delle velocità di generazione dei dati sia inferiore alla capacità di ogni collegamento. È richiesta una forma di controllo della congestione? Perché?

**P4.** Considerate la rete a commutazione di circuito della Figura 1.13, ricordando che ci sono 4 circuiti su ogni collegamento. Denominate i 4 commutatori con le lettere A, B, C e D in senso orario.

- Qual è il massimo numero di connessioni contemporanee attive in un dato istante in questa rete?
- Si supponga che tutte le connessioni avvengano tra A e C. Qual è il numero massimo di connessioni simultanee in corso?
- Si supponga di voler effettuare 4 connessioni tra A e C e altre 4 tra B e D. Possiamo instradare queste

**R28.** Supponete che Alice e Bob si stiano inviando pacchetti tramite una rete di calcolatori. Supponete che Trudy si posizioni nella rete in modo da poter catturare tutti i pacchetti inviati da Alice e Bob e possa inviar loro qualsiasi pacchetto. Elencate gli attacchi che Trudy potrebbe compiere da questa posizione.

chiamate sui 4 collegamenti in modo da effettuare tutte e otto le connessioni?

**P5.** Rivedete l'analogia con l'autostrada presentata nel Paragrafo 1.4, ipotizzando ancora una volta che la velocità di propagazione delle vetture sia di 100 km/h.

- Se la coda di auto viaggia per 175 km, iniziando da un primo casello, passando per un secondo e concludendo il viaggio in prossimità del terzo casello, qual è il ritardo accumulato?
- Ripetete il punto (a), nell'ipotesi che la coda di veicoli sia composta da 8 auto anziché da 10.

**P6.** Con questo problema si comincia a esplorare il ritardo di propagazione e di trasmissione, due concetti centrali nel networking. Consideriamo due host, A e B, collegati da una singola connessione con velocità di  $R$  bps. Supponiamo che i due host siano separati da  $m$  metri e che la velocità di propagazione lungo il collegamento sia di  $v$  m/s. L'Host A sta per inviare un pacchetto di  $L$  bit all'Host B.

- Esprimete il ritardo di propagazione,  $d_{\text{prop}}$ , in funzione di  $m$  e  $v$ .
- Determinate il tempo di trasmissione del pacchetto,  $d_{\text{trasm}}$ , in termini di  $L$  e  $R$ .
- Tralasciando i ritardi di elaborazione e di accodamento, ricavate un'espressione del ritardo end-to-end.
- Supponete che l'Host A cominci a trasmettere il pacchetto all'istante  $t = 0$ . All'istante  $t = d_{\text{trasm}}$  dove si trova l'ultimo bit del pacchetto?
- Supponete che  $d_{\text{prop}}$  sia maggiore di  $d_{\text{trasm}}$ . All'istante  $t = d_{\text{trasm}}$  dove si trova il primo bit del pacchetto?
- Supponete che  $d_{\text{prop}}$  sia minore di  $d_{\text{trasm}}$ . All'istante  $t = d_{\text{trasm}}$  dove si trova il primo bit del pacchetto?
- Supponete che  $s = 2,5 \times 10^8$  m/s,  $L = 1500$  bytes e  $R = 10$  Mbps. Determinate la distanza  $m$  tale per cui  $d_{\text{prop}}$  sia uguale a  $d_{\text{trasm}}$ .

**P7.** Nel seguente problema prendete in considerazione una trasmissione di voce in tempo reale dall'Host A all'Host B su una rete a commutazione di pacchetto (VoIP). L'Host A converte al volo la voce in un flusso digitale di bit a 64 kbps. Poi raggruppa i bit in pacchetti da 56 byte. Tra A e B esiste un solo collegamento, con velocità di trasmissione 10 Mbps e ritardo di propagazione di 10 millisecondi. Non appena l'Host A compone un pacchetto lo invia all'host B. Quando quest'ultimo riceve un intero pacchetto, lo converte in un segnale analogico. Quanto tempo intercorre dall'istante in cui un bit viene creato (a partire dal segnale analogico originario nell'Host A) al momento in cui il bit viene decodificato (come parte del segnale analogico nell'Host B)?

**P8.** Supponete che alcuni utenti condividano un collegamento da 10 Mbps, e che ciascuno richieda 200 kbps quando trasmette, ma che ogni utente trasmetta solo per il 10% del tempo. (Si veda la discussione sulla commutazione di pacchetto e di circuito nel Paragrafo 1.3.).

- (a) Quando si utilizza la commutazione di circuito, quanti utenti si possono supportare?
- (b) Per il resto del problema, si supponga l'adozione della commutazione di pacchetto. Determinate la probabilità che un certo utente stia trasmettendo.
- (c) Si ipotizzi, ora, che ci siano 120 utenti. Determinate la probabilità che, a ogni dato istante, esattamente  $n$  utenti stiano trasmettendo contemporaneamente (*suggerimento*: usate la distribuzione binomiale).
- (d) Calcolate la probabilità di avere 51 o più utenti contemporaneamente in fase di trasmissione.

**P9.** Considerate la discussione sulla commutazione di pacchetto e di circuito illustrata nel Paragrafo 1.3, nella quale viene presentato un esempio con un collegamento da 1 Mbps. Gli utenti quando sono attivi generano dati alla velocità di 100 kbps, ma sono attivi con probabilità  $p = 0,1$ . Ipotizziamo di sostituire il collegamento da 1 Mbps con uno da 1 Gbps.

- (a) Quanto vale  $N$ , numero massimo di utenti che possono essere supportati contemporaneamente dalla commutazione di circuito?
- (b) Si consideri ora la commutazione di pacchetto e una popolazione di utenti composta da  $M$  elementi. Esprimete una formula (in funzione di  $p$ ,  $M$  e  $N$ ) per la probabilità che più di  $N$  utenti stiano inviando dati.

**P10.** Considerate un pacchetto di lunghezza  $L$  che viene creato da A e che viaggia su tre collegamenti connessi da due commutatori di pacchetto. Siano  $d_i$ ,  $s_i$ , e  $R_i$  la lunghezza, la velocità di propagazione e la velocità di trasmissione del collegamento  $i$ , per  $i = 1,2,3$ . Il commutatore di pacchetti ritarda ciascun pacchetto di  $d_{\text{proc}}$ . Assumendo che non vi siano ritardi di accodamento, in termini di  $d_i$ ,  $s_i$  e  $R_i$  (per  $i = 1,2,3$ ) e  $L$ , qual è il ritardo totale end-to-end per il pacchetto? Supponete ora che il pacchetto sia di 1500 byte, la velocità di propagazione su entrambi i collegamenti di  $2,5 \times 10^8$  m/s, la velocità di trasmissione di entrambi i collegamenti di 2,5 Mbps, il ritardo di elaborazione del commutatore di pacchetti di 3 ms, la lunghezza del primo collegamento di 5000 km, del secondo di 4000 km e la lunghezza dell'ultimo collegamento di 1000 km. Per questi valori qual è il ritardo end-to-end?

**P11.** Nel problema precedente, supponete che  $R_1 = R_2 = R_3 = R$  e  $d_{\text{proc}} = 0$ . Inoltre, supponete che il commutatore di pacchetti non utilizzi il metodo store-and-forward, ma che invece trasmetta immediatamente ciascun bit che riceve senza aspettare che il pacchetto arrivi. Qual è il ritardo end-to-end?

**P12.** Un commutatore di pacchetti riceve un pacchetto e determina il collegamento di uscita sul quale dovrà essere inoltrato. Quando il pacchetto arriva, un altro pacchetto è stato già per metà trasmesso su questo collegamento di uscita e altri quattro pacchetti stanno aspettando di

essere trasmessi. I pacchetti sono trasmessi nell'ordine di arrivo. Supponete che tutti i pacchetti siano di 1500 byte e che la velocità del collegamento sia di 2 Mbps. Qual è il ritardo di accodamento per il pacchetto? Più in generale, qual è il ritardo di accodamento quando tutti i pacchetti sono di lunghezza  $L$ , la velocità di trasmissione è  $R$ ,  $x$  bit di un pacchetto sono stati trasmessi e  $n$  pacchetti sono già nella coda?

**P13.** (a) Supponete che  $N$  pacchetti arrivino contemporaneamente a un collegamento nel quale non vi sono pacchetti in corso di trasmissione o in coda. Ciascun pacchetto è di lunghezza  $L$  e la velocità di trasmissione è  $R$ . Qual è il ritardo medio di accodamento per gli  $N$  pacchetti?

- (b) Supponete che gli  $N$  pacchetti arrivino al collegamento ogni  $LN/R$  secondi. Qual è il ritardo medio di accodamento per gli  $N$  pacchetti?

**P14.** Considerate il ritardo di accodamento nel buffer di un router. Sia  $I$  l'intensità di traffico, ossia  $I = La/R$ . Supponete che il ritardo di accodamento sia della forma  $IL/R(1 - I)$  quando  $I < 1$ .

- (a) Trovate una formula per il ritardo totale, ossia per il ritardo di accodamento più il ritardo di trasmissione.
- (b) Esprimete il ritardo totale in funzione di  $L/R$ .

**P15.** Sia  $a$  la velocità con cui i pacchetti arrivano su un collegamento espresso in pacchetti/secondo e sia  $\mu$  la velocità di trasmissione del collegamento in pacchetti/secondo. Sulla base della formula del ritardo totale (ritardo di accodamento sommato al ritardo di trasmissione) derivata nel problema precedente, si derivi una formula per il ritardo totale espressa in funzione di  $a$  e  $\mu$ .

**P16.** Si consideri il buffer che precede un collegamento in uscita di un router. In questo problema si farà uso della formula di Little, una formula famosa della teoria delle code. Sia  $N$  il numero medio di pacchetti del buffer più il pacchetto in via di trasmissione. Sia  $a$  la velocità con cui i pacchetti arrivano sul collegamento. Sia  $d$  il ritardo totale medio (ritardo di coda più ritardo di trasmissione) di un pacchetto. La formula di Little è  $N = a \cdot d$ . Si supponga che in media il buffer contenga 100 pacchetti e che il ritardo di accodamento medio sia di 20 ms. La velocità di trasmissione del collegamento è pari a 100 pacchetti al secondo. Utilizzando la formula di Little si calcoli la velocità media di arrivo dei pacchetti, assumendo che non ci siano perdite di pacchetti.

**P17.** (a) Generalizzate l'Equazione 1.2 per il ritardo totale end-to-end presentata nel Paragrafo 1.4.3 al caso di velocità di elaborazione, velocità di trasmissione e ritardi di propagazione eterogenei.

- (b) Ripetete il punto (a), supponendo ora la presenza di un ritardo medio di accodamento  $d_{\text{acc}}$  a ogni nodo.

**P18.** Impiegate Traceroute tra sorgente e destinazione poste sullo stesso continente a tre diverse ore del giorno.

- (a) Calcolate la media e la deviazione standard dei ritardi di andata e ritorno nei tre orari.
- (b) Trovate il numero di router nel percorso a ciascuno dei tre orari. Al variare dell'ora cambia il percorso?

- (c) Cercate di identificare il numero di reti ISP che i pacchetti di Traceroute attraversano nel loro viaggio dall'origine alla destinazione. I router con nomi e/o indirizzi IP simili dovrebbero essere considerati parte dello stesso ISP. Nei vostri esperimenti, il ritardo più esteso si verifica presso le interfacce di peering tra ISP adiacenti?
- (d) Ripetete l'esercizio nel caso di sorgente e destinazione poste su continenti diversi. Confrontare i risultati precedenti con quelli del nuovo esperimento.

**P19.** La legge di Metcalfe afferma che il valore di una rete di computer è proporzionale al quadrato del numero di utenti collegati al sistema. Sia  $n$  il numero di utenti in una rete di computer. Supponendo che ogni utente invii un messaggio a ciascuno degli altri utenti, quanti messaggi verranno inviati? La risposta supporta la legge di Metcalfe?

**P20.** Considerate l'esempio del throughput corrispondente alla Figura 1.20 (b). Ora supponete che vi siano  $M$  coppie client-server piuttosto che 10. Siano  $R_s$ ,  $R_c$  e  $R$  le velocità dei collegamenti verso il client, il server e il collegamento di rete. Assumete che tutti gli altri collegamenti abbiano capacità in eccesso e che non vi sia altro traffico in rete a parte quello generato tra le  $M$  coppie client-server. Ricavate un'espressione generale per il throughput in termini di  $R_s$ ,  $R_c$ ,  $R$  e  $M$ .

**P21.** Si consideri la Figura 1.19(b). Si supponga che ci siano  $M$  percorsi tra il server e il client. Non ci sono collegamenti condivisi tra i percorsi. Il percorso  $k$  ( $k = 1, \dots, M$ ) consiste di  $N$  collegamenti con velocità di trasmissione  $R_{k,1}^K, R_{k,2}^K, \dots, R_{k,N}^K$ . Se il server può usare solo un percorso per inviare i dati al client, qual è il throughput massimo raggiungibile dal server? Se il server potesse usare gli  $M$  percorsi per trasmettere i dati, quale sarebbe il suo massimo throughput?

**P22.** Si consideri la Figura 1.19(b). Si supponga che tutti i collegamenti tra il server e il client abbiano una probabilità di perdita dei pacchetti pari a  $p$  e che tali probabilità siano indipendenti. Si calcoli la probabilità che un pacchetto inviato dal server venga ricevuto dal destinatario. Se il pacchetto viene perso durante il percorso dal server al client, il server ritrasmette il pacchetto. Si calcoli quante ritrasmissioni del pacchetto in media il server deve effettuare perché il pacchetto venga ricevuto con successo dal client.

**P23.** Si consideri la Figura 1.19(a). Si assuma che il collegamento collo di bottiglia sul percorso dal server al client sia il primo e abbia velocità  $R_s$  bps. Si supponga di inviare una coppia di pacchetti uno dietro l'altro dal server al client e che non ci sia altro traffico sul percorso. Si assume inoltre che ogni pacchetto sia di  $L$  bit e che entrambi i collegamenti abbiano un ritardo di propagazione  $d_{prop}$ .

- Qual è il tempo di inter-arrivo dei pacchetti alla destinazione, ossia a quanto ammonta il tempo che intercorre da quando arriva l'ultimo bit del primo pacchetto a quando arriva l'ultimo bit del secondo pacchetto?

- Si assume ora che il collo di bottiglia sia il secondo collegamento ( $R_c < R_s$ ). Si dica se è possibile che il

secondo pacchetto attenda nella coda di input del secondo collegamento e se ne esprima la motivazione. Si supponga ora che il server trasmetta il secondo pacchetto  $T$  secondi dopo aver inviato il primo. Si calcoli quanto grande debba essere  $T$  affinché non ci sia coda prima del secondo collegamento.

**P24.** Supponete di voler consegnare velocemente 50 terabyte da Boston a Los Angeles. È disponibile un collegamento dedicato al trasferimento dati di 100 Mbps. Dite, motivandolo, se scegliereste di trasmettere i dati su questo collegamento o usando la consegna notturna di un corriere espresso.

**P25.** Supponete che due host, A e B, siano separati da 20.000 chilometri e connessi da un collegamento diretto da  $R = 5$  Mbps. Ipotizzate che la velocità di propagazione lungo il collegamento sia di  $2,5 \times 10^8$  m/s.

- Calcolate il prodotto larghezza di banda-ritardo,  $R \times d_{prop}$ .
- Consideriamo l'invio di un file di 800.000 bit dall'Host A all'Host B. Supponendo che il file venga inviato in modo continuato come un unico grande messaggio, qual è il massimo numero di bit che si troveranno nel collegamento a ogni dato istante?
- Fornite un'interpretazione del prodotto larghezza di banda-ritardo.
- Quanto vale l'ampiezza (in metri) di un bit nel collegamento? Risulta più lunga rispetto a un campo da calcio?
- Ricavate un'espressione generale per l'ampiezza di un bit in funzione della velocità di propagazione  $v$ , del tasso trasmissivo  $R$  e della lunghezza  $m$  del collegamento.

**P26.** In riferimento al Problema 24, supponete di poter modificare  $R$ . Per quale valore di  $R$  l'ampiezza di un bit risulta pari alla lunghezza del collegamento?

**P27.** Considerate il Problema 24, ma con un collegamento da  $R = 500$  Mbps.

- Calcolate il prodotto larghezza di banda-ritardo,  $R \times d_{prop}$ .
- Considerate l'invio di un file di 800.000 bit dall'Host A all'Host B. Supponendo che il file venga inviato in modo continuato come un unico grande messaggio, qual è il massimo numero di bit che si troveranno nel collegamento a ogni dato istante?
- Quanto vale l'ampiezza (in metri) di un bit nel collegamento?

**P28.** Fate ancora una volta riferimento al Problema 24.

- Quanto richiede l'invio del file, assumendo un invio continuativo?
- Supponete ora che il file venga spezzato in 20 pacchetti, ciascuno da 40.000 bit. Ipotizzate che la ricezione di ogni pacchetto venga confermata dal destinatario e che il tempo di trasmissione di un pacchetto di notifica sia trascurabile. Infine, assumete che il mittente non possa inviare un pacchetto fino a quando il pacchetto precedente non sia stato confermato. Quanto richiede l'invio del file?
- Confrontate i risultati di (a) e (b).

**P29.** Considerate un collegamento a microonde da 10 Mbps tra un satellite geostazionario e la sua stazione base sulla Terra. A ogni minuto il satellite scatta una foto digitale e la invia alla stazione base. Ipotizzate una velocità di propagazione pari a  $2,4 \times 10^8$  m/s.

- Qual è il ritardo di propagazione del collegamento?
- Quanto vale il prodotto larghezza di banda-ritardo,  $R \times d_{\text{prop}}$ ?
- Sia  $x$  la dimensione della foto. Qual è il valore minimo di  $x$  affinché il collegamento a microonde trasmetta con continuità?

**P30.** Si consideri l'analogia del viaggio su linea aerea presentata nel Paragrafo 1.5 a proposito della stratificazione, e l'aggiunta di intestazioni alle unità di dati del protocollo quando esse scendono lungo la pila di protocolli. Esiste una nozione equivalente alle informazioni di intestazione che viene aggiunta ai passeggeri e al bagaglio quando ci si sposta verso il basso lungo la pila di protocolli della linea aerea?

**P31.** Nelle odierne reti a commutazione di pacchetto, l'host sorgente suddivide i messaggi lunghi a livello di applicazione (per esempio, un'immagine o un file di musica) in pacchetti più piccoli e invia questi ultimi nella rete. A questo punto il destinatario riassembra i pacchetti per ricostruire il messaggio originario. Chiamiamo questo processo **segmentazione del messaggio**. La Figura 1.27 illustra il trasporto end-to-end di un messaggio con e senza segmentazione. Si consideri un messaggio lungo  $10^6$  bit e si supponga che ogni collegamento sia da 5 Mbps. Tralasciate i ritardi di propagazione, di accodamento e di elaborazione.

- Considerate l'invio del messaggio dalla sorgente alla destinazione senza segmentazione. Quanto tempo è richiesto per spostare il messaggio dall'host sorgente al primo commutatore di pacchetto? Tenendo a mente che ogni commutatore utilizza commutazione di pacchetto store-and-forward, qual è il tempo totale richiesto per trasferire il messaggio tra l'host sorgente e quello di destinazione?
- Supponete ora che il messaggio venga segmentato in 100 pacchetti, da 10.000 bit. Quanto tempo è ri-

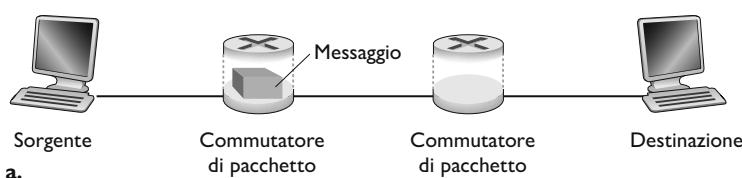
chiesto per trasferire il primo pacchetto dall'host sorgente al primo commutatore di pacchetto? Quando il primo pacchetto sta per essere inviato dal primo al secondo commutatore, il secondo pacchetto sta per essere inviato dall'host sorgente al primo commutatore. In quale istante il secondo pacchetto sarà completamente ricevuto dal primo commutatore?

- Quanto tempo richiede la trasmissione del file se si usa la segmentazione dei messaggi? Confrontate questo risultato con la risposta del punto (a) e commentatelo.
- Oltre a ridurre il ritardo, a che cos'altro serve la segmentazione dei messaggi?
- Discutete gli svantaggi legati alla segmentazione dei messaggi.

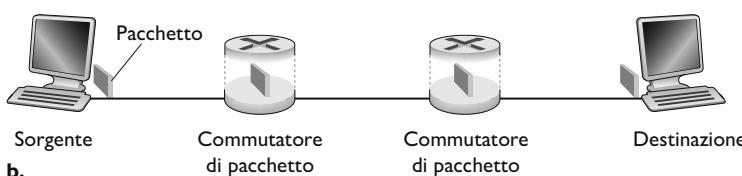
**P32.** Effettuate alcuni esperimenti con l'animazione interattiva di segmentazione dei messaggi presente sulla piattaforma MyLab del libro. I ritardi dell'animazione corrispondono a quelli della precedente domanda? Che influenza hanno i ritardi di propagazione sul ritardo complessivo end-to-end nel caso di commutazione di pacchetto (con segmentazione del messaggio) e nel caso di commutazione di messaggio?

**P33.** Considerate l'invio di un file da  $F$  bit tra gli host A e B. Tra A e B si trovano tre collegamenti (e due commutatori), e i collegamenti non sono congestionati (quindi non si verificano ritardi di accodamento). A segmenta il file in porzioni di  $S$  bit ciascuna e aggiunge 80 bit di intestazione a ciascun pacchetto, formando pacchetti da  $L = 80 + S$  bit. Ogni collegamento ha una velocità di trasmissione di  $R$  bps. Trovate il valore di  $S$  che minimizza il ritardo di trasferimento da A a B. Trascurate il ritardo di propagazione.

**P34.** Skype offre un servizio che permette le chiamate da PC a telefono normale. Questo implica che la chiamata vocale viaggia sia su Internet sia sulla rete telefonica. Discutete come possa essere implementato questo servizio.



**Figura 1.27** Trasporto di messaggi dalla sorgente alla destinazione: (a) senza segmentazione; (b) con segmentazione.



## Esercitazioni Wireshark

“Dimmi e io dimentico. Mostrami e io ricordo. Coinvolgimi e io comprendo.”

*Proverbio cinese*

La comprensione personale dei protocolli di rete può essere approfondita vedendoli in azione e provandoli, osservando la sequenza di messaggi scambiati tra le due entità di protocollo, approfondendo i dettagli delle operazioni di protocollo e forzando i protocolli a effettuare determinate azioni per verificarne le conseguenze. Tutto questo può essere fatto o in scenari simulati o in un ambiente di rete reale, come Internet. Le animazioni interattive di questo libro scelgono il primo approccio.

Nelle esercitazioni Wireshark intraprenderemo invece il secondo approccio. Lancerete applicazioni di rete in diversi scenari e osserverete i protocolli di rete nel vostro calcolatore, interagendo e scambiando messaggi con l’entità di protocollo in esecuzione da qualche parte in Internet. Pertanto, voi e il vostro computer sarete parte integrante di queste esercitazioni dal vivo. Osserverete e imparerete, facendo.

Lo strumento di base per osservare i messaggi scambiati tra entità di protocollo in esecuzione viene detto **packet sniffer**. Come suggerisce il nome, esso copia passivamente (ossia “sniffa, annusa”) i messaggi che vengono inviati e ricevuti dal vostro computer; inoltre mostra i contenuti dei vari campi di protocollo dei messaggi catturati. Nella Figura 1.28 viene mostrata una videata del packet sniffer Wireshark. Il software è gratuito e viene eseguito da macchine Windows, Linux/Unix e Mac. Nel corso del testo troverete esercitazioni Wireshark che vi consentiranno di esplorare numerosi protocolli studiati nel testo. In questa prima esercitazione richiederete e installerete una copia di Wireshark, accederete a un sito web e catturerete ed esaminerete i messaggi di protocollo scambiati tra il vostro browser e il web server.

Potete trovare tutti i dettagli su questa prima esercitazione Wireshark (comprese le istruzioni su come ottenere e installare il software) sulla piattaforma MyLab del volume.

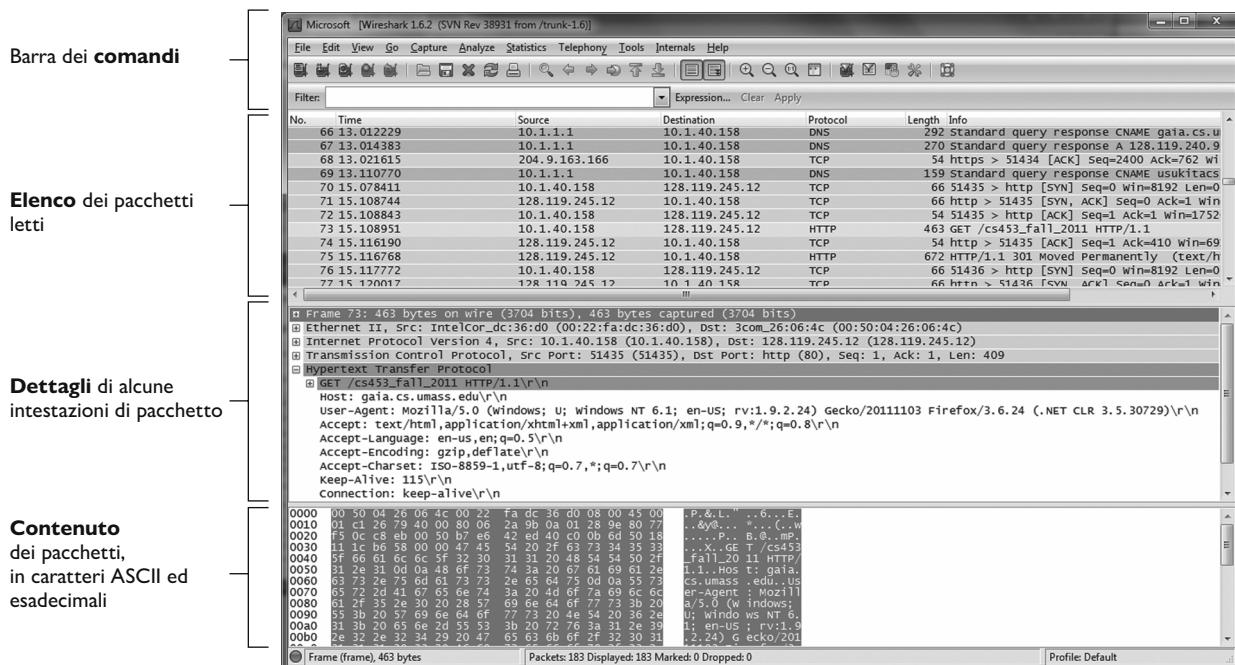


Figura 1.28 Una schermata di Wireshark (per gentile concessione della Wireshark Foundation).

# Livello di applicazione

Le applicazioni di rete sono la *raison d'être* delle reti di calcolatori. Se non riuscissimo a concepire applicazioni utili non ci sarebbe alcun bisogno di progettare protocolli di rete per supportarle. Sin dagli esordi di Internet sono state sviluppate numerose applicazioni utili e dilettevoli. Tali applicazioni sono state la forza propulsiva del successo di Internet, spingendo le persone a fare in modo che la rete fosse parte integrante della loro vita quotidiana a casa, sul lavoro, nelle scuole e negli uffici pubblici.

Sono applicazioni Internet quelle che divennero popolari negli anni '70 e '80, tra cui e-mail, trasferimento file e newsgroup, ma anche la *killer application*<sup>1</sup> di metà degli anni '90: il World Wide Web, che comprende il web surfing (navigazione sul Web), la ricerca di informazioni e l'e-commerce. Sin dall'inizio del nuovo millennio continuano a emergere nuove, avvincenti applicazioni tra cui la telefonia su IP (voice-over-IP, VoIP), la videoconferenza su IP come Skype, Facetime and Google Hangouts; la distribuzione di video generati dagli utenti come YouTube e il cinema on demand come Netflix. E ancora giochi on-line multiutente, come Second Life e World of Warcraft.

Più recentemente sono comparse le applicazioni di social networking di nuova generazione come Facebook, Instagram e Twitter che hanno creato una rete di persone sopra la rete Internet composta da router e collegamenti fisici.

Più recentemente, con la diffusione degli smartphone e dell'accesso Internet wireless 4G/5G, sono apparse molte nuove applicazioni basate sulla posizione quali Yelp, Tinder e Waz, applicazioni di pagamento mobile quali WeChat e Apple Pay e applicazioni di messaggistica quali WeChat e WhatsApp. Il processo di creazione di nuove, eccitanti applicazioni per Internet non si arresta: magari sarà qualcuno di voi a creare la prossima *killer application* di Internet!

---

<sup>1</sup> *Killer application* significa letteralmente “applicazione assassina”, ma viene intesa in gergo nel senso metaforico di un’applicazione decisiva, vincente, rivoluzionaria o di estremo successo, grazie alla quale la tecnologia su cui essa si basa (Internet nel nostro caso) riesce a penetrare il mercato (N.d.R.).

In questo capitolo affronteremo gli aspetti concettuali e implementativi delle applicazioni di rete. Inizieremo definendo i concetti chiave del livello di applicazione, tra cui i servizi di rete richiesti dalle applicazioni, l'architettura client e server, i processi e le interfacce a livello di trasporto. Di seguito esamineremo in dettaglio alcune applicazioni, tra cui Web, posta elettronica, DNS, condivisione di file P2P. Successivamente tratteremo lo sviluppo delle applicazioni di rete, sia su TCP sia su UDP. In particolare studieremo la API delle socket e daremo uno sguardo ad alcune semplici applicazioni client-server in Python. Alla fine del capitolo presenteremo numerosi, divertenti e interessanti compiti di programmazione con le socket.

Il livello di applicazione è un ambiente particolarmente favorevole per iniziare lo studio dei protocolli perché ci è familiare. Ci fornirà una buona base per comprendere di che cosa si occupano i protocolli e introdurrà molti argomenti che rivedremo durante lo studio dei protocolli a livello di trasporto, di rete e di collegamento.

## 2.1 Princìpi delle applicazioni di rete

Immaginate di avere un'idea per una nuova applicazione di rete che forse renderà un importante servizio all'umanità o piacerà al vostro professore o vi renderà ricchi o semplicemente sarà divertente da sviluppare. Qualunque sia la motivazione che vi spinge, vedremo ora come trasformare un'idea in una reale applicazione di rete.

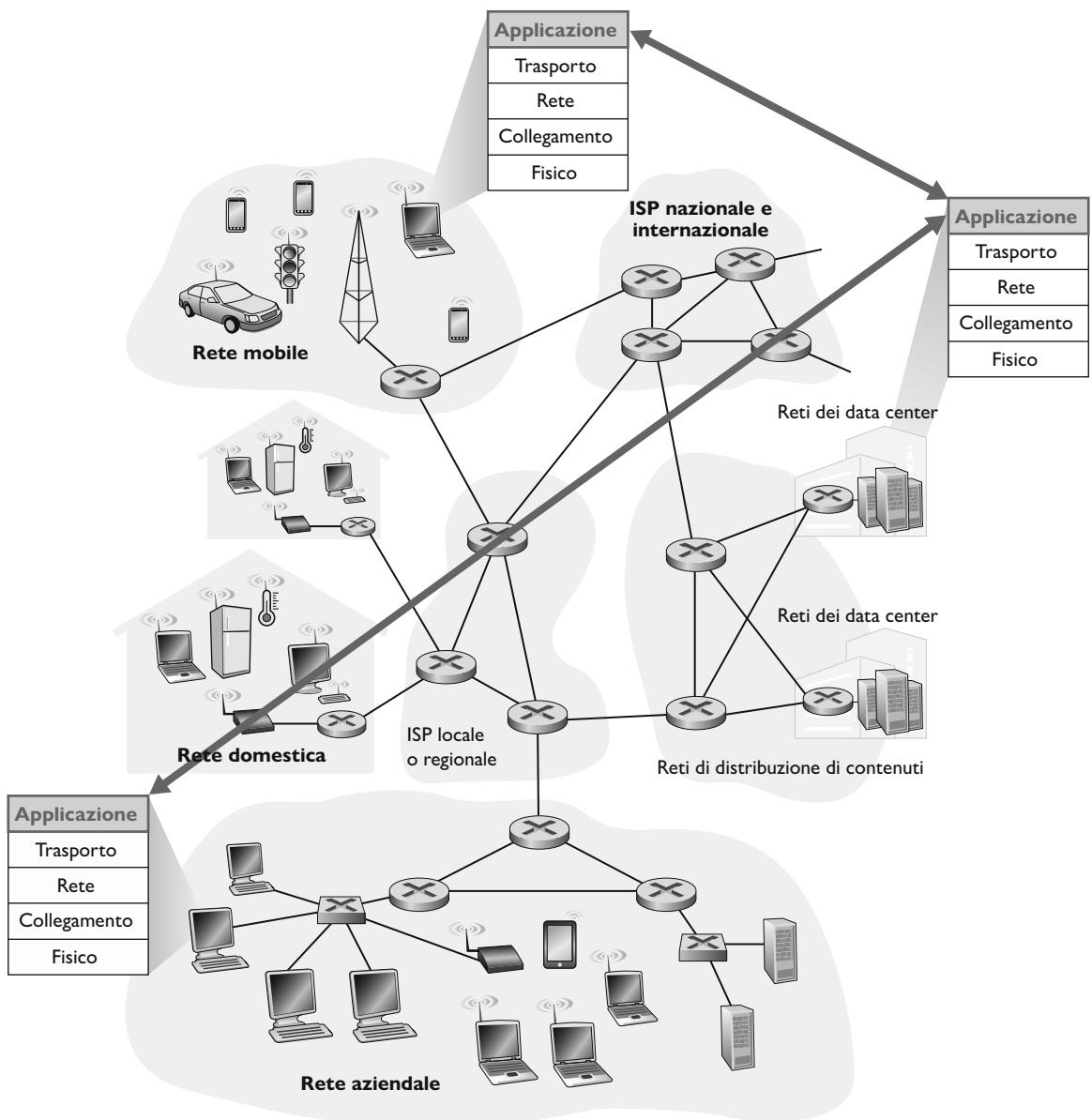
Il cuore dello sviluppo delle applicazioni di rete è costituito dalla creazione dei programmi che sono eseguiti dai sistemi periferici e che comunicano tra loro via rete. Per esempio, nelle applicazioni web esistono due programmi diversi che comunicano tra di loro: il browser, che viene eseguito dall'host dell'utente (desktop, laptop, tablet, smartphone e così via) e il web server che si trova nell'host che viene di solito chiamato anch'esso web server.

Un altro esempio è rappresentato dalle applicazioni di Video on Demand come Netflix (Paragrafo 2.6) in cui c'è un programma fornito da Netflix che viene eseguito sullo smartphone, tablet o computer dell'utente e un programma server in esecuzione su un host server di Netflix. Spesso, ma certamente non sempre, i server sono ospitati in data center, come mostrato nella Figura 2.1.

Nello sviluppo di una nuova applicazione dovete scrivere software in grado di funzionare su più macchine. Questo software può essere scritto, per esempio, in C, Java o Python ma, aspetto non trascurabile, non occorre predisporre programmi per i dispositivi del nucleo della rete, quali router o commutatori a livello di collegamento. E anche se lo voleste, non sareste in grado di farlo. Come appreso nel Capitolo 1 e come mostrato nella Figura 1.24, i dispositivi del nucleo della rete non lavorano a livello applicativo, ma ai livelli inferiori. La progettazione di base, cioè il confinamento delle applicazioni software nei sistemi periferici, ha facilitato il rapido sviluppo di una vasta gamma di applicazioni per Internet, come illustrato nella Figura 2.1.

### 2.1.1 Architetture delle applicazioni di rete

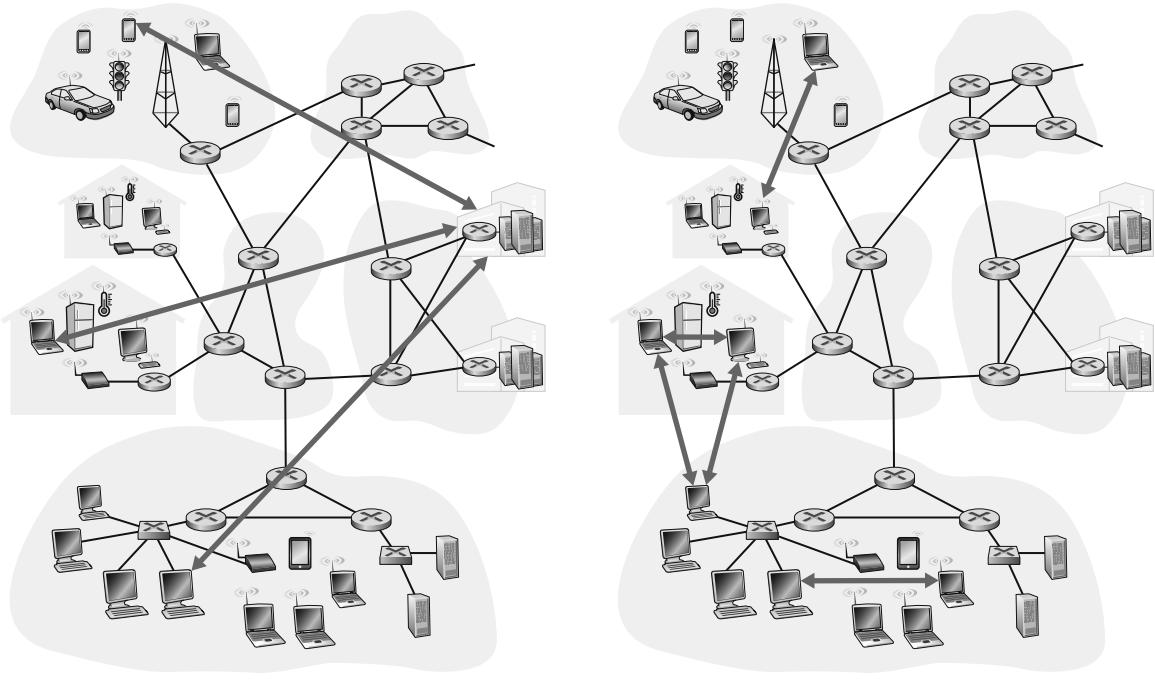
Prima di occuparci della codifica software è necessario disporre di un progetto dettagliato dell'architettura dell'applicazione, che non ha niente a che vedere con l'architettura di rete quale, per esempio, l'architettura di Internet a cinque livelli trattata nel Capitolo 1. Per lo sviluppatore di applicazioni l'architettura



**Figura 2.1** La comunicazione in un'applicazione di rete ha luogo tra sistemi periferici a livello di applicazione.

di rete è fissata e fornisce alle applicazioni uno specifico insieme di servizi; il suo compito è progettare l'**architettura dell'applicazione** (*application architecture*) e stabilire la sua organizzazione sui vari sistemi periferici, basandosi, probabilmente, su una delle due principali architetture di rete attualmente utilizzate: l'architettura client-server o peer-to-peer (P2P).

Nell'**architettura client-server** (*client-server architecture*) vi è un host sempre attivo, chiamato *server*, che risponde alle richieste di servizio di molti altri host, detti *client*. Un esempio classico è rappresentato dall'applicazione web, in cui



**Figura 2.2** (a) Architettura client-server; (b) architettura P2P.

un web server, sempre attivo, risponde alle richieste dei browser in funzione sui client. Il server, quando riceve una richiesta di un oggetto da parte di un client, risponde inviandolo. Si osservi che nell’architettura client-server i client non comunicano direttamente tra loro; così, in una applicazione web, i browser non interagiscono direttamente tra loro. Inoltre il server dispone di un indirizzo fisso e noto, detto indirizzo IP (che tratteremo presto). Il client può quindi contattare il server in qualsiasi momento, inviandogli un pacchetto. Tra le più note applicazioni con architettura client-server, ricordiamo il Web, il trasferimento dei file con FTP, Telnet e la posta elettronica. L’architettura client-server è mostrata nella Figura 2.2(a).

Spesso, in un’applicazione client-server, un singolo host che esegue un server non è in grado di rispondere a tutte le richieste dei suoi client. Per esempio, una social network con molti iscritti potrebbe rapidamente soccombere qualora si basasse su un solo server per gestire tutte le richieste. Per questo motivo nelle architetture client-server si usano spesso **data center** che, ospitando molti host, creano un potente server virtuale. I servizi più popolari come i motori di ricerca (per esempio, Google, Bing e Baidu), il commercio elettronico (per esempio, Amazon, eBay e Alibaba), la posta elettronica basata sul Web (per esempio, Gmail e Yahoo Mail), i social media (per esempio, Facebook, Instagram, Twitter e WeChat) utilizzano uno o più data center. Come discusso nel Paragrafo 1.3.3, Google dispone di 19 data center sparsi in tutto il mondo che collettivamente gestiscono ricerche, YouTube, Gmail e altri servizi. Un data

center può ospitare fino a centinaia di migliaia di server a cui deve essere fornita alimentazione e manutenzione. Inoltre, i fornitori di servizi devono pagare i costi di banda e di interconnessione per trasmettere i dati dai loro data center.

In un'**architettura P2P** l'infrastruttura di server in data center è minima o del tutto assente; si sfrutta, invece, la comunicazione diretta tra coppie di host, chiamate *peer* (ossia pari), collegate in modo intermittente. I peer non appartengono a un fornitore di servizi, ma sono computer fissi e portatili, controllati dagli utenti, che per lo più si trovano nelle abitazioni, nelle università e negli uffici. Dato che i peer comunicano senza passare attraverso un server specializzato, l'architettura viene detta *peer-to-peer*. Molte tra le applicazioni attualmente più diffuse e con elevata intensità di traffico sono basate su un'architettura P2P. Queste applicazioni includono la condivisione di file come l'applicazione BitTorrent, per esempio.

Uno dei punti di forza dell'architettura P2P è la sua intrinseca **scalabilità**. In un'applicazione di condivisione dei file P2P, ogni peer, sebbene generi carico di lavoro richiedendo dei file, aggiunge anche capacità di servizio al sistema, rispondendo alle richieste di altri *peer*. Le architetture P2P sono anche economicamente convenienti, perché normalmente non richiedono per i server né una significativa infrastruttura né una disponibilità di banda elevata (al contrario dell'architettura client-server con data center). Tuttavia, in futuro le applicazioni P2P dovranno affrontare grandi sfide riguardanti sicurezza, prestazioni e affidabilità.

### 2.1.2 Processi comunicanti

Prima di costruire un'applicazione di rete dovete anche conoscere come comunicano tra loro i programmi in esecuzione su diversi host. Nel gergo dei sistemi operativi non si parla in effetti di programmi, ma di **processi comunicanti**. Si può pensare a un **processo** come a un programma in esecuzione su un sistema. Processi in esecuzione sullo stesso sistema comunicano utilizzando un approccio interprocesso (*interprocess communication*). Le regole di questo tipo di comunicazione sono governate dal sistema operativo del calcolatore in questione. Ma in questo libro non siamo interessati a come comunicano i processi all'interno dello stesso host, bensì a come comunicano i processi in esecuzione su sistemi *diversi*, che potrebbero anche avere sistemi operativi diversi.

I processi su due host diversi comunicano scambiandosi **messaggi** attraverso la rete: il processo mittente crea messaggi e li invia attraverso la rete e il processo destinatario li riceve e, quando previsto, invia messaggi di risposta. La Figura 2.1 mostra come i processi comunicano utilizzando il livello di applicazione della pila a cinque livelli dei protocolli Internet.

#### Processi client e server

Le applicazioni di rete sono costituite da una coppia di processi che si scambiano messaggi su una rete. Per esempio, nelle applicazioni web, un browser (processo client) scambia messaggi con un web server (processo server). In un sistema di condivisione P2P, i file sono trasferiti dal processo di un peer al processo in un altro peer. Per ciascuna coppia di processi comunicanti, generalmente ne etichettiamo uno come **client** e l'altro come **server**. Nel Web il

browser rappresenta un processo client, mentre il web server è un processo server. Nella condivisione di file tramite P2P il peer che scarica il file viene detto client, quello che lo invia è chiamato server.

In alcune applicazioni, come quelle di condivisione dei file via P2P, un processo può essere sia client sia server, in quanto può tanto inviare quanto ricevere file. Ciò nondimeno, nel contesto di una data sessione tra una coppia di processi, possiamo ancora etichettare l'uno come client e l'altro come server. Definiamo come segue i processi client e server.

*Nel contesto di una sessione di comunicazione tra una coppia di processi, quello che avvia la comunicazione (cioè, contatta l'altro processo all'inizio della sessione) è indicato come **client** mentre quello che attende di essere contattato per iniziare la sessione è detto **server**.<sup>2</sup>*

Nel Web un processo browser avvia il contatto con un processo web server; quindi il primo è il client e il secondo il server. Nella condivisione di file P2P, quando il peer A chiede al peer B di inviare un dato file, il primo rappresenta il client e il secondo il server nel contesto di questa specifica sessione di comunicazione. Occasionalmente, useremo anche la terminologia “**lato client** e **lato server** di un'applicazione”. Alla fine di questo capitolo mostreremo alcune semplici righe di codice per il lato client e il lato server delle applicazioni di rete.

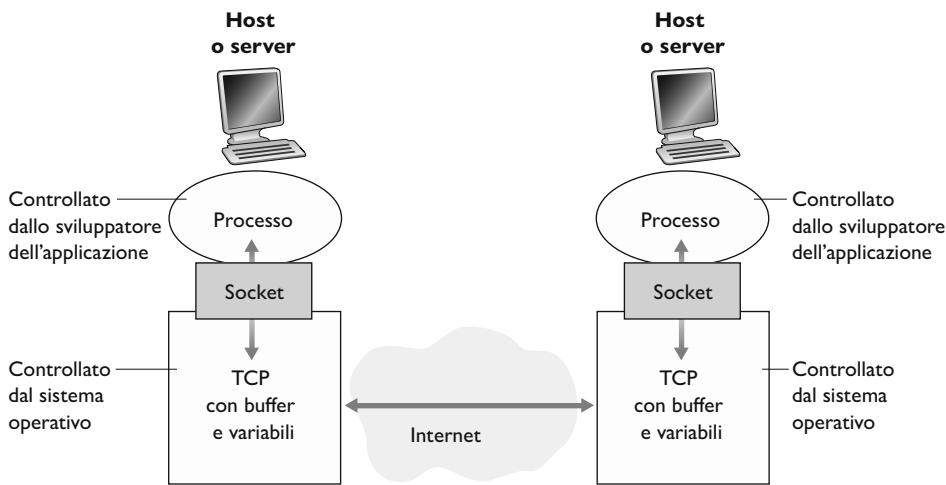
### L'interfaccia tra il processo e la rete

La maggior parte delle applicazioni consiste di coppie di processi comunicanti che si scambiano messaggi. Ogni messaggio inviato da un processo a un altro deve passare attraverso la rete sottostante. Un processo invia messaggi nella rete e riceve messaggi dalla rete attraverso un'interfaccia software detta **socket**. Usiamo un'analogia che ci aiuterà a comprendere i processi e le socket. Un processo è assimilabile a una casa e le socket sono il corrispettivo delle porte. Un processo che vuole inviare un messaggio a un altro processo o a un altro host, fa uscire il messaggio dalla propria porta (*socket*). Il processo presuppone l'esistenza di un'infrastruttura esterna che trasporterà il messaggio attraverso la rete fino alla porta del processo di destinazione. Quando il messaggio giunge al destinatario, attraversa la porta (*socket*) del processo ricevente che infine opera sul messaggio.

La Figura 2.3 mostra la comunicazione tra le socket di due processi che comunicano via Internet. Tale figura ipotizza che il protocollo di trasporto sottostante sia TCP. Come mostrato nella figura, una socket è l'interfaccia tra il livello di applicazione e il livello di trasporto all'interno di un host. Si parla anche di **API** (*Application Programming Interface*) tra l'applicazione e la rete, dato

---

<sup>2</sup> Questa definizione si riferisce a una famiglia specifica di servizi: quelli fruiti in modalità pull; infatti, il client tira verso di sé i dati. Il lettore cerchi di non confondersi con i servizi fruiti in modalità push, dove il server spinge i dati verso il client, come nel caso dei servizi di messaggistica istantanea in cui il client riceve dati in modalità asincrona. Possiamo rileggere la definizione dicendo che nel contesto di una sessione di comunicazione tra una coppia di processi, quello che richiede il servizio o le informazioni è indicato come client, mentre quello che eroga il servizio o procura le informazioni è detto server (N.d.R.).



**Figura 2.3** Processi, socket e protocollo di trasporto sottostante.

che la socket rappresenta l’interfaccia di programmazione con cui le applicazioni di rete vengono costruite. Il progettista dell’applicazione esercita un controllo totale sul livello applicativo della socket, ma ne ha ben poco su quello di trasporto. Il progettista può (1) scegliere il protocollo di trasporto e (2) a volte determinare alcuni parametri a livello di trasporto, quali la dimensione massima del buffer e del segmento (che verranno esaminate nel Capitolo 3). Dopo aver scelto il protocollo di trasporto (sempre che la scelta sia possibile), si costruisce l’applicazione usando i servizi del livello di trasporto forniti dal protocollo. Approfondiremo le socket nel corso del Paragrafo 2.7.

### Indirizzamento

Come nella posta tradizionale, affinché la consegna possa essere effettuata, il destinatario deve avere un indirizzo. Anche in Internet i processi riceventi devono averne uno per ricevere i messaggi inviati da un processo in esecuzione su un altro host. Per identificare il processo ricevente, è necessario specificare due informazioni: (1) l’indirizzo dell’host e (2) un identificatore del processo ricevente sull’host di destinazione.

In Internet, gli host vengono identificati attraverso i loro **indirizzi IP**, argomento che approfondiremo nel Capitolo 4. Per ora è sufficiente sapere che un indirizzo IP è un numero di 32 bit che possiamo pensare identifichi univocamente l’host. Oltre a conoscere l’indirizzo dell’host cui è destinato il messaggio, il mittente deve anche identificare il processo destinatario, più specificatamente la socket che deve ricevere il dato. Questa informazione è necessaria in quanto, in generale, sull’host potrebbero essere in esecuzione molte applicazioni di rete. Un **numero di porta di destinazione** assolve questo compito. Alle applicazioni più note sono stati assegnati numeri di porta specifici. Per esempio, i web server sono identificati dal numero di porta 80. Il processo di un server di posta che usa il protocollo SMTP è identificato dal numero di porta 25. La lista di numeri di porta noti per tutti i protocolli standard Internet può essere reperita su <http://www.iana.org/>. Esamineremo nel dettaglio i numeri di porta nel Capitolo 3.

### 2.1.3 Servizi di trasporto disponibili per le applicazioni

Ricordiamo che una socket è l’interfaccia tra un processo applicativo e il protocollo a livello di trasporto. L’applicazione lato mittente spinge fuori i messaggi tramite la socket; dall’altra parte, lato ricevente, il protocollo a livello di trasporto ha la responsabilità di consegnare i messaggi alla socket del processo ricevente.

Molte reti, Internet inclusa, mettono a disposizione vari protocolli di trasporto. Nel progetto di un’applicazione occorre scegliere il protocollo a livello di trasporto. Ma come? Occorre valutare i servizi resi disponibili dai protocolli a livello di trasporto e scegliere quello che fornisce i servizi più confacenti all’applicazione. La situazione è simile alla scelta che possiamo effettuare doven-  
do viaggiare da una città a un’altra, tra il treno e l’aereo, cioè tra due modalità di trasporto che offrono differenti caratteristiche (per esempio, il treno consente di partire e arrivare in centro città, mentre l’aereo offre tempi di percorrenza assai più brevi sulle lunghe distanze).

Quali servizi il protocollo a livello di trasporto può offrire a un’applicazione che li invoca? Li possiamo classificare a grandi linee secondo quattro dimensioni: trasferimento dati affidabile, throughput, temporizzazione e sicurezza.

#### Trasferimento dati affidabile

Come abbiamo visto nel Capitolo 1, in una rete di calcolatori i pacchetti possono andare perduti. Per esempio, un pacchetto può far traboccare un buffer in un router o può essere scartato da un host o da un router in quanto alcuni suoi bit sono corrotti. In alcune applicazioni – quali posta elettronica, messaggistica istantanea, trasmissione di file, accesso a host remoti, invio di documenti web e applicazioni finanziarie – la perdita di informazioni potrebbe causare gravi conseguenze. Quindi, per supportare queste applicazioni, occorre garantire che i dati inviati siano consegnati corretti e completi. Se un protocollo fornisce questo tipo di servizio di consegna garantita dei dati, si dice che fornisce un **trasfe-  
rimento dati affidabile** (*reliable data transfer*). Un importante servizio che un protocollo a livello di trasporto può potenzialmente fornire a un’applicazione è il trasferimento dati affidabile tra processi. Quando un protocollo a livello di trasporto fornisce tale servizio, il processo mittente può passare i propri dati alla socket e sapere con assoluta certezza che quei dati arriveranno senza errori al processo ricevente.

Quando un protocollo a livello di trasporto non fornisce trasferimento dati affidabile, i dati inviati dal processo mittente potrebbero non arrivare mai a quello ricevente. Ciò potrebbe essere accettabile per le **applicazioni che tolle-  
rano le perdite** (*loss-tolerant applications*): in particolare le applicazioni multi-  
mediali audio/video a uso personale possono tollerare una certa quantità di dati perduti. In queste applicazioni multimediali, le perdite di dati possono dar luogo a un piccolo difetto nell’audio/video riprodotto che però rappresenta un dete-  
rioramento non critico.

#### Throughput

Nel Capitolo 1 abbiamo introdotto il concetto di throughput disponibile che, nel contesto di una sessione di comunicazione tra due processi lungo un per-

corso in rete, è la velocità al quale il processo mittente può inviare i bit al processo ricevente. Dato che altre sessioni condivideranno la banda sul percorso di rete e poiché queste sessioni verranno istituite e rilasciate dinamicamente, il throughput disponibile può fluttuare nel tempo. Queste osservazioni ci guidano verso un altro naturale servizio che un protocollo a livello di trasporto può fornire, cioè un throughput disponibile garantito. Con tale servizio l'applicazione può richiedere un throughput garantito di  $r$  bps e il protocollo di trasporto assicurerà poi che il throughput disponibile sia sempre almeno di  $r$  bps. Questo tipo di servizio di throughput garantito interesserebbe a molte applicazioni; per esempio, un'applicazione di telefonia su Internet che codifica la voce a 32 kbps ha necessità che i dati siano inviati nella rete e siano consegnati all'applicazione ricevente alla stessa velocità. Se il protocollo a livello di trasporto non può fornire questo throughput, l'applicazione deve codificare i dati a un livello inferiore (per ricevere un throughput sufficiente per sostenere questa codifica inferiore) o rinunciare; infatti ricevere, per esempio, la metà del throughput necessario le può servire poco o nulla. Le applicazioni che hanno requisiti di throughput vengono dette **applicazioni sensibili alla banda** (*bandwidth-sensitive applications*). Molte delle odiere applicazioni multimediali sono sensibili alla banda, sebbene alcune possano utilizzare tecniche di codifica adattativa per codificare a una banda che coincida con il throughput disponibile in quel momento.

Mentre le applicazioni sensibili alla banda hanno requisiti specifici di throughput, le **applicazioni elastiche** (*elastic applications*) possono far uso di tanto o di poco throughput, a seconda di quanto ce ne sia disponibile. La posta elettronica, il trasferimento di file e il Web sono tutte applicazioni elastiche. Ovviamente, maggiore è il throughput, meglio è.

Come dice il proverbio, non si può essere troppo ricchi, troppo magri, o avere troppo throughput!

## Temporizzazione

Un protocollo a livello di trasporto può anche fornire garanzie di temporizzazione (*timing*) che, come quelle per il throughput, possono assumere varie forme. Per esempio, la garanzia potrebbe essere che ogni bit che il mittente invia sulla socket venga ricevuto dalla socket di destinazione non più di 100 millisecondi più tardi. Questo tipo di servizio potrebbe interessare le applicazioni interattive in tempo reale (come la telefonia Internet, gli ambienti virtuali, la teleconferenza e i giochi multiutente) che, per essere efficaci, richiedono stretti vincoli temporali sulla consegna dei dati [Gauthier 1999; Ramjee 1994]. Lunghi ritardi nella telefonia via Internet, per esempio, tendono a causare pause innaturali durante la conversazione. Similmente in un gioco multiutente o in un ambiente virtuale interattivo, un lungo ritardo tra l'azione e la corrispondente risposta rendono l'applicazione meno realistica. Per le applicazioni non in tempo reale, ritardi inferiori sono sempre preferibili a ritardi più consistenti, ma non si pongono stretti vincoli sui ritardi end-to-end.

## Sicurezza

Infine, un protocollo a livello di trasporto può fornire a un'applicazione uno o più servizi di sicurezza. Per esempio, nell'host mittente, un protocollo di trasporto può cifrare tutti i dati trasmessi dal processo mittente e, nell'host di destina-

zione, il protocollo di trasporto può decifrare i dati prima di consegnarli al processo ricevente. Questo tipo di servizio fornirebbe riservatezza tra i due processi, anche se i dati vengono in qualche modo osservati tra il processo mittente e ricevente. Un protocollo a livello di trasporto può fornire altri servizi di sicurezza oltre alla riservatezza, compresi l'integrità dei dati e l'autenticazione che tratteremo in dettaglio nel Capitolo 8, disponibile in inglese sulla piattaforma MyLab.

### 2.1.4 Servizi di trasporto offerti da Internet

Fin qui abbiamo considerato i servizi di trasporto che una rete di calcolatori potrebbe fornire in generale. Entriamo ora nello specifico ed esaminiamo il tipo di supporto alle applicazioni fornito da Internet. Internet, come ogni rete TCP/IP, mette a disposizione delle applicazioni due protocolli di trasporto: UDP e TCP. Quando gli sviluppatori di software realizzano una nuova applicazione di rete per Internet, una delle prime decisioni che devono prendere riguarda la scelta tra TCP e UDP: due protocolli che offrono modelli di servizio diversi. La Figura 2.4 riassume i requisiti di alcune applicazioni in Internet.

#### Servizi di TCP

TCP prevede un servizio orientato alla connessione e il trasporto affidabile dei dati. Quando un'applicazione invoca TCP come protocollo di trasporto, riceve entrambi questi servizi.

- *Servizio orientato alla connessione (connection-oriented service).* TCP fa in modo che client e server si scambino informazioni di controllo a livello di trasporto *prima* che i messaggi a livello di applicazione comincino a fluire. Questa procedura, detta di *handshaking*, mette in allerta client e server, preparandoli alla partenza dei pacchetti. Dopo la fase di handshaking, si dice che esiste una **connessione TCP** tra le socket dei due processi. Tale connessione è full-duplex, nel senso che i due processi possono scambiarsi contemporaneamente messaggi sulla connessione. L'applicazione deve chiudere la connessione quando termina di inviare messaggi. Nel corso del Capitolo 3 discuteremo il servizio orientato alla connessione e ne esamineremo l'implementazione.

| Applicazione                            | Tolleranza alla perdita di dati | Throughput  | Sensibilità al fattore tempo |
|---|---------------------------------|---|------------------------------|
| Trasferimento file/download             | No                              | Variabile   | No                           |
| E-mail                                  | No                              | Variabile   | No                           |
| Web                                     | No                              | Variabile (pochi kbps)                                      | No                           |
| Telefonia su Internet/ video conferenza | Si                              | Audio: da pochi kbps a 1 Mbps<br>Video: da 10 kbps a 5 Mbps | Si: centinaia di ms          |
| Streaming audio/video memorizzati       | Si                              | Come sopra  | Si: pochi secondi            |
| Giochi interattivi                      | Si                              | Fino a 10 kbps  | Si: centinaia di ms          |
| Messaggistica istantanea                | No                              | Variabile   | Si e no                      |

**Figura 2.4** Requisiti di alcune applicazioni di rete.

**BOX 2.1** **FOCUS SULLA SICUREZZA****Rendere sicuro TCP**

Né TCP né UDP forniscono forme di cifratura: i dati che il processo mittente passa alla sua socket sono gli stessi che viaggiano in rete fino al processo destinatario. Così, per esempio, se il processo mittente manda una password in chiaro, cioè non cifrata, tramite socket, questa viaggerà in chiaro su tutti i collegamenti tra mittente e ricevente e potenzialmente potrà essere intercettata e individuata in uno qualsiasi dei collegamenti coinvolti. Dato che riservatezza e altre questioni di sicurezza sono diventate critiche per molte applicazioni, la comunità di Internet ha sviluppato un elemento aggiuntivo per TCP, chiamato **Transport Layer Security (TLS)** [RFC 5246]. TCP, arricchito da TLS, non solo fa tutto ciò di cui è capace il TCP tradizionale, ma fornisce anche servizi critici di sicurezza tra processi, compresi la cifratura, il controllo dell'integrità dei dati e l'autenticazione end-to-end. Sottolineiamo che TLS non è un terzo protocollo di trasporto per Internet, allo stesso livello di TCP e UDP, ma un arricchimento di TCP, dove gli arricchimenti sono implementati a livello di applicazione. In particolare, se un'applicazione vuole usare i servizi di TLS, ha necessità di includere del codice TLS (library e classi esistenti molto ottimizzate) sia sul lato client sia sul lato server dell'applicazione. TLS possiede delle proprie API per le socket, simili alle API per le socket tradizionali di TCP. Quando un'applicazione utilizza SSL, il processo mittente trasferisce i dati in chiaro alla socket TLS, che li cifra e li passa alla socket TCP. I dati cifrati viaggiano su Internet sino alla socket del processo ricevente, la quale passa i dati cifrati a TLS, che li decifra. Infine, TLS passa i dati in chiaro attraverso la sua socket TLS al processo ricevente. Tratteremo TLS in maggior dettaglio nel Capitolo 8, disponibile in inglese sulla piattaforma MyLab.

- *Servizio di trasferimento affidabile (reliable data transfer service).* I processi comunicanti possono contare su TCP per trasportare i dati senza errori e nel giusto ordine. Quando un lato dell'applicazione passa un flusso di byte alla sua socket, può affidarsi a TCP per consegnare lo stesso flusso di byte alla socket di ricezione, senza perdita o duplicazione di byte.

TCP include anche un meccanismo di controllo della congestione, un servizio che riguarda il benessere generale di Internet e non quello diretto dei processi comunicanti. Il meccanismo di controllo della congestione esegue una “strozzatura” del processo d’invio (client o server) quando il traffico in rete appare eccessivo. Come vedremo nel Capitolo 3, il controllo di congestione di TCP cerca anche di confinare le connessioni TCP all’interno della loro porzione di ampiezza di banda.

**Servizi di UDP**

UDP è un protocollo di trasporto leggero e senza fronzoli, dotato di un modello di servizio minimalista. UDP è senza connessione, non necessita quindi di handshaking, e fornisce un servizio di trasferimento dati non affidabile. Così, quando un processo invia un messaggio tramite la socket UDP, il protocollo non garantisce che questo raggiunga il processo di destinazione. Inoltre i messaggi potrebbero giungere a destinazione non in ordine.

UDP non include un meccanismo di controllo della congestione, pertanto un processo d’invio UDP può “spingere” i dati al livello sottostante (di rete) a

qualsiasi velocità. Si noti, tuttavia, che il reale throughput end-to-end disponibile può essere inferiore a tale velocità, a causa della banda limitata dei collegamenti coinvolti o a causa della congestione.

### Servizi non forniti dai protocolli di trasporto Internet

Abbiamo organizzato i possibili servizi a livello di trasporto lungo quattro dimensioni: trasporto affidabile dei dati, throughput, temporizzazione e sicurezza. Quali di questi servizi sono forniti da TCP e UDP? Abbiamo già evidenziato che TCP fornisce un trasporto affidabile end-to-end e sappiamo che TCP può essere facilmente arricchito a livello applicativo con TLS per fornire servizi di sicurezza. Ma nella nostra breve descrizione di TCP e UDP è evidentemente assente qualsiasi riferimento a garanzie di throughput e temporizzazione, servizi non forniti dagli odierni protocolli di trasporto di Internet. Ciò significa che le applicazioni in cui i ritardi relativi sono un fattore critico, come la telefonia, non possono funzionare sull'Internet odierna? La risposta è chiaramente negativa: Internet ospita questo genere di applicazioni da molti anni. Tali applicazioni in genere funzionano piuttosto bene in quanto sono state progettate per convivere, nel miglior modo possibile, con l'assenza di garanzie. Anche una progettazione intelligente incontra però dei limiti quando il ritardo è eccessivo o il throughput è molto limitato. In conclusione, attualmente Internet può, in molti casi, offrire un servizio soddisfacente alle applicazioni sensibili ai ritardi, ma non garanzie sulla temporizzazione o sul throughput disponibile.

La Figura 2.5 indica i protocolli di trasporto utilizzati da alcune famose applicazioni Internet. Vediamo che posta elettronica, accesso a terminale remoto, Web e trasferimento file usano tutti TCP. Questo soprattutto perché TCP fornisce un servizio di trasferimento dati affidabile, garantendo che, prima o poi, tutti i dati giungano a destinazione. Poiché le applicazioni di telefonia Internet come Skype possono tollerare perdite, ma hanno requisiti minimi di velocità di trasmissione per essere efficienti, gli sviluppatori scelgono UDP per evitare il controllo di congestione di TCP e i conseguenti messaggi aggiuntivi. Tuttavia, siccome molti firewall sono configurati per bloccare la maggior parte del traffico UDP, molte applicazioni di telefonia su Internet sono progettate per usare TCP come opzione di riserva nei casi in cui UDP non sia utilizzabile.

| Applicazione               | Protocollo a livello di applicazione                                | Protocollo di trasporto sottostante |
|----------------------------|---|-------------------------------------|
| Posta elettronica          | SMTP [RFC 5321]   | TCP                                 |
| Accesso a terminali remoti | Telnet [RFC 854]  | TCP                                 |
| Web                        | HTTP 1.1 (RFC 7230)   | TCP                                 |
| Trasferimento file         | FTP [RFC 959]   | TCP                                 |
| Multimedia streaming       | HTTP (per esempio: YouTube), DASH                                   | TCP                                 |
| Telefonia Internet         | SIP [RFC 3261], RTP [RFC 3550], o proprietario (per esempio: Skype) | UDP o TCP                           |

**Figura 2.5** Alcune applicazioni Internet con relativo protocollo a livello di applicazione e sottostante protocollo a livello di trasporto.

### 2.1.5 Protocolli a livello di applicazione

Abbiamo visto come i processi di rete comunichino tra loro inviando messaggi tra socket. Ma come sono strutturati questi messaggi? Qual è il significato dei loro campi? Quando vengono inviati? Queste domande ci conducono nel campo dei protocolli a livello di applicazione. Un **protocollo a livello di applicazione** definisce come i processi di un'applicazione, in esecuzione su sistemi periferici diversi, si scambiano i messaggi. In particolare, un protocollo a livello di applicazione definisce:

- i tipi di messaggi scambiati (per esempio, di richiesta o di risposta)
- la sintassi dei vari tipi di messaggio (per esempio, quali sono i campi nel messaggio e come vengono descritti)
- la semantica dei campi, ossia il significato delle informazioni che contengono
- le regole per determinare quando e come un processo invia e risponde ai messaggi.

Alcuni protocolli a livello di applicazione vengono specificati nelle RFC e sono pertanto di pubblico dominio. Per esempio, il protocollo a livello di applicazione del Web, HTTP (*HyperText Transfer Protocol*) è descritto nella RFC 7230. Se lo sviluppatore di un browser si attiene alle regole che vi sono esposte, il suo browser sarà in grado di recuperare pagine web da qualsiasi server che segua quelle stesse regole. Altri protocolli a livello di applicazione sono privati e volutamente non disponibili al pubblico (per esempio Skype).

È importante distinguere tra applicazioni di rete e protocolli a livello di applicazione. Un protocollo a livello di applicazione è solo una parte (benché molto importante) di un'applicazione di rete. Consideriamo un paio di esempi. Il Web è un'applicazione client-server che consente agli utenti di ottenere su richiesta documenti dai web server. L'applicazione web consiste di molte componenti, tra cui uno standard per i formati di documento (HTML), un browser (per esempio Chrome e Microsoft Internet Explorer), un web server (per esempio Apache e Microsoft) e un protocollo a livello di applicazione. HTTP, il protocollo a livello di applicazione del Web, definisce il formato e la sequenza dei messaggi scambiati tra browser e web server. È, pertanto, solo una parte dell'applicazione web (benché importante). Come ulteriore esempio, vedremo nel Paragrafo 2.6 che anche il servizio video di Netflix ha molte componenti, tra cui server che archiviano e trasmettono video, altri server che gestiscono la fatturazione e altre funzioni, client quali la app di Netflix su smartphone, tablet o computer e un protocollo DASH a livello di applicazione che definisce il formato e la sequenza di messaggi scambiati tra un server e un client Netflix. Pertanto, DASH è solo una componente, anche se importante, dell'applicazione Netflix.

### 2.1.6 Applicazioni di rete trattate in questo libro

Ogni giorno vengono sviluppate nuove applicazioni per Internet. Così, piuttosto che trattarne molte in modo encyclopedico, abbiamo scelto di concentrarci su quelle più diffuse e importanti. In questo capitolo ne prendiamo in considerazione cinque: il Web, la posta elettronica, il servizio di directory, lo streaming video e le applicazioni P2P. Per prima cosa tratteremo il Web, non solo in quanto

rappresenta un'applicazione estremamente diffusa, ma anche perché il suo protocollo a livello di applicazione, HTTP, è diretto e facile da comprendere. Esamineremo quindi la posta elettronica, la prima fondamentale applicazione Internet, più complessa rispetto al Web, in quanto utilizza non uno, ma svariati protocolli a livello di applicazione. Di seguito, tratteremo il DNS che fornisce a Internet un servizio di directory.<sup>3</sup> La maggior parte degli utenti non interagisce direttamente con il DNS, ma lo invoca indirettamente attraverso le proprie applicazioni (tra cui il Web, il trasferimento di file e la posta elettronica). DNS mostra, in modo elegante, come in Internet si possa implementare una funzionalità chiave della rete, quale la traduzione da nome di rete a indirizzo di rete a livello applicativo. Vedremo infine le applicazioni P2P di distribuzione di file e lo streaming video on demand, trattando anche le reti di distribuzione di contenuti.

## 2.2 Web e HTTP

Fino agli anni '90 Internet veniva principalmente utilizzata da ricercatori, docenti e studenti universitari per raggiungere host remoti, trasferire file, ricevere e inviare notizie e per la posta elettronica. Sebbene tali applicazioni fossero (e continuino a essere) estremamente utili, Internet era sostanzialmente sconosciuta al di fuori delle università e dei centri di ricerca. Poi, nei primi anni '90, comparve sulla scena una nuova importante applicazione: il World Wide Web [Berners-Lee 1994]. Il Web è stata la prima applicazione Internet che ha catturato l'attenzione del pubblico, cambiando profondamente il modo di interagire all'interno e all'esterno degli ambienti di lavoro. Il Web ha elevato Internet dal semplice rango di “una tra le reti per i dati” al rango di sola e unica rete per i dati.

Forse ciò che attira di più gli utenti è che il Web opera su richiesta (*on demand*): si può avere ciò che si vuole, quando lo si vuole. Questo aspetto lo differenzia sostanzialmente dalla trasmissione radiotelevisiva, dove gli utenti ricevono i contenuti solo nel momento in cui il fornitore li rende disponibili. Oltre a operare su richiesta, il Web presenta molte altre interessanti caratteristiche. È facilissimo per tutti rendere disponibili informazioni sul Web: chiunque può diventare editore a costi estremamente bassi. I collegamenti ipertestuali e i motori di ricerca ci aiutano a navigare in un oceano di siti. Foto e video stimolano i nostri sensi. I form, JavaScript, video e molti altri elementi ci consentono di interagire con le pagine e con i siti web. Inoltre il Web fornisce una piattaforma a molte applicazioni killer, tra cui YouTube, servizi di posta elettronica come Gmail, Instagram e Google Maps.

### 2.2.1 Panoramica di HTTP

**HTTP** (*HyperText Transfer Protocol*), protocollo a livello di applicazione del Web, definito in [RFC 1945, RFC 7230 e RFC 7540], costituisce il cuore del Web. Questo protocollo è implementato in due programmi, client e server, in esecuzione su sistemi periferici diversi che comunicano tra loro scambiandosi

---

<sup>3</sup> In questa accezione il termine inglese va letto come “guida” o “elenco telefonico”, e non con l’usuale traduzione di “cartella” come nel campo dei sistemi operativi (N.d.R.).

messaggi HTTP. Il protocollo definisce sia la struttura dei messaggi sia la modalità con cui client e server si scambiano i messaggi. Prima di affrontare in dettaglio HTTP soffermiamoci brevemente sulla terminologia web.

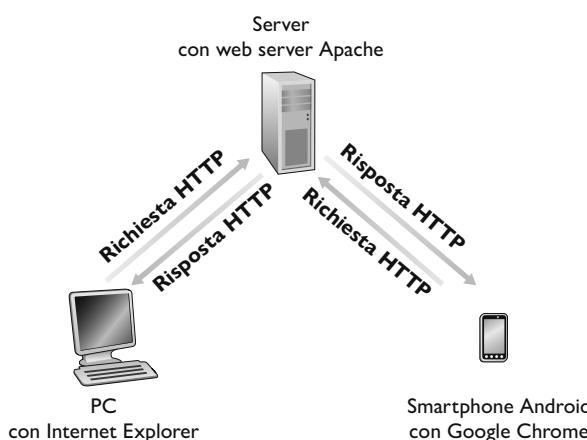
Una **pagina web** (*web page*), detta anche documento, è costituita da oggetti. Un **oggetto** (*object*) è semplicemente un file (quale un file HTML, un'immagine JPEG, un file Javascript, un foglio di stile CSS, una clip video e così via) indirizzabile tramite un URL. La maggioranza delle pagine web consiste di un **file HTML principale** (*base HTML file*) e diversi oggetti referenziati da esso. Per esempio, se una pagina web contiene testo in HTML e cinque immagini JPEG, allora la pagina nel complesso presenta sei oggetti: il file HTML più le cinque immagini. Il file HTML riferisce gli altri oggetti nella pagina tramite il loro URL. Ogni URL ha due componenti: il nome dell'host del server che ospita l'oggetto e il percorso dell'oggetto. Per esempio, l'URL

```
http://www.someSchool.edu/someDepartment/picture.gif
```

ha `www.someSchool.edu` come nome dell'host e `/someDepartment/picture.gif` come percorso. Un **browser web** (come Internet Explorer o Chrome) implementa il lato client di HTTP (quando parliamo di Web useremo le parole *browser* e *client* in modo intercambiabile). Un **web server**, che implementa il lato server di HTTP, ospita oggetti web, indirizzabili tramite URL. Tra i più popolari ricordiamo Apache e Microsoft Internet Information Server.

HTTP definisce in che modo i client web richiedono le pagine ai web server e come questi ultimi le trasferiscono ai client. Tratteremo più avanti l'interazione tra client e server, ma l'idea generale è illustrata nella Figura 2.6. Quando l'utente richiede una pagina web (per esempio, cliccando su un collegamento ipertestuale), il browser invia al server messaggi di richiesta HTTP per gli oggetti nella pagina. Il server riceve le richieste e risponde con messaggi di risposta HTTP contenenti gli oggetti.

HTTP utilizza TCP (anziché UDP) come protocollo di trasporto. Il client HTTP per prima cosa inizia una connessione TCP con il server. Una volta stabilita, i processi client e server accedono a TCP attraverso le proprie socket. Come descritto nel Paragrafo 2.1 l'interfaccia socket è la porta tra un processo e la sua connessione TCP. Il client invia richieste e riceve risposte HTTP tramite la propria interfaccia socket, analogamente il server riceve richieste e invia mes-



**Figura 2.6**  
Comportamento  
richiesta-risposta di HTTP.

saggi di risposta attraverso la propria interfaccia socket. Quando il client ha mandato un messaggio alla sua interfaccia socket, questo non è più in suo possesso, ma si trova “nelle mani” di TCP. Ricordiamo dal Paragrafo 2.1 che TCP mette a disposizione di HTTP un servizio di trasferimento dati affidabile; ciò implica che ogni messaggio di richiesta HTTP emesso da un processo client arriverà intatto al server e viceversa. Questo è uno dei grandi vantaggi di un’architettura organizzata a livelli: HTTP non si deve preoccupare dei dati smarriti o di come TCP recuperi le perdite o riordini i dati all’interno della rete: questi sono compiti di TCP e dei protocolli di livello inferiore.

È importante notare che il server invia i file richiesti ai client senza memorizzare alcuna informazione di stato a proposito del client. Per cui, in caso di ulteriore richiesta dello stesso oggetto da parte dello stesso client, anche nel giro di pochi secondi, il server procederà nuovamente all’invio, non avendo mantenuto alcuna traccia di quello precedentemente effettuato. Dato che i server HTTP non mantengono informazioni sui client, HTTP è classificato come **protocollo senza memoria di stato** (*stateless protocol*). Un web server è sempre attivo, ha un indirizzo IP fisso e risponde potenzialmente alle richieste provenienti da milioni di diversi browser.

La versione originale di HTTP si chiama HTTP / 1.0 e risale agli inizi degli anni 1990 [RFC 1945]. A partire dal 2020, la maggior parte delle transazioni HTTP avviene su HTTP/1.1 [RFC 7230]. Tuttavia, sempre più browser e web server supportano una nuova versione di HTTP chiamata HTTP/2 [RFC 7540] di cui forniremo un’introduzione alla fine del paragrafo.

### 2.2.2 Connessioni persistenti e non persistenti

In molte applicazioni per Internet, client e server comunicano per un lungo periodo di tempo, con il client che inoltra una serie di richieste e il server che risponde a ciascuna di esse. A seconda dell’applicazione e del suo impiego la serie di richieste potrebbe essere effettuata in sequenza, periodicamente a intervalli regolari o in maniera intermittente. Quando tale interazione client-server ha luogo su TCP, gli sviluppatori dell’applicazione devono prendere una decisione importante: ciascuna coppia richiesta/risposta deve essere inviata su una connessione TCP *separata* o devono essere inviate tutte *sulla stessa* connessione TCP? Nel primo approccio si dice che l’applicazione usa **connessioni non persistenti**, mentre nel secondo usa **connessioni persistenti**. Per avere una maggiore comprensione degli aspetti progettuali, esaminiamo vantaggi e svantaggi delle connessioni persistenti nel contesto di un’applicazione specifica, HTTP, che può usare entrambi i tipi di connessioni. Sebbene HTTP nella sua modalità di default usi connessioni persistenti, i client e i server HTTP possono essere configurati per usare quelle non persistenti.

#### HTTP con connessioni non persistenti

Seguiamo passo dopo passo il trasferimento di una pagina web dal server al client nel caso di connessioni non persistenti. Supponiamo che la pagina consista di un file HTML principale e di 10 immagini JPEG, e che tutti gli undici oggetti risiedano sullo stesso server. Ipotizziamo che l’URL del file HTML principale sia:

`http://www.someSchool.edu/someDepartment/home.index`

Ecco che cosa avviene.

1. Il processo client HTTP inizializza una connessione TCP con il server `www.someSchool.edu` sulla porta 80, che è la porta di default per HTTP. Associate alla connessione TCP ci saranno una socket per il client e una per il server.
2. Il client HTTP, tramite la propria socket, invia al server un messaggio di richiesta HTTP che include il percorso `/someDepartment/home.index`. Tratteremo in dettaglio i messaggi HTTP più avanti.
3. Il processo server HTTP riceve il messaggio di richiesta attraverso la propria socket associata alla connessione, recupera l'oggetto `/someDepartment-/home.index` dalla memoria (centrale o di massa), lo incapsula in un messaggio di risposta HTTP che viene inviato al client attraverso la socket.
4. Il processo server HTTP comunica a TCP di chiudere la connessione. Questo, però, non termina la connessione finché non sia certo che il client abbia ricevuto integro il messaggio di risposta.
5. Il client HTTP riceve il messaggio di risposta. La connessione TCP termina. Il messaggio indica che l'oggetto incapsulato è un file HTML. Il client estrae il file dal messaggio di risposta, esamina il file HTML e trova i riferimenti ai 10 oggetti JPEG.
6. Vengono quindi ripetuti i primi quattro passi per ciascuno degli oggetti JPEG referenziati.

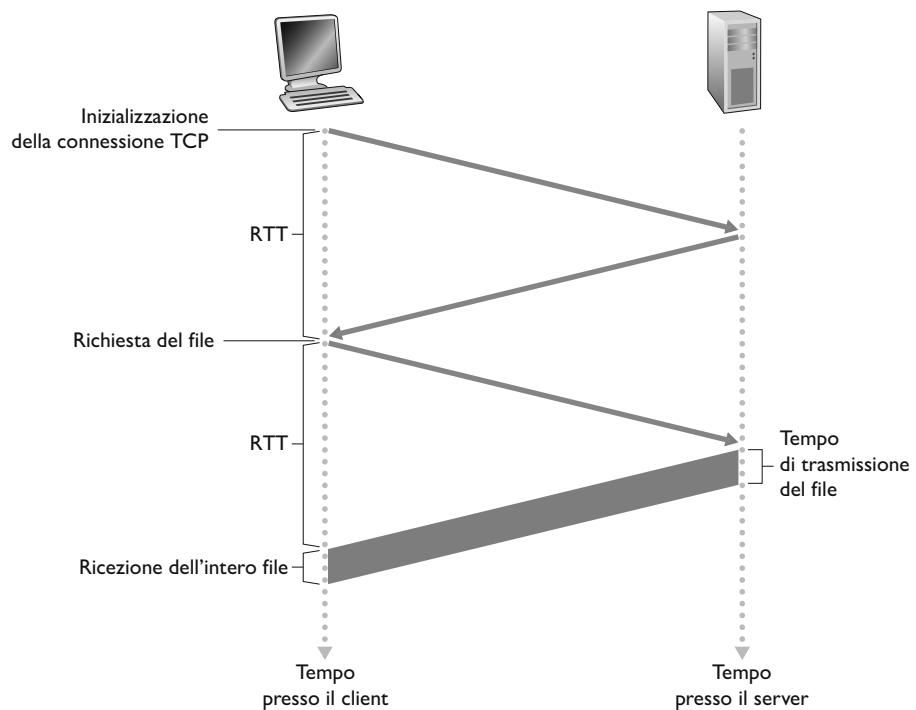
Quando il browser riceve la pagina web, la visualizza all'utente. Due browser diversi possono interpretare (e quindi mostrare all'utente) una stessa pagina web secondo modalità leggermente diverse. HTTP non ha niente a che vedere con l'interpretazione della pagina web da parte di un client. Le specifiche HTTP ([RFC 1945] e [RFC 7540]) definiscono solo il protocollo di comunicazione tra il programma HTTP client e il programma HTTP server.

I passi appena riportati illustrano l'utilizzo di connessioni non persistenti in cui ogni connessione TCP viene chiusa dopo l'invio dell'oggetto da parte del server: vale a dire che ciascuna trasporta soltanto un messaggio di richiesta e un messaggio di risposta. Pertanto, in questo esempio, quando l'utente richiede una pagina web, vengono generate 11 connessioni TCP. HTTP/1.0 utilizza connessioni TCP non persistenti.

Nei passi descritti sopra siamo rimasti intenzionalmente vaghi sul fatto che il client ottenga le 10 immagini JPEG su 10 connessioni TCP in serie anziché in parallelo. Infatti, alcuni browser possono essere configurati dagli utenti per controllare il grado di parallelismo. In modalità di default, la maggior parte dei browser apre da 5 a 10 connessioni TCP parallele, ognuna delle quali gestisce una transazione di richiesta-risposta. Se lo desidera, l'utente può impostare il numero di connessioni parallele a 1, caso in cui le 10 connessioni vengono stabilite in serie. Come vedremo nel prossimo capitolo, l'uso di connessioni in parallelo abbrevia i tempi medi di risposta.

Prima di procedere, facciamo un calcolo approssimativo per stimare l'intervallo di tempo che intercorre tra la richiesta di un file HTML da parte del client e il momento in cui l'intero file viene ricevuto dal client. A questo scopo, definiamo il **round-trip time (RTT)**, che rappresenta il tempo impiegato da un pic-

**Figura 2.7** Calcolo approssimato del tempo necessario per richiedere e ricevere un file HTML.



colo pacchetto per viaggiare dal client al server e poi tornare al client. RTT include i ritardi di propagazione, di accodamento nei router e nei commutatori intermedi nonché di elaborazione del pacchetto (trattati nel Paragrafo 1.4). Consideriamo ora che cosa succede quando un utente fa click con il mouse su un collegamento ipertestuale. Come mostrato nella Figura 2.7, questa azione fa sì che il browser inizializzi una connessione TCP con il web server. Ciò comporta un **handshake a tre vie** (*three-way handshake*): il client invia un piccolo segmento TCP al server e quest'ultimo manda una conferma per mezzo di un piccolo segmento TCP. Infine il client dà anch'esso una conferma di ritorno al server. Le prime due parti dell'handshake a tre vie richiedono un RTT. Dopo il loro completamento, il client invia un messaggio di richiesta HTTP combinato con la terza parte dell'handshake, la conferma di avvenuta ricezione (*acknowledgement*), tramite la connessione TCP. Quando il messaggio di richiesta arriva al server, quest'ultimo inoltra il file HTML sulla connessione TCP. La richiesta-risposta HTTP consuma un altro RTT. Pertanto, il tempo di risposta totale è, approssimativamente, di due RTT più il tempo di trasmissione da parte del server del file HTML.

### HTTP con connessioni persistenti

Le connessioni non persistenti presentano alcuni limiti: il primo è che per ogni oggetto richiesto occorre stabilire e mantenere una nuova connessione. Per ciascuna di queste connessioni si devono allocare buffer e mantenere variabili TCP sia nel client sia nel server. Ciò pone un grave onere sul web server, che può dover servire contemporaneamente richieste provenienti da centinaia di diversi

client. In secondo luogo, come abbiamo appena descritto, ciascun oggetto subisce un ritardo di consegna di due RTT, uno per stabilire la connessione TCP e uno per richiedere e ricevere un oggetto.

Con HTTP 1.1 nelle connessioni persistenti il server lascia la connessione TCP aperta dopo l'invio di una risposta, per cui le richieste e le risposte successive tra gli stessi client e server possono essere trasmesse sulla stessa connessione. In particolare, non solo il server può inviare un'intera pagina web (nell'esempio, il file HTML principale e le 10 immagini) su una sola connessione TCP permanente, ma può anche spedire allo stesso client più pagine web. Queste richieste di oggetti possono essere effettuate una di seguito all'altra senza aspettare le risposte delle richieste pendenti (*pipelining*). In generale, il server HTTP chiude la connessione quando essa rimane inattiva per un dato lasso di tempo (un intervallo configurabile). Quando il server riceve richieste in sequenza, invia gli oggetti con la stessa modalità. La modalità di default di HTTP impiega connessioni persistenti con pipelining. Confronteremo quantitativamente le prestazioni delle connessioni non persistenti e persistenti nei problemi elencati alla fine di questo capitolo e del Capitolo 3. Vi suggeriamo inoltre la lettura di [Heidemann 1997], [Nielsen 1997] e [RFC 7540].

### 2.2.3 Formato dei messaggi HTTP

Le specifiche HTTP [RFC 1945; RFC 7230; RFC 7540] includono la definizione dei due formati dei messaggi HTTP, di richiesta e di risposta.

#### Messaggio di richiesta HTTP

Ecco un tipico messaggio di richiesta HTTP:

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

Osservandolo accuratamente, notiamo innanzitutto che il messaggio è scritto in testo ASCII, in modo che l'utente sia in grado di leggerlo. Inoltre, notiamo che consiste di cinque righe, ciascuna seguita da un carattere di ritorno a capo (*carriage return*) e un carattere di nuova linea (*line feed*).<sup>4</sup> L'ultima riga è seguita da una coppia di caratteri di ritorno a capo e nuova linea aggiuntivi. In generale, i messaggi di richiesta possono essere costituiti da un numero indefinito di righe, anche una sola. La prima riga è detta **riga di richiesta** (*request line*), quelle successive **righe di intestazione** (*header lines*). La riga di richiesta presenta tre campi: il campo metodo, il campo URL e il campo versione di HTTP. Il campo metodo può assumere diversi valori, tra cui GET, POST, HEAD, PUT e DELETE. La maggioranza dei messaggi di richiesta HTTP usa il metodo GET, adottato quando il browser richiede un oggetto identificato dal campo URL. La versione è

---

<sup>4</sup> Caratteri ovviamente non visibili se non per il fatto che vi sono dei ritorni a capo nel testo (N.d.R.).

auto esplicativa: nell'esempio, il browser, che implementa la versione HTTP/1.1, sta richiedendo l'oggetto `/somedir/page.html`.

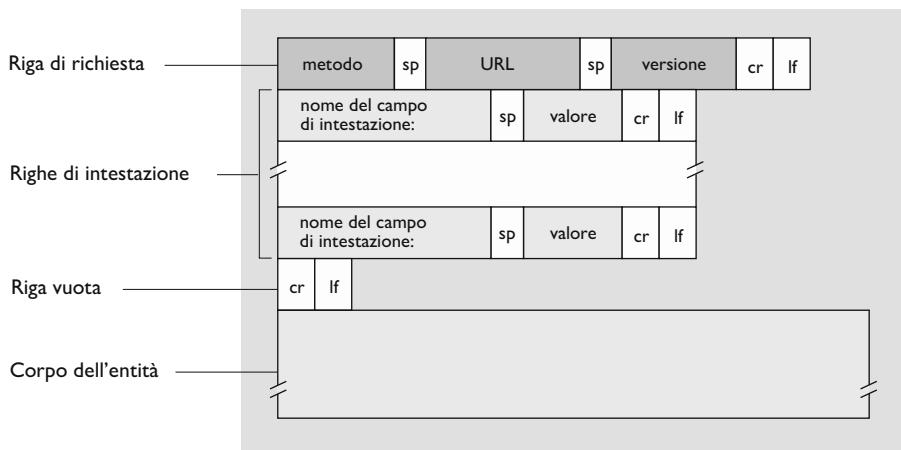
Consideriamo ora le righe di intestazione dell'esempio.

La riga `Host: www.someschool.edu` specifica l'host su cui risiede l'oggetto. Si potrebbe pensare che questa riga di intestazione non sia necessaria, dato che è già in corso una connessione TCP con l'host. Ma, come vedremo nel Paragrafo 2.2.5, l'informazione fornita dalla linea di intestazione dell'host viene richiesta dalle cache dei proxy. Includendo la linea di intestazione `Connection: close`, il browser sta comunicando al server che non si deve occupare di connessioni persistenti, ma vuole che questi chiuda la connessione dopo aver inviato l'oggetto richiesto. La riga di intestazione `User-agent: Mozilla/5.0`, un browser Firefox. Questa riga è utile in quanto il server può inviare versioni diverse dello stesso oggetto a browser di tipi diversi. Ciascuna delle versioni viene indirizzata dallo stesso URL. Infine `Accept-language: fr`: indica che l'utente preferisce ricevere una versione in francese dell'oggetto se disponibile; altrimenti, il server dovrebbe inviare la versione di default. La riga `Accept-language: fr`: rappresenta solo una delle molte intestazioni di negoziazione dei contenuti disponibili in HTTP.

A questo punto concentriamoci sul formato generale di un messaggio di richiesta (Figura 2.8) che notiamo seguendo da vicino l'esempio precedente. Potreste aver osservato, tuttavia, che dopo le linee di intestazione (e i caratteri di ritorno a capo e di nuova linea) si trova un "corpo" (*entity body*). Quest'ultimo è vuoto nel caso del metodo GET, ma viene utilizzato dal metodo POST. Un client HTTP usa in genere il metodo POST quando l'utente riempie un form: per esempio, quando un utente fornisce le voci da trovare a un motore di ricerca. Nel caso di messaggio POST, l'utente sta ancora richiedendo una pagina web al server, ma i contenuti specifici della pagina dipendono da ciò che l'utente ha immesso nei campi del form. Se il valore del campo metodo è POST, allora il corpo contiene ciò che l'utente ha immesso nei campi del form.

Saremmo imprecisi se non menzionassimo che le richieste generate con il form non devono necessariamente usare il metodo POST. Anzi, i form HTML

**Figura 2.8** Formato generale dei messaggi di richiesta di HTTP.



usano spesso il metodo GET e includono i dati immessi (nei campi del form) nell'URL richiesto. Per esempio, se un form impiega il metodo GET e presenta due campi e se i dati immessi sono scimmie e banane, allora l'URL ha la struttura:

```
www.somesite.com/animalsearch?scimmie&banane
```

Nella navigazione quotidiana nel Web, avrete probabilmente notato URL estesi di questo tipo.

Il metodo HEAD è simile a GET. Quando un server riceve una richiesta con il metodo HEAD, risponde con un messaggio HTTP, ma tralascia gli oggetti richiesti. Gli sviluppatori di applicazioni usano spesso il metodo HEAD per verificare la correttezza del codice prodotto. Il metodo PUT, frequentemente usato assieme agli strumenti di pubblicazione sul Web, consente agli utenti di inviare un oggetto a un percorso specifico (*directory*) su uno specifico web server. Il metodo PUT viene anche utilizzato dalle applicazioni che richiedono di inviare oggetti ai web server. Il metodo DELETE consente invece la cancellazione di un oggetto su un server.<sup>5</sup>

### Messaggio di risposta HTTP

Presentiamo ora un tipico messaggio di risposta HTTP che potrebbe rappresentare la risposta al messaggio di richiesta dell'esempio precedente.

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html
(data data data data data ...)
```

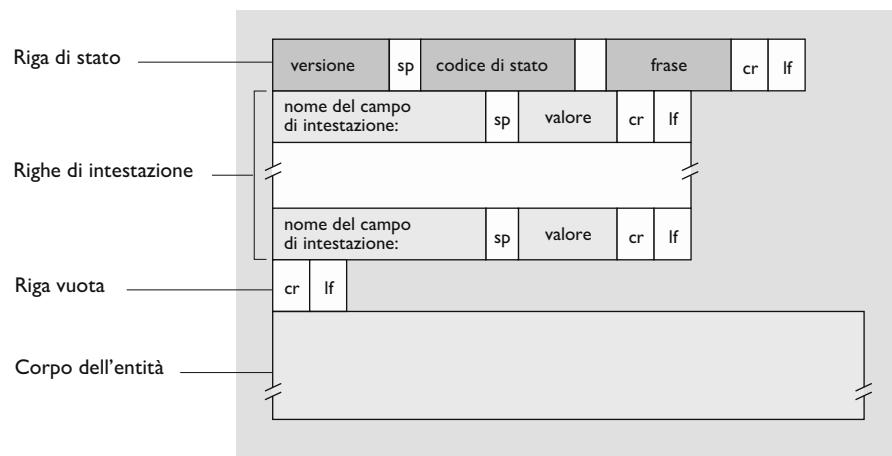
Analizzando in dettaglio questo messaggio di risposta, osserviamo tre sezioni: una **riga di stato** (*status line*) iniziale, sei **righe di intestazione** (*header lines*) e il **corpo** (*entity body*). Quest'ultimo è il fulcro del messaggio: contiene l'oggetto richiesto (rappresentato da: data data data data data ...). La riga di stato presenta tre campi: la versione del protocollo, un codice di stato e un corrispettivo messaggio di stato. In questo esempio, la riga di stato indica che il server sta usando HTTP/1.1 e che tutto va bene (OK), ossia che il server ha trovato e sta inviando l'oggetto richiesto.

Osserviamo ora le righe di intestazione. Il server utilizza la riga di intestazione **Connection: close** per comunicare al client che ha intenzione di chiudere la connessione TCP dopo l'invio del messaggio. La riga **Date:** indica l'ora e la data di creazione e invio, da parte del server, della risposta HTTP. Si noti

---

<sup>5</sup> Per motivi di sicurezza i metodi PUT e DELETE sono spesso disabilitati nei web server e si preferisce surrogarli con il metodo POST in cui si specifica nei vari campi che cosa aggiungere o cancellare dal server; in questo modo l'applicazione è in grado di fare dei controlli aggiuntivi (N.d.R.).

**Figura 2.9** Formato generale dei messaggi di risposta di HTTP.



che non si tratta dell’istante in cui l’oggetto è stato creato o modificato per l’ultima volta, ma del momento in cui il server recupera l’oggetto dal proprio file system, lo inserisce nel messaggio di risposta e invia il messaggio. La riga **Server**: indica che il messaggio è stato generato da un web server Apache; essa è analoga alla riga **User-agent**: nel messaggio di richiesta HTTP. La riga **Last-Modified**: indica l’istante e la data il cui l’oggetto è stato creato o modificato per l’ultima volta. Tale riga (che tratteremo presto più in dettaglio) è importante per la gestione dell’oggetto nelle cache, sia nel client locale sia in alcuni server in rete (*proxy server* o *proxy*). La riga di intestazione **Content-Length**: contiene il numero di byte dell’oggetto inviato. La riga **Content-Type**: indica che l’oggetto nel corpo è testo HTML. Il tipo dell’oggetto viene ufficialmente identificato tramite l’intestazione **Content-Type**: e non tramite l’estensione del file.

Dopo l’esempio esaminiamo il formato generale di un messaggio di risposta (Figura 2.9) che rispecchia il precedente esempio. Spendiamo qualche parola aggiuntiva sui codici di stato e sulle loro espressioni. Il codice di stato e l’espressione associata indicano il risultato della richiesta. Tra i più comuni codici di stato e relative espressioni troviamo:

- 200 OK: la richiesta ha avuto successo e in risposta si invia l’informazione.
- 301 Moved Permanently: l’oggetto richiesto è stato trasferito in modo permanente; il nuovo URL è specificato nell’intestazione **Location**: del messaggio di risposta. Il client recupererà automaticamente il nuovo URL.
- 400 Bad Request: si tratta di un codice di errore generico che indica che la richiesta non è stata compresa dal server.
- 404 Not Found: il documento richiesto non esiste sul server.
- 505 HTTP Version Not Supported: il server non dispone della versione di protocollo HTTP richiesta.

Vi piacerebbe vedere un reale messaggio di risposta HTTP? La cosa è altamente consigliata e molto semplice da realizzare. Per prima cosa collegatevi via Telnet al vostro web server preferito. Poi immettete un messaggio di richiesta

di una riga per un oggetto ospitato sul server. Per esempio, se avete accesso a un prompt dei comandi, scrivete:

```
telnet gaia.cs.umass.edu 80
GET /kurose_ross/interactive/index.php HTTP/1.1
Host: gaia.cs.umass.edu
```

Dopo aver immesso l'ultima riga, premete Invio due volte. Telnet apre una connessione TCP alla porta 80 dell'host `gaia.cs.umass.edu` e quindi invia il messaggio di richiesta HTTP. Dovreste vedere un messaggio di risposta che include il file HTML base dei problemi interattivi del libro. Se volete vedere solo le righe del messaggio HTTP e non ricevere l'oggetto stesso, sostituite GET con HEAD.

In questo paragrafo abbiamo trattato varie righe di intestazione utilizzabili nei messaggi di richiesta e di risposta HTTP. Le specifiche HTTP ne definiscono moltissime altre che possono essere inserite da browser, web server e proxy. Ne vedremo alcune più avanti, quando discuteremo i proxy in rete nel Paragrafo 2.2.5. Una trattazione del protocollo HTTP esaustiva e di agevole lettura, comprendente intestazioni e codici di stato, è quella di [Krishnamurty 2001].

Su che base un browser decide quali righe di intestazione includere in un messaggio di richiesta? Con che criterio un server decide quali includere in un messaggio di risposta? Un browser genera righe di intestazione a seconda del tipo e della versione, della configurazione da parte dell'utente (per esempio, la lingua preferita) e a seconda del fatto che possieda in cache una versione dell'oggetto, magari scaduta. I web server si comportano in modo simile: esistono diversi prodotti, versioni e configurazioni che influenzano le righe dell'intestazione dei messaggi di risposta.

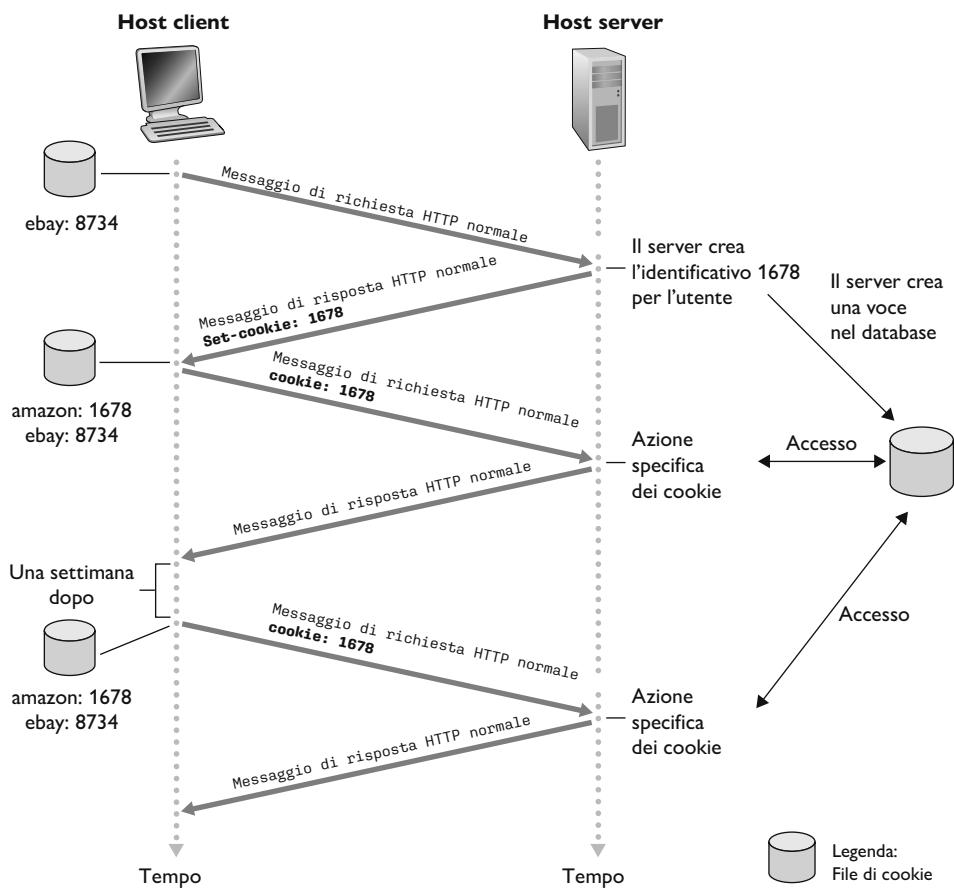
#### 2.2.4 Interazione utente-server: i cookie

Abbiamo precedentemente visto che i server HTTP sono privi di stato. Ciò semplifica la progettazione e consente di sviluppare web server ad alte prestazioni, in grado di gestire migliaia di connessioni TCP simultanee. Tuttavia, è spesso auspicabile che i web server possano autenticare gli utenti, sia per limitare l'accesso da parte di questi ultimi sia per fornire contenuti in funzione della loro identità. A questo scopo, HTTP adotta i **cookie**. I cookie, definiti in [RFC 6265], consentono ai server di tener traccia degli utenti. La maggior parte dei siti commerciali usa i cookie.

Come mostrato nella Figura 2.10, la tecnologia dei cookie presenta quattro componenti: (1) una riga di intestazione nel messaggio di risposta HTTP, (2) una riga di intestazione nel messaggio di richiesta HTTP, (3) un file mantenuto sul sistema dell'utente e gestito dal browser e (4) un database sul sito. Usando la Figura 2.10 vediamo un esempio di come funzionano i cookie. Supponiamo che Susan, che accede sempre al Web con Internet Explorer dal proprio PC di casa, contatti per la prima volta il sito di Amazon.com. Supponiamo inoltre che in passato abbia già visitato il sito di eBay. Quando giunge la richiesta al web server di Amazon, il sito crea un identificativo unico e una voce nel proprio database, indicizzata dal numero identificativo. A questo punto il server risponde al browser di Susan, includendo nella risposta HTTP l'intestazione Set-cookie:

**Figura 2.10**

Memorizzazione dello stato dell'utente con i cookie.



che contiene il numero identificativo. Per esempio, la riga di intestazione potrebbe essere:

**Set-cookie: 1678**

Quando il browser di Susan riceve il messaggio di risposta HTTP, vede l'intestazione **Set-cookie:**. Il browser allora aggiunge una riga al file dei cookie che gestisce. Questa riga include il nome dell'host del server e il numero identificativo nell'intestazione **Set-cookie:**. Si noti che il file di cookie contiene già una voce per eBay, dato che Susan ha già visitato quel sito in passato. Mentre Susan continua a navigare nel sito di Amazon, ogni volta che richiede una pagina web, il suo browser consulta il suo file dei cookie, estrae il suo numero identificativo per il sito e pone nella richiesta HTTP una riga di intestazione del cookie che include tale numero. Più nello specifico, ciascuna delle sue richieste HTTP al server di Amazon include la riga di intestazione:

**Cookie: 1678**

In tal modo è possibile monitorare l'attività di Susan nel sito. Sebbene esso non ne conosca necessariamente il nome, sa esattamente quali pagine sono state visitate dall'utente 1678, in quale ordine e a quali orari! Amazon usa i cookie per

fornire il suo servizio di carrello della spesa virtuale: durante una particolare sessione, il sito può mantenere una lista di tutti gli acquisti di Susan, di modo che l'utente possa effettuare tutti gli acquisti assieme alla fine della sessione.

Se Susan torna nel sito, magari una settimana più tardi, il suo browser continuerà a inserire la riga di intestazione **Cookie**: 1678 nei messaggi di richiesta. Amazon può suggerire a Susan dei prodotti sulla base delle pagine web che ha visitato in passato. Se poi Susan si registra sul sito, fornendo il suo nome completo, l'indirizzo di posta elettronica, un recapito postale e informazioni sulla carta di credito, Amazon può includere queste informazioni nel proprio database e quindi associare il nome di Susan al suo numero identificativo (e a tutte le pagine precedentemente visitate in quel sito). Ecco come Amazon e altri siti di commercio elettronico forniscono il cosiddetto “one-click shopping”: quando Susan sceglie di comprare un articolo durante una successiva visita, non le viene richiesto di immettere nuovamente il proprio nome, numero di carta di credito o indirizzo.

Da quanto precedentemente detto si evince che i cookie possono essere usati per identificare gli utenti. La prima volta che visita un sito, un utente può fornire un'identificazione (magari il suo nome). Successivamente il browser passa un'intestazione di cookie al server durante tutte le successive visite al sito, identificando quindi l'utente sul server. I cookie possono anche essere usati per creare un livello di sessione utente al di sopra di HTTP che è privo di stato. Per esempio, quando un utente si identifica in un'applicazione di posta elettronica basata su Web, il browser invia le informazioni del cookie al server, permettendo a quest'ultimo di identificare l'utente attraverso la sessione utente dell'applicazione.

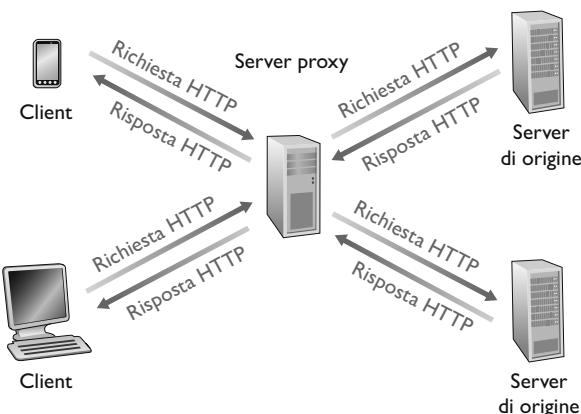
Nonostante i cookie semplifichino lo shopping via Internet, sono fonte di controversie, in quanto possono essere considerati una violazione della privacy dell'utente. Usando una combinazione di cookie e di informazioni fornite dall'utente, un sito web può imparare molto sull'utente e potrebbe vendere quanto sa a una terza parte.

## 2.2.5 Web caching

Una **web cache**, nota anche come **proxy server**, è un'entità di rete che soddisfa richieste HTTP al posto del web server effettivo. Il proxy ha una propria memoria su disco (una cache) in cui conserva copie di oggetti recentemente richiesti. Come illustrato nella Figura 2.11, il browser di un utente può essere configurato in modo che tutte le richieste HTTP dell'utente vengano innanzitutto dirette al proxy server. Una volta configurato il browser, ogni richiesta di oggetto da parte del browser viene inizialmente diretta al proxy [RFC 7234]. Supponiamo per esempio che un browser stia richiedendo l'oggetto <http://www.someschool.edu/campus.gif>. Ecco che cosa succede.

1. Il browser stabilisce una connessione TCP con il proxy server e invia una richiesta HTTP per l'oggetto specificato.
2. Il proxy controlla la presenza di una copia dell'oggetto memorizzata localmente. Se l'oggetto viene rilevato, il proxy lo inoltra all'interno di un messaggio di risposta HTTP al browser.

**Figura 2.11** Client che richiedono oggetti attraverso un server proxy.



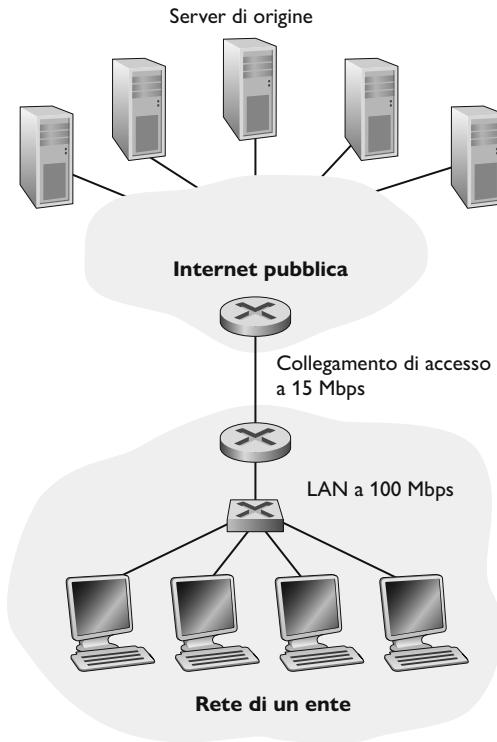
3. Se, invece, la cache non dispone dell'oggetto, il proxy apre una connessione TCP verso il server di origine; ossia, nel nostro esempio, [www.someschool.edu](http://www.someschool.edu). Quindi, il proxy invia al server una richiesta HTTP per l'oggetto. Una volta ricevuta tale richiesta, il server di origine invia al proxy l'oggetto all'interno di una risposta HTTP.
4. Quando il proxy riceve l'oggetto, ne salva una copia nella propria memoria locale e ne inoltra un'altra copia, all'interno di un messaggio di risposta HTTP, al browser (sulla connessione TCP esistente tra il browser e il proxy).

Si noti che il proxy è contemporaneamente server e client: quando riceve richieste da un browser e gli invia risposte agisce da server, quando invia richieste e riceve risposte da un server di origine funziona da client.

Generalmente un proxy server è acquistato e installato da un ISP. Per esempio un'università può installare un proxy sulla propria rete e configurare tutti i browser della propria sede per puntare a quel proxy. Oppure un grande ISP (quale ComCast) potrebbe installare uno o più proxy nella propria rete e pre-configurare i browser in modo che vi puntino.

Il web caching si è sviluppato in Internet per due ragioni. Innanzitutto, un proxy può ridurre in modo sostanziale i tempi di risposta alle richieste dei client, in particolare se l'ampiezza di banda che costituisce il collo di bottiglia tra il client e il server di origine è molto inferiore rispetto all'ampiezza di banda minima tra client e proxy. Se esiste una connessione ad alta velocità tra il client e il proxy, come spesso avviene, e se l'oggetto è nella cache, questa sarà in grado di consegnare rapidamente l'oggetto al client. In secondo luogo, come vedremo nel prossimo esempio, i proxy possono ridurre sostanzialmente il traffico sul collegamento di accesso a Internet, con il vantaggio di non dover aumentare l'ampiezza di banda frequentemente e ottenere quindi una riduzione dei costi. Inoltre i proxy possono ridurre in modo sostanziale il traffico globale del Web in Internet, migliorando di conseguenza le prestazioni di tutte le applicazioni.

Per comprendere meglio i benefici delle cache consideriamo un esempio nel contesto della Figura 2.12. La figura mostra due reti: la rete di un ente e la parte pubblica di Internet. La rete dell'ente è una LAN ad alta velocità. Un collegamento a 15 Mbps connette un router della prima rete a uno della seconda. I ser-



**Figura 2.12** Collo di bottiglia tra una LAN e Internet.

ver di origine sono collegati a Internet e situati in diverse parti del mondo. Supponiamo che la dimensione media di un oggetto sia 1 Mbit e che i browser dell'ente abbiano una frequenza media di 15 richieste ai server di origine al secondo. Ipotizziamo che i messaggi di richiesta HTTP siano trascurabilmente piccoli e non creino pertanto traffico nelle reti o nel collegamento di accesso (tra i due router). Supponiamo, inoltre, che la quantità di tempo che intercorre da quando il router sul lato Internet del collegamento di accesso della Figura 2.12 inoltra una richiesta HTTP (all'interno di un datagramma IP) a quando riceve la risposta (generalmente, all'interno di molti datagrammi IP) sia mediamente di due secondi. In modo informale ci riferiamo a questo ultimo ritardo come “ritardo Internet”.

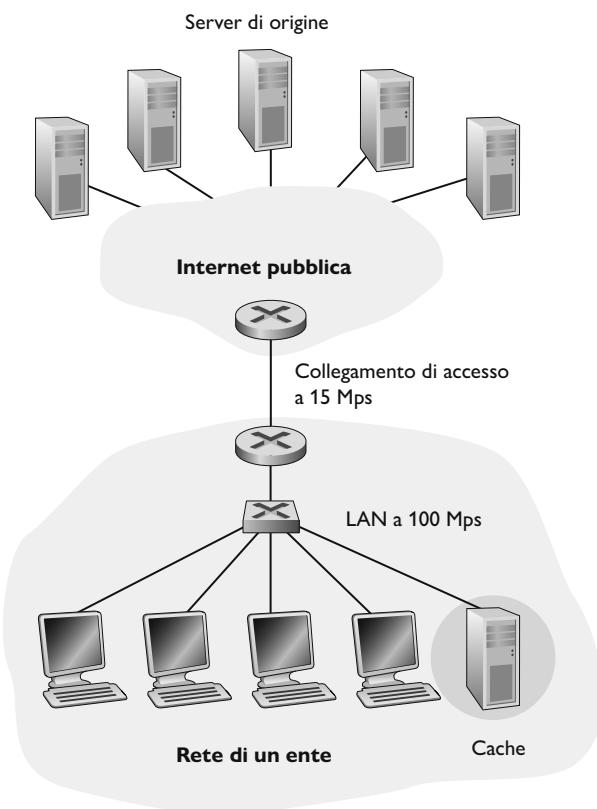
Il tempo totale di risposta, ossia il tempo che intercorre tra la richiesta da parte del browser di un oggetto fino alla corrispondente ricezione dell'oggetto, è la somma del ritardo sulla rete locale, del ritardo di accesso (ritardo tra i due router) e del ritardo Internet. Diamo ora una stima molto sommaria di tale ritardo. L'intensità di traffico sulla rete locale (si veda il Paragrafo 1.4.2) è pari a

$$(15 \text{ richieste/secondo}) \cdot (1 \text{ Mbit/richiesta}) / (100 \text{ Mbps}) = 0,15$$

mentre l'intensità di traffico sul collegamento di accesso (dal router Internet al router dell'ente) vale

$$(15 \text{ richieste/secondo}) \cdot (1 \text{ Mbit/richiesta}) / (15 \text{ Mbps}) = 1$$

**Figura 2.13** Aggiunta di una cache a una rete locale.



Un'intensità di traffico di 0,15 su una rete locale provoca generalmente alcune decine di millisecondi di ritardo che può, quindi, essere trascurato. Tuttavia, come detto nel Paragrafo 1.4.2, quando l'intensità di traffico si avvicina a 1 (come nel caso del collegamento di accesso della Figura 2.12), il ritardo su un collegamento diventa notevole e cresce senza limiti. Pertanto il tempo di risposta medio per soddisfare le richieste diventa dell'ordine dei minuti, se non superiore, il che è inaccettabile per gli utenti. Chiaramente occorre fare qualcosa.

Una possibilità consiste nell'incremento della banda sul collegamento di accesso a Internet, per esempio da 15 Mbps a 100 Mbps. Ciò abbasserà l'intensità di traffico sul collegamento di accesso fino a 0,15, il che si traduce in ritardi trascurabili tra i due router. In questo caso, il tempo totale di risposta sarà di circa due secondi, ossia il ritardo Internet. Ma questa soluzione implica che l'ente debba aggiornare il proprio collegamento a Internet, il che può risultare costoso.

Una soluzione alternativa (Figura 2.13) consiste nell'adozione di un proxy nella rete dell'istituzione. Le percentuali di successo (o *hit rate*), cioè la frazione di richieste soddisfatte dalla cache, tipicamente variano tra 0,2 e 0,7 nella pratica. A scopo esemplificativo, supponiamo che per la nostra istituzione l'*hit rate* sia 0,4. Dato che i client e il proxy sono collegati alla stessa rete locale ad alta velocità, il 40% delle richieste verrà soddisfatto dalla cache quasi immediatamente, ossia entro 10 millisecondi. Ciò nondimeno, il restante 60% delle richieste deve ancora essere soddisfatto dai server di origine. Ma ora solo il 60% degli

oggetti richiesti passa attraverso il collegamento di accesso, e l'intensità di traffico sul collegamento di accesso si riduce da 1,0 a 0,6. In generale, un'intensità di traffico inferiore a 0,8 su un collegamento a 15 Mbps corrisponde a un piccolo ritardo, dell'ordine delle decine di millisecondi. Questo ritardo è trascurabile rispetto ai due secondi di ritardo Internet. Sulla base di queste considerazioni, il ritardo medio è pertanto

$$0,4 \cdot (0,01 \text{ secondi}) + 0,6 \cdot (2,01 \text{ secondi})$$

che è solo leggermente superiore a 1,2 secondi. Questa seconda soluzione fornisce un tempo di risposta perfino inferiore rispetto alla prima e non richiede l'aggiornamento del collegamento dell'istituzione a Internet. Questa deve, ovviamente, acquistare e installare un proxy, una spesa comunque contenuta: molti proxy sono software di pubblico dominio che possono essere eseguiti su PC economici.

Con l'aumento dell'utilizzo delle *Content Distribution Network* (CDN) i proxy server giocano un ruolo sempre più importante in Internet. Un'azienda di CDN installa molte cache distribuite geograficamente, localizzando il traffico. Ci sono CDN condivise (come Akamai e Limelight) e CDN dedicate (come Google e Netflix). Ne discuteremo nel Paragrafo 2.6.

### **GET condizionale**

Sebbene il web caching riduca i tempi di risposta percepiti dall'utente, introduce un nuovo problema: la copia di un oggetto che risiede in cache potrebbe essere scaduta. In altre parole, l'oggetto ospitato nel web server potrebbe esser stato modificato rispetto alla copia nel client (sia esso un proxy o un browser). Fortunatamente, HTTP presenta un meccanismo che permette alla cache di verificare se i suoi oggetti sono aggiornati. Questo meccanismo è chiamato **GET condizionale** (*conditional GET*) [RFC 7232]. Un messaggio di richiesta HTTP viene detto messaggio di GET condizionale se (1) usa il metodo GET e (2) include una riga di intestazione *If-modified-since*:

Per mostrare il funzionamento del GET condizionale, consideriamo un esempio. Per prima cosa un proxy invia un messaggio di richiesta a un web server per conto del browser richiedente:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
```

Poi, il web server invia al proxy un messaggio di risposta con l'oggetto richiesto:

```
HTTP/1.1 200 OK
Date: Sat, 3 Oct 2015 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Wed, 9 Sep 2015 09:23:24
Content-Type: image/gif
(data data data data ...)
```

Il proxy inoltra l'oggetto al browser richiedente e pone anche l'oggetto nella cache locale. Va sottolineato che la cache memorizza con l'oggetto anche la data di ultima modifica. Poi, una settimana più tardi, un altro browser richiede lo stesso oggetto attraverso il proxy, e l'oggetto si trova ancora nella cache. Dato

che tale oggetto può essere stato modificato nel web server durante la settimana trascorsa, il proxy effettua un controllo di aggiornamento inviando un GET condizionale. Più nello specifico invia:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 9 Sep 2015 09:23:24
```

Si osservi che il valore della riga di intestazione `If-modified-since`: equivale esattamente al valore della riga di intestazione `Last-Modified`: inviata dal server una settimana prima. Questo GET condizionale sta comunicando al server di inviare l'oggetto solo se è stato modificato rispetto alla data specificata. Supponiamo che l'oggetto non sia stato modificato dalle 9:23:24 del 9 settembre 2015. Allora il web server invia un messaggio di risposta al proxy:

```
HTTP/1.1 304 Not Modified
Date: Sat, 10 Oct 2015 15:39:29
Server: Apache/1.3.0 (Unix)
(corpo vuoto)
```

Notiamo che in risposta a un GET condizionale, il web server invia ancora un messaggio di risposta, ma non include l'oggetto richiesto, in quanto ciò implicherebbe solo spreco di banda e incrementerebbe il tempo di risposta percepito dall'utente, in particolare se l'oggetto è grande. La riga di stato `304 Not Modified` comunica al proxy che può procedere e inoltrare al browser richiedente la copia dell'oggetto presente in cache.

## 2.2.6 HTTP/2

HTTP/2 [RFC 7540], standardizzato nel 2015, è stata la prima nuova versione di HTTP da HTTP/1.1, che fu standardizzato nel 1997. Fin dalla standardizzazione HTTP/2 è decollato, con oltre il 40% dei 10 milioni di grandi siti web che supportano nel 2020 HTTP/2 [W3Techs]. La maggior parte dei browser, inclusi Google Chrome, Internet Explorer, Safari, Opera e Firefox, supporta HTTP/2.

Gli obiettivi principali di HTTP/2 sono quello di ridurre la latenza percepita attivando il multiplexing di richiesta e risposta su una singola connessione TCP, di fornire supporto per la priorità delle richieste e il server push e di fornire una compressione efficiente dei campi di intestazione HTTP. HTTP/2 non modifica metodi, codici di stato, URL e campi di intestazione di HTTP, ma cambia il modo in cui i dati vengono formattati e trasportati tra il client e il server.

Per capire la necessità di introdurre HTTP/2 si ricordi che HTTP/1.1 utilizza connessioni TCP persistenti, consentendo l'invio di una pagina web dal server al client su un'unica connessione TCP. Avendo solo una connessione TCP per pagina web, il numero di socket sul server è limitato e ogni pagina web trasportata ottiene una equa condivisione della larghezza di banda di rete (come discusso di seguito). Tuttavia gli sviluppatori di browser web hanno rapidamente scoperto che l'invio di tutti gli oggetti in una pagina web su una singola connessione TCP ha il problema dell'**Head of Line (HOL) blocking** (“blocco in testa alla coda”). Per comprendere il blocco HOL, si consideri una pagina web che include una pagina di base HTML, un grande video clip vicino alla parte su-

riore della pagina web e molti piccoli oggetti sotto il video. Supponiamo inoltre che ci sia un collegamento a collo di bottiglia con velocità medio-bassa (per esempio, un collegamento wireless a bassa velocità) sul percorso tra server e client. Utilizzando una singola connessione TCP, il video clip impiegherebbe molto tempo per passare attraverso il collegamento a collo di bottiglia e gli oggetti piccoli subirebbero un ritardo, in quanto attendono dietro a esso; ovvero il videoclip in testa alla coda blocca i piccoli oggetti dietro di esso. I browser HTTP/1.1 in genere aggirano questo problema aprendo più connessioni TCP parallele, avendo così oggetti della stessa pagina web inviati parallelamente al browser. In questo modo gli oggetti piccoli possono arrivare ed essere riprodotti nel browser molto più velocemente, riducendo così il ritardo percepito dall'utente.

Il controllo di congestione TCP, discusso in dettaglio nel Capitolo 3, fornisce ai browser un incentivo non voluto a utilizzare più connessioni TCP parallele piuttosto che una singola connessione persistente. Il controllo di congestione TCP mira ad assegnare a ogni connessione TCP che condivide un collegamento a collo di bottiglia una quota uguale di larghezza di banda; quindi, se ci sono  $n$  connessioni TCP che operano su un link a collo di bottiglia, ogni connessione ottiene approssimativamente  $1/n$  della larghezza di banda. Aprendo più connessioni TCP parallele per trasportare una singola pagina web, il browser può "imbrogliare" e prendersi una parte maggiore della larghezza di banda del collegamento. Molti browser HTTP/1.1 aprono fino a sei connessioni TCP parallele, non solo per aggirare il blocco HOL, ma anche per ottenere più larghezza di banda.

Uno degli obiettivi principali di HTTP/2 è eliminare o almeno ridurre il numero di connessioni TCP parallele per il trasporto di una singola pagina web. Questo non solo riduce il numero di socket che deve essere aperto e mantenuto sui server, ma consente anche al controllo di congestione TCP di funzionare come previsto. Tuttavia con una sola connessione TCP per il trasporto di una pagina web, per evitare il blocco HOL HTTP/2 richiede meccanismi progettati con cura.

## Framing HTTP/2

La soluzione HTTP/2 del blocco HOL consiste nel suddividere ogni messaggio in piccoli frame e alternare i messaggi di richiesta e risposta sulla stessa connessione TCP. Si consideri nuovamente l'esempio di una pagina web costituita da un unico grande video clip e 8 oggetti più piccoli. Pertanto il server riceverà 9 richieste simultanee da qualsiasi browser che voglia vedere questa pagina web. Per ciascuna di queste richieste, il server deve inviare al browser 9 messaggi di risposta HTTP concorrenti.

Supponiamo che tutti i fotogrammi abbiano una lunghezza fissata, il video clip sia composto da 1000 frame e ognuno degli oggetti più piccoli sia costituito da due frame. Se si intervallano i frame, il primo frame di tutti gli oggetti più piccoli viene inoltrato dopo aver inviato un frame del video clip. Quindi, dopo aver inviato il secondo frame del video clip, viene inoltrato l'ultimo frame di ciascuno degli oggetti piccoli. Pertanto, tutti gli oggetti più piccoli vengono inviati dopo aver inoltrato un totale di 18 frame. Se i frame non venissero inter-

vallati, gli oggetti più piccoli verrebbero inviati solo dopo aver inoltrato 1016 frame. Quindi, il meccanismo di framing HTTP/2 può ridurre significativamente il ritardo percepito dall’utente.

La capacità di suddividere un messaggio HTTP in frame indipendenti, intervallarli e poi ricostruirli dall’altra parte è il principale miglioramento apportato da HTTP/2. Il processo di framing viene eseguito dal sottolivello di framing del protocollo HTTP/2. Quando un server vuole inviare una risposta HTTP, la risposta viene elaborata dal sottolivello di framing, dove viene scomposta in frame. Il campo di intestazione della risposta diventa un frame e il corpo del messaggio viene suddiviso in più frame aggiuntivi. I frame della risposta sono quindi intervallati dal sottolivello di framing nel server con i frame di altre risposte e inviati tramite singola connessione TCP persistente. I frame, man mano che arrivano al client, vengono prima ricostruiti nei messaggi di risposta originali nel sottolivello di framing e quindi elaborati dal browser come al solito. allo stesso modo, le richieste HTTP del client vengono suddivise in frame e intervallate.

Oltre a suddividere ogni messaggio HTTP in frame indipendenti, il sottolivello di framing esegue anche una codifica binaria dei frame. I protocolli binari sono più efficienti da analizzare, portano a frame leggermente più piccoli e sono meno soggetti a errori.

### Priorità dei messaggi di risposta e server pushing

Il meccanismo di priorità dei messaggi consente agli sviluppatori di personalizzare la priorità relativa delle richieste per ottimizzare le prestazioni dell’applicazione. Come abbiamo appena appreso, il sottolivello di framing organizza i messaggi in stream paralleli di dati destinati allo stesso richiedente. Un client, quando invia richieste simultanee a un server, può dare priorità alle risposte che richiede assegnando un peso compreso tra 1 e 256 a ciascun messaggio, dove il numero più alto indica una priorità più alta. Usando questi pesi, il server può inviare prima i frame per le risposte con la priorità più alta. Inoltre, il client indica anche la dipendenza di ogni messaggio da altri messaggi specificando l’ID del messaggio da cui dipende.

Un’altra caratteristica di HTTP/2 è la capacità di un server di inviare più risposte per una singola richiesta del cliente. Cioè, oltre alla risposta alla richiesta originale, il server può effettuare un invio *push* al client di oggetti aggiuntivi, senza che il client debba richiedere ognuno di essi. Ciò è possibile poiché la pagina di base HTML indica gli oggetti necessari per eseguire il rendering completo della pagina web. Quindi, invece di aspettare le richieste HTTP per questi oggetti, il server può analizzare la pagina HTML, identificare gli oggetti necessari e inviarli al client *prima di ricevere le richieste esplicite per essi*. Il server push elimina la latenza aggiuntiva dovuta all’attesa per le richieste.

## HTTP/3

QUIC, che verrà discusso nel Capitolo 3, è un nuovo protocollo di “trasporto” implementato nel livello di applicazione sul protocollo UDP. QUIC offre diverse opportune funzionalità per HTTP, come il multiplexing dei messaggi (*interleaving*), il controllo di flusso per ogni stream e la creazione di connessioni a bassa latenza. HTTP/3 è un altro, nuovo protocollo HTTP progettato per fun-

zionare su QUIC. A partire dal 2020, HTTP/3 viene descritto nei draft Internet e non è stato ancora completamente standardizzato. Molte delle funzionalità HTTP/2 (come l'interleaving dei messaggi) sono incluse in QUIC, facilitando la progettazione di HTTP/3.

## 2.3 Posta elettronica in Internet

La posta elettronica è presente fin dagli albori di Internet e ne è stata l'applicazione più diffusa durante i suoi primi anni [Segaller 1998]; con il trascorrere del tempo è diventata sempre più elaborata e potente. Anche ora rappresenta una delle più importanti *killer application* di Internet.

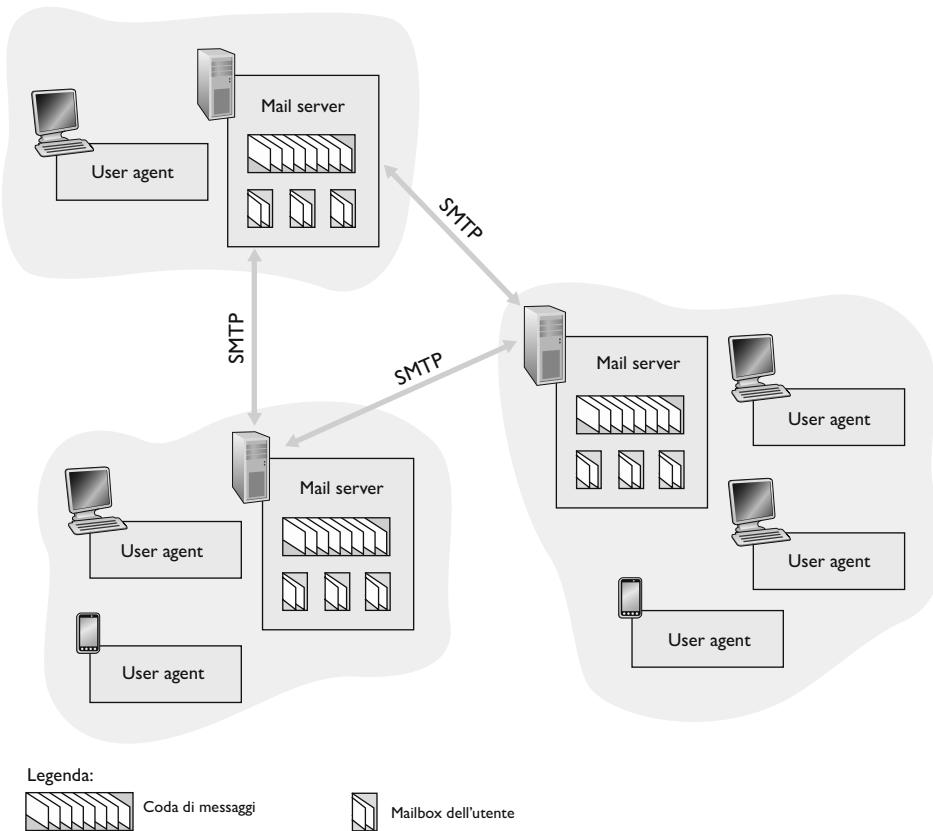
Come il servizio postale ordinario, l'e-mail rappresenta un mezzo di comunicazione asincrono: le persone inviano e leggono i messaggi nel momento per loro più opportuno, senza doversi coordinare con altri utenti. A differenza del servizio postale ordinario, però, la posta elettronica è veloce, facile da distribuire e gratuita. La moderna posta elettronica ha molte caratteristiche importanti quali gli allegati (*attachment*), i collegamenti ipertestuali, il testo con formattazione HTML e le foto incorporate.

In questo paragrafo esaminiamo i protocolli a livello di applicazione alla base della posta elettronica su Internet. Prima di addentrarci nella trattazione approfondita di tali protocolli ci soffermiamo su una panoramica dei sistemi di posta e i loro componenti chiave.

La Figura 2.14 presenta una visione ad alto livello del sistema postale di Internet. Da questo diagramma notiamo la presenza di tre componenti principali: gli **user agent** (o *agenti utente*), i **mail server** (*server di posta*) e il **protocollo SMTP** (*Simple Mail Transfer Protocol*). Per descrivere questi componenti consideriamo l'esempio di un mittente, Alice, che invia un messaggio a un destinatario, Bob. Gli user agent (per esempio Microsoft Outlook, Apple Mail e Google Gmail) consentono agli utenti di leggere, rispondere, inoltrare, salvare e comporre i messaggi. Quando Alice ha finito di comporre il messaggio, il suo user agent lo invia al server di posta, dove viene posto nella coda di messaggi in uscita. Quando Bob vuole leggere il messaggio, il suo user agent lo recupera dalla casella di posta nel suo mail server.

I mail server costituiscono la parte centrale dell'infrastruttura del servizio di posta elettronica. Ciascun destinatario, come per esempio Bob, ha una **mailbox** (*casella di posta*) collocata in un mail server. La mailbox di Bob gestisce e contiene messaggi a lui inviati. Un tipico messaggio inizia il proprio viaggio dallo user agent, giunge al mail server del mittente e prosegue fino al mail server del destinatario, dove viene depositato nella sua casella. Bob, per accedere ai messaggi della propria casella, deve essere autenticato dal server che lo ospita tramite nome utente e password. Il mail server di Alice deve anche gestire eventuali problemi del server di Bob. Il server di Alice, se non può consegnare la posta a quello di Bob, la trattiene in una **coda di messaggi** (*message queue*) e cerca di trasferirla in un secondo momento. In genere i tentativi vengono effettuati ogni 30 minuti e il server, se dopo alcuni giorni non è riuscito a effettuare il trasferimento, rimuove il messaggio e notifica la mancata consegna al mittente (Alice) con un messaggio di posta elettronica.

**Figura 2.14** Visione ad alto livello del sistema di posta elettronica di Internet.



SMTP rappresenta il principale protocollo a livello di applicazione per la posta elettronica su Internet. Fa uso del servizio di trasferimento dati affidabile proprio di TCP per trasferire la mail dal server del mittente a quello del destinatario. Così come accade per la maggior parte dei protocolli a livello di applicazione, SMTP presenta un lato client, in esecuzione sul mail server del mittente e un lato server, in esecuzione sul server del destinatario. Entrambi i lati possono essere eseguiti su tutti i server di posta. Quando un server invia posta a un altro, agisce come client SMTP; quando invece la riceve, funziona come server SMTP.

### 2.3.1 SMTP

Questo protocollo, definito nell'RFC 5321, costituisce il cuore della posta elettronica su Internet. Come detto precedentemente, SMTP trasferisce i messaggi dal mail server del mittente a quello del destinatario. SMTP è assai più vecchio di HTTP (l'RFC originale su SMTP risale al 1982 e il protocollo era già utilizzato da tempo). Sebbene presenti peculiarità meravigliose, evidenziate dalla sua onnipresenza in Internet, SMTP rappresenta una tecnologia ereditata con caratteristiche “arcaiche”. Per esempio, tratta il corpo (non solo le intestazioni) di tutti i messaggi di posta come semplice ASCII a 7 bit. Questa restrizione aveva senso nei primi anni '80, quando la capacità trasmittiva era scarsa e nessuno inviava per posta elettronica grandi allegati quali immagini, audio o video, ma

oggi, in piena era multimediale, la restrizione all'ASCII a 7 bit è piuttosto penalizzante, in quanto richiede che i dati multimediali binari vengano codificati in ASCII prima di essere inviati e che il messaggio venga nuovamente decodificato in binario dopo il trasporto. Ricordiamo, dal Paragrafo 2.2, che HTTP non richiede la codifica ASCII dei dati multimediali prima del trasferimento.

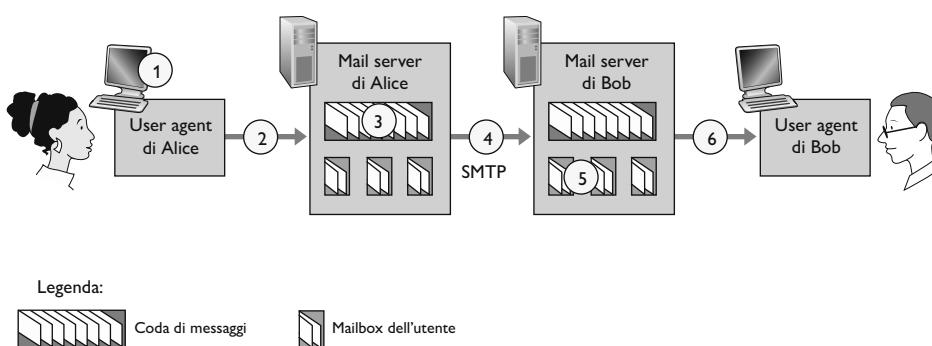
Al fine di illustrare le operazioni di base di SMTP presentiamo uno scenario tipico. Supponiamo che Alice voglia inviare a Bob un semplice messaggio ASCII.

1. Alice invoca il proprio user agent per la posta elettronica, fornisce l'indirizzo di posta di Bob (per esempio `bob@someschool.edu`), compone il messaggio e dà istruzione allo user agent di inviarlo.
2. Lo user agent di Alice invia il messaggio al suo mail server, dove è collocato in una coda di messaggi.
3. Il lato client di SMTP, eseguito sul server di Alice, vede il messaggio nella coda dei messaggi e apre una connessione TCP verso un server SMTP in esecuzione sul mail server di Bob.
4. Dopo un handshaking SMTP, il client SMTP invia il messaggio di Alice sulla connessione TCP.
5. Presso il mail server di Bob, il lato server di SMTP riceve il messaggio che viene posizionato nella casella di Bob.
6. Bob, quando lo ritiene opportuno, invoca il proprio user agent per leggere il messaggio.

Tale scenario viene riassunto nella Figura 2.15.

È importante osservare che di solito SMTP non usa mail server intermedi per inviare la posta, anche quando i mail server finali sono collocati agli angoli opposti del mondo. Se il server di Alice si trova a Hong Kong e quello di Bob a St. Louis, la connessione TCP ha luogo direttamente tra le due città. In particolare, se il mail server di Bob è spento, il messaggio rimane nel mail server di Alice e attende un nuovo tentativo. Il messaggio non viene posto in alcun mail server intermedio.

Osserviamo attentamente il trasferimento SMTP di un messaggio da un mail server a un altro. Scopriremo che il protocollo SMTP presenta molte somiglianze con i protocolli usati per l'interazione umana faccia a faccia. Primo, il client SMTP (in esecuzione sul mail server di invio) fa stabilire a TCP una connessione



**Figura 2.15** Alice invia un messaggio a Bob.

sulla porta 25 verso il server SMTP (in esecuzione sul mail server in ricezione). Se il server è inattivo, il client riprova più tardi. Una volta stabilita la connessione, il server e il client effettuano una qualche forma di handshaking a livello applicativo. Proprio come le persone che si presentano prima di scambiarsi informazioni, client e server SMTP si presentano prima di effettuare scambi di informazioni. Durante questa fase, il client indica l'indirizzo e-mail del mittente (la persona che ha generato il messaggio) e quello del destinatario. Dopo la reciproca presentazione, il client invia il messaggio. SMTP può contare sul servizio di trasferimento dati affidabile proprio di TCP per recapitare il messaggio senza errori. Il client ripete il processo sulla stessa connessione TCP se ha altri messaggi da inviare al server, altrimenti ordina a TCP di chiudere la connessione.

Diamo ora uno sguardo a un esempio di trascrizione di messaggi scambiati tra un client SMTP (C) e un server SMTP (S). Il nome dell'host del client è `crepes.fr` mentre il nome dell'host del server è `hamburger.edu`. Le righe di testo ASCII precedute da C: sono esattamente quelle che il client invia nella propria socket TCP, mentre le righe precedute da S: sono esattamente quelle che il server invia nella propria socket TCP. La seguente trascrizione inizia appena si stabilisce la connessione TCP:

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr ... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with “.” on a line by itself
C: Ti piace il ketchup?
C: Che cosa ne pensi dei cetrioli?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Nell'esempio sopra riportato, il client invia un messaggio (“Ti piace il ketchup? Che cosa ne pensi dei cetrioli?”) dal server di posta `crepes.fr` al server di posta `hamburger.edu`. Come parte del dialogo, il client ha inviato cinque comandi: HELO (abbreviazione di HELLO), MAIL FROM, RCPT TO, DATA e QUIT. Questi comandi sono abbastanza auto-esplicativi traducendoli dall'inglese (“ciao”, “mail da”, “recapitare a”, “dati”, “basta”). Il client invia anche una riga che consiste unicamente di un punto, che indica al server la fine del messaggio. SMTP termina ogni riga con i caratteri di ritorno a capo e nuova linea. Il server invia risposte a ogni comando, e ciascuna presenta un codice di risposta e qualche spiegazione (opzionale) in inglese. Ricordiamo che SMTP fa uso di connessioni persistenti: se il mail server di invio ha molti messaggi da inviare allo stesso mail server in ricezione, può mandarli tutti sulla stessa connessione TCP. Per ciascun messaggio il client inizia il processo con un nuovo MAIL FROM: `crepes.fr` e stabilisce la fine del messaggio con un punto isolato. Il comando QUIT viene inviato solo dopo aver spedito tutti i messaggi.

Per effettuare un dialogo diretto con un server SMTP è possibile usare Telnet.

Per farlo, usate il comando

```
telnet serverName 25
```

dove `serverName` è il nome di un mail server locale: ciò stabilisce una connessione TCP tra il vostro host locale e il mail server. Dopo aver digitato questa riga, dovreste ricevere immediatamente la risposta 220 da parte del server. Quindi, immettete i comandi SMTP HELO, MAIL FROM, RCPT TO, DATA, ". " e QUIT al momento opportuno. Vi invitiamo a svolgere l'Esercizio di programmazione 3 alla fine di questo capitolo, che vi porterà a costruire un'implementazione del lato client di SMTP. Potrete così inviare un messaggio di posta elettronica a qualunque destinatario tramite un mail server locale.

### 2.3.2 Formati dei messaggi di posta

Alice, scrivendo una lettera di posta ordinaria a Bob, potrebbe includere come intestazione tutte le informazioni accessorie in cima alla lettera, tra cui l'indirizzo di Bob, l'indirizzo del mittente e la data. Analogamente, il corpo dei messaggi di posta elettronica è preceduto da un'intestazione contenente informazioni di servizio. Tale informazione periferica è contenuta in una serie di righe di intestazione, definite nell'RFC 5322. Queste righe sono separate dal corpo del messaggio mediante una riga senza contenuto. L'RFC 5322 specifica il formato esatto delle righe di intestazione della posta e la loro interpretazione. Come avviene per HTTP, queste righe contengono testo leggibile, costituito da una parola chiave seguita da due punti a loro volta seguiti da un valore. Alcune parole chiave sono obbligatorie, mentre altre sono opzionali. Ogni intestazione deve avere una riga `From:` e una riga `To:`. Un'intestazione può includere una riga `Subject:` e altre righe di intestazione opzionali. È importante osservare che queste righe sono differenti dai comandi SMTP analizzati nel Paragrafo 2.3.1 (anche se contengono alcune parole comuni quali “*from*” e “*to*”). I comandi di tale paragrafo facevano parte del protocollo SMTP; le righe di intestazione esaminate qui sono invece parte del messaggio stesso.

Ecco una tipica intestazione di messaggio.

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Alla ricerca del significato della vita.
```

Dopo l'intestazione, segue una riga vuota; quindi troviamo il corpo del messaggio (in ASCII). Provate a usare telnet per inviare a un server di posta un messaggio che contiene alcune righe di intestazione tra cui `Subject:`. Per farlo, utilizzate `telnet serverName 25`, come indicato nel Paragrafo 2.3.1.

### 2.3.3 Protocolli di accesso alla posta

Quando SMTP consegna il messaggio di Alice al mail server destinatario, questo lo colloca nella casella di posta di Bob.

Dato che Bob (il destinatario) esegue il proprio user agent sul suo host locale, quale per esempio uno smartphone o un PC, verrebbe naturale pensare di collocarvi anche un mail server. Con questo approccio, il mail server di Alice dialogherebbe direttamente con il PC di Bob. Tuttavia esiste un problema col-

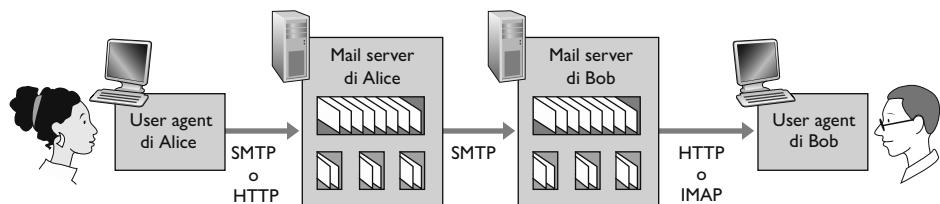
legato a questa soluzione. Ricordiamo che un server di posta gestisce caselle ed esegue il lato client e server di SMTP. Se il server di posta di Bob dovesse risiedere sul suo host locale, quest'ultimo dovrebbe rimanere sempre acceso e connesso a Internet al fine di ricevere nuova posta che può giungere in qualsiasi istante. Tale procedura sarebbe poco pratica per cui si preferisce che l'utente abbia in esecuzione uno user agent sul'host locale, ma acceda alla propria casella memorizzata su un mail server condiviso con altri utenti e sempre attivo.

Consideriamo ora il percorso che un messaggio di posta intraprende quando viene spedito da Alice a Bob. Abbiamo appena imparato che, in qualche punto del percorso, il messaggio deve essere depositato nel mail server di Bob. Questo si potrebbe fare semplicemente forzando lo user agent di Alice a spedire messaggi direttamente al mail server di Bob. Di solito però lo user agent del mittente non dialoga in modo diretto con il server del destinatario. Piuttosto, come mostrato nella Figura 2.16, lo user agent di Alice utilizza SMTP per spingere i messaggi di posta elettronica nel suo mail server, che adotta SMTP (come client SMTP) per comunicare il messaggio al mail server di Bob. Qual è il motivo di questa procedura in due fasi? Principalmente perché, se non utilizzasse il suo mail server come punto intermedio, lo user agent di Alice non saprebbe come gestire un mail server di destinazione non raggiungibile. Alice deve dapprima depositare la e-mail nel proprio mail server, che può ripetutamente tentare l'invio del messaggio al mail server di Bob, per esempio ogni trenta minuti, finché questo non diventa operativo.

Tuttavia nel nostro puzzle manca ancora un pezzo. Come fa un destinatario (quale Bob), che esegue uno user agent sul proprio PC locale, a ottenere i messaggi che si trovano nel mail server del suo provider? Osserviamo che lo user agent di Bob non può usare SMTP per ottenere tali messaggi dato che si tratta di un'operazione di *pull*, mentre SMTP è un protocollo *push*.

Attualmente vengono utilizzati due modi per permettere a Bob di recuperare la sua e-mail da un mail server. Se Bob utilizza la posta elettronica basata sul Web o una app per smartphone (come Gmail), lo user agent utilizzerà HTTP per recuperare la posta elettronica di Bob. Questa modalità richiede che il mail server di Bob abbia un'interfaccia HTTP e un'interfaccia SMTP (per comunicare con il mail server di Alice). Il metodo alternativo, tipicamente utilizzato con i client di posta come Microsoft Outlook, utilizza il protocollo **IMAP** (*Internet Mail Access Protocol*) definito nell'RFC 3501. Entrambi gli approcci, HTTP e IMAP, consentono a Bob di gestire le cartelle conservate nel server di posta di Bob. Bob può spostare i messaggi nelle cartelle che crea, eliminarli, contrassegnarli come importanti e così via.

**Figura 2.16** Entità comunicanti e protocolli per la posta elettronica.



## 2.4 DNS: il servizio di directory di Internet

Le persone possono essere identificate in molti modi. Per esempio, si può utilizzare il nome di battesimo, il codice fiscale o il numero della patente. A seconda dei contesti, ciascuno di questi identificatori può essere più appropriato di un altro. Per esempio, i calcolatori dell'IRS (*Internal Revenue Service*), il famigerato ministero delle finanze degli Stati Uniti, preferiscono usare codici a lunghezza fissa detti *Social Security Number*, piuttosto che il nome sul certificato di nascita. Al contrario, nei rapporti interpersonali si utilizza normalmente il nome di battesimo. “Ciao. Il mio nome è 132-67-9875. Ti presento mio marito, 178-87-1146.” Potreste immaginare un dialogo del genere?

Proprio come le persone, anche gli host Internet possono essere identificati in vari modi. Gli **hostname** quali `www.facebook.com`, `www.google.com` e `gaia.cs.umass.edu`, risultano abbastanza appropriati per le persone, ma non forniscano ben poca informazione sulla loro collocazione all'interno di Internet. Un nome quale `www.eurecom.fr`, che termina con il suffisso del paese `.fr`, ci dice che l'host si trova probabilmente in Francia, ma niente di più. Inoltre, dato che questi nomi sono costituiti da un numero variabile di caratteri alfanumerici, sarebbero difficilmente elaborabili dai server. Per tali motivi, gli host vengono identificati anche dai cosiddetti **indirizzi IP**.

Approfondiremo gli indirizzi IP nel Capitolo 4, ma è comunque utile spendere ora qualche parola sull'argomento. Un indirizzo IP consiste di quattro byte e presenta una rigida struttura gerarchica. Ha una forma del tipo `121.7.106.83`, in cui ogni punto separa uno dei byte espressi con un numero decimale compreso tra 0 e 255. Un indirizzo IP è gerarchico perché, leggendolo da sinistra a destra, otteniamo informazioni sempre più specifiche sulla collocazione dell'host in Internet (ossia sulla sua appartenenza a quale rete all'interno della rete di reti). In modo simile, quando analizziamo un indirizzo postale dal basso verso l'alto, otteniamo dettagli sempre più specifici sull'ubicazione del destinatario.

### 2.4.1 Servizi forniti da DNS

Abbiamo appena visto che esistono due modi per identificare gli host: il nome e l'indirizzo IP. Le persone preferiscono il primo, mentre i router prediligono gli indirizzi IP a lunghezza fissa e strutturati in modo gerarchico. Al fine di conciliare i due approcci è necessario un servizio in grado di tradurre gli hostname nei loro indirizzi IP. Si tratta del principale compito del *domain name system* (DNS) di Internet. DNS è (1) un database distribuito implementato in una gerarchia di **DNS server** e (2) un protocollo a livello di applicazione che consente agli host di interrogare il database. I DNS server sono generalmente macchine UNIX che eseguono un software chiamato BIND (*Berkeley Internet Name Domain*) [BIND 2020]. Il protocollo DNS utilizza UDP e la porta 53.

DNS viene comunemente utilizzato da altri protocolli a livello di applicazione, tra cui HTTP e SMTP, per tradurre i nomi di host forniti dall'utente in indirizzi IP. Per esempio, consideriamo che cosa succede quando un browser (ossia un client HTTP) in esecuzione sull'host di un utente richiede l'URL `www.someschool.edu/index.html`. L'host dell'utente, per essere in grado di

inviare un messaggio di richiesta HTTP al web server `www.someschool.edu`, deve come prima cosa ottenere il suo indirizzo IP. Ciò avviene come segue.

1. La stessa macchina utente esegue il lato client dell'applicazione DNS.<sup>6</sup>
2. Il browser estrae il nome dell'host, `www.someschool.edu`, dall'URL e lo passa al lato client dell'applicazione DNS.
3. Il client DNS invia una interrogazione (query) contenente l'hostname a un DNS server.
4. Il client DNS prima o poi riceve una risposta, che include l'indirizzo IP corrispondente all'hostname.
5. Una volta ricevuto l'indirizzo IP dal DNS, il browser può dare inizio a una connessione TCP verso il processo server HTTP collegato alla porta 80 di quell'indirizzo IP.

Da questo esempio vediamo che il DNS introduce un ritardo aggiuntivo, talvolta sostanziale, alle applicazioni Internet che lo utilizzano. Fortunatamente, come vedremo più avanti, l'indirizzo IP desiderato si trova spesso nella cache di un DNS server vicino, il che aiuta a ridurre il traffico DNS in rete e il ritardo medio del servizio.

Oltre alla traduzione degli hostname in indirizzi IP, DNS mette a disposizione altri importanti servizi.

- **Host aliasing.** Un host dal nome complicato può avere uno o più sinonimi (*alias*). Per esempio, `relay1.west-coast.enterprise.com` potrebbe avere, diciamo, due sinonimi quali `enterprise.com` e `www.enterprise.com`. In questo caso, si dice che il nome `relay1.west-coast.enterprise.com` è un **hostname canonico**. I sinonimi, se presenti, sono generalmente più facili da ricordare rispetto ai nomi canonici. Il DNS può essere invocato da un'applicazione per ottenere l'hostname canonico di un sinonimo, così come l'indirizzo IP dell'host.
- **Mail server aliasing.** Per ovvi motivi è fortemente auspicabile che gli indirizzi di posta elettronica siano facili da ricordare. Per esempio, se Bob ha un account Hotmail, il suo indirizzo di posta elettronica potrebbe essere semplicemente `bob@yahoo.com`. Tuttavia, l'hostname del server di posta Hotmail è molto più complicato e assai meno facile da ricordare rispetto a `yahoo.com`. Per esempio, il nome canonico potrebbe assomigliare a `relay1.west-coast.yahoo.com`. Un'applicazione di posta può invocare il DNS per ottenere il nome canonico di un sinonimo fornito, così come l'indirizzo IP dell'host. Infatti, il record MX (si veda più avanti) permette al server di posta di una società e al web server di avere hostname (*alias*) identici. Per esempio, il web server di un'azienda e il suo mail server possono essere entrambi chiamati `enterprise.com`.<sup>7</sup>

---

<sup>6</sup> Spesso, la sua implementazione viene chiamata “resolver”(N.d.R.).

<sup>7</sup> Detto in altri termini, con il record MX è possibile associare un alias al mail server responsabile di una certa organizzazione; questo alias può, visto che è ben chiaro il servizio fornito, coincidere con l'alias dato a un host con un altro nome canonico (N.d.R.).

- **Distribuzione del carico di rete (*load distribution*)**. Il DNS viene anche utilizzato per distribuire il carico tra server replicati, per esempio dei web server. I siti con molto traffico, quali [cnn.com](#), vengono replicati su più server, ognuno eseguito su un host diverso con un indirizzo IP differente. Nel caso di web server replicati, va dunque associato a ogni hostname canonico un insieme di indirizzi IP. Il database DNS contiene questo insieme di indirizzi. Quando i client effettuano una query DNS per un nome associato a un insieme di indirizzi, il server risponde con l'intero insieme di indirizzi, ma ne varia l'ordinamento a ogni risposta. Dato che generalmente un client invia il suo messaggio di richiesta HTTP al primo indirizzo IP elencato nell'insieme, la rotazione DNS distribuisce il traffico sui server replicati. La rotazione viene anche utilizzata affinché più server di posta possano condividere lo stesso nome. Inoltre, le società di distribuzione di contenuti quali Akamai hanno fatto uso del DNS in modi più sofisticati [Dilley 2002] per effettuare la diffusione di contenuti web (si veda il Paragrafo 2.6.3).

DNS è specificato nelle RFC 1034 e 1035 ed è stato aggiornato nelle successive RFC aggiuntive. Si tratta di un sistema complesso; in questa sede ci occuperemo solo degli aspetti chiave del suo funzionamento. Il lettore può far riferimento alle RFC citate e al libro di Abitz e Liu [Abitz 1993]; si veda inoltre l'articolo retrospettivo [Mockapetris 1988] che fornisce un'interessante descrizione di DNS e dei suoi scopi e anche [Mockapetris 2005].

### 2.4.2 Panoramica del funzionamento di DNS

Presentiamo ora una panoramica ad alto livello del funzionamento del DNS. La nostra trattazione si concentrerà sul servizio di traduzione da hostname a indirizzo IP.

Supponiamo che una certa applicazione (quale per esempio un browser web o un programma per la lettura della posta) in esecuzione sull'host di un utente abbia necessità di tradurre un hostname in un indirizzo IP. L'applicazione invocherà il lato client del DNS, specificando l'hostname da tradurre. Su molte macchine basate su UNIX, `gethostbyname()` è la chiamata di funzione effet-

**BOX 2.2**

#### TEORIA E PRATICA

##### DNS: funzioni critiche per la rete attraverso il paradigma client-server

Come HTTP, FTP e SMTP, anche DNS è un protocollo a livello di applicazione dato che (1) viene usato tra sistemi periferici che comunicano tra loro adottando il paradigma client-server e (2) si affida a un protocollo sottostante di trasporto end-to-end per trasferire i messaggi. Da un'altra prospettiva, tuttavia, il ruolo del DNS differisce dalle applicazioni di trasferimento file via Web e dalla posta elettronica in quanto non interagisce direttamente con gli utenti. In effetti, il DNS fornisce una funzione vitale per Internet, cioè la traduzione dei nomi degli host nei loro indirizzi IP associati per le applicazioni utente e altri software in Internet. Abbiamo notato nel Paragrafo 1.2 che la maggior parte della complessità nell'architettura di Internet si trova "sul bordo" della rete. Il DNS, che implementa il processo critico di traduzione da nome a indirizzo utilizzando client e server collocati ai margini della rete, è un ulteriore esempio di tale filosofia progettuale.

tuata da un'applicazione per ottenere il servizio di traduzione. Il DNS sull'host prende poi il controllo, inviando un messaggio di richiesta (*query*) sulla rete. Tutte le query DNS e i messaggi di risposta vengono inviati all'interno di datagrammi UDP diretti alla porta 53. Dopo un ritardo che varia dai millisecondi ai secondi, il client DNS sull'host dell'utente riceve un messaggio di risposta contenente la corrispondenza desiderata, che viene poi passata all'applicazione che ne ha fatto richiesta. Pertanto, dal punto di vista dell'applicazione nell'host utente, il DNS è una scatola nera che fornisce un servizio di traduzione semplice e diretto. Tuttavia nei fatti la scatola nera è costituita da un gran numero di DNS server distribuiti per il mondo e da un protocollo a livello di applicazione che specifica la comunicazione tra DNS server e host richiedenti.

Un primo approccio potrebbe essere quello di utilizzare un DNS server contenente tutte le corrispondenze. In questo schema centralizzato, i client dirigerebbero semplicemente tutte le richieste al singolo server e quest'ultimo risponderebbe loro direttamente. Sebbene tale semplicità progettuale sia attraente, sarebbe inappropriata per l'attuale Internet, dotata di un vasto e sempre crescente numero di host.

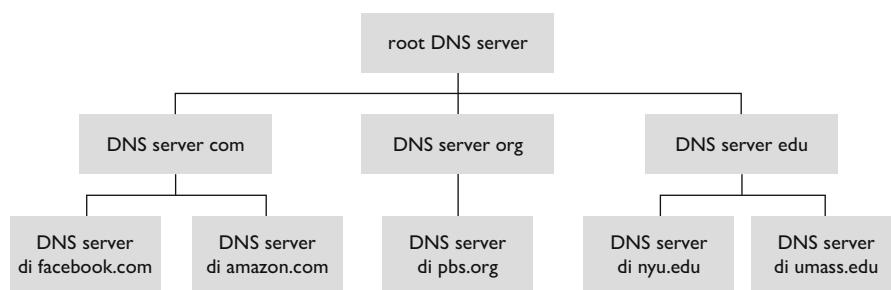
Tra i problemi legati a uno schema centralizzato ricordiamo i seguenti.

- **Un solo point of failure** (*punto di vulnerabilità*). Se il DNS server si guasta, ne soffre l'intera Internet.
- **Volume di traffico.** Un singolo DNS server dovrebbe gestire tutte le richieste (per tutte le richieste HTTP e i messaggi di posta elettronica generati da centinaia di milioni di host).
- **Database centralizzato distante.** Un singolo DNS server non può essere vicino a tutti i client. Se il server si trovasse a New York, le query provenienti dall'Australia dovrebbero viaggiare fino all'altro capo del mondo, magari su collegamenti lenti e congestionati, causando ritardi significativi.
- **Manutenzione.** Il singolo DNS server dovrebbe contenere record relativi a tutti gli host di Internet. Non solo tale database centralizzato sarebbe vasto, ma dovrebbe essere aggiornato frequentemente per tener conto di ogni nuovo host.

In conclusione, un database centralizzato su un singolo DNS server non è in grado di adattarsi alla crescita esponenziale della rete (ovvero, non è scalabile). Di conseguenza, il DNS è stato progettato in maniera distribuita e costituisce un magnifico esempio di come si possa implementare un database distribuito su Internet.

### **Un database distribuito e gerarchico**

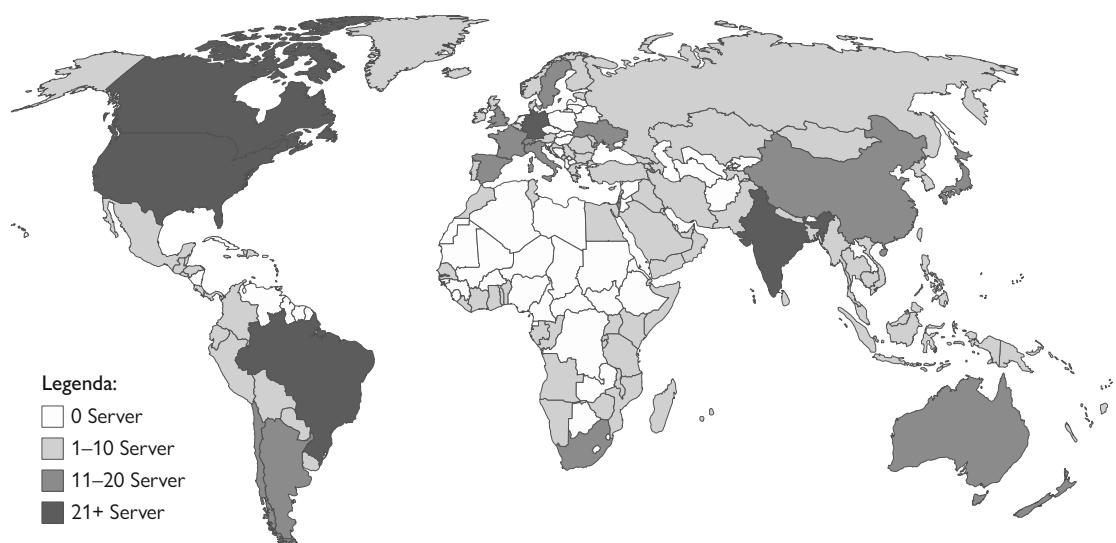
Per trattare il problema della scalabilità, il DNS utilizza un grande numero di server, organizzati in maniera gerarchica e distribuiti nel mondo. Nessun DNS server ha le corrispondenze per tutti gli host in Internet, che sono invece distribuite tra tutti i DNS server. In prima approssimazione, esistono tre classi di DNS server: i **root server**, i **top-level domain (TLD) server** e i **server autoritativi**, organizzati in una gerarchia, come mostrato nella Figura 2.17. Per comprendere l'interazione tra le classi, supponiamo che un client DNS voglia determinare l'indirizzo IP relativo all'hostname `www.amazon.com`. Per fare ciò, il client dap-



**Figura 2.17** Gerarchia parziale di server DNS.

prima contatta uno dei root server, che gli restituisce uno o più indirizzi IP relativi al TLD server per il dominio com (detto “dominio di primo livello” o anche “top-level”, in quanto è in cima alla gerarchia). Quindi contatta uno di questi TLD server, che gli restituisce uno o più indirizzi IP del server autoritativo per amazon.com. Infine, contatta uno dei server autoritativi per amazon.com, che gli restituisce l’indirizzo IP dell’hostname www.amazon.com. Esamineremo fra breve, più dettagliatamente, il processo di ricerca DNS. Prima però analizziamo più da vicino le tre classi di DNS server.

- **Root server.** In Internet esistono più di 1000 root server, dislocati in tutto il mondo. La dislocazione dei root server è indicata nella Figura 2.18. Tali root server sono copie di 13 differenti root server gestiti da 12 diverse organizzazioni, coordinate attraverso la *Internet Assigned Numbers Authority* [IANA 2020]. Il loro elenco insieme all’indirizzo IP e all’ente che gestisce ognuno dei root server è disponibile tramite [Root-servers 2020]. I root server forniscono gli indirizzi IP dei TLD server.
- **Top-level domain (TLD) server.** Questi server si occupano dei domini di primo livello quali com, org, net, edu e gov, e di tutti i domini di primo livello



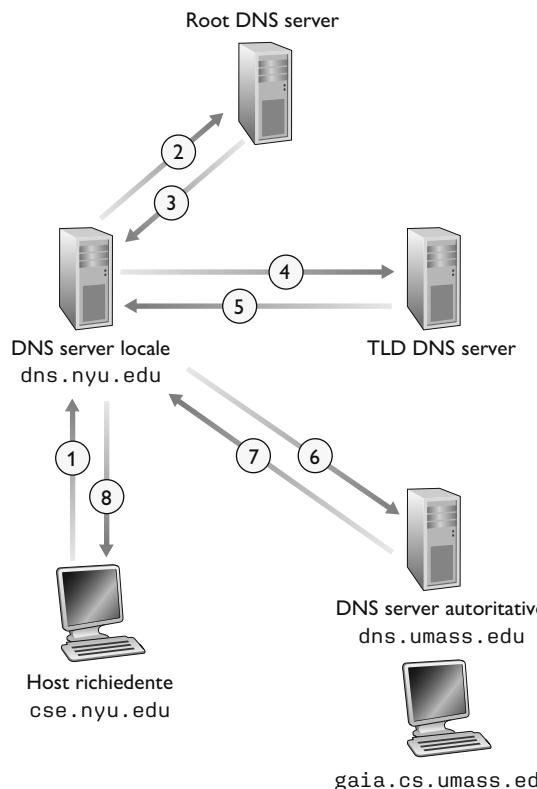
**Figura 2.18** Root DNS server nel 2020.

relativi ai vari paesi, come it, uk, fr, ca e jp. L’azienda Verisign Global Registry Services gestisce i TLD server per il dominio com (si veda [Osterweil 2012] per una bella trattazione di questa infrastruttura di rete così vasta e complessa) ed Educause amministra quelli per il dominio edu. Per una lista dei TLD server si veda [TLD list 2020]. I TLD server forniscono gli indirizzi IP dei server autoritativi.

- **DNS server autoritativi.** Ogni organizzazione dotata di host pubblicamente accessibili tramite Internet (quali web server e mail server) deve fornire record DNS pubblicamente accessibili che associno i nomi di tali host a indirizzi IP. Il DNS server autoritativo dell’organizzazione ospita questi record. Un’organizzazione può scegliere di implementare il proprio server autoritativo o di pagare un fornitore di servizi per ospitare questi record su un suo server. La maggior parte delle università e delle grandi società implementa e gestisce dei propri server autoritativi primario e secondario (di backup).

I DNS server appena descritti sono tutti collocati in una gerarchia di DNS server, come mostrato nella Figura 2.17. Esiste un altro importante tipo di DNS, detto **DNS server locale**, che non appartiene strettamente alla gerarchia di server, ma che è comunque centrale nell’architettura DNS. Ciascun ISP, come un’università, un dipartimento universitario, un’azienda o un ISP residenziale, ha un DNS server locale (detto anche **default name server**). Quando un host si connette a un ISP, quest’ultimo gli fornisce un indirizzo IP tratto da uno o più dei suoi DNS server locali, generalmente tramite DHCP (trattato nel Capitolo 4). Si può facilmente determinare l’indirizzo IP del proprio DNS server locale accedendo alla finestra di stato della rete in Windows o UNIX. Un DNS server locale è solitamente “vicino” all’host. Nel caso di un ISP istituzionale si può trovare sulla stessa rete locale dell’host, mentre negli ISP residenziali sono abitualmente separati da un numero limitato di router. Quando un host effettua una richiesta DNS, la query viene inviata al DNS server locale, che opera da proxy e inoltra la query alla gerarchia dei DNS server, come vedremo in maggiore dettaglio tra breve.

Consideriamo un semplice esempio. Supponiamo che l’host cse.nyu.edu voglia l’indirizzo IP di gaia.cs.umass.edu. Supponiamo, inoltre, che il DNS server locale per cse.nyu.edu sia dns.nyu.edu, mentre un server autoritativo per gaia.cs.umass.edu sia dns.umass.edu. Come mostrato nella Figura 2.19, l’host cse.nyu.edu dapprima invia un messaggio di richiesta DNS al proprio server locale dns.nyu.edu. Il messaggio contiene il nome da tradurre, ossia gaia.cs.umass.edu. Il server locale inoltra il messaggio di richiesta a un root server. Quest’ultimo prende nota del suffisso .edu e restituisce al server locale un elenco di indirizzi IP per i TLD server responsabili di .edu. Il server locale rinvia quindi il messaggio di richiesta a uno di questi ultimi. Il TLD server prende nota del suffisso .umass.edu e risponde con l’indirizzo IP del server autoritativo per l’Università del Massachusetts, ossia dns.umass.edu. Infine, il DNS server locale rimanda il messaggio di richiesta direttamente a dns.umass.edu, che risponde con l’indirizzo IP di gaia.cs.umass.edu. Si noti che in questo esempio, per ottenere la mappatura di un hostname, sono stati inviati ben otto messaggi DNS: quattro messaggi di richiesta e quattro messaggi di risposta. Presto vedremo come il caching riduca tale traffico.

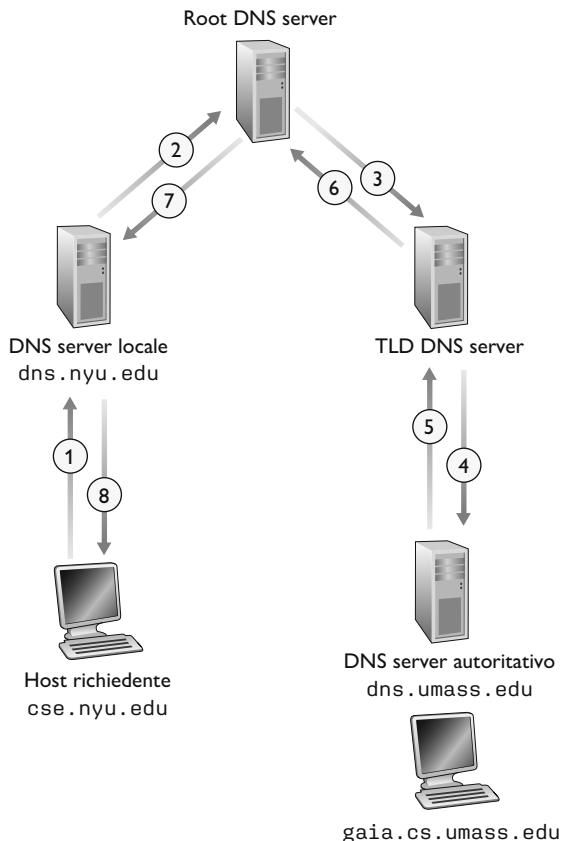


**Figura 2.19** Interazione tra i diversi DNS server.

Il nostro esempio ipotizzava che il TLD server conoscesse il server autoritativo per quel dato nome, ma in generale non è così. Può capitare che il TLD server conosca solo un DNS server intermedio, il quale a sua volta conosce il server autoritativo relativo all'hostname. Per esempio, supponiamo ancora che Università del Massachusetts abbia un proprio DNS server, chiamato dns.umass.edu. Ipotizziamo, inoltre, che ciascun dipartimento dell'università abbia un proprio server, competente per tutti gli host presenti in dipartimento. In questo caso, quando il DNS server intermedio, dns.umass.edu, riceve una richiesta da dns.nyu.edu per un host il cui nome termina con cs.umass.edu, gli restituisce l'indirizzo IP di dns.cs.umass.edu, che rappresenta il server autoritativo per tutti i nomi di host terminanti con cs.umass.edu. Il server locale dns.nyu.edu invia quindi la richiesta al server autoritativo, che restituisce la corrispondenza desiderata al DNS server locale, il quale a sua volta la restituisce all'host richiedente. In questo caso, vengono inviati ben dieci messaggi DNS.

L'esempio mostrato nella Figura 2.19 fa uso sia di **query ricorsive** sia di **query iterative**. La richiesta inviata da cse.nyu.edu a dns.nyu.edu è ricorsiva, in quanto richiede a dns.nyu.edu di ottenere l'associazione per conto del richiedente. Le successive tre richieste sono invece iterative, dato che tutte le risposte sono restituite direttamente a dns.nyu.edu. In teoria, ogni richiesta DNS può essere iterativa o ricorsiva. Per esempio, la Figura 2.20 mostra una concatenazione di richieste ricorsive. In pratica, le query seguono in genere lo schema della Figura 2.19: la richiesta dall'host iniziale al DNS server locale è ricorsiva, mentre le restanti richieste sono iterative.

**Figura 2.20** Query DNS ricorsive.



### DNS Caching

Fino a questo momento abbiamo ignorato il **DNS caching**, una caratteristica di fondamentale importanza. In verità, il DNS sfrutta in modo estensivo il caching per migliorare le prestazioni di ritardo e per ridurre il numero di messaggi DNS che “rimbalzano” su Internet. L’idea alla base del DNS caching è molto semplice. In una concatenazione di richieste, il DNS server che riceve una risposta DNS (contenente, per esempio, la traduzione da hostname a indirizzo IP), può mettere in cache le informazioni contenute.

Nella Figura 2.19, per esempio, ogni volta che il server locale `dns.nyu.edu` riceve una risposta da qualche DNS server, può conservare in cache le informazioni contenute nella risposta. Se una coppia `hostname/indirizzo IP` è nella cache di un DNS server e giunge al server un’altra richiesta con lo stesso `hostname`, il DNS server può fornire l’indirizzo IP desiderato, anche se non è autoritativo per tale indirizzo. Dato che gli host e le associazioni tra nome e indirizzo IP non sono in alcun modo permanenti, i DNS server invalidano le informazioni in cache dopo un periodo di tempo fissato (in genere di 2 giorni).

Supponiamo, per esempio, che un host `apricot.nyu.edu` richieda a `dns.nyu.edu` l’indirizzo IP dell’hostname `cnn.com`. Supponiamo, inoltre, che poche ore dopo un altro host della stessa organizzazione, per esempio `kiwi.nyu.edu`, faccia richiesta a `dns.nyu.edu` con lo stesso `hostname`. Grazie al caching, il DNS server locale sarà in grado di restituire immediatamente l’in-

dirizzo IP di `cnn.com` a questo secondo host richiedente, senza dover interrogare alcun altro DNS server. Un DNS server locale può, inoltre, memorizzare in cache gli indirizzi IP dei TLD server, consentendogli di aggirare i root server nella catena di richieste (e ciò avviene di frequente).

### 2.4.3 Record e messaggi DNS

I server che implementano il database distribuito di DNS memorizzano i cosiddetti **record di risorsa** (*RR, resource record*), tra cui quelli che forniscono le corrispondenze tra nomi e indirizzi. Ogni messaggio di risposta DNS trasporta uno o più record di risorse. In questo paragrafo e nei successivi presentiamo una breve panoramica dei record di risorsa e dei messaggi. Maggiori informazioni le potrete trovare in [Abitz 1993] o nelle RFC relative al DNS [RFC 1034; RFC 1035].

Un record di risorsa contiene i seguenti campi:

`(Name, Value, Type, TTL)`

TTL è il time to live, ossia il tempo residuo di vita di un record e determina quando una risorsa vada rimossa dalla cache. Nei record di esempio di seguito riportati ignoreremo il campo TTL. Il significato di Name e Value dipende da Type:

- Se `Type=A`, allora `Name` è il nome dell'host e `Value` è il suo indirizzo IP. Pertanto un record di tipo A fornisce la corrispondenza tra hostname standard e indirizzo IP. Per esempio, `(relay1.bar.foo.com, 145.37.93.126, A)` è un record di tipo A.
- Se `Type=NS`, allora `Name` è un dominio (quale `foo.com`) e `Value` è l'hostname del DNS server autoritativo che sa come ottenere gli indirizzi IP degli host nel dominio. Questo record viene usato per instradare le richieste DNS successive alla prima nella concatenazione delle query. Per esempio, `(foo.com, dns.foo.com, NS)` è un record di tipo NS.
- Se `Type=CNAME`, allora `Value` rappresenta il nome canonico dell'host per il sinonimo `Name`. Questo record può fornire agli host richiedenti il nome canonico relativo a un hostname. Per esempio, `(foo.com, relay1.bar.foo.com, CNAME)` è un record CNAME.
- Se `Type=MX`, allora `Value` è il nome canonico di un mail server che ha il sinonimo `Name`. Per esempio, `(foo.com, mail.bar.foo.com, MX)` è un record MX. Questo tipo di record consente agli hostname dei mail server di avere sinonimi semplici. Notiamo che, usando il record MX, una società può avere gli stessi sinonimi per il proprio server di posta e per uno dei propri altri server (per esempio, il web server). Per ottenere il nome canonico del server di posta, un client DNS dovrebbe interrogare un record MX. Per ottenere il nome canonico dell'altro server, il client DNS dovrebbe interrogare il record CNAME.<sup>8</sup>

---

<sup>8</sup> Solitamente si associa il record MX al nome del dominio per indicare che quel mail server è il punto di ingresso per la posta elettronica di tutto il dominio (*N.d.R.*).

Un DNS server autoritativo per un certo hostname contiene un record di tipo A per l'hostname. Anche se il server non fosse autoritativo, potrebbe tuttavia contenere un record di tipo A nella propria cache. Se un server non è quello autoritativo per un certo hostname, allora conterrà un record di tipo NS per il dominio che include l'hostname, e conterrà anche un record di tipo A che fornisce l'indirizzo IP del DNS server nel campo *Value* del record NS. Per esempio, supponiamo che un TLD server *edu* non sia competente per l'host *gaia.cs.umass.edu*. Allora conterrà un record per un dominio che include l'host *gaia.cs.umass.edu*, per esempio (*umass.edu*, *dns.umass.edu*, *NS*). Il TLD server *edu* conterebbe inoltre un record di tipo A, che indica il DNS server *dns.umass.edu* in un indirizzo IP, per esempio (*dns.umass.edu*, *128.119.40.111*, *A*).

### Messaggi DNS

Abbiamo precedentemente fatto riferimento agli unici due tipi di messaggio DNS: le query e i messaggi di risposta che presentano, entrambi, lo stesso formato (Figura 2.21).

La semantica dei campi dei messaggi DNS è la seguente.

- I primi 12 byte rappresentano la *sezione di intestazione (header section)*, che a sua volta contiene un certo numero di campi. Il primo è un numero di 16 bit che identifica la richiesta. Tale identificatore viene copiato nei messaggi di risposta a una richiesta, consentendo al client di far corrispondere le risposte ricevute con le query inviate. Troviamo poi il campo flag. Il primo di questi, il bit di richiesta/risposta (*query/reply*), indica se il messaggio è una richiesta (0) o una risposta (1). Un bit viene impostato nei messaggi di risposta quando il DNS server è autoritativo per il nome richiesto. Un ulteriore bit, chiamato **richiesta di ricorsione (recursion-desired flag)**, viene impostato quando un client (sia esso un host o un DNS server) desidera che il DNS server effettui ricorsione quando non dispone del record. Si imposta un campo

**Figura 2.21** Formato dei messaggi DNS.

| Identificazione   | Flag                     |  |
|---|--------------------------|--|
| Numero di domande   | Numero di RR di risposta | 12 byte  |
| Numero di RR autoritativi   | Numero di RR addizionali |  |
| Sezione delle domande<br>(numero variabile di domande)            |                          | Campi di nome e tipo di una query                      |
| Sezione delle risposte<br>(numero variabile di record di risorsa) |                          | RR in risposta alla query                              |
| Sezione autoritativa<br>(numero variabile di record di risorsa)   |                          | Record per i server autoritativi                       |
| Sezione aggiuntiva<br>(numero variabile di record di risorsa)     |                          | Informazione aggiuntiva “d'aiuto” che può essere usata |

da 1 bit di ricorsione disponibile (*recursion-available*) all'interno di una risposta se il DNS server supporta la ricorsione. Nell'intestazione troviamo anche quattro campi il cui nome comincia con “numero di”, che indicano il numero di occorrenze delle quattro sezioni di tipo dati che seguono l'intestazione.

- La *sezione delle domande* contiene informazioni sulle richieste che stanno per essere effettuate. Inoltre, include (1) un campo nome con il nome che sta per essere richiesto, e (2) un campo tipo che indica il tipo della domanda sul nome: per esempio, un indirizzo di host associato a un nome (tipo A) o il server di posta per un nome (tipo MX).
- In una risposta proveniente da DNS server, la *sezione delle risposte* contiene i record di risorsa relativi al nome originariamente richiesto. Ricordiamo che in ogni record di risorsa troviamo Type (A, NS, CNAME o MX), Value e TTL. Una risposta può restituire più RR, dato che un hostname può avere più indirizzi IP. È il caso dei web server replicati, cui si è fatto precedentemente cenno.
- La *sezione autoritativa* contiene i record di altri server autoritativi.
- La *sezione aggiuntiva* racchiude altri record utili: per esempio, se il campo di risposta relativo a una richiesta MX contiene un record di risorsa che fornisce l'hostname canonico del server di posta, la sezione aggiuntiva contiene un record di tipo A che fornisce l'indirizzo IP relativo all'hostname canonico del server di posta.

Vi piacerebbe inviare un messaggio di richiesta DNS a un DNS server, direttamente dalla macchina su cui state lavorando? Lo si può fare facilmente con il **programma nslookup**, disponibile nella maggior parte delle piattaforme Windows e UNIX. Per esempio, da un host Windows, aprete il prompt dei comandi e invocate il programma nslookup digitando semplicemente “nslookup”. Potete ora inviare una richiesta DNS a un qualsiasi DNS server (root, TDL o autoritativo). Una volta ricevuto il messaggio di risposta dal server, nslookup mostrerà i record inclusi nella risposta in un formato leggibile. In alternativa a lanciare nslookup dal vostro host potete visitare uno dei molti siti web che consentono di utilizzare tale servizio in remoto. È sufficiente digitare “nslookup” in un motore di ricerca e verrete indirizzati su uno di tali siti. Il laboratorio di Wireshark sul DNS alla fine di questo capitolo vi sarà utile per esplorare il DNS in maggiore dettaglio.

### Inserimento di record nel database DNS

La precedente trattazione si è concentrata su come vengono recuperati i record dal database DNS. A questo punto potreste chiedervi come siano inseriti i record nel database. Vediamolo nel contesto di uno specifico esempio. Supponiamo che abbiate appena fondato e avviato una nuova, entusiasmante società chiamata Network Utopia. La prima cosa che sicuramente desiderate fare è registrare il nome di dominio `networkutopia.com` presso un ente di registrazione (**registrar**). Un **registrar** è un'azienda che verifica l'unicità del nome di dominio, lo inserisce nel database DNS (come vedremo più avanti) e vi richiede una piccola somma di denaro per i propri servizi. Prima del 1999, un'uni-

ca società, la Network Solutions, aveva il monopolio sulla registrazione dei nomi di dominio terminanti in com, net e org. Ora però esistono molti registrar correnti, accreditati dalla *Internet Corporation for Assigned Names and Numbers* (ICANN). La loro lista completa è disponibile presso <http://www.internic.net>.

Quando registrate il nome di dominio `networkkutopia.com` presso un registrar, dovete fornirgli anche i nomi e gli indirizzi IP dei vostri DNS server autoritativi primario e secondario. Supponiamo che i nomi e gli indirizzi IP siano `dns1.networkkutopia.com`, `dns2.networkkutopia.com`, `212.2.212.1` e `212.212.212.2`. Per ciascuno di questi due server autoritativi l'ente si accerterà dell'inserimento di un record di tipo NS e di tipo A nei TLD server relativi al suffisso com. Più nello specifico, per il server autoritativo primario di `networkkutopia.com` il registrar inserirebbe nel sistema DNS i due seguenti record di risorsa:

```
(networkkutopia.com, dns1.networkkutopia.com, NS)  
(dns1.networkkutopia.com, 212.212.212.1, A)
```

Dovrete inoltre accertarvi che il record di risorsa di tipo A per il vostro web server `www.networkkutopia.com` e che il record di risorsa di tipo MX per il vostro server di posta `mail.networkkutopia.com` vengano immessi nei vostri DNS server autoritativi. Fino a poco tempo fa i contenuti di ciascun DNS server venivano configurati in modo statico, per esempio da un file di configurazione creato da un gestore del sistema. Più di recente, al protocollo DNS è stata aggiunta l'opzione UPDATE, per consentire l'aggiunta o la cancellazione dinamica di dati dal database attraverso messaggi DNS. Le [RFC 2136] e [RFC 3007] specificano questi aggiornamenti dinamici.

Una volta completati questi passi sarà possibile visitare il vostro sito web e spedire posta elettronica ai dipendenti della vostra società. Concludiamo la nostra trattazione su DNS verificando che questa affermazione sia vera. La verifica aiuta anche a consolidare ciò che abbiamo appena imparato. Supponiamo che Alice, in Australia, voglia visitare la pagina web `www.networkkutopia.com`. Come detto precedentemente, il suo host invierà una richiesta DNS al suo DNS server locale. Quest'ultimo contatterà poi un TLD server per il suffisso com. Il DNS server locale dovrà inoltre contattare un root server nel caso in cui l'indirizzo del TLD server responsabile per il suffisso com non si trovi in cache. Questo TLD server contiene i record di risorsa di tipo NS e A elencati precedentemente, dato che il registrar li ha inseriti in tutti i TLD server relativi al suffisso com. Il TLD server di com invia una risposta al server locale di Alice, e la risposta contiene i due record di risorsa. Il server locale invia poi una richiesta DNS a `212.212.212.1`, cercando il record di tipo A che corrisponde a `www.networkkutopia.com`. Tale record fornisce l'indirizzo IP del web server desiderato, per esempio `212.212.71.4`, restituito poi dal server locale all'host di Alice. Il browser di Alice può ora inizializzare una connessione TCP verso l'host `212.212.71.4` e inviare una richiesta HTTP sulla connessione. Wow! Succede molto di più di quello che si riesce a vedere quando usiamo il Web!

**BOX 2.3** **FOCUS SULLA SICUREZZA****Vulnerabilità del DNS**

Abbiamo visto che il DNS è un componente critico dell'infrastruttura di Internet, in quanto molti servizi importanti, tra cui il Web e la posta elettronica, non possono farne a meno. Ci chiediamo naturalmente come sia possibile attaccare il DNS. È il DNS un facile bersaglio in attesa di essere messo fuori combattimento, trascinando con sé molte altre applicazioni Internet?

Il primo attacco che viene in mente è quello DDoS con flooding di banda (si veda il Paragrafo 1.6) contro i DNS server. Per esempio, un attaccante può tentare di inondare di pacchetti ciascun root server, in modo che molte delle richieste DNS legittime rimangano senza risposta. Un tipo di attacco DDoS su così larga scala a DNS server ebbe effettivamente luogo il 21 ottobre 2002. In questo attacco, gli aggressori fecero leva su una botnet per mandare enormi quantità di messaggi ICMP (come utilizzati dall'applicativo "ping") a ciascuno dei 13 root server. I messaggi ICMP saranno trattati nel Paragrafo 5.6; per ora è sufficiente sapere che i pacchetti ICMP sono speciali datagrammi IP. Fortunatamente, questo attacco su larga scala causò danni contenuti, avendo avuto un impatto minimo o nullo sull'uso di Internet da parte degli utenti. Gli attaccanti ebbero successo nell'inondare di pacchetti i root server, ma molti di questi server erano protetti da sistemi di filtraggio dei pacchetti, configurati per bloccare i messaggi ICMP. I server furono così risparmiati e funzionarono come al solito. Inoltre, molti DNS server locali mantengono in cache gli indirizzi IP dei server di livello più alto, consentendo spesso al processo di richiesta di non dover interagire con i root server.

Un attacco DDoS potenzialmente più efficace contro il DNS sarebbe quello di inondare di richieste DNS i server di primo livello, per esempio tutti i server che gestiscono il dominio .com. Con i DNS server di primo livello sarebbe più difficile filtrare le richieste DNS e il loro uso non può essere evitato facilmente come nel caso dei root server. Tuttavia la gravità di questo attacco sarebbe mitigata dalle cache dei DNS server locali. Un attacco di questo tipo è avvenuto contro il server di primo livello Dyn il 21 ottobre 2016. Questo attacco DDoS fu realizzato attraverso un gran numero di richieste DNS da una botnet composta da circa centomila dispositivi IoT come stampanti, telecamere IP, gateway residenziali e baby monitor che erano stati infettati dal malware Mirai. Per quasi un giorno intero, Amazon, Twitter, Netflix, Github e Spotify ebbero dei problemi.

Teoricamente, il DNS può essere attaccato in altri modi. In un attacco di tipo man-in-the-middle l'aggressore intercetta le richieste di un host e fornisce risposte. Diversamente, nell'attacco di DNS poisoning, l'attaccante invia risposte ingannevoli, riuscendo a far inserire record fasulli nella cache del DNS server. L'uno o l'altro di questi attacchi possono essere usati, per esempio, per ri-direzionare un utente web ignaro verso il sito web dell'attaccante. Sono state progettate ed implementate delle DNS Security Extensions (DNSSEC [Gieben 2004; RFC 4033]). DNSSEC, una versione sicura del DNS, affronta molti di questi possibili attacchi e sta guadagnando popolarità in Internet.

## 2.5 Distribuzione di file P2P

Tutte le applicazioni descritte in questo capitolo, compreso il Web, la posta elettronica e il DNS, utilizzano un'architettura client-server con una significativa dipendenza da un'infrastruttura di server sempre attivi. Ricordiamo dal Paragrafo 2.1.1 che nell'architettura peer-to-peer (P2P) questa dipendenza è

minima o del tutto assente. Ci sono, invece, coppie di host connessi in modo intermittente, chiamati peer, che comunicano direttamente l'uno con l'altro. I peer non appartengono ai fornitori dei servizi, ma sono computer fissi e portatili e smartphone controllati dagli utenti.

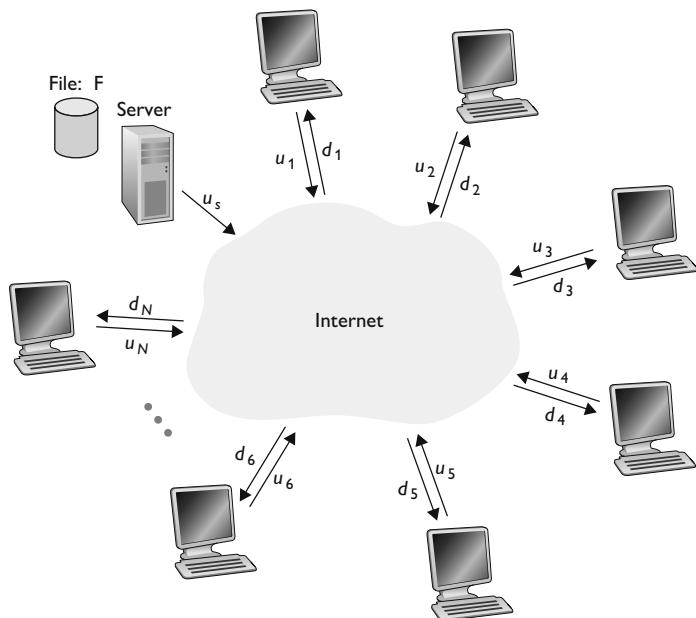
In questo paragrafo esamineremo un'applicazione particolarmente adatta per una struttura P2P: la distribuzione di un file voluminoso da un singolo server a un gran numero di host, chiamati peer. Il file potrebbe essere una nuova versione del sistema operativo Linux, una patch software per un'applicazione o un sistema operativo esistente o un file video MPEG. In una distribuzione di file client-server, il server deve inviare una copia del file a ciascun peer, ponendo un enorme fardello sul server e consumandone un'elevata quantità di banda. In una distribuzione di file con P2P ciascun peer può ridistribuire agli altri qualsiasi porzione del file abbia ricevuto, aiutando in questo modo il server nel processo di distribuzione. Attualmente, nel 2020, uno tra i più diffusi protocolli di distribuzione di file tramite P2P è BitTorrent. Originariamente sviluppato da Bram Cohen, esistono oggi diversi client BitTorrent indipendenti e conformi al protocollo BitTorrent, esattamente come ci sono parecchi client web, che sono conformi al protocollo HTTP. In questo paragrafo esamineremo dapprima la scalabilità dell'architettura P2P nel contesto della distribuzione di file e poi descriveremo BitTorrent, evidenziandone le caratteristiche e gli aspetti principali.

### Scalabilità dell'architettura P2P

Per confrontare le architetture client-server e P2P e illustrare la scalabilità intrinseca di quest'ultima, considereremo ora un semplice modello quantitativo per la distribuzione di file a un insieme fisso di peer per entrambe le tipologie di architettura. Come mostrato nella Figura 2.22, server e peer sono collegati a Internet con collegamenti di accesso: sia  $u_s$  la banda di upload del collegamento

**Figura 2.22**

Un problema illustrativo di distribuzione file.



di accesso del server,  $u_i$  la banda di upload del collegamento di accesso dell' $i$ -esimo peer e  $d_i$  la banda di download del collegamento di accesso dell' $i$ -esimo peer. Siano, inoltre,  $F$  la dimensione del file da distribuire (in bit) e  $N$  il numero di peer che vuole una copia del file. Il **tempo di distribuzione** (*distribution time*) è il tempo richiesto perché tutti gli  $N$  peer ottengano una copia del file. Nella nostra analisi sul tempo di distribuzione per entrambe le architetture client-server e P2P, per semplificare facciamo l'ipotesi (che si dimostra essere piuttosto accurata [Akella 2003]), che il nucleo di Internet abbia banda in abbondanza: ciò implica che tutti i colli di bottiglia siano nelle reti di accesso. Supponiamo anche che i server e i peer/client non stiano partecipando a nessun'altra applicazione di rete, in modo che la loro banda di accesso in upload e download possa essere completamente dedicata alla distribuzione di questo file.

Determiniamo in primo luogo il tempo di distribuzione del file per l'architettura client-server, che indichiamo con  $D_{cs}$ . Nell'architettura client-server nessuno dei peer aiuta nella distribuzione del file.

Facciamo le seguenti osservazioni.

- Il server deve trasmettere una copia del file a ciascuno degli  $N$  peer, cioè  $NF$  bit. Dato che la banda di upload del server è  $u_s$ , il tempo per distribuire il file deve essere almeno  $NF/u_s$ .
- Sia  $d_{\min}$  la banda di download del peer avente il valore più basso, cioè  $d_{\min} = \min\{d_1, d_2, \dots, d_N\}$ . Il peer con la banda di download più bassa non può ricevere tutti gli  $F$  bit del file in meno di  $F/d_{\min}$  secondi. Quindi il tempo minimo di distribuzione è almeno  $F/d_{\min}$ .

Mettendo assieme queste due osservazioni otteniamo che:

$$D_{cs} \geq \max\left\{\frac{NF}{u_s}, \frac{F}{d_{\min}}\right\}.$$

Ciò fornisce un limite inferiore al tempo di distribuzione minimo per l'architettura client-server. In un problema in fondo al capitolo vi verrà chiesto di mostrare che il server può programmare la sua trasmissione in modo che il limite inferiore sia effettivamente raggiunto. Consideriamo questo limite inferiore, fornito precedentemente, come il tempo di distribuzione effettivo, cioè:

$$D_{cs} = \max\left\{\frac{NF}{u_s}, \frac{F}{d_{\min}}\right\}. \quad (2.1)$$

Vediamo dall'Equazione 2.1 che per  $N$  sufficientemente grande, il tempo di distribuzione nel caso client-server è generalmente dato da  $NF/u_s$ . Quindi, il tempo di distribuzione aumenta linearmente con il numero  $N$  di peer. Per esempio, se il numero di peer da una settimana alla successiva aumenta da un migliaio a un milione, il tempo richiesto per distribuire il file a tutti i peer aumenta di 1000 volte.

Facciamo adesso un'analogia analisi per l'architettura P2P, nella quale ciascun peer assiste il server nella distribuzione del file. In particolare, quando un peer riceve alcuni dati del file, può usare la propria capacità di upload per redistribuire i dati agli altri peer. Il calcolo del tempo di distribuzione per un'architettura P2P è più complicato che per l'architettura client-server perché di-

pende da come ciascun peer distribuisce le porzioni del file agli altri peer. Ciononostante si può ottenere una semplice espressione del tempo minimo di distribuzione [Kumar 2006]. A questo scopo, facciamo in primo luogo le seguenti osservazioni.

- All'inizio della distribuzione solo il server dispone del file. Per trasmetterlo all'interno della comunità dei peer il server deve inviare ciascun bit del file almeno una volta nel collegamento di accesso. Quindi il minimo tempo di distribuzione è  $F/u_s$ . Diversamente dallo schema client-server, un bit inviato una volta dal server può non dover essere inviato di nuovo, in quanto i peer possono re-distribuire i bit tra loro.
- Come per l'architettura client-server, il peer con la velocità di download più bassa non può ottenere tutti i bit del file in meno di  $F/d_{\min}$  secondi. Quindi il tempo minimo di distribuzione è almeno  $F/d_{\min}$ .
- Infine, si osservi che la capacità totale di upload del sistema nel suo complesso è uguale alla velocità di upload del server più quella di ciascun peer, cioè  $u_{\text{tot}} = u_s + u_1 + \dots + u_N$ . Il sistema deve consegnare (upload)  $F$  bit a ciascuno degli  $N$  peer, consegnando quindi un totale di  $NF$  bit. Ciò non può essere fatto a una velocità più grande di  $u_{\text{tot}}$ . Quindi, il tempo di distribuzione minimo è almeno:

$$NF/(u_s + u_1 + \dots + u_N).$$

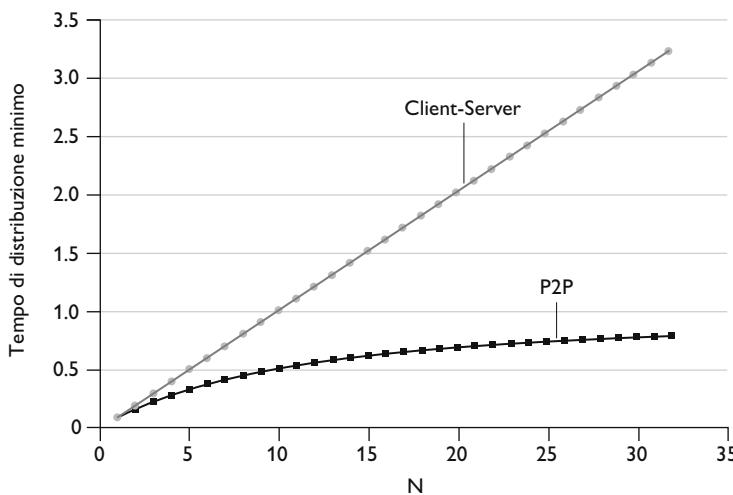
Mettendo assieme queste tre osservazioni, otteniamo il tempo di distribuzione minimo per l'architettura P2P, indicato con  $D_{\text{P2P}}$ :

$$D_{\text{P2P}} \geq \max \left\{ \frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{NF}{u_s + \sum_{i=1}^N u_i} \right\} \quad (2.2)$$

L'Equazione 2.2 fornisce un limite inferiore al tempo di distribuzione minimo per l'architettura peer-to-peer. Risulta che, se immaginiamo che ciascun peer possa re-distribuire un bit appena lo riceve, vi sia uno schema di re-distribuzione che effettivamente raggiunge questo limite inferiore [Kumar 2006]. Dimostreremo un caso speciale di questo risultato nei problemi a fine capitolo. In realtà, se al posto dei bit singoli vengono re-distribuiti grosse porzioni del file, l'Equazione 2.2 serve come buona approssimazione dell'effettivo tempo minimo di distribuzione. Quindi, consideriamo il limite inferiore fornito dall'Equazione 2.2 come il tempo di distribuzione minimo, cioè:

$$D_{\text{P2P}} = \max \left\{ \frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{NF}{u_s + \sum_{i=1}^N u_i} \right\} \quad (2.3)$$

Nella Figura 2.23 è illustrato il confronto tra il tempo di distribuzione minimo per le architetture client-server e P2P, supponendo che tutti i peer abbiano la stessa velocità di upload  $u$ . Nella Figura 2.23 abbiamo posto  $F/u = 1$  ora,  $u_s = 10u$  e  $d_{\min} \geq u_s$ . Quindi, un peer può trasmettere l'intero file in un'ora, la velocità di trasmissione del server è dieci volte la velocità di upload dei peer e, per semplicità, le velocità di download dei peer sono grandi abbastanza da non avere



**Figura 2.23** Tempo di distribuzione per le architetture P2P e client-server.

effetto. Dalla Figura 2.23 vediamo che, nell’architettura client-server, il tempo di distribuzione cresce linearmente e senza limite al crescere del numero di peer. Tuttavia, nell’architettura P2P, il tempo di distribuzione minimo non è solo sempre minore di quello dell’architettura client-server, ma è anche minore di un’ora per qualsiasi numero di peer  $N$ . Quindi, le applicazioni con architettura P2P possono essere scalabili e la scalabilità è una diretta conseguenza del fatto che i peer re-distribuiscono i bit oltre che a scaricarli.

### BitTorrent

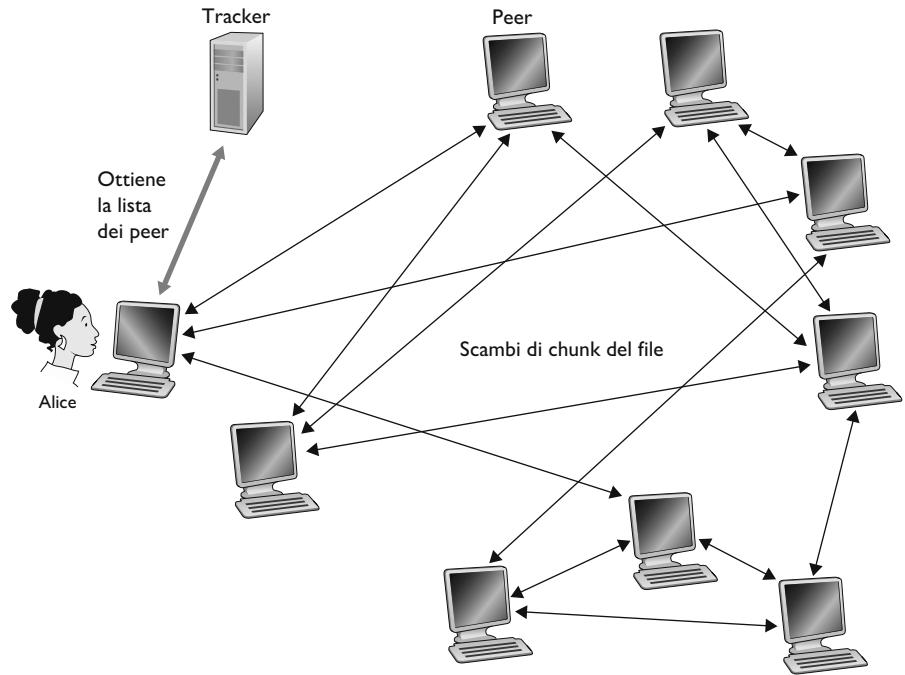
BitTorrent è un diffuso protocollo P2P per la distribuzione di file [Chao 2011]. Nel gergo di BitTorrent, l’insieme di tutti i peer che partecipano alla distribuzione di un particolare file è chiamato **torrent** (*torrente*). I peer in un torrent scaricano **chunk** (*parti*) del file di uguale dimensione, con una dimensione tipica di 256 kbyte. Un peer, quando entra a far parte di un torrent per la prima volta, non ha chunk del file. Col passare del tempo accumula sempre più parti che, mentre scarica, invia agli altri peer. Una volta che un peer ha acquisito l’intero file, può (egoisticamente) lasciare il torrent o (altruisticamente) rimanere nel torrent e continuare a inviare chunk agli altri peer. Inoltre, qualsiasi peer può lasciare il torrent in qualsiasi momento con solo un sottoinsieme dei chunk del file e rientrare a far parte del torrent in seguito.

Illustriamo ora più da vicino come funziona BitTorrent. Dato che si tratta di un protocollo piuttosto complicato, descriveremo solo i suoi meccanismi principali, lasciando da parte alcuni dettagli. Ciascun torrent ha un nodo di infrastruttura chiamato **tracker** (*tracciatore*). Quando un peer entra a far parte di un torrent, si registra presso il tracker e periodicamente lo informa che è ancora nel torrent. In questo modo, il tracker tiene traccia dei peer che stanno partecipando al torrent. Un certo torrent può avere centinaia o migliaia di peer che partecipano in un dato istante.

Come mostrato nella Figura 2.24, quando un nuovo peer, Alice, entra a far parte di un torrent, il tracker seleziona in modo casuale un sottoinsieme di peer (diciamo 50) dall’insieme dei peer che stanno partecipando a quel torrent, e in-

**Figura 2.24**

Distribuzione di file con BitTorrent.



via l'indirizzo IP di questi 50 peer ad Alice. Avendo la lista dei peer, Alice cerca di stabilire delle connessioni TCP contemporanee con tutti i peer della lista. Chiamiamo i peer con i quali Alice riesce a stabilire una connessione TCP "peer vicini" (*neighboring peer*). Nella Figura 2.24 si vede che Alice ha solo tre peer vicini (ma normalmente ne avrebbe molti di più). Col passare del tempo, alcuni di questi peer possono lasciare il torrent, mentre altri (al di fuori dei 50 iniziali) possono cercare di stabilire una connessione TCP con Alice: quindi i peer vicini a un dato peer cambiano nel tempo.

A un determinato istante ciascun peer avrà un sottoinsieme dei chunk di un file e peer diversi ne avranno differenti sottoinsiemi. Periodicamente Alice chiederà a ciascuno dei suoi vicini, tramite le connessioni TCP, la lista dei chunk del file in loro possesso. Tramite questa conoscenza, Alice invierà richieste, di nuovo sulle connessioni TCP, per i chunk del file che ancora le mancano.

Di conseguenza, in un dato istante Alice avrà un sottoinsieme dei chunk del file e saprà quali chunk hanno i suoi vicini. Con queste informazioni, Alice deve prendere due importanti decisioni: in primo luogo quali chunk deve richiedere per primi ai suoi vicini e, secondariamente, a quali vicini dovrebbe inviare i chunk a lei richiesti. Nel decidere quali chunk richiedere, Alice adotta la tecnica del **rarest first** ("il più raro per primo"). L'idea è determinare quali sono i chunk più rari tra quelli che ancora le mancano, cioè i chunk con il minor numero di copie ripetute tra i suoi vicini, e richiederli per primi. In questo modo, i chunk più rari vengono ridistribuiti più velocemente, cercando, approssimativamente, di rendere uguale il numero di copie di ciascun chunk nel torrent.

Per determinare a quali richieste Alice debba rispondere, BitTorrent usa un intelligente algoritmo di trading ("scambio"). L'idea di base è che Alice attribuisca priorità ai vicini che le stanno inviando dati in questo momento alla ve-

locità più alta. Specificatamente, per ciascuno dei suoi vicini, Alice misura continuamente la velocità alla quale riceve i bit e determina i quattro peer che le stanno passando i bit alla velocità più elevata. Alice poi contraccambia inviando chunk del file a quegli stessi quattro peer. Ogni 10 secondi ricalcola la velocità e può darsi che modifichi l'insieme dei quattro peer. Nel gergo di BitTorrent questi quattro peer vengono detti **unchoked** (“non soffocati” o “non limitati”). Un aspetto importante è che ogni 30 secondi sceglie casualmente un vicino in più e gli invia dei chunk. Chiamiamo Bob il peer scelto casualmente. Nel gergo di BitTorrent Bob viene detto **optimistically unchoked** (“non limitato/soffocato in maniera ottimistica”). Dato che Alice sta inviando dati a Bob, potrebbe diventare uno dei quattro peer unchoked di Bob, nel qual caso Bob inizierebbe a inviare dati ad Alice. Se la velocità alla quale Bob manda i dati ad Alice fosse abbastanza alta, Bob, a sua volta, potrebbe diventare uno dei quattro peer unchoked di Alice. In altre parole, ogni 30 secondi Alice sceglie casualmente un nuovo compagno di scambi e inizia a scambiare chunk con quello. Se i due peer sono soddisfatti degli scambi, l'uno metterà l'altro nella propria lista dei quattro unchoked e continueranno a effettuare scambi tra loro finché uno non trovi un partner migliore. L'effetto è che i peer in grado di inviare dati a velocità compatibili tendono a trovarsi. La selezione casuale dei vicini consente anche a nuovi peer di ottenere chunk del file, in modo che abbiano qualcosa da scambiare. Tutti gli altri peer, a parte quei cinque (i quattro unchoked e un peer di prova), sono detti *choked* (“soffocati”, “limitati”), cioè non ricevono alcun chunk da Alice. BitTorrent possiede molti altri meccanismi interessanti che non verranno trattati qui (mini chunk, pipelining, prima selezione casuale, modalità endgame e anti-snubbing) [Cohen 2003].

Il meccanismo di incentivazione degli scambi descritto precedentemente viene spesso chiamato *tit-for-tat* (“pan per focaccia”) [Cohen 2003]. Benché sia stato dimostrato che può essere aggirato [Liogkas 2006, Locher 2006, Piatek 2008], l'ecosistema BitTorrent ha un grande successo con milioni di peer che si scambiano file simultaneamente in centinaia di migliaia di torrent. Se BitTorrent fosse stato progettato senza tit-for-tat o una sua variante, probabilmente ora non esisterebbe più, perché la maggior parte degli utenti sarebbe stata dei “freeriders”, avrebbe cioè scaricato senza contribuire al torrent [Saroiu, 2002].

Chiudiamo la trattazione sulle applicazioni P2P accennando alle **tabelle hash distribuite (DHT)**, semplici database i cui record sono distribuiti tra i peer di un sistema P2P. Esse sono molto diffusamente implementate, per esempio in BitTorrent, e studiate. Si veda la VideoNote per una trattazione più ampia.

## 2.6 Streaming video e reti per la distribuzione di contenuti

Si stima che lo streaming video, quale Netflix, YouTube e Amazon Prime, rappresenti circa l'80% del traffico Internet nel 2020 [Cisco 2020]. In questo paragrafo forniremo una panoramica di come i servizi di streaming video vengano implementati nell'Internet di oggi. Vedremo come vengono implementati utilizzando protocolli del livello di applicazione e server che funzionano in un certo senso come una cache.

### 2.6.1 Video su Internet

Nelle applicazioni di streaming di video registrati, i contenuti sono video, quali film, trasmissioni televisive, eventi sportivi o video registrati, come quelli su YouTube, memorizzati su server a disposizione degli utenti su richiesta (*on demand*). Migliaia di siti forniscono oggi streaming di audio e video registrati, tra cui Netflix, YouTube (Google), Amazon e TikTok.

Prima di entrare nel dettaglio della discussione sullo streaming video è utile vedere le caratteristiche dei video. Un video è una sequenza di immagini, visualizzate tipicamente a velocità costante di, per esempio, 24 o 30 immagini al secondo. Un'immagine non compressa e codificata digitalmente consiste di un array di pixel, ognuno dei quali codificato con un numero di bit per rappresentare luminanza e crominanza. I video possono essere compressi in modo da raggiungere un compromesso tra qualità del video e bit rate. Gli algoritmi di compressione oggi disponibili sono in grado di comprimere un video a qualsiasi bit rate si desideri. Ovviamente, più alto è il bit rate, migliore è la qualità dell'immagine e l'esperienza visiva globale dell'utente.

Dal punto di vista del networking, forse la caratteristica più saliente del video è l'elevata velocità con cui è necessario inviare i bit sulla rete (**bit rate**). I video distribuiti in Internet variano da 100 kbps per i video a bassa qualità a oltre 4 Mbps per i film in streaming ad alta definizione per arrivare a più di 10 Mbps per i video 4K. Ciò può tradursi in enormi quantità di traffico e di spazio di archiviazione (storage). Per esempio, un singolo video di 2 Mbps con una durata di 67 minuti consuma 1 gigabyte di spazio di archiviazione e di traffico. La misura in assoluto più importante per lo streaming video è il throughput medio, il cui valore deve essere almeno pari a quello del bit rate del video per avere una riproduzione continua.

La compressione può essere usata anche per creare versioni multiple dello stesso video, a livelli di qualità diversi. Per esempio, si può usare la compressione per creare tre versioni dello stesso video, a 300 kbps, 1 Mbps e 3 Mbps. Gli utenti possono decidere quale versione vogliono guardare in funzione della larghezza di banda disponibile. Gli utenti con connessioni Internet ad alta velocità potrebbero scegliere la versione a 3 Mbps; gli utenti che guardano i video tramite 3G su uno smartphone potrebbero scegliere quella a 300 kbps.

### 2.6.2 Streaming HTTP e DASH

Nello streaming HTTP il video viene semplicemente memorizzato in un server HTTP come un file ordinario con un URL specifico. Quando un utente vuole vedere un video, il client stabilisce una connessione TCP con il server e invia una richiesta GET HTTP per il suo URL. Il server invia il file video, all'interno di un messaggio di risposta HTTP, più velocemente possibile dati il traffico e i protocolli di rete. Sul lato client i byte vengono memorizzati in un buffer dell'applicazione client. Quando il numero di byte nel buffer supera una soglia fissata, l'applicazione client inizia la riproduzione: periodicamente prende i frame video dal buffer, li decomprime e li visualizza all'utente. Quindi l'applicazione di video streaming consiste nel visualizzare i frame mentre li riceve, memorizzando nel buffer gli ultimi.

Lo streaming HTTP, sebbene molto usato dalle aziende del settore, quali YouTube, presenta un grande svantaggio: i client ricevono tutti la stessa versio-

ne codificata del video, nonostante abbiano a disposizione una larghezza di banda che può variare sensibilmente da un caso all'altro e nel tempo. Per superare il problema è stato sviluppato un nuovo tipo di streaming basato su HTTP, chiamato **streaming dinamico adattativo su HTTP** (*Dynamic Adaptive Streaming Over HTTP*, DASH). In DASH, i video vengono codificati in diverse versioni, ognuna avendo un bit rate differente e quindi un differente livello di qualità. Il client richiede pezzi di segmenti video lunghi alcuni secondi da versioni differenti in modo dinamico. Quando la banda disponibile è elevata, il client seleziona automaticamente i blocchi da una versione con alto bit rate, mentre quando la banda disponibile è poca, seleziona una versione a basso bit rate. Il client seleziona un blocco alla volta con un messaggio di richiesta GET HTTP [Akhshabi 2011].

Da una parte, DASH permette a client con accessi a Internet diversi di fare streaming dei video con codifiche diverse: client con connessioni lente 3G possono ricevere versioni a basso bit rate e quindi a bassa qualità, mentre client con connessioni in fibra possono ricevere una versione ad alta qualità. D'altro canto, DASH permette anche a un client di adattare la scelta della versione alla banda disponibile se varia durante la sessione. Tale caratteristica è particolarmente importante per gli utenti mobili, per i quali la banda disponibile fluttua quando cambiano stazione base.

Con DASH, i video nelle varie versioni sono memorizzati nel server HTTP, ognuno a un URL diverso. Il server HTTP ha anche un file di descrizione (detto anche **manifest file**, o **file manifesto**) che per ogni versione fornisce il rispettivo URL insieme al bit rate. Il client richiede innanzitutto il file di descrizione per venire a conoscenza delle varie versioni disponibili. Quindi seleziona a ogni istante un blocco, specificando nel messaggio di richiesta GET di HTTP un URL e un intervallo di byte. Mentre scarica i blocchi, il client misura la banda di ricezione ed esegue un algoritmo per selezionare il blocco successivo. Naturalmente il client, se ha molti dati del video memorizzati nel buffer e la banda di ricezione misurata è larga, sceglie una versione ad alto bit rate per il blocco successivo; al contrario, se ha pochi dati nel buffer e banda bassa, sceglie una versione a basso rate. Pertanto DASH permette al client di cambiare liberamente il livello di qualità.

### 2.6.3 Reti per la distribuzione di contenuti

Oggi giorno molte aziende di video in Internet distribuiscono quotidianamente video on demand con streaming dell'ordine dei Mbps a milioni di utenti. YouTube, per esempio, con un'offerta di centinaia di milioni di video, li distribuisce costantemente a utenti sparsi in tutto il mondo. È chiaro che mandare in streaming una tale mole di traffico a utenti sparsi in tutto il mondo, fornendo riproduzione continua e alta interattività, è davvero una grande sfida.

Forse l'approccio più diretto per le aziende di streaming video in Internet sarebbe costruire un unico enorme data center, memorizzare in esso tutti i video e mandarli in streaming direttamente. Tale approccio però presenta tre grandi problemi. In primo luogo, se il client è lontano dal data center, i pacchetti dal server al client devono percorrere un lungo cammino passando per molti ISP, magari in continenti diversi. Se uno dei collegamenti ha un throughput minore della velocità di consumo del video, anche il throughput totale end-to-end lo

sarà, causando all’utente continui e fastidiosi fermi immagine. Infatti, come visto nel Capitolo 1, il throughput end-to-end è determinato da quello del collegamento che fa da collo di bottiglia, quindi più grande è il numero di collegamenti da attraversare, più è probabile trovare un collo di bottiglia. Un secondo svantaggio deriva dal fatto che un video molto popolare verrebbe inviato molte volte sullo stesso collegamento, non solo sprecando banda, ma costringendo anche l’azienda a pagare tutte le volte il suo ISP (collegato al data center) per inviare gli stessi dati. Un terzo problema riguarda il fatto che un singolo data center rappresenta un singolo punto di rottura: se il data center o il collegamento da esso a Internet si interrompesse, l’azienda non sarebbe più in grado di distribuire alcun video.

Per superare queste problematiche, quasi tutte le maggiori aziende di video streaming usano le **CDN** (*Content Distribution Networks*, reti di distribuzioni di contenuti). Una CDN gestisce server distribuiti in molti posti diversi, memorizza copie dei video e di altri contenuti web nei server e cerca di dirigere le richieste degli utenti al punto della CDN in grado di offrire il servizio migliore. La CDN può essere una **CDN privata**, cioè proprietaria del fornitore di contenuti, come la CDN di Google che distribuisce i contenuti di YouTube. Alternativamente può essere la **CDN di terze parti** che distribuisce contenuti per conto di molti fornitori di contenuti come Akamai, Limelight e Level-3. Per una trattazione sulle moderne CDN si veda [Leighton 2009; Nygren 2010].

Le CDN generalmente adottano una delle due politiche di dislocazione dei server che seguono [Huang 2008]:

- **Enter deep** (“entrare in profondità”). Una filosofia, proposta da Akamai, è quella di entrare in profondità nelle reti di accesso degli ISP installando gruppi di server, detti anche cluster, negli ISP di accesso sparsi in tutto il mondo (Paragrafo 1.3). Akamai usa tale approccio distribuendo server in migliaia di posti diversi. L’obiettivo è quello di essere vicini agli utenti finali in modo da migliorare il ritardo percepito dall’utente e il throughput, diminuendo il numero di collegamenti e router tra l’utente finale e il cluster CDN da cui riceve i contenuti. Tale approccio, altamente distribuito, pone però grandi sfide nella manutenzione e gestione dei cluster.
- **Bring home** (“portare a casa”). Il secondo approccio, seguito da Limelight e molti altri fornitori di CDN, è quello di portarsi in casa l’ISP costruendo grandi cluster in pochi (decine, per esempio) punti chiave e interconnetterli usando una rete privata ad alta velocità. Invece di entrare negli ISP di accesso, queste CDN pongono i loro cluster negli IXP (Paragrafo 1.3). Tale approccio presenta meno problemi di gestione e manutenzione, ma spesso anche una minore qualità di servizio.

Una volta posizionati i cluster, la CDN replica i contenuti su di essi. I video raramente richiesti o che sono popolari solo in alcuni paesi non vengono copiati su tutti i cluster. Molte CDN non spingono (*push*) i loro video verso i cluster, ma usano una semplice strategia per cui se un client richiede un video a un cluster che non lo ha memorizzato, il cluster recupera il video da un archivio centrale o da un altro cluster e ne memorizza localmente una copia mentre lo manda in streaming al client. Come nel caso dei proxy per il Web (Paragrafo 2.2.5) i video meno richiesti vengono rimossi quando lo spazio disponibile si esaurisce.

**BOX 2.4****UN CASO DI STUDIO****L'infrastruttura di rete di Google**

Google, per supportare i suoi numerosi servizi cloud, tra i quali il motore di ricerca, Gmail, il calendario, YouTube, le mappe, editor di documenti e social network, ha installato un'estesa rete privata e un'infrastruttura CDN con tre livelli di cluster di server.

- Diciannove “mega data center” negli Stati Uniti, in Europa e in Asia [Google Locations 2020], ognuno dei quali contiene circa 100.000 server. Sono deputati a offrire contenuti dinamici e spesso personalizzati; tra questi, i risultati delle ricerche e i messaggi Gmail.
- Circa 90 cluster dislocati in IXP di tutto il mondo, ognuno dei quali con centinaia di server [Adhikari 2011a] [Google CDN 2020]. Questi sono deputati a offrire contenuti statici, tra cui i video YouTube.
- Molte centinaia di cluster “enter-deep” sono posizionati in ISP di accesso. In questo caso ogni cluster è costituito da decine di server all'interno di un singolo rack. Eseguono lo splitting TCP (Paragrafo 3.7) e servono contenuti statici [Chen 2011], tra cui le parti statiche di pagine web dei risultati delle ricerche.

Tutti questi data center e cluster sono interconnessi attraverso la rete privata di Google. Ecco quello che sommariamente avviene quando un utente fa un'interrogazione di ricerca: spesso la prima interrogazione viene inviata attraverso l'ISP locale a una cache enter-deep vicina, dalla quale viene recuperato il contenuto statico; la cache, mentre fornisce al client il contenuto statico, inoltre la domanda attraverso la rete privata di Google a uno dei mega data center, dal quale recupera i risultati personalizzati della ricerca.

Un video YouTube può arrivare da una cache bring-home, mentre parti della pagina web relativa al video possono essere prese da una cache enter-deep vicina; la parte di pubblicità arriva dal data center. Riassumendo, a parte l'ISP locale, i servizi cloud di Google sono offerti da un'infrastruttura di rete indipendente dalla Internet pubblica.

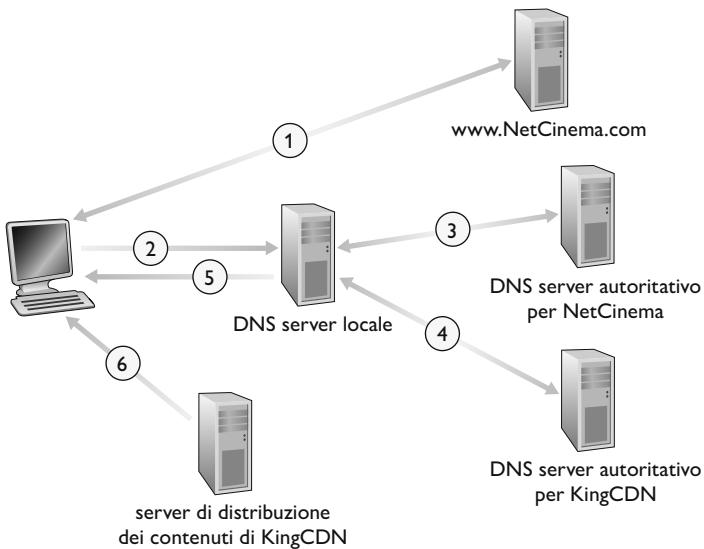
**Come funziona la CDN**

Vediamo ora come funziona una CDN. Quando un browser nell'host di un utente chiede il recupero di uno specifico video identificato da un URL, la CDN deve intercettare la richiesta in modo da poter (1) determinare il cluster di server più appropriato per quel cliente a quell'istante e (2) dirigere la richiesta del client a uno dei server di quel cluster. Esamineremo ora i meccanismi che stanno dietro l'intercettazione e il reindirizzamento di una richiesta, per poi discutere come una CDN determini quale sia il cluster più adatto.

Molte CDN sfruttano il servizio DNS per intercettare e ridirigere le richieste [Vixie 2009]. Consideriamo il seguente esempio: supponete che un fornitore di contenuti, NetCinema, impieghi una CDN di terza parte, KingCDN, per distribuire i suoi video. A ogni video sulla pagina web di NetCinema viene assegnato un URL che include la stringa “video” e un identificatore univoco per tale video; per esempio, al video *Transformers 7* potrebbe essere assegnato <http://video.netcinema.com/6Y7B23V>. Come mostrato nella Figura 2.25, vengono quindi compiuti 6 passi:

1. L'utente visita la pagina web di NetCinema.
2. Quando l'utente seleziona il link <http://video.netcinema.com/6Y7B23V>, il suo host invia una interrogazione DNS a [video.netcinema.com](http://video.netcinema.com).

**Figura 2.25** Il DNS dirige la richiesta di un utente a un server della CDN.



3. Il **DNS server locale (LDNS)** dell’utente invia l’interrogazione DNS a un DNS server autoritativo per NetCinema che osserva la stringa “video” nel nome dell’host `video.netcinema.com`. Il DNS server autoritativo per NetCinema per passare l’interrogazione DNS a KingCDN, invece di restituire un indirizzo IP, restituisce all’LDNS il nome di un host nel dominio di KingCDN quale, per esempio, `a1105.kingcdn.com`.
4. Da questo momento l’interrogazione DNS entra nell’infrastruttura DNS privata di KingCDN. Lo LDNS dell’utente invia quindi una seconda interrogazione, per `a1105.kingcdn.com`, e infine il sistema DNS di KingCDN restituisce gli indirizzi IP di un server di KingCDN all’LDNS. È quindi qui, all’interno del sistema DNS di KingCDN, che viene specificato il server CDN da cui il client riceverà il contenuto.
5. LDNS inoltra l’indirizzo IP del nodo CDN che fornirà il contenuto all’host dell’utente.
6. Il client, una volta ricevuto l’indirizzo IP del server di KingCDN, stabilisce una connessione TCP diretta con il server a quell’indirizzo IP e gli invia una richiesta GET HTTP per il video. Nel caso venga impiegato DASH, il server innanzitutto invierà al client il file manifesto (*manifest file*) con una lista di URL, uno per ogni versione del video, e il client selezionerà in modo dinamico i blocchi da versioni differenti.

### Strategie di selezione dei cluster

Il cuore dell’installazione di una CDN è la **strategia di selezione del cluster** (*cluster selection strategy*): un meccanismo per dirigere dinamicamente i client a un cluster di server o a un data center della CDN. Come appena visto, la CDN apprende l’indirizzo IP del server LDNS del client attraverso la richiesta DNS del client. Dopo aver appreso tale indirizzo IP, la CDN deve selezionare un cluster appropriato basandosi su di esso. Le strategie di selezione del cluster sono

generalmente proprietarie. Descriveremo brevemente alcuni approcci, ognuno con i suoi vantaggi e svantaggi.

Una semplice strategia consiste nell'assegnare a un client il cluster *geograficamente più vicino*. Usando un database commerciale di geo-localizzazione (come Quova [Quova 2020] e Max-Mind [MaxMind 2020]), gli indirizzi IP degli LDNS vengono associati a un luogo geografico. La CDN, quando riceve una richiesta DNS da un particolare LDNS, sceglie il cluster a esso più vicino geograficamente in linea d'aria. Tale soluzione funziona abbastanza bene per una grande percentuale di client [Agarwal 2009]. Tuttavia per alcuni client potrebbe non andare bene, perché il cluster più vicino geograficamente potrebbe essere diverso da quello più vicino dal punto di vista del percorso in rete. Un altro problema proprio di tutti gli approcci basati sul DNS deriva dal fatto che alcuni utenti sono configurati per usare LDNS remoti [Shaikh 2001; Mao 2002]; in questo caso l'LDNS potrebbe trovarsi lontano dal client. Inoltre questa semplice strategia non tiene conto delle variazioni temporali del ritardo di rete e della banda disponibile, assegnando sempre lo stesso cluster a un client.

Un secondo approccio determina il cluster migliore per un client basandosi sulle condizioni di traffico correnti, effettuando **misure in tempo reale** (*real-time measurements*) delle prestazioni di ritardo e perdita tra cluster e client; per esempio, facendo sondare periodicamente dai propri cluster, tramite messaggi ping o interrogazioni DNS, tutti gli LDNS sparsi nel mondo. Un problema di questo approccio è che molti LDNS sono configurati per non rispondere a tali richieste.

## 2.7 Programmazione delle socket: come creare un'applicazione di rete

Dopo aver considerato alcune importanti applicazioni di rete, vediamo ora come i relativi programmi vengono effettivamente scritti. Ricordiamo, dal Paragrafo 2.1, che una tipica applicazione di rete consiste in una coppia di programmi, detti client e server, che risiedono su sistemi differenti. Questi due programmi, quando vengono eseguiti, creano un processo client e un processo server che comunicano “scrivendo in” e “leggendo da” delle socket. Quando si crea una nuova applicazione di rete il compito principale dello sviluppatore è la scrittura del codice per entrambi i programmi client e server.

Esistono due tipi di applicazioni di rete. Un tipo consiste nelle applicazioni che implementano un protocollo standard definito, per esempio, in una RFC, spesso dette “open” perché le regole sono note a tutti. In questo caso, i programmi client e server devono conformarsi alle regole imposte. Per esempio, il programma client potrebbe essere l’implementazione del lato client del protocollo HTTP, descritto nel Paragrafo 2.2 ed esplicitamente definito nell’RFC 2616. Similmente, il programma server potrebbe essere un’implementazione del protocollo server HTTP, anch’esso definito esplicitamente nell’RFC 2616. Se uno sviluppatore scrive codice per il client e un altro fa lo stesso per il server ed entrambi seguono in modo attento le regole dell’RFC, i due programmi saranno in grado di interagire. Infatti, la maggior parte delle odierni applicazioni di rete implica la comunicazione tra programmi client e server creati da svilup-

patori indipendenti. Si consideri, per esempio, un browser Google Chrome che comunica con un web server Apache o un client BitTorrent che comunica con un tracker BitTorrent.

L’altro tipo di applicazioni di rete è costituito da applicazioni proprietarie. In questo caso il protocollo a livello di applicazione usato dai programmi client e server non è tenuto a conformarsi ad alcuna RFC. Un singolo sviluppatore (o un team di sviluppatori) crea tanto i programmi client quanto quelli server e ha completo controllo sul codice. Però, dato che il codice non implementa un protocollo di pubblico dominio, altri sviluppatori indipendenti non saranno in grado di sviluppare codice in grado di interagire con l’applicazione.

In questo paragrafo esaminiamo i problemi chiave nello sviluppo di applicazioni client-server e ci “sporcheremo le mani” con il codice di una semplice applicazione. Durante la fase di sviluppo, una delle prime decisioni da prendere è se l’applicazione debba sfruttare TCP o UDP. Ricordiamo che TCP è orientato alla connessione e fornisce un canale affidabile per il flusso dei byte. UDP è invece senza connessione e invia pacchetti di dati indipendenti da un sistema all’altro, senza garanzie sulla consegna. Un programma client o server, quando implementa un protocollo definito in una RFC, dovrebbe utilizzare il numero di porta noto e associato al servizio. Nello sviluppo di un’applicazione proprietaria, invece, lo sviluppatore deve evitare di utilizzare numeri di porta noti definiti nelle RFC. Abbiamo presentato una breve panoramica dei numeri di porta nel Paragrafo 2.1 e l’argomento verrà ripreso in maggiore dettaglio nel capitolo successivo.

Introduciamo la programmazione delle socket sviluppando due semplice applicazioni UDP e TCP e facendo uso del linguaggio Python 3. Mostreremo le applicazioni scritte in Python perché queste illustrano chiaramente i concetti chiave delle socket, ma avremmo potuto scriverle anche in Java, C o C++. Python consente un minor numero di righe di codice, ciascuna delle quali può essere spiegata al programmatore inesperto senza grandi difficoltà. Non c’è da spaventarsi se non avete familiarità con Python; dovreste essere in grado di seguire facilmente il codice se avete esperienza con gli altri linguaggi di programmazione.

Se siete interessati alla programmazione client-server in Java potete consultare la piattaforma MyLab di questo libro dove troverete molti esempi, inclusi quelli presentati in questo paragrafo. Per i lettori interessati alla programmazione client-server in C esistono diversi interessanti riferimenti bibliografici [Donahoo 2001; Stevens 1997; Frost 1994].

### 2.7.1 Programmazione delle socket con UDP

In questo paragrafo scriveremo dei semplici programmi client-server che usano UDP, nel successivo che usano TCP. Ricordiamo, dal Paragrafo 2.1, che i processi in esecuzione su macchine diverse comunicano inviando messaggi tramite le socket. Abbiamo detto che ciascun processo è analogo a una casa e che la sua socket ne rappresenta la porta. L’applicazione risiede da un lato della porta, il protocollo di trasporto sull’altro lato, quello esterno. Lo sviluppatore ha il controllo di tutto ciò che sta sul lato del livello applicativo della socket, ma ha poco controllo sul lato a livello di trasporto.

Vediamo ora da vicino l'interazione tra due processi comunicanti che usano socket UDP. Il processo di origine, prima di inviare con UDP un pacchetto di dati fuori dalla socket rappresentata dalla porta di casa, deve attaccare l'indirizzo di destinazione al pacchetto. Dopo che il pacchetto è stato inviato attraverso la socket del mittente, Internet userà l'indirizzo di destinazione per instradare il pacchetto verso la socket del processo ricevente. Quando il pacchetto arriva alla socket di destinazione, il processo ricevente recupera il pacchetto attraverso la socket, ne ispeziona il contenuto ed esegue l'azione appropriata.

Potreste chiedervi che cosa venga messo nell'indirizzo di destinazione attaccato al pacchetto. Come potevate aspettarvi, l'indirizzo IP dell'host di destinazione fa parte dell'indirizzo di destinazione. Includendo nel pacchetto l'indirizzo IP di destinazione, i router in Internet saranno in grado di instradare il pacchetto all'host di destinazione. Ma, poiché su un host possono essere in esecuzione più processi applicativi di rete, ognuno con una o più socket, è necessario identificare la specifica socket nell'host di destinazione. Quando viene creata una socket, le si assegna un identificatore, chiamato **numero di porta** (*port number*). Così, come potevate aspettarvi, l'indirizzo di destinazione del pacchetto include anche il numero di porta della socket. Riassumendo, il processo mittente attacca al pacchetto l'indirizzo di destinazione che consiste nell'indirizzo IP dell'host di destinazione e del numero di porta della socket di destinazione. Inoltre, come vedremo presto, viene attaccato al pacchetto anche l'indirizzo sorgente, cioè l'indirizzo del mittente, che consiste nell'indirizzo IP dell'host di origine e del numero di porta della socket di origine. Di questa operazione non devono occuparsi i programmi applicativi, ci penserà automaticamente il sistema operativo.

Per dare una dimostrazione di come si programma con le socket (TCP o UDP) utilizzeremo la seguente semplice applicazione client-server.

1. Un client legge una riga di caratteri (dati) dalla tastiera e la invia tramite la propria socket al server.
2. Il server riceve i dati e converte i caratteri in maiuscole.
3. Il server invia i dati modificati al client.
4. Il client legge i caratteri modificati e li stampa sul proprio schermo.

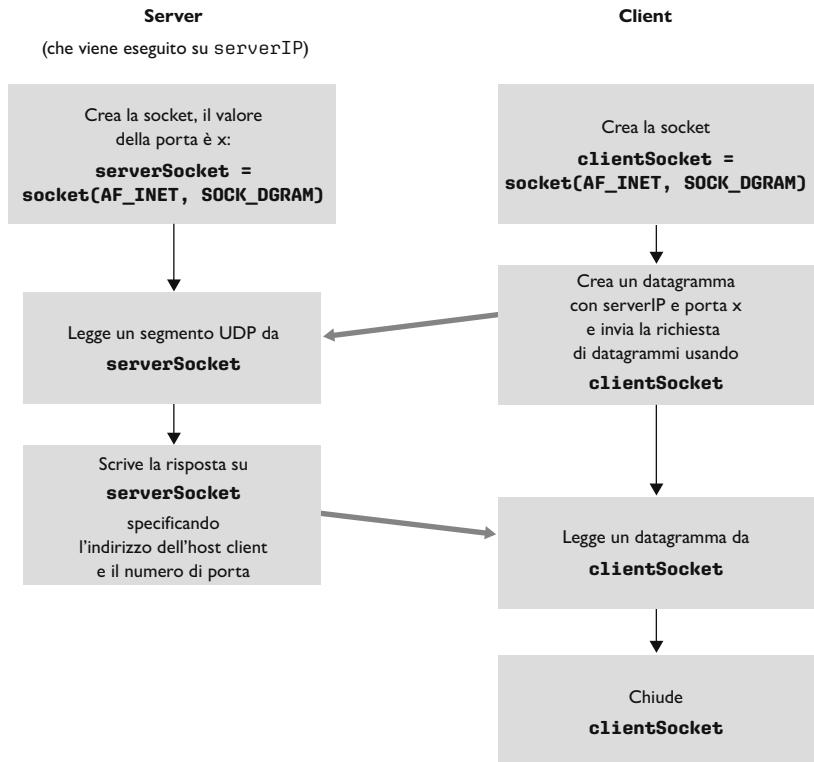
La Figura 2.26 mostra le principali attività relative alle socket del client e del server su UDP.

Sporchiamoci ora le mani andando ad analizzare il codice della coppia di programmi client-server riga per riga. Cominceremo col client UDP che invierà un semplice messaggio a livello di applicazione al server. Il server, per essere in grado di ricevere e rispondere al messaggio del client, deve essere pronto e in esecuzione come processo prima che il client invii il messaggio.

Il programma client è denominato `UDPClient.py`, quello server `UDPServer.py`. Intenzionalmente forniamo un codice minimale che ci permette di evidenziare i punti chiave. Un buon codice avrebbe sicuramente qualche linea ausiliaria in particolare per gestire i casi di errore. Per questa applicazione abbiamo arbitrariamente scelto per il server il numero di porta 12000.

**Figura 2.26**

Applicazione client-server  
con UDP.



### UDPClient.py

Di seguito presentiamo il codice per il lato client dell'applicazione:

```

from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
message = input('Frase in minuscolo:')
clientSocket.sendto(message.encode(),(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage.decode())
clientSocket.close()

```

Vediamo ora le varie linee di codice di UDPClient.py

```
from socket import *
```

Il modulo socket è la base di tutte le comunicazioni di rete in Python. Includendo questa linea, saremo in grado di creare socket nei nostri programmi.

```

serverName = 'hostname'
serverPort = 12000

```

La prima linea assegna alla variabile di tipo stringa `serverName` il nome dell'host. Qui possiamo dare una stringa che contiene o l'indirizzo IP del server

(per esempio, “128.138.32.126”) o l’hostname del server (per esempio, “cis.poly.edu”). Se usiamo l’hostname, verrà effettuata automaticamente una ricerca DNS per avere l’indirizzo IP. La seconda riga assegna alla variabile di tipo intero serverPort il valore 12000.

```
clientSocket = socket(AF_INET, SOCK_DGRAM)
```

Questa linea crea la socket lato client, memorizzandone un riferimento in una variabile chiamata `clientSocket`. Il primo parametro indica la famiglia di indirizzi: in particolare, `AF_INET` indica che la rete sottostante usa IPv4, trattata nel Capitolo 4. Il secondo parametro indica che la socket è di tipo `SOCK_DGRAM`, che significa che è una socket UDP, piuttosto che TCP. Si noti che quando creiamo la socket lato client non ne specifichiamo il numero di porta, lasciamo che lo faccia il sistema operativo per noi. Ora che la porta del processo client è stata creata, vogliamo creare un messaggio da inviare attraverso di essa.

```
message = input('Frase in minuscolo:')
```

In Python `input()` è una funzione interna. Quando questo comando viene eseguito, sul prompt del client compare la stringa utilizzata come parametro. L’utente inserisce da tastiera una linea che viene assegnata alla variabile `message`. Ora che abbiamo la socket e il messaggio vogliamo inviare quest’ultimo attraverso la socket all’host di destinazione.

```
clientSocket.sendto(message.encode(), (serverName, serverPort))
```

In questa linea innanzitutto convertiamo il messaggio da tipo stringa a byte per poterlo inviare nella socket; il metodo `sendto()` attacca l’indirizzo di destinazione (`serverName, serverPort`) al messaggio e invia il pacchetto risultante nella socket `clientSocket`. Ricordiamo che anche l’indirizzo della sorgente è attaccato al pacchetto, ma l’operazione è effettuata automaticamente. Inviare un messaggio da client a server su UDP è proprio semplice! Dopo aver inviato il pacchetto il client aspetta i dati dal server.

```
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
```

Con questa linea, quando un pacchetto arriva da Internet alla socket client, i dati contenuti vengono assegnati alla variabile `modifiedMessage`, mentre l’indirizzo sorgente del pacchetto viene messo nella variabile `serverAddress`. La variabile `serverAddress` contiene sia l’indirizzo IP sia il numero di porta del server. Il programma `UDPClient` in effetti non ha bisogno dell’informazione sull’indirizzo del server che conosce già dall’inizio; tuttavia questa linea lo fornisce. Il metodo `recvfrom` prende inoltre in input la grandezza del buffer (2048).

```
print(modifiedMessage.decode())
```

Questa linea visualizza `modifiedMessage` sul display dell’utente, dopo averlo convertito da byte a stringa. Dovrebbe essere il messaggio originale scritto dall’utente, ma con caratteri maiuscoli.

```
clientSocket.close()
```

Questa linea chiude la socket, il processo termina.

## UDPServer.py

Vediamo ora il lato server dell'applicazione:

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "Il server è pronto a ricevere"
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

Si noti che all'inizio UDPServer è simile a UDPClient. Anche esso importa il modulo socket, assegna 12000 alla variabile intera serverPort e crea una socket UDP di tipo SOCK\_DGRAM (una socket UDP). La prima linea del codice significativamente diversa da UDPClient è:

```
serverSocket.bind(('', serverPort))
```

che assegna alla socket lato server il numero di porta 12000. Quindi in UDPServer il codice scritto dallo sviluppatore assegna esplicitamente il numero di porta alla socket. In questo modo, quando qualcuno invia un pacchetto alla porta 12000 all'indirizzo IP del server, il pacchetto verrà diretto a questa socket. UDPServer entra ora in un ciclo while che gli permette di ricevere ed elaborare pacchetti dai client indefinitamente. All'interno del ciclo UDPServer aspetta che i pacchetti arrivino.

```
message, clientAddress = serverSocket.recvfrom(2048)
```

Questa linea di codice è analoga a quella vista in UDPClient. Quando un pacchetto arriva alla socket lato server i dati vengono assegnati alla variabile message e l'indirizzo sorgente a clientAddress che contiene l'indirizzo IP e il numero di porta del client. In questo caso UDPServer userà l'indirizzo come indirizzo di risposta, come avviene con la posta ordinaria. Con questa informazione sull'indirizzo il server ora sa a chi inviare la risposta.

```
modifiedMessage = message.decode().upper()
```

Questa linea è il cuore della nostra semplice applicazione. Prende la linea inviata dal client e usa il metodo upper() per renderla maiuscola.

```
serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

Quest'ultima linea attacca l'indirizzo del client (indirizzo IP e numero di porta) al messaggio in caratteri maiuscoli e invia il pacchetto risultante tramite la socket lato server. Come detto prima, anche l'indirizzo del server è attaccato al pacchetto, ma ciò avviene in modo automatico e non esplicitamente nel codice. Internet consegnerà il pacchetto all'indirizzo del client. Il server, dopo aver inviato il pacchetto, rimane nel ciclo while aspettando che arrivi un altro pacchetto UDP (da un qualunque client su un qualunque host).

Per provare la coppia di programmi, installate e compilate `UDPClient.py` in un host e `UDPServer.py` in un altro host.<sup>9</sup> Accertatevi di includere il nome o l'IP di server opportuno in `UDPClient.py`. Mandate poi in esecuzione `UDPServer.py` sull'host server; si crea così un processo server che rimane inattivo finché non viene contattato da un client. Mandate ora in esecuzione `UDPClient.py` sull'host client. Questo creerà un processo sul client. Infine, per usare l'applicazione sul client, scrivete una frase e premete Invio.

Per sviluppare una vostra applicazione client-server su UDP potete iniziare a modificare leggermente il programma client o server. Per esempio il server potrebbe, anziché convertire i caratteri in maiuscolo, contare il numero di volte che la lettera `s` appare e restituire tale numero. Oppure potete modificare il client in modo che l'utente, dopo aver ricevuto il messaggio in maiuscolo, possa inviare altre frasi al server.

### 2.7.2 Programmazione delle socket con TCP

Al contrario di UDP, TCP è un protocollo orientato alla connessione. Ciò significa che, prima di iniziare a scambiarsi dati, client e server devono effettuare un'operazione di handshake e stabilire una connessione TCP. Un capo della connessione TCP è attaccato alla socket lato client, l'altro a quella lato server. A una connessione TCP, quando viene creata, viene associato sia l'indirizzo della socket lato client (indirizzo IP e numero di porta) sia l'indirizzo della socket lato server (indirizzo IP e numero di porta). Una volta stabilita la connessione TCP, quando un lato vuole inviare dati all'altro, deve solo mettere i dati nella connessione TCP attraverso la sua socket. Ciò si differenzia da quanto visto prima con UDP, dove il server doveva attaccare l'indirizzo di destinazione al pacchetto prima di inviarlo nella socket.

Diamo uno sguardo più da vicino all'interazione tra i programmi client e server in TCP. Il client ha il compito di dare inizio al contatto con il server. Affinché il server sia in grado di reagire al contatto iniziale del client, esso deve essere pronto. Ciò implica due cose: innanzitutto, come con UDP, il programma server non può essere dormiente, ossia deve essere in esecuzione come processo prima che il client tenti di iniziare il contatto. In secondo luogo, il programma server deve avere una porta speciale, una socket speciale, che dia il benvenuto al contatto iniziale stabilito da un processo client in esecuzione su un qualsiasi host. Utilizzando l'analogia della casa e della porta con il processo e la socket, indicheremo talvolta il contatto iniziale del client con la locuzione “bussare alla porta di benvenuto”.

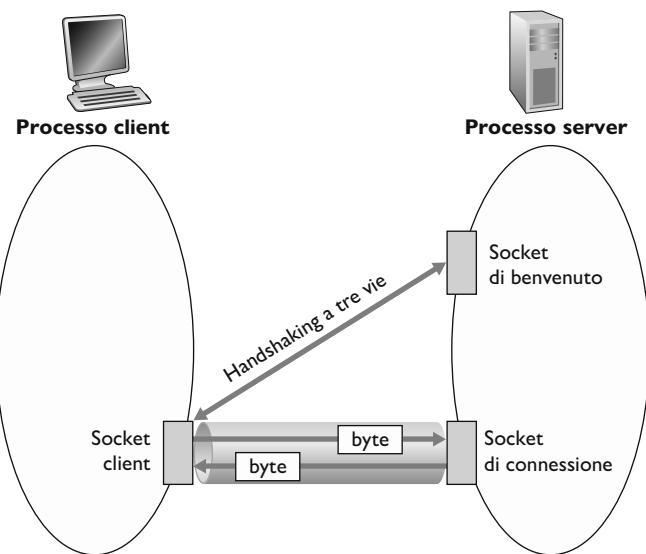
Con il processo server in esecuzione, il processo client può inizializzare una connessione TCP verso il server. Ciò viene fatto nel programma client creando una socket. Al momento della costituzione, il client specifica l'indirizzo della socket di benvenuto sul server, ossia l'indirizzo IP dell'host server e il numero di porta del relativo processo. Dopo la creazione della socket nel programma client, TCP avvia un handshake a tre vie e stabilisce una connessione TCP con

---

<sup>9</sup> In realtà la cosa funziona anche se i due programmi vengono eseguiti sullo stesso host in finestre diverse (N.d.R.).

**Figura 2.27**

Un processo server TCP ha due socket.

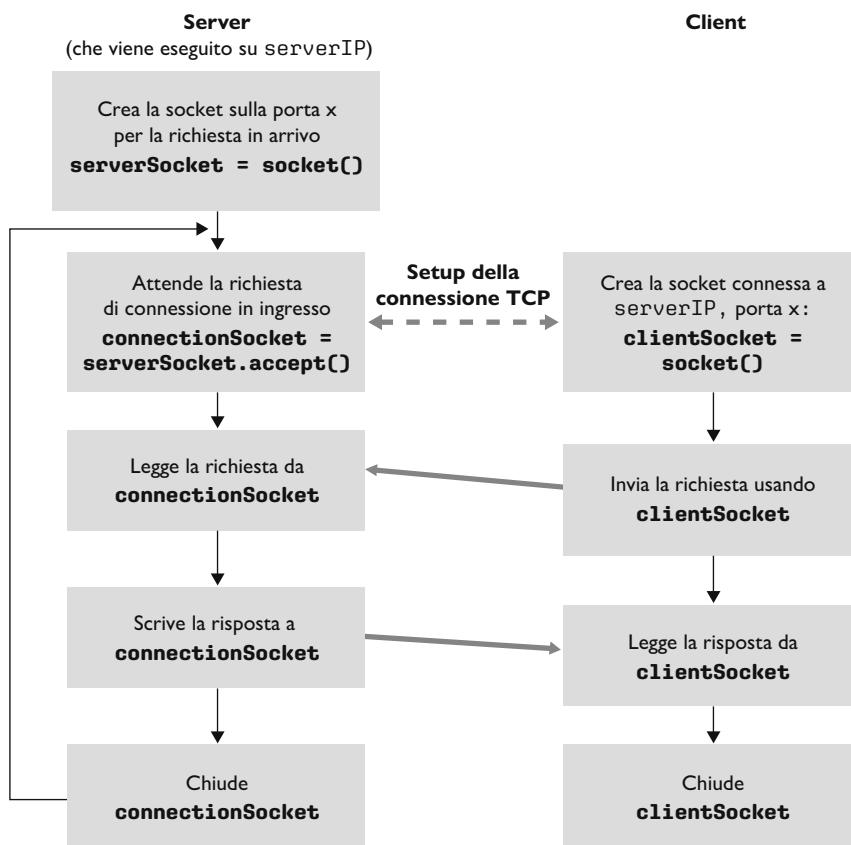


il server. L'handshake ha luogo a livello di trasporto ed è completamente trasparente ai programmi client e server.

Durante l'handshake a tre vie il client bussa alla porta di benvenuto del processo server. Quando il server “sente bussare” crea una nuova porta (più precisamente, una nuova socket) dedicata a quel particolare client. Nel prossimo esempio, la porta di benvenuto è un oggetto che rappresenta una socket TCP e che chiameremo `ServerSocket`. La socket appena creata dedicata al client che fa la connessione sarà chiamata `connectionSocket`. Spesso chi affronta per la prima volta le socket TCP confonde la socket di benvenuto (che è il punto iniziale di contatto di tutti i client che vogliono comunicare col server) e tutte le socket di connessione create via via dal lato server per comunicare con ogni client.

Dal punto di vista dell'applicazione, la connessione TCP è un condotto virtuale diretto tra la socket del client e la socket di connessione del server. Come mostrato nella Figura 2.27, il processo client può inviare quanti byte vuole nella sua socket e TCP garantisce che il processo server (attraverso la socket di connessione) riceverà tutti i byte nell'ordine di invio. Pertanto, TCP fornisce un servizio affidabile tra client e server. Inoltre, così come le persone possono entrare e uscire dalla stessa porta, il processo client invia e riceve byte tramite la socket; altrettanto avviene per il processo server, attraverso la propria socket di connessione.

Per dare una dimostrazione di come si programma con socket TCP usiamo la stessa applicazione client-server vista con UDP. Il client invia una riga di dati al server, che li trasforma in maiuscole e rimanda al client. La Figura 2.28 mette in evidenza le principali attività delle socket lato client e server che comunicano per mezzo di un servizio di trasporto TCP.



### TCPClient.py

Ecco il codice per il lato client dell'applicazione:

```

from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
sentence = input('Frase in minuscolo:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print('Dal server:', modifiedSentence.decode())
clientSocket.close()

```

Analizziamo le righe di codice che differiscono in modo significativo da quelle dell'implementazione con UDP. La prima riguarda la creazione della socket lato client:

```
clientSocket = socket(AF_INET, SOCK_STREAM)
```

chiamata `clientSocket`. Il primo parametro indica come prima che la rete sottostante usa IPv4. Il secondo indica che la socket è di tipo `SOCK_STREAM`, che

significa che è una socket TCP e non UDP. Si noti che, anche qui, non specifichiamo il numero di porta della socket lato client quando la creiamo; lasciamo che lo faccia il sistema operativo. La successiva linea di codice è invece molto diversa da quella UDP:

```
clientSocket.connect([serverName, serverPort])
```

Ricordiamo che prima che il client possa inviare dati al server, e viceversa, usando una socket TCP è necessario stabilire una connessione TCP tra i due. Questa linea inizializza una connessione TCP tra client e server. Il parametro del metodo `connect()` è l'indirizzo del lato server della connessione. Dopo l'esecuzione di questa linea di codice, viene effettuata la procedura di handshake a tre vie e viene stabilita una connessione tra client e server.

```
sentence = input('Frase in minuscolo:')
```

Come in `UDPclient`, questa riga prende una frase dall'utente. La stringa `sentence` continua a raccogliere caratteri finché l'utente non termina la linea con un ritorno a capo.

Anche la linea seguente è molto diversa da `UDPclient`:

```
clientSocket.send(sentence.encode())
```

che invia la stringa `sentence` attraverso la socket del client alla connessione TCP. Si noti che il programma non crea esplicitamente un pacchetto attaccandoci l'indirizzo di destinazione come nelle socket UDP. In TCP il programma client semplicemente lascia cadere i byte della stringa `sentence` nella connessione TCP. A questo punto il client si mette in attesa di ricevere byte dal server.

```
modifiedSentence = clientSocket.recv(2048)
```

I caratteri arrivati dal server vengono assegnati alla stringa `modifiedSentence`, dove continuano ad accumularsi fino al carattere di ritorno a capo. Dopo aver visualizzato la frase in caratteri maiuscoli, si chiude la socket lato client con l'istruzione:

```
clientSocket.close()
```

Tale linea chiude la socket e quindi la connessione TCP tra client e server.

### **TCPServer.py**

Trattiamo ora il programma server.

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen(1)
print ('Il server è pronto a ricevere')
while True:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.encode())
    connectionSocket.close()
```

Analizziamo ora le linee di codice che differiscono in modo significativo da quelle di UDPServer e TCPClient. Come in TCPClient, il server crea una socket TCP con:

```
serverSocket=socket(AF_INET,SOCK_STREAM)
```

Analogamente a UDPServer associamo il numero di porta sul server serverPort a questa socket:

```
serverSocket.bind(('',serverPort))
```

Ma, con TCP, serverSocket è la nostra socket di benvenuto. Dopo aver stabilito questa porta di ingresso, dovremo metterci in ascolto di qualche client che vi bussi.

```
serverSocket.listen(1)
```

Questa linea fa in modo che il server possa usare la socket per ascoltare sulla connessione TCP le richieste di connessione da parte dei client. Il parametro specifica il numero massimo di connessioni da tenere in coda.

```
connectionSocket, addr = serverSocket.accept()
```

Il metodo accept() blocca temporaneamente l'esecuzione del programma finché il client non bussa alla porta di benvenuto. Nel riprendere l'esecuzione viene creata una nuova socket nel server, chiamata connectionSocket e dedicata allo specifico client che ha bussato. Il client e il server a questo punto completano la procedura di handshaking, creando una connessione TCP tra clientSocket e connectionSocket. Stabilita la connessione TCP, il client e il server possono scambiarsi byte. Con TCP, tutti i byte hanno garanzia non solo di arrivare a destinazione, ma anche di arrivarci nell'ordine corretto.

```
connectionSocket.close()
```

In questo programma, dopo aver inviato la frase modificata al client, chiudiamo la connessione. Ma, poiché serverSocket rimane aperta, un altro client può bussare alla porta e inviare al server una frase da modificare.

Con questo abbiamo terminato la nostra trattazione della programmazione di socket TCP. Siete invitati a provare a eseguire i due programmi su host diversi e a modificarli per ottenere delle applicazioni anche solo leggermente diverse. Dovreste inoltre provare a paragonare la coppia di programmi UDP e quella TCP per vederne le differenze. Dovreste anche esercitarvi nella programmazione delle socket con gli Esercizi proposti alla fine dei Capitoli 2 e 4. Infine, speriamo che un giorno, dopo aver padroneggiato questi e altri, più avanzati, programmi con le socket, scriviate la vostra applicazione di rete che diventi popolare e vi renda ricchi e famosi, e che vi ricordiate degli autori di questo libro!

## 2.8 Riepilogo

In questo capitolo abbiamo studiato gli aspetti concettuali e implementativi delle applicazioni di rete. Siamo entrati in contatto con l'onnipresente architettura client-server adottata da molte applicazioni per Internet e abbiamo visto il suo uso nei protocolli HTTP, SMTP e DNS. Abbiamo studiato questi importanti protocolli a livello di applicazione e le loro corrispettive applicazioni (Web, tra-

sferimento file, e-mail e DNS). Abbiamo, inoltre, studiato l’architettura P2P e l’abbiamo paragonata all’architettura client-server. Abbiamo trattato lo streaming video e come i moderni sistemi di distribuzione di video sfruttino le CDN. Abbiamo esaminato come le socket possano essere utilizzate per costruire applicazioni di rete. Ci siamo avventurati nell’uso delle socket per i servizi di trasporto end-to-end orientati alla connessione (TCP) e non (UDP). Il nostro primo passo nel viaggio attraverso l’architettura stratificata della rete è ora completo.

Proprio all’inizio del libro (Paragrafo 1.1) abbiamo presentato una definizione di protocollo piuttosto vaga e scarna: “Il formato e l’ordine dei messaggi scambiati tra due o più entità comunicanti, così come le azioni intraprese alla trasmissione e/o alla ricezione di un messaggio o di un altro evento”. Il materiale presentato nel corso del capitolo, e in particolare lo studio dettagliato di HTTP, SMTP e DNS, hanno ora aggiunto parecchia sostanza a questa definizione. I protocolli rappresentano un concetto chiave nel mondo delle reti; il nostro studio ci ha dato l’opportunità di sviluppare una maggiore coscienza in merito a quello di cui i protocolli si occupano.

Nel Paragrafo 2.1 abbiamo descritto i modelli di servizio offerti da TCP e UDP alle applicazioni che li invocano. Abbiamo analizzato dettagliatamente questi modelli di servizio quando abbiamo sviluppato semplici applicazioni che fanno uso di TCP e UDP (Paragrafo 2.7). Tuttavia abbiamo detto poco sul modo in cui i due protocolli forniscono tali modelli di servizio. Per esempio, non ci siamo occupati di come TCP garantisca alle proprie applicazioni un servizio di trasferimento affidabile per i dati. Nel prossimo capitolo esamineremo con attenzione non solo che cosa fanno i protocolli di trasporto, ma anche come e perché lo fanno.

Forti delle nostre conoscenze sulla struttura delle applicazioni di Internet e sui protocolli a livello applicativo, siamo ora pronti a scendere ulteriormente lungo la pila dei protocolli e a esaminare il livello di trasporto nel corso del Capitolo 3.

## Domande e problemi

---

### Domande di revisione

#### PARAGRAFO 2.1

- R1.** Elencate cinque applicazioni di Internet non proprietarie e i protocolli a livello di applicazione che utilizzano.
- R2.** Qual è la differenza tra l’architettura di una rete e l’architettura di un’applicazione?
- R3.** In una sessione di comunicazione tra una coppia di processi, chi è il client e chi il server?
- R4.** Per le applicazioni P2P, siete d’accordo con l’affermazione “Non esiste la nozione di lato client e server in una sessione di comunicazione”? Perché?
- R5.** Quali informazioni usa un processo in esecuzione su un host per identificare un processo di un altro host?
- R6.** Supponete di voler fare una transazione da un client remoto a un server il più velocemente possibile. Usereste TCP o UDP? Perché?

- R7.** In riferimento alla Figura 2.4, vediamo che nessuna delle applicazioni elencate ha requisiti sulla perdita di dati o sulla temporizzazione. Riuscite a immaginare un’applicazione che richieda di non perdere alcun dato e che sia contemporaneamente molto sensibile alle tempistiche?
- R8.** Elencate le quattro grandi classi di servizi che può fornire un protocollo di trasporto e per ciascuna indicate se TCP o UDP o entrambi forniscono quel tipo di servizio.
- R9.** Ricordiamo che TCP può essere arricchito con TLS per fornire un servizio di sicurezza tra processi, compresa la cifratura. TLS funziona a livello di trasporto o applicativo? Se uno sviluppatore vuole che TCP sia arricchito con TLS, che cosa deve fare?

## PARAGRAFI 2.2 - 2.4

- R10.** Che cosa si intende per protocollo che fa uso di handshaking?
- R11.** Perché HTTP, SMTP e IMAP fanno uso di TCP e non di UDP?
- R12.** Prendete in considerazione un sito web di commercio elettronico che vuole mantenere traccia degli acquisti di ciascun cliente. Descrivete come sia possibile farlo utilizzando i cookie.
- R13.** Descrivete il modo in cui il web caching riduce il ritardo di ricezione di un oggetto richiesto. Questo si verificherà per tutti gli oggetti richiesti o solo per alcuni? Perché?
- R14.** Collegatevi con telnet a un web server e inviate un messaggio di richiesta su più righe. Includete nel messaggio di richiesta la riga di intestazione `If-modified-since`: per ottenere un messaggio di risposta con il codice di stato 304 `Not Modified`.
- R15.** Elencate alcune applicazioni di messaggistica. Utilizzano lo stesso protocollo di SMS?
- R16.** Supponete che Alice, con un account di posta elettronica basato sul Web (quale Hotmail o Gmail) invii un messaggio a Bob che accede alla propria casella sul proprio server usando IMAP. Descrivete come il messaggio giunge dall'host di Alice a quello di Bob. Assicuratevi di elencare la serie di protocolli a livello di applicazione usati per trasferire il messaggio tra i due host.
- R17.** Stampate l'intestazione di un messaggio di posta elettronica che avete ricevuto di recente. Quante righe di intestazione `Received`: sono presenti? Analizzatele una per una.
- R18.** Qual è il problema del blocco HOL (Head of Line blocking) in HTTP/1.1? Come tenta di risolverlo HTTP/2?
- R19.** È possibile che il web server e il mail server di un'azienda abbiano esattamente lo stesso sinonimo di un hostname (per esempio, `foo.com`)? Quale sarebbe il tipo di RR che contiene l'hostname del server di posta?
- R20.** Si consideri un'e-mail inviata da un utente con indirizzo e-mail che termina con `.edu`. Sarebbe possibile determin-

nare dall'intestazione l'indirizzo IP dell'host da cui è stata inviata l'e-mail? E se il messaggio fosse stato inviato da un account Gmail?

## PARAGRAFO 2.5

- R21.** Supponete che Alice fornisca in BitTorrent chunk di un file a Bob durante tutto un intervallo di 30 secondi. Bob ricambierà necessariamente il favore inviando ad Alice dei chunk nello stesso intervallo? Spiegate perché.
- R22.** Considerate un nuovo peer, Alice, che entra a far parte di BitTorrent senza aver nessun chunk. Senza chunk, non può diventare uno dei quattro peer unchoked di qualcuno degli altri peer, in quanto non ha nulla da inviare. Come fa Alice a ottenere il suo primo chunk?
- R23.** Che cos'è una rete di overlay? Include i router? Quali sono i collegamenti della rete di overlay?

## PARAGRAFO 2.6

- R24.** Le CDN adottano in generale due diversi approcci per la dislocazione dei server. Elencateli e descriveteli brevemente.
- R25.** Oltre alle considerazioni legate alla rete, come ritardi, perdite e prestazioni di banda, ci sono molti altri fattori importanti che intervengono nella scelta della strategia di selezione dei cluster. Quali sono?

## PARAGRAFO 2.7

- R26.** Il server UDP, descritto nel Paragrafo 2.7, richiedeva una sola socket, mentre il server TCP ne richiedeva due. Perché? Se il server TCP dovesse supportare  $n$  connessioni contemporanee, ciascuna proveniente da un diverso client, di quante socket avrebbe bisogno?
- R27.** Nell'applicazione client/server su TCP descritta nel Paragrafo 2.7, perché il programma server deve essere mandato in esecuzione prima del client? E, nel caso dell'applicazione client/server su UDP, perché il programma client può essere mandato in esecuzione prima del server?

## Problemi

### P1. Vero o falso?

- Quando un utente richiede una pagina web costituita da testo e tre immagini, il client invia un messaggio di richiesta e riceverà quattro messaggi di risposta.
- Due pagine web distinte (per esempio, `www.mit.edu/research.html` e `www.mit.edu/students.html`) possono essere inviate sulla stessa connessione persistente.
- Con connessioni non persistenti tra browser e server di origine, un singolo segmento TCP può portare due diversi messaggi di richiesta HTTP.

- L'intestazione `Date`: nel messaggio di risposta HTTP indica quando è stato modificato per l'ultima volta l'oggetto nella risposta.
- I messaggi di risposta HTTP non possono avere il corpo del messaggio vuoto.

- SMS, iMessage, Wechat e WhatsApp sono sistemi di messaggistica istantanea per dispositivi mobili. Effettuate una ricerca sui protocolli che utilizzano e le loro differenze.
- Considerate un client HTTP che voglia recuperare un documento web da un determinato URL e che l'indirizzo IP del server HTTP non sia noto inizialmente. In

questo scenario, oltre a HTTP, quali protocolli a livello di trasporto e di applicazione sono richiesti?

- P4.** Considerate la seguente stringa di caratteri ASCII catturata da Wireshark quando il browser ha inviato un messaggio HTTP GET (cioè questo è l'effettivo contenuto del messaggio HTTP GET). I caratteri <cr> e <lf> sono il carattere di ritorno a capo e nuova riga (cioè, i caratteri corsivi della stringa <cr> nel testo seguente rappresentano un singolo carattere di ritorno a capo che è contenuto in quel punto nell'intestazione HTTP). Rispondete alle seguenti domande, indicando dove, nel messaggio HTTP GET seguente, trovate le risposte.

```
GET /cs453/index.html HTTP/1.1<cr><lf>Host: gai
a.cs.umass.edu<cr><lf>User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.2) Gecko/20040804 Netscape/7.2 (ax) <cr><lf>Accept: text/xml, application/xml, application/xhtml+xml, text/html;q=0.9, text/plain;q=0.8, image/png,*/*;q=0.5
<cr><lf>Accept-Language: en-us,en;q=0.5<cr><lf>Accept-Encoding: zip,deflate<cr><lf>Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7<cr><lf>Keep-Alive: 300<cr><lf>Connection:keep-alive<cr><lf><cr><lf>
```

- (a) Qual è la URL del documento richiesto dal browser?
- (b) Qual è la versione di HTTP che sta usando il browser?
- (c) Il browser richiede una connessione persistente o non persistente?
- (d) Qual è l'indirizzo IP dell'host sul quale è in esecuzione il browser?
- (e) Che tipo di browser invia il messaggio? Perché è necessario che in un messaggio di richiesta HTTP ci sia il tipo di browser?

- P5.** Il seguente testo mostra la risposta manda dal server dopo aver ricevuto il messaggio HTTP GET della domanda precedente. Rispondete alle seguenti domande, indicando dove, nel messaggio HTTP GET sottostante, trovate le risposte.

```
HTTP/1.1 200 OK<cr><lf>Date: Tue, 07 Mar 2008
12:39:45GMT<cr><lf>Server: Apache/2.0.52 (Fedora)
<cr><lf>Last-Modified: Sat, 10 Dec2005 18:27:46
GMT<cr><lf>ETag: "526c3-f22-a88a4c80"<cr><lf>Accept-
Ranges: bytes<cr><lf>Content-Length: 3874<cr><lf>
Keep-Alive: timeout=max=100<cr><lf>Connection:
Keep-Alive<cr><lf>Content-Type: text/html; charset=
ISO-8859-1<cr><lf><cr><lf><!doctype html public "-//w3c//dtd html 4.0 transitional//en"><lf><html><lf>
<head><lf><meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1"><lf><meta
name="GENERATOR" content="Mozilla/4.79 [en] (Windows NT
5.0; U) Netscape]"><lf><title>CMPSCI 453 / 591 /
NTU-ST550A Spring 2005 homepage</title><lf></head><lf>
<much more document text following here (not shown)>
```

- (a) È stato capace il server di trovare il documento? In quale istante il documento di risposta è stato fornito?
- (b) Quando è stato modificato l'ultima volta il documento?

- (c) Quanti byte ci sono nel documento inviato al client?

- (d) Che cosa sono i primi 5 byte del documento inviato al client? Il server ha accettato la connessione persistente?

- P6.** Recuperate le specifiche di HTTP/1.1 (RFC 2616) e rispondete alle seguenti domande:

- (a) Qual è il meccanismo di segnalazione usato tra client e server per indicare che una connessione persistente sta per essere chiusa? Tale chiusura può essere segnalata dal client, dal server o da entrambi?
- (b) Quali servizi di crittografia sono forniti da HTTP?
- (c) Può un client aprire simultaneamente tre o più connessioni con un dato server?
- (d) Si supponga che su una connessione client-server uno dei due rimanga inattivo per un lasso di tempo. Può uno dei due chiudere la connessione di trasporto? È possibile che un lato inizi a chiudere la connessione mentre l'altro sta trasmettendo? Perché?

- P7.** Supponete di selezionare, all'interno del browser, un collegamento ipertestuale per ottenere una pagina web. L'indirizzo IP dell'URL relativo non si trova nella cache del vostro host locale, e dunque è necessaria una ricerca DNS per ottenerlo. Supponiamo di visitare  $n$  DNS server prima di ricevere l'indirizzo IP dal DNS. Le visite successive incorrono in un RTT di  $RTT_1, \dots, RTT_n$ . Supponete inoltre che la pagina web associata al collegamento contenga un solo oggetto consistente in una piccola quantità di testo HTML. Chiamiamo  $RTT_0$  il valore di RTT tra l'host locale e il server contenente l'oggetto. Supponendo nullo il tempo di trasmissione dell'oggetto, quanto tempo intercorre tra l'attimo del click sul collegamento e la ricezione dell'oggetto presso il client?

- P8.** In riferimento al Problema 7, supponete che il file HTML referenzi otto oggetti molto piccoli sullo stesso server. Ignorando il tempo di trasmissione, quanto tempo intercorre con

- (a) HTTP non persistente e senza connessioni TCP parallele?
- (b) HTTP non persistente e il browser configurato per gestire 6 connessioni TCP parallele?
- (c) HTTP persistente?

- P9.** Considerate la Figura 2.12, in cui si vede la rete di un'istituzione connessa a Internet. Supponete che la dimensione media degli oggetti sia 1000000 bit e che la frequenza media di richieste dai browser dell'istituzione verso i server di origine sia di 16 richieste al secondo. Ipotizzate, inoltre, che la quantità di tempo che intercorre da quando il router sul lato Internet del collegamento di accesso inoltra una richiesta HTTP a quando riceve la risposta sia mediamente di tre secondi (Paragrafo 2.2.5). Fate un modello del tempo medio totale di risposta come la somma del ritardo medio di accesso (ossia, il ritardo dal router Internet al router dell'istituzione) e del ritardo Internet medio. Per il ritardo medio di accesso, usate la formula  $\Delta/(1 - \Delta\beta)$ , ove  $\Delta$  è il tempo medio richiesto per inviare un oggetto sul colle-

gamento di accesso e  $\beta$  è la frequenza di arrivo di oggetti al collegamento di accesso.

- (a) Trovate il tempo medio totale di risposta.
- (b) Supponete ora che nella LAN dell'istituzione sia installato un proxy e che la frequenza con cui gli oggetti non sono in cache (*miss rate*) sia 0,4. Trovate il tempo totale di risposta.

**P10.** Considerate un breve collegamento di 10 metri, sul quale il mittente può trasmettere a una frequenza di 150 bps in entrambe le direzioni. Supponete che i pacchetti contenenti dati siano lunghi 100.000 bit e quelli contenenti solo controllo (per esempio ACK o handshake) siano lunghi 200 bit. Assumete che vi siano  $N$  connessioni parallele e ciascuna occupi  $1/N$  della banda del collegamento. Considerate ora il protocollo HTTP e supponete che ciascun oggetto scaricato sia lungo 100 kbit e che l'oggetto iniziale scaricato contenga 10 oggetti referenziati dallo stesso mittente. Ha senso lo scaricamento parallelo tramite istanze parallele non persistenti di HTTP? Considerate ora HTTP persistente. Vi aspettate un guadagno significativo rispetto al caso non persistente? Giustificate e spiegate la vostra risposta.

**P11.** Nello scenario del problema precedente supponete che il collegamento sia condiviso da Bob con altri quattro utenti. Bob usa istanze parallele di HTTP non persistenti, mentre gli altri quattro utenti usano HTTP non persistente senza parallelismo.

- (a) Le connessioni parallele di Bob lo aiutano a scaricare più velocemente le pagine web?
- (b) Se tutti e cinque gli utenti usassero cinque connessioni parallele, Bob ne beneficerebbe ancora? Motivate la risposta.

**P12.** Scrivete un semplice programma TCP per un server che accetta righe in ingresso da un client e le stampa sul proprio terminale. Potete farlo modificando il programma TCPServer.py del testo. Eseguite il vostro programma e, su una qualsiasi altra macchina, impostate il server proxy nel browser perché punti all'host dove è in esecuzione il vostro programma server; configurate inoltre il numero di porta in modo appropriato. Il vostro browser dovrebbe ora inviare i propri messaggi di richiesta GET al vostro server, e quest'ultimo dovrebbe mostrarli sul suo terminale. Usate questa configurazione per determinare se il vostro browser genera messaggi di GET condizionale per gli oggetti che si trovano nella cache locale.

**P13.** Si consideri l'invio su HTTP/2 di una pagina web composta da un video clip e cinque immagini. Supponiamo che il video clip venga trasportato in 2000 fotogrammi e che ogni immagine abbia tre fotogrammi.

- a. Se tutti i fotogrammi video vengono inoltrati per primi senza interleaving, quanti "frame time" sono necessari prima che tutte e cinque le immagini vengano inviate?
- b. Se i frame sono intervallati, quanti "frame time" sono necessari prima che tutte e cinque le immagini vengano inviate?

**P14.** Si consideri la pagina web nel Problema 13 e si utilizzi la priorità HTTP/2. Supponiamo che a tutte le immagini

venga data priorità sul video clip e che la prima immagine abbia priorità sulla seconda immagine, la seconda immagine sulla terza immagine e così via. Quanti frame time sono necessari prima che la seconda immagine venga inviata?

**P15.** Qual è la differenza tra MAIL FROM: in SMTP e From: nel messaggio di posta elettronica stesso?

**P16.** Come si indica la fine del corpo del messaggio in SMTP? E in HTTP? Può HTTP usare lo stesso metodo di SMTP? Perché?

**P17.** Leggete l'RFC 5321 di SMTP. Che cosa significa MTA? Si consideri la ricezione della seguente e-mail di spam, che è una modifica di una vera. Si supponga che solo l'host che ha originato la mail sia malintenzionato e tutti gli altri siano onesti. Si identifichi l'host che ha generato questa e-mail di spam:

```
From - Fri Nov 07 13:41:30 2008
Return-Path: <tennis5@pp33head.com>
Received: from barmail.cs.umass.edu
[barmail.cs.umass.edu [128.119.240.3]] by cs.umass.edu
(8.13.1/8.12.6) for <hg@cs.umass.edu>; Fri, 7 Nov 2008
13:27:10 -0500
Received: from asusus-4b96 ([localhost [127.0.0.1]]) by
barmail.cs.umass.edu (Spam Firewall) for
<hg@cs.umass.edu>; Fri, 7 Nov 2008 13:27:07 -0500
(EST)
Received: from asusus-4b96 ([58.88.21.177]) by
barmail.cs.umass.edu for <hg@cs.umass.edu>; Fri,
07 Nov 2008 13:27:07 -0500 (EST)
Received: from [58.88.21.177] by
inbnd55.exchangedd.com; Sat, 8 Nov 2008 01:27:07 +0700
From: "Jonny" <tennis5@pp33head.com>
To: <hg@cs.umass.edu>
Subject: How to secure your savings
```

- P18.** (a) Che cos'è un database *whois*?
- (b) Utilizzate più database whois in Internet per ottenere i nomi di due DNS server.
- (c) Utilizzate nslookup sul vostro host locale per inviare query DNS a tre DNS server: il vostro DNS server locale e i due DNS server che avete trovato nella parte (b). Provate query di tipo A, NS e MX e riassumete i risultati.
- (d) Utilizzare nslookup per trovare un web server che abbia più indirizzi IP. Il server della vostra organizzazione ha più indirizzi IP?
- (e) Utilizzate il database whois ARIN per determinare il range di indirizzi IP della vostra università.
- (f) Descrivete come si possa utilizzare i database whois e lo strumento nslookup per effettuare un'esplorazione del vostro istituto prima di lanciare un attacco.
- (g) Discutete perché i database whois dovrebbero essere disponibili pubblicamente.

**P19.** In questo problema usiamo l'utile tool dig disponibile su Unix e Linux per esplorare la gerarchia dei DNS server. Si ricordi che, come descritto nella Figura 2.19, un DNS server che si trova a un livello più alto della gerarchia delega a un server a un livello più in basso una ri-

- chiesta DNS, inviando indietro al client DNS il suo nome. Dopo aver letto le istruzioni di dig rispondete alle seguenti domande:
- Partendo da un root server, iniziate una sequenza di richieste dell'indirizzo IP del web server del vostro dipartimento usando dig. Mostrate la lista dei nomi dei DNS server coinvolti nella catena di deleghe utilizzata per rispondere alla vostra richiesta.
  - Ripetete il punto a. per siti web popolari come google.com, yahoo.com e amazon.com.
- P20.** Supponete di poter accedere alla cache del DNS server locale del vostro dipartimento. Sapreste indicare un modo per determinare sommariamente quali sono i web server esterni più popolari tra gli utenti del vostro dipartimento? Spiegate tale metodo.
- P21.** Supponete che il vostro dipartimento abbia un DNS server locale per tutti i computer del dipartimento. Voi siete un utente ordinario, non un amministratore di rete o di sistema. Potreste determinare se un sito web esterno è stato probabilmente visitato un paio di secondi fa da un computer del dipartimento? Come?
- P22.** Considerate la distribuzione di un file di dimensione  $F = 20$  Gbit a  $N$  peer. Il server ha una velocità di upload di  $u_s = 30$  Mbps e ciascun peer ha una banda di download di  $d_i = 2$  Mbps e di upload di  $u$ . Per  $N = 10, 100$  e  $1000$  e  $u = 300$  kbps, 700 kbps e 2 Mbps, preparate un grafico che rappresenti per entrambi i metodi di distribuzione, client-server e P2P, il tempo di distribuzione minimo per ciascuna combinazione di  $N$  e  $u$ .
- P23.** Considerate la distribuzione di un file di  $F$  bit a  $N$  peer, usando un'architettura client-server. Assumete una situazione in cui il server può trasmettere contemporaneamente a più peer, trasmettendo a ciascuno con velocità diverse, finché le velocità combinate non superano  $u_s$ .
- Supponete che  $u_s/N \leq d_{\min}$ . Specificate uno schema di distribuzione che abbia un tempo di distribuzione di  $NF/u_s$ .
  - Supponete che  $u_s/N \geq d_{\min}$ . Specificate uno schema di distribuzione che abbia un tempo di distribuzione di  $F/d_{\min}$ .
  - Concludete che il tempo di distribuzione minimo è in generale dato da  $\max[NF/u_s, F/d_{\min}]$ .
- P24.** Considerate la distribuzione di un file di  $F$  bit a  $N$  peer, usando un'architettura P2P. Fate la stessa assunzione del problema precedente riguardo al server. Per semplificità assumete che  $d_{\min}$  sia molto grande, in modo che la banda di download dei peer non sia un collo di bottiglia.
- Supponete che  $u_s \leq (u_s + u_1 + \dots + u_N)/N$ . Specificate uno schema di distribuzione che abbia un tempo di distribuzione di  $F/u_s$ .
  - Supponete che  $u_s \geq (u_s + u_1 + \dots + u_N)/N$ . Specificate uno schema di distribuzione che abbia un tempo di distribuzione di  $NF/(u_s + u_1 + \dots + u_N)$ .
  - Concludete che il tempo di distribuzione minimo è in generale dato dal  $\max[F/u_s, NF/(u_s + u_1 + \dots + u_N)]$ .
- P25.** Supponete di vedere attivi  $N$  peer in una rete di overlay, e che ogni coppia di essi presenti una connessione TCP attiva. Inoltre, ipotizzate che le connessioni TCP passino attraverso  $M$  router. Quanti nodi e collegamenti costituiscono la rete di overlay?
- P26.** Supponete che Bob si unisca a un torrent di BitTorrent, ma che non voglia fare alcun upload di dati (comportamento detto *free-riding*).
- Bob sostiene di poter ricevere una copia completa del file condiviso dal torrent. Bob ha ragione? Perché?
  - Bob sostiene inoltre che può rendere più efficiente la sua procedura usando l'insieme dei computer del suo laboratorio nel dipartimento, ognuno con indirizzo IP diverso. Come può farlo?
- P27.** Considerate un sistema DASH con  $N$  versioni video, a  $N$  bit rate e qualità diversi, e  $N$  versioni audio, a  $N$  bit rate e qualità diversi. Supponete di voler dare la possibilità all'utente di scegliere in ogni istante quale delle  $N$  versioni video e audio voglia.
- Se i file che creiamo mescolano audio e video, in modo che il server invii solo uno stream in un dato istante, quanti file deve memorizzare il server (ognuno a un URL diverso)?
  - Se invece il server invia separatamente gli stream audio e video e il client li sincronizza, quanti file deve memorizzare il server?
- P28.** Installate e compilate i programmi Phyton TCPClient e UDPClient su un host e TCPServer e UDPServer su un altro.
- Supponete di lanciare TCPClient prima di TCPServer. Che cosa succede e perché?
  - Ipotizzate che lanciate UDPClient prima di UDPServer. Che cosa succede e perché?
  - Che cosa avviene se usate numeri di porta diversi per il lato client e il lato server?
- P29.** Immaginate di aggiungere in `UDPClient.py`, dopo aver creato la socket la seguente linea di codice:
- ```
clientSocket.bind(('', 5432));
```
- Sarà necessario cambiare `UDPServer.py`? Quali sono i numeri di porta delle socket in `UDPClient` e `UDPServer`? Quali erano prima di operare il cambioamento?
- P30.** Potete configurare il vostro browser in modo che apra connessioni multiple simultanee verso un sito web? Quali sono i vantaggi e quali gli svantaggi di avere connessioni multiple simultanee?
- P31.** Abbiamo visto che le socket TCP trattano i dati in trasmissione come un flusso di byte, ma che le socket UDP riconoscono l'inizio e la fine dei singoli messaggi. Enunciate un vantaggio e uno svantaggio di avere API orientate ai byte rispetto a quelle che riconoscono e conservano i confini dei messaggi definiti dall'applicazione.
- P32.** Che cos'è il web server Apache? Quanto costa? Quali funzionalità ha attualmente? Se lo ritenevate opportuno potete consultare Wikipedia per rispondere.

## Esercizi di programmazione delle socket

La piattaforma MyLab di questo libro contiene sei esercizi di programmazione delle socket. I primi quattro sono spiegati di seguito. Il quinto usa il protocollo ICMP e verrà spiegato alla fine del Capitolo 5. È fortemente consigliato che gli studenti completino alcuni, se non tutti, gli esercizi. Trovate tutte le informazioni e alcuni frammenti di codice Python sulla piattaforma MyLab del volume.

### Esercizio 1: Web server

In questo esercizio userete Python per sviluppare un web server in grado di rispondere a una richiesta sola. Nello specifico il vostro web server dovrà: (i) creare una socket di connessione quando contattato dal client (browser), (ii) ricevere la richiesta HTTP su questa connessione, (iii) analizzare la richiesta per determinare il file da inviare, (iv) prendere il file richiesto dal file system del server, (v) creare un messaggio di risposta HTTP che consiste del file richiesto preceduto dalle linee di intestazione e (vi) inviare la risposta sulla connessione TCP al browser richiedente. Se il browser chiede un file non esistente nel server, il server deve restituire il messaggio di errore ‘404 Not Found’. La piattaforma MyLab del libro fornisce uno scheletro del codice per il vostro server che voi dovete completare, eseguite il vostro server e provate a inviare richieste da differenti host. Se eseguite il vostro server su un host che ha già in esecuzione un web server dovete usare un numero di porta diverso da 80.

### Esercizio 2: Ping UDP

In questo esercizio implemeterete un semplice client ping basato su UDP in Python. Il vostro client spedirà un messaggio di ping al server, riceverà indietro il messaggio di pong e quindi determinerà il tempo di andata e ritorno (RTT) tra l'invio del ping e il ricevimento del pong. Le funzionalità fornite da questi programmi sono simili a quelle del comando ping disponibile negli attuali sistemi operativi. Il comando ping standard fa uso di un protocollo chiamato *Internet Control Message Protocol* (ICMP), che studieremo nel Capitolo 5. In questo esercizio implemeteremo un ping non standard (ma semplice!) a livello applicativo usando socket UDP.

Il vostro programma ping deve inviare 10 messaggi ping a un server bersaglio usando UDP. Per ogni messaggio il client deve determinare l'RTT e visualizzarlo quando il messaggio pong arriva. Poiché UDP non è affidabile, uno dei due messaggi potrebbe andare perso, motivo per cui il client non può aspettare all'infinito. Il vostro client dovrebbe aspettare la risposta dal server un secondo e quindi assumere che il messaggio sia stato perso e visualizzare un messaggio opportuno.

Troverete sulla piattaforma MyLab del libro il programma server completo. Dovete implementare il client dopo aver studiato con attenzione il codice del programma server, che sarà molto simile. Potete riutilizzare liberamente il codice del server facendo copia e incolla.

### Esercizio 3: Client di posta

In questo esercizio svilupperete un semplice user agent che invii e-mail a un destinatario. Il vostro client dovrà creare una connessione TCP tra il client di posta elettronica e il mail server (per esempio quello di Google), inviargli comandi SMTP per inviare una e-mail (a un vostro amico, per esempio) e infine chiudere la connessione TCP col server. La piattaforma MyLab del libro fornisce lo scheletro del codice per il vostro client che voi dovete completare e provare inviando e-mail a differenti account. Potete anche provare a inviare messaggi tramite mail server diversi (per esempio quello di Google e quello della vostra università).

### Esercizio 4: Web proxy

In questo esercizio svilupperete un proxy server. Tale server riceverà un messaggio HTTP da un browser, genererà una nuova richiesta HTTP per lo stesso oggetto e la in-

vierà al server destinatario. Quando riceverà da quest’ultimo il messaggio di risposta HTTP, creerà una nuova risposta HTTP con l’oggetto incluso e la inoltrerà al client. Potete trovare lo scheletro del programma sulla piattaforma MyLab del libro. Completatelo e provatelo con diversi browser che facciano richieste di oggetti web tramite il proxy.

## Esercitazione Wireshark: HTTP

Dopo aver rotto il ghiaccio con lo sniffer di pacchetti della prima esercitazione, siamo ora pronti a usare Wireshark per investigare i protocolli in azione. In questa esercitazione studieremo molti aspetti del protocollo HTTP: l’interazione tra GET e risposta, i formati dei messaggi HTTP, il recupero di grandi file HTML, il recupero di file HTML con URL all’interno, le connessioni persistenti e non persistenti nonché l’autenticazione e la sicurezza in HTTP.

Come per tutte le esercitazioni Wireshark è possibile trovare la descrizione completa (in inglese) sulla piattaforma MyLab del libro.

## Esercitazione Wireshark: DNS

Qui consideriamo più da vicino il lato client del DNS, il protocollo che traduce gli hostnames di Internet in indirizzi IP. Ricordiamo, dal Paragrafo 2.5, che il ruolo del client nel DNS è relativamente semplice: inviare una richiesta al proprio DNS server locale e attendere la risposta. Avviene molto “sotto coperta”, invisibile ai client DNS, dato che i server gerarchici DNS comunicano reciprocamente per risolvere la richiesta DNS del client o in modo ricorsivo o in modo iterativo. Dal punto di vista del client DNS, comunque, il protocollo è piuttosto semplice: viene inviata una richiesta verso il DNS server locale e si riceve una risposta da parte di tale server. In questa esercitazione osserviamo il DNS in azione. La descrizione completa (in lingua inglese) dell’esercitazione si trova sulla piattaforma MyLab del volume.

# Livello di trasporto

Posto tra il livello di applicazione e quello di rete, il livello di trasporto costituisce una parte centrale dell'architettura stratificata delle reti e riveste la funzione critica di fornire servizi di comunicazione direttamente ai processi applicativi in esecuzione su host differenti. L'approccio che adottiamo in questo capitolo alterna la trattazione dei principi alla base del livello di trasporto alla discussione su come questi vengano implementati nei protocolli esistenti; come sempre, si darà particolare enfasi ai protocolli di Internet, soprattutto a TCP e UDP.

Inizieremo affrontando la relazione che intercorre tra i livelli di trasporto e di rete. Ciò porta all'esame della prima funzione critica del livello di trasporto: estendere il servizio tra due sistemi periferici, proprio del livello di rete, a un servizio di trasporto tra processi a livello di applicazione in esecuzione su host diversi. Illustreremo tale funzione nella nostra trattazione di UDP: il protocollo di trasporto di Internet non orientato alla connessione.

Torneremo poi ai principi di base per studiare uno dei principali problemi del networking: come due entità siano in grado di comunicare in modo affidabile attraverso un mezzo che può smarrire e alterare i dati. Tramite una serie di scenari sempre più complessi (e realistici) definiremo le tecniche adottate dai protocolli a livello di trasporto per risolvere il problema. Mostreremo poi come questi principi trovino applicazione in TCP, il protocollo di trasporto di Internet orientato alla connessione.

Ci concentreremo poi su un secondo fondamentale problema: il controllo della velocità di trasmissione (*transmission rate*) delle entità a livello di trasporto al fine di evitare o risolvere una congestione nella rete. Prenderemo in esame cause e conseguenze della congestione, così come le tecniche comunemente adottate per il suo controllo. Passeremo quindi all'approccio TCP.

### 3.1 Introduzione e servizi a livello di trasporto

Nei due capitoli precedenti abbiamo considerato il ruolo del livello di trasporto e i servizi che esso fornisce ai livelli limitrofi. Rivediamo brevemente ciò che abbiamo già imparato. Un protocollo a livello di trasporto mette a disposizione una **comunicazione logica** (*logical communication*) tra processi applicativi di host differenti.

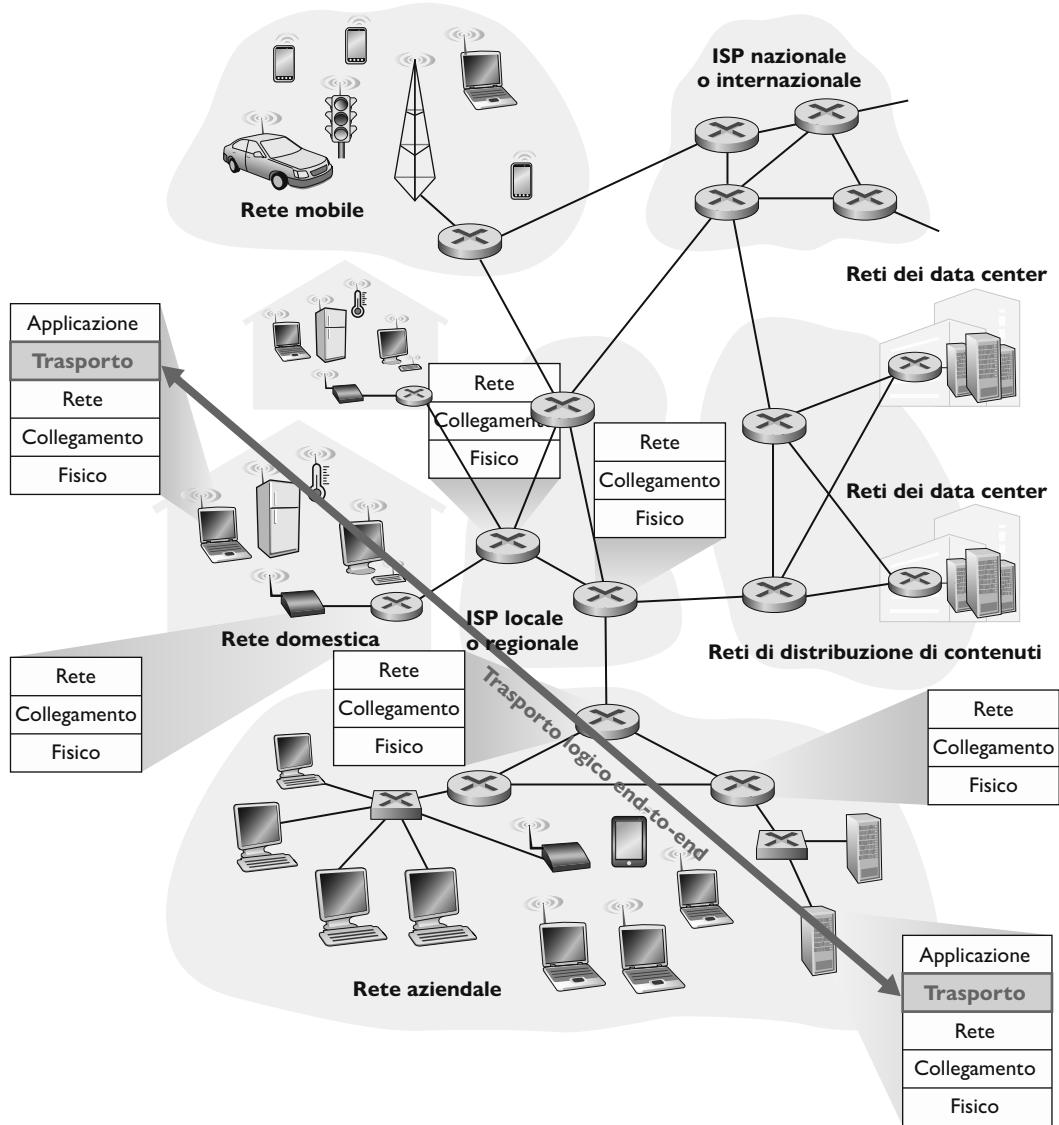
Per comunicazione logica si intende, dal punto di vista dell'applicazione, che tutto proceda come se gli host che eseguono i *processi* fossero direttamente connessi; in realtà, gli host si possono trovare agli antipodi del pianeta, connessi da numerosi router e da svariati tipi di collegamenti. I processi applicativi usano la comunicazione logica fornita dal livello di trasporto per scambiare messaggi, senza preoccuparsi dei dettagli dell'infrastruttura fisica utilizzata per trasportarli. La Figura 3.1 illustra il concetto di comunicazione logica.

Come si vede nella Figura 3.1, i protocolli a livello di trasporto sono implementati nei sistemi periferici, ma non nei router della rete. Lato mittente, il livello di trasporto converte i messaggi che riceve da un processo applicativo in pacchetti a livello di trasporto, noti secondo la terminologia Internet come **segmenti** (*transport-layer segment*). Questo avviene spezzando (se necessario) i messaggi applicativi in parti più piccole e aggiungendo a ciascuna di esse un'intestazione di trasporto per creare un segmento. Il livello di trasporto, quindi, passa il segmento al livello di rete, dove viene incapsulato all'interno di un pacchetto a livello di rete (datagramma) e inviato a destinazione. È importante notare che i router intermedi agiscono solo sui campi a livello di rete del datagramma, senza esaminare i campi del segmento incapsulato al suo interno. Lato ricevente, il livello di rete estrae il segmento dal datagramma e lo passa al livello superiore, quello di trasporto. Quest'ultimo elabora il segmento ricevuto, rendendo disponibili all'applicazione destinataria i dati del segmento. È possibile mettere a disposizione delle applicazioni di rete più di un protocollo a livello di trasporto. Internet, per esempio, ne possiede due, TCP e UDP, che forniscono servizi diversi alle applicazioni che ne fanno uso.

#### 3.1.1 Relazione tra i livelli di trasporto e di rete

Ricordiamo che il livello di trasporto è collocato esattamente sopra quello di rete nella pila di protocolli. Mentre un protocollo a livello di trasporto mette a disposizione una comunicazione logica tra processi che vengono eseguiti su host diversi, un protocollo a livello di rete fornisce comunicazione logica tra host. La distinzione è sottile, ma importante. Esaminiamo questa differenza con l'aiuto di un'analogia.

Consideriamo due condomini, uno a Milano e l'altro a Roma, e supponiamo che in ciascuno risieda una dozzina di ragazzi. Quelli di Milano sono cugini di quelli che vivono a Roma e a tutti piace molto scriversi delle lettere, per cui ogni ragazzo scrive ai cugini nell'altra città ogni settimana e le lettere sono recapitate tramite posta in buste separate. Pertanto, ogni settimana, ciascun gruppo di ragazzi invia all'altro 144 lettere (di certo i genitori risparmierebbero un sacco di soldi se usassero la posta elettronica). Andrea, a Milano, e Anna, a Roma, hanno l'incarico di raccogliere e distribuire la posta. Tutte le settimane Anna raccoglie la posta dai suoi fratelli e sorelle e provvede a imbucarla. Anna



**Figura 3.1** Il livello di trasporto fornisce comunicazione logica anziché fisica tra processi applicativi.

ha anche il compito di distribuire le lettere in arrivo, Andrea effettua le stesse operazioni a Milano.

In questo esempio il servizio postale fornisce comunicazione logica tra le due case: ossia trasferisce la posta da casa a casa, non da persona a persona. A loro volta Anna e Andrea forniscono comunicazione logica tra i cugini: infatti raccolgono la posta e la consegnano a chi di dovere. Si noti che, dal punto di vista dei cugini, Anna e Andrea costituiscono il servizio postale, anche se ne rappresentano solo una parte (quella periferica) nell'ambito del processo di consegna end-to-end. Questo esempio fornisce un'interessante analogia per spiegare come il livello di trasporto si relaziona con il livello di rete:

messaggi dell'applicazione = lettere nelle buste  
 processi = cugini  
 host (sistemi periferici) = condomini  
 protocollo a livello di trasporto = Anna e Andrea  
 protocollo a livello di rete = servizio postale (compresi i postini)

Proseguendo con questa analogia, si noti che Anna e Andrea svolgono il proprio lavoro localmente e non sono coinvolti nello smistamento della posta negli uffici postali intermedi o nel trasporto da un ufficio postale a un altro. Analogamente, i protocolli a livello di trasporto risiedono nei sistemi periferici. All'interno di un sistema periferico, un protocollo di trasporto trasferisce i messaggi dai processi applicativi al bordo della rete (ossia, al livello di rete) e viceversa, ma non fornisce alcuna indicazione su come i messaggi siano trasferiti all'interno della rete. Infatti, come mostrato nella Figura 3.1, i router intermedi non riconoscono né operano su alcuna informazione che il livello di trasporto possa aver aggiunto ai messaggi delle applicazioni.

Continuando con la saga familiare, supponiamo ora che, quando Anna e Andrea sono in vacanza, un'altra coppia di cugini (diciamo Susanna ed Enrico) li sostituiscano nella raccolta e consegna interna della posta. Sfortunatamente, Susanna ed Enrico non effettuano il loro compito con la stessa attenzione di Anna e Andrea. Essendo più giovani e inesperti, raccolgono e distribuiscono la posta con frequenza minore e, occasionalmente, smarriscono alcune lettere (che vengono, talvolta, mordicchiate dal cane di famiglia). Di conseguenza, Susanna ed Enrico non mettono a disposizione lo stesso insieme di servizi (ossia, lo stesso modello di servizio) di Anna e Andrea. In modo analogo, una rete di calcolatori può rendere disponibili più protocolli di trasporto, ciascuno dei quali può offrire alle applicazioni un modello di servizio differente.

I servizi forniti da Anna e Andrea sono chiaramente vincolati dai possibili servizi messi a disposizione dal sistema postale. Se questo, per esempio, non prevede un limite massimo di tempo per il recapito della posta tra le due case (supponiamo, tre giorni), allora non c'è modo per Anna e Andrea di garantire un limite al ritardo per la consegna delle lettere. Analogamente i servizi che un protocollo di trasporto può offrire sono sovente vincolati al modello di servizio del protocollo sottostante a livello di rete. Se quest'ultimo non può fornire garanzie sul ritardo o sulla banda per i segmenti a livello di trasporto scambiati tra host, il protocollo a livello di trasporto non può certo offrire garanzie sul ritardo o sulla banda per i messaggi applicativi inviati tra processi.

Ciò nondimeno, determinati servizi possono essere garantiti da un protocollo di trasporto anche se il sottostante protocollo di rete non offre un servizio corrispondente. Per esempio, come vedremo nel corso di questo capitolo, un protocollo di trasporto può offrire il servizio di trasferimento dati affidabile alle applicazioni anche quando il sottostante protocollo di rete non è affidabile, ossia anche quando il protocollo di rete smarrisce, altera o duplica i pacchetti. Considerando un altro aspetto (che studieremo dettagliatamente nel Capitolo 8, disponibile in inglese sulla piattaforma MyLab, quando tratteremo la sicurezza di rete), un protocollo di trasporto può usare la cifratura (*encryption*) per garantire che i messaggi delle applicazioni non vengano letti da intrusi, anche

quando il livello di rete non può garantire la riservatezza dei segmenti a livello di trasporto.

### 3.1.2 Panoramica del livello di trasporto di Internet

Internet, e più in generale una rete TCP/IP, mette a disposizione del livello di applicazione due diversi protocolli. Uno è **UDP** (*User Datagram Protocol*), che fornisce alle applicazioni un servizio non affidabile e non orientato alla connessione, l'altro è **TCP** (*Transmission Control Protocol*), che offre un servizio affidabile e orientato alla connessione. Nel progettare applicazioni di rete, lo sviluppatore deve scegliere uno di questi due protocolli. Come abbiamo visto nel Paragrafo 2.7, lo sviluppatore delle applicazioni sceglie tra UDP e TCP quando crea le socket.

Per rendere un po' più semplice la terminologia nel contesto di Internet chiameremo *segmento* il pacchetto a livello di trasporto. Ricordiamo, tuttavia, che la letteratura su Internet (per esempio, le RFC) definisce segmento un pacchetto a livello di trasporto per TCP, ma molte volte definisce datagramma un pacchetto per UDP. In altri casi, viene usato il termine *datagramma* anche per il pacchetto a livello di rete. In un testo introduttivo come questo crediamo risulti più chiaro riferirsi ai pacchetti TCP e UDP chiamandoli entrambi segmenti e riservare il termine datagramma ai pacchetti a livello di rete.

Prima di procedere nella nostra introduzione su UDP e TCP risulta utile soffermarci brevemente sul livello di rete di Internet (esaminato in dettaglio nel Capitolo 4 e nel Capitolo 5). Il protocollo a livello di rete di Internet ha un nome: IP (*Internet Protocol*). Tale protocollo fornisce comunicazione logica tra host. Il suo modello di servizio prende il nome di **best-effort delivery service** o, più comunemente, **best-effort** (“massimo sforzo”): questo significa che IP fa “del suo meglio” per consegnare i segmenti tra host comunicanti, *ma non offre garanzie*. In particolare, non assicura né la consegna dei segmenti né il rispetto dell’ordine originario e non garantisce neppure l’integrità dei dati all’interno dei segmenti. Per queste ragioni si dice che IP offre un **servizio non affidabile** (*unreliable service*). Ricordiamo anche che ciascun host presenta quantomeno un indirizzo a livello di rete: il suo indirizzo IP. Nel Capitolo 4 esamineremo l’indirizzamento IP nel dettaglio; per ora occorre solo tenere a mente che *ciascun host possiede un indirizzo IP*.

A questo punto diamo uno sguardo ai modelli di servizio offerti da UDP e TCP, il cui principale compito è l’estensione del servizio di consegna di IP “tra sistemi periferici” a quello di consegna “tra processi in esecuzione sui sistemi periferici”. Questo passaggio, da consegna host-to-host a consegna process-to-process, viene detto **multiplexing e demultiplexing a livello di trasporto** (*transport layer multiplexing/demultiplexing*) e verrà trattato nel prossimo paragrafo. UDP e TCP forniscono, inoltre, un controllo di integrità includendo campi per il riconoscimento di errori nelle intestazioni dei propri segmenti. Questi due servizi minimi a livello di trasporto – la consegna di dati da processo a processo e il controllo degli errori – sono gli unici offerti da UDP. In particolare, esattamente come IP, UDP costituisce un servizio inaffidabile: non garantisce che i dati inviati da un processo arrivino intatti (e neppure che arrivino) al processo destinatario (Paragrafo 3.3).

TCP, d'altra parte, offre alle applicazioni diversi servizi aggiuntivi. Innanzitutto, fornisce un **trasferimento dati affidabile** (*reliable data transfer*). Grazie al controllo di flusso, ai numeri di sequenza, agli acknowledgment e ai timer, tecniche che esploreremo in dettaglio in questo capitolo, assicura che i dati vengano trasferiti da un processo a un altro in modo corretto e ordinato. Pertanto TCP converte il servizio inaffidabile tra sistemi periferici, tipico di IP, in un servizio affidabile di trasporto dati tra processi. TCP fornisce anche il **controllo di congestione** (*congestion control*), un servizio offerto alle applicazioni e a Internet nel suo complesso. In pratica, evita che le connessioni TCP intasino i collegamenti e i router tra gli host comunicanti con un'eccessiva quantità di traffico. In linea di principio, TCP permette alle proprie connessioni di attraversare un collegamento di rete congestionato in modo da condividere equamente la larghezza di banda del collegamento stesso. Questo risultato viene raggiunto regolando la velocità (*rate*) alla quale il lato mittente della connessione TCP immette traffico in rete. Il traffico UDP, al contrario, non viene regolato. Le applicazioni che usano UDP possono spedire a qualsiasi velocità desiderata e per tutto il tempo voluto.

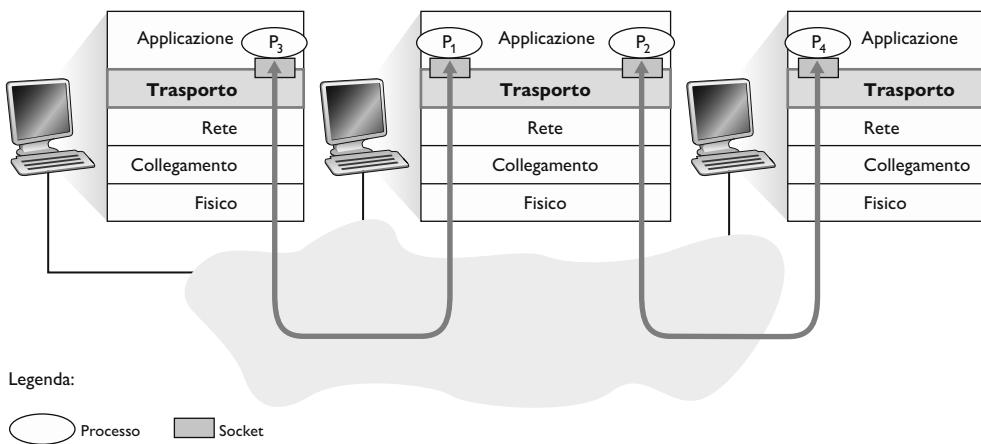
Un protocollo che offre trasferimento dati affidabile e controllo di congestione deve essere necessariamente complesso. Ci serviranno svariate pagine per spiegarne i principi e alcune parti aggiuntive per trattare il protocollo TCP stesso (dal Paragrafo 3.4 al Paragrafo 3.7). L'approccio adottato in questo capitolo è quello di alternare i principi base al protocollo TCP. Innanzitutto esporremo il trasferimento affidabile di dati in uno scenario generale per poi indicare come TCP lo metta a disposizione. Analogamente, tratteremo prima il controllo di congestione in termini generali per poi vedere come viene effettuato da TCP. Diamo tuttavia prima uno sguardo al multiplexing e al demultiplexing a livello di trasporto.

## 3.2 Multiplexing e demultiplexing

In questo paragrafo analizzeremo il multiplexing e il demultiplexing, cioè come il servizio di trasporto da host a host fornito dal livello di rete possa diventare un servizio di trasporto da processo a processo per le applicazioni in esecuzione sugli host. Per praticità tratteremo questo servizio di base del livello di trasporto nel contesto di Internet, anche se il servizio di multiplexing e demultiplexing è presente in tutte le reti di calcolatori.

Nell'host destinatario il livello di trasporto riceve segmenti dal livello di rete immediatamente sottostante. Il livello di trasporto ha il compito di consegnare i dati di questi segmenti al processo applicativo appropriato in esecuzione nell'host. Consideriamo un esempio. Supponiamo che vi troviate di fronte al vostro computer e che stiate scaricando pagine web mentre sono in esecuzione una sessione FTP e due sessioni Telnet. Avrete pertanto quattro processi applicativi in esecuzione: due Telnet, uno FTP e uno HTTP. Il livello di trasporto nel vostro calcolatore, quando riceve dati dal livello di rete sottostante, li deve indirizzare a uno di questi quattro processi. Esaminiamo come ciò venga realizzato.

Innanzitutto ricordiamo (Paragrafo 2.7) che un processo (come parte di un'applicazione di rete) può gestire una o più **socket**, attraverso le quali i dati fluiscano dalla rete al processo e viceversa. Di conseguenza (Figura 3.2) il livello



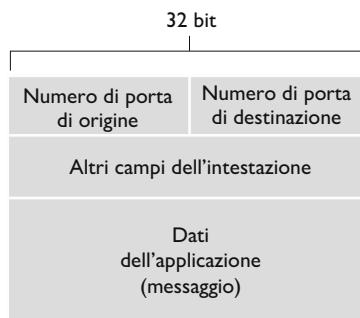
**Figura 3.2** Multiplexing e demultiplexing a livello di trasporto.

di trasporto nell'host di ricezione in realtà non trasferisce i dati direttamente a un processo, ma piuttosto a una socket che fa da intermediario. Siccome, a ogni dato istante, può esserci più di una socket nell'host di ricezione, ciascuna avrà un identificatore univoco il cui formato dipende dal fatto che si tratti di socket UDP o TCP, come vedremo tra breve.

Consideriamo ora come l'host in ricezione indirizzi verso la socket appropriata il segmento a livello di trasporto in arrivo. Ciascun segmento a livello di trasporto ha vari campi deputati allo scopo. Lato ricevente, il livello di trasporto esamina questi campi per identificare la socket di ricezione e quindi vi dirige il segmento. Il compito di trasportare i dati dei segmenti a livello di trasporto verso la giusta socket viene detto **demultiplexing**. Il compito di radunare frammenti di dati da diverse socket sull'host di origine e incapsularne ognuno con intestazioni a livello di trasporto (che verranno poi utilizzate per il demultiplexing) per creare dei segmenti e passarli al livello di rete, viene detto **multiplexing**. Si noti che il livello di trasporto nell'host centrale della Figura 3.2 deve effettuare il demultiplexing dal livello di rete di segmenti che possono arrivare sia per il processo P<sub>1</sub> sia per P<sub>2</sub>; ciò avviene indirizzando i dati del segmento in ingresso alla giusta socket. Il livello di trasporto nell'host centrale deve, inoltre, raccogliere i dati in uscita dalle socket dei due processi, creare i segmenti a livello di trasporto e passarli al livello di rete. Sebbene abbiamo introdotto il multiplexing e il demultiplexing nel contesto dei protocolli di trasporto Internet, è importante rendersi conto che queste funzioni hanno uno specifico interesse ogni volta che un protocollo a un qualsiasi livello (di trasporto o qualsiasi altro) è utilizzato da più entità al livello immediatamente superiore.

Per mostrare il compito del multiplexing e del demultiplexing richiamiamo l'analogia del paragrafo precedente. Anna effettua un'operazione di multiplexing quando raccoglie le lettere dai mittenti e le imbuca. Nel momento in cui Andrea riceve le lettere dal postino, effettua un'operazione di demultiplexing leggendo il nome riportato sopra la busta e consegnando ciascuna missiva al rispettivo destinatario.

**Figura 3.3** I campi del numero di porta di origine e di destinazione nei segmenti a livello di trasporto.



Ora che abbiamo compreso i ruoli del multiplexing e del demultiplexing a livello di trasporto esaminiamo come vengono realizzati negli host. Da quanto visto in precedenza, sappiamo che il multiplexing a livello di trasporto richiede (1) che le socket abbiano identificatori unici e (2) che ciascun segmento presenti campi che indichino la socket cui va consegnato il segmento. Questi (Figura 3.3) sono il **campo del numero di porta di origine** (*source port number field*) e il **campo del numero di porta di destinazione** (*destination port number field*). I segmenti UDP e TCP presentano anche altri campi, come vedremo più avanti. I numeri di porta sono di 16 bit e vanno da 0 a 65535, quelli che vanno da 0 a 1023 sono chiamati **numeri di porta noti** (*well-known port number*) e sono riservati per essere usati da protocolli applicativi ben noti quali HTTP (porta 80) e FTP (porta 21). L'elenco dei numeri di porta noti è fornito nell'RFC 1700: la sua versione aggiornata è consultabile tramite <http://www.iana.org> [RFC 3232]. Quando si sviluppa una nuova applicazione, è necessario assegnarle un numero di porta.

Ora dovrebbe essere chiaro come il livello di trasporto possa implementare il servizio di demultiplexing: ogni socket nell'host deve avere un numero di porta e, quando un segmento arriva all'host, il livello di trasporto esamina il numero della porta di destinazione e dirige il segmento verso la socket corrispondente. I dati del segmento passano, quindi, dalla socket al processo assegnato. Come vedremo, questo è fondamentalmente il modo in cui agisce UDP, mentre il multiplexing/demultiplexing TCP è ancora più raffinato.

### Multiplexing e demultiplexing non orientati alla connessione

Ricordiamo, dal Paragrafo 2.7.1, che i programmi Python in esecuzione possono creare una socket UDP con l'istruzione

```
clientSocket = socket(AF_INET, SOCK_DGRAM)
```

Quando una socket UDP viene definita in questo modo, il livello di trasporto le assegna automaticamente un numero di porta compreso tra 1024 e 65535 che non sia ancora stato utilizzato. In alternativa, un programma Python potrebbe creare una socket UDP associata a uno specifico numero di porta (per esempio, 19157) con il metodo **bind()**:

```
clientSocket.bind(('', 19157))
```

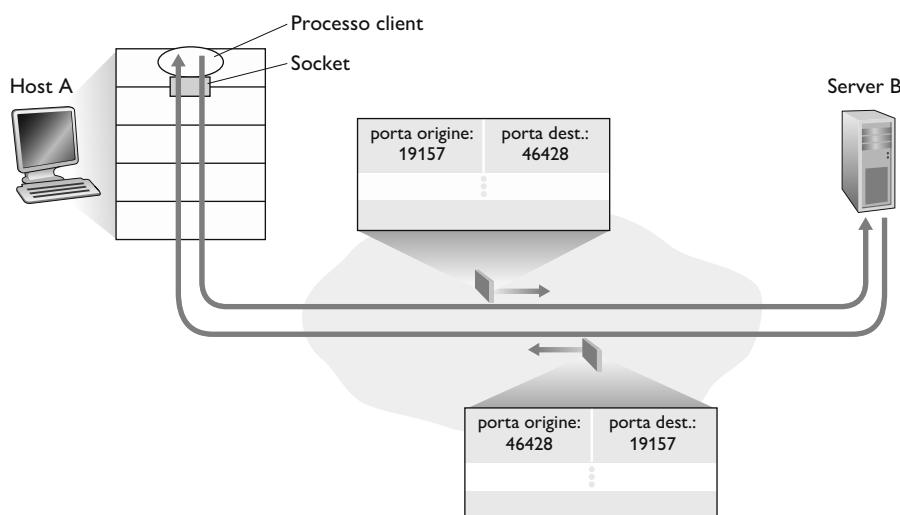
Se il programmatore stesse implementando il lato server di un “protocollo noto”, dovrebbe assegnargli il corrispondente numero di porta. Generalmente, il lato client dell'applicazione consente al livello di trasporto l'assegnazione au-

tomatica (e trasparente) del numero di porta, mentre il lato server dell'applicazione assegna un numero di porta specifico.

Ora che alle socket UDP sono stati assegnati i numeri di porta, possiamo descrivere in modo preciso il multiplexing/demultiplexing UDP. Supponiamo che un processo nell'Host A, con porta UDP 19157, voglia inviare un blocco di dati applicativi a un processo con porta UDP 46428 nell'Host B. Il livello di trasporto di A crea un segmento che include i dati applicativi, i numeri di porta di origine (19157) e di destinazione (46428) e due altri valori (che non sono importanti per l'attuale discussione e verranno trattati più avanti). Il livello di trasporto passa, quindi, il segmento risultante al livello di rete, che lo incapsula in un datagramma IP, ed effettua un tentativo best-effort di consegna del segmento all'host in ricezione. Se il segmento arriva all'Host B, il suo livello di trasporto esamina il numero di porta di destinazione del segmento (46428) e lo consegna alla propria socket identificata da 46428. Osserviamo che l'Host B potrebbe avere in esecuzione più processi, ciascuno con la propria socket UDP e relativo numero di porta. Quando i segmenti UDP giungono dalla rete, l'Host B dirige ciascun segmento (ossia ne esegue il demultiplexing) alla socket appropriata esaminando il numero di porta di destinazione del segmento.

È importante notare che una socket UDP viene identificata completamente da una coppia che consiste di un indirizzo IP e di un numero di porta di destinazione. Di conseguenza, due segmenti UDP che presentano diversi indirizzi IP e/o diversi numeri di porta di origine, ma hanno stesso indirizzo IP e stesso numero di porta di destinazione, vengono diretti allo stesso processo di destinazione tramite la medesima socket.

Per quanto riguarda il numero di porta di origine, osserviamo la Figura 3.4 in cui, nel segmento che va da A verso B, il numero di porta di origine serve come parte di un "indirizzo di ritorno": quando B vuole restituire il segmento ad A, la porta di destinazione del segmento da B verso A assumerà il valore della porta di origine del segmento da A verso B. L'indirizzo di ritorno completo è costituito dall'indirizzo IP di A più il numero di porta di origine. Per



**Figura 3.4** Inversione dei numeri di porta di origine e di destinazione.

esempio, vediamo il programma server UDP studiato nel Paragrafo 2.7. In `UDPServer.py`, il server utilizza il metodo `recvfrom()` per estrarre il numero di porta di origine dal segmento ricevuto dal client; poi invia un nuovo segmento al client, in cui il numero di porta di origine estratto viene usato come numero di porta di destinazione del nuovo segmento.

### Multiplexing e demultiplexing orientati alla connessione

Per comprendere il demultiplexing TCP, dobbiamo analizzare da vicino le socket TCP e il modo in cui si stabiliscono le connessioni TCP. Una sottile differenza tra una socket TCP e una socket UDP risiede nel fatto che la prima è identificata da quattro parametri: indirizzo IP di origine, numero di porta di origine, indirizzo IP di destinazione e numero di porta di destinazione. Pertanto, quando un segmento TCP giunge dalla rete in un host, quest'ultimo utilizza i quattro valori per dirigere (fare demultiplexing) il segmento verso la socket appropriata. In particolare, e al contrario di UDP, due segmenti TCP in arrivo, aventi indirizzi IP di origine o numeri di porta di origine diversi, vengono diretti a due socket differenti, anche a fronte di indirizzo IP e porta di destinazione uguali, con l'eccezione dei segmenti TCP che trasportano la richiesta per stabilire la connessione. Per chiarire meglio questo aspetto riconSIDeriamo l'esempio del Paragrafo 2.7.2.

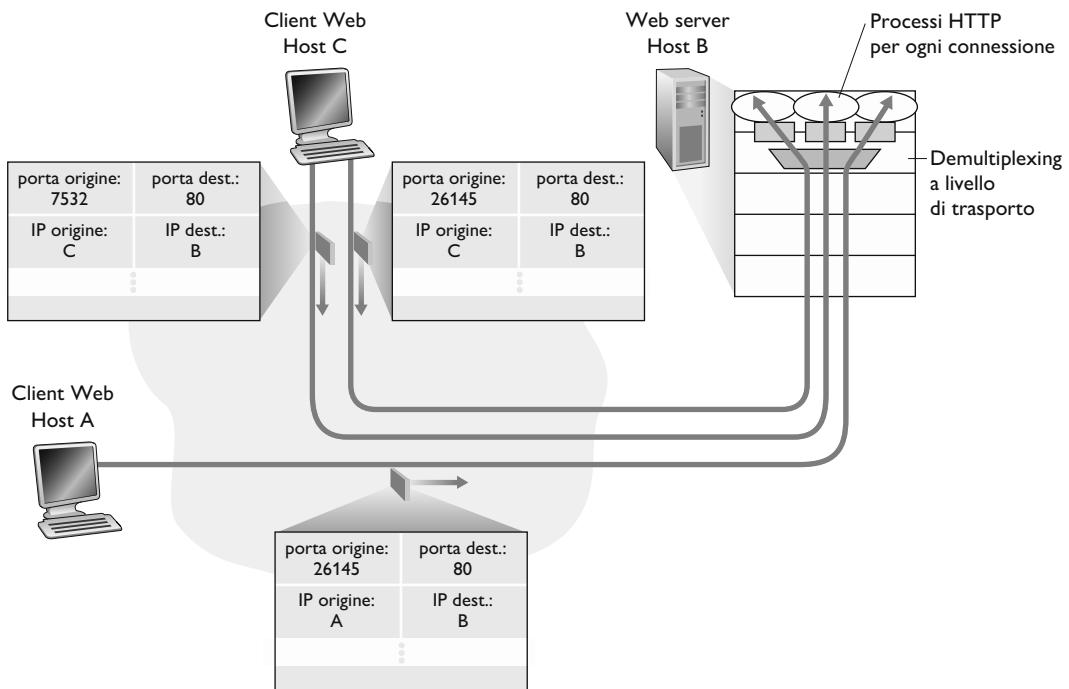
- L'applicazione server TCP presenta una “socket di benvenuto” che si pone in attesa di richieste di connessione da parte dei client TCP (Figura 2.28) sulla porta numero 12000.
- Il client TCP crea una socket e genera un segmento per stabilire la connessione tramite le seguenti linee di codice:

```
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, 12000))
```

- Una richiesta di connessione non è nient'altro che un segmento TCP con numero di porta di destinazione 12000 e uno speciale bit di richiesta di connessione posto a 1 nell'intestazione (Paragrafo 3.5). Il segmento include anche un numero di porta di origine, scelto dal client.
- Il sistema operativo dell'host che esegue il processo server, quando riceve il segmento con la richiesta di connessione con porta di destinazione 12000, lo calizza il processo server in attesa di accettare connessioni sulla porta 12000. Il processo server crea quindi una nuova connessione:

```
connectionSocket, addr = serverSocket.accept()
```

- Inoltre il livello di trasporto sul server prende nota dei seguenti valori nel segmento con la richiesta di connessione: (1) numero di porta di origine nel segmento, (2) indirizzo IP dell'host di origine, (3) numero di porta di destinazione nel segmento e (4) il proprio indirizzo IP. La socket di connessione appena creata viene identificata da questi quattro valori. Tutti i segmenti successivi la cui porta di origine, indirizzo IP di origine, porta di destinazione e indirizzo IP di destinazione coincidono con tali valori verranno diretti verso questa socket. Ora che la connessione TCP è attiva, client e server possono scambiarsi dati.



**Figura 3.5** Due client che usano lo stesso numero di porta di destinazione (80) per comunicare con la stessa applicazione sul web server.

L'host server può ospitare più socket TCP contemporanee collegate a processi diversi, ognuna identificata da una specifica quaterna di valori. Quando il segmento TCP arriva all'host, i quattro campi citati prima vengono utilizzati per dirigere (fare demultiplexing) il segmento verso la socket appropriata.

La situazione è schematizzata nella Figura 3.5 dove l'Host C dà inizio a due sessioni HTTP verso il server B, mentre l'Host A apre una sessione verso B. Gli Host A e C e il server B hanno ciascuno il proprio indirizzo IP, indicati rispettivamente con A, C e B. L'Host C assegna due diversi numeri di porta di origine (26145 e 7532) alle sue due connessioni HTTP. Dato che l'Host A sta scegliendo i numeri di porta di origine indipendentemente da C, potrebbe anch'esso attribuire una porta di origine 26145 alla sua connessione HTTP. Ma questo non è un problema: il server B sarà ancora in grado di fare correttamente demultiplexing delle due connessioni che, pur avendo lo stesso numero di porta di origine, hanno indirizzi IP differenti.

### Web server e TCP

Prima di concludere l'argomento, è utile spendere qualche parola ulteriore sui web server e sul loro utilizzo dei numeri di porta. Consideriamo un host che stia eseguendo un web server, supponiamo Apache, sulla porta 80. Quando i client (per esempio, i browser) inviano segmenti al server, *tutti* i segmenti hanno porta di destinazione 80. In particolare, sia i segmenti per stabilire la connessione iniziale sia quelli che trasportano messaggi di richiesta HTTP hanno porta di destinazione 80. Come abbiamo appena descritto, il server distingue i seg-

**BOX 3.1****FOCUS SULLA SICUREZZA****Il controllo delle porte**

Abbiamo visto come un processo server aspetti pazientemente su una porta aperta di essere contattato da un client remoto. Alcune porte sono riservate ad applicazioni note (per esempio, Web, FTP, DNS, SMTP server); altre porte sono usate per convenzione da applicazioni diffuse (per esempio, Microsoft Windows SQL server attende richieste UDP sulla porta 1434). Quindi, se determiniamo che c'è una porta aperta su un host, potremo essere in grado di far corrispondere quella porta alla specifica applicazione in esecuzione sull'host. Questo è molto utile per gli amministratori di sistema che hanno spesso bisogno di conoscere quali applicazioni di rete sono in esecuzione sugli host delle loro reti. Anche gli attaccanti, tuttavia, con lo scopo di raccogliere informazioni in vista di un attacco, vogliono sapere quali porte sono aperte su un host bersaglio. Se scoprono che un host sta eseguendo un'applicazione con un problema di sicurezza (per esempio, SQL server era soggetto a problemi di buffer overflow che consentivano a un utente remoto di eseguire codice arbitrario sull'host e questo problema di sicurezza veniva sfruttato dal worm Slammer [CERT 2003-04]), allora quell'host è pronto per essere attaccato.

La determinazione di quali applicazioni siano in ascolto su quali porte è un compito relativamente facile. In verità esistono programmi di pubblico dominio, chiamati **port scanner**, che svolgono questo compito. Forse il più diffuso tra questi è **nmap**, disponibile gratuitamente su <http://nmap.org> e incluso in molte distribuzioni Linux. Per TCP, nmap esegue una scansione sequenziale delle porte, cercando quelle che accettano connessioni. Per UDP, nmap di nuovo effettua una scansione sequenziale delle porte, cercando quelle che rispondono a segmenti UDP trasmessi. In entrambi i casi, nmap restituisce una lista delle porte aperte, chiuse o non raggiungibili. Un host che esegue nmap può tentare di operare una scansione di un qualsiasi bersaglio da qualunque punto di Internet. Rivedremo nmap nel Paragrafo 3.5.6, quando tratteremo della gestione della connessione TCP.

menti provenienti da client diversi tramite gli indirizzi IP e i numeri di porta di origine.

La Figura 3.5 mostra un web server che genera un nuovo processo per ogni connessione. Ciascuno di questi processi ha una propria socket attraverso la quale giungono richieste e sono inviate risposte HTTP. Sottolineiamo, tuttavia, che non esiste sempre una corrispondenza uno a uno tra le socket di connessione e i processi. Infatti gli odierni web server ad alte prestazioni spesso utilizzano solo un processo, ma creano un nuovo thread (che può essere visto come una sorta di sottoprocesso molto leggero da gestire per il sistema operativo) e una nuova socket di connessione per ciascun client. Il primo compito di programmazione del Capitolo 2 chiedeva di costruire un web server che eseguisse proprio questa operazione. In tale server, a ogni dato istante, si possono avere molte socket di connessione (con identificatori diversi) collegate allo stesso processo.

Client e server, se usano HTTP persistente, scambiano messaggi HTTP attraverso la stessa socket per tutta la durata della connessione. Se, invece, client e server usano HTTP non persistente, viene creata e chiusa una nuova connessione TCP per ciascuna coppia richiesta/risposta, e da quel momento viene crea-

ta e chiusa una nuova socket per ogni richiesta/risposta. La creazione e la chiusura frequente di socket può avere un forte impatto sulle prestazioni di un web server (sebbene il problema possa essere mitigato dal sistema operativo). I lettori interessati alle problematiche dei sistemi operativi inerenti a HTTP persistente e non persistente possono consultare [Nielsen 1997; Nahum 2002].

Ora che abbiamo discusso di multiplexing e demultiplexing a livello di trasporto, tratteremo uno dei protocolli di trasporto di Internet: UDP. Nel prossimo paragrafo vedremo come UDP non aggiunga quasi nulla al protocollo a livello di rete rispetto a un semplice servizio di multiplexing/demultiplexing.

### **3.3 Trasporto non orientato alla connessione: UDP**

Analizziamo ora da vicino le azioni di UDP; vi invitiamo a riguardare il Paragrafo 2.1 che include una panoramica del modello dei servizi UDP e il Paragrafo 2.7.1 che tratta la programmazione delle socket usando UDP.

Supponiamo di essere interessati al progetto di uno protocollo di trasporto ridotto all'osso. Immaginate, pertanto, di partire da un protocollo di trasporto “vuoto”. Lato mittente, prendiamo i messaggi dal processo applicativo e li passiamo direttamente a livello di rete; lato ricevente, prendiamo i messaggi in arrivo dal livello di rete e li trasferiamo direttamente al processo applicativo. Ma, come abbiamo appreso nel paragrafo precedente, dobbiamo fare qualcosa di più. Quantomeno, il livello di trasporto deve fornire un servizio di multiplexing/demultiplexing al fine di trasferire dati tra il livello di rete e il processo corretto a livello di applicazione.

UDP, definito in [RFC 768], fa praticamente il minimo che un protocollo di trasporto debba fare. A parte la funzione di multiplexing/demultiplexing e una forma di controllo degli errori molto semplice, non aggiunge nulla a IP. Quando, infatti, lo sviluppatore sceglie UDP anziché TCP, l'applicazione dialoga quasi in modo diretto con IP. UDP prende i messaggi dal processo applicativo, aggiunge il numero di porta di origine e di destinazione per il multiplexing/demultiplexing, aggiunge altri due piccoli campi e passa il segmento risultante al livello di rete. Questi incapsula il segmento in un datagramma IP e quindi effettua un tentativo di consegnarlo all'host di destinazione in modalità best-effort. Se il segmento arriva a destinazione, UDP utilizza il numero di porta di destinazione per consegnare i dati del segmento al processo applicativo corretto. Notiamo che in UDP non esiste handshaking tra le entità di invio e di ricezione a livello di trasporto. Per questo motivo, si dice che UDP è *non orientato alla connessione* (*connectionless*).

DNS è un tipico esempio di protocollo a livello applicativo che utilizza UDP. Quando l'applicazione DNS in un host vuole effettuare una query (*interrogazione*), costruisce un messaggio di query DNS e lo passa a UDP. Senza effettuare alcun handshaking con l'entità UDP in esecuzione sul sistema di destinazione, il sistema aggiunge i campi d'intestazione al messaggio e trasferisce il segmento risultante al livello di rete. Quest'ultimo incapsula il segmento UDP in un datagramma e lo invia a un server DNS. L'applicazione DNS sull'host che ha effettuato la richiesta aspetta quindi una risposta. Se non ne riceve (magari perché la rete sottostante ha smarrito la richiesta o la risposta), l'applicazione

tenta di inviare la richiesta a un altro DNS server oppure informa l'applicazione dell'impossibilità di ottenere una risposta.

Ora ci si potrebbe chiedere perché uno sviluppatore dovrebbe scegliere di costruire un'applicazione su UDP anziché su TCP. Non è forse sempre preferibile TCP, visto che fornisce un servizio di trasferimento dati affidabile mentre UDP no? La risposta è no, in quanto molte applicazioni risultano più adatte a UDP per i motivi che seguono.

- *Controllo più preciso a livello di applicazione su quali dati sono inviati e quando.* Non appena un processo applicativo passa dei dati a UDP, quest'ultimo li impacchetta in un segmento che trasferisce immediatamente al livello di rete. TCP, invece, dispone di un meccanismo di controllo della congestione che ritarda l'invio a livello di trasporto quando uno o più collegamenti tra l'origine e la destinazione diventano eccessivamente congestionati. TCP continua, inoltre, a inviare il segmento fino a quando viene notificata la sua ricezione da parte della destinazione, incurante del tempo richiesto per un trasporto affidabile. Dato che le applicazioni in tempo reale spesso richiedono una velocità minima di invio (*minimum sending rate*) e non sopportano ritardi eccessivi nella trasmissione dei pacchetti mentre tollerano una certa perdita di dati, il modello di servizio TCP non si adatta particolarmente bene a queste esigenze. Come vedremo più avanti, queste applicazioni possono usare UDP e implementare funzionalità aggiuntive rispetto al servizio minimale offerto dal protocollo, come parte dell'applicazione.
- *Nessuna connessione stabilita.* Come vedremo più avanti, TCP utilizza un handshake a tre vie prima di iniziare il trasferimento dei dati. UDP invece “spara” dati a raffica senza alcun preliminare formale. Pertanto, UDP non introduce alcun ritardo nello stabilire una connessione. Questo è probabilmente il motivo principale per cui DNS utilizza UDP anziché TCP (con cui risulterebbe molto più lento). HTTP invece usa TCP, dato che l'affidabilità risulta critica per le pagine web con testo. Ma, come abbiamo visto nel Paragrafo 2.2, il ritardo per stabilire una connessione TCP in HTTP contribuisce in maniera significativa ai ritardi associati allo scaricamento di documenti web. Il protocollo QUIC (*Quick UDP Internet Connection*, [IETF QUIC 2020]), utilizzato nel browser Chrome di Google, utilizza UDP come protocollo di trasporto e implementa l'affidabilità in un protocollo a livello di applicazione. Tratteremo brevemente il protocollo QUIC nel Paragrafo 3.8.
- *Nessuno stato di connessione.* TCP mantiene lo stato della connessione nei sistemi periferici. Questo stato include buffer di ricezione e di invio, parametri per il controllo di congestione e parametri sul numero di sequenza e di acknowledgment (*notifica, riscontro*). Vedremo nel corso del Paragrafo 3.5 che queste informazioni di stato sono richieste per implementare il servizio di trasferimento dati affidabile proprio di TCP e per fornire il controllo di congestione. UDP, invece, non conserva lo stato della connessione e non tiene traccia di questi parametri. Per questo motivo, un server dedicato a una particolare applicazione può generalmente supportare molti più client attivi quando l'applicazione utilizza UDP anziché TCP.
- *Minor spazio usato per l'intestazione del pacchetto.* L'intestazione dei pacchetti TCP aggiunge 20 byte, mentre UDP solo 8.

| Applicazione                      | Protocollo a livello di applicazione | Protocollo di trasporto sottostante |
|-----------------------------------|--------------------------------------|-------------------------------------|
| Posta elettronica                 | SMTP                                 | TCP                                 |
| Accesso a terminali remoti        | Telnet                               | TCP                                 |
| Accesso sicuro a terminali remoti | SSH                                  | TCP                                 |
| Web                               | HTTP, HTTP/3                         | TCP (per HTTP), UDP (per HTTP/3)    |
| Trasferimento file                | FTP                                  | TCP                                 |
| Server di file remoti             | NFS                                  | generalmente UDP                    |
| Dati multimediali in streaming    | DASH                                 | TCP                                 |
| Telefonia su Internet             | generalmente proprietario            | UDP o TCP                           |
| Gestione di rete                  | SNMP                                 | generalmente UDP                    |
| Traduzione di nomi                | DNS                                  | generalmente UDP                    |

**Figura 3.6** Protocolli di trasporto delle più diffuse applicazioni Internet.

La Figura 3.6 elenca alcune diffuse applicazioni per Internet e i relativi protocolli di trasporto usati. Come potevamo aspettarci, la posta elettronica, l'accesso a terminali remoti e il trasferimento di file utilizzano TCP in quanto richiedono un servizio di trasferimento dati affidabile. Come visto nel Capitolo 2, le prime versioni di HTTP utilizzavano TCP, ma quelle più recenti usano UDP, fornendo il proprio controllo degli errori e il proprio controllo di congestione (tra gli altri servizi) al livello di applicazione. Ciò nondimeno, molte applicazioni importanti scelgono UDP; per esempio UDP viene utilizzato per trasportare dati di gestione della rete (SNMP, Paragrafo 5.7). In questo caso UDP viene preferito a TCP perché le applicazioni di gestione della rete vanno spesso in esecuzione quando la rete stessa è in uno stato di stress, più precisamente quando è difficile trasferire dati controllando la congestione o in maniera affidabile. Inoltre, come abbiamo menzionato precedentemente, anche il DNS fa uso di UDP per evitare i ritardi dovuti alla creazione di una connessione TCP.

Come illustrato nella Figura 3.6, sia UDP che TCP sono oggi utilizzati per le applicazioni multimediali, quali la telefonia su Internet, la videoconferenza in tempo reale e lo streaming audio e video. Tutte queste applicazioni possono tollerare una piccola quantità di perdita di pacchetti e, quindi, il trasferimento dati affidabile non risulta assolutamente essenziale per il loro successo. Inoltre le applicazioni in tempo reale quali la telefonia su Internet e la videoconferenza reagiscono molto male al controllo di congestione TCP. Per questi motivi, gli sviluppatori di applicazioni multimediali spesso scelgono UDP anziché TCP. Quando il tasso di perdita dei pacchetti è basso e a causa del fatto che alcune istituzioni bloccano il traffico UDP per ragioni di sicurezza (si veda il Capitolo 8, disponibile in inglese sulla piattaforma MyLab), TCP diventa sempre più alllettante come protocollo per il trasporto dello streaming multimediale.

Sebbene costituisca oggi una prassi comune, l'uso di UDP per le applicazioni multimediali va trattato con attenzione. Come già detto, UDP non implementa il controllo di congestione, necessario per evitare che la rete entri in uno stato di congestione tale per cui poco traffico utile raggiunge la sua de-

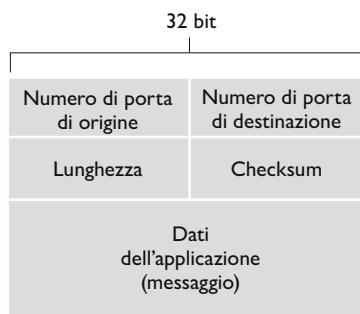
stinazione. Se tutti facessero uso di streaming video con grande richiesta di banda e senza alcun controllo di congestione, i router subirebbero un tale sovraccarico che ben pochi pacchetti UDP riuscirebbero a raggiungere la destinazione. Inoltre, l'alta frequenza di perdite indotta da invii UDP incontrollati provocherebbe una drammatica diminuzione del tasso degli invii TCP (che, come vedremo, diminuisce la propria velocità di trasmissione per fronteggiare la congestione). Di conseguenza, la mancanza di controllo di congestione di UDP può avere come risultato un'alta percentuale di perdite tra mittente e destinatario UDP, nonché uno schiacciamento soffocante delle sessioni TCP. Molti ricercatori hanno proposto nuovi meccanismi per forzare tutte le sorgenti, comprese quelle UDP, a effettuare controlli di congestione adattivi [Mahdavi 1997; Floyd 2000; Kohler 2006; RFC 4340].

Prima di trattare la struttura dei segmenti UDP menzioniamo che le applicazioni *possono* ottenere un trasferimento dati affidabile anche con UDP. Ciò avviene se l'affidabilità è insita nell'applicazione stessa (per esempio, con meccanismi di notifica e ritrasmissione come quelli che studieremo successivamente). Come già menzionato, il protocollo QUIC (*Quick UDP Internet Connection*, [IETF QUIC 2020]) utilizza UDP come protocollo di trasporto e implementa l'affidabilità in un protocollo a livello di applicazione. Si tratta tuttavia di un compito non banale, che terrebbe occupato un programmatore per molto tempo. Ciò nondimeno, avere affidabilità direttamente nell'applicazione consentirebbe ai processi applicativi di comunicare in modo affidabile senza essere soggetti ai vincoli sulla velocità di trasmissione imposti dai meccanismi del controllo di congestione di TCP.

### 3.3.1 Struttura dei segmenti UDP

L'RFC 768 definisce la struttura dei segmenti UDP (Figura 3.7), nei quali i dati dell'applicazione occupano il campo Dati. Per esempio, per DNS, il campo Dati contiene un messaggio di richiesta o di risposta, mentre nel caso delle applicazioni di streaming audio sono i campioni audio a riempire il campo Dati. L'intestazione UDP presenta solo quattro campi di due byte ciascuno. Come visto nel precedente paragrafo, i numeri di porta consentono all'host di destinazione di trasferire i dati applicativi al processo corretto (ossia di effettuare il demultiplexing). Il campo Lunghezza specifica il numero di byte del segmento UDP (intestazione più dati). Un valore esplicito di lunghezza è necessario per-

**Figura 3.7** Struttura dei segmenti UDP.



ché la grandezza del campo Dati può essere diversa tra un segmento e quello successivo. L'host ricevente utilizza il checksum per verificare se sono avvenuti errori nel segmento. In realtà, oltre che sul segmento UDP, il checksum è calcolato anche su alcuni campi dell'intestazione IP, ma a questo punto della trattazione ignoriamo tale dettaglio. Tratteremo il calcolo del checksum nel prossimo paragrafo. I principi di base sul rilevamento degli errori sono descritti nel Paragrafo 6.2.

### 3.3.2 Checksum UDP

Il checksum UDP serve per il rilevamento degli errori. In altre parole, viene utilizzato per determinare se i bit del segmento UDP sono stati alterati durante il loro trasferimento (per esempio, a causa di disturbi nei collegamenti o quando sono stati memorizzati in un router) da sorgente a destinazione. Lato mittente UDP effettua il complemento a 1 della somma di tutte le parole da 16 bit nel segmento, e l'eventuale riporto finale viene sommato al primo bit. Tale risultato viene posto nel campo Checksum del segmento UDP. Qui di seguito forniamo un semplice esempio di calcolo del checksum; i dettagli per una implementazione efficiente del calcolo si possono trovare nell'RFC 1071 e considerazioni sulle prestazioni con dati reali in [Stone 1998; Stone 2000]. Come esempio supponiamo di avere le seguenti tre parole di 16 bit:

```
0110011001100000
0101010101010101
100011100001100
```

La somma delle prime due è:

```
0110011001100000
0101010101010101
1011101110110101
```

Sommendo la terza parola al risultato precedente otteniamo:

```
1011101110110101
100011100001100
0100101011000010
```

Notiamo che il riporto di quest'ultima somma è stato sommato al primo bit. Il complemento a 1 si ottiene convertendo i bit 0 in 1 e viceversa. Di conseguenza, il checksum sarà 1011010100111101. In ricezione, si sommano le tre parole iniziali e il checksum. Se non ci sono errori nel pacchetto, l'addizione darà 1111111111111111, altrimenti se un bit vale 0 sappiamo che è stato introdotto almeno un errore nel pacchetto.

Prima di tutto ci si potrebbe chiedere perché UDP metta a disposizione un checksum, dato che anche molti protocolli a livello di collegamento (tra cui Ethernet) prevedono il controllo degli errori. Il motivo è che non c'è garanzia che tutti i collegamenti tra origine e destinazione controllino gli errori. Inoltre, anche se i segmenti fossero trasferiti correttamente lungo un collegamento, si potrebbe verificare un errore mentre il segmento si trova nella memoria di un router. Dato che non sono garantiti né l'affidabilità del singolo collegamento

né il rilevamento di errori in memoria, UDP deve mettere a disposizione a livello di trasporto un meccanismo di verifica su base end-to-end se si vuole che il servizio trasferimento dati sia in grado di rilevare eventuali errori. Questo è un esempio del celebrato **principio end-to-end** nella progettazione dei sistemi [Saltzer 1984] in base al quale, dato che determinate funzionalità (in questo caso il rilevamento degli errori) devono essere implementate su base end-to-end, “le funzionalità posizionate ai livelli inferiori possono diventare ridondanti o di scarso valore se confrontate con le stesse funzionalità offerte dai livelli superiori”.

Dato che si suppone che IP funzioni correttamente con tutti i protocolli di secondo livello, è utile che il livello di trasporto fornisca un controllo degli errori come misura di sicurezza. Sebbene metta a disposizione tale controllo, UDP non fa nulla per risolvere le situazioni di errore; alcune implementazioni di UDP si limitano a scartare il segmento danneggiato, altre lo trasmettono all'applicazione con un avvertimento.

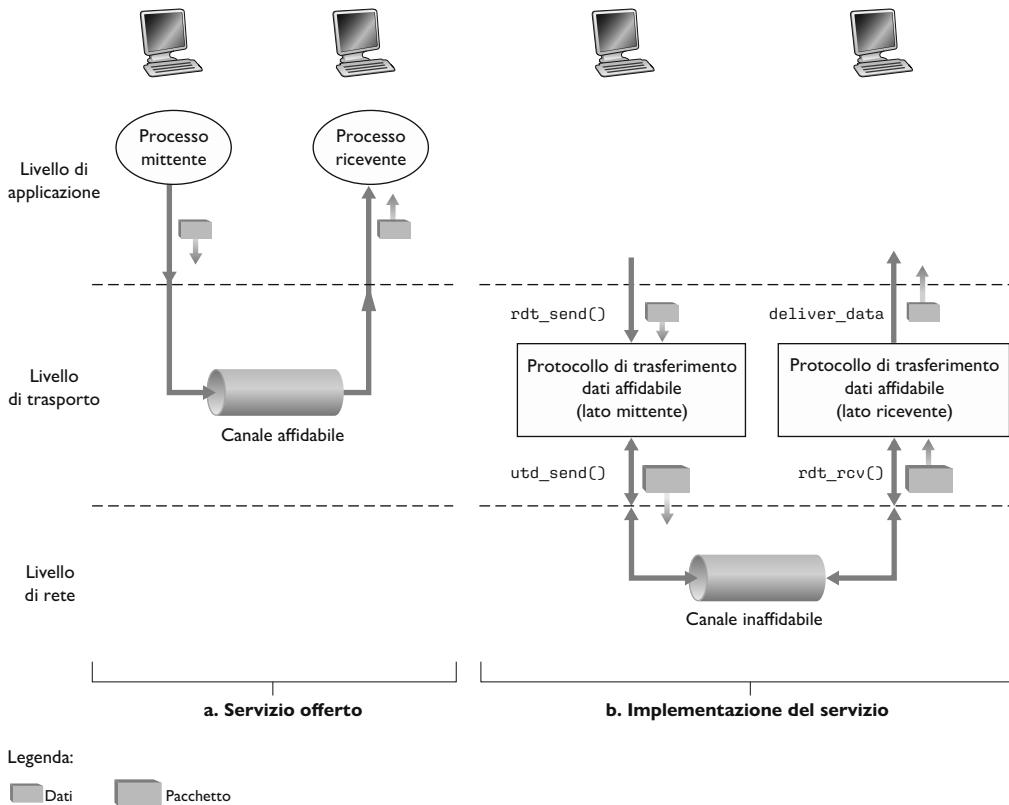
Questo chiude la nostra discussione su UDP. Vedremo presto che TCP fornisce un trasferimento dati affidabile alle proprie applicazioni, così come altri servizi che UDP non offre. Naturalmente, TCP è più complesso e, prima di analizzarlo, risulta utile fare un passo indietro e trattare i principi alla base del trasferimento affidabile dei dati.

### 3.4 Princìpi del trasferimento dati affidabile

Consideriamo ora il problema del trasferimento dati affidabile in un contesto generale. Questa scelta è opportuna, dato che il problema dell'implementazione di un trasferimento dati affidabile si verifica non solo a livello di trasporto, ma anche a livello di collegamento e di applicazione. Il problema generale è, quindi, di fondamentale importanza nel campo del networking; dovendone indicare i primi dieci problemi, questo potrebbe candidarsi come primo della lista. Nel prossimo paragrafo esamineremo TCP e vedremo come questo sfrutti molti dei principi che stiamo per descrivere.

La Figura 3.8 illustra il contesto della nostra trattazione sul trasferimento dati affidabile. L'astrazione del servizio offerta alle entità dei livelli superiori è quella di un canale affidabile tramite il quale si possono trasferire dati. Con un canale affidabile a disposizione nessun bit dei dati trasferiti è corrotto (0 al posto di 1 o viceversa) o va perduto e tutti i bit sono consegnati nell'ordine di invio. Si tratta precisamente del modello di servizio offerto da TCP alle applicazioni per Internet che ne fanno uso.

Il compito di un **protocollo di trasferimento dati affidabile** (*reliable data transfer protocol*) è l'implementazione di questa astrazione del servizio. Ciò è complicato dalla possibile inaffidabilità del livello “al di sotto” del protocollo di trasferimento dati. Per esempio, TCP è un protocollo di trasferimento dati affidabile implementato appoggiandosi a un livello di rete (IP) che non è affidabile end-to-end. Più in generale, il livello sottostante ai due punti terminali, che comunicano in modo affidabile, può consistere di un singolo collegamento fisico (come nel caso di un protocollo di trasferimento dati a livello di collegamento) o di una rete (come nel caso di un protocollo a livello di trasporto). Per



**Figura 3.8** Trasferimento dati affidabile: modello di servizio e implementazione.

i nostri scopi, tuttavia, possiamo vedere questo livello inferiore semplicemente come un canale punto a punto inaffidabile.

In questo paragrafo svilupperemo in modo complementare i lati mittente e ricevente di un protocollo di trasferimento dati affidabile, considerando modelli via via più complessi del sottostante canale. Per esempio, considereremo quali meccanismi di protocollo siano necessari quando il canale sottostante può corrompere i bit o perdere interi pacchetti. Nella nostra discussione assumeremo che i pacchetti vengano consegnati nell'ordine con cui sono stati inviati, ma che alcuni possano andare persi; vale a dire che il canale sottostante non riordina i pacchetti. La Figura 3.8(b) mostra le interfacce per il nostro protocollo di trasferimento dati. Il lato mittente del protocollo di trasferimento dati sarà invocato tramite una chiamata a `rdt_send()` e trasferirà i dati da consegnare al livello superiore sul lato ricevente. In questo caso `rdt` sta per “reliable data transfer” (trasferimento dati affidabile) e `_send` indica la chiamata al lato mittente di `rdt`. Quando un pacchetto raggiunge il lato ricevente del canale, viene chiamata `rdt_rcv()`. Il protocollo `rdt`, quando vuole consegnare i dati al livello superiore, chiama `deliver_data()`. Da qui in avanti useremo *pacchetto* anziché *segmento* per il livello di trasporto; infatti, dato che la teoria sviluppata in questo paragrafo si applica alle reti di calcolatori in generale e non solo al li-

vello di trasporto in Internet, il termine generico pacchetto è forse più appropriato.

In questo paragrafo consideriamo solo il caso di **trasferimento dati unidirezionale** (*unidirectional data transfer*). Il caso del **trasferimento dati bidirezionale** (*bidirectional data transfer*), cioè full-duplex, non è concettualmente più difficile, ma assai più noioso da spiegare. Tuttavia è importante notare che i lati mittente e ricevente del nostro protocollo avranno la necessità di trasmettere pacchetti in *entrambe* le direzioni (Figura 3.8). Vedremo che, oltre a scambiare pacchetti contenenti dati da trasferire, i due lati di rdt necessiteranno anche del reciproco scambio di pacchetti di controllo: entrambi inviano pacchetti tramite una chiamata a `udt_send()` (UDT, *Unreliable Data Transfer*).

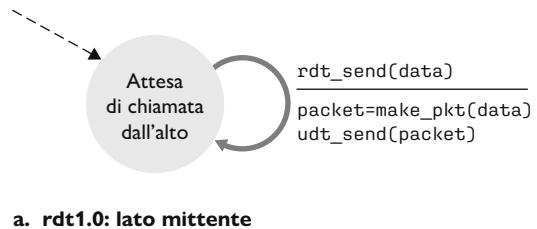
### 3.4.1 Costruzione di un protocollo di trasferimento dati affidabile

Passiamo ora in rassegna una serie di protocolli via via sempre più complessi, per arrivare a un impeccabile protocollo di trasferimento dati affidabile.

#### Trasferimento dati affidabile su un canale perfettamente affidabile: rdt1.0

Innanzitutto consideriamo il caso più semplice, in cui il canale sottostante è completamente affidabile. Il protocollo che chiameremo rdt1.0 è banale. Le definizioni della **macchina a stati finiti** (FSM, *Finite-State Machine*) del mittente e del destinatario rdt1.0 sono illustrate nella Figura 3.9: la FSM (a) definisce le operazioni del mittente, l'altra (b) mostra come opera il destinatario. È importante notare che esistono due FSM separate, una per il mittente e una per il destinatario. Dato che le FSM della figura hanno un unico stato, le transizioni (indicate dalle frecce) hanno luogo necessariamente tra quello stato e sé stesso. L'evento che causa la transizione è scritto sopra la linea orizzontale che la eti-

**Figura 3.9** rdt1.0: protocollo per un canale completamente affidabile.



a. rdt1.0: lato mittente



b. rdt1.0: lato ricevente

chetta, e le azioni intraprese in seguito all'evento sono scritte sotto. Quando un evento non determina un'azione e quando viene intrapresa un'azione senza il verificarsi di un evento, useremo la lettera greca Λ rispettivamente sotto o sopra la linea orizzontale per denotare esplicitamente la mancanza di un'azione o di un evento. Lo stato iniziale della FSM è indicato dalla freccia tratteggiata. Sebbene le FSM della Figura 3.9 abbiano un unico stato, quelle che vedremo tra breve hanno molti stati ed è quindi importante identificare quello iniziale.

Il lato mittente di rdt accetta semplicemente dati dal livello superiore tramite l'evento `rdt_send(data)`, crea un pacchetto contenente dati con l'azione `make_pkt(data)` e lo invia sul canale. In pratica, l'evento `rdt_send(data)` è il risultato di una chiamata a procedura, per esempio, a `rdt_send()`, da parte dell'applicazione al livello superiore.

Per quanto riguarda il lato ricevente, rdt raccoglie i pacchetti dal sottostante canale tramite l'evento `rdt_rcv(packet)`, rimuove i dati dai pacchetti tramite l'azione `extract(packet, data)` e li passa al livello superiore con l'azione `deliver_data(data)`. In pratica, l'evento `rdt_rcv(packet)` è il risultato di una chiamata a procedura, per esempio a `rdt_rcv()`, da parte del protocollo di livello inferiore.

In questo semplice protocollo non c'è differenza tra un'unità di dati e un pacchetto. Inoltre, tutti i pacchetti fluiscano dal mittente al destinatario; con un canale perfettamente affidabile, non c'è alcun bisogno che il lato ricevente fornisca informazioni al mittente, dato che nulla può andare storto. Si noti che abbiamo anche ipotizzato che il destinatario possa ricevere dati al tasso di invio del mittente. Pertanto, il destinatario non dovrà mai chiedere al mittente di rallentare.

### **Trasferimento dati affidabile su un canale con errori sui bit: rdt2.0**

Un modello più realistico del canale sottostante è quello in cui i bit in un pacchetto possono essere corrotti. Tali errori si verificano nei componenti fisici delle reti quando il pacchetto viene trasmesso, propagato o inserito nei buffer. Continueremo ad assumere, per il momento, che tutti i pacchetti trasmessi vengano ricevuti nell'ordine di invio, anche se i loro bit possono essere corrotti.

Prima di sviluppare un protocollo per la comunicazione affidabile su tale canale consideriamo come le persone agirebbero in una situazione analoga. Analizziamo come viene dettato un lungo messaggio al telefono. In uno scenario tipico, chi raccoglie il messaggio potrebbe dire “OK” dopo ogni frase che ha sentito, compreso e memorizzato. Se la persona che prende nota non capisce una frase, chiede di ripeterla. Questo protocollo di dettatura dei messaggi usa **notifiche positive** (*positive acknowledgments*) (“OK”) e **notifiche negative** (“Per favore, ripeti”). Tali messaggi di controllo consentono al destinatario di far sapere al mittente che cosa sia stato ricevuto correttamente e che cosa no, chiedendone quindi la ripetizione. Nel contesto di una rete di calcolatori, i protocolli di trasferimento dati affidabili basati su ritrasmissioni sono noti come **protocolli ARQ** (*Automatic Repeat reQuest*).

Fondamentalmente, per gestire la presenza di errori nei bit, i protocolli ARQ devono avere tre funzionalità aggiuntive.

- *Rilevamento dell'errore (error detection)*. Innanzitutto è richiesto un meccanismo che consenta al destinatario di rilevare gli errori sui bit. Ricordiamo

che UDP utilizza il campo di checksum per questo preciso scopo. Nel corso del Capitolo 6 esamineremo in maggior dettaglio le tecniche di rilevamento e correzione degli errori. Per ora è sufficiente sapere che tali tecniche richiedono l'invio di bit extra (oltre a quelli dei dati da trasferire) tra mittente e destinatario. Questi bit verranno raccolti nel campo di checksum nel pacchetto dati `rdt2.0`.

- *Feedback del destinatario (receiver feedback)*. Dato che mittente e destinatario sono generalmente in esecuzione su sistemi periferici diversi, magari separati da migliaia di chilometri, l'unico modo che ha il mittente per conoscere la “visione del mondo” del destinatario (in questo caso, se un pacchetto sia stato ricevuto correttamente o meno) consiste nel feedback esplicito del destinatario. Le risposte di notifica positiva (ACK) e negativa (NAK) nello scenario del messaggio dettato al telefono sono esempi di feedback. Analogamente il nostro protocollo `rdt2.0` manderà pacchetti ACK e NAK dal destinatario al mittente. In linea di principio, tali pacchetti potrebbero essere costituiti da un solo bit: per esempio, 0 per NAK e 1 per ACK.
- *Ritrasmissione (retransmission)*. Un pacchetto ricevuto con errori sarà ritrasmesso dal mittente.

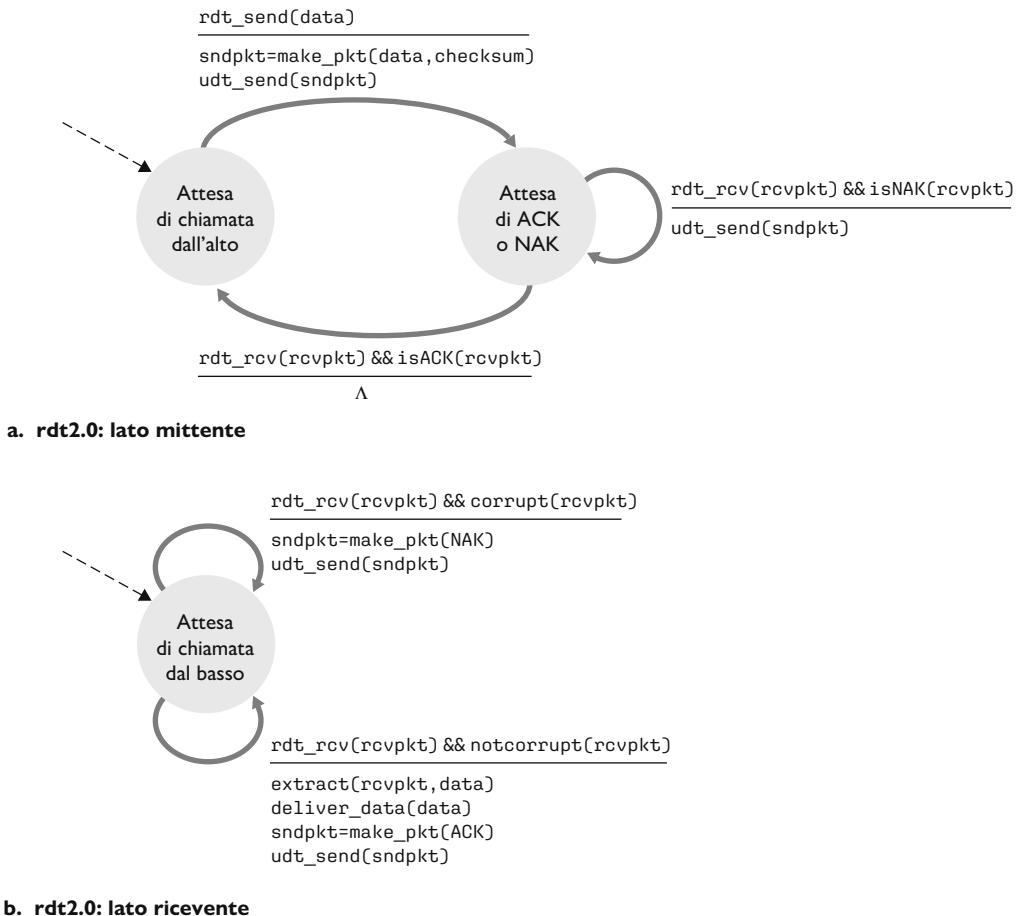
La Figura 3.10 illustra l'automa che descrive `rdt2.0`, un protocollo che utilizza il rilevamento di errore, le notifiche positive e le notifiche negative.

Il lato mittente di `rdt2.0` presenta due stati. In quello di sinistra, il protocollo lato mittente sta attendendo i dati da raccogliere dal livello superiore. Quando si verifica l'evento `rdt_send(data)`, il mittente crea un pacchetto (`sndpkt`) contenente i dati da inviare, insieme al checksum (si veda il Paragrafo 3.3.2 nel caso di un pacchetto UDP) e infine spedisce il pacchetto tramite l'operazione `udt_send(sndpkt)`. Nello stato di destra, il protocollo mittente è in attesa di un pacchetto ACK o NAK dal destinatario. Se riceve un ACK (evento denotato da `rdt_recv(rcvpkt) && isACK(rcvpkt)` nella Figura 3.10), il mittente sa che il pacchetto trasmesso più di recente è stato ricevuto correttamente e pertanto il protocollo ritorna allo stato di attesa dei dati provenienti dal livello superiore. Invece, se riceve un NAK, il protocollo ritrasmette l'ultimo pacchetto e attende una risposta alla ritrasmissione. È importante notare che quando il mittente è nello stato di attesa di ACK o NAK, *non può* recepire dati dal livello superiore; in altre parole, non può aver luogo l'evento `rdt_send()`, che invece si verificherà solo dopo la ricezione di un ACK che permette al mittente di cambiare stato. Quindi, il mittente non invia nuovi dati finché non è certo che il destinatario abbia ricevuto correttamente il pacchetto corrente. È proprio per questo comportamento che i protocolli quali `rdt2.0` sono noti come **protocolli stop-and-wait**.

La FSM lato ricevente di `rdt2.0` ha ancora un solo stato. All'arrivo del pacchetto, il destinatario risponde o con un ACK o con un NAK, a seconda che il pacchetto sia corrotto o meno.

Nella Figura 3.10 la notazione `rdt_recv(rcvpkt) && corrupt(rcvpkt)` corrisponde al caso in cui si riceve un pacchetto con qualche errore.

Il protocollo `rdt2.0` sembrerebbe funzionare ma, sfortunatamente, presenta un grave difetto; infatti non abbiamo tenuto conto della possibilità che i pacchetti ACK o NAK possano a loro volta essere alterati. Prima di procedere,



**Figura 3.10** rdt2.0: protocollo per un canale con errori sui bit.

dobbiamo assolutamente pensare a come risolvere questo problema, quanto meno dovremmo aggiungere dei bit di checksum ai pacchetti ACK/NAK per rilevare tali errori. Il problema più difficile riguarda come il protocollo dovrebbe risolvere gli errori nei pacchetti ACK o NAK. Infatti se un ACK o un NAK è corrotto, il mittente non ha modo di sapere se il destinatario abbia ricevuto correttamente l'ultimo blocco di dati trasmessi.

Prendiamo in considerazione tre possibilità per gestire gli ACK e i NAK corrotti.

- Nel primo caso vediamo come agirebbe una persona nella situazione della dettatura di messaggi. Se chi detta non comprende la risposta “OK” o “Per favore, ripeti” da parte del destinatario, probabilmente chiederà “Che cosa hai detto?” (introducendo nel protocollo un nuovo tipo di pacchetto dal mittente al destinatario). Il destinatario ripeterà, quindi, la risposta. Ma che cosa succede se il messaggio “Che cosa hai detto?” da parte di chi detta è a sua volta corrotto? Il destinatario, non sapendo se la frase confusa fa parte della dettatura o è una richiesta di ripetere l'ultima risposta, risponderebbe pro-

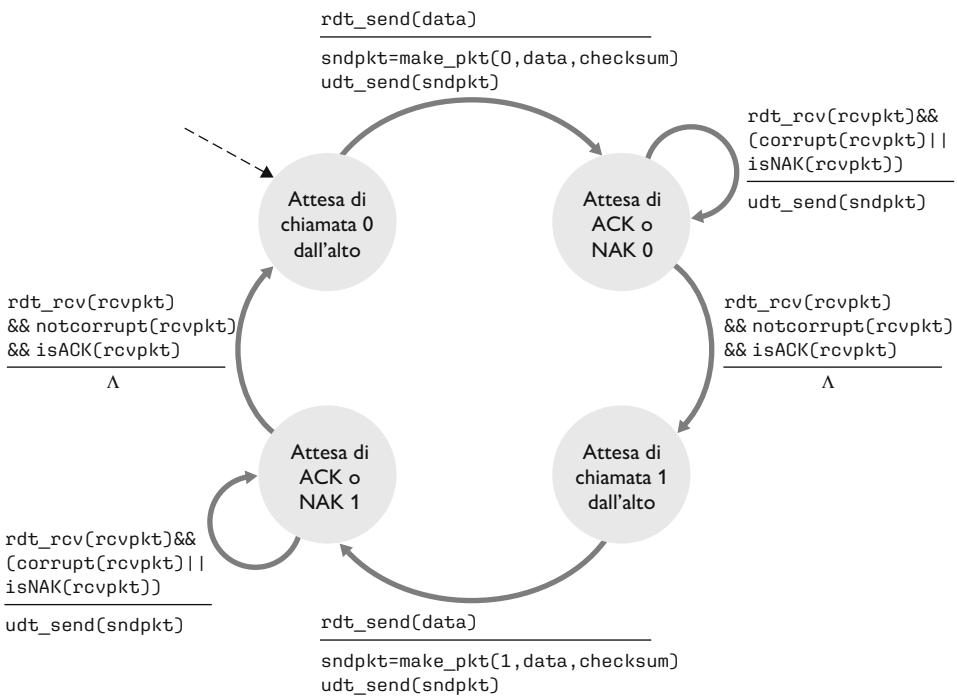
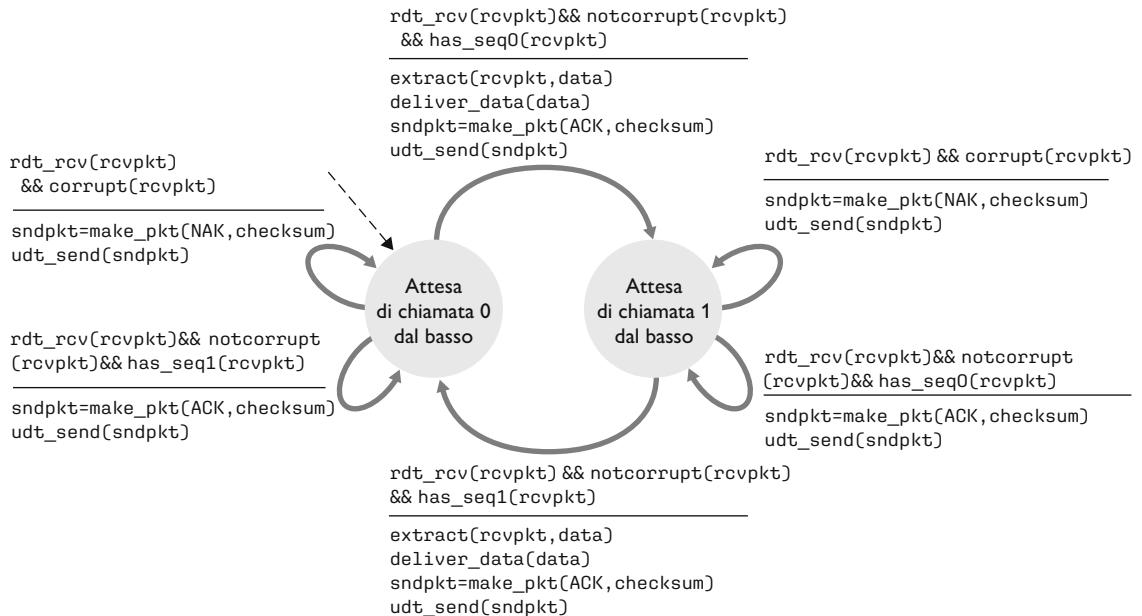
babilmente con “Che cos’hai detto *tu*?”. Ovviamente, ancora una volta tale risposta potrebbe essere confusa. Non c’è dubbio che stiamo camminando sulle sabbie mobili.

- Un’alternativa è l’aggiunta di bit di checksum sufficienti a consentire al mittente non solo di trovare, ma anche di correggere gli errori sui bit. Ciò risolve il problema solo per un canale che può danneggiare pacchetti, ma non perderli.
- Un terzo approccio prevede semplicemente che il mittente rinvii il pacchetto di dati corrente a seguito della ricezione di un pacchetto ACK o NAK alterato. Questo approccio, tuttavia, introduce **pacchetti duplicati** (*duplicate packets*) nel canale. La fondamentale difficoltà insita nella duplicazione di pacchetti è che il destinatario non sa se l’ultimo ACK o NAK inviato sia stato ricevuto correttamente dal mittente. Di conseguenza, non può sapere “a priori” se un pacchetto in arrivo contenga dati nuovi o rappresenti una ritrasmissione.

Una soluzione semplice a questo nuovo problema (adottata in quasi tutti i protocolli di trasferimento dati, tra cui TCP) consiste nell’aggiungere un campo al pacchetto dati, obbligando il mittente a numerare i propri pacchetti dati con un **numero di sequenza** (*sequence number*) nel nuovo campo. Al destinatario sarà sufficiente controllare questo numero per sapere se il pacchetto ricevuto rappresenta meno una ritrasmissione. Per questo semplice protocollo stop-and-wait, un numero di sequenza da 1 bit sarà sufficiente, dato che consentirà al destinatario di sapere se il mittente stia ritrasmettendo un pacchetto o inviandone uno già trasmesso. Nel primo caso il numero di sequenza del pacchetto ha lo stesso numero di sequenza del pacchetto appena ricevuto, nel secondo caso il numero di sequenza sarà diverso. Dato che stiamo ipotizzando che il canale non perda pacchetti, i pacchetti ACK e NAK non devono indicare il numero di sequenza del pacchetto di cui rappresentano la notifica. Il mittente sa che un pacchetto ricevuto di tipo ACK o NAK (alterato o meno) è stato generato come risposta al pacchetto dati trasmesso più di recente.

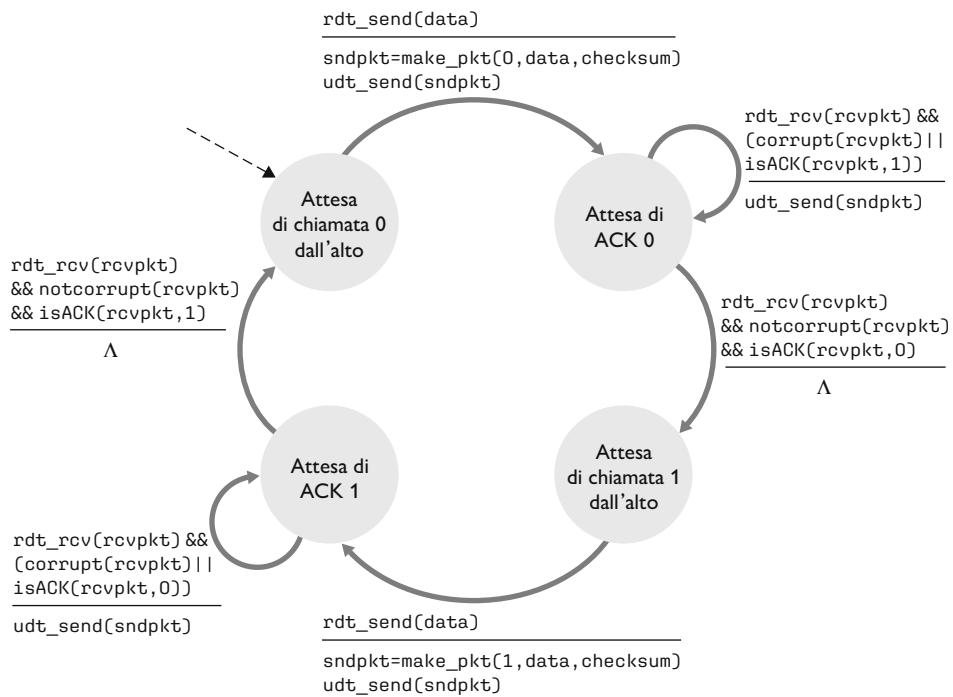
Le Figure 3.11 e 3.12 illustrano le FSM di rdt2.1 (la nostra versione corretta di rdt2.0). Le FSM di mittente e destinatario hanno ora il doppio degli stati precedenti. Questo avviene perché lo stato del protocollo deve riflettere il fatto che il pacchetto attualmente in invio o in ricezione abbia numero di sequenza 0 o 1. Notiamo che le azioni negli stati di invio e di attesa di un pacchetto numerato 0 sono immagini speculari delle azioni negli stati in cui viene spedito o si attende un pacchetto numerato 1. L’unica differenza riguarda la gestione del numero di sequenza.

Il protocollo rdt2.1 usa acknowledgments positivi e negativi dal destinatario verso il mittente. Il destinatario manda un acknowledgment positivo quando riceve un pacchetto fuori sequenza, e un acknowledgment negativo, quando riceve un pacchetto alterato. Possiamo ottenere lo stesso effetto di un NAK spendendo piuttosto un ACK per il più recente pacchetto ricevuto correttamente. Un mittente che riceve due ACK per lo stesso pacchetto (ossia riceve **ACK duplicati**) sa che il destinatario non ha ricevuto correttamente il pacchetto successivo a quello confermato due volte. Il nostro protocollo di trasferimento dati affidabile e privo di NAK per un canale con errori sui bit è rdt2.2, (Figure 3.13 e 3.14). Una sottile distinzione tra rdt2.1 e rdt2.2 consiste nel fatto che

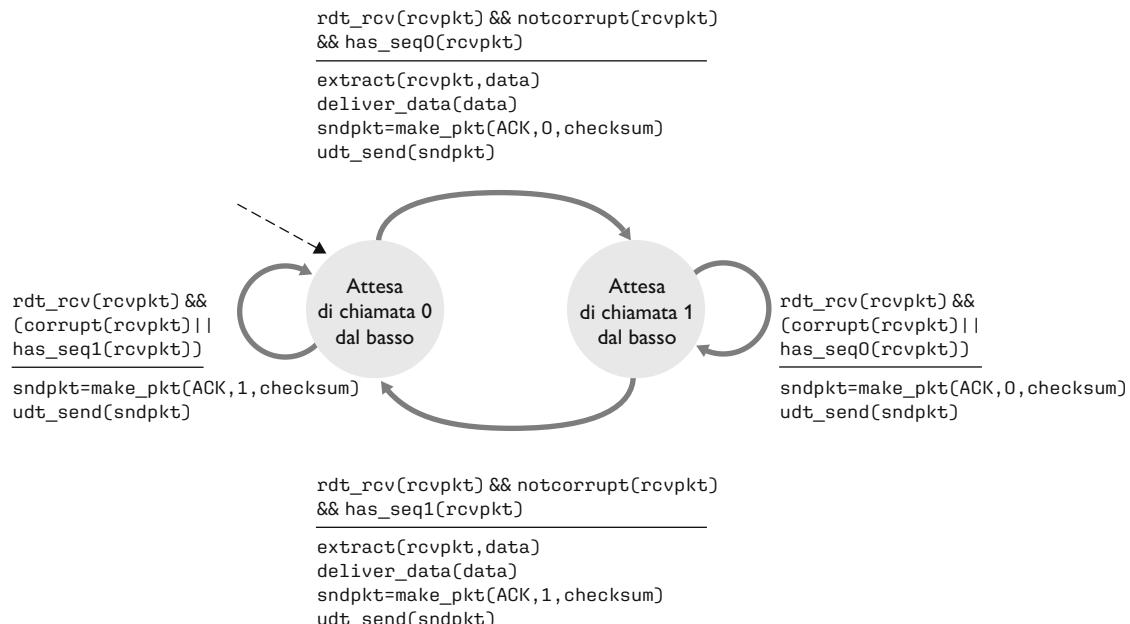
**Figura 3.11** Mittente rdt2.1.**Figura 3.12** Ricevente rdt2.1.

**Figura 3.13** Mittente

rdt2.2.



il destinatario deve ora includere il numero di sequenza del pacchetto di cui invia l'acknowledgment all'interno del messaggio ACK (il che viene effettuato

**Figura 3.14** Ricevente rdt2.2.

includendo un argomento ACK, 0 o ACK, 1 nella funzione `make_pkt()` della FSM del destinatario), e il mittente deve ora controllare il numero di sequenza del pacchetto confermato da un messaggio ACK ricevuto (il che viene effettuato includendo un argomento con valore 0 o 1 nella funzione `isACK()` della FSM del mittente).

### **Trasferimento dati affidabile su un canale con perdite ed errori sui bit: rdt3.0**

Supponiamo ora che il canale di trasmissione, oltre a danneggiare i bit, possa anche *smarrire* i pacchetti, un evento non raro sulle odierne reti di calcolatori (Internet compresa). Il protocollo ora deve preoccuparsi di due aspetti aggiuntivi: come rilevare lo smarrimento di pacchetti e che cosa fare quando ciò avviene. L'utilizzo di checksum, numeri di sequenza, pacchetti ACK e ritrasmissione, tecniche già sviluppate in rdt2.2, ci consentirà di trovare una soluzione per quest'ultimo problema. Per gestire il primo problema, invece, dovremo aggiungere al protocollo un nuovo meccanismo.

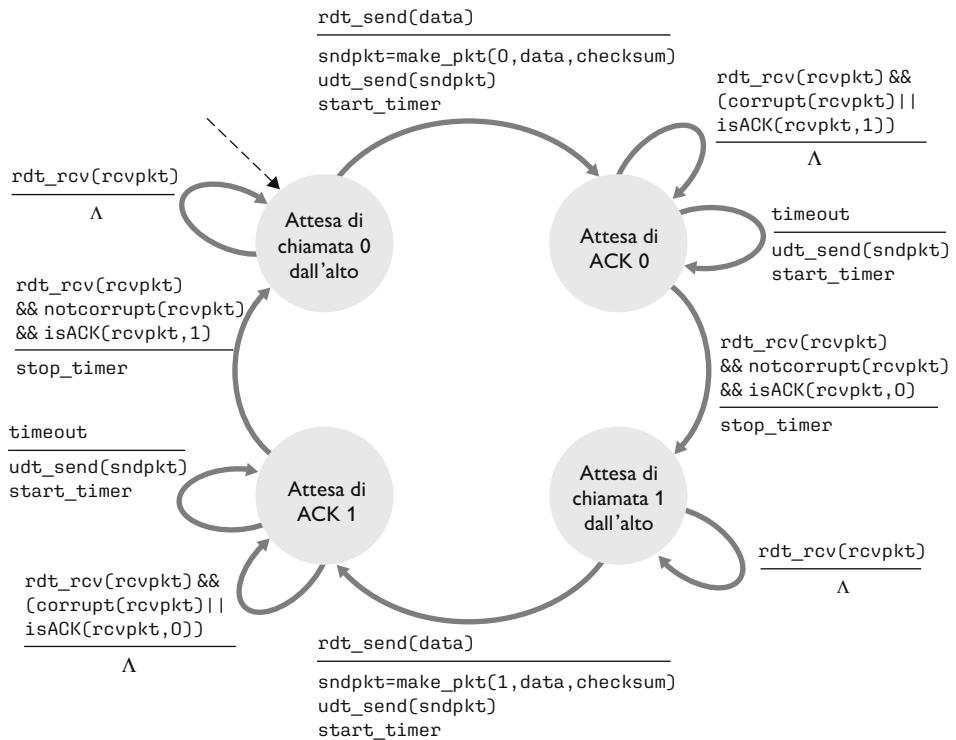
Esistono molti approcci al problema della perdita di pacchetti (parecchi dei quali saranno esplorati nei Problemi alla fine del capitolo). In questa sede, assegneremo al mittente l'onere di rilevare e risolvere la perdita di pacchetti. Supponiamo che il mittente spedisca un pacchetto dati e che questo o l'ACK corrispondente del ricevente vada smarrito. In entrambi i casi, il mittente non otterrà alcuna risposta da parte del destinatario. Se il mittente è disposto ad attendere un tempo sufficiente per essere *certo* dello smarrimento del pacchetto, può semplicemente ritrasmettere il pacchetto di dati.

Ma quanto tempo deve attendere il mittente? Certamente, almeno per il minimo ritardo di andata e ritorno tra mittente e destinatario (il che può includere il tempo di buffering sui router intermedi) più il tempo richiesto per l'elaborazione di un pacchetto da parte del destinatario. In molte reti questo ritardo relativo al caso peggiore è difficile perfino da stimare, oltre che da sapere con certezza. Inoltre, il protocollo dovrebbe ipoteticamente porre rimedio alla perdita di pacchetti non appena possibile: aspettare per il tempo di ritardo del caso peggiore potrebbe tradursi in una lunga attesa prima dell'inizio della risoluzione dell'errore. Di conseguenza, l'approccio adottato nella pratica è scegliere in modo assennato un valore di tempo tale per cui la perdita di pacchetti risulti probabile, anche se non garantita. Se non si riceve un ACK in questo lasso di tempo, il pacchetto viene ritrasmesso. Notiamo che il mittente potrebbe ritrasmettere un pacchetto che sperimenta un ritardo particolarmente lungo, anche se né il pacchetto di dati stesso né il suo ACK sono stati smarriti. Ciò introduce la possibilità di **pacchetti di dati duplicati** (*duplicate data packets*) sul canale tra mittente e destinatario. Fortunatamente, il protocollo rdt2.2 presenta già una funzionalità (i numeri di sequenza) per gestire il caso dei pacchetti duplicati.

Dal punto di vista del mittente, la ritrasmissione è una panacea. Il mittente non sa se un pacchetto dati sia andato perduto, se sia stato smarrito un ACK o se il pacchetto o l'ACK abbiano semplicemente subito un notevole ritardo. In tutti questi casi, l'azione intrapresa è la stessa: ritrasmettere. Implementare un meccanismo di ritrasmissione basato sul tempo richiede un **contatore** (*count-down timer*) in grado di segnalare al mittente l'avvenuta scadenza di un dato lasso di tempo. Il mittente dovrà quindi essere in grado (1) di inizializzare il

**Figura 3.15** Mittente

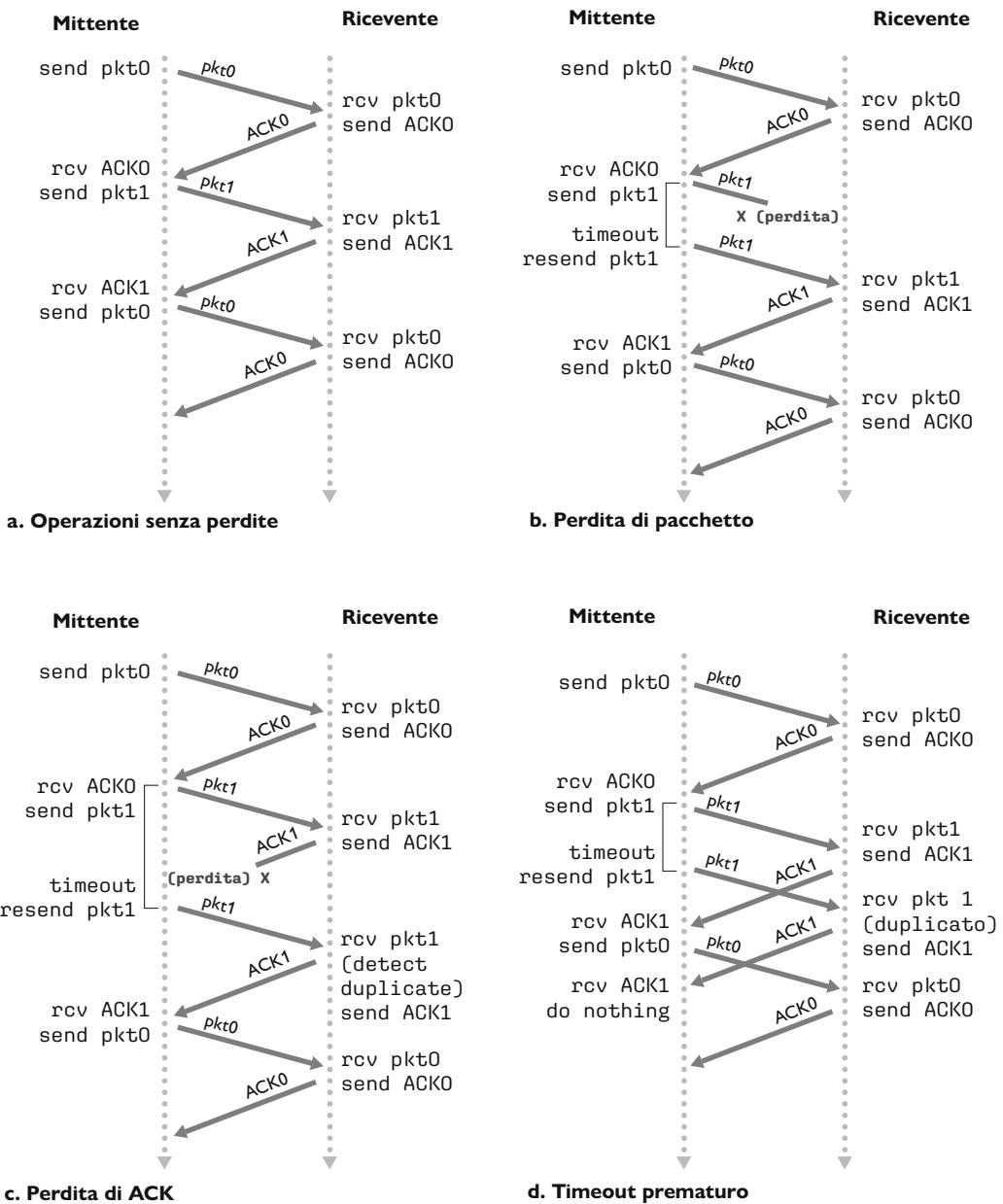
rdt3.0.



contatore ogni volta che invia un pacchetto (che si tratti del primo invio o di una ritrasmissione), (2) di rispondere a un interrupt generato dal timer con l’azione appropriata e (3) di fermare il contatore.

La Figura 3.15 mostra la FSM del mittente in rdt3.0, un protocollo che trasferisce in modo affidabile i dati su un canale che può alterare o perdere pacchetti; in un problema a fine capitolo vi verrà chiesto di realizzare la FSM del destinatario di rdt3.0. La Figura 3.16 mostra come il protocollo operi senza pacchetti smarriti o in ritardo e come gestisca i pacchetti di dati persi. Nella Figura 3.16 il tempo procede dall’alto verso il basso del diagramma; notiamo che il momento di ricezione di un pacchetto è necessariamente successivo all’istante del suo invio, a causa dei ritardi di trasmissione e propagazione. Nella Figura 3.16, nelle parti (b), (c) e (d), la parentesi quadra lato mittente indica gli istanti in cui il contatore viene impostato e in cui scade. Altri aspetti più sottili di questo protocollo vengono indagati nei problemi alla fine del capitolo. Dato che i numeri di sequenza dei pacchetti si alternano tra 0 e 1, il protocollo rdt3.0 viene talvolta detto **protocollo ad alternanza di bit**.

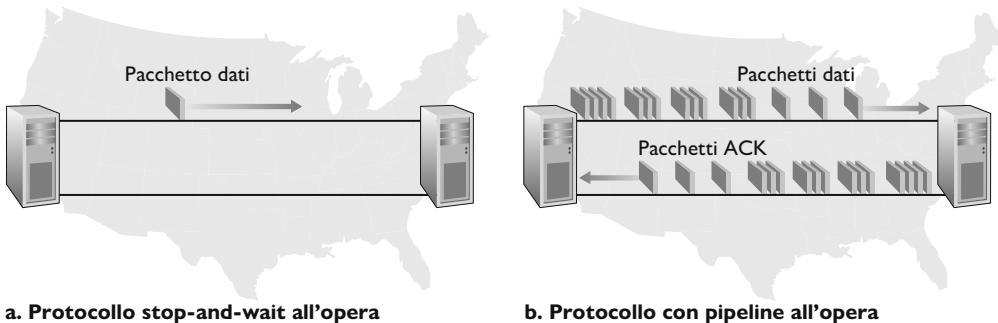
Abbiamo quindi assemblato gli elementi chiave di un protocollo di trasferimento dati. Checksum, numeri di sequenza, contatori e pacchetti di notifica positiva e negativa giocano tutti un ruolo cruciale e necessario per il funzionamento del protocollo. A questo punto abbiamo a disposizione un protocollo funzionante per il trasferimento dati affidabile.



**Figura 3.16** Operazioni di rdt3.0, il protocollo ad alternanza di bit.

### 3.4.2 Protocolli per il trasferimento dati affidabile con pipeline

Il protocollo rdt3.0 è corretto dal punto di vista funzionale, ma difficilmente qualcuno apprezzerebbe le sue prestazioni, in particolare nelle odierne reti ad alta velocità. Il problema delle prestazioni risiede nel fatto che si tratta di un protocollo stop-and-wait.



**Figura 3.17** Confronto tra protocolli stop-and-wait e con pipeline.

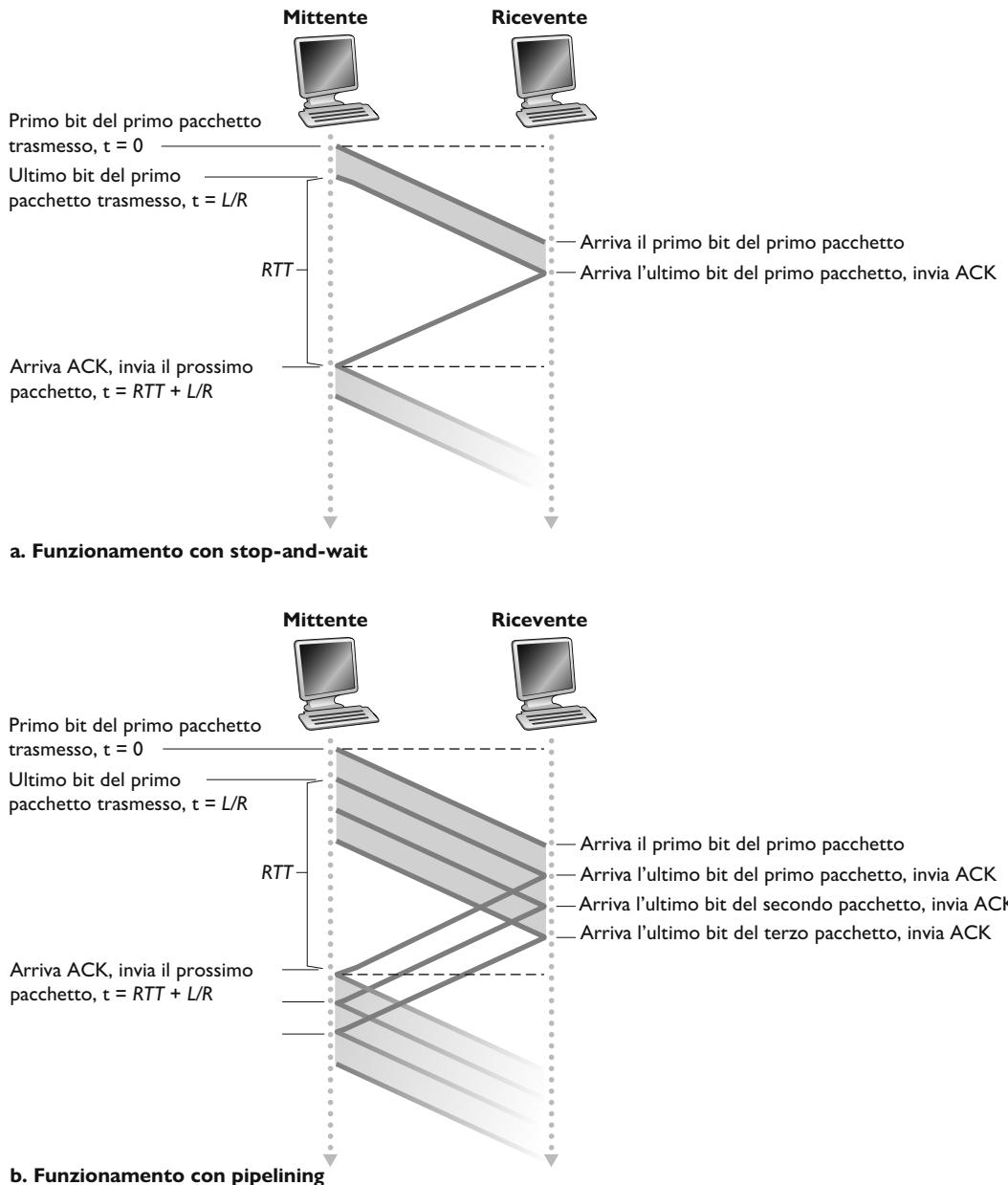
Per valutare l'impatto delle prestazioni, consideriamo il caso ideale di due host, uno sulla costa occidentale degli Stati Uniti e l'altro sulla costa orientale (Figura 3.17). Il ritardo di propagazione di andata e ritorno (*RTT*) alla velocità della luce per questi due sistemi è approssimativamente di 30 millisecondi. Supponiamo che i due sistemi siano connessi da un canale con velocità di trasmissione  $R$  di 1 Gbps ( $10^9$  bit al secondo). Con pacchetti di dimensione  $L$  di 1000 byte (8000 bit) inclusi campi di intestazione e dati, il tempo effettivamente richiesto per trasmettere il pacchetto sul collegamento è:

$$d_t = \frac{L}{R} = \frac{8000 \text{ bit}}{10^9 \text{ bit/s}} = 8 \text{ microsecondi}$$

La Figura 3.18(a) mostra che nel nostro protocollo stop-and-wait, se il mittente comincia a inviare pacchetti a  $t = 0$ , l'ultimo bit entra nel canale lato mittente al tempo  $t = L/R = 8 \mu\text{s}$ . Il pacchetto effettua quindi un viaggio di 15 ms attraverso il continente, e l'ultimo bit del pacchetto giunge al destinatario all'istante  $t = RTT/2 + L/R = 15.008 \text{ ms}$ . Assumendo per semplicità che i pacchetti ACK siano estremamente piccoli (e, pertanto, il loro tempo di trasmissione sia trascurabile) e che il destinatario possa spedire un ACK non appena venga ricevuto l'ultimo bit di un pacchetto di dati, l'ACK giunge al mittente all'istante  $t = RTT/2 + L/R + RTT/2 = 30.008 \text{ ms}$ . A questo punto, il mittente può trasmettere il successivo messaggio. Quindi, in un arco di 30,008 ms, il mittente ha trasmesso solo per 0,008 ms. Se definiamo l'**utilizzo** (*utilization*) del mittente (o del canale) come la frazione di tempo in cui il mittente è stato effettivamente occupato nell'invio di bit sul canale, l'analisi della Figura 3.18(a) mostra che il protocollo stop-and-wait presenta un triste utilizzo del mittente,  $U_{\text{mittente}}$ , pari a

$$U_{\text{mittente}} = \frac{L/R}{RTT + L/R} = \frac{0,008}{30,008} = 0,00027$$

In altre parole, il mittente è stato attivo per soli 2,7 centesimi dell'1% del tempo. Visto in altro modo, il mittente è stato in grado di spedire solo 1000 byte in 30,008 ms, con un throughput effettivo di soli 267 kbps, nonostante fosse disponibile un collegamento da 1 Gbps. Immaginate la rabbia del gestore della rete che ha appena pagato una fortuna per un collegamento da 1 Gbps, ma che non riesce a ottenere un throughput maggiore di 267 kbps! Questo è un esempio



**Figura 3.18** Invio per il protocollo stop-and-wait e con pipelining.

pratico di come i protocolli di rete possano limitare il rendimento dell'hardware di rete sottostante. Inoltre abbiamo trascurato i tempi di elaborazione del protocollo ai livelli inferiori presso il mittente e il destinatario, così come i ritardi di elaborazione e di accodamento che si verificherebbero sui router intermedi tra il mittente e il destinatario. Considerare questi effetti non farebbe altro che incrementare ulteriormente il ritardo, accentuando così le prestazioni scadenti.

La soluzione a questo particolare problema è semplice: anziché operare in modalità stop-and-wait, si consente al mittente di inviare più pacchetti senza attendere gli acknowledgment, come mostrato nella Figura 3.17(b). La Figura 3.18(b) illustra che, se si consente al mittente di trasmettere tre pacchetti senza dover aspettare gli acknowledgment, l'utilizzo viene sostanzialmente triplicato. Dato che molti pacchetti in transito dal mittente al destinatario possono essere visualizzati come il riempimento di una tubatura, questa tecnica è nota come **pipelining** (da *pipe*, “tubo”). Le conseguenze su un protocollo di trasferimento dati affidabile sono le seguenti.

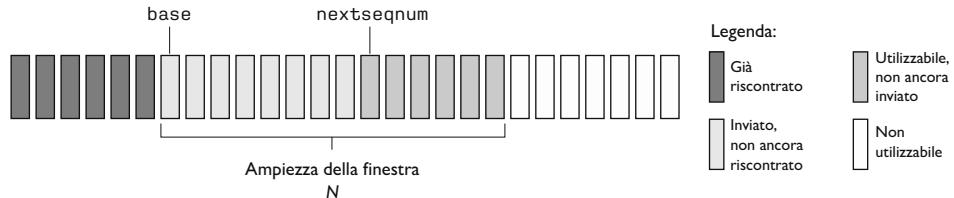
- L'intervallo dei numeri di sequenza disponibili deve essere incrementato, dato che ogni pacchetto in transito (senza contare le ritrasmissioni) deve presentare un numero di sequenza univoco e che ci potrebbero essere più pacchetti in transito ancora in attesa di acknowledgment.
- I lati di invio e di ricezione dei protocolli possono dover memorizzare in un buffer più di un pacchetto. Quantomeno, il mittente dovrà memorizzare i pacchetti trasmessi, ma il cui acknowledgment non è ancora stato ricevuto. Inoltre, come vedremo più avanti, anche al destinatario potrebbe essere richiesta la memorizzazione dei pacchetti ricevuti correttamente.
- La quantità di numeri di sequenza necessari e i requisiti di buffer dipendono dal modo in cui il protocollo di trasferimento dati reagisce ai pacchetti smarriti, alterati o troppo in ritardo. Si possono identificare due approcci di base verso la risoluzione degli errori con pipeline: **Go-Back-N** e **ripetizione selettiva** (*selective repeat*).

### 3.4.3 Go-Back-N (GBN)

In un **protocollo Go-Back-N (GBN)** il mittente può trasmettere più pacchetti senza dover attendere alcun acknowledgment, ma non può avere più di un dato numero massimo consentito  $N$  di pacchetti (se disponibili) in attesa di acknowledgment nella pipeline. Descriveremo il protocollo GBN con alcuni dettagli in questo paragrafo, ma prima di procedere siete invitati a provare l'animazione interattiva GBN presente sulla piattaforma MyLab di questo libro.

La Figura 3.19 mostra la visione del mittente sull'intervallo di numeri di sequenza in un protocollo GBN. Se definiamo **base** come il numero di sequenza del pacchetto più vecchio che non ha ancora ricevuto un acknowledgment e **nextseqnum** il più piccolo numero di sequenza inutilizzato (ossia il numero di sequenza del prossimo pacchetto da inviare), allora si possono identificare quattro intervalli di numeri di sequenza. I numeri di sequenza nell'intervallo

**Figura 3.19** Visione del mittente sui numeri di sequenza nel protocollo Go-Back-N.



$[0, \text{base}-1]$  corrispondono ai pacchetti già trasmessi e che hanno ricevuto acknowledgment. L'intervallo  $[\text{base}, \text{nextseqnum}-1]$  corrisponde ai pacchetti inviati, ma che non hanno ancora ricevuto alcun acknowledgment. I numeri di sequenza nell'intervallo  $[\text{nextseqnum}, \text{base}+N-1]$  possono essere utilizzati per i pacchetti da inviare immediatamente, nel caso arrivassero dati dal livello superiore. Infine, i numeri di sequenza maggiori o uguali a  $\text{base}+N$  non possono essere utilizzati finché il mittente non riceva un acknowledgment relativo a un pacchetto che si trova nella pipeline ed è ancora privo di acknowledgment (nello specifico, il pacchetto con il numero di sequenza uguale a  $\text{base}$ ).

Come suggerito dalla Figura 3.19, l'intervallo di numeri di sequenza ammissibili per i pacchetti trasmessi, ma che non hanno ancora ricevuto alcun acknowledgment, può essere visto come una finestra di dimensione  $N$  sull'intervallo dei numeri di sequenza. Quando il protocollo è in funzione, questa finestra trasla lungo lo spazio dei numeri di sequenza. Per questo motivo,  $N$  viene spesso chiamato **ampiezza della finestra** (*window size*) e il protocollo GBN viene detto **protocollo a finestra scorrevole** (*sliding-window protocol*). Ci si potrebbe chiedere per quale motivo dovremmo limitare il numero di pacchetti in sospeso a  $N$ ; perché non consentire un numero illimitato di tali pacchetti? Vedremo nel corso del Paragrafo 3.5 che il controllo di flusso rappresenta una delle ragioni per imporre un limite al mittente. Esamineremo un altro motivo nel Paragrafo 3.7, quando affronteremo il controllo di congestione di TCP.

In pratica, il numero di sequenza di un pacchetto viene scritto in un campo a dimensione fissa dell'intestazione del pacchetto. Detto  $k$  il numero di bit di tale campo, l'intervallo di possibili numeri di sequenza è  $[0, 2^k - 1]$ . Avendo un intervallo finito di numeri di sequenza, tutte le operazioni aritmetiche che coinvolgono i numeri di sequenza devono essere effettuate in modulo  $2^k$ . In altre parole, lo spazio dei numeri di sequenza può essere pensato come un insieme ciclico di  $2^k$  elementi in cui il numero di sequenza  $2^k - 1$  sia immediatamente seguito da 0. Ricordiamo che rdt3.0 aveva un numero di sequenza a un bit e numeri di sequenza 0 o 1. Diversi Problemi alla fine di questo capitolo indagano le conseguenze di un intervallo finito di numeri di sequenza. Vedremo nel corso del Paragrafo 3.5 che TCP ha un campo a 32 bit per i numeri di sequenza, e che i numeri di sequenza TCP contano i byte nel flusso dei dati anziché i pacchetti.

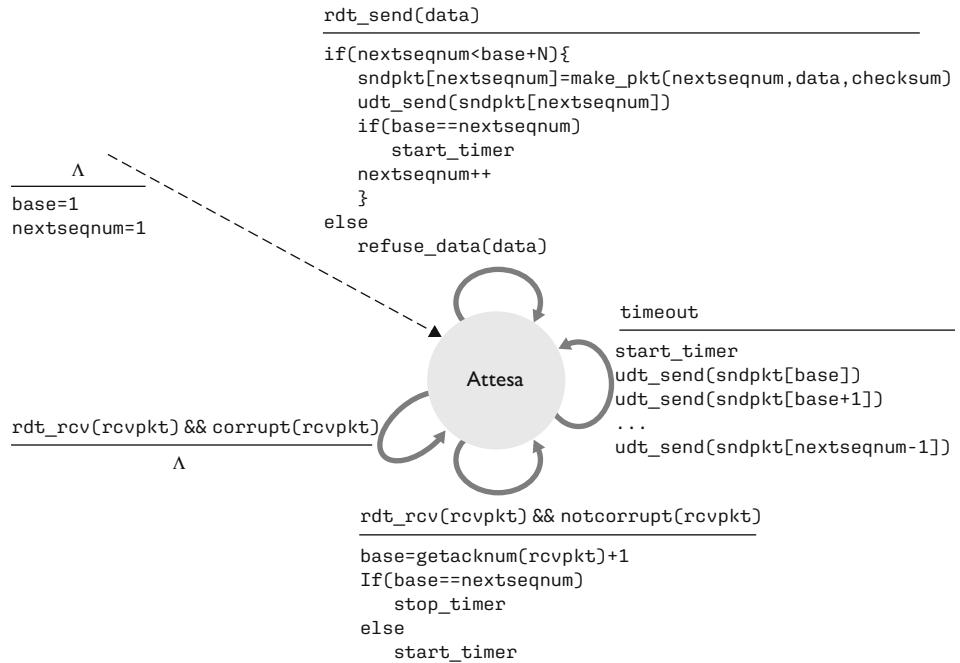
Le Figure 3.20 e 3.21 forniscono una descrizione estesa delle FSM per mittente e ricevente di un protocollo GBN basato su ACK e privo di NAK. Facciamo riferimento a queste FSM come FSM estese, perché abbiamo aggiunto delle variabili (simili a quelle dei linguaggi di programmazione) per base e nextseqnum, oltre che operazioni su queste variabili e azioni condizionali che le riguardano. Notiamo che la descrizione tramite FSM estese comincia ad assomigliare a quella di un linguaggio di programmazione. [Bochman 1984] presenta un eccellente saggio sulle tecniche di estensione delle FSM e su altre basate su linguaggi di programmazione per le specifiche dei protocolli.

Il mittente GBN deve rispondere a tre tipi di evento.

- *Invocazione dall'alto.* Quando dall'alto si chiama `rdt_send()`, come prima cosa il mittente controlla se la finestra sia piena, ossia se vi siano  $N$  pacchetti in sospeso senza acknowledgment. Se la finestra non è piena, crea e invia un pacchetto e le variabili vengono aggiornate di conseguenza. Se la finestra è

**Figura 3.20**

Descrizione con automa esteso del mittente GBN.

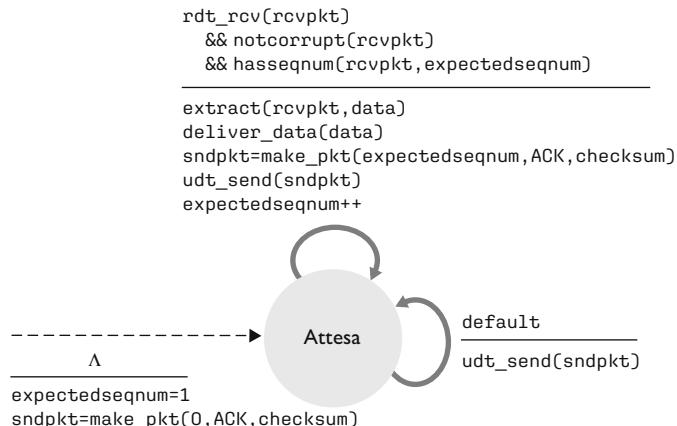


piena, il mittente restituisce i dati al livello superiore che, presumibilmente, ritenterà più tardi. In una reale implementazione è più probabile che il mittente mantenga questo dato nei buffer (pronto all'invio immediato) o implementi un meccanismo di sincronizzazione (per esempio, un semaforo o un flag) che consenta al livello superiore di invocare `rdt_send()` solo quando la finestra non sia piena.

- *Ricezione di un ACK.* Nel nostro protocollo GBN, l'acknowledgment del pacchetto con il numero di sequenza  $n$  verrà considerato un **acknowledgment cumulativo** (*cumulative acknowledgment*), che indica che tutti i pacchetti con un numero di sequenza minore o uguale a  $n$  sono stati correttamente ricevuti

**Figura 3.21**

Descrizione con automa esteso del ricevente GBN.



dal destinatario. Torneremo sull'argomento tra breve, quando esamineremo GBN sul lato ricevente.

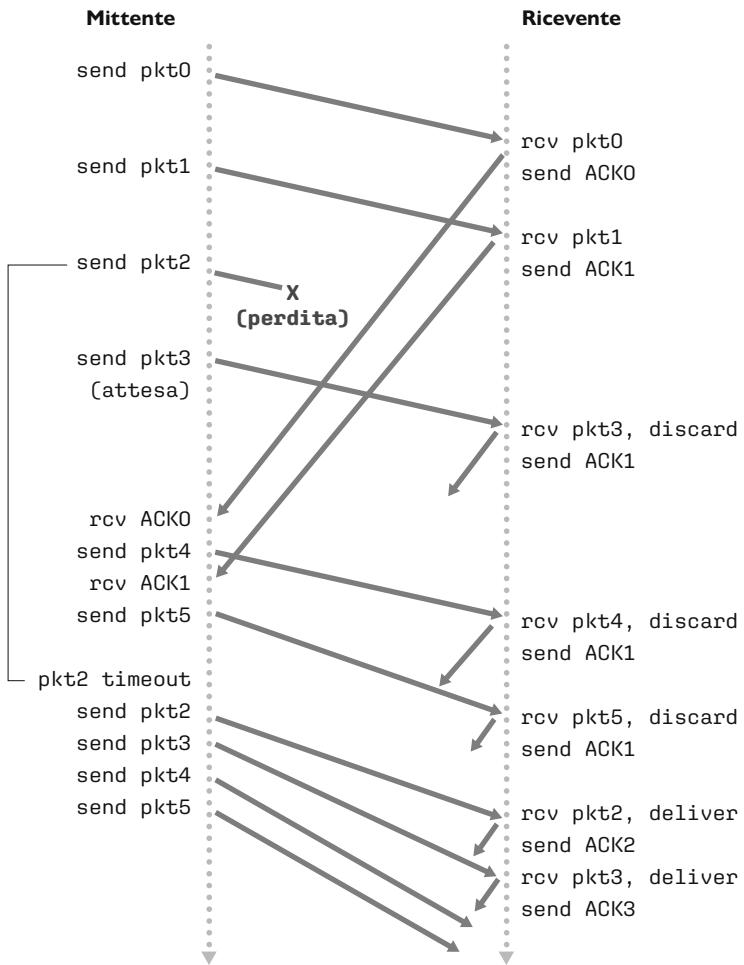
- *Evento di timeout.* Il nome del protocollo “Go-Back-N” deriva dal comportamento del mittente in presenza di pacchetti persi o eccessivamente in ritardo. Come nei protocolli stop-and-wait, si usa ancora un contatore per risolvere il problema di pacchetti dati o acknowledgment persi. Quando si verifica un timeout (tempo scaduto), il mittente invia nuovamente *tutti* i pacchetti spediti che ancora non hanno ricevuto un acknowledgment. Il mittente nella Figura 3.20 usa un singolo contatore che può essere pensato come un timer per il pacchetto trasmesso meno di recente per cui non sia ancora stato ricevuto un acknowledgment. Se si riceve un ACK, ma ci sono ancora pacchetti aggiuntivi trasmessi e non riscontrati, il timer viene fatto ripartire. Se, invece, non ci sono pacchetti in sospeso in attesa di acknowledgment, il contatore viene fermato.

Anche le azioni del destinatario GBN sono semplici. Se un pacchetto con numero di sequenza  $n$  viene ricevuto correttamente ed è in ordine (ossia, gli ultimi dati consegnati al livello superiore provengono da un pacchetto con numero di sequenza  $n - 1$ ), il destinatario manda un ACK per quel pacchetto e consegna i suoi dati al livello superiore. In tutti gli altri casi, il destinatario scarta i pacchetti e rimanda un ACK per il pacchetto in ordine ricevuto più di recente. Notiamo che, essendo i pacchetti consegnati uno alla volta al livello superiore, se il pacchetto  $k$  è stato ricevuto e consegnato, tutti i pacchetti con un numero di sequenza inferiore sono anch’essi stati consegnati. Pertanto l’uso di acknowledgment cumulativi è una scelta naturale per GBN.

Nel nostro protocollo GBN il destinatario scarta i pacchetti fuori sequenza. Sebbene possa sembrare sciocco e inutile eliminare un pacchetto ricevuto correttamente, ma non nella giusta sequenza, esiste una buona ragione per farlo. Ricordiamo che il destinatario deve fornire i dati in modo ordinato al livello superiore. Supponiamo che sia atteso il pacchetto  $n$ , ma che arrivi il pacchetto  $n + 1$ . Visto che i dati devono essere consegnati in ordine, il destinatario potrebbe mettere in un buffer il pacchetto  $n + 1$  e quindi consegnarlo al livello superiore solo dopo la ricezione e la consegna del pacchetto  $n$ . Tuttavia, se il pacchetto  $n$  va perduto, sia quest’ultimo sia il pacchetto  $n + 1$  verranno ritrasmessi per via della regola di ritrasmissione GBN. Di conseguenza, il destinatario può semplicemente scartare il pacchetto  $n + 1$ . Il vantaggio di questo approccio è la semplicità: il destinatario non deve memorizzare nel buffer i pacchetti che giungono fuori sequenza. Quindi, mentre il mittente deve mantenere i limiti superiore e inferiore della propria finestra e la posizione di nextseqnum all’interno di tale finestra, l’unica parte delle informazioni che il destinatario deve memorizzare è il numero di sequenza del successivo pacchetto nell’ordine. Questo valore viene salvato nella variabile expectedseqnum, mostrata nella FSM della Figura 3.21. Ovviamente lo svantaggio di eliminare un pacchetto ricevuto correttamente è che la sua ritrasmissione potrebbe andare persa o essere alterata, il che richiederebbe ulteriori ritrasmissioni.

La Figura 3.22 mostra come opera il protocollo GBN con una finestra di 4 pacchetti. A causa dei limiti dell’ampiezza della finestra, il mittente invia i pacchetti da 0 a 3, ma poi deve attendere la notifica di ricezione di uno di loro prima

**Figura 3.22** Go-Back-N  
in funzione.



di poter procedere. Quando giungono i successivi ACK (per esempio, ACK0 e ACK1), la finestra scorre in avanti e il mittente può trasmettere un nuovo pacchetto (rispettivamente, pkt4 e pkt5). Lato ricevente, il pacchetto 2 viene perso e pertanto i pacchetti 3, 4 e 5 non rispettano l'ordine e sono scartati.

Prima di concludere la nostra trattazione di GBN, vale la pena notare che un'implementazione di questo protocollo in una pila di protocolli avrebbe probabilmente una struttura simile a quella della FSM estesa della Figura 3.20 e comprenderebbe diverse procedure che stabiliscono le azioni da intraprendere in risposta agli eventi che man mano si verificano. In una tale **programmazione basata su eventi** (*event-based programming*) le diverse procedure vengonoificate da altre procedure nella pila di protocolli o in conseguenza di un interrupt. Nel mittente, questi eventi sarebbero (1) una chiamata dall'entità del livello superiore per invocare `rdt_send()`, (2) un interrupt dovuto al timer e (3) una chiamata dal livello inferiore per invocare `rdt_rcv()` quando giunge un pacchetto. Gli Esercizi di programmazione alla fine del capitolo vi daranno la

possibilità di implementare effettivamente queste routine in uno scenario di rete simulato, ma realistico.

Notiamo che il protocollo GBN incorpora quasi tutte le tecniche che incontreremo nello studio dei componenti TCP per il trasferimento dati affidabile (Paragrafo 3.5). Tali tecniche includono l'uso di numeri di sequenza, riscontri cumulativi, checksum e operazioni di timeout/ritrasmissione.

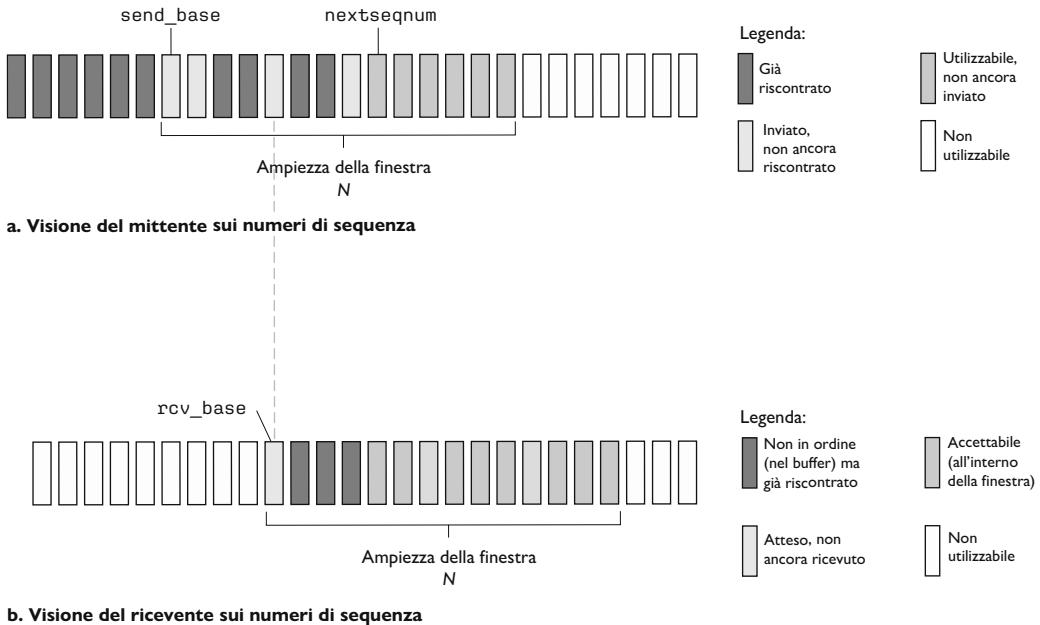
### 3.4.4 Ripetizione selettiva

Potenzialmente, il protocollo GBN consente al mittente di “riempire la tubatura” della Figura 3.17 con pacchetti, evitando così i problemi di scarso utilizzo del canale che abbiamo riscontrato nei protocolli stop-and-wait. Esistono, tuttavia, scenari in cui lo stesso GBN ha problemi di prestazioni. In particolare, quando l'ampiezza della finestra e il prodotto tra larghezza di banda e ritardo sono entrambi grandi, nella pipeline si possono trovare numerosi pacchetti. Un errore su un solo pacchetto può pertanto provocare un elevato numero di ritrasmissioni, in molti casi inutili. Al crescere della probabilità di errore sul canale, la pipeline può saturarsi a causa di queste ritrasmissioni non necessarie. Immaginiamo, nel nostro scenario di dettatura del messaggio, di dover ripetere 1000 parole (questa è l'ampiezza della nostra finestra) ogni volta che si verifica un errore su una singola parola. La dettatura risulterebbe molto rallentata.

Come suggerisce il nome, i **protocolli a ripetizione selettiva** (SR, *Selective-Repeat Protocol*) evitano le ritrasmissioni non necessarie facendo ritrasmettere al mittente solo quei pacchetti su cui esistono sospetti di errore (ossia, smarimento o alterazione). Questa forma di ritrasmissione a richiesta e personalizzata costringe il destinatario a mandare acknowledgment specifici per i pacchetti ricevuti in modo corretto. Si userà nuovamente un'ampiezza di finestra pari a  $N$  per limitare il numero di pacchetti privi di acknowledgment nella pipeline. Tuttavia, a differenza di GBN, il mittente avrà già ricevuto gli ACK di qualche pacchetto nella finestra. La Figura 3.23 mostra la visione del mittente SR dello spazio dei numeri di sequenza; la Figura 3.24 illustra in dettaglio le azioni intraprese dal mittente SR.

Il destinatario SR invia un acknowledgment per i pacchetti correttamente ricevuti sia in ordine sia fuori sequenza. Questi vengono memorizzati in un buffer finché non sono stati ricevuti tutti i pacchetti mancanti (ossia quelli con numeri di sequenza più bassi), momento in cui un blocco di pacchetti può essere trasportato in ordine al livello superiore. La Figura 3.25 organizza per punti le diverse azioni intraprese dal destinatario SR. La Figura 3.26 mostra un esempio di comportamento SR in presenza di pacchetti persi. Notiamo che in quest'ultima figura inizialmente il destinatario memorizza i pacchetti 3, 4 e 5 nel buffer e li consegna al livello superiore insieme al pacchetto 2 quando quest'ultimo viene finalmente ricevuto.

È importante notare che al punto 2 della Figura 3.25 il destinatario spedisce nuovamente un acknowledgment (anziché ignorare) dei pacchetti già ricevuti con certi numeri di sequenza *al di sotto* dell'attuale base della finestra. Questa nuova notifica è necessaria; considerando l'ambito dei numeri di sequenza del mittente e del destinatario nella Figura 3.23 se, per esempio, tra destinatario e mittente non si propaga un ACK per il pacchetto con numero di sequenza



**Figura 3.23** Visione del mittente e del ricevente a ripetizione selettiva sullo spazio dei numeri di sequenza.

`send_base`, il mittente potrebbe ritrasmetterlo anche se è chiaro (a noi, non al mittente) che il destinatario lo ha già ricevuto. Se il destinatario non invia un acknowledgement per questo pacchetto, la finestra del mittente non può avanzare. Questo esempio mostra un importante aspetto dei protocolli SR (e anche di molti altri): non sempre mittente e destinatario hanno la stessa visuale su che cosa sia stato ricevuto correttamente. Nei protocolli SR ciò significa che le finestre del mittente e del destinatario non sempre coincidono.

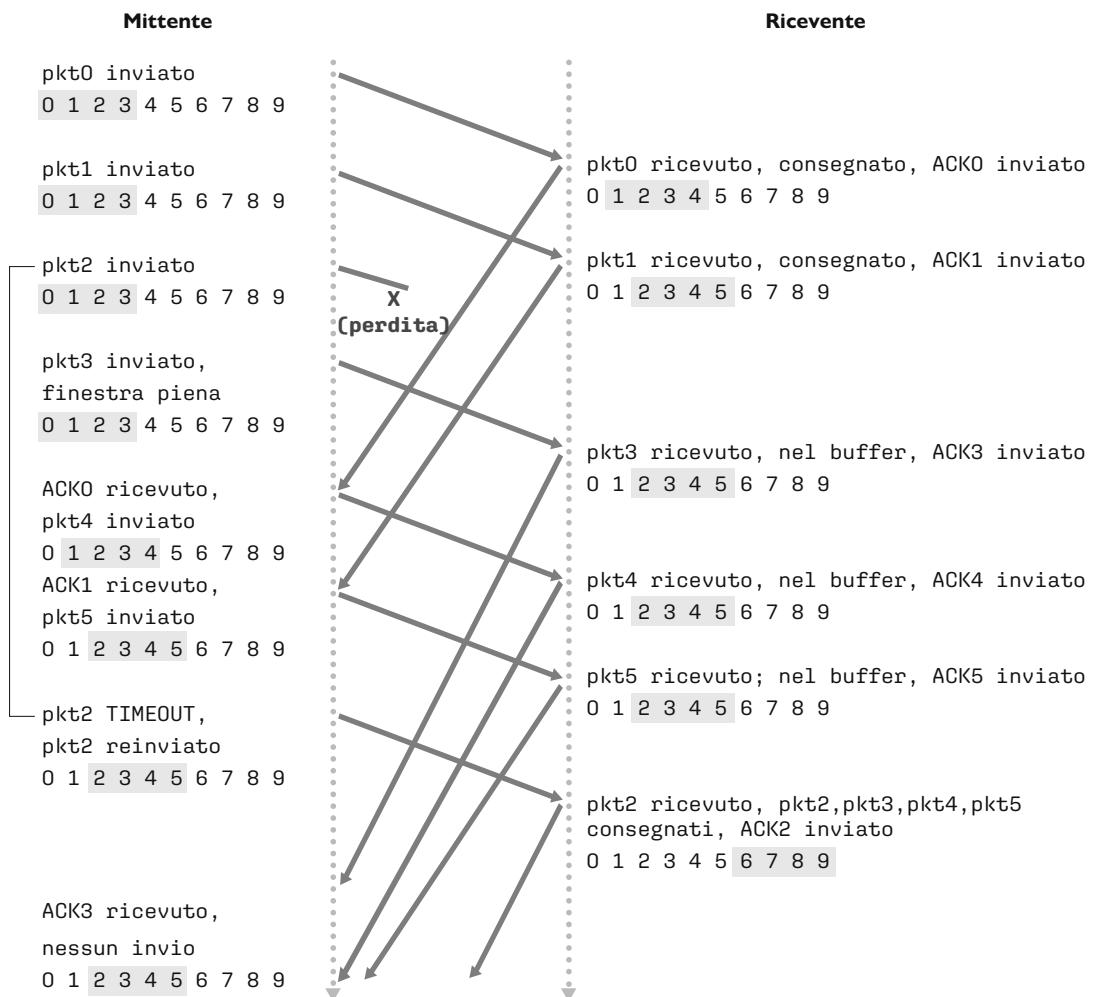
La mancanza di sincronizzazione tra le finestre del mittente e del destinatario ha conseguenze importanti quando abbiamo a che fare con un intervallo finito di numeri di sequenza. Consideriamo che cosa succederebbe, per esempio, con un intervallo di quattro numeri di sequenza per i pacchetti 0, 1, 2 e 3, e un'ampiezza di finestra pari a 3.

- |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ol style="list-style-type: none"> <li>1. <i>Dati ricevuti dall'alto.</i> Quando si ricevono dati dall'alto, il mittente SR controlla il successivo numero di sequenza disponibile per il pacchetto. Se è all'interno della finestra del mittente, i dati vengono impacchettati e inviati; altrimenti sono salvati nei buffer o restituiti al livello superiore per una successiva ritrasmissione, come in GBN.</li> </ol>                                                                                                                         |
| <ol style="list-style-type: none"> <li>2. <i>Timeout.</i> Vengono usati ancora i contatori per cautelarsi contro la perdita di pacchetti. Ora però ogni pacchetto deve avere un proprio timer logico, dato che al timeout sarà ritrasmesso un solo pacchetto. Si può utilizzare un solo contatore hardware per simulare le operazioni di più timer logici [Varghese 1997].</li> </ol>                                                                                                                                                              |
| <ol style="list-style-type: none"> <li>3. <i>ACK ricevuto.</i> Se si riceve un ACK, il mittente SR etichetta tale pacchetto come ricevuto, ammesso che sia nella finestra. Se il numero di sequenza del pacchetto è uguale a <code>send_base</code>, la base della finestra si muove verso il pacchetto che non ha ricevuto acknowledgement con il più piccolo numero di sequenza. Se la finestra si sposta e ci sono pacchetti non trasmessi con numero di sequenza che ora cade all'interno della finestra, questi vengono trasmessi.</li> </ol> |

**Figura 3.24** Eventi e azioni di un mittente SR.

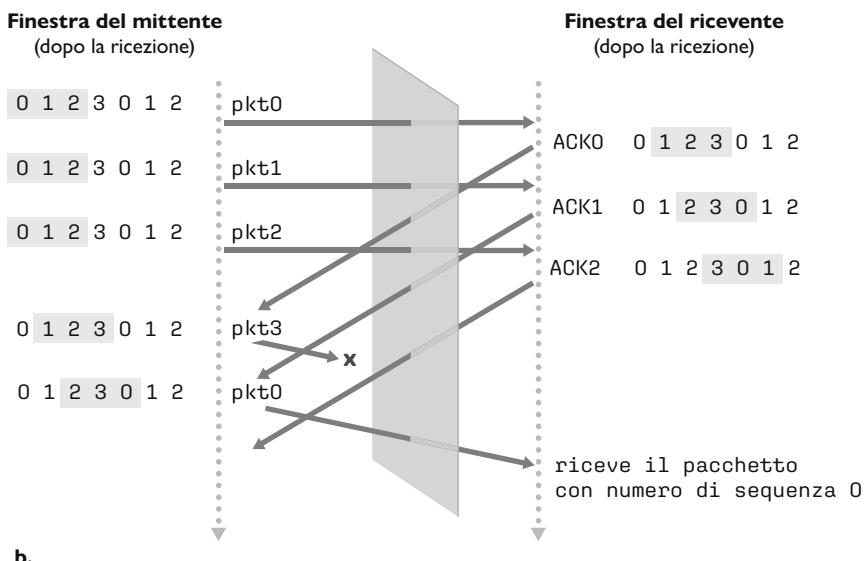
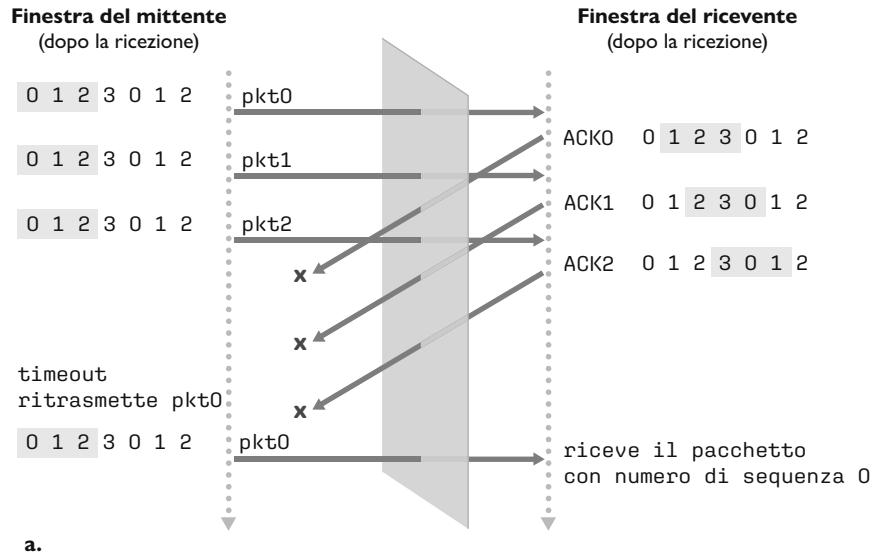
- 1.** Il pacchetto con numero di sequenza nell'intervallo  $[rcv\_base, rcv\_base+N-1]$  viene ricevuto correttamente. In questo caso, il pacchetto ricevuto ricade all'interno della finestra del ricevente e al mittente viene restituito un pacchetto di ACK selettivo. Se il pacchetto non era già stato ricevuto viene inserito nel buffer. Se presenta un numero di sequenza uguale alla base della finestra di ricezione ( $rcv\_base$  nella Figura 3.22), allora questo pacchetto e tutti i pacchetti nel buffer aventi numeri consecutivi (a partire da  $rcv\_base$ ) vengono consegnati al livello superiore. Per un esempio, consideriamo la Figura 3.26. Quando si riceve un pacchetto con numero di sequenza  $rcv\_base = 2$ , è possibile consegnarlo al livello superiore insieme ai pacchetti 3, 4 e 5.
- 2.** Viene ricevuto il pacchetto con numero di sequenza nell'intervallo  $[rcv\_base-N, rcv\_base-1]$ . In questo caso si deve generare un ACK, anche se si tratta di un pacchetto che il ricevente ha già riscontrato.
- 3.** Altrimenti si ignora il pacchetto.

**Figura 3.25** Eventi e azioni di un ricevente SR.



**Figura 3.26** Funzionamento di SR.

**Figura 3.27** Dilemma del ricevente SR con finestre troppo grandi: nuovo pacchetto o ritrasmissione?



Supponiamo che i pacchetti 0, 1 e 2 vengano trasmessi e ricevuti correttamente e che il destinatario invii gli ACK. A questo punto, la finestra del destinatario si sposta sul quarto, quinto e sesto pacchetto, che presentano rispettivamente numeri di sequenza 3, 0 e 1. Consideriamo ora due scenari. Nel primo, Figura 3.27(a), gli ACK dei primi tre pacchetti vanno persi e il mittente ritrasmette i pacchetti. Il destinatario riceve quindi un pacchetto con numero di sequenza 0, copia del primo pacchetto inviato.

Nel secondo scenario, Figura 3.27(b), gli ACK dei primi tre pacchetti vengono tutti consegnati correttamente. Il mittente, di conseguenza, sposta in avan-

ti la propria finestra e spedisce il quarto, il quinto e il sesto pacchetto, con numeri di sequenza rispettivamente pari a 3, 0 e 1. Il pacchetto con numero di sequenza 3 va perso, ma arriva il pacchetto con numero di sequenza 0, che contiene nuovi dati. Consideriamo ora il punto di vista del destinatario presentato nella Figura 3.27, che pone una cortina immaginaria tra mittente e destinatario, dato che quest'ultimo non può vedere le azioni intraprese dal primo. Tutto ciò che il destinatario osserva è la sequenza di messaggi ricevuti dal canale e che esso stesso manda nel canale. Per quanto lo riguarda, i due scenari della Figura 3.27 sono identici. Non esiste alcun modo per distinguere la ritrasmissione del primo pacchetto dalla trasmissione originaria del quinto. Chiaramente, un'ampiezza di finestra inferiore di uno rispetto a quella dello spazio dei numeri di sequenza non funziona. Ma quanto deve essere piccola la finestra? In uno dei Problemi alla fine del capitolo vi verrà chiesto di dimostrare che la finestra deve avere ampiezza inferiore o uguale alla metà dello spazio dei numeri di sequenza dei protocolli SR.

Sulla piattaforma MyLab del volume trovate un'animazione che mostra il funzionamento di un protocollo SR. Provate a rifare gli stessi esperimenti che avete eseguito con l'animazione su GBN e valutate se i risultati sono quelli che vi aspettavate.

Questo completa la nostra trattazione sui protocolli per il trasferimento dati affidabile. Abbiamo percorso molta strada e introdotto numerosi meccanismi, riassunti nella Tabella 3.1. Ora che abbiamo analizzato il funzionamento di questi strumenti e riusciamo a vederne il quadro d'insieme, vi incoraggiamo a rileg-

**Tabella 3.1** Riepilogo dei meccanismi di trasferimento dati affidabile e loro utilizzo.

| Meccanismo                    | Uso e commenti                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Checksum                      | Utilizzato per rilevare errori sui bit in un pacchetto trasmesso.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Timer                         | Serve a far scadere un pacchetto e ritrasmetterlo, forse perché il pacchetto (o il suo ACK) si è smarrito all'interno del canale. I timeout si possono verificare per via dei ritardi anziché degli smarrimenti (timeout prematuro), o quando il pacchetto è stato ricevuto dal destinatario, ma è andato perduto il relativo ACK dal destinatario al mittente. Per questi motivi il destinatario può ricevere copie duplicate di un pacchetto.                                                                                                   |
| Numero di sequenza            | Usato per numerare sequenzialmente i pacchetti di dati che fluiscono tra mittente e destinatario. Le discontinuità nei numeri di sequenza di pacchetti ricevuti consentono al destinatario di rilevare i pacchetti persi. I pacchetti con numero di sequenza ripetuto consentono al destinatario di rilevare pacchetti duplicati.                                                                                                                                                                                                                 |
| Acknowledgment (ACK)          | Utilizzato dal destinatario per comunicare al mittente che un pacchetto o un insieme di pacchetti sono stati ricevuti correttamente. Gli acknowledgement trasporteranno generalmente i numeri di sequenza del pacchetto o dei pacchetti da confermare. A seconda del protocollo, i riscontri possono essere individuali o cumulativi.                                                                                                                                                                                                             |
| Acknowledgment negativo (NAK) | Usato dal destinatario per comunicare al mittente che un pacchetto non è stato ricevuto correttamente. Gli acknowledgement negativi trasporteranno generalmente il numero di sequenza del pacchetto che non è stato ricevuto correttamente.                                                                                                                                                                                                                                                                                                       |
| Finestra e pipelining         | Il mittente può essere forzato a inviare soltanto pacchetti con numeri di sequenza che ricadono in un determinato intervallo. Consentendo a più pacchetti di essere trasmessi e non aver ancora ricevuto acknowledgement si può migliorare l'utilizzo del canale rispetto alla modalità operativa stop-and-wait. Vedremo tra breve che l'ampiezza della finestra può essere impostata sulla base della capacità del destinatario di ricevere e memorizzare messaggi in un buffer, su quella del livello di congestione della rete, o su entrambe. |

gere questo paragrafo per comprendere come tali meccanismi siano stati aggiunti incrementalmente per gestire modelli sempre più complessi (e realistici) di canale tra mittente e destinatario o per migliorare le prestazioni dei protocolli.

Concludiamo la nostra trattazione considerando un’ultima assunzione nel nostro modello di canale sottostante. Ricordiamo di aver ipotizzato che i pacchetti non possono essere riordinati all’interno del canale tra il mittente e il destinatario. Si tratta di un’assunzione ragionevole quando mittente e destinatario sono collegati da un singolo cavo fisico. Ma, quando il canale che li collega è una rete, può capitare che i pacchetti vengano mischiati. Una manifestazione di questo fenomeno è la possibile comparsa di vecchie copie di un pacchetto con numero di sequenza o di acknowledgment  $x$ , anche nel caso in cui non sia contenuto né nella finestra del mittente né in quella del destinatario. Con il riordinamento dei pacchetti, si può pensare al canale come a un buffer che memorizza i pacchetti e li ritrasmette spontaneamente in un momento successivo. Dato che si possono riutilizzare i numeri di sequenza, si deve prestare attenzione alla duplicazione dei pacchetti. L’approccio intrapreso nella pratica consiste nell’assicurarsi che un numero di sequenza non venga riutilizzato, finché il mittente non sia “sicuro” che ogni pacchetto mandato precedentemente con numero di sequenza  $x$  non si trovi più nella rete. Questo risultato viene raggiunto assumendo che un pacchetto non possa “sopravvivere” nella rete per un periodo superiore a un lasso di tempo fissato. Nelle estensioni TCP per le reti ad alta velocità [RFC 7323] si ipotizza un tempo di vita massimo per il pacchetto di circa tre minuti. [Sunshine 1978] descrive un metodo di utilizzo dei numeri di sequenza che permette di evitare nel modo più assoluto i problemi di riordinamento.

## 3.5 Trasporto orientato alla connessione: TCP

Ora che abbiamo trattato i principi alla base del trasferimento dati affidabile, concentriamoci su TCP, il protocollo di Internet a livello di trasporto affidabile e orientato alla connessione. In questo paragrafo vedremo che, per offrire trasferimento dati affidabile, TCP si basa su molti dei principi precedentemente trattati, compresi la rilevazione degli errori, le ritrasmissioni, gli acknowledgment cumulativi, i contatori e i campi nell’intestazione per numeri di sequenza e acknowledgment. TCP viene definito nelle RFC 793, 1122, 2018, 5681 e 7323.

### 3.5.1 Connessione TCP

TCP viene detto **orientato alla connessione** (*connection-oriented*) in quanto, prima di effettuare lo scambio dei dati, i processi devono effettuare l’handshake, ossia devono inviarsi reciprocamente alcuni segmenti preliminari per stabilire i parametri del successivo trasferimento dati. Come parte dell’instaurazione della connessione TCP, entrambe le parti inizializzano molte variabili di stato (Paragrafo 3.7) associate alla connessione.

La “connessione” TCP non è un circuito end-to-end TDM o FDM, come in una rete a commutazione di circuito, in quanto lo stato della connessione risiede completamente nei due sistemi periferici. Dato che il protocollo TCP va in esecuzione solo sui sistemi periferici e non negli elementi di rete intermedi (router e switch a livello di collegamento), questi ultimi non salvano lo stato della con-

nessione TCP. Infatti, i router intermedi sono completamente ignari delle connessioni TCP; essi vedono datagrammi, non connessioni.

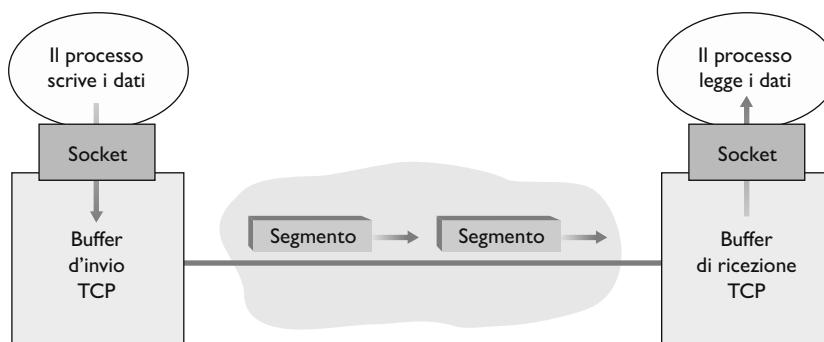
Una connessione TCP offre un **servizio full-duplex**: su una connessione TCP tra il processo A su un host e il processo B su un altro host, i dati a livello di applicazione possono fluire dal processo A al processo B nello stesso momento in cui fluiscono in direzione opposta. Una connessione TCP è anche **punto a punto** (*point-to-point*), ossia ha luogo tra un singolo mittente e un singolo destinatario. Il cosiddetto “multicast”, ossia il trasferimento di dati da un mittente a molti destinatari in un'unica operazione, con TCP non è possibile.

Diamo ora uno sguardo a come si instaurano le connessioni TCP. Supponiamo che il processo di un host voglia inizializzare (e quindi è il processo *client*) una connessione con il processo in un altro host (il processo *server*). Il processo applicativo client innanzitutto informa il livello di trasporto client di voler stabilire una connessione verso un processo nel server. Ricordiamo, dal Paragrafo 2.7.2, che un programma client in Python effettua l'operazione invocando il comando

```
clientSocket.connect((serverName, serverPort))
```

dove *serverName* è il nome del server, mentre *serverPort* identifica il processo sul server. Il TCP in esecuzione sul client procede quindi a stabilire una connessione con il TCP del server. Alla fine di questo paragrafo analizzeremo più dettagliatamente come si instaura la connessione; per ora è sufficiente sapere che il client invia per primo uno speciale segmento TCP; il server risponde con un secondo segmento speciale TCP; infine il client risponde con un terzo segmento speciale. I primi due non trasportano payload, ossia non hanno dati a livello applicativo; il terzo può invece trasportare informazioni utili. Dato che i due host si scambiano tre segmenti, questa procedura che instaura una connessione viene spesso detta **handshake a tre vie** (*three-way handshake*).

Una volta instaurata una connessione TCP, i due processi applicativi si possono scambiare dati. Consideriamo l'invio di dati dal processo client al processo server. Il primo manda un flusso di dati attraverso la socket (Paragrafo 2.7). Questi, quando hanno attraversato il punto di uscita, sono nelle mani di TCP in esecuzione nel client. Come illustrato nella Figura 3.28, TCP dirige i dati al **buffer di invio** della connessione, uno dei buffer riservato durante l'handshake



**Figura 3.28** Buffer di invio e ricezione TCP.

a tre vie, da cui, di tanto in tanto, preleverà blocchi di dati e li passerà al livello di rete.

Un aspetto interessante è che le specifiche TCP [RFC 793] non si preoccupano di indicare quando TCP debba effettivamente inviare i dati nel buffer, affermando che TCP dovrebbe “spedire tali dati in segmenti quando è più conveniente”. La massima quantità di dati prelevabili e posizionabili in un segmento viene limitata dalla **dimensione massima di segmento** (MSS, *Maximum Segment Size*). Questo valore viene generalmente impostato determinando prima la lunghezza del frame più grande che può essere inviato a livello di collegamento dall’host mittente locale, la cosiddetta **unità trasmissiva massima** (MTU, *Maximum Transmission Unit*) e poi scegliendo un MSS tale che il segmento TCP (una volta incapsulato in un datagramma IP) stia all’interno di un singolo frame a livello di collegamento, considerando anche la lunghezza dell’intestazione TCP/IP normalmente pari a 40 byte. I protocolli Ethernet e PPP hanno un MTU di 1500 byte, quindi un valore tipico di MSS è 1460 byte. Sono stati inoltre proposti approcci per scoprire la MTU lungo un percorso, ossia il frame più grande a livello di collegamento che può essere spedito su tutti i collegamenti tra la sorgente e la destinazione [RFC 1191] e per impostare MSS sulla base del valore così determinato. Si noti che l’MSS rappresenta la massima quantità di dati a livello di applicazione nel segmento e non la massima dimensione del segmento TCP con intestazioni incluse. Questa terminologia può confondere, ma dobbiamo convivere con essa, essendo piuttosto diffusa.

TCP accoppia ogni blocco di dati del client a una intestazione TCP, andando pertanto a formare **segmenti TCP**. Questi vengono passati al sottostante livello di rete, dove sono incapsulati separatamente nei datagrammi IP a livello di rete che vengono poi immessi nella rete. Quando all’altro capo TCP riceve un segmento, i dati del segmento vengono memorizzati nel buffer di ricezione della connessione TCP (Figura 3.28). L’applicazione legge il flusso di dati da questo buffer. Ogni lato della connessione presenta un proprio buffer di invio e di ricezione. A questo proposito, potete guardare l’animazione del controllo di flusso in rete, presentata sulla piattaforma MyLab di questo libro, che fornisce un’animazione dei buffer di invio e di ricezione.

Abbiamo visto che una connessione TCP è costituita da buffer, variabili e una connessione socket al processo in un host e da un altro insieme di buffer, variabili e connessione socket al processo in un altro host. Come precedentemente accennato, negli elementi di rete tra gli host (router, switch e ripetitori) non vengono allocati alla connessione buffer o variabili.

### 3.5.2 Struttura dei segmenti TCP

Dopo aver visto rapidamente le connessioni TCP esaminiamo la struttura dei suoi segmenti. Il segmento TCP consiste di campi Intestazione e di un campo contenente un blocco di dati proveniente dall’applicazione. Come accennato precedentemente, la MSS limita la dimensione massima del campo Dati di un segmento. TCP, quando invia un file di grandi dimensioni come, per esempio, un’immagine che fa parte di una pagina web, di solito frammenta il file in porzioni di dimensione MSS (a eccezione dell’ultima, che avrà generalmente di-

**BOX 3.2****UN CASO DI STUDIO****Vinton Cerf, Robert Kahn e TCP/IP**

Nei primi anni '70 cominciarono a proliferare le reti a commutazione di pacchetto; ARPAnet – precursore di Internet – era solo una delle tante. Ciascuna di queste reti aveva un proprio protocollo. Due ricercatori, Vinton Cerf e Robert Kahn, riconobbero l'importanza di interconnettere le varie reti e idearono un protocollo trasversale chiamato TCP/IP (*Transmission Control Protocol/Internet Protocol*). Sebbene i due ricercatori avessero concepito il protocollo come un'entità singola, questo venne successivamente suddiviso nelle sue due parti, TCP e IP, che operarono in modo separato. Cerf e Kahn pubblicarono un articolo su TCP/IP nel 1974 sulla rivista *IEEE Transactions on Communications Technology* [Cerf 1974].

Il protocollo TCP/IP venne progettato prima di PC e workstation, smartphone e tablet, e prima della diffusione delle reti Ethernet, DSL e WiFi e delle altre tecnologie per le reti di accesso, prima del Web, dei social media e dello streaming video. Cerf e Kahn previdero la necessità di un protocollo di rete che da un lato offrisse ampio supporto ad applicazioni non ancora definite e, dall'altro, consentisse l'interoperabilità tra host e protocolli a livello di collegamento.

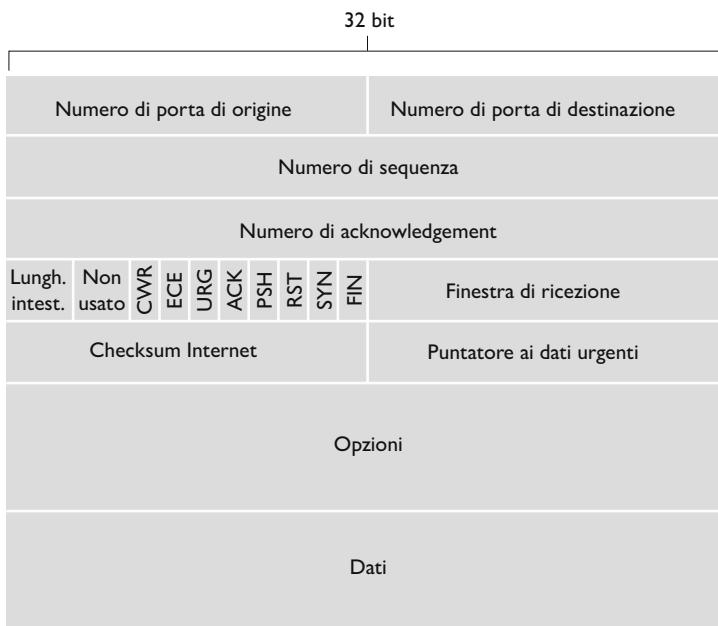
Nel 2004 Cerf e Kahn ricevettero l'*ACM Turing Award*, considerato il premio Nobel per l'informatica, per il "loro lavoro pionieristico sull'internetworking, che comprende la progettazione e l'implementazione dei protocolli di comunicazione di base di Internet, TCP/IP, e per il geniale ruolo ricoperto nel campo delle reti".

mensioni inferiori). Le applicazioni interattive, invece, trasmettono spesso blocchi di dati più piccoli di MSS; per esempio, nelle applicazioni di login remoto quali Telnet e SSH (*secure shell*), il campo dati del segmento TCP è in genere di un solo byte. Dato che l'intestazione TCP occupa comunemente 20 byte (12 in più dell'intestazione UDP), i segmenti inviati da Telnet e SSH possono avere dimensioni di soli 21 byte.

La Figura 3.29 mostra la **struttura dei segmenti TCP**. Come in UDP, l'intestazione include **numeri di porta di origine e di destinazione**, utilizzati per il multiplexing/demultiplexing dei dati da e verso le applicazioni del livello superiore, e un campo **Checksum**. L'intestazione dei segmenti TCP comprende poi i seguenti campi (*field*).

- Il campo **Numero di sequenza** (*sequence number field*) e il campo **Numero di acknowledgment** (*acknowledgment number field*), entrambi di 32 bit, vengono utilizzati dal mittente e dal destinatario TCP per implementare il trasferimento dati affidabile, che descriveremo tra breve.
- Il campo **Finestra di ricezione** (*receive window field*), di 16 bit, viene utilizzato per il controllo di flusso. Vedremo tra poco il suo uso per indicare il numero di byte che il destinatario è disposto ad accettare.
- Il campo **Lunghezza dell'intestazione** (*header length field*), di 4 bit, specifica la lunghezza dell'intestazione TCP in multipli di 32 bit. L'intestazione TCP ha lunghezza variabile a causa del campo delle opzioni TCP. Generalmente, il campo delle opzioni è vuoto e, pertanto, la lunghezza consueta è di 20 byte.

**Figura 3.29** Struttura dei segmenti TCP.



- Il campo **Opzioni (options)**, facoltativo e di lunghezza variabile, viene utilizzato quando mittente e destinatario negoziano la dimensione massima del segmento (MSS) o come fattore di scala per la finestra nelle reti ad alta velocità. Viene, inoltre, definita l'opzione di time-stamping; per dettagli aggiuntivi, potete consultare gli RFC 854 e 1323.
- Il campo **Flag** è di 6 bit. Il bit **ACK** viene usato per indicare che il valore trasmesso nel campo di acknowledgement è valido; ossia, il segmento contiene un acknowledgement per un segmento che è stato ricevuto con successo. I bit **RST**, **SYN** e **FIN** vengono utilizzati per impostare e chiudere la connessione, come vedremo alla fine di questo paragrafo. I bit **CWR** ed **ECE** sono usati nel controllo di congestione esplicito che tratteremo nel Paragrafo 3.7.2. Se il bit **PSH** ha valore 1 il destinatario dovrebbe inviare immediatamente i dati al livello superiore. Infine, si utilizza il bit **URG** per indicare nel segmento la presenza di dati che l'entità mittente a livello superiore ha marcato come “urgenti”. La posizione dell'ultimo byte di dati urgenti viene denotata dal campo **Puntatore ai dati urgenti (urgent data pointer field)**, di 16 bit. Quando ci sono dati urgenti, TCP deve informare l'entità destinataria al livello superiore e passarle un puntatore alla fine dei dati urgenti. Nella pratica, PSH, URG e il puntatore non vengono usati, ma li citiamo per completezza.

La nostra esperienza come insegnanti è che gli studenti a volte trovano la trattazione del formato dei pacchetti piuttosto arida e noiosa. Per una trattazione divertente e fantasiosa, soprattutto se amate Lego™ come noi, consultate [Pomeranz 2010].

## Numeri di sequenza e numeri di acknowledgment

Due tra i campi più importanti dell'intestazione del segmento TCP contengono il numero di sequenza e il numero di acknowledgment, che rappresentano una parte critica del servizio di trasferimento dati affidabile proprio di TCP. Prima di trattarne l'utilizzo, spieghiamo che cosa TCP mette al loro interno.

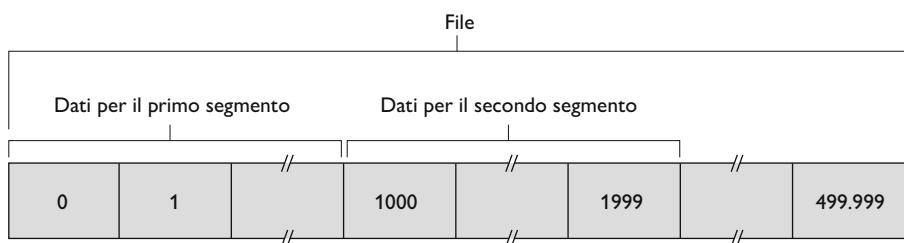
TCP vede i dati come un flusso di byte non strutturati, ma ordinati. L'uso dei numeri di sequenza in TCP riflette questa visione, dato che i numeri di sequenza si applicano al flusso di byte trasmessi e non alla serie di segmenti trasmessi. Il **numero di sequenza per un segmento** (*sequence number for a segment*) è pertanto il numero nel flusso di byte del primo byte del segmento. Per esempio, supponiamo che un processo nell'Host A voglia inviare un flusso di dati a un processo sull'Host B su una connessione TCP. TCP sull'Host A numera implicitamente ogni byte del flusso di dati. Ipotizziamo che il flusso di dati consista in un file da 500.000 byte, che MSS valga 1000 byte e che il primo byte del flusso sia numerato con 0. Come mostra la Figura 3.30, TCP costruisce 500 segmenti per questo flusso di dati. Al primo segmento viene assegnato numero di sequenza 0, al secondo 1000, al terzo 2000 e così via. Ogni numero di sequenza viene inserito nel campo numero di sequenza dell'intestazione del segmento TCP appropriato.

Prendiamo ora in considerazione i numeri di acknowledgment. L'argomento è leggermente più complicato rispetto ai numeri di sequenza. Ricordiamo che TCP è full-duplex, di conseguenza l'Host A può contemporaneamente inviare e ricevere dati dall'Host B (come parte della stessa connessione TCP). I segmenti che provengono dall'Host B presentano un numero di sequenza relativo ai dati che fluiscono da B ad A. *Il numero di acknowledgment che l'Host A scrive nei propri segmenti è il numero di sequenza del byte successivo che l'Host A attende dall'Host B.* Per comprendere il processo è bene presentare alcuni esempi. Supponiamo che l'Host A abbia ricevuto da B tutti i byte numerati da 0 a 535 e che A stia per mandare un segmento all'Host B. L'Host A è in attesa del byte 536 e dei successivi byte nel flusso di dati di B. Pertanto, l'Host A scrive 536 nel campo del numero di acknowledgment del segmento che spedisce a B.

Come ulteriore esempio, supponiamo che l'Host A abbia ricevuto un segmento dall'Host B contenente i byte da 0 a 535 e un altro segmento contenente i byte da 900 a 1000. Per qualche motivo l'Host A non ha ancora ricevuto i byte da 536 a 899. In questo esempio, l'Host A sta ancora attendendo il byte 536 (e i successivi) per ricreare il flusso di dati di B. Perciò il prossimo segmento di A destinato a B conterrà 536 nel campo del numero di acknowledgment. Dato che TCP effettua l'acknowledgment solo dei byte fino al primo byte mancante nel flusso, si dice che tale protocollo offre **acknowledgment cumulativi** (*cumulative acknowledgment*).

L'ultimo esempio che presentiamo indaga un aspetto importante, ma delicato. L'Host A ha ricevuto il terzo segmento (i byte da 900 a 1000) prima di aver ricevuto il secondo (i byte da 536 a 899). Pertanto, il terzo segmento non è arrivato in ordine. La questione è delicata: che cosa fa un host quando riceve segmenti fuori sequenza nella connessione TCP? Sorprendentemente, gli RFC non impongono regole su questo aspetto e lasciano la decisione a chi implementa TCP. Esistono in sostanza due scelte: (1) il destinatario scarta immediatamente i segmenti non ordinati (il che può semplificare la progettazione del

**Figura 3.30** Divisione di un file di dati in segmenti TCP.



destinatario, come abbiamo visto), oppure (2) il destinatario mantiene i byte non ordinati e attende quelli mancanti per colmare i vuoti. Chiaramente, la seconda scelta è più efficiente in termini di banda occupata e rappresenta l'approccio solitamente utilizzato.

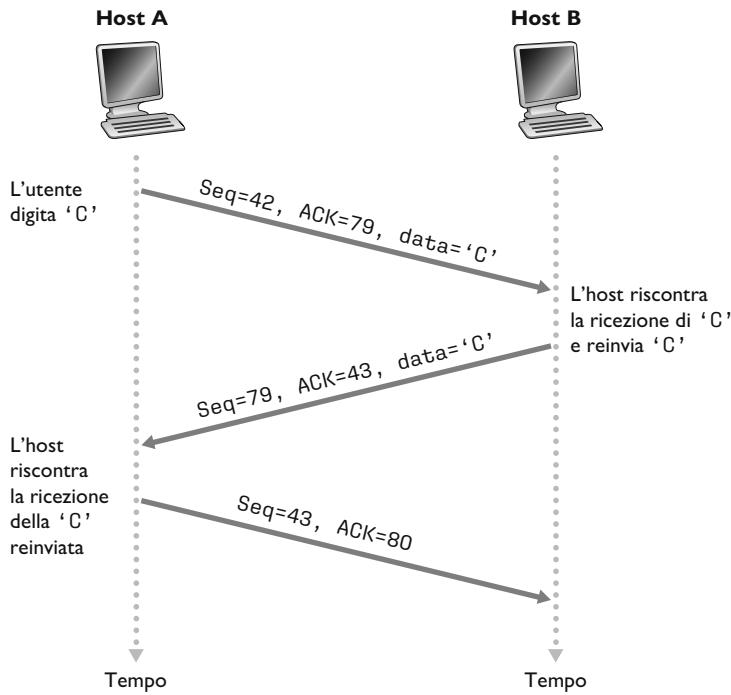
Nella Figura 3.30 abbiamo ipotizzato che il numero di sequenza iniziale fosse 0. In verità, i partecipanti alle connessioni TCP scelgono a caso un numero di sequenza iniziale. Ciò minimizza la possibilità che un segmento ancora presente nella rete, per via di una connessione tra due host precedente e già terminata, venga interpretato erroneamente come segmento valido in una connessione successiva tra gli stessi due host (che devono anche utilizzare gli stessi numeri di porta della vecchia connessione) [Sunshine 1978].

### Telnet: un caso di studio per i numeri di sequenza e di acknowledgment

Telnet, definito nell'RFC 854, è un protocollo a livello di applicazione impiegato per il login remoto che utilizza TCP ed è progettato per funzionare tra qualsiasi coppia di host. A differenza delle applicazioni di trasferimento massiccio di dati descritte nel Capitolo 2, Telnet è un'applicazione interattiva. Presentiamo ora un esempio su Telnet, dato che illustra bene i numeri di sequenza e di acknowledgment di TCP. Abbiamo notato che oggi giorno molti utenti preferiscono usare il protocollo SSH (*secure shell*) anziché Telnet, in quanto con quest'ultimo i dati inviati (tra cui le password) non vengono criptati, rendendo Telnet vulnerabile agli attacchi di intercettazione (*eavesdropping attacks*) trattati nel Paragrafo 8.7 (il Capitolo 8 è disponibile in inglese sulla piattaforma MyLab).

Supponiamo che l'Host A inizi una sessione Telnet con l'Host B. Il primo viene etichettato come client, il secondo come server. Ogni carattere immesso dall'utente (nel client) verrà spedito all'host remoto; quest'ultimo restituirà una copia di ciascun carattere, che sarà mostrata sullo schermo dell'utente. Questa "eco" viene utilizzata per assicurarsi che i caratteri visibili all'utente siano già stati ricevuti ed elaborati nel sito remoto. Quindi, nel lasso di tempo che intercorre tra la digitazione sul tasto e la visualizzazione sul monitor, ciascun carattere attraversa la rete due volte.

Supponiamo ora che l'utente digiti la lettera 'C' ... e poi vada a bere un caffè. Esaminiamo i segmenti TCP scambiati tra client e server. Come mostrato nella Figura 3.31 ipotizziamo che i numeri di sequenza iniziali siano rispettivamente 42 e 79 per client e server. Ricordiamo che il numero di sequenza di un segmento è il numero di sequenza del primo byte nel campo Dati. Pertanto, il primo



**Figura 3.31** Numeri di sequenza e di acknowledgement per una semplice applicazione Telnet su TCP.

segmento inviato dal client avrà numero di sequenza 42 e quello inviato dal server 79. Ricordiamo che il numero di acknowledgement è il numero di sequenza del successivo byte di dati che l'host sta aspettando. Dopo l'instaurazione della connessione TCP, ma prima dell'invio dei dati, il client è in attesa del byte 79 e il server del byte 42.

Come mostrato nella Figura 3.31 vengono spediti tre segmenti. Il primo è inviato dal client al server e contiene nel campo dati il byte ASCII per la lettera 'C'. Il campo numero di sequenza del primo segmento contiene 42. Inoltre, visto che il client non ha ancora ricevuto dati dal server, questo primo segmento avrà 79 nel proprio campo Numero di acknowledgment.

Il secondo segmento spedito dal server al client ha un duplice scopo. Innanzitutto, fa un acknowledgement dei dati ricevuti dal server. Ponendo 43 nel campo Acknowledgment, il server indica al client di aver ricevuto con successo i primi 42 byte e di attendere i byte da 43 in poi. L'altra finalità di questo segmento è di mandare indietro una "eco" della lettera 'C'. Di conseguenza, il secondo segmento presenta nel proprio campo dati la lettera 'C', in ASCII, e ha numero di sequenza 79, ossia il numero iniziale del flusso di dati da server al client in questa connessione TCP, dato che si tratta proprio del primo byte di dati spedito dal server. Notiamo che il l'acknowledgment dei dati dal client al server viene trasportato in un segmento che a sua volta trasporta dati dal server al client; in gergo, si dice che tale acknowledgement è **piggybacked** (o anche **in piggyback**) sul segmento dati dal server al client.

Il terzo segmento viene inviato dal client al server e ha come unico scopo dare un acknowledgement ai dati inviati dal server. Ricordiamo che il secondo segmento conteneva dati, la lettera 'C' dal server al client. Al contrario, il campo

dati di questo segmento è vuoto (in altre parole, l'acknowledgment non è in piggyback ad alcun dato da client al server). Il segmento ha 80 nel suo campo Numero di acknowledgment, in quanto il client ha ricevuto il flusso di byte fino al numero di sequenza 79 e ora è in attesa dei byte dall'80 in avanti. Si potrebbe considerare strano che questo segmento abbia un numero di sequenza pur senza contenere dati. In ogni caso, dato che TCP prevede un campo Numero di sequenza, il segmento deve averne uno.

### 3.5.3 Timeout e stima del tempo di andata e ritorno (RTT)

TCP, come pure il protocollo rdt descritto nel Paragrafo 3.4, utilizza un meccanismo di timeout (*tempo scaduto*) e ritrasmissione per recuperare i segmenti persi. Sebbene tutto questo sia concettualmente semplice, l'implementazione di tale meccanismo in un protocollo come TCP crea alcuni problemi. Forse la questione più ovvia è la durata degli intervalli di timeout. Chiaramente, il timeout dovrebbe essere più grande del tempo di andata e ritorno sulla connessione (**RTT, Round-Trip Time**), ossia del tempo trascorso da quando si invia un segmento a quando se ne riceve l'acknowledgment, altrimenti ci sarebbero delle ritrasmissioni inutili. Ma di quanto deve essere maggiore? E, innanzitutto, come dovrebbe essere stimato *RTT*? Si dovrebbe associare un timer a ogni segmento non riscontrato? La discussione del presente paragrafo si basa sul lavoro contenuto in [Jacobson 1988] e sulle attuali raccomandazioni IETF per gestire i timer TCP [RFC 6298].

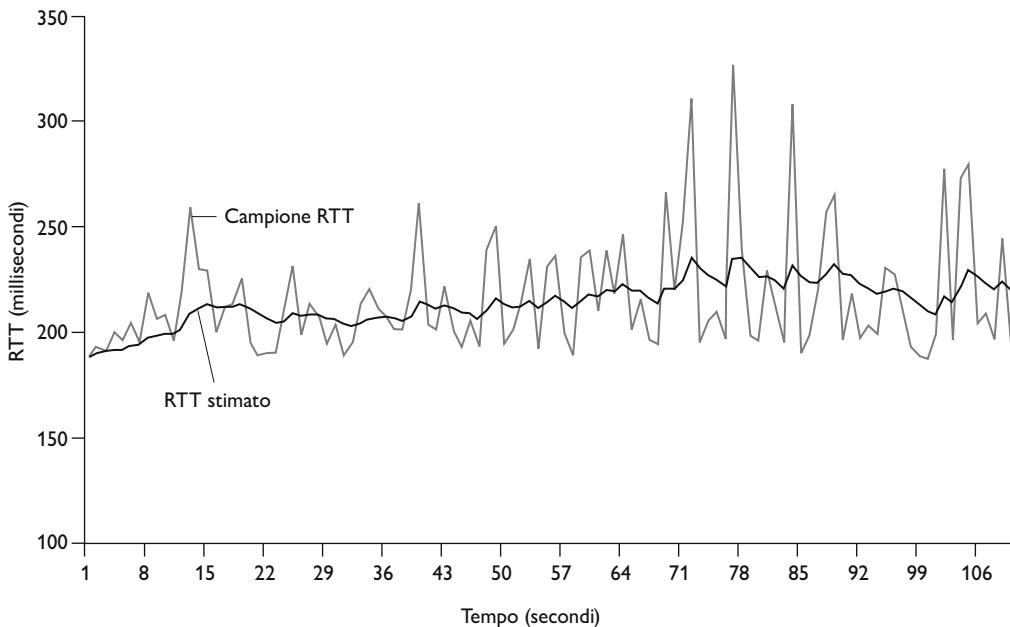
#### Stima del tempo di andata e ritorno (RTT)

Cominciamo lo studio della gestione dei timer TCP considerando come il protocollo stimi il tempo di andata e ritorno tra mittente e destinatario. L'*RTT* misurato di un segmento, denotato come *SampleRTT*, è la quantità di tempo che intercorre tra l'istante di invio del segmento (ossia quando viene passato a IP) e quello di ricezione dell'acknowledgment del segmento. Anziché misurare un *SampleRTT* per ogni segmento, la maggior parte delle implementazioni TCP effettua una sola misurazione di *SampleRTT* alla volta. Ossia, in ogni istante di tempo, *SampleRTT* viene valutato per uno solo dei segmenti trasmessi e per cui non si è ancora ricevuto acknowledgment, il che comporta approssimativamente la misurazione di un nuovo valore di *SampleRTT* a ogni *RTT*. Inoltre, TCP non calcola mai il *SampleRTT* per i segmenti ritrasmessi, cioè lo calcola soltanto per i segmenti trasmessi una volta sola [Karn 1987] (in uno dei Problemi a fine capitolo vi si chiederà il perché).

Ovviamente, i campioni variano da segmento a segmento in base alla congestione nei router e al diverso carico sui sistemi periferici. A causa di tale fluttuazione, ogni valore di *SampleRTT* può essere atipico. Per effettuare una stima risulta naturale calcolare una media dei valori di *SampleRTT*, che TCP chiama *EstimatedRTT*. Quando si ottiene un nuovo *SampleRTT*, TCP aggiorna *EstimatedRTT* secondo la formula:

$$\text{EstimatedRTT} = (1 - \alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{SampleRTT}$$

La formula è scritta come in un linguaggio di programmazione: il nuovo valore di *EstimatedRTT* è una combinazione ponderata del suo precedente valore e



**Figura 3.32** Campioni e stima di RTT.

del nuovo valore di SampleRTT. Il valore raccomandato per  $\alpha$  è 0,125 (ossia 1/8) [RFC 6298], nel qual caso la formula precedente diventa:

$$\text{EstimatedRTT} = 0,875 \cdot \text{EstimatedRTT} + 0,125 \cdot \text{SampleRTT}$$

Si noti che EstimatedRTT è una media ponderata dei valori SampleRTT. Come discusso in uno dei Problemi alla fine di questo capitolo, tale media attribuisce maggiore importanza ai campioni recenti rispetto a quelli vecchi. Ciò è naturale, dato che i primi riflettono meglio la congestione attuale della rete. In statistica, una media costruita in tal modo è detta **media mobile esponenziale ponderata** (EWMA, *Exponential Weighted Moving Average*). La parola “esponenziale” compare nell’acronimo EWMA in quanto il peso dei campioni decresce esponenzialmente al procedere degli aggiornamenti. Nei Problemi a fine capitolo vi sarà chiesto di derivare il termine esponenziale in EstimatedRTT.

La Figura 3.32 mostra i valori SampleRTT ed EstimatedRTT con  $\alpha = 1/8$  per una connessione TCP tra gaia.cs.umass.edu (ad Amherst, Massachusetts) e fantasia.eurecom.fr (nel sud della Francia). Come si può vedere, le variazioni di SampleRTT vengono “smussate” dal calcolo di EstimatedRTT.

Oltre ad avere una stima di RTT è anche importante possedere la misura della sua variabilità. [RFC 6298] definisce la variazione RTT, DevRTT, come una stima di quanto SampleRTT generalmente si discosta da EstimatedRTT:

$$\text{DevRTT} = (1 - \beta) \times \text{DevRTT} + \beta \cdot | \text{SampleRTT} - \text{EstimatedRTT} |$$

Si noti che DevRTT è un EWMA della differenza tra SampleRTT e EstimatedRTT. Se i valori di SampleRTT presentano fluttuazioni limitate, allora DevRTT sarà piccolo; in caso di notevoli fluttuazioni, DevRTT sarà grande. Il valore suggerito per  $\beta$  è 0,25.

### Impostazione e gestione del timeout di ritrasmissione

Dati i valori di EstimatedRTT e DevRTT, quali valori andrebbero usati per l'intervallo di timeout di TCP? Chiaramente, l'intervallo non può essere inferiore a quello di EstimatedRTT, altrimenti verrebbero inviate ritrasmissioni non necessarie. Ma l'intervallo stesso non dovrebbe essere molto maggiore di EstimatedRTT, altrimenti TCP non ritrasmetterebbe rapidamente il segmento perduto, il che comporterebbe gravi ritardi sul trasferimento dei dati. È pertanto auspicabile impostare il timeout a EstimatedRTT più un certo margine che dovrebbe essere grande quando c'è molta fluttuazione nei valori di SampleRTT e piccolo in caso contrario. Qui entra in gioco il valore di DevRTT. Tutti questi aspetti vengono presi in considerazione dal modo in cui TCP determina l'intervallo di timeout di ritrasmissione:

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \cdot \text{DevRTT}$$

In [RFC 6298] viene raccomandato un valore iniziale di TimeoutInterval pari a 1 secondo. Inoltre, quando si verifica un timeout, TimeoutInterval viene raddoppiato per evitare un timeout prematuro riferito a un segmento successivo per cui si riceverà presto un acknowledgment. Tuttavia, appena viene ricevuto un segmento ed EstimatedRTT viene aggiornato, TimeoutInterval viene ric算calcolato secondo la formula precedente.

#### 3.5.4 Trasferimento dati affidabile

Abbiamo già visto che il servizio di Internet a livello di rete (servizio IP) non è affidabile: non garantisce la consegna in sequenza, né la consegna stessa dei da-

**BOX 3.3**

 **TEORIA E PRATICA**

#### TCP

TCP offre un trasferimento dati affidabile usando acknowledgment e timer secondo modalità simili a quelle studiate nel Paragrafo 3.4. TCP riscontra i dati ricevuti correttamente e ritrasmette i segmenti quando ritiene che questi o i corrispondenti acknowledgment siano andati perduti o siano stati alterati. Alcune versioni di TCP presentano anche un meccanismo隐式的 di NAK: con un meccanismo di fast retransmit, la ricezione di tre ACK duplicati per un dato segmento equivale a un NAK隐式的 del segmento seguente, generando la sua ritrasmissione prima del timeout. TCP utilizza numeri di sequenza per consentire al destinatario l'identificazione di segmenti persi o duplicati. Proprio come nel nostro protocollo di trasferimento dati affidabile, rdt3.0, TCP non può affermare con certezza se un segmento, o il suo ACK, siano stati persi, alterati o eccessivamente ritardati. Lato mittente, la reazione di TCP sarà sempre la stessa: ritrasmettere il segmento.

TCP usa anche il pipelining, consentendo al mittente di avere più segmenti trasmessi, ma non ancora riscontrati in ogni dato istante. Abbiamo visto precedentemente che il pipelining può migliorare enormemente il throughput di una sessione quando il rapporto tra la dimensione del segmento e il ritardo end-to-end è piccolo. Il numero specifico di segmenti non ancora riscontrati dipende dai meccanismi di TCP per il controllo di flusso (alla fine di questo paragrafo) e di congestione (Paragrafo 3.7). Al momento, ci basta sapere che il mittente TCP fa uso di pipelining.

tagrammi, né l'integrità dei dati. Con il servizio IP, i datagrammi possono sovraffollare i buffer dei router e non raggiungere la destinazione o arrivare in ordine casuale e con bit alterati. Dato che sono inglobati nei datagrammi IP, anche i segmenti a livello di trasporto possono risentire di questi problemi.

TCP crea un **servizio di trasporto dati affidabile** (*reliable data transfer service*) al di sopra del servizio inaffidabile e best-effort di IP, assicurando che il flusso di byte che i processi leggono dal buffer di ricezione TCP non sia alterato, non abbia buchi, non presenti duplicazioni e rispetti la sequenza originaria; in altre parole, il flusso di dati in arrivo è esattamente quello spedito. Come TCP fornisca questo trasferimento dati è una questione che coinvolge molti dei principi studiati nel Paragrafo 3.4.

Nel nostro precedente sviluppo di tecniche per il trasferimento affidabile dei dati era concettualmente più semplice associare un singolo timer a ciascun segmento trasmesso per cui non si sia ancora ricevuto un acknowledgment. Ma se tutto ciò funziona bene in teoria, la gestione dei timer può richiedere considerevoli risorse aggiuntive. Pertanto, le procedure suggerite per la gestione dei timer di TCP [RFC 6298] utilizzano un solo timer di ritrasmissione, anche in presenza di più segmenti trasmessi. Il protocollo TCP presentato in questo paragrafo segue tale raccomandazione.

Vedremo ora come TCP offra il trasferimento affidabile dei dati in due passi successivi. Dapprima presentiamo una descrizione molto semplificata di un mittente TCP che fa uso solo di timeout per recuperare i segmenti persi e poi una più completa che utilizza anche gli acknowledgment duplicati. Nella trattazione che segue, supponiamo che i dati vengano inviati solo in una direzione, dall'Host A all'Host B e che il primo stia trasmettendo un file di grandi dimensioni.

La Figura 3.33 offre una rappresentazione fortemente semplificata di un mittente TCP. Esistono tre eventi principali relativi alla trasmissione e ritrasmissione dei dati: dati provenienti dall'applicazione, timeout e ricezione di un ACK. Quando si verifica il primo evento, TCP incapsula i dati che gli giungono dall'applicazione in un segmento e lo passa a IP. Notiamo che ciascun segmento include un numero di sequenza che rappresenta il numero del primo byte di dati del segmento nel flusso di byte (Paragrafo 3.5.2). Inoltre, se il timer non è già in funzione per qualche altro segmento, TCP lo avvia quando il segmento viene passato a IP. È utile pensare che il timer sia associato al più vecchio segmento che non ha ricevuto acknowledgment. L'intervallo di scadenza per il timer è il `TimeoutInterval`, che viene calcolato in termini di `EstimatedRTT` e `DevRTT` (Paragrafo 3.5.3).

Il secondo evento è il timeout, cui TCP risponde ritrasmettendo il segmento che lo ha causato e quindi riavviando il timer.

Il terzo evento, che deve essere gestito dal mittente TCP, è l'arrivo del segmento di acknowledgment con un valore valido nel campo ACK. Quando si verifica tale evento, TCP confronta il valore  $y$  di ACK con la propria variabile `SendBase`. La variabile di stato TCP `SendBase` è il numero di sequenza del più vecchio byte che non ha ancora ricevuto un acknowledgment. Di conseguenza  $SendBase - 1$  è il numero di sequenza dell'ultimo byte che si sa essere stato ricevuto correttamente e nell'ordine giusto. Come precedentemente indicato, TCP utilizza acknowledgment cumulativi, pertanto  $y$  conferma la ricezione di tutti i byte precedenti al byte numero  $y$ . Se  $y$  è maggiore di `SendBase`, allora

**Figura 3.33** Mittente

```

/* Ipotizziamo che il mittente non subisca imposizioni dal
controllo di flusso o di congestione TCP, che i dati dall'alto
abbiano dimensione inferiore a MSS e che il trasferimento dati
avvenga in un'unica direzione */

NextSeqNum = InitialSeqNumber
SendBase = InitialSeqNumber

loop (per sempre) {
    switch (evento)

        evento: dati ricevuti dall'applicazione a livello
                 superiore crea il segmento TCP con numero di sequenza
                 NextSeqNum
        if (il timer attualmente non è in funzione)
            avvia il timer
            passa il segmento a IP
            NextSeqNum = NextSeqNum + lunghezza(dati)
            break;

        evento: timeout del timer
            ritrasmetti il segmento che non ha ricevuto ACK con il
            più piccolo numero di sequenza
            avvia il timer
            break;

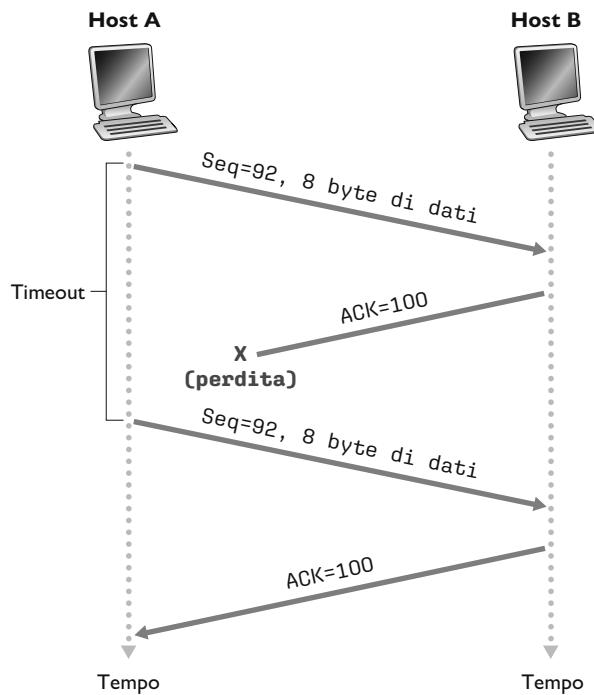
        evento: ACK ricevuto, con valore del campo ACK pari a y
            if (y > SendBase) {
                SendBase = y
                if (esistono attualmente segmenti senza ACK)
                    avvia il timer
            }
            break;
    } /* fine del loop */
}

```

l'ACK si riferisce a uno o più segmenti che precedentemente non avevano ancora ricevuto conferma. Quindi, il mittente aggiorna la propria variabile `SendBase` e poi, se non ci sono segmenti che ancora necessitano di acknowledgment, riavvia il timer.

### Alcuni scenari interessanti

Abbiamo appena visto come TCP fornisca un trasferimento dati affidabile che presenta, nonostante si tratti di una versione fortemente semplificata, dell'ingegnosità. Per comprendere come funziona il protocollo, consideriamo qualche semplice scenario. La Figura 3.34 mostra un caso in cui l'Host A spedisce un

**Figura 3.34**

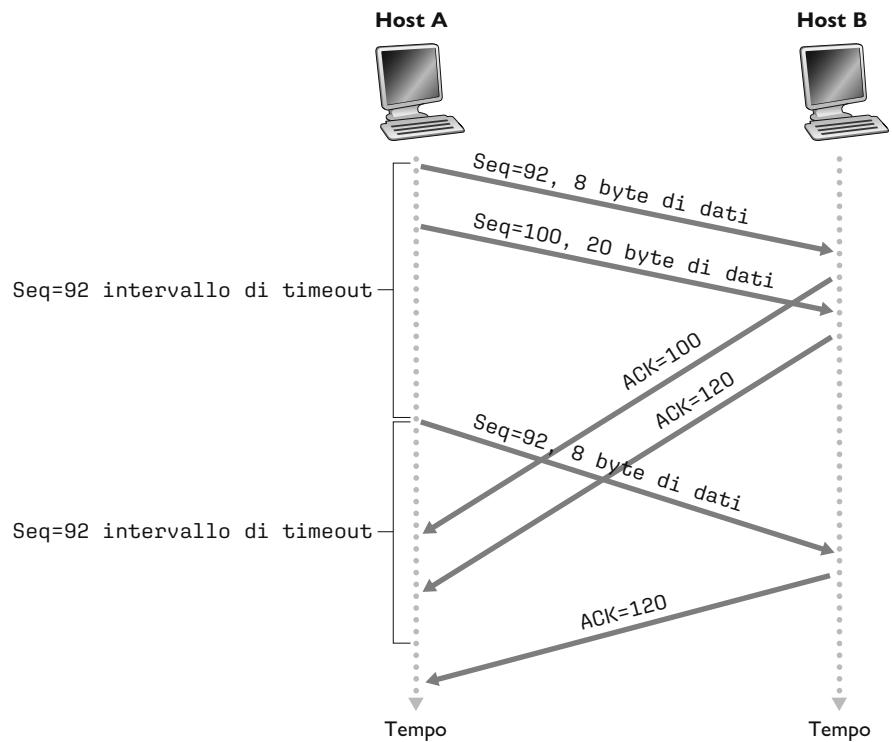
Ritrasmissione dovuta alla perdita di un acknowledgement.

segmento all'Host B. Supponiamo che questo segmento abbia numero di sequenza 92 e contenga 8 byte di dati. Dopo aver inviato il segmento, A attende un segmento da B con numero di acknowledgment 100. Sebbene il segmento in questione sia stato ricevuto da B, l'acknowledgment sul percorso inverso viene smarrito. In questo caso si verifica l'evento di timeout e l'Host A ritrasmette lo stesso segmento. Ovviamente, quando l'Host B riceve la ritrasmissione, rileva dal numero di sequenza che il segmento contiene dati che sono già stati ricevuti. Quindi, l'Host B scarta i byte del segmento ritrasmesso.

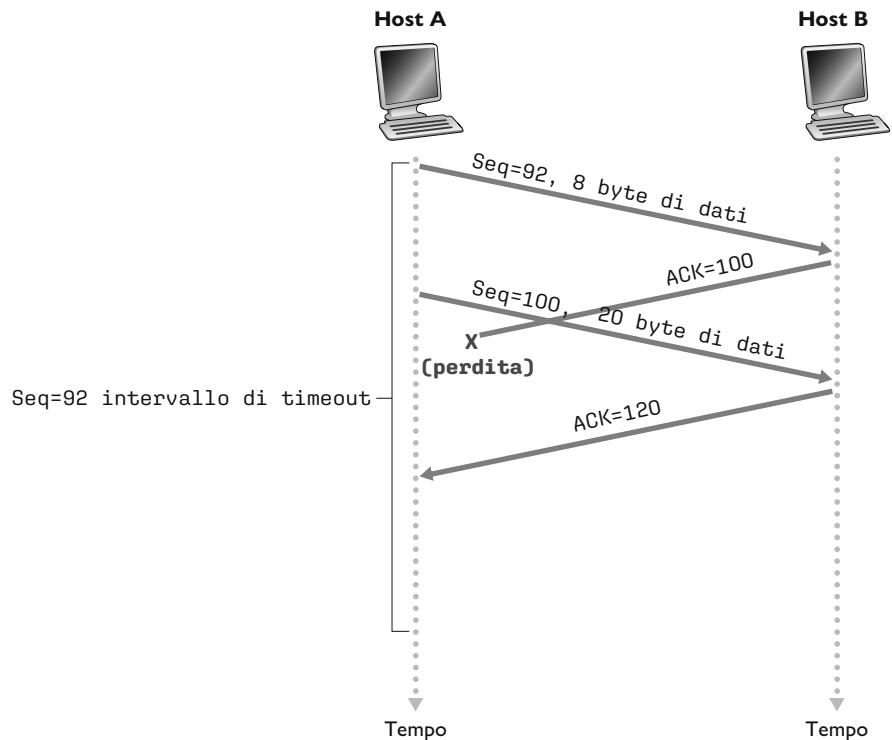
In un secondo scenario (Figura 3.35), A invia due segmenti. Il primo ha numero di sequenza 92 e 8 byte di dati, il secondo ha numero di sequenza 100 e 20 byte di dati. Supponiamo che entrambi arrivino intatti a B e che questo invii due acknowledgment separati per i segmenti, il primo numerato 100, il secondo 120. Supponiamo ora che nessuno degli acknowledgment arrivi all'Host A prima del timeout. Quando si verifica il timeout, l'Host A rispedisce il primo segmento con numero di sequenza 92 e riavvia il timer. Fino a quando l'ACK del secondo segmento non arriva prima del nuovo timeout, il secondo segmento non sarà ritrasmesso.

Supponiamo infine (Figura 3.36) che l'Host A invii due segmenti, esattamente come nel secondo esempio. L'acknowledgment del primo segmento viene perso nella rete ma, appena prima dell'evento di timeout, l'Host A riceve un acknowledgment con numero 120. È, pertanto, a conoscenza che l'Host B ha ricevuto tutto fino al byte 119; quindi non rispedisce nessuno dei due segmenti.

**Figura 3.35** Segmento 100 non ritrasmesso.



**Figura 3.36**  
Acknowledgement cumulativo che evita le ritrasmissioni del primo segmento.



### Raddoppio dell'intervallo di timeout

Vediamo ora alcune varianti utilizzate dalla maggior parte delle implementazioni TCP. La prima riguarda la lunghezza dell'intervallo di timeout dopo la scadenza di un timer. Con questa modifica, in tutti i casi in cui si verifica un timeout, TCP, come descritto precedentemente, ritrasmette il segmento con il più basso numero di sequenza che non abbia ancora ricevuto acknowledgment. Tuttavia, ogni volta che questo si verifica, TCP imposta il successivo intervallo di timeout al doppio del valore precedente, anziché derivarlo dagli ultimi EstimatedRTT e DevRTT (Paragrafo 3.5.3). Supponiamo per esempio che il TimeoutInterval associato al più vecchio segmento che non ha ancora ricevuto acknowledgment sia pari a 0,75 secondi quando il timer scade per la prima volta. TCP ritrasmetterà quindi questo segmento e imposterà il nuovo tempo di scadenza del timer a 1,5 secondi. Se il timer scade ancora, TCP ritrasmetterà il segmento stabilendo il tempo di scadenza a 3 secondi. Di conseguenza, gli intervalli crescono esponenzialmente a ogni ritrasmissione. Tuttavia, tutte le volte che il timer viene avviato dopo uno degli altri due eventi possibili (ossia la ricezione di dati dall'applicazione superiore e la ricezione di un ACK), il TimeoutInterval viene ricavato dai più recenti valori di EstimatedRTT e DevRTT.

Questa modifica offre una forma limitata di controllo di congestione. Forme più complete saranno studiate nel Paragrafo 3.7. La scadenza del timer viene probabilmente causata dalla congestione nella rete, ossia troppi pacchetti arrivano presso una (o più) code dei router nel percorso tra l'origine e la destinazione, provocando l'eliminazione dei pacchetti e/o lunghi ritardi di accodamento. Nei periodi di congestione, se le sorgenti continuano a ritrasmettere pacchetti, la congestione può peggiorare. TCP si comporta in maniera più rispettosa della rete, dato che ciascun mittente ritrasmette dopo intervalli sempre più lunghi. Vedremo l'utilizzo di un principio simile con Ethernet quando studieremo CSMA/CD nel corso del Capitolo 6.

### Ritrasmissione rapida

Uno dei problemi legati alle ritrasmissioni è che il periodo di timeout può rivelarsi relativamente lungo. Quando si smarrisce un segmento, il lungo periodo di timeout impone al mittente di ritardare il nuovo invio del pacchetto perso, incrementando di conseguenza il ritardo end-to-end. Fortunatamente, il mittente può in molti casi rilevare la perdita dei pacchetti ben prima che si verifichi l'evento di timeout grazie agli **ACK duplicati** relativi a un segmento il cui ACK è già stato ricevuto dal mittente. Per comprendere la reazione del mittente a un ACK duplicato dobbiamo innanzitutto capire perché il destinatario ne invia uno.

La Tabella 3.2 riassume la politica di generazione degli ACK di TCP [RFC 5681]. Quando il destinatario TCP riceve un segmento con numero di sequenza superiore al successivo numero di sequenza atteso e in ordine, rileva un buco nel flusso di dati, ossia un segmento mancante. Tale vuoto potrebbe essere il risultato di segmenti persi o riordinati all'interno della rete. Il destinatario non può inviare un acknowledgment negativo esplicito al mittente, dato che TCP non lo prevede, ma si limita a mandare nuovamente un acknowledgment relativo all'ultimo byte di dati che ha ricevuto in ordine (duplicando così un ACK).

**Tabella 3.2** Raccomandazioni sulla generazione degli acknowledgment [RFC 5681].

| Evento                                                                                                                                | Azione del ricevente TCP                                                                                                                                            |
|---------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Arrivo ordinato di segmento con numero di sequenza atteso. Tutti i dati fino al numero di sequenza atteso sono già stati riscontrati. | ACK ritardato. Attende fino a 500 millisecondi per l'arrivo ordinato di un altro segmento. Se in questo intervallo non arriva il successivo segmento, invia un ACK. |
| Arrivo ordinato di segmento con numero di sequenza atteso. Un altro segmento ordinato è in attesa di trasmissione dell'ACK.           | Invia immediatamente un singolo ACK cumulativo, riscontrando entrambi i segmenti ordinati.                                                                          |
| Arrivo non ordinato di segmento con numero di sequenza superiore a quello atteso. Viene rilevato un buco.                             | Invia immediatamente un ACK duplicato, indicando il numero di sequenza del prossimo byte atteso (che è l'estremità inferiore del buco).                             |
| Arrivo di segmento che colma parzialmente o completamente il buco nei dati ricevuti.                                                  | Invia immediatamente un ACK, ammesso che il segmento cominci all'estremità inferiore del buco.                                                                      |

Notiamo che la Tabella 3.2 contempla il caso in cui il destinatario non scarti i segmenti non ordinati.

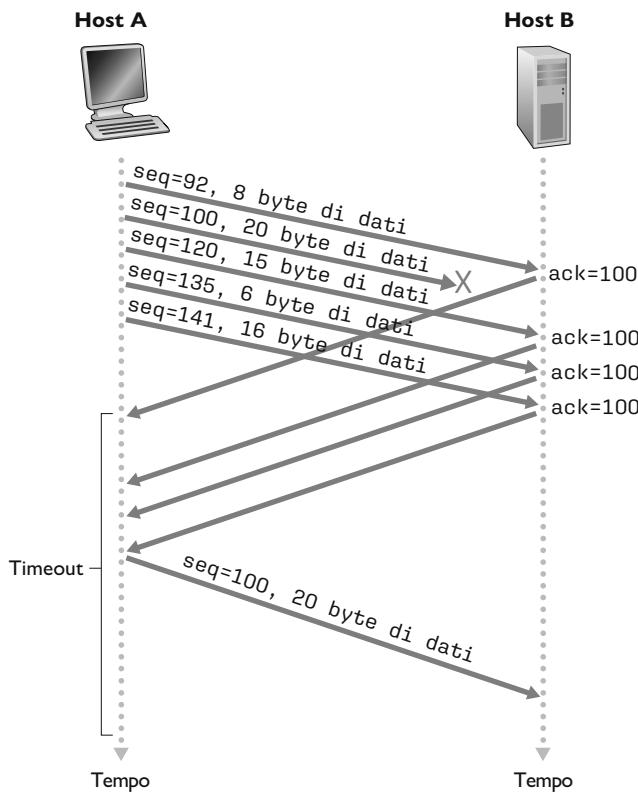
Dato che in molti casi il mittente invia un gran numero di segmenti, se uno di questi viene smarrito ci saranno probabilmente molti ACK duplicati. Se il mittente TCP riceve tre ACK duplicati per lo stesso dato, considera questo evento come indice che il successivo segmento sia andato perduto (nei Problemi alla fine del capitolo ci chiederemo il motivo per cui il mittente attenda tre ACK ripetuti anziché uno solo). Nel caso in cui siano stati ricevuti tre ACK duplicati, il mittente TCP effettua una **ritrasmissione rapida** (*fast retransmit*) [RFC 5681], rispedendo il segmento mancante prima che scada il timer (si veda la Figura 3.37 dove viene perduto il secondo segmento e ritrasmesso prima che il suo timer scada). In TCP con ritrasmissione rapida il seguente frammento di codice sostituisce l'evento di ACK ricevuto nella Figura 3.33.

```

evento: ACK ricevuto, con valore del campo ACK pari a y
    if (y > SendBase) {
        SendBase = y
        if (esistono attualmente segmenti che non hanno
            ricevuto un ACK)
            avvia il timer
    }
else { /* un ACK duplicato per un segmento */
    incrementa il numero di ACK duplicati ricevuti per y
    if (numero di ACK duplicati ricevuti per y == 3) {
        /* ritrasmissione rapida */
        rispedisci il segmento con numero di sequenza y
    }
break;

```

Avevamo già fatto notare come, quando in un vero protocollo si implementa un meccanismo di timeout e ritrasmissione, sorgano molti problemi. Le proce-



**Figura 3.37**  
Ritrasmissione veloce:  
il segmento perduto viene  
ritrasmesso prima che  
il suo timer scada.

dure qui descritte, evolute nel tempo e risultato di più di vent'anni di esperienza con i timer TCP, confermano questa osservazione.

### Go-Back-N (GBN) o ripetizione selettiva (SR)?

Chiudiamo la nostra trattazione sui meccanismi TCP di ripristino dagli errori considerando la seguente domanda: TCP è un protocollo GBN o SR? Ricordiamo che gli acknowledgment TCP sono cumulativi e che i segmenti ricevuti correttamente, ma in modo disordinato, non vengono notificati singolarmente dal destinatario. Quindi (Figure 3.33 e 3.19), il mittente TCP deve solo memorizzare il numero di sequenza più basso tra i byte trasmessi che non hanno ancora ricevuto acknowledgment (SendBase) e il numero di sequenza del successivo byte da inviare (NextSeqNum). In questo senso, TCP assomiglia molto a un protocollo di tipo GBN. Tuttavia esistono alcune differenze chiave tra TCP e Go-Back-N. Molte implementazioni TCP memorizzano in un buffer i segmenti ricevuti correttamente, ma non in ordine [Stevens 1994]. Considerate che cosa succede quando il mittente invia una sequenza di segmenti 1, 2, ..., N che arrivano al destinatario in ordine e senza errori. Ipotizziamo, inoltre, che l'acknowledgment per il pacchetto  $n < N$  vada perduto, ma che i restanti  $N - 1$  giungano al mittente prima dei rispettivi timeout. In questo esempio GBN ritrasmetterebbe non solo il pacchetto  $n$ , ma anche tutti i pacchetti da  $n + 1$  a  $N$ . TCP, invece, ritrasmetterebbe al massimo il segmento  $n$ . Inoltre, TCP non ri-

trasmetterebbe neppure questo segmento se ricevesse l'acknowledgment del segmento  $n + 1$  prima della scadenza del timeout del segmento  $n$ .

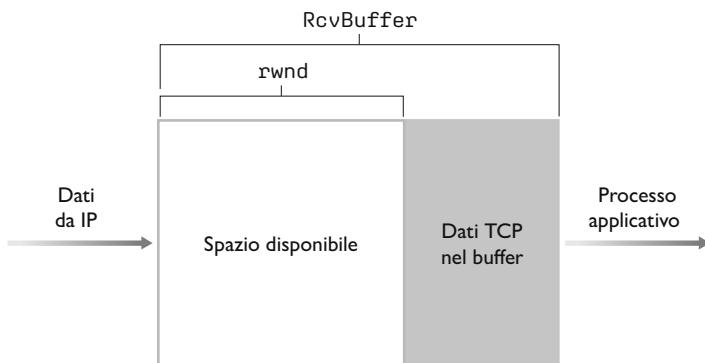
Una modifica proposta di TCP, il cosiddetto **acknowledgment selettivo** (*selective acknowledgment*) [RFC 2018], consente al destinatario di mandare acknowledgment in modo selettivo per i segmenti non in ordine anziché cumulativamente per l'ultimo segmento ricevuto senza errori e nell'ordine giusto. Se combinato con la ritrasmissione selettiva (vale a dire, evitando la ritrasmissione dei segmenti per cui si è già ricevuto un acknowledgment in modo selettivo dal destinatario), TCP è molto simile a un generico protocollo SR. È quindi opportuno classificare il meccanismo di ripristino dagli errori proprio di TCP come un ibrido tra i protocolli di tipo GBN e SR.

### 3.5.5 Controllo di flusso

Ricordiamo che gli host agli estremi delle connessioni TCP riservano dei buffer di ricezione per la connessione. Quando la connessione TCP riceve byte corretti e in sequenza, li posiziona nel buffer di ricezione. Il processo applicativo associato legge i dati da questo buffer, ma non necessariamente nell'istante in cui arrivano. Infatti, l'applicazione ricevente potrebbe essere impegnata in qualche altro compito e potrebbe non leggere i dati fino a un istante di tempo molto successivo al loro arrivo. Se l'applicazione è relativamente lenta nella lettura dei dati può accadere che il mittente mandi in overflow il buffer di ricezione inviando molti dati troppo rapidamente.

TCP offre un **servizio di controllo di flusso** (*flow-control service*) alle proprie applicazioni per evitare che il mittente saturi il buffer del ricevente. Il controllo di flusso è pertanto un servizio di confronto sulla velocità, dato che paragona la frequenza di invio del mittente con quella di lettura dell'applicazione ricevente. Come notato in precedenza, i mittenti TCP possono anche essere rallentati dalla congestione nella rete IP. Questa forma di controllo del mittente viene detta **controllo di congestione** (*congestion control*), un argomento che studieremo in dettaglio nei Paragrafi 3.6 e 3.7. Sebbene le azioni intraprese dal controllo di flusso e di congestione siano simili (il rallentamento del mittente), sono causate da ragioni molto differenti. Così, anche se molti autori tendono a utilizzare le due locuzioni in modo intercambiabile, il lettore di buon senso farebbe bene a distinguergli. Trattiamo ora il controllo di flusso TCP. Al momento supponiamo che l'implementazione obblighi il destinatario TCP a scartare i segmenti non in ordine.

TCP offre il controllo di flusso facendo mantenere al mittente una variabile chiamata **finestra di ricezione** (*receive window*) che, in sostanza, fornisce al mittente un'indicazione dello spazio libero disponibile nel buffer del destinatario. Dato che TCP è full-duplex, i due mittenti mantengono finestre di ricezione distinte. Vediamo ora il concetto di finestra di ricezione nel contesto di un trasferimento di file. Supponiamo che l'Host A stia inviando un file di grandi dimensioni all'Host B su una connessione TCP. Quest'ultimo alloca un buffer di ricezione per la connessione, la cui dimensione è denotata come `recvBuffer`. Di tanto in tanto, il processo applicativo nell'Host B legge dal buffer. Definiamo le seguenti variabili.



**Figura 3.38** Finestra di ricezione (*rwnd*) e buffer di ricezione (*RcvBuffer*).

- *LastByteRead*: numero dell'ultimo byte nel flusso di dati che il processo applicativo in B ha letto dal buffer.
- *LastByteRcvd*: numero dell'ultimo byte, nel flusso di dati, che proviene dalla rete e che è stato copiato nel buffer di ricezione di B.

Dato che TCP non può mandare in overflow il buffer allocato, dovremo per forza avere:

$$\text{LastByteRcvd} - \text{LastByteRead} \leq \text{RcvBuffer}$$

La finestra di ricezione, indicata con *rwnd*, viene impostata alla quantità di spazio disponibile nel buffer:

$$\text{rwnd} = \text{RcvBuffer} - [\text{LastByteRcvd} - \text{LastByteRead}]$$

Dato che lo spazio disponibile varia col tempo, *rwnd* è dinamica, come illustrato nella Figura 3.38.

Come viene usata la variabile *rwnd* da parte della connessione per offrire il servizio di controllo di flusso? L'Host B comunica all'Host A quanto spazio disponibile sia presente nel buffer della connessione, scrivendo il valore corrente di *rwnd* nel campo apposito dei segmenti che manda ad A. L'Host B inizializza *rwnd* con il valore di *RcvBuffer* e, ovviamente, deve tenere traccia di variabili specifiche per ogni connessione.

A sua volta, l'Host A tiene traccia di due variabili, *LastByteSent* e *LastByteAcked*, il cui significato è rispettivamente “ultimo byte mandato” e “ultimo byte per cui si è ricevuto acknowledgment”. Si noti che la differenza tra i valori di queste due variabili esprime la quantità di dati spediti da A per cui non si è ancora ricevuto un acknowledgment. Mantenendo la quantità di dati senza acknowledgment sotto il valore di *rwnd*, si garantisce che l'Host A non mandi in overflow il buffer di ricezione dell'Host B. Quindi, l'Host A si assicura che per tutta la durata della connessione sia rispettata la disuguaglianza

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{rwnd}$$

In questo schema esiste un problema tecnico secondario. Per accorgersene, supponiamo che il buffer di ricezione dell'Host B si riempia, di modo che *rwnd* = 0 e che, dopo averlo notificato all'Host A, non abbia più nulla da inviare ad A e vediamo che cosa succede. Quando il processo applicativo in B svuota il

buffer, TCP non invia nuovi segmenti con nuovi valori di rwnd; infatti, TCP fa pervenire un segmento all’Host A solo se ha dati o un acknowledgment da mandare. Di conseguenza, quest’ultimo non viene informato del fatto che si sia liberato un po’ di spazio nel buffer di ricezione dell’Host B: l’Host A è bloccato e non può trasmettere ulteriori dati! Per risolvere questo problema, le specifiche TCP richiedono che l’Host A continui a inviare segmenti con un byte di dati quando la finestra di ricezione di B è zero. Il destinatario risponderà a questi segmenti con un acknowledgment. Prima o poi il buffer inizierà a svuotarsi e i riscontri conterranno un valore non nullo per rwnd.

Sulla piattaforma MyLab di questo libro è disponibile un’animazione interattiva che mostra le operazioni della finestra di ricezione TCP.

Dopo aver descritto il servizio di controllo di flusso di TCP, menzioniamo brevemente che UDP non offre controllo di flusso. Per comprendere l’argomento, consideriamo l’invio di una serie di segmenti UDP da un processo sull’Host A verso un processo sull’Host B. In una tipica implementazione UDP, tale protocollo metterà in coda i segmenti in un buffer di dimensione finita che “precede” la corrispondente socket (che fa da punto di collegamento con il processo). Se il processo non legge i segmenti dal buffer a velocità sufficiente, si verifica un overflow e alcuni segmenti verranno persi.

### 3.5.6 Gestione della connessione TCP

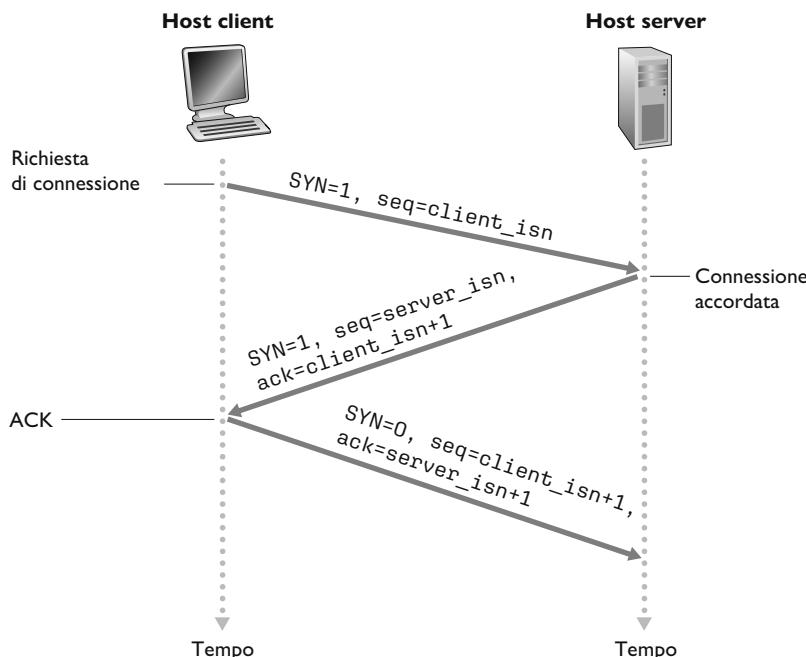
Diamo ora un’occhiata più approfondita a come viene stabilita e rilasciata una connessione TCP. L’argomento riveste una certa importanza, in quanto stabilire connessioni TCP può aggiungere ritardi chiaramente percepiti da chi, per esempio, naviga nel Web. Inoltre, molti dei più frequenti attacchi in rete, tra cui i ben noti attacchi di SYN flooding (si veda il BOX 3.4 “Attacco SYN flood”), sfruttano la vulnerabilità nella gestione della connessione TCP. Consideriamo innanzitutto come viene stabilita una connessione TCP. Supponiamo che un processo in esecuzione in un host (client) voglia inizializzare una connessione con un altro processo in un altro host (server). Il processo applicativo client dapprima informa il lato client di TCP di voler stabilire una connessione verso un processo nel server. Il TCP nel client quindi procede a stabilire una connessione TCP con il TCP nel server nel modo descritto di seguito.

- *Passo 1.* TCP lato client invia uno speciale segmento al TCP lato server. Questo segmento speciale non contiene dati a livello applicativo, ma uno dei bit nell’intestazione del segmento (Figura 3.29), il bit SYN, è posto a 1. Per questo motivo, tale segmento viene detto segmento SYN. Inoltre il client sceglie a caso un numero di sequenza iniziale (`client_isn`) e lo pone nel campo numero di sequenza del segmento SYN iniziale. Quest’ultimo viene incapsulato in un datagramma IP e inviato al server. Ci sono stati numerosi studi su come generare casualmente un appropriato `client_isn` al fine di evitare alcuni attacchi alla sicurezza [CERT 2001–09; RFC 4987].
- *Passo 2.* Quando il datagramma IP contenente il segmento TCP SYN arriva all’host server (ammesso che arrivi), il server estrae il segmento dal datagramma, alloca i buffer e le variabili TCP alla connessione e invia un segmento di connessione approvata al client TCP. Anche questo segmento non

contiene dati a livello applicativo, ma nella sua intestazione vi sono tre informazioni importanti. Innanzitutto, il bit SYN è posto a 1. In secondo luogo, il campo ACK assume il valore `client_isn+1`. Infine, il server sceglie il proprio numero di sequenza iniziale (`server_isn`) e lo pone nel campo del numero di sequenza. È come se il segmento stesse dicendo: “Ho ricevuto il tuo pacchetto SYN per iniziare una connessione con il tuo numero di sequenza iniziale, `client_isn`. Sono d'accordo nello stabilire questa connessione. Il mio numero di sequenza iniziale è `server_isn`”. Il segmento di connessione approvata viene talvolta detto **segmento SYNACK**.

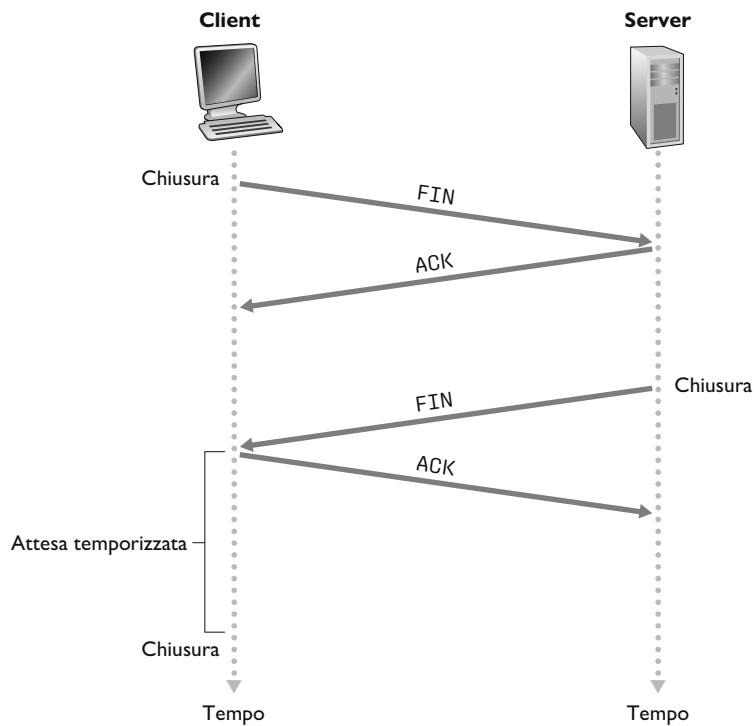
- *Passo 3.* Alla ricezione del segmento SYNACK, anche il client alloca buffer e variabili alla connessione. L'host client invia quindi al server un altro segmento in risposta al segmento di connessione approvata del server. Tale operazione viene svolta dal client ponendo il valore `server_isn+1` nel campo ACK dell'intestazione del segmento TCP. Il bit SYN è posto a zero, dato che la connessione è stata stabilita. In questo terzo passo dell'handshake a tre vie il campo dati del segmento può contenere informazioni che vanno dal client verso il server.

Una volta completati questi tre passi, gli host client e server possono scambiarsi segmenti contenenti dati. In ciascuno di questi futuri segmenti, il bit SYN sarà posto a zero. Notiamo che al fine di stabilire la connessione, i due host si scambiano tre pacchetti (Figura 3.39). Per questo motivo tale procedura viene detta **handshake a tre vie**. Diversi aspetti di questa procedura vengono esplorati nei Problemi alla fine del capitolo (perché sono richiesti i numeri di sequenza iniziale? Perché si richiede un handshake a tre vie anziché a due?). È interessante notare che un alpinista e il suo secondo di cordata utilizzano un protocollo di



**Figura 3.39** Handshake a tre vie di TCP: scambio di segmenti.

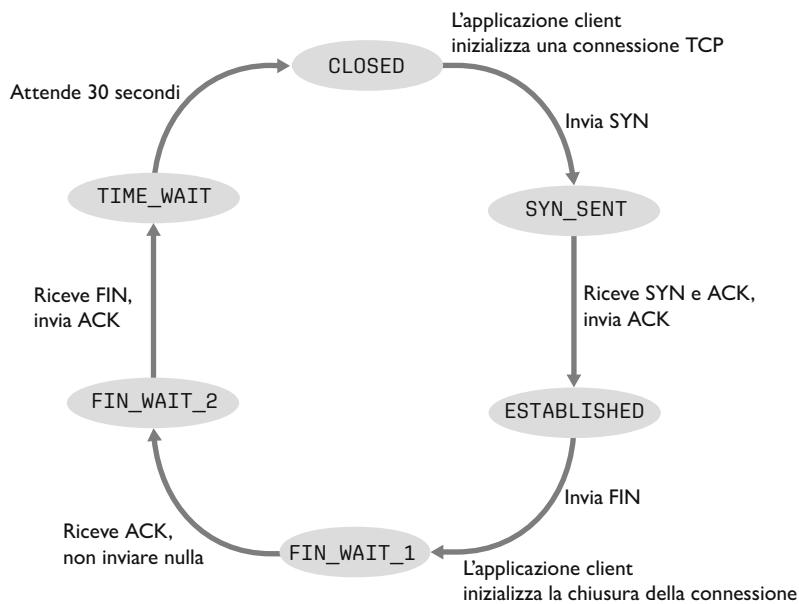
**Figura 3.40** Chiusura di una connessione TCP.



comunicazione con handshake a tre vie identico a quello del TCP per assicurarsi che entrambi i partecipanti siano pronti prima dell'inizio di un tratto impegnativo di una scalata.

Ogni cosa bella ha una fine, e questo è vero anche per le connessioni TCP. Ciascuno dei due processi che partecipano alla connessione può terminarla. E, in tal caso, le “risorse” negli host (ossia buffer e variabili) vengono deallocate. Per esempio, supponiamo che il client decida di chiudere la connessione (Figura 3.40). Il processo applicativo client invia un comando di chiusura, che forza il client TCP a inviare un segmento TCP speciale al processo server. Nell'intestazione di tale segmento troviamo il bit FIN (Figura 3.29) con valore 1. Quando il server riceve questo segmento, risponde inviando un acknowledgment al client. Il server spedisce quindi il proprio segmento di shutdown, con il bit FIN uguale a 1. Infine, il client manda a sua volta un acknowledgment a quest'ultimo segmento del server. A questo punto, tutte le risorse degli host risultano deallocate.

Nell'arco di una connessione TCP, i protocolli TCP in esecuzione negli host attraversano vari **stati TCP** (*TCP states*). La Figura 3.41 mostra una tipica sequenza di stati visitati dal client TCP. Quest'ultimo parte dallo stato CLOSED. L'applicazione sul lato client inizializza una nuova connessione TCP (creando un oggetto Socket nei nostri esempi in Python del Capitolo 2). Questo spinge il TCP nel client a inviare un segmento SYN al TCP nel server. Una volta spedito il segmento SYN, il client TCP entra nello stato SYN\_SENT, durante il quale attende dal server TCP un segmento con un acknowledgment per un precedente segmento del client e che abbia il bit SYN posto a 1. Una volta ricevuto tale segmento,



**Figura 3.41** Tipica sequenza di stati visitati da un client TCP.

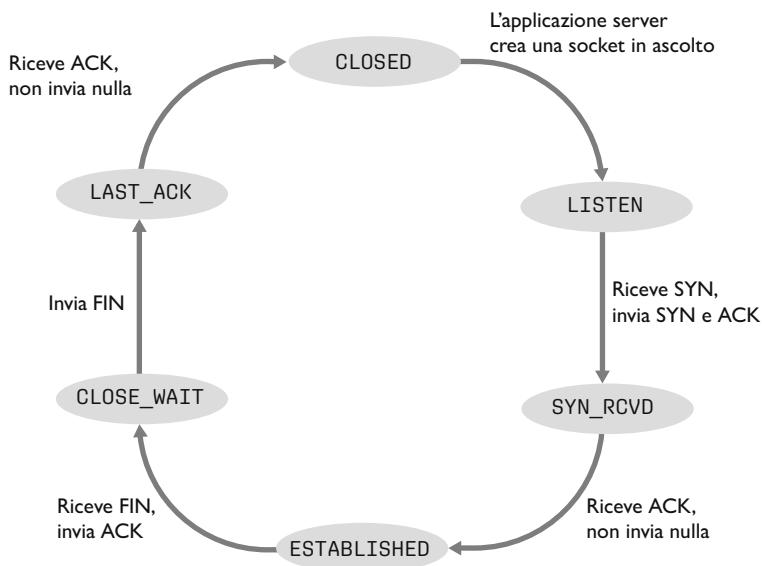
il client TCP entra nello stato ESTABLISHED, durante il quale può inviare e ricevere segmenti che contengono dati utili (ossia generati dall'applicazione).

Supponiamo che l'applicazione client decida di voler chiudere la connessione. Notiamo che anche il server potrebbe scegliere di terminarla. Ciò spinge il client TCP a inviare un segmento con il bit FIN impostato a 1 e a entrare nello stato FIN\_WAIT\_1. Quando si trova in questo stato, il client TCP attende dal server un segmento TCP con un acknowledgment e, quando lo riceve, entra nello stato FIN\_WAIT\_2, in cui il client attende un altro segmento dal server con bit FIN impostato a 1. Dopo aver ricevuto questo segmento, il client TCP manda un acknowledgment ed entra nello stato TIME\_WAIT, che consente al client TCP di inviare nuovamente l'ultimo acknowledgment nel caso in cui l'ACK vada perduto. Il tempo trascorso nello stato TIME\_WAIT dipende dall'implementazione, ma i valori tipici sono di 30 secondi, di 1 o di 2 minuti. Dopo l'attesa, la connessione viene formalmente chiusa e tutte le risorse sul lato client (compresi i numeri di porta) vengono rilasciate.

La Figura 3.42 mostra una tipica serie di stati visitati dal lato server di TCP, in cui il client avvia la chiusura della connessione. Le transizioni sono abbastanza auto-esplicative. In questi due diagrammi abbiamo mostrato la normale apertura e chiusura di una connessione TCP, mentre non abbiamo descritto che cosa succeda in determinati scenari “patologici”, per esempio quando entrambi i lati di una connessione vogliono iniziare o chiuderla nello stesso istante. Se siete interessati all'argomento, vi invitiamo a leggere l'esaustivo libro di Stevens [Stevens 1994].

La trattazione fin qui svolta ha assunto che client e server siano entrambi preparati alla comunicazione, cioè che il server sia in ascolto sulla porta alla quale il client invierà il suo segmento di SYN. Consideriamo ora che cosa succede quando un host riceva un segmento TCP i cui numeri di porta o il cui in-

**Figura 3.42** Tipica sequenza di stati visitati da un server TCP.



dirizzo IP di origine non corrispondano ad alcuna socket attiva nell'host. Per esempio, supponiamo che l'host riceva un pacchetto TCP SYN con porta di destinazione 80, ma che non stia accettando connessioni su quella porta (ossia, non ha in esecuzione un web server sulla porta 80). In tal caso invierà al mittente un segmento speciale di reset con il bit RST (Paragrafo 3.5.2) impostato a 1, allo scopo di comunicare alla sorgente: “Non ho una socket per quel segmento. Per favore non rimandarlo”. Quando un host riceve un pacchetto UDP il cui numero di porta di destinazione non corrisponde a una socket UDP attiva, invia uno speciale datagramma ICMP, come trattato nel Capitolo 5.

Ora che abbiamo raggiunto un buon livello di comprensione della gestione della connessione TCP analizziamo di nuovo lo strumento per la scansione delle porte nmap ed esaminiamo attentamente come funziona. Per analizzare una specifica porta TCP, per esempio 6789, su un host bersaglio, nmap manderà a quell'host un segmento TCP SYN con porta di destinazione 6789. I possibili risultati sono i seguenti tre.

- *L'host sorgente riceve un segmento TCP SYNACK dall'host bersaglio.* Dato che questo significa che un'applicazione è in esecuzione con la porta TCP 6789 sul sistema bersaglio, nmap restituisce “open” (aperta).
- *L'host sorgente riceve un segmento TCP RST dall'host bersaglio.* Questo significa che il segmento SYN ha raggiunto l'host bersaglio, ma su quest'ultimo non è in esecuzione alcuna applicazione che usa la porta TCP 6789. L'aggressore tuttavia sa almeno che il segmento destinato all'host sulla porta 6789 non è bloccato da alcun firewall sul percorso tra la sorgente e l'host bersaglio (i firewall verranno trattati nel Capitolo 8, disponibile in inglese sulla piattaforma MyLab).
- *La sorgente non riceve nulla.* Questo probabilmente significa che il segmento SYN è stato bloccato da un firewall che è intervenuto nella comunicazione e non ha mai raggiunto l'host bersaglio.

Nmap è uno strumento potente, che può raccogliere informazioni non solo sulle porte TCP aperte, ma anche sulle porte UDP aperte, sui firewall e sulle loro configurazioni, e anche sulle versioni delle applicazioni e dei sistemi operativi.

**BOX 3.4**
**FOCUS SULLA SICUREZZA**

### Attacco SYN flood

Abbiamo visto nella nostra trattazione dell'handshake a tre vie di TCP che un server alloca e inizializza le variabili e i buffer della connessione in risposta a un SYN ricevuto. Il server manda poi un SYNACK in risposta e attende un segmento di ACK dal client, terzo e ultimo passo nell'handshake prima che la connessione sia completamente instaurata. Se il client non manda un ACK per completare il terzo passo dell'handshake a tre vie, alla fine (spesso dopo un minuto o più) il server termina la connessione mezza aperta e dealloca le risorse.

Questo protocollo di gestione della connessione TCP pone le basi per un classico attacco DoS (*Denial of Service*, negazione del servizio), chiamato **attacco SYN flood** (*SYN flood attack*). In questo attacco l'aggressore (o attaccante) manda un gran numero di segmenti TCP SYN, senza completare il terzo passo dell'handshake. L'attacco può essere amplificato mandando i SYN da più sorgenti e creando così un attacco SYN flood di tipo DDoS (*Distributed Denial of Service*). Con questa inondazione di segmenti SYN, le risorse del server riservate alle connessioni possono esaurirsi velocemente perché allocate (ma mai usate) alle connessioni mezze aperte. Se le risorse del server sono esaurite, agli utenti legittimi è negato il servizio. Gli attacchi SYN flood sono tra i primi attacchi DoS di cui si ha notizia [CERT SYN 1996]. Fortunatamente, esiste una difesa efficace, chiamata **SYN cookie** [RFC 4987] già inclusa nella maggior parte dei sistemi operativi; i SYN cookie funzionano come segue.

- Quando un server riceve un segmento SYN, non sa se il segmento arriva da un utente legittimo o se è parte di un attacco SYN flood. Quindi non crea una connessione TCP mezza aperta, ma crea un numero di sequenza TCP iniziale come funzione hash degli indirizzi IP e numeri di porta di sorgente e destinazione del segmento SYN e una chiave segreta nota solo al server. Il server usa la stessa chiave segreta per un ampio numero di connessioni. Questo numero di sequenza iniziale, attentamente costruito, è il cosiddetto “cookie”. Il server manda poi un pacchetto SYNACK con questo numero di sequenza iniziale. *L'aspetto più importante è che il server non memorizza il cookie o qualsiasi altra informazione di stato corrispondente al SYN.*
- Se il client è legittimo risponde con un segmento di ACK. Il server, alla ricezione di questo ACK, ha necessità di verificare se corrisponde allo stesso SYN inviato precedentemente. Come si fa, se il server non ha memoria sui segmenti SYN? Come potete immaginare, viene fatto con il cookie. Specificatamente, per un ACK legittimo, il valore nel campo di acknowledgment è uguale al numero di sequenza del SYNACK (il valore del cookie) più uno (Figura 3.39). Il server eseguirà poi la stessa funzione hash citata prima usando i campi del segmento SYNACK (che erano poi gli stessi del SYN originale) e la chiave segreta. Se il risultato della funzione più uno è lo stesso numero del campo di acknowledgment, il server conclude che l'ACK corrisponde al precedente segmento SYN e quindi è valido. Il server crea quindi una connessione TCP completamente aperta e una socket.
- Viceversa, se il client non risponde con un segmento di ACK, allora il SYN originale non ha fatto danni, in quanto non erano state allocate risorse.

Molto di ciò viene fatto manipolando i segmenti per la gestione delle connessioni TCP. Potete scaricare nmap da <http://www.nmap.org>.

Questo completa la nostra introduzione al controllo degli errori e di flusso in TCP. Nel Paragrafo 3.7 torneremo a TCP e analizzeremo in dettaglio il suo controllo di congestione. Prima di farlo, tuttavia, facciamo un passo indietro ed esaminiamo i problemi relativi al controllo di congestione in un contesto più ampio.

## 3.6 Princìpi del controllo di congestione

Nei precedenti paragrafi abbiamo visto, sia in generale sia rispetto a TCP, quali siano i meccanismi adottati per offrire un servizio di trasferimento dati affidabile a fronte di una perdita di pacchetti. Abbiamo menzionato che, in pratica, tale perdita è tipicamente il risultato di un overflow dei buffer nei router quando il traffico in rete diventa eccessivo. La ritrasmissione di pacchetti tratta quindi un sintomo della congestione di rete (la perdita di uno specifico segmento a livello di trasporto), ma non le cause della congestione di rete: il tentativo da parte di troppe sorgenti di inviare dati a ritmi troppo elevati. In questo caso sono richiesti meccanismi per adeguare l'attività dei mittenti in relazione al traffico.

Consideriamo ora il problema del controllo di congestione in un contesto generale, cercando di comprendere perché la congestione sia un aspetto negativo, come si manifesti nelle prestazioni delle applicazioni dei livelli superiori e i vari approcci che possono essere scelti per evitarla o reagire correttamente. Tale approfondimento è opportuno dato che, esattamente come il trasferimento affidabile dei dati, si trova in cima alla nostra lista dei dieci problemi più importanti nel networking.

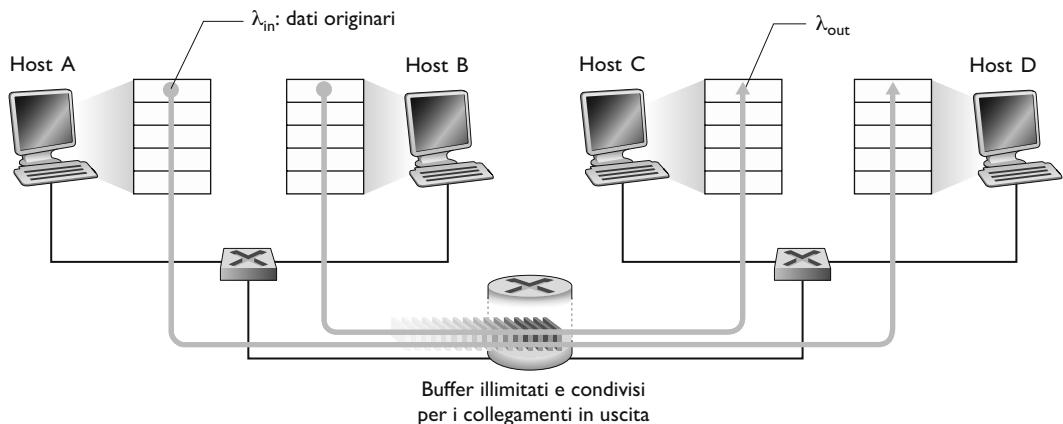
### 3.6.1 Cause e costi della congestione

Cominciamo la nostra trattazione generale del controllo di congestione esaminando tre scenari via via più complessi. In ciascun caso vedremo perché la rete sia congestionata e ne valuteremo le conseguenze in termini di basso utilizzo delle risorse e di scarse prestazioni percepite dai sistemi periferici. Per ora non ci concentreremo su come reagire alla congestione o come evitarla, ma piuttosto sul compito più semplice di comprendere che cosa avvenga quando gli host aumentano il proprio tasso trasmissivo e le reti diventano congestionate.

#### Scenario 1: due mittenti e un router con buffer illimitati

Iniziamo considerando lo scenario di congestione più semplice possibile: due host (A e B) con una connessione che condivide un singolo router intermedio (Figura 3.43).

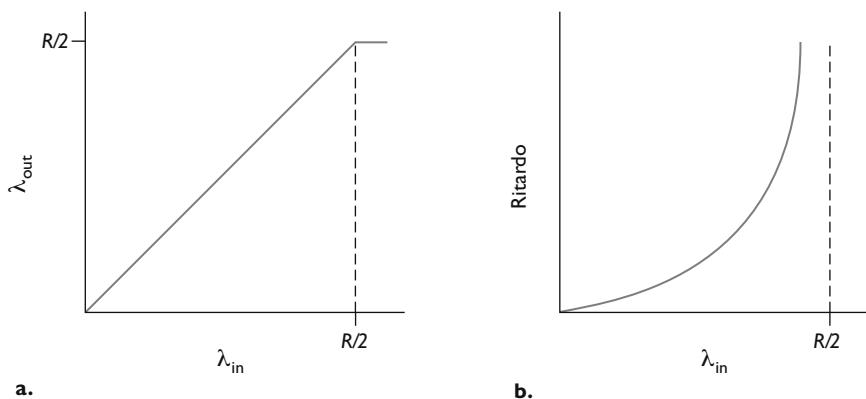
Ipotizziamo che un'applicazione nell'Host A stia inviando dati sulla connessione (per esempio, stia passando dati al protocollo a livello di trasporto attraverso una socket) a una frequenza media di  $\lambda_{in}$  byte/s. Tali dati sono originali, nel senso che ciascuna unità di dati viene mandata nella socket solo una volta. Il sottostante protocollo a livello di trasporto è semplice. I dati vengono incapsulati e inviati, senza porre rimedio a eventuali errori (per esempio tramite ritrasmissione), controllo di flusso o di congestione. Ignorando l'overhead ag-



**Figura 3.43** Scenario di congestione 1: due connessioni che condividono un hop con buffer illimitato.

giuntivo dovuto alle informazioni dell'intestazione di trasporto e dell'intestazione del livello inferiore, il tasso al quale l'Host A presenta traffico al router in questo primo scenario è pertanto  $\lambda_{in}$  byte/s. L'Host B opera in modo simile, e noi assumeremo per semplicità che stia trasmettendo anch'esso a  $\lambda_{in}$  byte/s. I pacchetti dall'Host A e dall'Host B passano attraverso un router e un collegamento uscente condiviso di capacità  $R$ . Il router possiede buffer che gli consentono di memorizzare i pacchetti entranti quando il tasso di arrivo dei pacchetti supera la capacità del collegamento uscente. In questo primo scenario ipotizziamo inoltre che i buffer del router abbiano dimensione illimitata.

La Figura 3.44 illustra le prestazioni della connessione dell'Host A in questo primo scenario. Il grafico di sinistra mostra il **throughput per connessione** (numero di byte per secondo al ricevente) in funzione del tasso di invio. Finché non supera il valore di  $R/2$ , il throughput del ricevente equivale al tasso di invio del mittente: tutto quello che viene trasmesso dal mittente viene ricevuto dal destinatario con un ritardo finito. Ma se il tasso di invio supera  $R/2$ , il throughput resta  $R/2$ . Questo limite superiore sul throughput è conseguenza della condivisione della capacità di collegamento tra le due connessioni. Il collega-



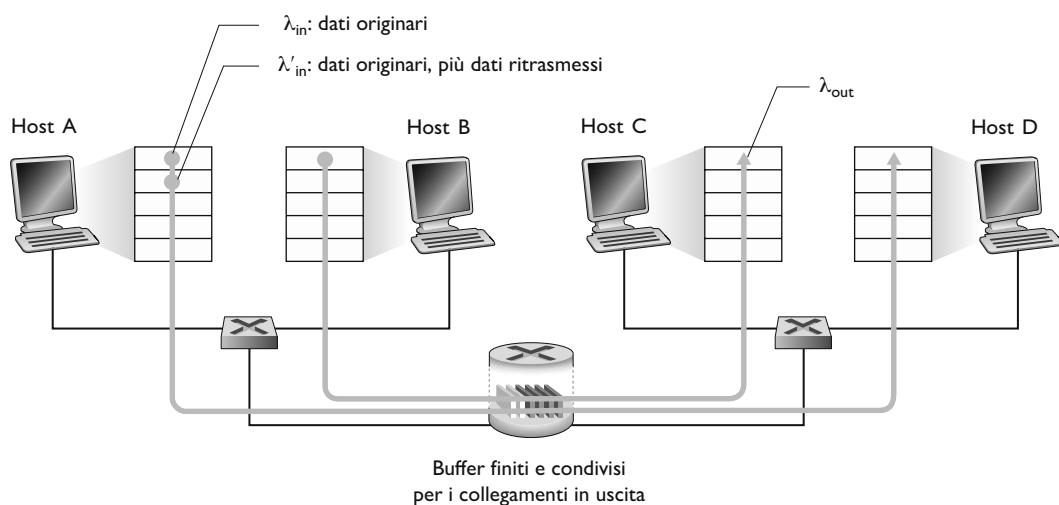
**Figura 3.44** Scenario di congestione 1: throughput e ritardi in funzione della frequenza trasmittiva dell'host.

mento, semplicemente, non è in grado di consegnare pacchetti al destinatario a un tasso superiore a  $R/2$ . Per quanto elevata sia il tasso di invio da parte degli Host, né A né B avranno mai un throughput superiore a  $R/2$ .

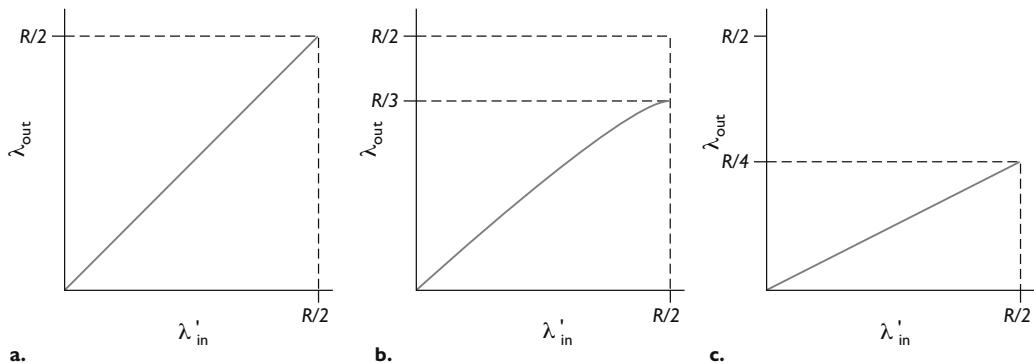
Ottenere un throughput per connessione pari a  $R/2$  potrebbe in effetti sembrare un ottimo risultato, dato che il collegamento viene completamente utilizzato nella consegna di pacchetti alla propria destinazione. Il grafico a destra nella Figura 3.44 mostra però le conseguenze di operare al limite della capacità di collegamento. Quando il tasso di invio si avvicina a  $R/2$  (da sinistra), il ritardo medio cresce sempre più. Quando supera  $R/2$ , il numero medio di pacchetti in coda nel router cresce senza limite, e il ritardo medio tra origine e destinazione tende all'infinito (nell'ipotesi che le connessioni mantengano questo tasso di invio per un periodo di tempo infinito e che la capacità dei buffer sia infinita). Di conseguenza, avere un throughput aggregato vicino a  $R$  potrebbe sembrare ideale dal punto di vista del throughput, ma non lo è certo dal punto di vista del ritardo. Perfino in questo scenario (estremamente) idealizzato abbiamo già trovato un costo dovuto alla congestione delle reti: quando il tasso di arrivo dei pacchetti si avvicina alla capacità del collegamento, si rilevano lunghi ritardi di accodamento.

### Scenario 2: due mittenti e un router con buffer limitati

Una prima modifica dello scenario precedente consiste nell'assumere che la dimensione dei buffer nel router sia limitata (Figura 3.45). Una conseguenza pratica di quest'ipotesi è che i pacchetti che giungono in un buffer già pieno sono scartati. In secondo luogo, supponiamo che le due connessioni siano affidabili: se un pacchetto che contiene un segmento a livello di trasporto viene scartato dal router, il mittente prima o poi lo ritrasmetterà. Dato che i pacchetti possono essere ritrasmessi, dobbiamo ora prestare attenzione all'uso della locuzione “tasso di invio” (*sending rate*).



**Figura 3.45** Scenario 2: due host (con ritrasmisioni) e un router con buffer di dimensione finita.



**Figura 3.46** Prestazioni dello Scenario 2 con buffer di dimensione finita.

In particolare, denotiamo ancora il tasso di trasmissione (*transmission rate*) verso la socket con  $\lambda_{\text{in}}$  byte/s, e indichiamo con  $\lambda'_{\text{in}}$  byte/s il tasso al quale il livello di trasporto invia segmenti (contenenti dati originali e dati ritrasmessi), una grandezza talvolta detta **carico offerto** (*offered load*) alla rete.

Le prestazioni che si riscontrano in questo scenario dipendono fortemente da come si effettua la ritrasmissione. Innanzitutto, consideriamo il caso poco probabile in cui l'Host A sia in grado di determinare, come per magia, se il buffer nel router abbia spazio a disposizione e trasmetta pertanto un pacchetto solo quando il buffer è libero. In questo caso non si verificherebbe alcun smarrimento, avremmo  $\lambda'_{\text{in}} = \lambda_{\text{in}}$  e il throughput della connessione sarebbe  $\lambda_{\text{in}}$ , caso mostrato nella Figura 3.46(a). Dal punto di vista del throughput, le prestazioni sono ideali: tutto quanto viene trasmesso è ricevuto. Notiamo che la velocità di invio media dell'host in questo scenario non supera  $R/2$ , visto che abbiamo ipotizzato che nessun pacchetto vada smarrito.

Consideriamo ora il caso, un po' più realistico, in cui il mittente ritrasmette solo quando è certo che un pacchetto sia andato perduto. Un'ipotesi leggermente forzata. Tuttavia, l'host mittente potrebbe impostare il proprio timeout con un valore sufficientemente grande da essere praticamente certo che il pacchetto di cui non si è ancora ricevuto acknowledgment sia stato perduto. In questo caso, le prestazioni potrebbero avere l'aspetto mostrato nella Figura 3.46(b). Per comprendere appieno che cosa stia avvenendo, consideriamo il caso in cui il carico offerto,  $\lambda'_{\text{in}}$  (il tasso di trasmissione dei dati originali più le ritrasmissioni) valga  $R/2$ . Secondo la Figura 3.46(b), con questo valore di carico offerto alla rete, il tasso con cui i dati vengono consegnati all'applicazione destinataria è  $R/3$ . Quindi, su  $0,5 R$  unità di dati trasmessi,  $0,333 R$  byte/s (in media) sono quelli originali e  $0,166 R$  byte/s (in media) sono quelli ritrasmessi. Rileviamo così un altro costo legato alla congestione di rete: il mittente deve effettuare ritrasmissioni per compensare i pacchetti scartati (perduti) a causa dell'overflow nei buffer.

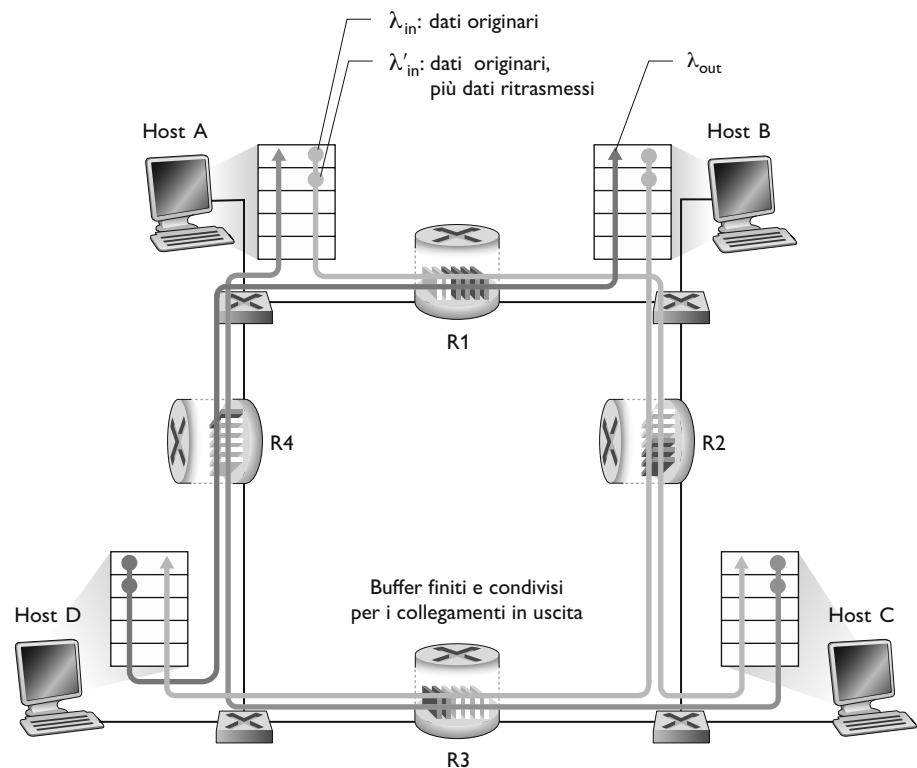
Prendiamo in esame infine la situazione in cui il mittente possa andare in timeout prematuramente e ritrasmettere un pacchetto che abbia subito ritardi in coda, ma non sia stato perduto. In questo caso, sia il pacchetto originale sia quello ritrasmesso possono raggiungere il destinatario. Ovviamente, al destina-

tario è sufficiente una copia di tale pacchetto e scarterà le altre. In questo caso, il lavoro effettuato dal router per instradare la copia ritrasmessa del pacchetto è sprecato, dato che il destinatario avrà già ricevuto la copia originale. Il router avrebbe potuto utilizzare meglio la capacità trasmissiva del collegamento trasmettendo un altro pacchetto. Ecco un ulteriore costo legato alla congestione di rete: ritrasmissioni non necessarie da parte del mittente come risposta a lunghi ritardi possono costringere un router a utilizzare la larghezza di banda del collegamento per instradare copie non necessarie di un pacchetto. La Figura 3.46(c) confronta throughput e traffico immesso nella rete nell'ipotesi che ciascun pacchetto sia instradato mediamente due volte dal router. In questo scenario, il throughput assumerà asintoticamente il valore  $R/4$  quando il carico offerto tende a  $R/2$ .

### Scenario 3: quattro mittenti, router con buffer finiti e percorsi composti da più collegamenti

In questo caso supponiamo che i pacchetti siano trasmessi da quattro host, ciascuno su percorsi composti da due collegamenti sovrapposti tra loro (Figura 3.47); ciascun host, inoltre, utilizza un meccanismo di timeout e ritrasmissione per implementare il servizio affidabile di trasferimento dati, e tutti e quattro hanno lo stesso valore di  $\lambda_{in}$ . Supponiamo anche che la capacità dei collegamenti dei router sia di  $R$  byte/s.

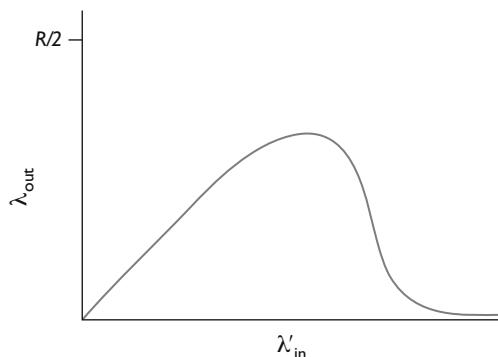
**Figura 3.47** Scenario 3:  
quattro mittenti, router  
con buffer di dimensione  
finita e percorsi multihop.



Consideriamo la connessione dall'Host A all'Host C che passa per i router R1 e R2. Questa connessione condivide il router R1 con la connessione D-B e il router R2 con la connessione B-D. Per valori estremamente piccoli di  $\lambda_{in}$ , gli overflow dei buffer sono rari (come negli Scenari 1 e 2), e il throughput è approssimativamente uguale al traffico inviato in rete. Per valori leggermente più grandi di  $\lambda_{in}$ , il corrispondente throughput è anch'esso più grande, dato che più dati originali vengono trasmessi nella rete e consegnati alla destinazione, mentre gli overflow sono ancora piuttosto rari. Di conseguenza, per piccoli valori di  $\lambda_{in}$ , un incremento di  $\lambda_{in}$  provoca un incremento di  $\lambda_{out}$ .

Una volta esaminato il caso di traffico estremamente scarso, passiamo a considerare il caso in cui  $\lambda_{in}$  (e quindi  $\lambda'_{in}$ ) sia molto grande. Prendiamo in considerazione il router R2. Il traffico da A verso C che giunge al router R2, dopo essere stato infiltrato da R1, non può presentare un tasso di arrivo maggiore di  $R$ , la capacità del collegamento da R1 a R2, indipendentemente dal valore di  $\lambda_{in}$ . Se  $\lambda'_{in}$  è estremamente grande per tutte le connessioni (B-D inclusa), il tasso di arrivo del traffico B-D su R2 può essere molto più elevato di quello del traffico A-C. Dato che sul router R2 il traffico da A verso C e quello da B verso D sono in competizione per il limitato spazio nei buffer, la quantità di traffico A-C che passa con successo attraverso R2 (ossia, che non viene persa a causa dell'overflow) diventa sempre più piccola al crescere del traffico trasportato da B-D. Al limite, quando questo tende a infinito, un buffer vuoto presso R2 viene immediatamente colmato da un pacchetto B-D, e il throughput della connessione A-C presso R2 tende a 0. Ne segue che il throughput end-to-end di A-C si annulla in caso di traffico pesante. Queste considerazioni originano il compromesso tra traffico inviato e throughput mostrato nella Figura 3.48.

Il motivo della diminuzione del throughput al crescere del traffico diventa evidente quando si considera la quantità di lavoro sprecato da parte della rete. Nello scenario di traffico intenso che abbiamo delineato, ogni qualvolta un pacchetto viene scartato sul router del secondo hop, il lavoro effettuato dal router del primo hop nell'instradamento del pacchetto verso il secondo router finisce per essere "sprecato". La rete avrebbe funzionato altrettanto bene (anzi, altrettanto male) se il primo router avesse semplicemente scartato il pacchetto e fosse rimasto inattivo. Più nello specifico, la capacità trasmissiva utilizzata dal primo router per instradare il pacchetto al secondo potrebbe essere utilizzata in modo più proficuo trasmettendo un altro pacchetto. Per esempio, nel selezionare un



**Figura 3.48** Prestazioni dello Scenario 3 (buffer di dimensione finita e percorsi multihop).

pacchetto per la trasmissione, sarebbe preferibile che il router desse priorità ai pacchetti che hanno già superato un certo numero di router. Quindi, anche in questo caso, vediamo un costo legato all'eliminazione dei pacchetti per via della congestione: quando un pacchetto viene scartato lungo il percorso, la capacità trasmisiva, utilizzata sui collegamenti per instradare il pacchetto fino al punto in cui è scartato, risulta sprecata.

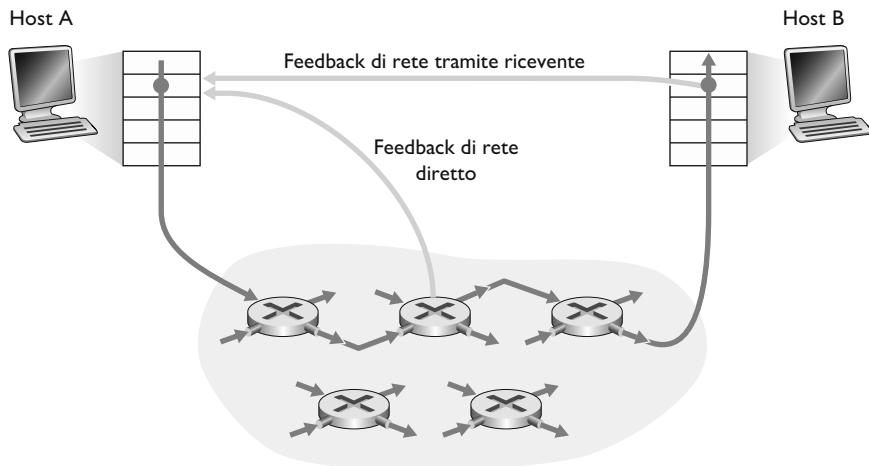
### 3.6.2 Approcci al controllo di congestione

Nel Paragrafo 3.7 approfondiremo dettagliatamente l'approccio specifico di TCP al controllo di congestione. Per il momento ci limitiamo a identificare i due principali orientamenti al controllo di congestione utilizzati nella pratica e a discuterne le relative architetture di rete e i protocolli.

Ad alto livello, possiamo distinguere tra livelli di rete che offrono o meno assistenza esplicita al livello di trasporto al fine di controllare la congestione.

- *Controllo di congestione end-to-end.* Il livello di rete non fornisce supporto esplicito al livello di trasporto per il controllo di congestione la cui presenza deve essere dedotta dai sistemi periferici sulla base dell'osservazione del comportamento della rete (per esempio, perdita di pacchetti e ritardi). Vedremo nel Paragrafo 3.7.1 che TCP deve necessariamente utilizzare questo approccio end-to-end, dato che il livello IP non offre feedback relativamente alla congestione della rete. La perdita di segmenti TCP (indicata da un timeout o da acknowledgment triplicati) viene considerata chiara indicazione di congestione di rete e TCP diminuisce, di conseguenza, l'ampiezza della propria finestra. Vedremo anche che le nuove proposte di TCP fanno uso di valori crescenti di ritardo di RTT, come indicatori di traffico sempre più intenso sulla rete.
- *Controllo di congestione assistito dalla rete.* I componenti a livello di rete (ossia i router) forniscono un feedback esplicito al mittente sullo stato di congestione della rete. Questo avviso può essere semplicemente un bit che indica traffico su un collegamento; tale approccio è adottato nelle prime architetture SNA di IBM [Schwartz 1982] e DECnet della DEC [Jain 1989; Ramakrishnan 1990] e ATM [Black 1995]. È anche possibile fare uso di un feedback di rete più sofisticato. Per esempio, una forma di **controllo di congestione ATM ABR** (*Available Bit Rate*) consente a un router di informare il mittente in modo esplicito sulla frequenza trasmisiva che il router può supportare su un collegamento uscente. Le versioni di default di TCP e IP in Internet adottano l'approccio end-to-end, ma possono anche implementare l'opzione di controllo di congestione assistito dalla rete, come vedremo nel Paragrafo 3.7.2.

Nel caso del controllo assistito dalla rete, l'informazione di congestione viene solitamente fornita dalla rete al mittente in due modi (Figura 3.49). Può essere trasmesso un avviso diretto da un router al mittente tramite un **chokepacket** (“pacchetto di strozzatura”), che riferisce: “Sono congestionato!”. Il secondo tipo di notifica ha luogo quando un router imposta un campo in un pacchetto che fluisce dal mittente al destinatario, per indicare congestione. Alla ricezione



**Figura 3.49** Due modalità per ricevere feedback riguardo alla congestione di rete.

di un pacchetto marcato, il destinatario notifica al mittente l'indicazione di congestione. Notiamo che questa forma di notifica richiede quantomeno un *RTT*.

## 3.7 Controllo di congestione TCP

Torniamo ora al nostro studio di TCP che, come abbiamo visto nel Paragrafo 3.5, offre un servizio affidabile di trasporto tra due processi in esecuzione su host diversi e presenta, come altro componente chiave, il meccanismo di controllo della congestione.

Come indicato nel paragrafo precedente, quello che potremmo chiamare TCP “classico”, la versione di TCP standardizzata in [RFC 2581] e più recentemente in [RFC 5681], utilizza il controllo di congestione end-to-end anziché il controllo di congestione assistito dalla rete, in quanto il livello IP non fornisce ai sistemi periferici alcun feedback esplicito per quanto riguarda la congestione della rete. Cominceremo col trattare approfonditamente questa versione “classica” di TCP nel Paragrafo 7.3.1 (il Capitolo 7 è disponibile in inglese sulla piattaforma MyLab). Esamineremo quindi nel Paragrafo 7.3.2 le versioni più recenti di TCP che utilizzano un’indicazione di congestione esplicita fornita dal livello di rete e presentano molte altre lievi differenze rispetto a TCP “classico”. Vedremo quindi come affrontare la sfida a livello di trasporto di fornire equità tra i flussi che condividono un collegamento congestionato.

### 3.7.1 Controllo di congestione di TCP classico

L’approccio scelto da TCP consiste nell’imporre a ciascun mittente un limite al tasso di invio sulla propria connessione in funzione della congestione di rete percepita. Se il mittente TCP si accorge di condizioni di scarso traffico sul percorso che porta alla destinazione, incrementa il proprio tasso di trasmissione; se, invece, percepisce traffico lungo il percorso, lo riduce. Ma tale approccio solleva tre domande. Innanzitutto, come può il mittente TCP limitare il tasso di invio del traffico sulla propria connessione? Secondo, come percepisce la congestione sul percorso che porta alla destinazione? E, infine, quale algoritmo

dovrebbe essere usato dal mittente per variare il tasso di invio in funzione della congestione end-to-end?

Innanzitutto, esaminiamo come TCP possa ridurre la velocità di invio del traffico sulla propria connessione. Nel Paragrafo 3.5 abbiamo visto che gli estremi di una connessione TCP gestiscono un buffer di ricezione, uno di invio e diverse variabili tra cui `LastByteRead` e `rwnd`. Il meccanismo di controllo di congestione TCP fa tener traccia agli estremi della connessione di una variabile aggiuntiva: la **finestra di congestione** (*congestion window*), indicata con `cwnd`, che impone un vincolo alla velocità di immissione di traffico sulla rete da parte del mittente. Nello specifico, la quantità di dati che non hanno ancora ricevuto acknowledgment inviata da un mittente non può eccedere il minimo tra i valori di `cwnd` e `rwnd`, ossia:

$$\text{LastByteSent} - \text{LastByteAcked} \leq \min\{\text{cwnd}, \text{rwnd}\}$$

Al fine di concentrarci sul controllo di congestione (anziché su quello di flusso), assumiamo che il buffer di ricezione sia sufficientemente capiente da poter ignorare il vincolo della finestra di ricezione; pertanto, la quantità di dati che non hanno ricevuto acknowledgment è limitata soltanto da `cwnd`. Assumiamo inoltre che il mittente abbia sempre dati da inviare, per cui la finestra di congestione sia sempre completamente in uso.

Osserviamo che il vincolo sopra citato limita il tasso di trasmissione del mittente soltanto in modo indiretto. Per rendercene conto, consideriamo una connessione in cui la perdita di pacchetti e i ritardi di trasmissione siano trascurabili. Di conseguenza, approssimativamente all'inizio di ogni *RTT*, il vincolo consente al mittente di trasmettere `cwnd` byte di dati sulla connessione; al termine del *RTT* il mittente riceve gli acknowledgment relativi ai dati. Quindi, il tasso di invio del mittente è approssimativamente  $\text{cwnd}/\text{RTT}$  byte/s. Modificando il valore di `cwnd`, il mittente può regolare la velocità di invio dei dati sulla propria connessione.

Consideriamo ora come il mittente TCP percepisce la presenza di congestione sul percorso verso la destinazione. Definiamo “evento di perdita” per il mittente TCP l’occorrenza o di un timeout o della ricezione di tre ACK duplicati da parte del destinatario (ricordiamo la nostra discussione nel Paragrafo 3.5.4 dell’evento di timeout nella Figura 3.33 e la successiva modifica per includere la ritrasmissione rapida alla ricezione di tre ACK duplicati). In presenza di una congestione eccessiva, uno o più buffer dei router lungo il percorso vanno in overflow, causando l’eliminazione di un datagramma (che contiene un segmento TCP). Il datagramma eliminato, a sua volta, costituisce un evento di perdita presso il mittente (sotto forma di timeout o come ricezione di tre ACK duplicati), che lo considera come un’indicazione di congestione sul percorso tra sé e il destinatario.

Avendo considerato come viene rilevata la congestione, esaminiamo il caso più ottimistico di una rete priva di congestione, in cui non si verificano smarimenti. In questo scenario, gli acknowledgment relativi ai vari segmenti verranno ricevuti dal mittente TCP. Come vedremo, TCP considera l’arrivo di tali acknowledgment come un’indicazione che tutto va bene, ossia che i segmenti trasmessi sulla rete sono stati consegnati con successo a destinazione e utilizza gli acknowledgment per aumentare l’ampiezza della propria finestra di congestio-

ne (e, di conseguenza, la velocità di trasmissione). Notiamo che se gli acknowledgment arrivano con frequenza relativamente bassa (per esempio, se il percorso end-to-end presenta ritardi elevati o transita per un collegamento lento), allora la finestra di congestione verrà ampliata piuttosto lentamente. Se, invece, gli acknowledgment giungono con una frequenza alta, allora la finestra di congestione verrà ampliata più rapidamente. Dato che TCP utilizza gli acknowledgment per scatenare (o temporizzare) gli incrementi dell'ampiezza della finestra di congestione, si dice che TCP è **auto-temporizzato** (*self-clocking*).

Dato il meccanismo di modifica del valore di cwnd per controllare il tasso di trasmissione, la questione critica è come un mittente TCP debba determinare il tasso a cui dovrebbe trasmettere. Se i mittenti TCP tutti insieme trasmettessero troppo velocemente, potrebbero congestionare la rete portandola al tipo di collasso di congestione mostrato nella Figura 3.48. Quindi, la versione di TCP che studieremo è stata sviluppata in risposta al collasso di congestione osservato in Internet [Jacobson 1988] con le versioni precedenti di TCP. Tuttavia, se i mittenti TCP fossero troppo cauti e trasmettessero troppo lentamente, potrebbero sottoutilizzare l'ampiezza di banda della rete; i mittenti TCP potrebbero trasmettere a tasso più elevato senza congestionare la rete. Allora come fanno i mittenti TCP a determinare la loro velocità di trasmissione in modo da non congestionare la rete, ma allo stesso tempo utilizzare tutta la banda disponibile? I mittenti TCP sono esplicitamente coordinati o esiste un approccio distribuito in cui stabiliscono i loro tassi di trasmissione basandosi solo su informazioni locali?

TCP risponde a queste domande sulla base dei seguenti principi guida.

- *Un segmento perso implica congestione, quindi i tassi di trasmissione del mittente TCP dovrebbero essere decrementati quando un segmento viene perso.* Ricordiamo dal Paragrafo 3.5.4 che un evento di timeout o la ricezione di quattro acknowledgment per un dato segmento (uno originale e tre duplicati) viene interpretato come un'indicazione implicita di “evento di perdita” del segmento che è stato seguito da quattro acknowledgment, e ciò scatena una ritrasmissione del segmento ritenuto perso. Dal punto di vista del controllo di congestione la domanda è come il mittente TCP debba decrementare l'ampiezza della sua finestra di congestione e quindi il proprio tasso di trasmissione in risposta a questo evento di perdita.
- *Un acknowledgment indica che la rete sta consegnando i segmenti del mittente al ricevente e quindi il tasso di trasmissione del mittente può essere aumentato quando arriva un acknowledgment non duplicato.* L'arrivo degli acknowledgment viene interpretato come un'indicazione implicita di funzionamento della trasmissione: i segmenti vengono consegnati con successo dal mittente al ricevente e quindi la rete non è congestionata. Di conseguenza la finestra di congestione può essere incrementata.
- *Rilevamento della larghezza di banda.* Avendo acknowledgment che indicano che il percorso dalla sorgente alla destinazione è privo di congestione ed eventi di perdita che indicano un percorso congestionato, la strategia di TCP per regolare il tasso di trasmissione è di incrementarlo in risposta all'arrivo di acknowledgment finché non si verifica un evento di perdita; a questo punto la velocità di trasmissione viene decrementata. Il mittente TCP aumenta

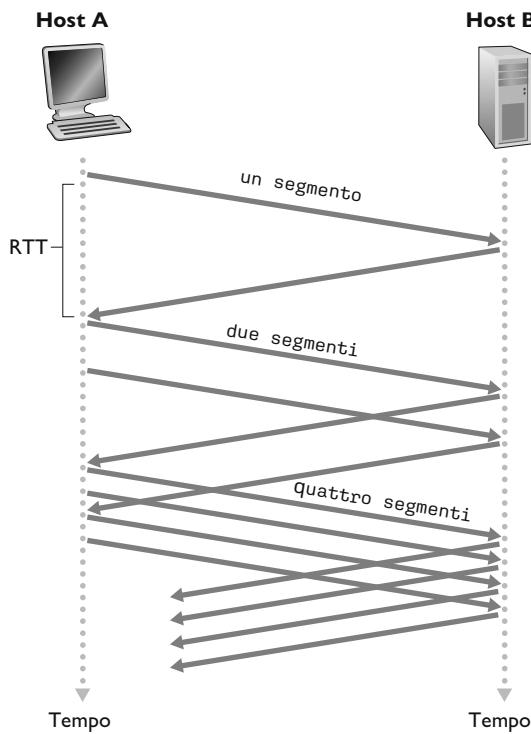
quindi il suo tasso di trasmissione per rilevare quando cominci a verificarsi la congestione, rallenta e quindi inizia di nuovo la fase di rilevamento per trovare se il punto di inizio della congestione è cambiato. Il comportamento del mittente TCP è analogo a quello di un bambino che chiede sempre più giocattoli finché i genitori infine gli dicono di no, smette per un po' e subito dopo ricomincia. Si noti che non c'è un segnale esplicito di congestione da parte della rete; gli acknowledgment e gli eventi di perdita hanno la funzione di segnali impliciti, e quindi ogni mittente TCP agisce sulla base di informazioni locali in modo asincrono rispetto agli altri.

Possiamo ora prendere in considerazione i dettagli del celebrato **algoritmo di controllo di congestione di TCP** (*TCP congestion-control algorithm*), descritto per la prima volta in [Jacobson 1988] e standardizzato in [RFC 5681]. L'algoritmo presenta tre componenti o fasi principali: (1) *slow start*, (2) *congestion avoidance* e (3) *fast recovery*. Slow start e congestion avoidance sono componenti obbligatorie di TCP e differiscono nel modo in cui aumentano la grandezza di *cwnd* in risposta agli acknowledgment ricevuti. Vedremo tra breve che slow start incrementa la dimensione della *cwnd* molto più rapidamente (nonostante il nome, che si può tradurre con “partenza lenta”!) di congestion avoidance. La fast recovery (*recupero veloce*) è suggerita, ma non obbligatoria, per i mittenti TCP.

### **Slow start**

Quando si stabilisce una connessione TCP, il valore di *cwnd* viene in genere inizializzato a 1 MSS [RFC 3390], il che comporta una velocità di invio iniziale di circa  $MSS/RTT$ . Per esempio, se  $MSS = 500$  byte e  $RTT = 200$  ms, la velocità iniziale è solo di circa 20 kbps. Dato che la banda disponibile alla connessione può essere molto più grande di  $MSS/RTT$ , il mittente TCP gradirebbe scoprire velocemente la banda disponibile. Quindi, durante la fase iniziale, detta **slow start**, il valore di *cwnd* parte da 1 MSS e si incrementa di 1 MSS ogni volta che un segmento trasmesso riceve un acknowledgment. Nello specifico (Figura 3.50) TCP invia il primo segmento nella rete e attende un acknowledgment. Se il segmento riceve un acknowledgment prima che si verifichi un evento di perdita, il mittente incrementa la finestra di congestione di 1 MSS e invia due segmenti di dimensione massima. Questi segmenti ricevono a loro volta degli acknowledgment e il mittente incrementa la finestra di congestione di 1 MSS per ciascuno di essi portandola a 4 MSS e così via. Questo processo ha come effetto il radoppio della velocità di trasmissione a ogni  $RTT$ . Quindi, in TCP, la velocità di trasmissione parte lentamente, ma cresce in modo esponenziale durante la fase di slow start.

Quando tuttavia dovrebbe terminare questa crescita esponenziale? Slow start fornisce alcune risposte a questa domanda. Innanzitutto, se c'è un evento di perdita (e quindi una congestione) indicato da un evento di timeout, il mittente TCP pone il valore di *cwnd* pari a 1 e inizia di nuovo il processo di slow start. Inoltre pone il valore di una seconda variabile di stato, *ssthresh* (forma contratta per “slow start threshold” o “soglia di slow start”) a  $cwnd/2$ : metà del valore che aveva la finestra di congestione quando la congestione è stata rilevata. Il secondo modo in cui la fase di slow start può terminare è legato direttamente al numero di segmenti inviati.

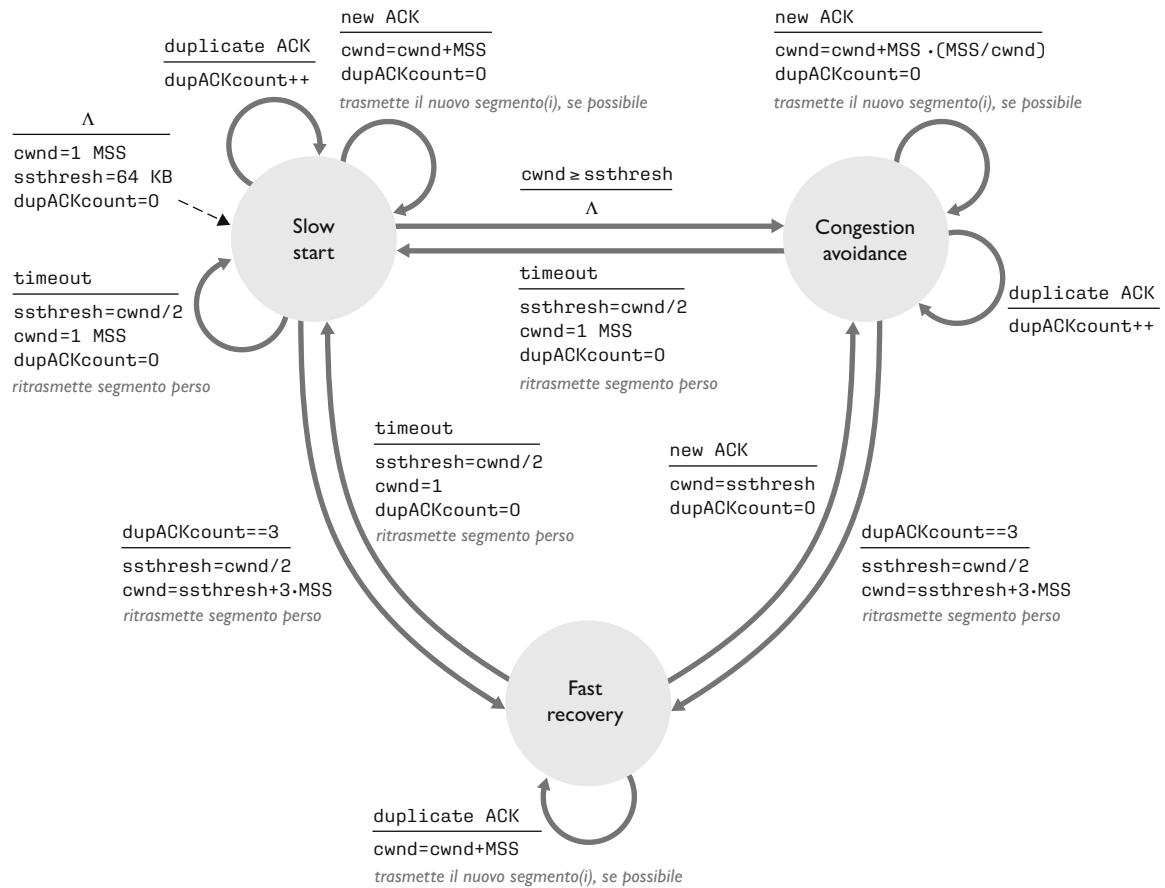


**Figura 3.50** Slow start di TCP.

tamente al valore di `ssthresh`. Poiché `ssthresh` è impostato a metà del valore di `cwnd` all'ultimo rilievo di congestione, potrebbe essere un po' temerario continuare a raddoppiare il valore di `cwnd` quando raggiunge o sorpassa il valore di `ssthresh`. Quindi, quando il valore di `cwnd` è pari a `ssthresh`, la fase di slow start termina e TCP entra in modalità di congestion avoidance. Come vedremo, TCP incrementa `cwnd` più cautamente quando agisce in modalità congestion avoidance. L'ultimo modo in cui la fase di slow start può terminare è quando vengono rilevati tre acknowledgment duplicati, nel qual caso TCP opera una ritrasmissione rapida (si veda il Paragrafo 3.5.4) ed entra nello stato di fast recovery, come discusso di seguito. Il comportamento di TCP nella fase di slow start è riassunto nell'automa a stati finiti rappresentato nella Figura 3.51. L'algoritmo di slow start affonda le sue radici in [Jacobson 1988] e un approccio simile a slow start venne anche proposto in maniera indipendente in [Jain 1986].

### Congestion avoidance

Quando TCP entra nello stato di congestion avoidance (*prevenzione della congestione*), il valore di `cwnd` è circa la metà di quello che aveva l'ultima volta in cui era stata rilevata la congestione (che potrebbe essere ancora dietro l'angolo). Quindi, invece di raddoppiare il valore di `cwnd` ogni *RTT*, TCP adotta un approccio più conservativo, incrementando `cwnd` di 1 MSS ogni *RTT* [RFC 5681]. Ciò si può ottenere in diversi modi: un approccio comune è l'incremento da parte del mittente TCP della propria `cwnd` di  $\text{MSS} \times (\text{MSS}/\text{cwnd})$  byte ogni qualvolta riceva un nuovo acknowledgment. Per esempio, se MSS vale 1460 byte e `cwnd` 14.600 byte, allora in un *RTT* vengono spediti dieci segmenti. Ciascun



**Figura 3.51** Descrizione tramite automa a stati finiti del controllo di congestione TCP.

ACK in arrivo (assumendo un ACK per segmento) incrementa l'ampiezza della finestra di congestione di 1/10 MSS e quindi, dopo la ricezione degli acknowledgment relativi a tutti e dieci i segmenti, il valore della finestra di congestione sarà stato aumentato di un MSS.

Ma quando finisce l'incremento lineare (1 MSS per RTT) durante la congestion avoidance? L'algoritmo di congestion avoidance, quando si verifica un timeout, si comporta nello stesso modo di slow start: il valore di cwnd è posto uguale a 1 MSS e il valore di ssthresh viene impostato alla metà del valore di cwnd al momento del timeout. Si ricordi, tuttavia, che un evento di perdita può essere anche il risultato della ricezione di tre acknowledgment duplicati; in tal caso però la rete continua a consegnare segmenti dal mittente al ricevente, per cui la risposta di TCP a questo tipo di evento dovrebbe essere meno drastica di quella adottata nel caso di timeout. In caso di acknowledgment duplicati TCP dimezza il valore di cwnd (aggiungendo 3 MSS per tenere conto dei duplicati ricevuti) e imposta il valore di ssthresh pari a metà del valore di cwnd al momento del ricevimento dei tre ACK duplicati. Infine, TCP entra nello stato di fast recovery.

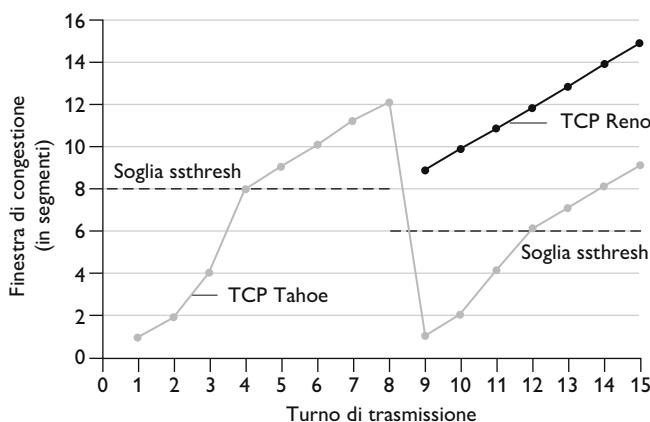
### Fast recovery

Durante la fase di fast recovery (*recupero veloce*) il valore di cwnd è incrementato di 1 MSS per ogni ACK duplicato ricevuto relativamente al segmento perso che ha causato l'entrata di TCP nello stato di fast recovery. Infine, quando arriva un ACK per il segmento perso, TCP entra nello stato di congestion avoidance dopo aver ridotto il valore di cwnd. Se si verifica un timeout vi è invece una transizione dallo stato di fast recovery a quello di slow start dopo avere effettuato le stesse azioni presenti sia in slow start che in congestion avoidance: il valore di cwnd è posto a 1 MSS e il valore di ssthresh è impostato a metà del valore di cwnd nel momento in cui si è riscontrato l'evento di perdita.

Fast recovery è un componente raccomandato, ma non obbligatorio di TCP [RFC 5681]. È interessante sapere che una prima versione di TCP, nota come **TCP Tahoe**, portava in modo incondizionato la finestra di congestione a 1 MSS ed entrava nella fase di slow start dopo qualsiasi tipo di evento di perdita. La versione più recente, **TCP Reno**, adotta invece fast recovery.

Nella Figura 3.52, che mostra l'evoluzione della finestra di congestione TCP con Reno e con Tahoe, il valore iniziale della soglia ssthresh è di 8 MSS. Per i primi 8 cicli di trasmissione Tahoe e Reno si comportano allo stesso modo. La finestra di congestione cresce in modo esponenziale e supera la soglia nel quarto turno di trasmissione. La finestra di congestione quindi sale in modo lineare fino al verificarsi di un triplice ACK duplicato, appena dopo l'ottavo turno di trasmissione. Notiamo che la finestra di congestione vale 12 MSS quando si verifica l'evento di perdita. La soglia ssthresh viene quindi impostata a  $0,5 \cdot \text{cwnd} = 6$  MSS. Con TCP Reno, la finestra di congestione è posta a 6 MSS e poi cresce in modo lineare. Con TCP Tahoe, la finestra di congestione viene posta a 1 MSS e cresce in modo esponenziale fino a raggiungere la soglia ssthresh, punto dal quale cresce linearmente.

La Figura 3.51 mostra la descrizione completa degli algoritmi di congestione (slow start, congestion avoidance e fast recovery) tramite un FSM. La figura indica anche in quali punti può avvenire la trasmissione di nuovi segmenti o la ritrasmissione di segmenti già inviati. Sebbene sia importante distinguere in TCP



**Figura 3.52** Evoluzione della finestra di congestione TCP (Tahoe e Reno).

**BOX 3.5****TEORIA E PRATICA**

### **TCP splitting: ottimizzazione delle prestazioni dei servizi di cloud computing**

Per servizi di cloud computing quali motori di ricerca, e-mail e social network è preferibile fornire un alto livello in interazione all'utente, dandogli idealmente l'impressione che i servizi siano in esecuzione sul proprio sistema locale (smartphone inclusi). Questo può rappresentare un grave problema, in quanto gli utenti sono spesso molto lontani dal data center responsabile di fornire i contenuti dinamici associati al servizio cloud. Indubbiamente, se l'utente è molto lontano dal data center, l'RTT sarà molto grande e porterà potenzialmente a delle pessime prestazioni a causa dello slow start di TCP.

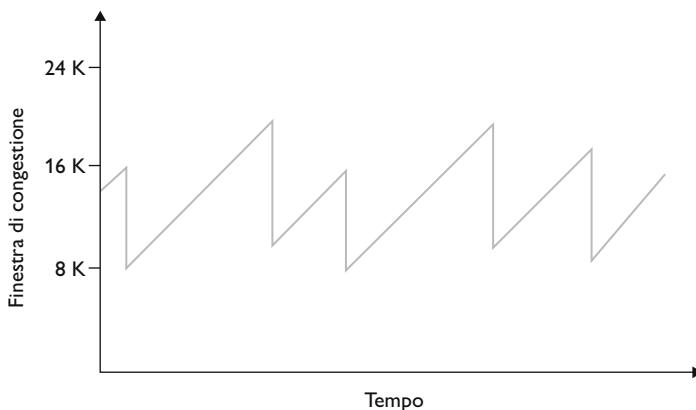
Come caso di studio si consideri il ritardo nella ricezione della risposta da un motore di ricerca. Tipicamente, per inviare i dati, il server ha bisogno di allocare finestre TCP durante lo slow start [Pathak 2010]. Quindi, l'intervallo di tempo che trascorre da quando il sistema inizializza una connessione TCP a quando l'ultimo pacchetto della risposta viene ricevuto è di circa 4·RTT (un RTT per inizializzare la connessione e 3 RTT per l'invio delle tre finestre con i dati) più il tempo di elaborazione da parte del data center. Questi ritardi dovuti allo RTT possono causare un ritardo non trascurabile nell'ottenere la risposta di un numero significativo di richieste. Inoltre, è possibile ci sia una perdita di pacchetti considerevole nelle reti di accesso, cosa che può comportare ritrasmissioni e ritardi ancora maggiori.

Un modo per mitigare questo problema e migliorare le prestazioni percepite dall'utente è di (1) installare dei server vicini all'utente (server di front-end) e (2) fare uso di **TCP splitting** interrompendo la connessione TCP in corrispondenza di questi server ravvicinati. Con tale tecnica il client stabilisce una connessione TCP con il server di front-end più vicino mentre quest'ultimo mantiene con il data center una connessione TCP permanente con una finestra di congestione molto grande [Tariq 2008, Pathak 2010, Chen 2011]. In questo modo il tempo di risposta diventa circa  $4 \cdot RTT_{FE} + RTT_{BE} + \text{tempo di elaborazione}$ . Dove  $RTT_{FE}$  è l'RTT tra client e server di front-end e  $RTT_{BE}$  è l'RTT tra il server di front-end e il data center (detto anche server di back-end). Se il server di front-end è davvero vicino all'utente, allora questo tempo diventa circa  $RTT + \text{tempo di elaborazione}$  in quanto  $RTT_{FE}$  è abbastanza piccolo da poter essere ignorato e  $RTT_{BE}$  è circa uguale a RTT. In totale, la tecnica di TCP splitting può ridurre il ritardo di rete approssimativamente da 4·RTT a RTT, migliorando in maniera significativa le prestazioni percepite dagli utenti, in particolar modo quelli lontani dal data center. Il TCP splitting aiuta inoltre a ridurre i ritardi di ritrasmissione causati dalle perdite di pacchetti nelle reti di accesso. Oggi, Google e Akamai fanno largo uso di TCP splitting nei loro server CDN (Paragrafo 2.6) a supporto dei loro servizi di cloud computing [Chen 2011].

tra il controllo degli errori/ritrasmissioni e il controllo di congestione, è anche importante capire come questi due aspetti siano inestricabilmente legati.

### **Retrospettiva sul controllo di congestione di TCP**

Dopo aver visto i dettagli delle fasi di slow start, congestion avoidance e fast recovery, vale la pena fare un passo indietro per averne una visione complessiva. Ignorando la parte iniziale della fase di slow start, quando inizia la connessione, e assumendo che le perdite siano indicate da un triplo ACK duplicato piuttosto che da eventi di timeout, il controllo di congestione di TCP consiste in un in-



**Figura 3.53** Controllo di congestione a incremento additivo e decremento moltiplicativo.

cremento additivo lineare della cwnd pari a 1 MSS per  $RTT$  e quindi di un decremento moltiplicativo che dimezza la cwnd in corrispondenza di un evento di triplice ACK duplicato. Per questa ragione, il controllo di congestione di TCP è spesso indicato come una forma di controllo di congestione **incremento additivo, decremento moltiplicativo** (AIMD, *Additive-Increase Multiplicative-Decrease*). Il controllo di congestione AIMD dà luogo al comportamento a dente di sega (“saw tooth”) mostrato nella Figura 3.53, che illustra chiaramente la nostra precedente intuizione sulla rilevazione della larghezza di banda: TCP incrementa linearmente l’ampiezza della propria finestra di congestione e quindi della velocità di trasmissione, finché si verifica un evento di triplice ACK duplicato. Quindi decrementa la propria finestra di congestione di un fattore due, ma riprende ancora a crescere linearmente per capire se ci sia ulteriore ampiezza di banda disponibile.

L’algoritmo AIMD di TCP fu sviluppato basandosi su una grande esperienza tecnica sul campo e con sperimentazioni sul controllo di congestione di reti effettivamente utilizzate. Dieci anni dopo lo sviluppo di TCP analisi teoriche hanno dimostrato che l’algoritmo di controllo di congestione di TCP svolge la funzione di un algoritmo distribuito di ottimizzazione asincrona che ottimizza contemporaneamente molteplici indici di prestazione per la rete e l’utente [Kelly 1998]. Da allora, si è sviluppata una ricca teoria sul controllo di congestione [Srikant 2012].

### TCP CUBIC

Dato l’approccio al controllo di congestione di TCP Reno con aumento additivo e decremento moltiplicativo, viene spontaneo chiedersi se sia questo il modo migliore per “sondare” una velocità di trasmissione dei pacchetti appena inferiore alla soglia di attivazione della perdita di pacchetti. Infatti una strategia che dimezza la velocità di invio (*sending rate*) (o peggio ancora, la riduce a un pacchetto per  $RTT$  come in una versione precedente di TCP nota come TCP Tahoe), e quindi la aumenta piuttosto lentamente, può essere eccessivamente cauta. Se lo stato del collegamento congestionato dove si è verificata la perdita di pacchetti non è cambiato molto, forse è meglio aumentare la velocità di invio rapidamente per avvicinarsi a quella pre-perdita dei pacchetti e quindi sondare

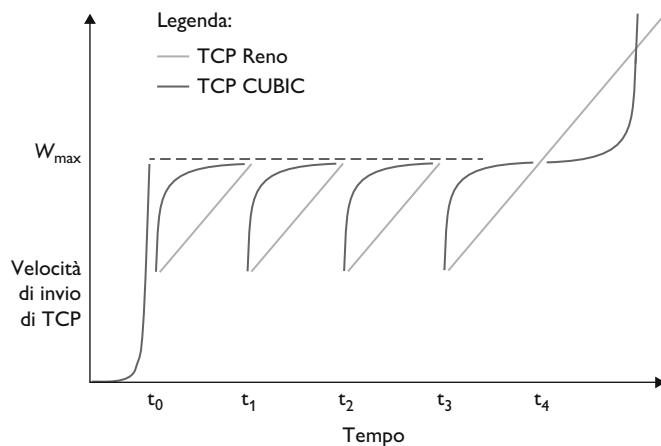
con cautela la larghezza di banda. Questa intuizione è al centro di una versione di TCP nota come TCP CUBIC [Ha 2008, RFC 8312].

TCP CUBIC differisce solo leggermente da TCP Reno. Ancora una volta, la finestra di congestione viene aumentata solo alla ricezione di un ACK e nelle fasi di slow start e fast recovery rimane la stessa. CUBIC modifica solo la fase di congestion avoidance nel modo descritto di seguito:

- Sia  $W_{\max}$  la dimensione della finestra del controllo di congestione di TCP nell'istante in cui viene rilevata l'ultima perdita e sia  $K$  l'istante nel futuro in cui la dimensione della finestra di TCP CUBIC raggiungerà nuovamente  $W_{\max}$ , assumendo che non vi sia alcuna perdita. Diversi parametri CUBIC determinano il valore di  $K$  che indica quanto velocemente la dimensione della finestra di congestione del protocollo raggiungerebbe  $W_{\max}$ .
- CUBIC aumenta la finestra di congestione in funzione del cubo della distanza tra l'istante corrente  $t$  e  $K$ . Pertanto, quando  $t$  è distante da  $K$ , gli incrementi della dimensione della finestra di congestione sono molto maggiori rispetto a quando  $t$  è vicino a  $K$ . Quindi CUBIC aumenta rapidamente la velocità di invio di TCP per avvicinarsi alla velocità di pre-perdita  $W_{\max}$  e solo quando si avvicina a  $W_{\max}$  esamina con cautela la larghezza di banda.
- La regola cubica implica che gli aumenti della finestra di congestione di CUBIC siano piccoli quando  $t$  è ancora vicino a  $K$  (il che è positivo se il livello di congestione del collegamento che causa la perdita non è cambiato molto), ma poi aumentino rapidamente appena  $t$  supera  $K$ , il che consente a CUBIC di trovare più rapidamente un nuovo punto di lavoro qualora il livello di congestione del collegamento che ha causato la perdita sia cambiato in modo significativo.

In base a queste regole, le prestazioni idealizzate di TCP Reno e TCP CUBIC sono paragonate nella Figura 3.54 adattata da [Huston 2017]. Vediamo la fase di slow start che termina a  $t_0$ . Quindi, quando a  $t_1$ ,  $t_2$  e  $t_3$  si verifica una perdita per congestione (*congestion loss*), CUBIC si avvicina a  $W_{\max}$  più rapidamente, godendo così di un throughput complessivo maggiore rispetto a TCP Reno. Possiamo vedere graficamente come TCP CUBIC tenti di mantenere per tutto

**Figura 3.54** Velocità di invio della congestione avoidance di TCP: TCP Reno e TCP Cubic.



il tempo possibile il flusso appena al di sotto della soglia di congestione, sconosciuta al mittente. Si noti che a  $t_3$  il livello di congestione è presumibilmente diminuito sensibilmente, consentendo sia TCP Reno che a TCP CUBIC di ottenere velocità di invio superiori a  $W_{\max}$ .

TCP CUBIC è attualmente ampiamente utilizzato. Mentre misurazioni effettuate intorno all'anno 2000 su web server popolari hanno mostrato che quasi tutti avevano in esecuzione qualche versione di TCP Reno [Padhye 2001], misurazioni più recenti sui 5000 grandi web server mostrano che quasi il 50% esegue una versione di TCP CUBIC [Yang 2014], che è anche la versione predefinita di TCP utilizzata nel sistema operativo Linux.

### Descrizione macroscopica del throughput di TCP

Assodato il comportamento a dente di sega di TCP Reno, è naturale chiedersi quale potrebbe essere il throughput medio di una connessione TCP Reno prolungata. In quest'analisi ignoreremo le fasi di slow start che si verificano dopo gli eventi di timeout (queste sono generalmente molto brevi, dato che il mittente aumenta esponenzialmente le sue trasmissioni). Durante un dato intervallo di  $RTT$ , la velocità di invio dei dati è funzione della finestra di congestione e dell' $RTT$  corrente. Se l'ampiezza della finestra vale  $w$  byte, la velocità di trasmissione è approssimativamente  $w/RTT$ . TCP va alla ricerca di banda aggiuntiva aumentando  $w$  di 1 MSS a ogni  $RTT$ , fino al verificarsi di un evento di perdita. Detto  $W$  il valore di  $w$  quando si verifica tale evento e assumendo che  $RTT$  e  $W$  siano approssimativamente costanti per la durata della connessione, la velocità di trasmissione TCP varia tra  $W/(2 \cdot RTT)$  e  $W/RTT$ .

Queste ipotesi forniscono un modello macroscopico estremamente semplificato del comportamento a regime di TCP. La rete elimina un pacchetto dalla connessione quando la velocità sale a  $W/RTT$ ; la velocità viene poi dimezzata e quindi incrementata di  $MSS/RTT$  a ogni  $RTT$ , fino a quando raggiunge ancora  $W/RTT$ . Questo processo si ripete in continuazione. Dato che il throughput (cioè la velocità) cresce in modo lineare tra i due valori estremi, abbiamo:

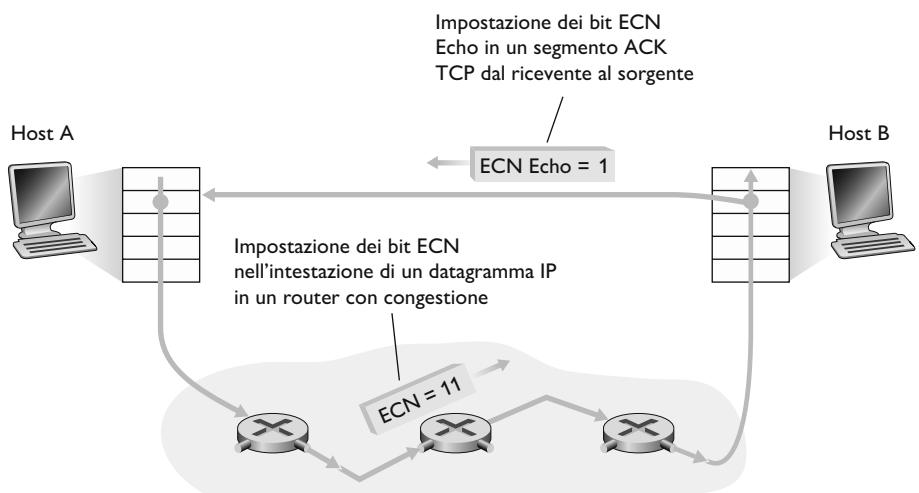
$$\text{throughput medio di una connessione} = \frac{0,75 \cdot W}{RTT}$$

Utilizzando questo modello fortemente idealizzato per la dinamica a regime di TCP, possiamo anche derivare un'interessante espressione che metta in relazione la frequenza di perdite di una connessione con la sua banda disponibile [Mathis 1997]. Questa conseguenza viene evidenziata nei Problemi alla fine del capitolo. Un modello più sofisticato, determinato in modo empirico sulla base di misurazioni, si trova in [Padhye 2000].

### 3.7.2 Notifica esplicita di congestione assistita dalla rete e controllo di congestione basato sul ritardo

Sin dalla prima standardizzazione dei meccanismi di slow start e congestion avoidance alla fine degli anni '80 [RFC 1122] il protocollo TCP ha implementato la forma end-to-end del controllo di congestione che abbiamo studiato nel Paragrafo 3.7.1: un mittente TCP non riceve alcuna notifica esplicita di congestione dal livello di rete, ma ne deduce l'esistenza osservando la perdita di pac-

**Figura 3.55** Notifica esplicita di congestione: controllo di congestione assistito dalla rete.



chetti. Più di recente sono state proposte, implementate e distribuite estensioni per IP e TCP [RFC 3168] che permettono alla rete di segnalare esplicitamente a mittente e ricevente una congestione. Inoltre sono state proposte una serie di varianti del protocollo di controllo di congestione TCP che deducono la congestione utilizzando misure di ritardo dei pacchetti. In questo paragrafo tratteremo il controllo di congestione sia assistito dalla rete sia basato sul ritardo.

### Notifica esplicita di congestione

L'Explicit Congestion Notification (ECN) [RFC 3168] è la forma di controllo di congestione assistita dalla rete effettuata in Internet. Come mostrato nella Figura 3.55 sono coinvolti sia TCP sia IP. A livello di rete vengono utilizzati due bit (quindi quattro possibili valori) nel campo Tipo di Servizio dell'intestazione IP. Se un router è congestionato, imposta tali bit e invia il pacchetto IP contrassegnato al destinatario, che quindi informa il mittente, come mostrato nella Figura 3.55. L'RFC 3168 non definisce quando un router è congestionato: tale decisione rappresenta una scelta di configurazione di pertinenza dell'operatore di rete, tuttavia raccomanda di segnalare una congestione solo in caso di congestione persistente. I bit ECN vengono inoltre utilizzati dal mittente per segnalare che mittente e ricevente sono abilitati all'uso di ECN.

Come mostrato nella Figura 3.55, quando il destinatario TCP riceve un'indicazione di congestione ECN, ne informa il mittente TCP impostando il bit ECE (*Explicit Congestion Notification Echo*) (si veda la Figura 3.29) all'interno di un segmento ACK. Il mittente TCP reagisce dimezzando la finestra di congestione, esattamente come farebbe in caso di perdita di un segmento usando il meccanismo di ritrasmissione rapida, e imposta il bit CWR (*Congestion Window Reduced*) nell'intestazione del successivo segmento che invia al ricevente.

Altri protocolli a livello di trasporto usano il protocollo ECN, *Datagram Congestion Control Protocol* (DCCP) [RFC 4340] che fornisce un controllo di congestione ECN leggero, ma non affidabile. Mentre il protocollo DCTCP

(*Data Center TCP*) [Alizadeh 2010, RFC 8257] è progettato specificatamente per infrastrutture di rete all'interno di data center.

Anche DCQCN (*Data Center Quantized Congestion Notification*) [Zhu 2015], progettato specificamente per le reti dei data center, si avvale dell'ECN. Recenti misurazioni di Internet mostrano un aumento dell'implementazione delle funzionalità ECN sia nei server più popolari come nei router lungo i percorsi verso tali server [Kuhlewind 2013].

### **Controllo di congestione basato sul ritardo**

Come visto nella precedente discussione su ECN, un router congestionato può impostare il bit di indicazione di congestione per segnalare l'inizio della congestione ai mittenti *prima* che i buffer si riempiano e causino la perdita dei pacchetti al router. Ciò consente ai mittenti di ridurre le velocità di invio in anticipo, auspicabilmente *prima* della perdita di pacchetti, evitando così costose perdite di pacchetti e ritrasmissioni. Un secondo approccio per evitare la congestione adotta un approccio basato sul ritardo (*delay-based*) che rileva l'insorgenza della congestione in modo proattivo, *prima* che si verifichi la perdita di pacchetti.

In TCP Vegas [Brakmo 1995] il mittente misura l'RTT del percorso dalla sorgente alla destinazione per tutti i pacchetti riscontrati. Sia  $RTT_{min}$  il valore minimo di tali misurazioni presso un mittente; ciò si verifica quando il percorso non è congestionato e i pacchetti sperimentano un ritardo di coda minimo. Se la dimensione della finestra di congestione di TCP Vegas è  $cwnd$ , il tasso di throughput non congestionato è  $cwnd/RTT_{min}$ . L'intuizione dietro TCP Vegas è che se il throughput effettivo misurato dal mittente è vicino a questo valore, la velocità di invio TCP può essere aumentata finché (per definizione e misurazione) il percorso rimanga non congestionato. Tuttavia, se la velocità effettiva misurata dal mittente è significativamente inferiore al tasso di throughput in assenza di congestione, il percorso è congestionato e il mittente TCP Las Vegas diminuisce la velocità di invio [Brakmo 1995].

TCP Vegas opera con l'intuizione che i mittenti TCP dovrebbero “keep the pipe just full, but no fuller” [Kleinrock 2018]. “Keeping the pipe full” significa che i collegamenti e, in particolare, il collegamento a collo di bottiglia che limita il throughput di una connessione, vengono mantenuti occupati a trasmettere, svolgendo un lavoro utile; “but no fuller” significa che non c’è niente da guadagnarci, – tranne un aumento del ritardo! – se si consente che si accumulino code di grandi dimensioni.

Il protocollo di controllo di congestione BBR (*Bottleneck Bandwidth and Roundtrip propagation time*) [Cardwell 2017] si basa sulle idee di TCP Vegas e incorpora meccanismi che gli consentono di competere in modo equo (si veda il Paragrafo 3.7.3) con mittenti TCP non BBR. [Cardwell 2017] riferisce che nel 2016 Google iniziò a utilizzare BBR invece di CUBIC per tutto il traffico TCP sulla sua rete privata B4 [Jain 2013] che interconnette i data center di Google. BBR viene utilizzato anche sui web server di Google e YouTube. Altri protocolli di controllo di congestione TCP basati sul ritardo includono TIMELY per le reti dei data center [Mittal 2015], Compound TCP (CTPC) [Tan 2006] e FAST [Wei 2006] per reti ad alta velocità e lunga distanza.

### 3.7.3 Fairness

Consideriamo  $K$  connessioni TCP, ciascuna con un differente percorso end-to-end, ma che passano tutte attraverso un collegamento con capacità trasmissiva di  $R$  bps che costituisce il collo di bottiglia del sistema; con ciò intendiamo che tutti gli altri collegamenti non siano congestionati e dispongano di elevata capacità trasmissiva, in confronto a quella del collegamento che fa da collo di bottiglia. Supponiamo che ogni connessione stia trasferendo un file di grandi dimensioni e che non ci sia traffico UDP attraverso il collo di bottiglia. Si dice che un meccanismo di controllo di congestione sia “fair” (*equo*) se la velocità di trasmissione media di ciascuna connessione è approssimativamente  $R/K$ ; in altre parole, ciascuna connessione ottiene la stessa porzione di banda del collegamento.

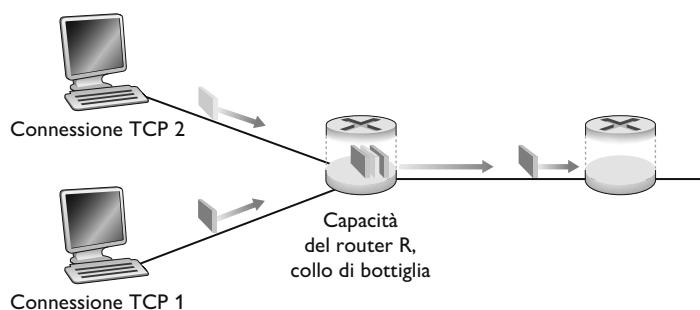
Si può dire che AIMD sia un algoritmo fair? La risposta deve tener conto del fatto che le connessioni TCP possono aver inizio in istanti diversi e quindi possono avere diverse dimensioni di finestra. [Chiu 1989] presenta una spiegazione elegante e intuitiva del perché il controllo di congestione di TCP tenda a offrire la stessa porzione di banda a connessioni TCP in competizione in un collegamento che costituisce un collo di bottiglia.

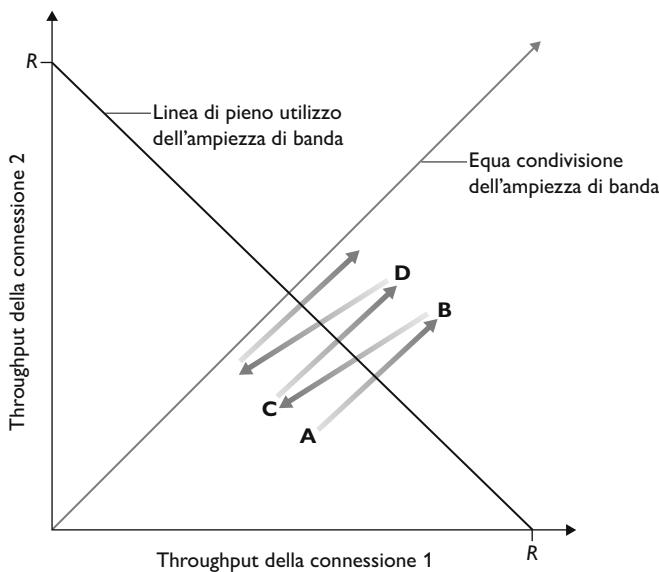
Consideriamo il caso semplice di due connessioni TCP che condividono un collegamento con capacità trasmissiva  $R$  (Figura 3.55). Assumiamo che le connessioni abbiano gli stessi valori di MSS e RTT (e pertanto abbiano uguali finestre di congestione e lo stesso throughput), che debbano trasmettere una gran quantità di dati e che non vi siano altre connessioni TCP o datagrammi UDP che attraversano questo collegamento condiviso. Inoltre, ignoriamo la fase di slow start e assumiamo che le connessioni operino in modalità congestion avoidance (AIMD) per tutto il tempo.

La Figura 3.56 traccia il throughput delle due connessioni. Se TCP sta suddividendo la larghezza di banda del collegamento in modo uguale tra le due connessioni, il throughput dovrebbe cadere sulla bisettrice del primo quadrante. Idealmente, la somma dei due throughput dovrebbe esser uguale a  $R$ . Certamente, non sarebbe piacevole che ciascuna connessione ricevesse la stessa porzione, nulla, della capacità del collegamento. Quindi, l’obiettivo dovrebbe essere ottenere throughput situati vicino all’intersezione tra la bisettrice e la linea di massimo utilizzo della banda (Figura 3.56).

Supponiamo che le dimensioni della finestra TCP siano tali che a un certo istante di tempo le connessioni 1 e 2 raggiungano i throughput corrispondenti al punto A della Figura 3.57. Dato che la porzione di banda del collegamento

**Figura 3.56** Due connessioni TCP che condividono un singolo collegamento che fa da collo di bottiglia.





**Figura 3.57**  
Throughput delle connessioni TCP 1 e 2.

utilizzata congiuntamente è minore di  $R$ , non si verificheranno perdite e le due connessioni aumenteranno la loro finestra di 1 MSS per  $RTT$  come risultato dell'algoritmo di congestion avoidance. Di conseguenza, il throughput congiunto procede lungo la semiretta a  $45^\circ$  (pari incremento per entrambe le connessioni) uscente dal punto  $A$ . La banda congiunta potrebbe essere maggiore di  $R$ , e si potrebbe quindi verificare una perdita di pacchetti. Se le connessioni 1 e 2 subiscono una perdita di pacchetti quando raggiungono i throughput indicati dal punto  $B$ , allora decrementeranno le loro finestre di un fattore 2. I throughput raggiunti di conseguenza si troveranno pertanto sul punto  $C$ , a metà strada lungo un segmento che collega il punto  $B$  all'origine. Essendo l'utilizzo di banda condivisa minore di  $R$  in prossimità del punto  $C$ , le due connessioni ancora una volta incrementano il loro throughput lungo la linea a  $45^\circ$  passante per  $C$ . Al punto  $D$  si possono ancora verificare delle perdite, nel qual caso le due connessioni decrementeranno di nuovo l'ampiezza delle loro finestre di un fattore 2, e così via. Dovreste esservi convinti che la banda utilizzata dalle due connessioni può fluttuare lungo la linea di equa condivisione della banda e che le due connessioni convergeranno a questo comportamento indipendentemente dal punto del piano in cui si trovano inizialmente. Sebbene basato su numerose ipotesi, tale scenario fornisce un'idea intuitiva del perché TCP porti a un'equa ripartizione della banda tra le connessioni.

Nel nostro scenario abbiamo ipotizzato che solo connessioni TCP attraversino il collo di bottiglia, che abbiano lo stesso  $RTT$  e che non ci siano connessioni multiple tra coppie di host. Nella pratica, queste condizioni non si verificano quasi mai e quindi le applicazioni client/server ottengono porzioni di banda assai diverse. In particolare, è stato dimostrato che quando più connessioni condividono un collo di bottiglia, quelle con  $RTT$  inferiore sono in grado di acquisire più rapidamente larghezza di banda su un particolare collegamento, non appena la banda si libera. In altre parole, aprono le proprie finestre di con-

gestione più rapidamente e quindi avranno throughput superiori rispetto alle connessioni con  $RTT$  più alto [Lakshman 1997].

### Fairness e UDP

Abbiamo appena visto come il meccanismo della finestra di congestione consenta al controllo di congestione TCP di regolare il tasso di trasmissione (*transmission rate*) delle applicazioni. Questo è il motivo per cui molte applicazioni multimediali, quali la fonia e la videoconferenza, non fanno uso di TCP: non vogliono che il loro tasso di trasmissione venga ridotto, anche se la rete è molto congestionata. Piuttosto, queste applicazioni preferiscono utilizzare UDP, che non incorpora il controllo di congestione, in modo da poter immettere il proprio audio e video sulla rete a frequenza costante e occasionalmente perdere pacchetti, piuttosto che non perderli, ma dover ridurre il loro tasso di trasmissione a livelli “equi” nei momenti di traffico. Dal punto di vista di TCP, le applicazioni multimediali che fanno uso di UDP non sono fair: non cooperano con altre né adeguano il loro tasso di trasmissione in modo appropriato. Dato che il controllo di congestione di TCP diminuisce il proprio tasso di trasmissione come risposta alla congestione crescente (e alle perdite), mentre UDP no, le sorgenti UDP possono soffocare il traffico TCP. Sono stati proposti molti meccanismi di controllo di congestione capaci di evitare che il traffico UDP blocchi il throughput di Internet [Floyd 1999, Floyd 2000, Kohler 2006, RFC 4340].

### Fairness e connessioni TCP parallele

Anche se potessimo forzare il traffico UDP a comportarsi in modo equo, il problema della fairness non sarebbe tuttavia completamente risolto, perché nulla può impedire a un'applicazione basata su TCP di usare più connessioni in parallelo. Per esempio, spesso i browser web usano più connessioni TCP in parallelo per trasferire il contenuto delle pagine. Nella maggior parte dei browser è possibile configurare il numero esatto di connessioni multiple. Le applicazioni che utilizzano connessioni in parallelo ottengono una porzione di banda maggiore sui collegamenti congestionati. Consideriamo per esempio un collegamento di capacità  $R$  cui accedono nove applicazioni client/server, ciascuna delle quali utilizza una connessione TCP. Se giunge un'altra applicazione con una connessione TCP, allora la frequenza trasmissiva di tutte le applicazioni sarà circa uguale a  $R/10$ . Ma se, invece, la nuova applicazione usa 11 connessioni TCP in parallelo, allora otterrà un'allocazione iniqua superiore a  $R/2$ . Dato che il traffico web è assai diffuso in Internet, le connessioni in parallelo non sono rare.

## 3.8 Evoluzione della funzionalità del livello di trasporto

La nostra discussione dei protocolli di trasporto di Internet si è concentrata su UDP e TCP: i due “cavalli di battaglia” del livello di trasporto di Internet. Tuttavia, come abbiamo visto, tre decenni di esperienza con questi due protocolli hanno identificato le circostanze in cui nessuno dei due è idealmente adatto, e così la progettazione e l'implementazione della funzionalità del livello di trasporto hanno continuato a evolversi. Negli ultimi dieci anni abbiamo assistito a una notevole evoluzione dell'uso di TCP. Nei Paragrafi 3.7.1 e 3.7.2 abbiamo

appreso che oltre alle versioni “classiche” di TCP come TCP Tahoe e Reno, ne esistono alcune versioni più recenti sviluppate, implementate, distribuite e che sono attualmente molto utilizzate, tra cui TCP CUBIC, DCTCP, CTCP, BBR e altre ancora. In effetti, le misurazioni in [Yang 2014] indicano che CUBIC (e il suo predecessore, BIC [Xu 2004]) e CTCP siano più diffusi sui web server rispetto al classico TCP Reno; abbiamo visto anche che BBR è implementato nella rete B4 interna di Google, così come in molti server di Google rivolti al pubblico.

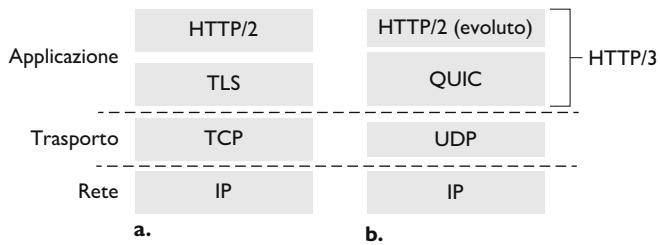
E ci sono molte (molte!) altre versioni di TCP. Esistono versioni di TCP progettate specificamente per l’uso su collegamenti wireless e su percorsi a larghezza di banda elevata con RTT di grandi dimensioni, su percorsi con riordino dei pacchetti e su percorsi brevi rigorosamente all’interno dei data center. Esistono versioni di TCP che implementano priorità diverse tra connessioni TCP in competizione per la larghezza di banda in un collegamento a collo di bottiglia e tra connessioni TCP i cui segmenti vengono inviati in parallelo su percorsi sorgente-destinazione diversi. Esistono anche varianti di TCP che trattano il riscontro dei pacchetti e l’apertura/chiusura delle sessioni in modo diverso da quello visto nel Paragrafo 3.5.6. In effetti, forse non è nemmeno corretto riferirsi a “il” protocollo TCP: le uniche caratteristiche comuni a questi protocolli consistono nell’utilizzare il formato del segmento TCP illustrato nella Figura 3.29 e nel dover competere “equamente” tra di loro in presenza di una congestione della rete! Per un approfondimento sulle molte versioni di TCP, si veda [Afanasyev 2010] e [Narayan 2018].

### **QUIC: Quick UDP Internet Connection**

Se i modelli di servizio offerti da UDP e TCP non sono adatti ai servizi del livello di trasporto richiesti da un’applicazione, gli sviluppatori dell’applicazione possono sempre usare un proprio protocollo a livello di applicazione. Tale situazione si può verificare perché un’applicazione necessita di più servizi di quelli forniti da UDP, ma non desidera tutte le specifiche funzionalità fornite da TCP o richiede servizi diversi da quelli forniti da TCP. Questo è l’approccio adottato nel protocollo **QUIC** (*Quick UDP Internet Connections*) [Langley 2017, QUIC 2020]. Nello specifico, QUIC è un nuovo protocollo a livello di applicazione, progettato da zero, per migliorare le prestazioni del livello di trasporto per HTTP sicuro. QUIC è già ampiamente in uso, sebbene sia ancora in fase di standardizzazione come Internet RFC [QUIC 2020]. Google ha implementato QUIC su molti dei suoi web server aperti al pubblico, nell’app YouTube, nel browser Chrome e nell’app Google Search su Android. Con oltre il 7% del traffico Internet attualmente su QUIC [Langley 2017] vorremmo dare uno sguardo più da vicino a questo protocollo. Lo studio di QUIC servirà anche come conclusione della trattazione sul livello di trasporto, poiché QUIC utilizza molti degli approcci per il trasferimento dati affidabile, il controllo di congestione e la gestione della connessione che abbiamo studiato in questo capitolo.

Come mostrato nella Figura 3.58, QUIC è un protocollo a livello di applicazione, che utilizza UDP come protocollo del livello di trasporto sottostante ed è progettato per interfacciarsi in modo specifico a una versione semplificata, ma evoluta, di HTTP/2. Prossimamente, HTTP/3 incorporerà nativamente QUIC [HTTP/3 2020].

**Figura 3.58** Stack di protocolli di (a) tradizionale HTTP sicuro e (b) QUIC sicuro basato su HTTP/3.

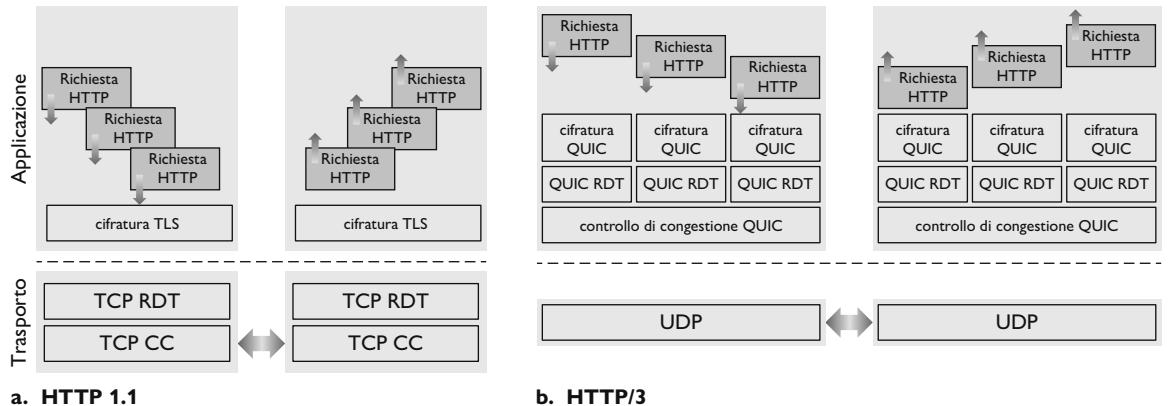


Alcune delle principali caratteristiche di QUIC sono le seguenti.

- **Orientato alla connessione e sicuro.** Come TCP, QUIC è un protocollo orientato alla connessione tra due endpoint. Ciò richiede una fase di handshaking tra gli endpoint per impostare lo stato della connessione QUIC. Due parti dello stato di connessione sono gli ID dell'origine e della destinazione della connessione. Tutti i pacchetti QUIC sono crittografati e, come suggerito nella Figura 3.58, QUIC combina l'handshaking necessario per stabilire la connessione con quelli necessari per l'autenticazione e la cifratura (*encryption*) che studieremo nel Capitolo 8, disponibile in inglese sulla piattaforma MyLab. In questo modo fornisce la connessione in modo più rapido rispetto allo stack di protocolli della Figura 3.58(a), dove sono necessari più RTT per stabilire prima una connessione TCP e poi una connessione TLS sopra la connessione TCP.
- **Flussi (*Streams*).** QUIC consente il multiplexing di diversi “flussi” a livello di applicazione attraverso una singola connessione QUIC; inoltre, una volta stabilita una connessione QUIC, nuovi flussi possono essere aggiunti rapidamente. Un flusso è un’astrazione per la consegna bidirezionale affidabile e in ordine dei dati tra due endpoint QUIC. Nel contesto di HTTP/3, per ogni oggetto in una pagina web ci sarebbe un flusso diverso. Ogni connessione ha un ID di connessione e ogni flusso all’interno di una connessione ha un ID di flusso; entrambi questi ID sono contenuti in un’intestazione del pacchetto QUIC (insieme ad altre informazioni di intestazione). I dati di più flussi possono essere contenuti all’interno di un singolo segmento QUIC, che viene trasferito su UDP.

Il protocollo Stream Control Transmission Protocol (SCTP) [RFC 4960, RFC 3286] è un precedente protocollo orientato al messaggio affidabile che ha aperto la strada al concetto di multiplexing di flussi multipli a livello di applicazione attraverso una singola connessione SCTP. Vedremo nel Capitolo 7, disponibile in inglese sulla piattaforma MyLab, che SCTP viene utilizzato nei protocolli del piano di controllo nelle reti wireless cellulari 4G/5G.

- **Trasferimento dati affidabile e con controllo di congestione.** Come illustrato nella Figura 3.59(b), QUIC fornisce un trasferimento dati affidabile a ogni flusso QUIC separatamente. La Figura 3.59(a) mostra il caso di HTTP/1.1 che invia più richieste HTTP, tutte su una singola connessione TCP. Dal momento che TCP fornisce un servizio di consegna dei byte affidabile e in ordine, le richieste HTTP multiple devono essere consegnate in ordine al server HTTP di destinazione. Pertanto, se i byte di una richiesta HTTP vengono persi, le restanti richieste HTTP non possono essere consegnate finché tali byte non vengono ritrasmessi e ricevuti correttamente da TCP sul server



**Figura 3.59** (a) singola connessione client-server che utilizza la cifratura TLS sul trasferimento dati affidabile (RDT) e il controllo di congestione TCP. (b) connessione client-server con flussi multipli che utilizza la cifratura, il trasferimento dati affidabile e il controllo di congestione di QUIC sopra il servizio non affidabile di UDP.

HTTP: il cosiddetto problema del blocco HOL che abbiamo incontrato già nel Paragrafo 2.2.5. Poiché QUIC fornisce una consegna in ordine e affidabile per flusso, un segmento UDP perso influisce solo sui flussi i cui dati sono stati trasportati in tale segmento; i messaggi HTTP di altri flussi possono continuare a essere ricevuti e consegnati all'applicazione. QUIC fornisce un trasferimento dati affidabile utilizzando meccanismi di acknowledgment simili a quelli di TCP, come specificato in [RFC 5681]. Il controllo di congestione di QUIC si basa su TCP NewReno [RFC 6582], una piccola variante del protocollo TCP Reno che abbiamo studiato nel Paragrafo 3.7.1. Nel Draft QUIC si legge: “I lettori che hanno familiarità con il rilevamento delle perdite e il controllo di congestione del protocollo TCP, troveranno qui algoritmi simili a quelli ben noti di TCP.”

Dal momento che abbiamo studiato attentamente il controllo di congestione di TCP nel Paragrafo 3.7.1, non avremo problemi a leggere a casa le specifiche dell'algoritmo del controllo di congestione QUIC! In conclusione, vale la pena sottolineare ancora una volta che QUIC è un protocollo a livello di applicazione che fornisce trasferimento dati tra due end-point affidabile e con controllo di congestione. Gli autori di QUIC [Langley 2017] sottolineano che questo significa che si possono apportare cambiamenti a QUIC con le tempistiche proprie dei cambiamenti sul livello di applicazione che sono molto più rapide di quelle per gli aggiornamenti di TCP o UDP.

## 3.9 Riepilogo

Abbiamo iniziato questo capitolo studiando i servizi che un protocollo a livello di trasporto può fornire alle applicazioni di rete. Da un lato, il protocollo a livello di trasporto può essere molto semplice e offrire alle applicazioni un servizio senza fronzoli, fornendo solo una funzione di multiplexing/demultiplexing per processi comunicanti. Il protocollo UDP di Internet è un esempio di questo genere di protocolli. Dall'altro, un protocollo può assicurare alle applicazioni

alcuni servizi quali la consegna affidabile dei dati e garanzie sui ritardi e sull'ampiezza di banda. Ciò nondimeno, i servizi che i protocolli di trasporto possono offrire sono spesso vincolati dal modello di servizio del sottostante protocollo a livello di rete. Se questo non può offrire ai segmenti garanzie sui ritardi o sulla banda, il protocollo a livello di trasporto non può a sua volta fornirle.

Nel Paragrafo 3.4 abbiamo appreso che un protocollo di trasporto può offrire trasferimento affidabile dei dati anche se il livello di rete è inaffidabile, che tale servizio presenta molti aspetti difficili da analizzare, ma che il compito può essere portato a termine combinando con attenzione acknowledgment, timer, ritrasmissioni e numeri di sequenza.

Sebbene abbiano trattato il trasferimento affidabile dei dati nel corso di questo capitolo, dovremmo tenere a mente che può essere offerto dai protocolli a livello di collegamento, di rete, di trasporto o di applicazione. Questi livelli possono implementare acknowledgment, timer, ritrasmissioni e numeri di sequenza e, quindi, offrire trasferimento affidabile dei dati al livello superiore. Infatti, nel corso degli anni, ingegneri e informatici hanno indipendentemente progettato e implementato protocolli a livello di collegamento, di rete, di trasporto e di applicazione che offrono un trasferimento affidabile dei dati (sebbene molti di questi protocolli siano scomparsi nel nulla).

Nel Paragrafo 3.5 abbiamo esaminato più da vicino TCP, il protocollo Internet a livello di trasporto affidabile e orientato alla connessione. Abbiamo visto che TCP è complesso e racchiude la gestione della connessione, controllo di flusso, stima del tempo di andata e ritorno e trasferimento affidabile dei dati. TCP è certamente più complicato della nostra descrizione: intenzionalmente non abbiamo trattato una serie di aggiustamenti e miglioramenti che sono diffusamente implementati in diverse versioni di TCP. Tutta questa complessità, comunque, è trasparente alle applicazioni di rete. Se un client vuole inviare dati in modo affidabile a un server su un altro host, può semplicemente aprire una socket TCP verso il server e immettervi i dati. L'applicazione client/server è completamente inconsapevole della complessità di TCP.

Nel Paragrafo 3.6 abbiamo affrontato il controllo di congestione e nel Paragrafo 3.7 abbiamo mostrato come viene implementato da TCP. Abbiamo imparato che il controllo di congestione è assolutamente necessario per il benessere della rete. Senza di esso, la rete potrebbe facilmente rimanere bloccata e non trasportare dati. Abbiamo quindi appreso che TCP classico implementa un meccanismo di controllo di congestione end-to-end che incrementa in modo additivo la velocità di trasmissione quando il percorso della connessione TCP è giudicato sgombro e la decrementa in modo moltiplicativo quando si verificano perdite. Questo meccanismo cerca inoltre di fornire a ciascuna connessione TCP che attraversa un collegamento congestionato una uguale porzione della capacità trasmissiva del collegamento. Abbiamo anche visto alcune varianti più recenti del controllo di congestione di TCP che tentano di determinare il tasso di invio di TCP più rapidamente di quanto faccia TCP classico, utilizzando un approccio basato sul ritardo o sulla notifica esplicita di rete. Abbiamo anche affrontato l'impatto della creazione di connessioni TCP e della slow start sulla latenza, osservando come, in molti scenari importanti, questi due aspetti contribuiscano in modo significativo al ritardo end-to-end. Ancora una volta sottolineiamo che, pur evolvendosi nel corso degli anni, il controllo di congestione TCP rimane un'area di ricerca attiva e probabilmente sarà così anche nei pros-

simi anni. Per concludere, nel Paragrafo 3.8 abbiamo studiato i recenti sviluppi dell'implementazione nel livello di applicazione di molte delle funzioni del livello di trasporto, quali trasferimento dati affidabile, controllo di congestione, creazione della connessione e altro ancora, utilizzando il protocollo QUIC.

Nel corso del Capitolo 1 avevamo detto che una rete di calcolatori può essere suddivisa in “periferia” e “nucleo”. La periferia della rete copre tutto ciò che accade nei sistemi periferici. Avendo ora trattato il livello di applicazione e quello di trasporto, la nostra discussione su questo argomento è completa. È quindi tempo di esplorare il nucleo della rete. Questo viaggio comincia nei prossimi due capitoli, dove studieremo il livello di rete, e continuerà con il Capitolo 6, in cui tratteremo il livello di collegamento.

## Domande e problemi

### Domande di revisione

#### PARAGRAFI 3.1-3.3

**R1.** Supponete che il livello di rete fornisca il seguente servizio. Il livello di rete nell'host sorgente accetta un segmento di dimensione massima di 1200 byte e un indirizzo dell'host di destinazione dal livello di trasporto. Il livello di rete poi garantisce di consegnare il segmento al livello di trasporto nell'host di destinazione. Supponete che molti processi applicativi di rete possano essere in esecuzione sull'host di destinazione.

- (a) Progettate un protocollo a livello di trasporto più semplice possibile che porti i dati applicativi al processo desiderato dell'host di destinazione. Assumete che il sistema operativo nell'host di destinazione abbia assegnato un numero di porta di 4 byte a ciascun processo applicativo in esecuzione.
- (b) Modificate questo protocollo in modo che fornisca un “indirizzo di ritorno” al processo di destinazione.
- (c) Nei vostri protocolli il livello di trasporto deve fare qualcosa nel nucleo della rete di calcolatori?

**R2.** Considerate un pianeta dove tutti appartengono a una famiglia di sei componenti. Tutte le famiglie vivono in una propria casa, ciascuna casa ha un unico indirizzo e ciascuna persona in una casa ha un nome unico. Supponete che questo pianeta abbia un servizio postale che consegna le lettere da una casa sorgente a una casa di destinazione. Il servizio postale richiede che (i) la lettera sia in una busta e (ii) che l'indirizzo della casa di destinazione (e niente più) sia chiaramente scritto sulla busta. Supponete che ciascuna famiglia abbia un membro incaricato che raccoglie e distribuisce le lettere per gli altri membri della famiglia. Le lettere non necessariamente forniscono un'indicazione del loro destinatario.

- (a) Usando la soluzione del problema R1 come riferimento, descrivete un protocollo che gli incaricati possono usare per consegnare le lettere da un membro della famiglia mittente a uno della famiglia destinataria.

(b) Nel vostro protocollo il servizio postale dovrà mai aprire le buste ed esaminare le lettere per fornire il proprio servizio?

**R3.** Considerate una connessione TCP tra gli Host A e B. Supponete che i segmenti TCP in viaggio tra A e B abbiano numero di porta di origine  $x$  e numero di porta di destinazione  $y$ . Quali sono i numeri di porta di origine e destinazione per i segmenti che viaggiano dall'Host B ad A?

**R4.** Descrivete perché lo sviluppatore di un'applicazione potrebbe scegliere di fare uso di UDP anziché di TCP.

**R5.** Perché nell'odierna Internet il traffico voce e video viene spesso inviato su TCP piuttosto che su UDP? *Suggerimento:* la risposta che stiamo cercando non ha nulla a che vedere con il meccanismo di controllo di congestione di TCP.

**R6.** È possibile che un'applicazione che fa uso di UDP ottenga un trasferimento dati affidabile? Come?

**R7.** Supponete che un processo in un host C abbia una socket UDP con un numero di porta 6789. Supponete che entrambi gli host A e B mandino ciascuno un segmento UDP all'host C con numero di porta di destinazione 6789. Entrambi questi segmenti saranno diretti alla stessa socket dell'host C? Se sì, come il processo sull'host C saprà che questi due segmenti hanno origine da due host diversi?

**R8.** Supponete che un web server sia in esecuzione sull'host C sulla porta 80. Supponete che questo web server usa le connessioni persistenti e stia al momento ricevendo richieste da due diversi host, A e B. Tutte le richieste vengono inviate attraverso la stessa socket sull'host C? Se vengono fatte passare attraverso socket diverse, entrambe hanno la porta 80? Discutete e spiegate questa situazione.

### PARAGRAFO 3.4

- R9.** Nei nostri protocolli rdt, perché abbiamo bisogno di introdurre i numeri di sequenza?
- R10.** Nei nostri protocolli rdt, perché abbiamo bisogno di introdurre i timer?
- R11.** Supponete che il ritardo end-to-end tra il mittente e il destinatario sia costante e noto al mittente. Sarebbe ancora necessario un timer nel protocollo rdt 3.0, assumendo che i pacchetti possano andare persi? Motivate la risposta.
- R12.** Provate l'animazione interattiva relativa al Go-back-N presente sulla piattaforma MyLab di questo libro.
- Fate mandare alla sorgente 5 pacchetti e poi fermate l'animazione prima che qualcuno dei 5 pacchetti raggiunga la destinazione. Poi buttate via il primo pacchetto e fate ripartire l'animazione. Descrivete che cosa accade.
  - Ripetete l'esperimento, ma ora lasciate che il primo pacchetto raggiunga la destinazione e buttate via il primo acknowledgment. Descrivete di nuovo che cosa accade.
  - Infine provate a mandare 6 pacchetti. Che cosa accade?
- R13.** Ripetete R12, ma con l'animazione relativa a Selective Repeat. In che modo Selective Repeat e Go-back-N sono diversi?

### PARAGRAFO 3.5

- R14.** Vero o falso?
- L'Host A sta trasmettendo a B un file di grandi dimensioni su una connessione TCP. Ipotizziamo che B non abbia dati da inviare ad A; quindi non manterrà acknowledgment a quest'ultimo, dato che non li può portare "sulle spalle" (*piggyback*) di pacchetti di dati.
  - La dimensione di *rwnd* in TCP non cambia mai per tutta la durata della connessione.
  - Supponiamo che l'Host A stia trasmettendo a B un file di grandi dimensioni su una connessione TCP. Il numero di byte inviati da A e che non hanno ricevuto acknowledgment non può superare la dimensione del buffer di ricezione.
  - Supponiamo che l'Host A stia trasmettendo a B un file di grandi dimensioni su una connessione TCP.

Se il numero di sequenza di un segmento di questa connessione è  $m$ , il numero di sequenza del segmento successivo sarà necessariamente  $m + 1$ .

- I segmenti TCP presentano nella propria intestazione un campo per *rwnd*.
  - Supponiamo che l'ultimo SampleRTT di una connessione TCP valga un secondo. Allora il valore corrente di *TimeoutInterval* per la connessione è necessariamente maggiore o uguale a un secondo.
  - Supponiamo che su una connessione TCP l'Host A stia trasmettendo a B un segmento con numero di sequenza 38 e 4 byte di dati. In questo stesso segmento il numero di acknowledgment è necessariamente 42.
- R15.** Supponiamo che l'Host A stia trasmettendo a B due segmenti su una connessione TCP. Il primo segmento ha numero di sequenza 90 e il secondo 110.
- Quanti dati si trovano nel primo segmento?
  - Supponiamo che il primo segmento vada perso, ma il secondo arrivi a B. Quale sarà il numero di acknowledgment che B manda ad A?
- R16.** Consideriamo l'esempio Telnet trattato nel Paragrafo 3.5. Qualche secondo dopo la digitazione da parte dell'utente della lettera 'C', viene immessa la lettera 'R'. Dopo la digitazione della lettera 'R', quanti segmenti risultano spediti e quali valori sono posti nei campi numero di sequenza e acknowledgment dei segmenti?

### PARAGRAFO 3.7

- R17.** Supponete che due connessioni TCP stiano utilizzando lo stesso collegamento da  $R$  bps che è per loro il collo di bottiglia. Entrambe le connessioni hanno un grosso file da trasmettere (nella stessa direzione) sul collegamento. Le transmissioni dei file iniziano allo stesso istante. Qual è la velocità di trasmissione che TCP vorrebbe assegnare a ciascuna delle connessioni?
- R18.** Considerate il controllo di congestione di TCP. Quando il timer del mittente scade, il valore della soglia *ssthresh* viene dimezzata. Vero o falso?
- R19.** Nella discussione relativa al TCP splitting si è affermato che il tempo di risposta sia circa:
- $$4 \cdot RTT_{FE} + RTT_{BE} + \text{tempo di elaborazione}.$$
- Giustificate questa affermazione.

## Problemi

- P1.** Supponete che il Client A dia inizio a una sessione Telnet con il Server S e che, quasi nello stesso istante, anche il Client B avvii una sessione Telnet con S. Fornite possibili numeri di porta di origine e destinazione per:
- i segmenti trasmessi da A a S
  - i segmenti trasmessi da B a S
  - i segmenti trasmessi da S ad A
  - i segmenti trasmessi da S a B.
- P2.** Relativamente alla Figura 3.5, quali sono i valori delle porte origine e destinazione nei segmenti che fluiscono dal server ai processi dei client? Quali sono gli indirizzi IP nei datagrammi a livello di rete che trasferiscono i segmenti a livello di trasporto?
- P3.** Se A e B sono host diversi, è possibile che il numero di porta di origine dei segmenti da A a S coincida con quello dei segmenti da B a S?
- P4.** Che cosa avviene se si tratta dello stesso host?

**P3.** UDP e TCP utilizzano il complemento a 1 per calcolare il checksum. Supponiamo di avere i seguenti tre byte: 01010011, 01100110 e 01110100. Qual è il complemento a 1 della loro somma? Notiamo che, sebbene UDP e TCP usino checksum da 16 bit, in questo problema consideriamo addendi a 8 bit. Mostrate tutto il procedimento. Perché UDP considera il complemento a 1 della somma, e non semplicemente la somma? Con lo schema del complemento a 1, come vengono rilevati gli errori dal destinatario? È possibile che un errore su 1 bit non venga rilevato? E che cosa avviene per gli errori su 2 bit?

- P4.** (a) Supponete di avere i seguenti 2 byte: 01011100 e 01100101. Qual è il complemento a 1 della somma di questi 2 byte?  
 (b) Supponete di avere i seguenti 2 byte: 11011010 e 01100101. Qual è il complemento a 1 della somma di questi 2 byte?  
 (c) Per i byte nella domanda (a), fornite un esempio dove un bit è cambiato in ciascuno dei 2 byte e il complemento a 1 non cambia.

**P5.** Supponete che un ricevente UDP calcoli il checksum Internet per il segmento UDP ricevuto e trovi che corrisponda al valore trasportato nel campo checksum. Può il ricevente essere assolutamente certo che non vi siano stati errori sui bit? Spiegate perché.

**P6.** Considerate le nostre motivazioni per la correzione del protocollo rdt2.1. Mostrate che quando il destinatario, presentato nella Figura 3.60, opera con il mittente presentato nella Figura 3.11, può portare se stesso e il mittente in uno stato di stallo, in cui entrambi sono in attesa di un evento che non si verificherà mai.

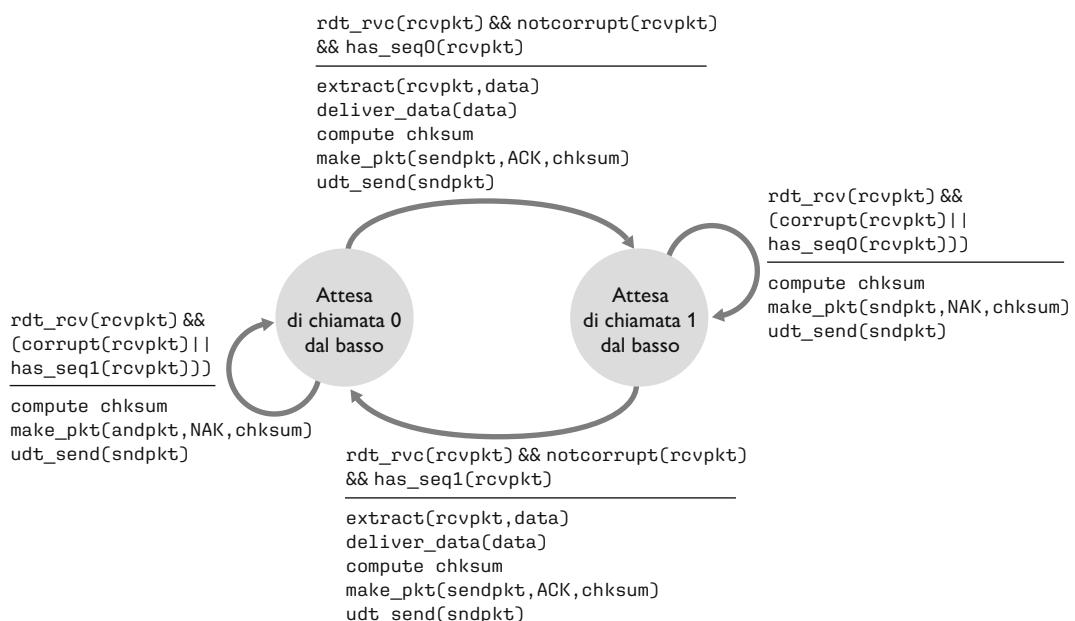
**P7.** Nel protocollo rdt3.0, i pacchetti ACK che fluiscono tra destinatario e mittente non hanno numeri di sequenza (sebbene abbiano un campo ACK contenente il numero di sequenza del pacchetto che stanno riscontrando). Perché i pacchetti ACK non richiedono numeri di sequenza?

**P8.** Disegnate l'automa a stati finiti del lato ricevente del protocollo rdt3.0.

**P9.** Presentate una traccia del funzionamento del protocollo rdt3.0, quando vengono alterati pacchetti di dati o di acknowledgment. La descrizione dovrebbe essere simile a quella della Figura 3.16.

**P10.** Considerate un canale che può perdere pacchetti, ma che presenta un ritardo massimo noto. Modificate il protocollo rdt2.1 per includere timeout e ritrasmissioni del mittente. Argomentate in modo informale la correttezza del vostro protocollo (su tale canale).

**P11.** Considerate il ricevente rdt2.2 della Figura 3.14 e la creazione di un nuovo pacchetto nell'auto-transizione (la transizione da uno stato a sé stesso) negli stati “attesa di chiamata 0 dal basso” e “attesa di chiamata 1 dal basso”: sndpkt=make\_pkt(ACK, 0, checksum) e sndpkt=make\_pkt(ACK, 1, checksum). Il protocollo funzionerebbe correttamente se questa azione venisse rimossa dall'auto-transizione “attesa di chiamata 1 dal basso”? Giustificate la risposta. E se rimossa dall'auto-transizione “attesa di chiamata 0 dal basso”? *Suggerimento:* In quest'ultimo caso considerate che cosa accadrebbe se il primo pacchetto da mittente a ricevente fosse corrotto.



**Figura 3.60** Un ricevente con dei problemi per il protocollo rdt 2.1.

**P12.** Il lato mittente di rdt3.0 ignora, cioè non esegue alcuna azione, sia i pacchetti con errori sia i pacchetti di acknowledgment con un valore errato nel campo acknum. Supponiamo che, in queste circostanze, rdt3.0 abbia semplicemente ritrasmesso il pacchetto di dati corrente. Il protocollo funzionerebbe ancora? *Suggerimento:* ricordate che, se ci sono soltanto errori da un bit, non si verificano perdite di pacchetti, ma possono esserci timeout prematuri. Considerate quante volte viene inviato l' $n$ -esimo pacchetto, quando  $n$  tende all'infinito.

**P13.** Considerate il protocollo rdt 3.0. Tracciate un grafico che mostri che, se la connessione di rete tra mittente e destinatario può riordinare i messaggi (ossia, i due messaggi che si propagano nel mezzo tra mittente e destinatario possono essere riordinati), il protocollo con alternanza di 1 bit non funziona correttamente. Accertatevi di aver identificato con chiarezza in quale direzione il protocollo non funziona. Il vostro diagramma dovrebbe presentare il mittente alla sinistra e il destinatario alla destra, con l'asse temporale che scende lungo la pagina, mostrando lo scambio di messaggi di dati (D) e acknowledgment (A). Accertatevi di indicare il numero di sequenza dei segmenti di dati e acknowledgment.

**P14.** Considerate un protocollo di trasferimento dati affidabile che usa soltanto acknowledgment negativi. Ipotizzate che il mittente trasmetta dati poco frequentemente. In questo caso sarebbe preferibile un protocollo dotato soltanto di NAK, rispetto a un protocollo che utilizza ACK? Perché? Supponete ora che il mittente abbia molti dati da inviare e che la connessione end-to-end presenti poche perdite. In questo secondo caso sarebbe preferibile un protocollo dotato soltanto di NAK o un protocollo che utilizza ACK? Perché?

**P15.** Considerate l'esempio della Figura 3.17. Quanto dovrebbe essere grande l'ampiezza della finestra affinché l'utilizzo del canale superi il 98%? Supponete che il pacchetto sia di 1500 byte, comprensivi di intestazione e dati.

**P16.** Supponete che un'applicazione usi come protocollo di livello di trasporto rdt 3.0. Poiché il protocollo stop-and-wait presenta un basso utilizzo del canale (mostrato nell'esempio in cui si passa da un lato all'altro del continente), i progettisti dell'applicazione lasciano che il ricevente continui a inviare indietro più di due acknowledgment alternati ACK 0 e ACK 1 anche se i dati corrispondenti non sono arrivati al ricevente. Questa progettazione dell'applicazione incrementa l'utilizzo del canale? Perché? Ci sono problemi potenziali in questo approccio? Spiegatelo.

**P17.** Considerate due entità di rete, A e B, connesse da un canale perfettamente bidirezionale (cioè ogni messaggio inviato verrà ricevuto correttamente; il canale non danneggerà, perderà o riordinerà i pacchetti). A e B devono consegnarsi dei messaggi vicendevolmente e in modo alternato: prima, A deve consegnare un messaggio a B, quindi B deve consegnare un messaggio ad A, poi A deve consegnare un messaggio a B e via così. Se un'entità si trova in uno stato in cui non deve tentare di consegnare un messaggio all'altra, è avvenire un evento

come una chiamata `rdt_send(data)` da sopra che tenta di passargli dati per trasmetterli, tale chiamata può essere semplicemente ignorata attraverso una chiamata a `rdt_unable_to_send(data)`, che informa il livello più alto di non essere attualmente in grado di inviare dati. *Nota:* questa assunzione semplificata è fatta in modo che non dobbiate preoccuparvi di gestire un buffer per i dati.

Disegnate una FSM per dare una specifica di questo protocollo (una FSM per A e una per B). Notate che qui non dovete preoccuparvi di un meccanismo affidabile; il punto principale di questo problema è creare una FSM che rifletta il comportamento sincronizzato delle due entità. Dovreste usare i seguenti eventi e azioni che hanno lo stesso significato che avevano nel protocollo rdt1.0 nella Figura 3.9:

```
rdt_send(data), packet=make_pkt(data),
udt_send(packet), rdt_rcv(packet),
extract(packet,data),
deliver_data(data).
```

Assicuratevi che il vostro protocollo rifletta l'alternanza stretta delle trasmissioni tra A e B e che indichiate gli stati iniziali di A e B nelle descrizioni tramite FSM.

**P18.** Nel generico protocollo SR studiato nel Paragrafo 3.4.4 il mittente trasmette un messaggio non appena questo è disponibile (se si trova nella finestra) senza aspettare un acknowledgment. Supponiamo: (1) di volere che il protocollo SR invii due messaggi alla volta (in altre parole, il mittente trasferirà una coppia di messaggi e invierà i due successivi soltanto quando è certo che la prima coppia sia stata ricevuta correttamente); (2) che il canale possa perdere messaggi senza però alterarli o cambiarne l'ordine. Progettate un protocollo unidirezionale di controllo dell'errore per il trasferimento affidabile dei messaggi. Disegnate le FSM di mittente e destinatario. Descrivete il formato dei pacchetti scambiati tra mittente e destinatario e viceversa. Se le vostre chiamate di procedura sono diverse da quelle del Paragrafo 3.4 (per esempio: `udt_send()`, `start_timer()`, `rdt_rcv()` e così via), dichiarate le loro azioni. Fornite un esempio (una linea temporale di mittente e destinatario) che mostri come il vostro protocollo reagisce alla perdita di pacchetti.

**P19.** Considerate uno scenario in cui l'Host A voglia simultaneamente inviare messaggi agli Host B e C. A è connesso a B e C tramite un canale broadcast: i pacchetti inviati da A sono trasportati sia a B sia a C. Supponete che il canale che collega i tre host possa perdere e alterare i messaggi (per esempio, un messaggio inviato da A potrebbe essere correttamente ricevuto da B, ma non da C). Progettate un protocollo di controllo d'errore di tipo stop-and-wait per trasferire in modo affidabile i pacchetti da A a B e C, di modo che A non prenda nuovi dati dal livello superiore finché non sia certo che B e C abbiano correttamente ricevuto il pacchetto corrente. Fornite le FSM per A e C. *Suggerimento:* la FSM di B dovrebbe essere essenzialmente la stessa di C. Inoltre, fornite una descrizione del formato o dei formati di pacchetto utilizzati.

**P20.** Considerate uno scenario nel quale l'Host A e l'Host B vogliono inviare messaggi a un Host C. Gli Host A e C sono collegati da un canale che può perdere e alterare (ma non riordinare) i messaggi. Gli Host B e C sono collegati da un altro canale (indipendente da quello che collega A e C) con le stesse caratteristiche. Il livello di trasporto all'Host C si dovrebbe alternare nell'invio di messaggi da A e B al livello superiore (cioè, dovrebbe prima consegnare i dati dei pacchetti da A, poi quelli dei pacchetti da B e così via). Progettate un protocollo per il controllo degli errori tipo stop-and-wait, per un trasferimento affidabile dei pacchetti da A, B e C, con la consegna alternata di C, descritta sopra. Fornite la descrizione delle FSM di A e C e fornite anche una descrizione dei formati dei pacchetti usati. *Suggerimento:* l'automa di B dovrebbe essere essenzialmente uguale a quello di A.

**P21.** Supponete di avere due entità di rete, A e B. I messaggi di dati di B verranno inviati ad A secondo le seguenti convenzioni.

- (a) Quando A riceve una richiesta dal livello superiore per ottenere il prossimo messaggio di dati (D) da B, deve inviare un messaggio di richiesta (R) a B sul canale tra A e B.
- (b) B può inviare un messaggio di dati (D) ad A sul canale tra B e A soltanto quando riceve un messaggio R.
- (c) A dovrebbe consegnare esattamente una copia di ciascun messaggio D al livello superiore.
- (d) Sul canale tra A e B i messaggi R possono essere perduti, ma non alterati.
- (e) I messaggi D, una volta inviati, vengono sempre consegnati correttamente.
- (f) Il ritardo lungo i canali non è noto ed è variabile.

Progettate (fornendo una descrizione con una FSM) un protocollo che incorpora i meccanismi appropriati per compensare le possibili perdite del canale tra A e B e che implementa il trasferimento di messaggi al livello superiore presso l'entità A, come discusso precedentemente. Utilizzate solo i meccanismi assolutamente necessari.

**P22.** Considerate il protocollo GBN con un'ampiezza della finestra mittente pari a 4 e intervallo di numeri di sequenza da 0 a 1024. Supponete che all'istante  $t$  il successivo pacchetto nella sequenza attesa dal destinatario abbia numero di sequenza  $k$ . Ipotizzate che il mezzo non effettui riordino dei messaggi. Rispondete alle seguenti domande.

- (a) Quali sono i possibili numeri di sequenza all'interno della finestra del mittente all'istante  $t$ ? Giustificate la vostra risposta.
- (b) Quali sono tutti i possibili valori del campo ACK nei possibili messaggi che si propagano all'indietro verso il mittente all'istante  $t$ ? Giustificate la vostra risposta.

**P23.** Considerate i protocolli GBN e SR. Supponete che lo spazio dei numeri di sequenza abbia dimensione  $k$ . Qual è, per i due protocolli, la finestra mittente più grande che evita il verificarsi di problemi come quelli presentati nella Figura 3.27?

**P24.** Rispondete alle seguenti domande, giustificando brevemente le vostre risposte.

- (a) Nel protocollo SR, il mittente può ricevere un ACK relativo a un pacchetto che ricade al di fuori della sua finestra corrente?
- (b) Nel protocollo GBN, il mittente può ricevere un ACK relativo a un pacchetto che ricade al di fuori della sua finestra corrente?
- (c) Il protocollo stop-and-wait è uguale al protocollo SR con ampiezza di finestra mittente e destinatario pari a 1?
- (d) Il protocollo stop-and-wait è uguale al protocollo GBN con ampiezza di finestra mittente e destinatario pari a 1?

**P25.** Abbiamo detto che un'applicazione potrebbe scegliere UDP come protocollo di trasporto, perché UDP offre un controllo applicativo più preciso di TCP su quali dati vengono inviati in un segmento e quando.

- (a) Perché un'applicazione deve avere più controllo su quali dati vengono inviati in un segmento?
- (b) Perché un'applicazione deve avere più controllo su quando un segmento viene inviato?

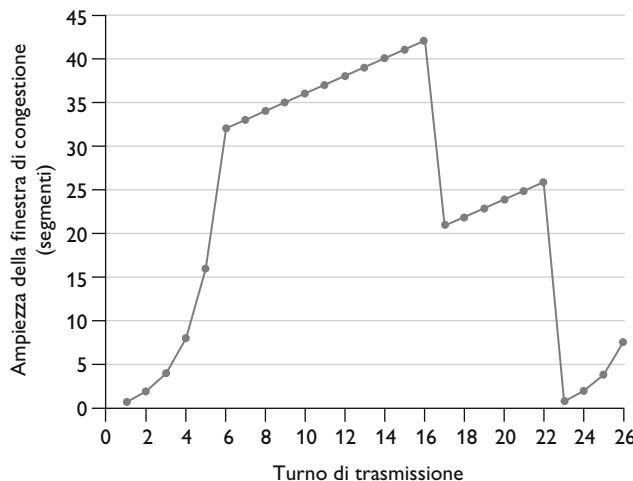
**P26.** Considerate il trasferimento di un file enorme di  $L$  byte dall'Host A all'Host B. Ipotizzate un MSS pari a 536 byte.

- (a) Qual è il valore massimo di  $L$  tale per cui i numeri di sequenza TCP non vengono esauriti? Ricordiamo che il campo Numero di sequenza in TCP è composto da quattro byte.
- (b) Per il valore di  $L$  ottenuto in (a) determinate quanto impiegherebbe la trasmissione del file. Ipotizzate l'aggiunta di un numero totale di 66 byte per le intestazioni di trasporto, di rete e di collegamento a ciascun segmento prima che il pacchetto risultante venga immesso su un collegamento da 155 Mbps. Ignorate il controllo di flusso e di congestione di modo che A possa immettere i segmenti end-to-end e con continuità.

**P27.** L'Host A e l'Host B stanno comunicando su una connessione TCP e l'Host B ha appena ricevuto da A tutti i byte fino al 126. Supponete che l'Host A mandi 2 segmenti all'Host B uno dietro l'altro. I due segmenti contengono rispettivamente 80 e 40 byte di dati. Nel primo segmento il numero di sequenza è 127, il numero di porta sorgente è 302 e il numero di porta di destinazione 80. L'Host B manda un acknowledgment non appena riceve un segmento dall'Host A.

- (a) Nel secondo segmento, mandato dall'Host A al B, quali sono il numero di sequenza, il numero di porta sorgente e il numero di porta di destinazione?
- (b) Se il primo segmento arriva prima del secondo, nell'acknowledgment del primo segmento arrivato, qual è il numero di acknowledgment, il numero di porta sorgente e il numero di porta di destinazione?
- (c) Se il secondo segmento arriva prima del primo, nell'acknowledgment del primo segmento arrivato, qual è il numero di acknowledgment, il numero di porta sorgente e il numero di porta di destinazione?

- (d) Supponete che i due segmenti inviati da A arrivino in ordine a B. Il primo acknowledgment va perduto e il secondo arriva dopo il primo intervallo di timeout. Disegnate un diagramma temporale che mostra questi segmenti e tutti gli altri segmenti e gli acknowledgment inviati. Assumete che non vi siano perdite aggiuntive di pacchetti. Per ciascun segmento nella vostra figura indicate il numero di sequenza, il numero di byte di dati, mentre, per ciascun acknowledgment che aggiungete, indicate il numero di acknowledgment.
- P28.** L'Host A e l'Host B sono direttamente connessi con un collegamento a 100 Mbps. C'è una connessione TCP tra i due host e l'Host A sta mandando all'Host B un grosso file su quella connessione. L'Host A può inviare dati applicativi nel collegamento a 120 Mbps, ma l'Host B può leggere dal suo buffer di ricezione TCP a una frequenza massima di 50 Mbps. Descrivete l'effetto del controllo di flusso TCP.
- P29.** I SYN cookie sono stati trattati nel Paragrafo 3.5.6.
- Perché è necessario che il server usi uno speciale numero di sequenza iniziale in SYNACK?
  - Supponete che un aggressore sappia che un host bersaglio usa i SYN cookie. L'aggressore può creare una connessione mezza aperta o completamente aperta inviando semplicemente un pacchetto ACK al bersaglio? Spiegate il perché.
  - Supponete che un aggressore raccolga una gran quantità di numeri di sequenza iniziali inviati dal server. L'aggressore può fare in modo che il server crei molte connessioni completamente aperte inviando ACK con i numeri di sequenza raccolti? Spiegate il perché.
- P30.** Considerate la rete mostrata nello Scenario 2 nel Paragrafo 3.6.1. Supponete che entrambi gli host mittente, A e B, abbiano dei valori fissati per il timeout.
- Dimostrate che aumentando la dimensione del buffer finito del router sia possibile diminuire il throughput ( $\lambda_{out}$ ).
  - Supponete ora che entrambi gli host regolino dinamicamente i loro valori di timeout, come fa TCP, in base al ritardo di accodamento sul router. L'incremento della dimensione del buffer dovrebbe aiutare a innalzare il valore di throughput? Perché?
- P31.** Supponete che cinque valori misurati di SampleRTT (si veda il Paragrafo 3.5.3) siano 106 ms, 120 ms, 140 ms, 90 ms e 115 ms. Calcolate EstimatedRTT dopo l'acquisizione di ogni valore di SampleRTT, usando un valore  $\alpha = 0,125$  e assumendo che il valore di EstimatedRTT appena prima dell'acquisizione del primo di questi cinque campioni fosse 100 ms. Calcolate anche DevRTT dopo l'acquisizione di ogni campione, assumendo  $\beta = 0,25$  e che il valore di DevRTT appena prima dell'acquisizione del primo di questi cinque campioni fosse 5 ms. Infine calcolate il valore di TCP TimeoutInterval dopo l'acquisizione di ogni campione.
- P32.** Considerate la procedura TCP per stimare RTT. Ipotizzate  $\alpha = 0,1$  e che SampleRTT<sub>1</sub> sia il più recente campione RTT, SampleRTT<sub>2</sub> sia il secondo più recente campione RTT, e così via.
- Per una data connessione TCP, supponete che siano stati restituiti 4 acknowledgment con i corrispondenti RTT campione: SampleRTT<sub>4</sub>, SampleRTT<sub>3</sub>, SampleRTT<sub>2</sub>, SampleRTT<sub>1</sub>. Esprimete EstimatedRTT in funzione dei quattro campioni.
  - Generalizzate la vostra formula per  $n$  campioni.
  - Nella formula al punto (b), fate tendere  $n$  a infinito. Spiegate perché questa procedura di media venga detta media mobile esponenziale.
- P33.** Nel Paragrafo 3.5.3 abbiamo discusso la stima di RTT in TCP. Perché ritenete che TCP eviti la misurazione del SampleRTT per i segmenti ritrasmessi?
- P34.** Qual è la relazione tra la variabile SendBase nel Paragrafo 3.5.4 e la variabile LastByteRcvd nel Paragrafo 3.5.5?
- P35.** Qual è la relazione tra la variabile LastByteRcvd nel Paragrafo 3.5.5 e la variabile y nel Paragrafo 3.5.4?
- P36.** Nel Paragrafo 3.5.4 abbiamo visto che TCP attende la ricezione di tre ACK duplicati prima di effettuare una ritrasmissione rapida. Perché pensate che i progettisti TCP abbiano scelto di non effettuare una ritrasmissione rapida dopo la ricezione del primo ACK duplicato per un segmento?
- P37.** Paragonate GBN, SR e TCP (senza il delayed ACK). Assumete che i valori di timeout per tutti questi tre protocolli siano sufficientemente grandi per cui cinque segmenti di dati consecutivi e i corrispondenti ACK possano essere ricevuti (se non persi nel canale) dall'host ricevente (Host B) e dall'host mittente (Host A) rispettivamente. Supponete che l'Host A invii cinque segmenti di dati all'Host B e che il secondo segmento (invia da A) venga perso. Alla fine, tutti e 5 i segmenti di dati sono stati ricevuti correttamente dall'Host B.
- Quanti segmenti ha inviato l'Host A in tutto e quanti ACK ha mandato l'Host B in tutto? Quali sono i loro numeri di sequenza? Rispondete a queste domande per tutti e tre i protocolli.
  - Se i valori di timeout per tutti e tre i protocolli sono molto più grandi di 5 RTT, quale protocollo consegnerà con successo tutti e cinque i segmenti di dati nell'intervallo di tempo più breve?
- P38.** Nella descrizione di TCP nella Figura 3.52 il valore della soglia, ssthresh, è posto a  $ssthresh=cwnd/2$  in molte parti e il valore di ssthresh è posto a metà dell'ampiezza di finestra quando accade un evento di perdita. Il tasso a cui il mittente deve trasmettere quando avviene un evento di perdita deve essere circa uguale a cwnd segmenti ogni RTT? Motivate la vostra risposta. Se la vostra risposta è no, potete suggerire un modo diverso in cui definire il valore di ssthresh?
- P39.** Considerate la Figura 3.46(b). Se  $\lambda'_{in}$  supera  $R/2$ , può  $\lambda'_{out}$  oltrepassare  $R/3$ ? Fornite una spiegazione. Considerate ora la Figura 3.46(c). Se  $\lambda'_{in}$  supera  $R/2$ , può  $\lambda'_{out}$  oltrepassare  $R/4$  sotto l'ipotesi che un pacchetto sia



**Figura 3.61** Ampiezza della finestra di TCP in funzione del tempo.

inoltrato in media due volte dal router al destinatario? Fornite una spiegazione.

**P40.** Considerate la Figura 3.61. Ipotizzando che sia il protocollo TCP Reno ad avere il comportamento illustrato nella figura, rispondete alle seguenti domande giustificando sempre la risposta.

- (a) Identificate gli intervalli di tempo in cui opera la slow start.
- (b) Identificate gli intervalli di tempo in cui opera la congestion avoidance.
- (c) Dopo il 16° turno di trasmissione, la perdita di segmenti viene rilevata da un triplice ACK duplicato o da un timeout?
- (d) Dopo il 22° turno di trasmissione, la perdita di segmenti viene rilevata da un triplice ACK duplicato o da un timeout?
- (e) Qual è il valore iniziale di  $ssthresh$  nel 1° turno di trasmissione?
- (f) Qual è il valore iniziale di  $ssthresh$  nel 18° turno di trasmissione?
- (g) Qual è il valore iniziale di  $ssthresh$  nel 24° turno di trasmissione?
- (h) Durante quale turno di trasmissione viene inviato il 70° segmento?
- (i) Ipotizzando il rilevamento della perdita di pacchetto dopo il 26° turno tramite ricezione di un triplice ACK duplicato, quali saranno i valori dell'ampiezza della finestra di congestione e di  $ssthresh$ ?
- (j) Supponete che venga usato TCP Tahoe (invece che TCP Reno) e assumete che un triplice ACK duplicato venga ricevuto al sedicesimo turno di trasmissione. Qual è il valore di  $ssthresh$  e dell'ampiezza della finestra di congestione al diciannovesimo turno?
- (k) Supponete ancora che venga usato TCP Tahoe e che si verifichi un evento di timeout al ventiduesimo round. Quanti pacchetti sono stati inviati dal diciassettesimo al ventiduesimo turno incluso?

**P41.** Fate riferimento alla Figura 3.57, che mostra la convergenza dell'algoritmo AIMD di TCP. Supponete che al posto di un decremento moltiplicativo, TCP diminuisca l'ampiezza della finestra di una quantità costante. Il risultante AIMD convergerebbe verso una ripartizione equa? Giustificate la vostra risposta utilizzando un diagramma simile a quello nella Figura 3.57.

**P42.** Nel Paragrafo 3.5.4 abbiamo descritto il raddoppio dell'intervalllo di timeout dopo un evento di timeout come una forma di controllo di congestione. Perché TCP necessita di un meccanismo basato su una finestra per il controllo di congestione (Paragrafo 3.7) oltre a quello di raddoppio dell'intervalllo di timeout?

**P43.** L'Host A sta inviando un file molto grande a B su una connessione TCP senza perdita di pacchetti e in cui i timer non scadono mai. Denotate il tasso trasmissivo del collegamento che connette l'Host A a Internet con  $R$  bps. Ipotizzate che il processo nell'Host A sia in grado di trasmettere dati nella sua socket TCP a un tasso di  $S$  bps, dove  $S = 10 \cdot R$ . Supponete, inoltre, che il buffer di ricezione TCP sia capiente a sufficienza per contenere l'intero file e che il buffer di invio possa contenere solo l'1% del file. Che cosa consentirebbe al processo nell'Host A di non passare continuamente dati alla propria socket alla frequenza  $S$  bps? Il controllo di flusso TCP? Il controllo di congestione TCP? O qualcos'altro?

**P44.** Considerate l'invio di un grosso file da un host all'altro su una connessione TCP senza perdite.

- (a) Supponete che TCP usi AIMD senza slow start per il controllo di congestione. Assumendo che  $cwnd$  aumenti di 1 MSS ogni volta che un gruppo di ACK viene ricevuto e assumendo approssimativamente costanti i tempi di andata e ritorno, quanto ci metterà  $cwnd$  ad aumentare da 6 MSS a 12 MSS (assumendo che non vi siano eventi di perdita)?
- (b) Qual è il throughput medio (in termini di MSS e RTT) per questa connessione fino all'istante pari a  $6 \cdot RTT$ ?

- P45.** Considerate la Figura 3.54. Supponete che al tempo  $t_3$  il tasso di invio a cui avverrebbe la successiva perdita si abbassi a  $0,75 * W_{\max}$  (all’insaputa dei mittenti TCP, ovviamente). Mostrate l’evoluzione di TCP Reno e TCP CUBIC per altri due round ciascuno (Suggerimento: notate che i tempi in cui TCP Reno e TCP CUBIC reagiscono alla perdita potrebbero non essere più gli stessi).
- P46.** Considerate di nuovo la Figura 3.54. Supponete che al tempo  $t_3$  il tasso di invio a cui avverrebbe la successiva perdita cresca  $1,5 * W_{\max}$ . Mostrate l’evoluzione di TCP Reno e TCP CUBIC per altri due round ciascuno (Suggerimento: si veda il suggerimento di P45).
- P47.** Si pensi alla descrizione ad alto livello del throughput di TCP. Nel periodo di tempo in cui la velocità di trasferimento della connessione varia da  $W/(2 \cdot RTT)$  a  $W/RTT$  viene smarrito un solo pacchetto (proprio alla fine dell’intervallo temporale).
- Mostrate che il tasso di perdita vale
- $$L = \text{tasso di perdita} = \frac{1}{\frac{3}{8}W^2 + \frac{3}{4}W}$$
- Usate il risultato precedente per dimostrare che, se una connessione presenta un tasso di perdita  $L$ , la sua banda media è approssimativamente data da
- $$\approx \frac{1,22 \cdot MSS}{RTT\sqrt{L}}$$
- P48.** Supponete che una singola connessione TCP Reno usi un collegamento da 10Mbps. Supponete che tale collegamento sia l’unico congestionato tra gli host mittente e ricevente. Assumete che il mittente TCP abbia un file di grandi dimensioni da inviare al ricevente e che il buffer di ricezione del destinatario sia molto più grande della finestra di congestione. Facciamo inoltre le seguenti ipotesi: la grandezza di ogni segmento TCP sia 1500 byte; il ritardo di propagazione nelle due direzioni della connessione sia di 150 ms; infine che questa connessione TCP sia sempre nella fase di congestione avoidance, ignorando lo slow start.
- Qual è la massima ampiezza della finestra, espressa in segmenti, che questa connessione TCP può raggiungere?
  - Qual è l’ampiezza media della finestra, espressa in segmenti, e il throughput medio, in bps, di questa connessione TCP?
  - Quanto ci mette questa connessione TCP per raggiungere di nuovo il valore massimo di finestra dopo il ripristino causato dalla perdita di un pacchetto?
- P49.** Considerate lo scenario descritto nel problema precedente. Supponete che il collegamento da 10Mbps possa utilizzare un buffer per un numero finito di segmenti. Dimostrate che, affinché il collegamento sia sempre occupato nella trasmissione di dati, dovremmo scegliere una dimensione del buffer che sia almeno pari al prodotto della velocità  $C$  del collegamento e del ritardo di propagazione nelle due direzioni tra il mittente e il ricevente.
- P50.** Ripetete il Problema P46 sostituendo il collegamento da 10 Mbps con uno da 10 Gbps. In particolare, nella vostra risposta al punto (c) vedrete che l’ampiezza della finestra di congestione ci metterà un tempo molto lungo per raggiungere di nuovo il valore massimo dopo il ripristino dovuto alla perdita di un pacchetto. Abbozzate una soluzione per risolvere questo problema.
- P51.** Sia  $T$  (misurato in  $RTT$ ) l’intervallo di tempo che una connessione TCP impiega per aumentare la sua finestra di congestione da  $W/2$  a  $W$ , dove  $W$  è la dimensione massima. Dimostrate che  $T$  è funzione del throughput medio.
- P52.** Considerate una versione semplificata dell’algoritmo AIMD di TCP nella quale la finestra di congestione sia misurata in segmenti invece che in byte. Nell’incremento additivo l’ampiezza della finestra di congestione cresce di un segmento per  $RTT$ . Nel decremento moltiplicativo la finestra di congestione decresce di metà (se il risultato non è un intero, arrotondate per difetto all’intero più vicino). Supponete che due connessioni TCP,  $C_1$  e  $C_2$ , condividano un singolo collegamento congestionato con velocità pari a 30 segmenti al secondo. Assumete che  $C_1$  e  $C_2$  siano nella fase di congestione avoidance. L’ $RTT$  della connessione  $C_1$  è di 50 ms, quello della connessione  $C_2$  di 100 ms. Assumete che quando il tasso di invio dei dati eccede la velocità del canale, tutte le connessioni TCP soffrano una perdita di pacchetti.
- Se entrambe  $C_1$  e  $C_2$  al tempo  $t_0$  hanno una finestra di congestione di 10 segmenti, qual è l’ampiezza della loro finestra di congestione dopo 1000 ms?
  - A lungo termine, queste due connessioni otterranno la stessa porzione di banda del collegamento congestionato? Motivate la risposta.
- P53.** Considerate la rete descritta nel precedente problema. Supponete ora che le due connessioni,  $C_1$  e  $C_2$ , abbiano lo stesso  $RTT$  di 100 ms. Supponete che al tempo  $t_0$ , l’ampiezza della finestra di congestione di  $C_1$  sia di 15 segmenti, quella di  $C_2$  di 10 segmenti.
- Quali sono le dimensioni delle loro finestre di congestione dopo 2200 ms?
  - A lungo termine, queste due connessioni otterranno circa la stessa porzione di banda del collegamento congestionato?
  - Diciamo che due connessioni sono sincronizzate se entrambe raggiungono il massimo valore di finestra contemporaneamente. A lungo termine queste due connessioni diventeranno infine sincronizzate? In tal caso qual è il valore massimo della finestra?
  - Tale sincronizzazione aiuterà ad aumentare l’utilizzo del collegamento condiviso? Perché? Abbozzate qualche idea per rompere tale sincronizzazione.
- P54.** Considerate una modifica dell’algoritmo di controllo di congestione TCP. Al posto di un incremento additivo, possiamo usarne uno moltiplicativo. Un mittente TCP aumenta la dimensione della sua finestra di una costante positiva piccola  $a$  ( $0 < a < 1$ ) ogni volta che riceve un ACK valido. Trovate la relazione funzionale tra il tasso di perdita  $L$  e la massima ampiezza della finestra di congestione  $W$ . Dimostrate che per questa versione modi-

ficata di TCP, indipendentemente dal throughput medio di TCP, la connessione spende sempre lo stesso tempo per aumentare la sua finestra di congestione da  $W/2$  a  $W$ .

**P55.** Nella nostra trattazione degli sviluppi futuri di TCP (Paragrafo 3.7), abbiamo notato che per ottenere un throughput di 10 Gbps, TCP potrebbe tollerare solo una probabilità di perdita di segmenti pari a  $2 \cdot 10^{-10}$  (o equivalentemente, una perdita ogni 5 miliardi di segmenti). Mostrate come si ricavano tali valori per l'RTT e l'MSS dati nel Paragrafo 3.7. Se TCP dovesse supportare una connessione a 100 Gbps, quale perdita sarebbe tollerabile?

**P56.** Nella nostra trattazione del controllo di congestione di TCP nel Paragrafo 3.7, abbiamo assunto implicitamente che il mittente TCP debba sempre inviare dati. Consideriamo ora il caso in cui il mittente TCP trasferisca una gran quantità di dati e poi diventi inattivo (non avendo altri dati da spedire) all'istante  $t_1$ . TCP rimane in tale stato per un periodo di tempo relativamente lungo e poi desidera inviare altri dati, precisamente all'istante  $t_2$ . Quali sono i vantaggi e gli svantaggi di far usare a TCP i valori cwnd e ssthresh dall'istante  $t_1$  all'istante  $t_2$ ? Quali alternative consigliereste? Perché?

**P57.** In questo problema studieremo se UDP o TCP forniscono un certo grado di autenticazione end-to-end.

- (a) Considerate un server che riceve una richiesta all'interno di un pacchetto UDP e risponde a quella

richiesta all'interno di un altro pacchetto UDP (per esempio, come fa il server DNS). Se un client con un indirizzo IP X camuffa il proprio indirizzo con l'indirizzo Y, dove manderà il server la sua risposta?

(b) Supponete che un server riceva un SYN con un indirizzo IP sorgente Y e dopo aver risposto con un SYNACK, riceva un ACK con l'indirizzo IP sorgente Y e numero di acknowledgment corretto. Assumendo che il server scelga un numero di sequenza iniziale casuale e non vi sia "qualcuno nel mezzo", può il server essere certo che il client sia effettivamente Y (e non qualcun altro con indirizzo X camuffato con Y)?

**P58.** In questo problema considereremo il ritardo introdotto dalla fase di slow start di TCP. Considerate un client e un web server direttamente connessi da un collegamento di capacità  $R$ . Supponete che il client voglia reperire un oggetto le cui dimensioni siano esattamente uguali a  $15 S$ , dove  $S$  è la dimensione massima del segmento (MSS). Indicate il tempo di andata e ritorno tra il client e il server con  $RTT$  (assumendo che sia costante). Ignorando le intestazioni del protocollo, determinate il tempo per ottenere l'oggetto (compreso il tempo di instaurazione della connessione TCP) quando:

- (a)  $4 S/R > S/R + RTT > 2S/R$
- (b)  $S/R + RTT > 4 S/R$
- (c)  $S/R > RTT$

## Esercizi di programmazione

### Protocollo di trasporto affidabile

Con questa esercitazione vi si richiede di scrivere il codice a livello di trasporto per l'invio e la ricezione, implementando un semplice protocollo di trasferimento dati affidabile. Esistono due versioni di questa esercitazione, quella per il protocollo ad alternanza di bit e quella con GBN. Questo compito dovrebbe essere divertente: la vostra implementazione si discosterà assai poco da quanto richiesto in una situazione reale.

Dato che probabilmente non avete macchine stand-alone (con un sistema operativo che potete modificare), il vostro codice dovrà essere eseguito in un ambiente simulato. In ogni caso, l'interfaccia di programmazione offerta alle vostre routine – il codice che chiama le vostre entità dall'alto e dal basso – si avvicina molto a quanto avviene in un reale ambiente UNIX. Infatti, le interfacce software descritte in questo compito sono molto più realistiche dei mittenti e dei destinatari con cicli infiniti descritti da molti testi. Viene simulato anche il blocco e la partenza dei timer, e i relativi interrupt provocheranno l'attivazione delle vostre routine di gestione dei timer.

L'esercitazione completa, come pure il codice che dovrete compilare insieme al vostro, è disponibile sulla piattaforma MyLab del volume.

### Esercitazione Wireshark: esplorando TCP

In questa esercitazione userete il vostro browser per accedere a un file di un web server. Anche qui utilizzerete Wireshark per catturare i pacchetti che giungono al vostro calcolatore. A differenza di quanto visto precedentemente, sarete in grado di scaricare dal

server un tracciato di pacchetti leggibile da Wireshark. In questa traccia troverete i pacchetti generati dal vostro accesso al server. Analizzerete le tracce dal lato client e server per esplorare gli aspetti di TCP. In particolare, valuterete le prestazioni della connessione TCP tra il vostro calcolatore e il web server. Terrete traccia del comportamento della finestra TCP, ricavando come conseguenza il comportamento sulla perdita di pacchetti, sulla ritrasmissione, sul controllo di flusso, sul controllo di congestione e sulla stima dell'*RTT*.

Come in tutte le altre esercitazioni Wireshark, la descrizione completa è disponibile sulla piattaforma MyLab di questo libro.

## **Esercitazione Wireshark: esplorando UDP**

In questa breve esercitazione catturerete dei pacchetti e analizzerete una delle vostre applicazioni preferite che usa UDP (per esempio, il DNS o un'applicazione multimediale come Skype). Come abbiamo visto nel Paragrafo 3.3, UDP è un protocollo di trasporto semplice e senza fronzoli. In questa esercitazione, esplorerete i campi dell'intestazione dai segmenti UDP come pure il calcolo del checksum.

Come in tutte le altre esercitazioni Wireshark, la descrizione completa è disponibile sulla piattaforma MyLab di questo libro.

# Livello di rete: il piano dei dati

Nel precedente capitolo abbiamo appreso che il livello di trasporto fornisce varie forme di comunicazione tra processi facendo affidamento sul servizio di comunicazione tra host a livello di rete, senza però sfruttare alcuna conoscenza di come tale servizio sia effettivamente implementato.

In questo e nel prossimo capitolo vedremo esattamente come fa il livello di rete a fornire tale servizio di comunicazione host-to-host. Vedremo che, a differenza dei livelli di trasporto e applicazione, *una parte del livello di rete è presente sia negli host sia nei router della rete*, ed è forse per questo che i protocolli risultano tra i più impegnativi e complessi, e pertanto tra i più interessanti, nella pila di protocolli.

Data la sua complessità e la quantità di argomenti, tratteremo il livello di rete in due capitoli. Vedremo che può essere diviso in due parti interagenti: il **piano dei dati** (*data plane*) e il **piano di controllo** (*control plane*).

Nel presente capitolo tratteremo dapprima le funzioni del piano dei dati del livello di rete: le funzioni che ogni router implementa localmente sulla base di regole date per determinare come un datagramma che arriva a uno dei collegamenti in entrata del router è inoltrato a uno dei collegamenti in uscita del router. Tratteremo sia l'inoltro (*forwarding*) IP tradizionale, basato sull'indirizzo di destinazione del datagramma, sia l'inoltro generalizzato, nel quale l'inoltro e le altre funzioni possano essere effettuate utilizzando i valori in alcuni campi dell'intestazione del datagramma. Studieremo nel dettaglio i protocolli IPv4 e IPv6 e l'indirizzamento Internet.

Nel Capitolo 5 tratteremo le funzioni del piano di controllo del livello di rete: la logica globale di rete (*network-wide*) che controlla come i datagrammi sono instradati tra i router su un percorso *da estremo a estremo* (*end-to-end*) tra host sorgente e host destinazione. Tratteremo gli algoritmi di instradamento, così come i protocolli di instradamento quali OSPF e BGP, ampiamente usati nella Internet di oggi.

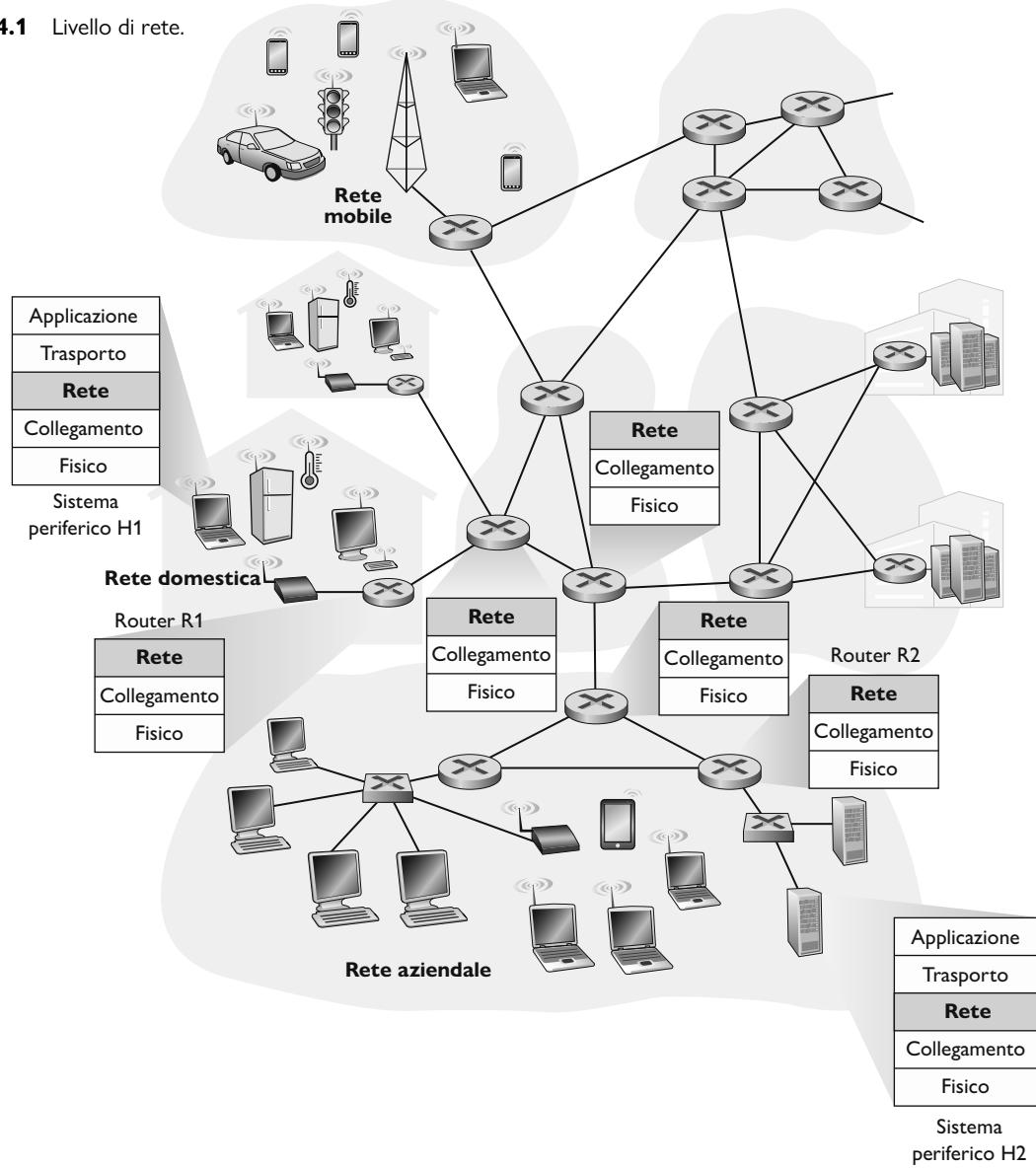
I protocolli di instradamento del piano di controllo e le funzioni di inoltro del piano dei dati sono tradizionalmente implementati insieme nei router con approccio comunemente indicato come "monolitico". Il Software-Defined Networking (SDN), invece, separa esplicitamente il piano dei dati e il piano di controllo, implementando le funzioni del piano di controllo come un servizio separato, tipicamente in un "controller" centralizzato e remoto. Anche i controller SDN saranno trattati nel Capitolo 5.

La distinzione tra le funzioni del piano dei dati e quelle del piano di controllo a livello di rete è un concetto chiave da tenere a mente per comprendere la visione moderna del ruolo del livello di rete nel networking.

## 4.1 Panoramica del livello di rete

Osserviamo la Figura 4.1 in cui viene illustrata una semplice rete con due host, H1 e H2 e alcuni router sul percorso che li collega. Consideriamo la funzione svolta dal livello di rete e il ruolo dei router nell’ambito di una trasmissione da

**Figura 4.1** Livello di rete.



H1 a H2. Lato mittente, il livello di rete in H1 prende i segmenti dal livello di trasporto, li incapsula in un datagramma, cioè un pacchetto a livello di rete, che trasmette al proprio router vicino, R1. Lato destinazione, il livello di rete in H2 riceve i datagrammi dal proprio router vicino R2, estrae i segmenti e li consegna al livello di trasporto. Nel tragitto, il ruolo primario del piano dei dati di ciascun router intermedio è quello di inoltrare il datagramma da link d'ingresso a link d'uscita, mentre quello del piano di controllo è coordinare queste azioni di inoltro locali in modo che alla fine i datagrammi vengano instradati da estremo a estremo (*end-to-end*) su percorsi di router tra mittente e destinatario. Notiamo che, nella figura, la pila di protocolli nei router non ha ulteriori livelli sopra quello di rete, in quanto i router non eseguono protocolli a livello di applicazione e di trasporto come quelli visti nei Capitoli 2 e 3.

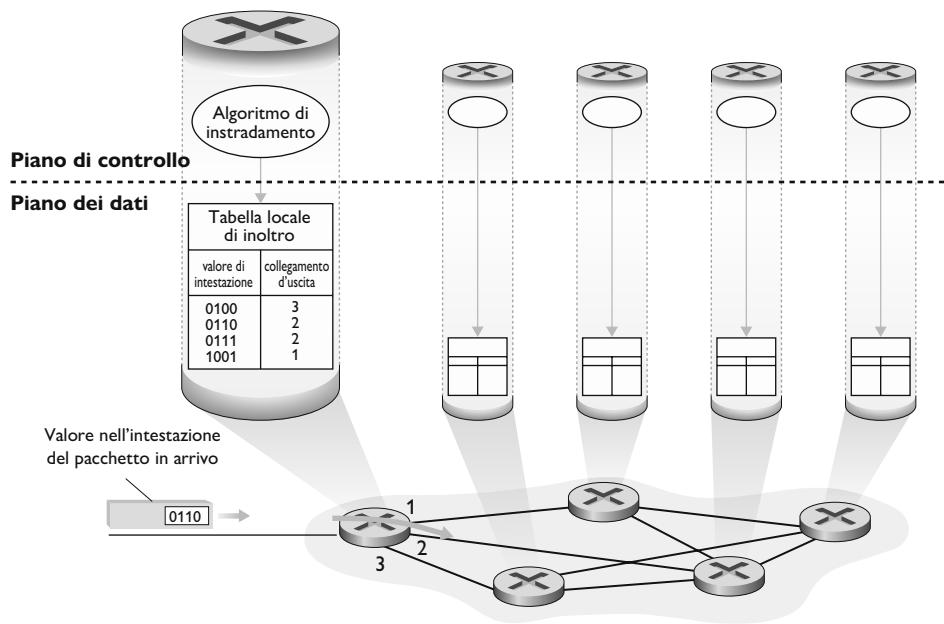
#### 4.1.1 Inoltro e instradamento: piano dei dati e piano di controllo

Il ruolo principale del livello di rete è quindi piuttosto semplice: trasferire pacchetti da un host a un altro. Per fare questo è possibile identificare due importanti funzioni.

- **Inoltro (forwarding).** Quando un router riceve un pacchetto, lo deve trasferire sull'appropriato collegamento di uscita. Per esempio, un pacchetto che arriva dall'host H1 al router R1, deve essere inoltrato al successivo router sul percorso verso H2. Vedremo che l'inoltro non è che una, anche se la più importante, delle funzioni implementate nel piano dei dati. Nel caso più generale, che tratteremo nel Paragrafo 4.4, un pacchetto può anche essere bloccato (per esempio se inviato da un mittente malevolo o destinato a un host vietato) o duplicato e inviato su più collegamenti di uscita.
- **Instrandamento (routing).** Il livello di rete deve determinare il percorso che i pacchetti devono seguire tramite **algoritmi di instradamento** (*algoritmi di routing*). Un algoritmo di instradamento determina, per esempio, il percorso seguito dai pacchetti da H1 a H2 nella Figura 4.1. La funzione di instradamento è implementata nel piano di controllo del livello di rete.

Nella letteratura inerente al livello di rete, i termini *inoltro* e *instradamento* sono sovente considerati sinonimi; in questo testo li distingueremo, utilizzandoli nella loro precisa accezione. Con **inoltro** faremo quindi riferimento all'azione locale con cui il router trasferisce i pacchetti da un'interfaccia di ingresso a quella di uscita. Poiché l'inoltro avviene su scala temporale molto piccola, dell'ordine di pochi nanosecondi, è usualmente implementato in hardware. Con **instradamento** indicheremo, invece, il processo globale di rete che determina i percorsi dei pacchetti nel loro viaggio dalla sorgente alla destinazione. Poiché l'instrandamento avviene su scale temporali più grandi, dell'ordine dei secondi, è usualmente implementato in software. A titolo esemplificativo, ripensiamo al viaggio da Milano a Conegliano illustrato nel Paragrafo 1.3.1. In questo caso, l'instrandamento corrisponde al processo svolto dai gestori stradali di pianificazione dei percorsi verso tutte le destinazioni possibili scelti tra i molteplici disponibili e al processo di installazione dei cartelli di direzione in ciascun incrocio. Lungo l'itinerario prestabilito, Giacomo percorre una serie di segmenti di strada e incontra numerosi svincoli il cui attraversamento può essere parago-

**Figura 4.2** Gli algoritmi di instradamento determinano i valori nelle tabelle di inoltro.



nato all'inoltro: l'auto entra nello svincolo, prende la direzione indicata dal cartello e si immette sul tratto di strada che porterà allo svincolo successivo.

Per inoltrare i pacchetti, i router estraggono da uno o più campi dell'intestazione (per esempio il campo Indirizzo di destinazione) i loro valori che utilizzano come indice nella **tabella di inoltro** (*tabella di forwarding* o *forwarding table*), un elemento chiave di qualsiasi router. Il risultato indica a quale interfaccia di uscita il pacchetto debba essere diretto. La Figura 4.2 ne fornisce un esempio: il pacchetto con valore del campo di intestazione pari a 0110 giunge a un router che, tramite la propria tabella di inoltro, individua che l'interfaccia in uscita è la 2 a cui lo inoltra internamente. Nel Paragrafo 4.2 guarderemo all'interno dei router ed esamineremo la funzione di inoltro in maggior dettaglio. L'inoltro è la funzione chiave del piano dei dati del livello di rete.

### Piano di controllo: l'approccio tradizionale

Vi starete indubbiamente chiedendo come vengano configurate le tabelle di inoltro nei router. Questo è un argomento cruciale che evidenzia l'importante interazione tra inoltro (nel piano dei dati) e instradamento (nel piano di controllo). Come mostrato nella Figura 4.2, l'algoritmo di instradamento determina i valori inseriti nelle tabelle di inoltro dei router. In questo esempio l'algoritmo di routing è implementato in ogni router, che quindi svolge sia la funzione di inoltro che quella di instradamento internamente. Vedremo nei Paragrafi 5.3 e 5.4 che le funzioni di instradamento nei router comunicano tra di loro per determinare i valori da inserire nelle tabelle di inoltro. Ma come avvengono tali comunicazioni? Attraverso i messaggi di un protocollo di instradamento. Tratteremo tali algoritmi e protocolli nei Paragrafi da 5.2 a 5.4.

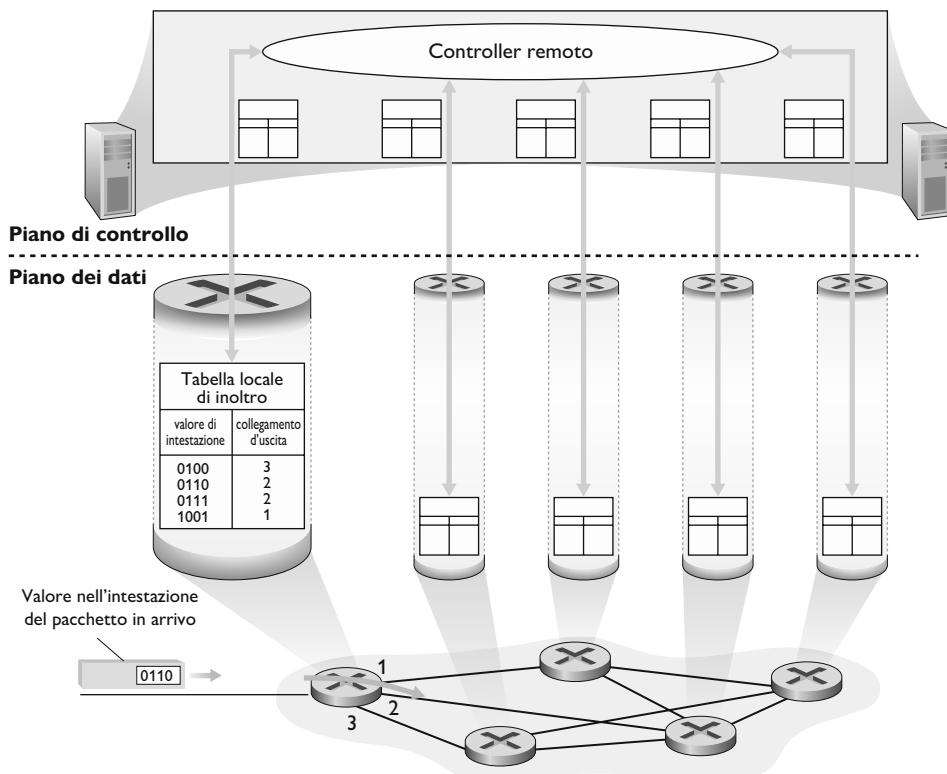
I diversi e distinti scopi delle funzioni di inoltro e di instradamento possono essere ulteriormente illustrati considerando l'ipotetico caso (irrealistico, ma tecnicamente possibile) di una rete in cui tutte le tabelle di inoltro siano configurate

direttamente da operatori di rete, fisicamente presenti presso i router. In questo caso non sarebbe richiesto alcun protocollo di instradamento! Ovviamente, gli operatori dovrebbero interagire tra loro per assicurarsi che le tabelle siano configurate in modo che i pacchetti raggiungano le proprie destinazioni. Probabilmente la configurazione effettuata da operatori umani sarebbe più soggetta a errore e molto più lenta nella risposta ai cambiamenti della topologia di rete rispetto a un protocollo di instradamento. Siamo pertanto fortunati che le reti abbiano sia la funzionalità di inoltro che quella di instradamento!

### Piano di controllo: l'approccio SDN

L'approccio di implementare la funzionalità di instradamento mostrato nella Figura 4.2, in cui ogni router ha una componente di instradamento che comunica con la corrispettiva negli altri router, è stato quello tradizionalmente adottato dalle aziende produttrici di router, almeno fino a poco tempo fa. La nostra osservazione che potremmo configurare manualmente le tabelle di inoltro suggerisce che potrebbero esistere modi alternativi, in cui le funzionalità del piano di controllo determinano le tabelle di inoltro del piano dei dati.

La Figura 4.3 mostra un approccio alternativo in cui un controller remoto, separato fisicamente dai router, calcola e distribuisce le tabelle di inoltro a tutti i router. Si noti che le componenti del piano dei dati nelle Figure 4.2 e 4.3 sono identiche. Tuttavia nella Figura 4.3 la funzionalità di instradamento del piano di controllo è separata fisicamente dal router; il dispositivo di instradamento effettua solo l'inoltro, mentre il controller remoto calcola e distribuisce le ta-



**Figura 4.3**  
Un controller remoto determina e distribuisce i valori delle tabelle di inoltro.

belle di inoltro. Il controller remoto potrebbe essere implementato in un data center remoto con elevata affidabilità e ridondanza e potrebbe essere gestito da un ISP o da una terza parte. Come potrebbero comunicare tra loro i router e il controller remoto? Scambiandosi messaggi contenenti le tabelle di inoltro e altre informazioni di instradamento. Il piano di controllo mostrato nella Figura 4.3 è il cuore del **Software-Defined Networking (SDN)**, nel quale la rete è “software-defined” (gestita da un software) perché il controller che calcola le tabelle di inoltro e interagisce coi router è implementato in software. Inoltre le implementazioni software sono spesso open, come Linux OS, ovvero il codice è disponibile pubblicamente, permettendo così agli ISP (e anche a ricercatori e studenti!) di proporre innovazioni e cambiamenti al software che controlla le funzionalità di rete. Tratteremo il piano di controllo SDN nel Paragrafo 5.5.

#### 4.1.2 Modelli di servizio del livello di rete

Prima di addentrarci nel piano dei dati del livello di rete concludiamo la nostra introduzione adottando una visione più ampia e considerando i diversi tipi di servizi che il livello di rete può offrire. Quando il livello di trasporto sull'host di invio trasmette un pacchetto nella rete può fare affidamento sul livello di rete per il recapito del pacchetto? I pacchetti saranno consegnati nell'ordine in cui sono stati inviati? L'intervallo di tempo tra l'invio di due pacchetti in sequenza sarà uguale a quello di ricezione? La rete offre riscontri sulla congestione? Le risposte a tali domande e ad altre ancora dipendono dal **modello di servizio del livello di rete**, che definisce le caratteristiche del trasporto end-to-end di pacchetti tra host di origine e di destinazione.

Consideriamo ora alcuni servizi che il livello di rete potrebbe offrire:

- **Consegna garantita.** Questo servizio assicura che il pacchetto giunga, prima o poi, alla propria destinazione.
- **Consegna garantita con ritardo limitato.** Questo servizio non solo garantisce la consegna del pacchetto, ma anche il rispetto di un limite di ritardo specificato (per esempio, 100 ms).
- **Consegna ordinata.** Questo servizio garantisce che i pacchetti giungano alla destinazione nell'ordine in cui sono stati inviati.
- **Banda minima garantita.** Questo servizio a livello di rete emula il comportamento di un collegamento trasmissivo con bit rate specificato (per esempio, 1 Mbps) tra host di invio e di destinazione, anche se l'effettivo percorso end-to-end può attraversare diversi collegamenti fisici. Finché l'host di invio trasmette bit (sotto forma di pacchetti) a una velocità inferiore al bit rate specificato, non si verifica perdita di pacchetti.
- **Servizi di sicurezza.** Il livello di rete dell'host sorgente può cifrare tutti i datagrammi inviati; il livello di rete nell'host di destinazione ha il compito di decifrarli. Con questo tipo di servizio, la riservatezza viene fornita a tutti i segmenti del livello di trasporto.

Si tratta, ovviamente, di una lista parziale, in quanto esistono innumerevoli possibili variazioni.

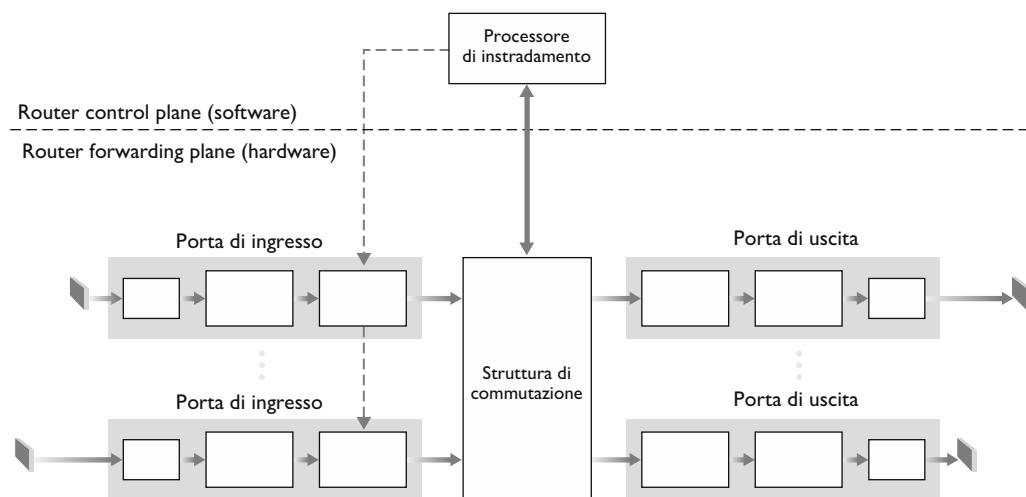
Il livello di rete di Internet mette a disposizione un solo servizio, noto come **servizio best-effort**, ossia “col massimo impegno possibile”. Con questo servizio,

non c'è garanzia che i pacchetti vengano ricevuti nell'ordine in cui sono stati inviati, così come non è garantita la loro eventuale consegna. Non c'è garanzia sul ritardo end-to-end, così come non c'è garanzia su una larghezza di banda minima garantita. Potrebbe sembrare che questa definizione sia in realtà un eufemismo per indicare "nessun servizio". In effetti, una rete che non consegnasse alcun pacchetto alla destinazione soddisfarebbe la definizione di servizio best-effort. Altre architetture di rete hanno definito e implementato modelli di servizi che vanno oltre il servizio best-effort di Internet. Per esempio, l'architettura ATM [Black 1995] offre servizi di consegna ordinata, ritardo limitato e banda minima garantiti. Sono state proposte alcune estensioni del modello di servizio dell'architettura di Internet: per esempio, l'architettura Intserv [RFC 1633] si ripropone di fornire garanzie sul ritardo end-to-end e comunicazioni senza congestione. Ma nonostante lo sviluppo di alternative, il modello di servizio best-effort combinato con un'adeguata larghezza di banda e protocolli a livello di applicazione in grado di adattarsi alla banda, come il protocollo DASH visto nel Paragrafo 2.6.2, si è dimostrato sufficientemente buono da supportare una stupefacente gamma di applicazioni quali i servizi di streaming video come Netflix, le applicazioni di video-over-IP e di conferenza in tempo reale come Skype e Facetime.

## Panoramica del Capitolo 4

Dopo aver fornito una panoramica del livello di rete, vedremo le componenti del piano dei dati del livello di rete nei paragrafi successivi. Nel Paragrafo 4.2 analizzeremo le operazioni hardware interne di un router, quale l'elaborazione dei pacchetti in ingresso e uscita, la struttura di commutazione interna del router, l'accodamento e lo *scheduling* dei pacchetti. Nel Paragrafo 4.3 vedremo l'inoltro IP tradizionale, in cui i pacchetti vengono inoltrati alle porte di uscita in base al loro indirizzo IP di destinazione. Vedremo l'indirizzamento IP, i noti protocolli IPv4 e IPv6 e molto altro. Nel Paragrafo 4.4 vedremo l'inoltro generalizzato, nel quale i pacchetti possono essere inoltrati alle porte di uscita sulla base del valore di più campi dell'intestazione (cioè, non solo in base all'indirizzo IP di destinazione). I pacchetti possono essere bloccati o duplicati nel router e il valore di alcuni campi dell'intestazione può venire riscritto, il tutto sotto il controllo del software. Questa forma più generalizzata di inoltro dei pacchetti è una componente chiave di un moderno piano dei dati, che comprende il piano dei dati nelle *reti software-defined* (SDN). Nel Paragrafo 4.5 vedremo i componenti aggiuntivi di rete comunemente denominati come *middlebox* che implementano funzioni aggiuntive all'inoltro.

Citiamo qui di sfuggita che i termini di *forwarding* e *switching* sono spesso usati in modo intercambiabile dai ricercatori e professionisti delle reti, e così faremo in questo testo. Sempre a proposito di terminologia, vale la pena menzionare altri termini che vengono spesso impiegati in modo intercambiabile, ma che noi utilizzeremo con maggior cautela. Riserveremo il termine **commutatore di pacchetto** (*packet switch* o semplicemente *switch*) per indicare un generico dispositivo che si occupa del trasferimento dei pacchetti da un'interfaccia in ingresso a quella in uscita, in base al valore dei campi nell'intestazione del pacchetto. Alcuni commutatori di pacchetti, chiamati **commutatori a livello di collegamento** (*link-layer switch*), esaminati nel Capitolo 6, stabiliscono l'inoltro in relazione al valore del campo del frame a livello di collegamento (livello 2). Altri, chiamati **router**, prendono le decisioni di inoltro basandosi sul valore nel



**Figura 4.4** Architettura di un router.

campo a livello di rete. I router sono quindi dispositivi a livello di rete (livello 3). Per comprendere l'importanza di questa distinzione, è opportuno rivedere il Paragrafo 1.5.2, in cui abbiamo discusso i datagrammi a livello di rete, i frame a livello di collegamento e le reciproche relazioni. Dato che in questo capitolo ci concentriamo sul livello di rete, useremo il termine *router* anziché *switch*.

## 4.2 Che cosa si trova all'interno di un router?

Ora che abbiamo esaminato il piano dei dati e il piano di controllo all'interno del livello di rete, l'importante distinzione tra inoltro (*forwarding*) e instradamento (*routing*), e i servizi e le funzioni del livello di rete, la nostra attenzione sarà specificamente rivolta alla funzione di inoltro: ossia, alle architetture dei router per trasferire i pacchetti dai collegamenti in ingresso a quelli in uscita.

Osserviamo la Figura 4.4 che presenta una visione ad alto livello di una generica architettura di router in cui si possono identificare quattro componenti.

- **Porte di ingresso** (*input port*). Svolgono le funzioni a livello fisico di terminazione di un collegamento in ingresso al router (riquadri più a sinistra nella porta di ingresso e più a destra nella porta di uscita nella Figura 4.4); svolgono anche funzioni a livello di collegamento (rappresentate dai riquadri centrali delle porte di ingresso e di uscita), necessarie per interoperate con le analoghe funzioni all'altro capo del collegamento di ingresso. Svolgono inoltre la cruciale funzione di ricerca (riquadri più a destra nella porta di ingresso), in modo che il pacchetto inoltrato nella struttura di commutazione del router esca sulla porta di uscita corretta. I pacchetti di controllo, per esempio quelli che trasportano informazioni sul protocollo di instradamento, sono inoltrati dalla porta di ingresso al processore di instradamento. Si noti che il termine “porta” qui si riferisce alle interfacce fisiche di input e output del router; sono quindi completamente distinte dalle porte software associate alle applicazioni di rete e alle socket discusse nei Capitoli 2 e 3. In pratica, il numero di porte supportate da un router può variare da un numero relativamente piccolo nei

router aziendali, a centinaia di porte a 10 Gbps in un router di bordo di un ISP, dove il numero di linee entranti tende a essere estremamente elevato. Il router di bordo Juniper MX2020, per esempio, supporta fino a 800 porte Ethernet a 100 Gbps, con una capacità totale del sistema di router di 800 Tbps [Juniper MX2020 2020].

- **Struttura di commutazione** (*switching fabric*). La struttura di commutazione, che connette fisicamente le porte di ingresso a quelle di uscita, è interamente contenuta all'interno del router: una vera e propria rete in un router di rete!
- **Porte di uscita** (*output port*). Memorizzano i pacchetti che provengono dalla struttura di commutazione e li trasmettono sul collegamento in uscita, operando le funzionalità necessarie del livello di collegamento e fisico. Nei collegamenti bidirezionali, che trasportano traffico in entrambe le direzioni, la porta di uscita verso un collegamento è solitamente accoppiata alla porta di ingresso di quel collegamento sulla stessa scheda di collegamento (detta anche *line card*).
- **Processore di instradamento** (*routing processor*). Esegue le funzioni del piano di controllo. Nei router tradizionali, esegue i protocolli di instradamento (Paragrafi 5.3 e 5.4), gestisce le tabelle di inoltro e le informazioni sui collegamenti attivi, ed elabora la tabella di inoltro per il router. Nei router SDN, il processore di instradamento è responsabile della comunicazione con il controller remoto, in modo da ricevere le occorrenze della tabella di inoltro e installarle alle porte di ingresso. Inoltre effettua le operazioni di gestione di rete che tratteremo nel Paragrafo 5.7.

In un router le porte di ingresso, le porte di uscita e la struttura di commutazione sono implementate quasi sempre in hardware specializzato, secondo uno schema simile a quello mostrato nella Figura 4.4. Per capire perché sia necessaria un'implementazione hardware basta pensare che con un collegamento di input a 100 Gbps e un datagramma IP di 64 byte, la porta di input ha solo 5,12 ns per elaborare il datagramma prima che il successivo possa arrivare. Se (come spesso succede) ci sono  $N$  porte su una line card, la pipeline di elaborazione dei datagrammi deve operare  $N$  volte più velocemente, cosa impossibile per un'implementazione software su hardware tradizionale (*general purpose*). L'hardware specializzato che si occupa dell'inoltro può essere sia proprietario del costruttore del router sia assemblato usando appositi chip di terze parti (come quelli venduti da Intel e Broadcom).

Mentre il piano dei dati opera sulla scala temporale dei nanosecondi, le funzioni di controllo del router, come l'esecuzione dei protocolli di instradamento, la risposta a eventuali malfunzionamenti dei collegamenti comunicanti nel caso di SDN e le funzioni di gestione delle prestazioni, operano sulla scala temporale dei millisecondi o dei secondi. Le funzioni del **piano di controllo** sono solitamente implementate via software ed eseguite sul processore di instradamento (che di solito è un hardware tradizionale basato su architettura CPU).

Prima di entrare nei dettagli riguardanti la struttura di un router e la modalità con cui vengono trasferiti i dati al suo interno, torniamo all'analogia esposta nel Paragrafo 4.1.1, nella quale l'inoltro dei pacchetti era paragonato all'entrata e all'uscita delle auto da uno svincolo. Supponiamo che lo svincolo in questione sia costituito da una rotonda, e che prima che un'auto entri sia richiesto un minimo di elaborazione:

- *Inoltro basato sulla destinazione*: si supponga che un'auto si fermi al casello e indichi la sua destinazione finale (non l'uscita della rotonda locale, ma la destinazione finale del viaggio). Un addetto al casello cerca la destinazione finale, determina l'uscita della rotonda che porta alla destinazione finale e la indica al guidatore.
- *Inoltro generalizzato*: l'addetto potrebbe anche determinare la rampa di uscita della vettura sulla base di molti altri fattori oltre la destinazione; per esempio, potrebbe dipendere dall'origine della vettura, come lo Stato che ha emesso la targa della vettura. Le auto di un certo insieme di stati potrebbero essere indirizzate a usare un'uscita (che porta alla destinazione tramite una strada lenta), mentre le auto provenienti da altri stati potrebbero essere dirette a utilizzare un'uscita diversa (che porta alla destinazione tramite superstrada). La stessa decisione potrebbe essere effettuata sulla base di modello, marca e anno della vettura. O una macchina non ritenuta idonea potrebbe essere bloccata. In caso di inoltro generalizzato, qualsiasi numero di fattori può contribuire alla scelta.

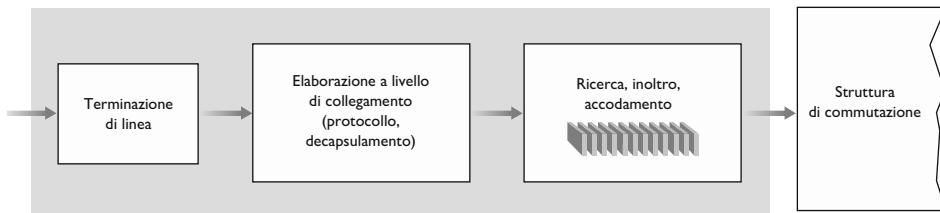
L'auto entra nella rotonda (che può avere altre auto entranti e uscenti) e infine esce sulla rampa di uscita prescritta, dove può incontrare altre auto.

Possiamo riconoscere le principali componenti del router disegnato nella Figura 4.4 in questa analogia: la strada di entrata e il casello corrispondono alle porte di input (con una funzione di ricerca che determina la porta locale di uscita), la rotonda corrisponde alla struttura di commutazione e la strada di uscita dalla rotonda corrisponde alla porta di output. Attraverso questa analogia è interessante capire dove si verificano colli di bottiglia. Che cosa accade se le auto arrivano molto velocemente ma l'addetto è lento? Quanto dovrebbe lavorare velocemente per assicurare che non ci sia coda sulla strada in ingresso? Anche se l'addetto fosse veloce, che cosa accadrebbe se le auto attraversassero la rotonda lentamente; potrebbe ancora verificarsi l'insorgenza di una coda? Che cosa accadrebbe se la maggior parte delle auto in entrata volesse lasciare la rotonda dalla stessa uscita: la coda si formerebbe sulla rampa di uscita o in qualche altro luogo? Come dovrebbe operare la rotonda se volessimo assegnare priorità diverse alle auto o prevenire completamente che alcune auto accedano alla rotonda? Queste sono situazioni critiche analoghe a quelle affrontate dai progettisti di router e switch.

Nei paragrafi seguenti vedremo più in dettaglio le funzioni dei router. [Turner 1988; McKeown 1997a; Partridge 1998; Iyer 2008; Serpanos 2011; Zilberman 2019] forniscono un'analisi di architetture specifiche di router. Per concretezza la discussione seguente si riferisce a decisioni di inoltro basate sull'indirizzo di destinazione del pacchetto; tratteremo l'inoltro generalizzato, basato su più campi di intestazione, nel Paragrafo 4.4.

#### 4.2.1 Elaborazione alle porte di ingresso e inoltro basato sull'indirizzo di destinazione

La Figura 4.5 fornisce una visione dettagliata delle funzionalità delle porte di ingresso. Come già detto, la funzione di terminazione (elettrica) della linea e l'elaborazione a livello di collegamento implementano rispettivamente il livello fisico e di collegamento associati a un singolo collegamento di ingresso al router. L'elaborazione effettuata alla porta di ingresso è centrale per la funzionalità



**Figura 4.5** Elaborazione alle porte di ingresso.

del router: è qui che, utilizzando le informazioni della tabella di inoltro, viene determinata la porta di uscita a cui dirigere un pacchetto attraverso la struttura di commutazione. La tabella di inoltro viene elaborata e aggiornata dal processore di instradamento o ricevuta da un controller SDN remoto. Una sua copia conforme è di solito memorizzata su ciascuna porta di ingresso. La tabella di inoltro viene copiata sulla line card dal processore di instradamento tramite un bus separato (come un bus PCI), indicato nella Figura 4.4 con una linea tratteggiata. Essendoci copie locali della tabella di inoltro, la decisione relativa può essere presa dalle porte di ingresso, senza invocare il processore di instradamento centralizzato.

Consideriamo il caso “più semplice” in cui l’inoltro sia basato sull’indirizzo di destinazione. Supponiamo che tutti gli indirizzi di destinazione siano a 32 bit (dimensione che è la lunghezza dell’indirizzo di destinazione nei datagrammi IPv4). Un’implementazione elementare della tabella di inoltro presenterebbe una riga per ogni possibile indirizzo di destinazione ma, dato che esistono più di 4 miliardi di possibili indirizzi, questa opzione non viene nemmeno presa in considerazione.

Supponiamo inoltre che il nostro router abbia quattro collegamenti, numerati da 0 a 3, e che i pacchetti debbano essere inoltrati verso le interfacce di collegamento come segue:

| <b>Intervallo degli indirizzi di destinazione</b> |          |          |          |          | <b>Interfaccia</b> |
|---------------------------------------------------|----------|----------|----------|----------|--------------------|
| da                                                | 11001000 | 00010111 | 00010000 | 00000000 | 0                  |
| a                                                 | 11001000 | 00010111 | 00010111 | 11111111 |                    |
| da                                                | 11001000 | 00010111 | 00011000 | 00000000 | 1                  |
| a                                                 | 11001000 | 00010111 | 00011000 | 11111111 |                    |
| da                                                | 11001000 | 00010111 | 00011001 | 00000000 | 2                  |
| a                                                 | 11001000 | 00010111 | 00011111 | 11111111 |                    |
| altrimenti                                        |          |          |          |          | 3                  |

Chiaramente, in questo esempio, non è necessario avere 4 miliardi di righe nelle tabelle di inoltro dei router. Potremmo, per esempio, avere la seguente tabella, con appena quattro voci:

| <b>Corrispondenza di prefisso</b> | <b>Interfaccia</b> |
|-----------------------------------|--------------------|
| 11001000 00010111 00010           | 0                  |
| 11001000 00010111 00011000        | 1                  |
| 11001000 00010111 00011           | 2                  |
| altrimenti                        | 3                  |

Con questa struttura il router confronta un **prefisso** dell’indirizzo di destinazione del pacchetto con una riga della tabella; se c’è corrispondenza, il router inoltra il pacchetto al collegamento associato. Per esempio, supponiamo che l’indirizzo di destinazione del pacchetto sia 11001000 00010111 00010110 10100001; dato che il prefisso a 21 bit di questo indirizzo rispecchia la prima riga nella tabella, il router inoltra il pacchetto all’interfaccia di collegamento 0. Se un prefisso non corrisponde alle prime tre righe, il router lo inoltra all’interfaccia 3. Sebbene tutto questo sembri sufficientemente semplice, in realtà nasconde un accorgimento ingegnoso. Avrete notato che un indirizzo di destinazione può corrispondere a più di una riga. Per esempio, i primi 24 bit dell’indirizzo 11001000 00010111 00011000 10101010 corrispondono alla seconda riga della tabella e i primi 21 alla terza riga della tabella. Quando si verificano corrispondenze multiple, il router adotta la **regola di corrispondenza a prefisso più lungo**; in altre parole, viene determinata la corrispondenza più lunga all’interno della tabella e i pacchetti vengono inoltrati all’interfaccia di collegamento associata. Ne vedremo il funzionamento esatto nel Paragrafo 4.3, che tratta l’indirizzamento in Internet più dettagliatamente.

La ricerca nella tabella di inoltro è concettualmente semplice e viene effettuata cercando il più lungo prefisso corrispondente. A tassi di trasmissione di Gigabit (*transmission rate*), tale ricerca deve tuttavia essere effettuata in nanosecondi (ricordiamoci il nostro esempio precedente di un collegamento a 10 Gbps e un datagramma IP a 64 byte). Quindi, non solo la ricerca va effettuata in hardware, ma sono necessarie tecniche che vadano oltre la semplice ricerca lineare su grandi tabelle. Una trattazione degli algoritmi di ricerca veloci può essere trovata in [Gupta 2001; Ruiz-Sanchez 2001]. Bisogna anche porre una particolare attenzione ai tempi di accesso alla memoria; si ricorre a memorie DRAM integrate direttamente nei processori e SRAM molto veloci (usate come cache per le DRAM). Vengono spesso usate per la ricerca anche le *Ternary Content Address Memories* (TCAM) [Yu 2004]. Con una TCAM, un indirizzo IP a 32 bit è passato alla memoria che restituisce il contenuto della tupla nella tabella di inoltro corrispondente a quell’indirizzo in un tempo essenzialmente costante. I router Cisco Catalyst 6500 e 7600 arrivano a tabelle di inoltro con milioni di occorrenze [Cisco TCAM 2014].

Una volta determinata la porta di output di un pacchetto, esso può essere inviato alla struttura di commutazione. In alcune architetture di router un pacchetto può essere temporaneamente fermato prima di entrare nella struttura di commutazione quando essa è utilizzata da altre porte di input. Un pacchetto bloccato verrà accodato sulla porta di input e quindi schedulato per il passaggio alla struttura di commutazione più tardi. Studieremo più in dettaglio il blocco, l’accodamento e lo scheduling dei pacchetti, sia sulle porte di ingresso sia di uscita. Sebbene la ricerca sia l’azione più importante dell’elaborazione alle porte di input, ve ne sono molte altre: (1) elaborazione a livello fisico e di collegamento, (2) il numero di versione del pacchetto, il checksum e il tempo di vita, che studieremo nel Paragrafo 4.3, devono essere controllati e gli ultimi due campi anche riscritti; (3) i contatori usati per la gestione di rete (come il numero di datagrammi IP ricevuti) vanno aggiornati.

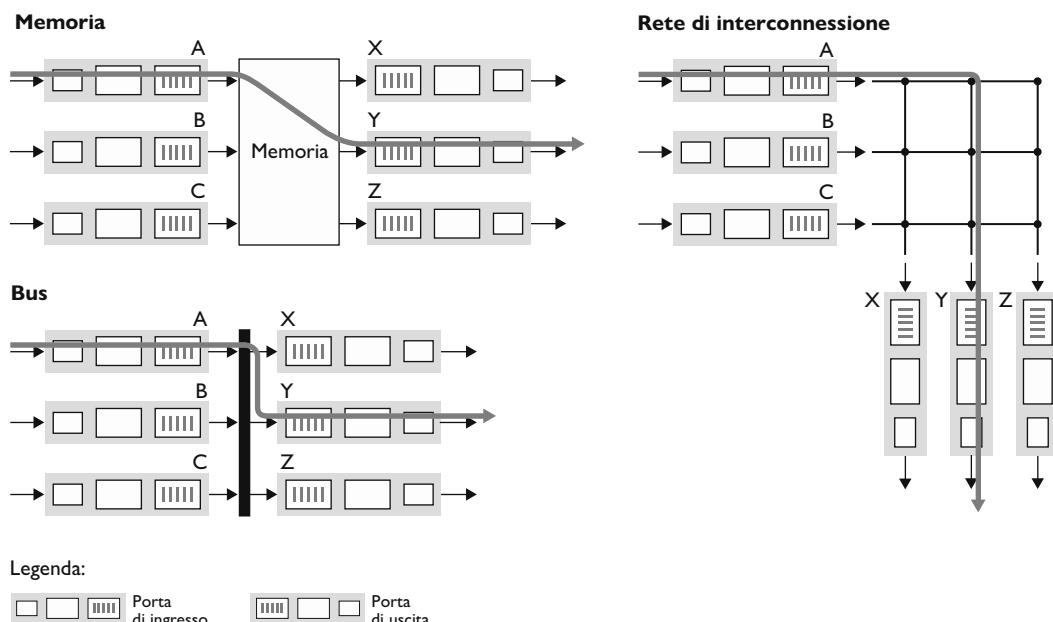
Chiudiamo la nostra discussione sull’elaborazione alle porte di ingresso notando che l’azione di cercare la corrispondenza tra l’indirizzo IP di destinazione (*match* – “confronto”, “corrispondenza”) per poi inviare il pacchetto alla porta

di uscita specificata attraverso la struttura di commutazione (*action* – “azione”) è un caso specifico di un’astrazione più generale “match-action” che viene eseguita in molti dispositivi di rete, non solo nei router. Negli switch di livello 2 (Capitolo 6), diverse azioni possono essere effettuate in aggiunta all’invio del frame nella struttura di commutazione verso la porta di uscita. Nei firewall (Capitolo 8, disponibile in inglese sulla piattaforma MyLab), dispositivi che filtrano i pacchetti in entrata, un pacchetto la cui intestazione corrisponde a un dato criterio (per esempio, una combinazione di indirizzi IP e numeri di porta di sorgente e destinazione) può essere scartato (azione). In un traduttore di indirizzi di rete (NAT, Paragrafo 4.3), un pacchetto in arrivo, il cui numero di porta a livello di trasporto corrisponde a un dato valore, vedrà il suo numero di porta riscritto prima di essere inoltrato (azione). In verità, l’astrazione “match-action” è molto potente e adottata negli odierni dispositivi di rete, ed è centrale alla nozione di inoltro generalizzato che studieremo nel Paragrafo 4.4.

### 4.2.2 Struttura di commutazione (switching)

La struttura di commutazione (*switching fabric*) rappresenta il vero e proprio cuore dei router, attraverso il quale i pacchetti vengono commutati (ossia inoltrati) dalla porta di ingresso alla porta di uscita. La commutazione può essere ottenuta in vari modi (Figura 4.6).

- **Commutazione in memoria.** I primi e più semplici router erano in genere calcolatori tradizionali, e la commutazione tra porte di ingresso e di uscita veniva effettuata sotto il controllo diretto della CPU (processore di instradamento). Le porte di ingresso e di uscita funzionavano come tradizionali dispositivi di I/O. Quando sopraggiungeva un pacchetto, la porta di ingresso ne segnalava l’arrivo tramite interrupt e quindi lo copiava nella memoria del



**Figura 4.6** Tre tecniche di commutazione.

processore di instradamento che procedeva a estrarre dall'intestazione l'indirizzo di destinazione. Quindi, individuava tramite la tabella di inoltro l'appropriata porta di uscita nel cui buffer copiava il pacchetto. Notiamo che se l'ampiezza di banda della memoria è tale da potervi scrivere o leggere  $B$  pacchetti al secondo, allora il throughput complessivo di inoltro (ossia la velocità massima alla quale i pacchetti vengono trasferiti dalle porte di ingresso a quelle di uscita) è necessariamente inferiore a  $B/2$ . Notiamo inoltre che due pacchetti non possono essere inoltrati contemporaneamente, anche se hanno differenti porte di destinazione, perché può essere effettuata solo un'operazione alla volta di scrittura/lettura in memoria tramite il bus di sistema.

Anche alcuni router attuali effettuano la commutazione in memoria. Esiste però una differenza sostanziale rispetto ai primi router: la ricerca dell'indirizzo di destinazione e la memorizzazione del pacchetto nella locazione di memoria opportuna vengono effettuate dai processori direttamente sulle line card di ingresso. In alcuni casi, i router che effettuano la commutazione in memoria assomigliano molto a sistemi multiprocessore a memoria condivisa dove l'elaborazione su una line card scrive direttamente i pacchetti nella memoria della porta di uscita appropriata. Gli switch Cisco Catalyst serie 8500 [Cisco 8500 2020] commutano i pacchetti tramite una memoria condivisa.

- **Commutazione tramite bus.** In questo approccio le porte di ingresso trasferiscono un pacchetto direttamente alle porte di uscita tramite un bus condiviso e senza intervento da parte del processore di instradamento. Questo viene tipicamente fatto aggiungendo un'etichetta interna di commutazione (intestazione) al pacchetto che indica la porta locale di output alla quale il pacchetto deve essere trasferito quando viene trasmesso sul bus. Il pacchetto viene ricevuto da tutte le porte di output, ma solo la porta corrispondente all'etichetta lo raccoglierà. L'etichetta viene quindi rimossa dalla porta di output in quanto usata solo all'interno della struttura di commutazione per attraversare il bus. Se più pacchetti arrivano contemporaneamente al router, ognuno su una porta di input diversa, tutti tranne uno dovranno aspettare, dato che sul bus si può trasferire soltanto un pacchetto alla volta. Poiché ciascun pacchetto deve attraversare il bus, la larghezza di banda della commutazione è limitata da quella del bus. Nella nostra analogia con il traffico alle rotonde stradali, ciò equivale a permettere che solo una macchina alla volta possa impegnare la rotonda. Questo tipo di commutazione tramite bus è comunque spesso sufficiente per router che operano in reti di accesso e in quelle aziendali. Gli switch Cisco della serie 6500 [Cisco 6500 2020] commutano pacchetti su un bus a 32 Gbps.
- **Commutazione attraverso rete di interconnessione.** Un modo per superare la limitazione di banda di un singolo bus condiviso è l'utilizzo di una rete di interconnessione più sofisticata, quale quella usata in passato nelle architetture multiprocessore. Una matrice di commutazione (*crossbar switch*) è una rete di interconnessione che consiste di  $2n$  bus che collegano  $n$  porte di ingresso a  $n$  porte di uscita (Figura 4.6). Ogni bus verticale interseca tutti i bus orizzontali a un punto di incrocio che può essere in qualsiasi momento aperto o chiuso dal controller della struttura di commutazione (la cui logica è parte stessa della struttura). Quando un pacchetto giunge a una porta di ingresso A e deve essere inoltrato alla porta Y, il controller chiude l'incrocio di A e Y e la porta A invia il pacchetto sul suo bus e solo il bus Y lo riceverà. Si noti

che un pacchetto dalla porta B può essere inoltrato a X nello stesso tempo, perché i pacchetti A-Y e B-X usano bus di input e output diversi. Quindi, al contrario degli altri due approcci, la commutazione attraverso rete di interconnessione è in grado di inoltrare più pacchetti in parallelo. Una matrice di commutazione è **non-blocking**: un pacchetto in via di inoltro verso una porta di uscita non viene bloccato a meno che esista un altro pacchetto in via di inoltro sulla stessa porta di uscita. Tuttavia, se due pacchetti provenienti da due diverse porte di input sono destinati alla stessa porta di output, uno dovrà accodarsi alla porta di input, perché solo un pacchetto alla volta può essere inoltrato su uno specifico bus. La serie di switch Cisco 12000 [Cisco 12000 2020] usa la commutazione attraverso rete di interconnessione; la serie Cisco 7600 [Cisco 7600 2020] può essere configurata per utilizzare sia un bus sia una matrice di commutazione.

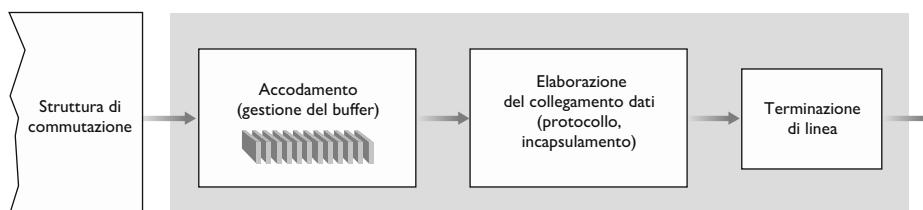
Reti di interconnessione più sofisticate usano elementi di commutazione a più stadi per permettere che pacchetti provenienti da porte di input diverse, ma con la stessa porta di output, possano attraversare la struttura di commutazione contemporaneamente. Si veda [Tobagi 1990] per un saggio sulle architetture di commutazione. I router Cisco CRS utilizzano una strategia di commutazione non bloccante a tre stadi. La capacità di commutazione di un router può scalare utilizzando un numero  $N$  di strutture di commutazione in parallelo. La porta di ingresso suddivide il pacchetto in  $K$  pezzi (*chunk*) più piccoli, e li invia (*spray*) attraverso  $K$  di queste  $N$  strutture di commutazione alla porta di uscita selezionata, che li riassemblerà nel pacchetto originale.

### 4.2.3 Elaborazione alle porte di uscita

L'elaborazione alle porte di uscita (Figura 4.7) prende i pacchetti dalla memoria della porta di uscita e li trasmette sul collegamento di uscita. Tale operazione comprende selezionare e togliere dalla coda i pacchetti per la trasmissione e operare le necessarie funzioni a livello di collegamento e fisico.

### 4.2.4 Dove si verifica l'accodamento?

È evidente che si possono formare code di pacchetti sia presso le porte di ingresso sia presso quelle di uscita (Figura 4.6), come i casi identificati nell'analogia delle rotonde, nelle quali le auto possono essere in coda sia agli ingressi sia alle uscite. Il luogo e la lunghezza della coda (sia alle porte di input che di output) dipendono dalla quantità di traffico di rete, dalle velocità relative della struttura di commutazione e della linea. Quando queste code crescono, la memoria del router può esaurirsi e quindi può avvenire una **perdita di pacchetti** nel caso non vi sia più memoria per immagazzinare quelli in arrivo. Nelle di-



**Figura 4.7** Elaborazione alle porte di uscita.

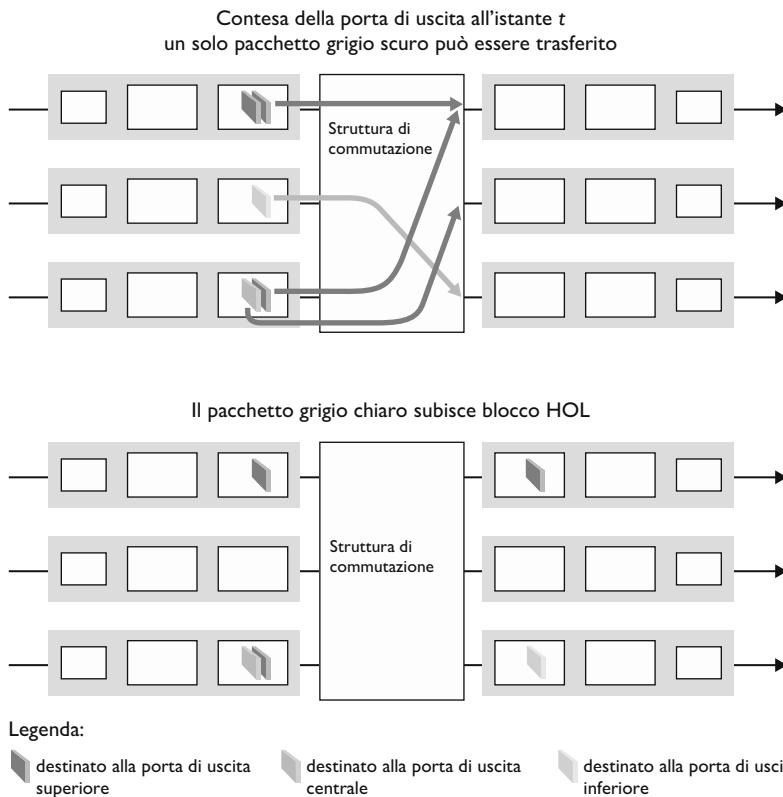
scussioni precedenti abbiamo detto che i pacchetti venivano “persi nella rete” o “scartati dai router”. *È qui, a queste code all'interno dei router, che tali pacchetti vengono effettivamente scartati e persi.*

Supponiamo che le velocità di linea di input e output abbiano tutte lo stesso tasso di trasmissione di  $R_{line}$  pacchetti al secondo e che ci siano  $N$  porte di input e  $N$  porte di output. Supponiamo inoltre, per semplicità, che tutti i pacchetti abbiano la stessa lunghezza e arrivino alle porte di input in modo sincrono, vale a dire che il tempo di invio di un pacchetto su qualsiasi collegamento sia pari al tempo di ricezione di un pacchetto e che in tale intervallo di tempo possa arrivare su un collegamento di input un pacchetto oppure nessun pacchetto. Definiamo il tasso di trasferimento della struttura di commutazione  $R_{switch}$  come il tasso al quale i pacchetti vengono trasferiti dalla porta di input a quella di output. Se  $R_{switch}$  è  $N$  volte più veloce di  $R_{line}$ , l'accodamento alle porte di input è trascurabile perché, anche nel caso peggiore in cui i pacchetti arrivino su tutte le  $N$  linee di input e debbano essere inoltrati alla medesima porta di output, ogni gruppo di  $N$  pacchetti, uno per porta di input, può essere elaborato dalla struttura di commutazione prima dell'arrivo del successivo.

### Accodamento in ingresso

Ma che cosa avviene se la struttura di commutazione non è sufficientemente rapida (rispetto alle linee in ingresso) nel trasferire *tutti* i pacchetti in arrivo senza ritardo? In questo caso può verificarsi accodamento anche alle porte di ingresso. Per illustrare un'importante conseguenza di tale accodamento, consideriamo una struttura di commutazione facente uso di una rete di interconnessione e supponiamo che (1) le velocità di tutti i collegamenti siano identiche, (2) un pacchetto possa essere trasferito da qualsiasi porta di ingresso a una data porta di uscita nello stesso intervallo di tempo richiesto perché un pacchetto sia ricevuto su un collegamento di ingresso e (3) che i pacchetti vengano trasferiti da una coda di ingresso alla relativa coda di uscita desiderata con procedura FCFS (*first-come-first-served*, primo arrivato primo servito). È possibile trasferire più pacchetti in parallelo quando le relative porte di uscita sono differenti. Tuttavia, se due pacchetti in testa a due code di ingresso sono destinati alla stessa coda di uscita, allora uno dei pacchetti sarà bloccato e dovrà attendere: istante per istante, la struttura di commutazione può trasferire solo un pacchetto a una certa porta di uscita.

La Figura 4.8 mostra un esempio in cui due pacchetti (contraddistinti in grigio scuro) in testa alle rispettive code di ingresso sono destinati alla stessa porta di uscita, in alto a destra. Supponiamo che la struttura di commutazione scelga di trasferire il primo pacchetto della coda in alto a sinistra. In questo caso, il pacchetto nella coda in basso a sinistra deve attendere, così come quello grigio più chiaro, che si trova dietro di lui, anche se non si verifica contesa per la porta di uscita centrale (che rappresenta la destinazione del pacchetto di colore chiaro). Questo fenomeno è noto come **blocco in testa alla coda (HOL, Head-Of-The-Line blocking)**: un pacchetto nella coda di ingresso deve attendere il trasferimento attraverso la struttura (anche se la propria porta di destinazione è libera) in quanto risulta bloccato da un altro pacchetto che lo precede. [Karol 1987] mostra che, sotto determinate ipotesi, a causa del blocco la coda di ingresso cresce indefinitamente (e quindi si verificheranno sostanziose perdite di pacchetti) quando il tasso di arrivo dei pacchetti sui collegamenti di ingresso



**Figura 4.8** Blocco in testa (HOL) alla coda di input di uno switch.

raggiunge solo il 58% della loro capacità. Numerose soluzioni al problema sono trattate in [McKeown 1997].

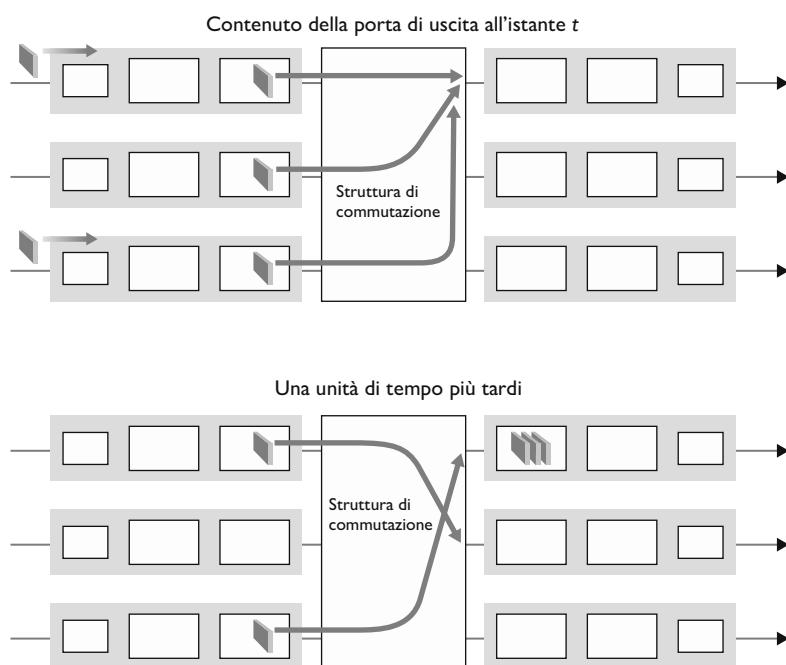
### Accodamento in uscita

Ma che cosa avviene alle porte di uscita? Supponiamo ancora che  $R_{switch}$  sia  $N$  volte più rapida di  $R_{line}$  e che i pacchetti che giungono a ciascuna delle  $N$  porte di ingresso siano destinati alla stessa porta di uscita; quindi, nel tempo richiesto per ricevere (o inviare) un singolo pacchetto, a questa porta di uscita arrivano  $N$  nuovi pacchetti (uno da ogni  $N$  porta di ingresso). Dato che la porta di uscita può trasmettere un solo pacchetto in un intervallo prestabilito (tempo di trasmissione del pacchetto), gli  $N$  pacchetti in arrivo dovranno mettersi in coda per la trasmissione sul collegamento in uscita. Di conseguenza, è possibile che giungano altri  $N$  pacchetti nel periodo necessario per trasmettere solo uno degli  $N$  pacchetti già accodati, e così via. Quindi è possibile che si formino code di pacchetti alle porte di uscita anche quando la struttura di commutazione è  $N$  volte più rapida delle velocità di linea delle porte. Il numero di pacchetti in coda può continuare a crescere fino a esaurire lo spazio di memoria sulla porta di uscita.

In assenza di sufficiente memoria per inserire nel buffer il nuovo pacchetto in ingresso, occorre stabilire se scartarlo (politica nota come **drop-tail**, eliminazione in coda) o se rimuoverne uno o più, fra quelli già in coda, per far posto al nuovo arrivato. In alcuni casi può risultare vantaggioso eliminare un pacchetto (o marcarne l'intestazione) prima che il buffer sia pieno, al fine di fornire al

**Figura 4.9**

Accodamento alle porte di uscita.



mittente un segnale di congestione. Tale operazione potrebbe essere effettuata utilizzando i bit di Notifica Esplicita di Congestione che abbiamo trattato nel Paragrafo 3.7.2. Sono state proposte e analizzate numerose politiche di eliminazione e marcatura dei pacchetti, tecniche che presentano il nome collettivo di algoritmi di **AQM** (**Active Queue Management**, *gestione attiva della coda*) [Labrador 1999; Hollot 2002]. Tra questi uno dei più ampiamente studiati e implementati è detto algoritmo **RED** (**Random Early Detection**) [Christiansen 2001]. Più recenti policy AQM includono PIE (*Proportional Integral controller Enhanced* [RFC 8033]) e CoDel [Nichols 2012]. Si osservi la Figura 4.9, in cui è esemplificato l'accodamento alla porta di uscita. All'istante  $t$ , a ciascuna delle porte di ingresso giunge un pacchetto destinato alla porta in uscita superiore. Ipotizzando velocità di linea identiche e un commutatore che opera al triplo della velocità di linea, nell'unità di tempo dopo, cioè dopo il tempo richiesto per ricevere o inviare un pacchetto, i tre pacchetti originali sono stati trasferiti sulla porta in uscita e sono accodati in attesa di trasmissione. Nell'unità di tempo successiva verrà trasmesso uno di questi tre pacchetti. Nel nostro esempio, due *nuovi* pacchetti sono giunti al commutatore; uno di questi pacchetti è destinato alla porta di uscita superiore. Quando vi sono più pacchetti accodati sulle porte di uscita, uno **schedulatore di pacchetti** (*packet scheduler*) deve stabilire in quale ordine trasmetterli.

### Quanta memoria di buffer è necessaria?

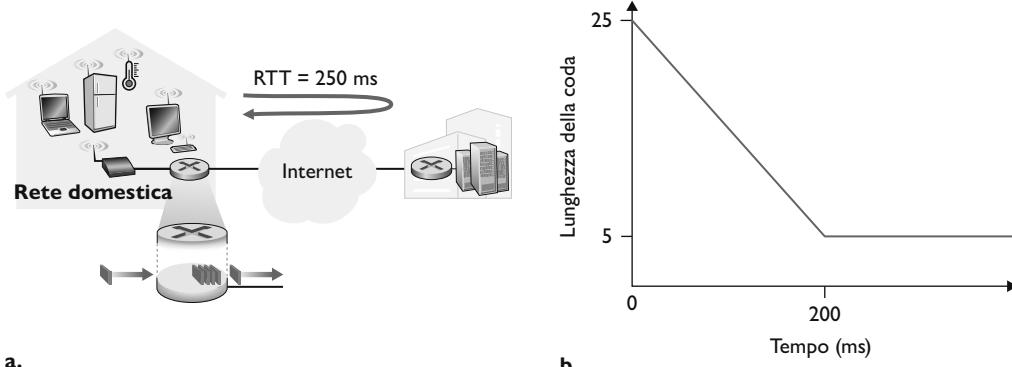
Abbiamo prima mostrato che una coda di pacchetti si forma quando gruppi (*burst*) di pacchetti arrivano alla porta d'ingresso del router, o più probabilmente alla porta d'uscita, e il tasso di arrivo dei pacchetti eccede temporaneamente il tasso con cui i pacchetti vengono inoltrati. Più è lungo il periodo di

tempo in cui tale situazione persiste, più si allunga la coda finché i buffer di una porta diventano pieni e i pacchetti vengono persi. Sorge naturale chiedersi quanto debba essere grande il buffer di una porta. La risposta a questa domanda è molto più complicata di quanto si immagini e insegna molto sulla sottile interazione che esiste tra i mittenti consapevoli della congestione nell'edge della rete e il core della rete.

Per molti anni la regola comunemente accettata per il buffer [RFC 3439] affermava che la grandezza del buffer (**B**) dovesse essere uguale al tempo di round-trip medio (**RTT**) moltiplicato per la capacità del link (**C**); quindi un collegamento a 10 Gbps con un RTT di 250 ms necessiterebbe di una grandezza di buffer pari a  $\mathbf{B} = \mathbf{RTT} * \mathbf{C} = 2,5$  Gbit di buffer. Tale risultato si basa su un'analisi della dinamica di accodamento di un numero di flussi TCP relativamente piccolo [Villamizar 1994]. Recenti studi teorici e sperimentali [Appenzeller 2004], tuttavia, suggeriscono che quando c'è un gran numero di flussi TCP indipendenti (**N**) che passano attraverso un collegamento, la quantità di buffer necessaria sia  $\mathbf{B} = \mathbf{RTT} \times \mathbf{C}/\sqrt{\mathbf{N}}$ . Con un gran numero di flussi, che tipicamente attraversano i grossi collegamenti dei router di dorsale, il valore di *N* può essere grande e la diminuzione nella quantità di buffer necessaria diventa abbastanza significativa. [Appenzeller 2004, Wischik 2005 e Beheshti 2008] forniscono una trattazione molto interessante sul problema del dimensionamento del buffer da un punto di vista teorico, implementativo e operativo.

Si è tentati di pensare che più buffering si ha, meglio è: buffer più grandi consentono a un router di assorbire fluttuazioni maggiori del tasso di arrivo dei pacchetti, riducendo così il tasso di perdita di pacchetti del router. Ma buffer più grandi significano anche, potenzialmente, ritardi di coda più lunghi. Per i giocatori e per gli utenti di teleconferenze interattive, decine di millisecondi contano. Aumentare la quantità di buffer per hop di un fattore 10 per diminuire la perdita di pacchetti potrebbe far aumentare il ritardo end-to-end di un fattore 10! L'incremento degli RTT rende i mittenti TCP meno reattivi e più lenti a rispondere alla congestione incipiente e/o alla perdita di pacchetti. Queste considerazioni basate sul ritardo mostrano che il buffering è un'arma a doppio taglio: il buffering può essere utilizzato per assorbire fluttuazioni statistiche a breve termine nel traffico, ma può anche portare a un aumento del ritardo e alle sue conseguenze. Il buffering è un po' come il sale: la giusta quantità di sale rende il cibo migliore, ma troppo lo rende immangiabile!

Nella discussione fatta finora abbiamo implicitamente assunto che molti mittenti indipendenti siano in competizione per la larghezza di banda e per i buffer su un collegamento. Tale assunzione va bene per i router all'interno del core della rete, ma non è più valida all'edge della rete. La Figura 4.10(a) mostra un router residenziale che invia segmenti TCP a un server remoto di gioco. Come in [Nichols 2012] si supponga che il server impieghi 20 ms per trasmettere un pacchetto contenente un segmento TCP del giocatore, che non ci siano ritardi di coda significativi sul percorso verso il server di gioco e che l'RTT sia di 200 ms. Come mostrato nella Figura 4.10(b) si supponga che al tempo  $t = 0$  un burst (gruppo, raffica) di 25 pacchetti arrivi alla coda; ogni 20 ms uno di questi pacchetti in coda viene quindi trasmesso in modo che al tempo  $t = 200$  ms arrivi il primo ACK, giusto quando il ventunesimo pacchetto viene trasmesso. L'arrivo di questo ACK fa in modo che il mittente TCP invii un altro pacchetto che viene messo in coda sul collegamento in uscita del router residenziale. Al tempo

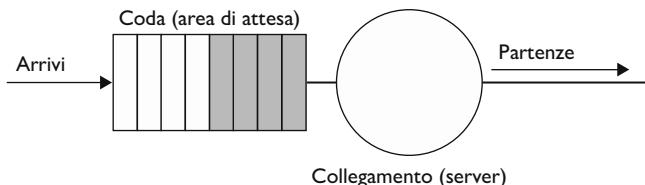


**Figura 4.10** Bufferbloat: code persistenti.

$t = 220$  arriva l'ACK seguente, un altro segmento TCP viene rilasciato dal giocatore e viene messo in coda quando il 22esimo pacchetto viene trasmesso, e così via. Dovreste convincervi che in questo scenario le tempistiche degli ACK implicano che un nuovo pacchetto arrivi alla coda ogni volta che viene trasmesso un pacchetto; il risultato è che la grandezza della coda sul link di uscita del router residenziale è *sempre* uguale a 5 pacchetti! Quindi la catena trasmissiva da estremo a estremo (che invia pacchetti alla destinazione sul percorso con collo di bottiglia di capacità di un pacchetto ogni 20 ms), è piena, ma il ritardo di coda è costante e *persistente*. Il risultato è che il giocatore è scontento del ritardo e il genitore (che pure conosce Wireshark!) è confuso, perché non capisce per quale motivo i ritardi siano persistenti ed eccessivamente lunghi in assenza di altro traffico sulla rete domestica. Lo scenario fin qui descritto di lunghi ritardi dovuti a un processo di buffer persistente è noto come **bufferbloat** e mostra come non solo il *throughput* (capacità complessiva) sia importante, ma anche come lo sia ogni minimo ritardo [Kleinrock 2018] e che l'interazione tra mittenti all'edge della rete e le code all'interno della rete siano in effetti complesse e sottili. Lo standard DOCSIS 3.1 per le reti cablate che tratteremo nel Capitolo 6 ha recentemente aggiunto un meccanismo AQM specifico [RFC 8033, RFC 8034] per contrastare il bufferbloat, preservando le performance di throughput.

#### 4.2.5 Schedulazione dei pacchetti

Torniamo ora alla domanda su come determinare l'ordine con cui i pacchetti vengono trasmessi sulla porta di uscita. Poiché indubbiamente vi sarà capitato spesso di avere a che fare con le code nella vostra vita quotidiana e di osservare come vengono gestite, troverete familiari molte delle procedure di coda comunemente usate nei router. Esiste la procedura FCFS (*first-come-first-served*, primo-arrivato-primo-servito), anche nota come FIFO (*first-in-first-out*). Gli inglesi sono famosi per stare pazientemente in code FCFS alla fermata dei mezzi o al supermercato. In altri paesi si opera seguendo delle priorità per cui alcuni clienti vengono serviti prima di altri. Infine ci sono le code round-robin, in cui i clienti vengono divisi come prima in classi che però vengono servite a rotazione.



**Figura 4.11**  
Un modello di coda FIFO.

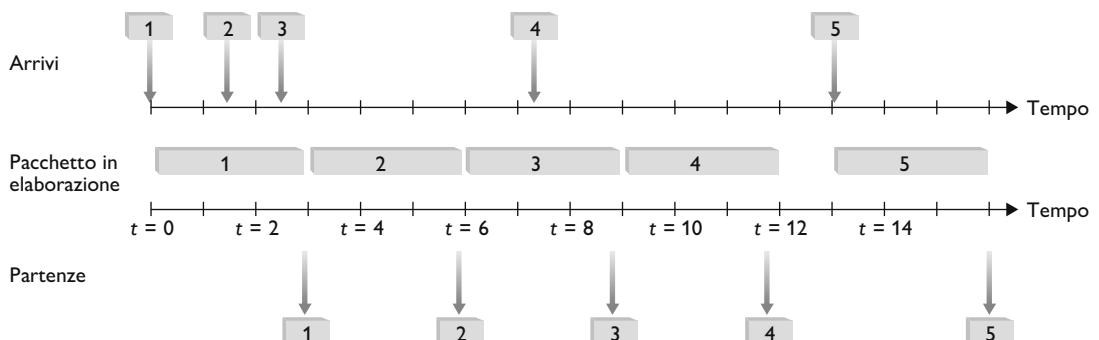
### First-In-First-Out (FIFO)

La Figura 4.11 mostra un modello di coda FIFO: “primo-arrivato-primo-servito”. I pacchetti che arrivano alla coda di uscita del collegamento aspettano di essere trasmessi se quest’ultimo è occupato nella trasmissione di un altro pacchetto. Se non c’è sufficiente spazio nel buffer per contenere il pacchetto che arriva, la politica di scarto dei pacchetti (*packet-dropping policy*) determina se il pacchetto va eliminato (e quindi perso) o se bisogna rimuovere dei pacchetti dalla coda per far spazio al pacchetto in arrivo. Nella parte successiva della nostra trattazione non terremo conto delle politiche di scarto e supporremo che il pacchetto venga rimosso solo dopo che è stato trasmesso.

I pacchetti vengono quindi trasmessi nello stesso ordine con cui sono arrivati in coda. Un esempio familiare sono le file nei centri servizi, dove i clienti che arrivano si mettono in coda di attesa, rimangono nell’ordine di arrivo e poi vengono serviti quando arriva il loro turno. La Figura 4.12 mostra un modello di coda FIFO. Le frecce numerate in alto indicano l’ordine di arrivo dei pacchetti, con il numero che indica l’ordine con cui il pacchetto è arrivato, quelle in basso mostrano la sequenza con cui i pacchetti sono trasmessi. L’intervallo durante il quale un pacchetto viene trasmesso (riceve il servizio) è contrassegnato dal rettangolo ombreggiato fra le due linee orizzontali. Dato che lo scheduling segue la strategia FIFO, cioè i pacchetti vengono inviati nello stesso ordine in cui sono arrivati, dopo la partenza del quarto pacchetto il collegamento resta inattivo fino all’arrivo del quinto.

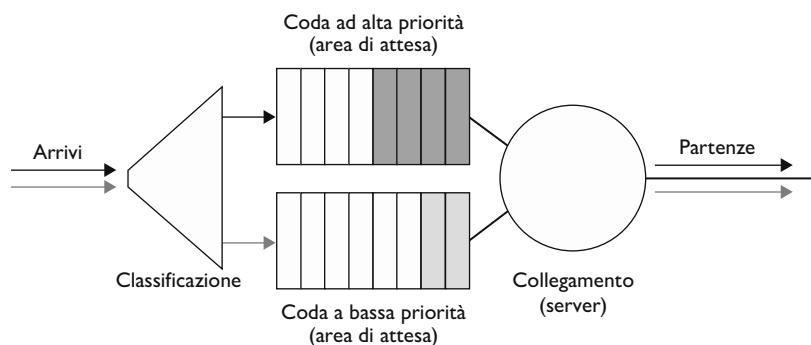
### Coda con priorità

Nel modello di accodamento con priorità (*priority queuing*) i pacchetti sono classificati in base a classi di priorità, come mostrato nella Figura 4.13. In pratica, un operatore di rete può configurare una coda in modo che i pacchetti con



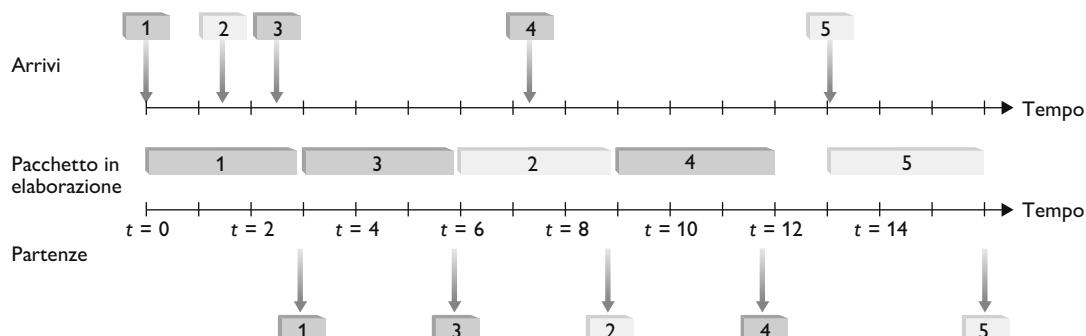
**Figura 4.12** Coda FIFO in azione.

**Figura 4.13** Coda con priorità.



informazioni di gestione di rete (identificati per esempio tramite il numero di porta UDP/TCP sorgente o di destinazione) abbiano la priorità rispetto al traffico utenti; inoltre i pacchetti che trasportano dati di applicazioni voice-over-IP in tempo reale possono ricevere priorità rispetto al traffico non in tempo reale, quale quello dei pacchetti SMTP o IMAP. Ciascuna classe di priorità ha la propria coda. La modalità di accodamento con priorità decide quindi di trasmettere i pacchetti non più nell'ordine generale di arrivo, ma selezionando di volta in volta il pacchetto dalla coda non vuota con priorità più alta. La scelta fra i pacchetti di una classe è di solito effettuata seguendo la strategia FIFO.

La Figura 4.14 mostra una coda con due classi di priorità; alla più alta appartengono i pacchetti 1, 3 e 4, mentre 2 e 5 appartengono alla classe con bassa priorità. Quando giunge il pacchetto 1, il collegamento è libero e questo inizia subito a essere trasmesso. Nel frattempo, pervengono i pacchetti 2 e 3 che sono posti nelle rispettive code. Dopo il primo invio viene selezionato il pacchetto 3 che, nonostante sia arrivato dopo, ha una classe di priorità più alta di 2. Una volta terminata la trasmissione del pacchetto 3 il pacchetto 2 inizia a essere trasmesso e durante la trasmissione sopraggiunge il pacchetto 4 ad alta priorità. Nella modalità di **accodamento a priorità senza prelazione** (*non-preemptive priority queuing*), la trasmissione dei pacchetti non può essere interrotta una volta iniziata. Pertanto il pacchetto 4 viene messo in coda e potrà essere inviato solo una volta terminata la trasmissione del pacchetto 2.



**Figura 4.14** Operazioni di una coda con priorità.

**BOX 4.1****TEORIA E PRATICA****Neutralità della rete**

Abbiamo visto che i meccanismi di scheduling dei pacchetti possono essere utilizzati per fornire livelli di servizio diversi a differenti “classi” di traffico. Sta all'ISP decidere che cosa costituisce precisamente una “classe” di traffico, ma potenzialmente potrebbe essere basata su qualsiasi insieme di campi nell'intestazione del datagramma IP; per esempio, il campo della porta nell'intestazione del datagramma IP potrebbe essere utilizzato per classificare i datagrammi secondo il “well-known service” associato a quella porta oppure il datagramma di gestione (porta 161) in una rete SNMP potrebbe essere assegnato a una classe di priorità più alta rispetto a un datagramma del protocollo di posta elettronica IMAP (porte 143 o 993) e quindi ricevere un servizio migliore.

Un ISP potrebbe anche utilizzare l'indirizzo IP di origine di un datagramma per assegnare la priorità ai datagrammi inviati da alcune società (che presumibilmente hanno pagato l'ISP per questo privilegio) rispetto ai datagrammi inviati da altre società (che non hanno pagato); un ISP potrebbe persino bloccare il traffico con un indirizzo IP di origine in una determinata azienda o nazione. Sono molti i meccanismi che consentirebbero a un ISP di fornire diversi livelli di servizio a diverse classi di traffico. La vera domanda è: quali politiche e quali leggi determinano che cosa un ISP possa effettivamente fare? Naturalmente, queste leggi varieranno a seconda del paese [Smithsonian 2017]. Considereremo brevemente solo la politica statunitense su ciò che è divenuto noto come *net neutrality*.

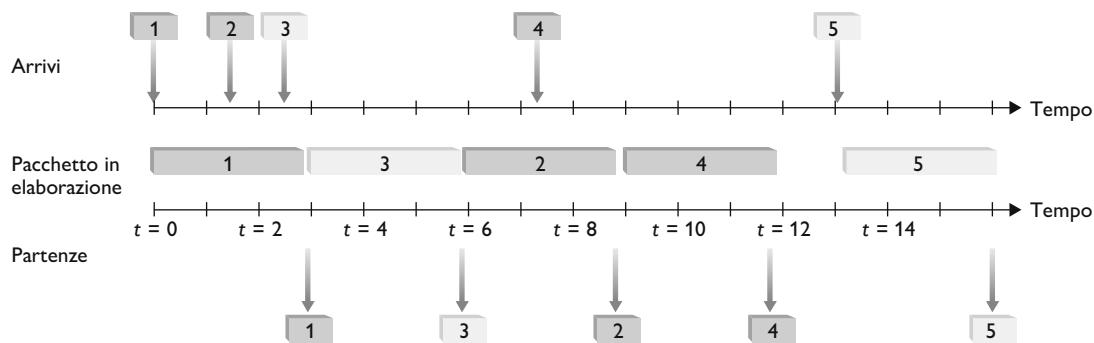
Il termine “*net neutrality*” non deriva da una precisa decisione, ma la *Order on Protecting and Promoting an Open Internet* del marzo 2015 [FCC 2015], emanata dalla US Federal Communication Commission, fornisce tre regole chiave, definite “bright line rules”, che vengono ora spesso associate al concetto di neutralità della rete.

- **No Blocking:** agli operatori di rete broadband è proibito il blocco di contenuti, applicazioni, servizi e dispositivi legali.
- **No Throttling:** agli operatori di rete broadband è proibito alterare o degradare il traffico Internet legale in base al contenuto, applicazione, servizio o dispositivo.
- **No Paid Prioritization:** agli operatori di rete broadband è proibito dare priorità sulla base di accordi commerciali, in modo diretto o indiretto, a parte del traffico Internet.

Si noti che prima di tale direttiva erano state osservate alcune violazioni delle prime due regole da parte degli ISP [Faulhaber 2012]. Nel 2005 un ISP in North Carolina accettò di bloccare la propria politica di impedire ai propri clienti di utilizzare Vonage, un servizio di voice over IP in competizione con il proprio servizio telefonico. Nel 2007 si giudicò che Comcast interferisse con il traffico P2P di BitTorrent creando e inviando internamente pacchetti RST TCP ai mittenti e destinatari BitTorrent, che si videro chiudere la connessione BitTorrent [FCC 2008].

Entrambi i risvolti della questione della neutralità della rete sono stati strettamente discussi, con particolare enfasi sulla misura con cui la neutralità della rete offra vantaggi ai clienti, promuovendo allo stesso tempo l'innovazione [Peha 2006, Faulhaber 2012, Economides 2017, Madhyastha 2017].

La *Order on Protecting and Promoting an Open Internet* del marzo 2015 [FCC 2015] sulla protezione e la promozione di una Internet aperta, che ha vietato agli ISP di bloccare, limitare o fornire priorità a pagamento è stata sostituita nel 2017 dal *Restoring Internet Freedom Order* [FCC 2017], che ha annullato tali divieti e si è invece concentrata sulla trasparenza degli ISP. Visti l'importanza della materia e i molti cambiamenti in atto, è lecito pensare di non siamo vicini alla fine del capitolo sulla *net neutrality* né negli Stati Uniti né altrove.



**Figura 4.15** Operazioni di una coda round robin con due classi.

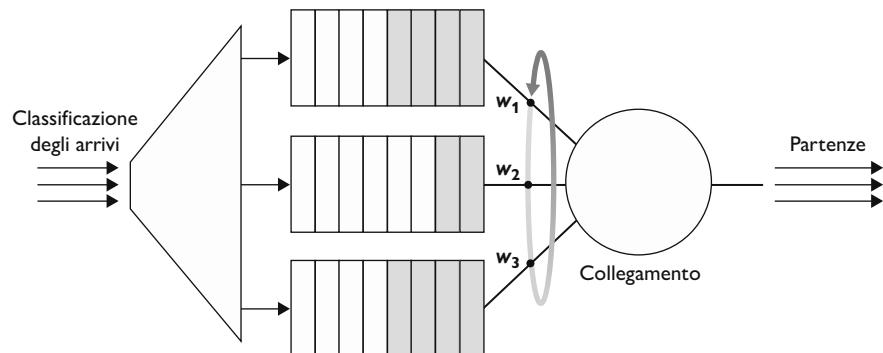
### Round Robin e Weighted Fair Queuing (WFQ)

Anche nella modalità di accodamento round robin (*round robin queuing*), i pacchetti sono suddivisi in classi, senza una rigida priorità di servizio, ma con un’alternanza tra le classi. La forma più semplice di round robin prevede l’alternanza della trasmissione: prima viene inviato un pacchetto della classe 1 e poi uno della classe 2, quindi nuovamente un pacchetto di classe 1, e così via. Nella **modalità conservativa** (*work-conserving round robin*), il collegamento non resta mai inattivo fintanto che ci sono pacchetti, di qualsiasi classe, da trasmettere; se la coda di una classe è vuota viene immediatamente controllata quella successiva.

Un esempio di modalità round robin con due classi è mostrato nella Figura 4.15. Nello schema i pacchetti 1, 2 e 4 sono assegnati alla prima classe e gli altri alla seconda. Il pacchetto 1 inizia a essere trasmesso subito dopo il suo ingresso nel buffer. Nel frattempo sopraggiungono i pacchetti 2 e 3 che vengono posti nelle rispettive code. Terminato il primo invio, lo scheduling cerca un pacchetto del secondo gruppo e quindi trasmette quello contrassegnato con il numero 3; poi estrae il pacchetto 2 dalla coda della prima classe e lo invia. Terminata la trasmissione e non essendoci altri pacchetti del secondo gruppo viene immediatamente inviato il pacchetto 4.

Un’astrazione generalizzata di questa strategia che ha trovato vasto impiego nei router è la cosiddetta modalità di **accodamento equo ponderato** (**WFQ**, *Weighted Fair Queuing*) [Demers 1990; Parekh 1993]. Nello schema WFQ, cui la Figura 4.16 fa riferimento, i pacchetti in arrivo sono classificati e accodati in

**Figura 4.16**  
Accodamento equo ponderato (WFQ).



base alla classe. Anche WFQ offre un servizio di tipo ciclico, per cui, nel caso di tre categorie, serve prima la classe 1, poi la 2 e infine la 3; per ricominciare quindi dal primo gruppo. Inoltre, anche WFQ è conservativa e pertanto quando la coda di una classe è vuota si sposta immediatamente a quella successiva.

Diversamente da round robin, con WFQ le varie classi possono ricevere un servizio differenziato. In particolare, a ciascuna classe  $i$  è assegnato un peso  $w_i$ . In qualsiasi momento in cui nella coda  $i$  siano presenti pacchetti che debbano essere spediti, WFQ garantisce che questi ricevano una frazione di servizio pari a:  $w_i/(\sum w_j)$ , dove al denominatore è indicata la somma effettuata su tutte le classi che hanno pacchetti da trasmettere. Anche nel caso peggiore, quando tutte le classi hanno pacchetti accodati, la classe  $i$  avrà la garanzia di ricevere una certa frazione di larghezza di banda. Quindi, per un collegamento con capacità trasmissiva  $R$ , la classe  $i$  avrà sempre un rendimento almeno pari a:  $R \cdot w_i/(\sum w_j)$ . Certamente, la nostra descrizione di WFQ è “idealizzata”, in quanto non è stato considerato il fatto che i pacchetti sono unità discrete e il loro invio non può essere interrotto per iniziare un’altra trasmissione. Questo aspetto è dettagliatamente esaminato in [Demers 1990] e [Parekh 1993].

## 4.3 Il Protocollo Internet (IP): IPv4, indirizzamento, IPv6 e altro ancora

Finora la nostra trattazione del livello di rete in questo capitolo (il concetto di piano dei dati e di piano di controllo del livello di rete, la distinzione tra inoltro e instradamento, l’identificazione di vari modelli di servizi di rete e l’analisi dell’interno dei router) spesso non ha fatto riferimento a un’architettura o a un protocollo di rete specifici. In questo paragrafo ci concentreremo sugli aspetti chiave del livello di rete dell’Internet di oggi e sul celebre Protocollo Internet (**IP**, *Internet Protocol*).

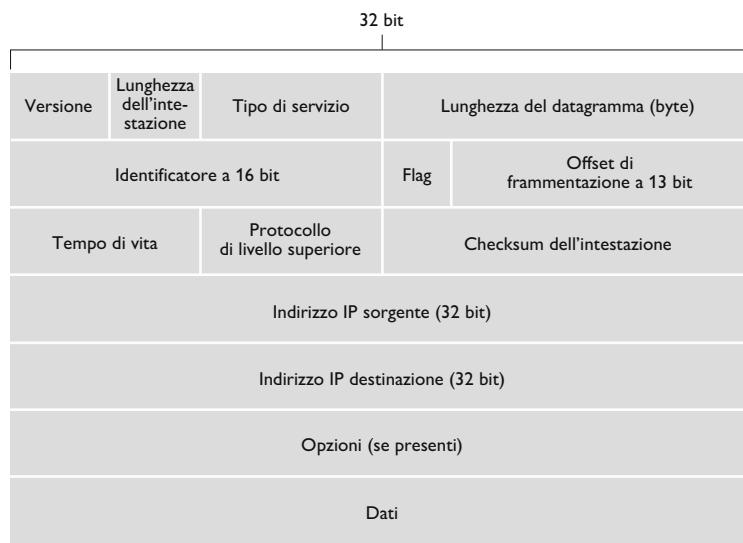
Attualmente sono in uso due versioni di IP. Esamineremo dapprima nel Paragrafo 4.3.1 il protocollo IP versione 4, ampiamente adottato, solitamente indicato con IPv4 [RFC 791]. Nel Paragrafo 4.3.4 tratteremo la versione 6 di IP [RFC 2460; RFC 4291], indicata con IPv6 proposta in sostituzione di IPv4. Nel mezzo, analizzeremo principalmente l’indirizzamento Internet, un argomento apparentemente piuttosto noioso e ostico, ma che si rivelerà cruciale per comprendere il funzionamento del livello di rete di Internet.

### 4.3.1 Formato dei datagrammi IPv4

Ricordiamo che il pacchetto a livello di rete è noto come *datagramma*. Iniziamo la nostra trattazione di IP con una panoramica della sintassi e della semantica dei datagrammi IPv4, argomento che potrebbe apparire arido e di scarso interesse. Ciò nonostante, il datagramma gioca un ruolo centrale in Internet: qualsiasi studente e professionista nel campo del networking deve necessariamente conoscerlo e padroneggiarlo (e a dimostrazione del fatto che le intestazioni di protocollo possono essere un argomento di studio divertente, consultate [Pomeranz 2010].) Il formato dei datagrammi Ipv4 è mostrato nella Figura 4.17; i principali campi dei datagrammi IPv4 sono i seguenti:

- **Numero di versione** (*version number*). Questi quattro bit, che specificano la versione del protocollo IP del datagramma, consentono al router la corretta

**Figura 4.17** Formato dei datagrammi IPv4.



interpretazione del datagramma; infatti, versioni diverse di IP hanno differenti formati per i datagrammi. La Figura 4.17 illustra il formato della versione corrente di IP, IPv4, mentre quello della nuova versione 6 (IPv6) verrà trattato nel Paragrafo 4.3.4.

- **Lunghezza dell'intestazione** (*header length*). Dato che un datagramma IPv4 può contenere un numero variabile di opzioni (incluse nell'intestazione), questi 4 bit indicano dove iniziano effettivamente i dati del datagramma. La maggior parte dei datagrammi IP non contiene opzioni, pertanto il tipico datagramma IP ha un'intestazione di 20 byte.
- **Tipo di servizio**. I bit relativi al tipo di servizio (TOS, *type of service*) sono stati inclusi nell'intestazione IPv4 per distinguere diversi tipi di datagrammi (per esempio, quelli che richiedono basso ritardo, alto throughput o affidabilità). Spesso è utile distinguere datagrammi in tempo reale (usati nelle applicazioni di telefonia) da altro traffico (per esempio, FTP). Lo specifico livello di servizio è determinato dall'amministratore del router. Abbiamo già visto nel Paragrafo 3.7.2 che i due bit del campo TOS sono utilizzati per la notifica di congestione esplicita.
- **Lunghezza del datagramma** (*datagram length*). Rappresenta la lunghezza totale del datagramma IP, intestazione più dati, misurata in byte. Considerato che questo campo è lungo 16 bit, la massima dimensione dei datagrammi IP è 65.535 byte, anche se raramente questi superano i 1500 in modo da non superare la lunghezza massima del campo dati dei frame Ethernet.
- **Identificatore, flag, offset di frammentazione** (*identifier, flags, fragmentation offset*). Questi tre campi servono per la cosiddetta frammentazione: un datagramma IP grande viene frammentato in datagrammi IP più piccoli che vengono inoltrati in modo indipendente alla destinazione dove vengono riassemblati prima che i dati di payload (si veda in seguito) vengano passati al livello di trasporto dell'host di destinazione. La nuova versione di IP, IPv6, non con-

sente la frammentazione. Non tratteremo qui la frammentazione che però può essere rintracciata nelle precedenti edizioni del libro reperibili on-line.

- **Tempo di vita.** Il campo *time-to-live* (TTL) è stato incluso per assicurare che i datagrammi non restino in circolazione per sempre nella rete (per esempio, a causa di un instradamento ciclico). Questo campo viene decrementato di un'unità ogni volta che il datagramma è elaborato da un router; quando raggiunge 0, il datagramma deve essere scartato.
- **Protocollo.** Questo campo è usato quando il datagramma raggiunge la destinazione finale. Il valore del campo indica lo specifico protocollo a livello di trasporto al quale vanno passati i dati del datagramma. Per esempio, il valore 6 indica che i dati sono destinati a TCP, mentre il valore 17 designa UDP. Tutti i valori possibili sono elencati in [IANA Protocol Numbers 2016]. Il numero di protocollo nel datagramma IP ha un ruolo analogo a quello del campo Numero di porta nel segmento a livello di trasporto. Il numero di protocollo è l'anello di collegamento tra i livelli di rete e di trasporto, mentre il numero di porta è il “collante” che lega i livelli di trasporto e di applicazione. Vedremo nel corso del Capitolo 6 come anche il frame a livello di collegamento presenti un campo speciale, che lega il livello di collegamento al livello di rete.
- **Checksum dell'intestazione (*header checksum*).** Consente ai router di rilevare gli errori sui bit nei datagrammi ricevuti. È calcolato trattando ogni coppia di byte dell'intestazione come numeri che sono poi sommati in complemento a 1. Come illustrato nel Paragrafo 3.3, il complemento a 1 di questa somma, noto come checksum Internet, viene memorizzato nel campo corrispondente. Un router calcola tale valore per ciascun datagramma IP ricevuto e rileva una condizione di errore se il checksum trasportato nell'intestazione del datagramma non corrisponde a quello calcolato. I router normalmente scartano i datagrammi in cui si verifica un errore. Notiamo che il checksum deve essere ricalcolato e aggiornato a ogni router, come anche il campo TTL e i campi opzione, che possono cambiare. Un'interessante discussione sugli algoritmi per il calcolo rapido del checksum Internet si trova in [RFC 1071]. Una domanda che spesso si pone a questo punto è: perché TCP/IP effettua la verifica di errore sia a livello di trasporto che di rete? Esistono vari motivi per questa ripetizione. Innanzitutto, notiamo che a livello IP il checksum riguarda soltanto l'intestazione, mentre il checksum TCP/UDP è calcolato sull'intero segmento. In secondo luogo, TCP/UDP e IP non appartengono necessariamente alla stessa pila di protocolli. TCP, in linea di principio, può essere usato su un protocollo diverso (per esempio, ATM) [Black 1995] e IP può trasportare dati che non verranno passati a TCP/UDP.
- **Indirizzi IP sorgente e destinazione (*source and destination IP addresses*).** Quando un host crea un datagramma, inserisce il proprio indirizzo IP nel campo indirizzo IP sorgente e quello della destinazione nel campo indirizzo IP destinazione. Spesso l'host sorgente determina l'indirizzo di destinazione attraverso una ricerca DNS (Capitolo 2). Tratteremo in dettaglio l'indirizzamento IP nel Paragrafo 4.3.2.
- **Opzioni (*options*).** Questi campi consentono di estendere l'intestazione IP. Le opzioni dell'intestazione sono state concepite per un utilizzo sporadico.

Da qui la decisione di non includere l'informazione dei campi opzione nell'intestazione di tutti i datagrammi. Tuttavia, le opzioni costituiscono un problema: dato che possono avere lunghezza variabile, non è possibile determinare a priori dove comincerà il campo Dati. Inoltre, dato che i datagrammi possono richiedere o non richiedere l'elaborazione delle opzioni, il tempo necessario per questa operazione su un router può variare in modo significativo. Queste considerazioni diventano particolarmente importanti nel caso di router e host ad alte prestazioni. Per questi e altri motivi le opzioni IP sono state eliminate dall'intestazione IPv6 (Paragrafo 4.3.4).

- **Dati (payload).** Nella maggior parte dei casi, il campo dati contiene il segmento a livello di trasporto (TCP o UDP) da consegnare alla destinazione. Tuttavia, può trasportare anche altri tipi di dati, quali i messaggi ICMP (Paragrafo 5.6).

I datagrammi IP hanno 20 byte di intestazione (*header*), escludendo le opzioni. I datagrammi non frammentati che trasportano segmenti TCP hanno 40 byte complessivi di intestazione: 20 di intestazione IP più 20 di intestazione TCP, assieme al messaggio di livello di applicazione.

### 4.3.2 Indirizzamento IPv4

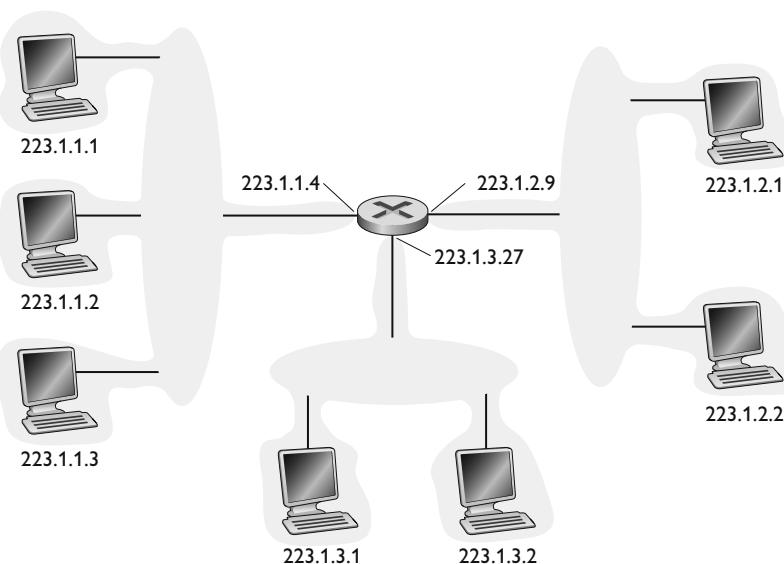
Sebbene si possa pensare che l'indirizzamento sia un argomento banale, con tutta probabilità entro la fine di questo capitolo vi sarete convinti che l'indirizzamento Internet non è solo interessante, ma anche di primaria importanza. Eccellenti trattazioni dell'indirizzamento IPv4 si trovano nel primo capitolo di [Stewart 1999].

Prima di iniziare la trattazione è necessario soffermarci sul modo in cui gli host e i router sono connessi per formare una rete. Generalmente un host ha un solo collegamento con la rete; quando il livello IP dell'host vuole inviare un datagramma, lo fa su tale collegamento. Il confine tra host e collegamento fisico viene detto **interfaccia (interface)**. Un router, invece, dato che ha il compito di ricevere datagrammi da un collegamento e inoltrarli su un altro, deve necessariamente essere connesso ad almeno due collegamenti. Anche il confine tra un router e i suoi collegamenti è chiamato interfaccia. Il router presenta più interfacce, una su ciascuno dei suoi collegamenti. Dato che host e router sono in grado di inviare e ricevere datagrammi, IP richiede che tutte le interfacce abbiano un proprio indirizzo IP. Pertanto, *l'indirizzo IP è tecnicamente associato a un'interfaccia, anziché all'host o al router che la contiene*.

Gli indirizzi IP sono lunghi 32 bit (4 byte) e quindi ci sono in totale  $2^{32}$  indirizzi IP, cioè circa 4 miliardi. Tali indirizzi sono solitamente scritti nella cosiddetta **notazione decimale puntata (dotted-decimal notation)**, in cui ciascun byte dell'indirizzo viene indicato in forma decimale ed è separato con un punto dagli altri byte dell'indirizzo. Per esempio, nell'indirizzo IP 193.32.216.9 il numero 193 è l'equivalente decimale dei primi 8 bit dell'indirizzo; 32 è l'equivalente dei secondi 8, e così via. Pertanto, l'indirizzo 193.32.216.9 in notazione binaria diventa

11000001 00100000 11011000 00001001

Ogni interfaccia di host e router di Internet ha un indirizzo IP globalmente univoco (eccetto quelle gestite da NAT, come vedremo nel Paragrafo 4.3.3).



**Figura 4.18** Indirizzi delle interfacce e sottoreti.

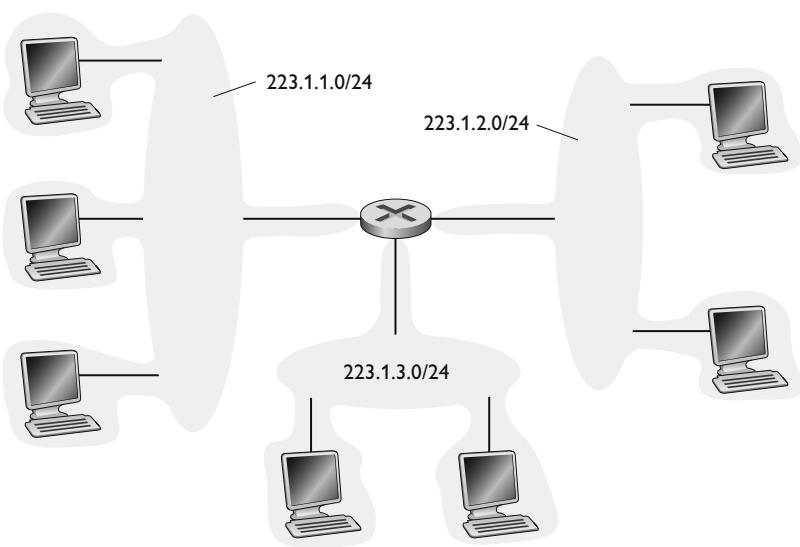
Tuttavia, tali indirizzi non possono essere scelti in modo arbitrario. Una parte dell'indirizzo di un'interfaccia è determinata dalla sottorete cui è collegata.

La Figura 4.18 mostra un router (con tre interfacce) che connette sette host. I tre a sinistra e l'interfaccia del router cui sono connessi hanno un indirizzo IP della forma 233.1.1.xxx: ossia, i 24 bit più a sinistra nell'indirizzo IP sono identici. Le quattro interfacce sono interconnesse da una rete che non contiene router. Se questa rete fosse, per esempio, una LAN Ethernet, le interfacce sarebbero interconnesse da uno switch Ethernet (Capitolo 6) o da un punto di accesso wireless (si veda il Capitolo 7, disponibile in inglese sulla piattaforma MyLab). Per adesso rappresentiamo la rete priva di router che connette questi host come una nuvola.

Per IP, questa rete che interconnette tre interfacce di host e l'interfaccia di un router forma una **sottorete** [RFC 950]. Nella letteratura relativa a Internet le sottoreti sono anche chiamate *reti IP* o semplicemente *reti*. IP assegna a questa sottorete l'indirizzo 223.1.1.0/24, dove la notazione /24, detta anche **maschera di sottorete** (*subnet mask*), indica che i 24 bit più a sinistra dell'indirizzo definiscono l'indirizzo della sottorete. Di conseguenza, la sottorete 223.1.1.0/24 consiste di tre interfacce di host (223.1.1.1, 223.1.1.2, 223.1.1.3) e di un'interfaccia di router (223.1.1.4). Ogni altro host connesso alla sottorete 223.1.1.0/24 deve avere un indirizzo della forma 223.1.1.xxx. Altre due sottoreti sono mostrate nella Figura 4.18: la rete 223.1.2.0/24 e la sottorete 23.1.1.xxx. La Figura 4.19 riporta gli indirizzi delle tre sottoreti.

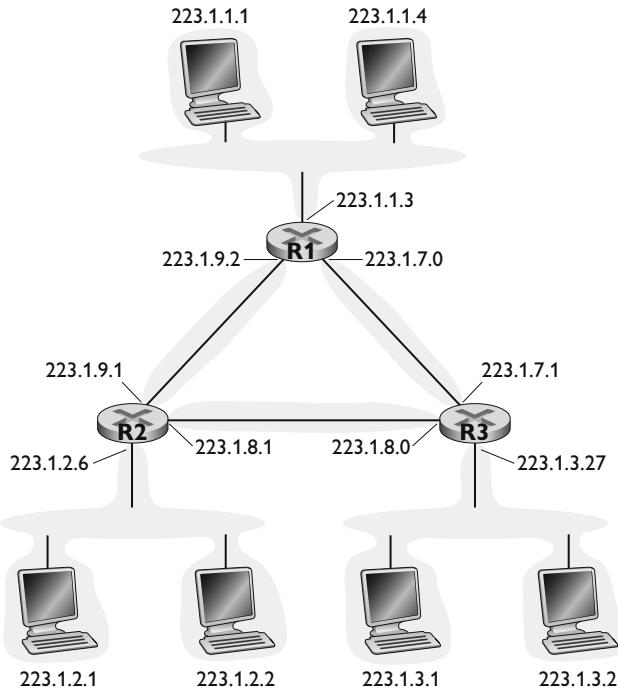
La definizione IP di sottorete non è ristretta a segmenti Ethernet che collegano più host all'interfaccia di un router. Per una miglior comprensione, consideriamo la Figura 4.20 che mostra 3 router connessi da collegamenti punto a punto (*point-to-point*). Ciascuno ha tre interfacce, due per i collegamenti punto a punto e una per il collegamento broadcast che connette direttamente il router a una coppia di host. Oltre alle tre sottoreti (223.1.1.0/24, 223.1.2.0/24 e 223.1.3.0/24), simili a quelle che abbiamo incontrato nella figura precedente, ne

**Figura 4.19** Indirizzi di sottorete.



esistono altre: (1) 223.1.9.0/24 per le interfacce che connettono i router  $R1$  e  $R2$ ; (2) 223.1.8.0/24 per le interfacce che connettono i router  $R2$  e  $R3$ ; (3) 223.1.7.0/24 per le interfacce che connettono i router  $R3$  e  $R1$ . In generale, possiamo usare la seguente procedura per definire le sottoreti di un sistema.

**Figura 4.20** Tre router che interconnettono sei sottoreti.



*Per determinare le sottoreti si sgancino le interfacce da host e router in maniera tale da creare isole di reti isolate delimitate dalle interfacce. Ognuna di queste reti isolate viene detta sottorete (subnet).*

Se applichiamo questa procedura al sistema interconnesso nella Figura 4.20, otteniamo sei isole e quindi sei sottoreti.

Dalla precedente discussione è chiaro che un'organizzazione (quale un'azienda o un'università) che disponga di più segmenti Ethernet e collegamenti punto a punto presenta più sottoreti i cui dispositivi hanno lo stesso indirizzo di sottorete. In linea di principio, le diverse sottoreti potrebbero avere indirizzi alquanto differenti, anche se, in pratica, questi presentano molti punti in comune. Per comprenderne il motivo, studiamo ora la gestione dell'indirizzamento nell'Internet globale.

La strategia di assegnazione degli indirizzi Internet è detta **Classless Interdomain Routing** [RFC 4632]. **CIDR** (che si pronuncia come l'inglese *cider*, ovvero sidro) generalizza la nozione di indirizzamento di sottorete. Come in quest'ultimo caso, l'indirizzo IP viene diviso in due parti e mantiene la forma decimale puntata  $a.b.c.d/x$ , dove  $x$  indica il numero di bit nella prima parte dell'indirizzo.

Gli  $x$  bit più a sinistra di un indirizzo della forma  $a.b.c.d/x$  costituiscono la porzione di rete dell'indirizzo IP e sono spesso detti **prefisso** (di rete) dell'indirizzo. A un'organizzazione viene generalmente assegnato un blocco di indirizzi contigui con un prefisso comune (si veda al riguardo il Box 4.1, *Neutralità della rete*) per tutti gli indirizzi IP dei dispositivi che si trovano al suo interno. Quando tratteremo il protocollo di instradamento BGP (Paragrafo 5.4) vedremo che i router esterni alla rete dell'organizzazione considerano solo gli  $x$  bit del prefisso, cioè, quando un router all'esterno dell'organizzazione inoltra un datagramma avente un indirizzo di destinazione che è interno, dovrà considerare solo i primi  $x$  bit dell'indirizzo. Questo riduce in modo considerevole la dimensione della tabella di inoltro dei router, dato che una sola riga della forma  $a.b.c.d/x$  è sufficiente per far pervenire i pacchetti all'organizzazione.

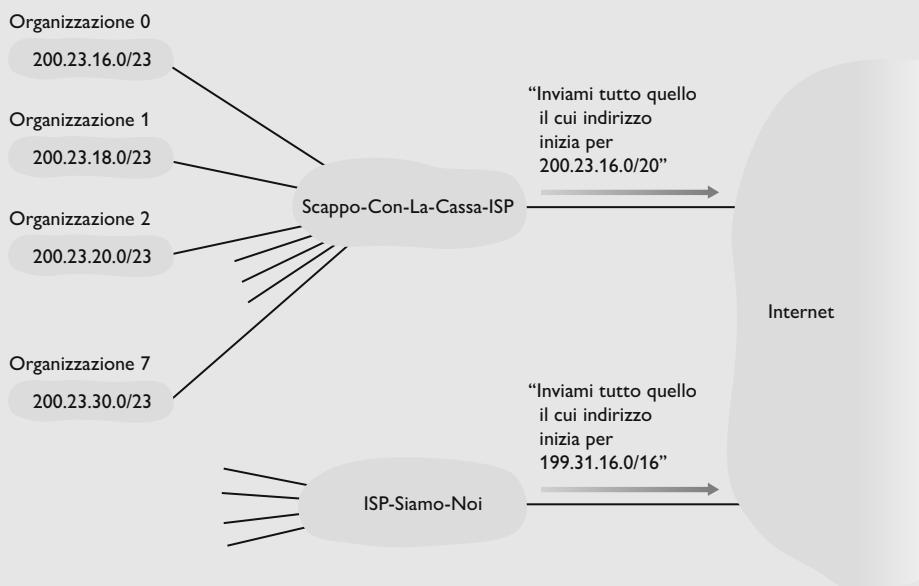
I rimanenti  $32-x$  bit di un indirizzo possono essere usati per distinguere i dispositivi interni dell'organizzazione, che hanno tutti lo stesso prefisso di rete. Saranno quindi i router della rete interna che utilizzeranno i restanti bit dell'indirizzo per indirizzarli al dispositivo destinatario. Tali bit potrebbero presentare un'aggiuntiva struttura di sottorete, come quella trattata precedentemente. Per esempio, supponiamo che i primi 21 bit dell'indirizzo CIDR  $a.b.c.d/21$  specifichino il prefisso della rete dell'organizzazione e siano comuni agli indirizzi IP dei suoi dispositivi. I restanti 11 bit allora identificheranno gli host dell'organizzazione. La struttura interna della rete potrebbe usare gli 11 bit più a destra per le sottoreti all'interno dell'organizzazione, come visto precedentemente. Per esempio,  $a.b.c.d/24$  potrebbe fare riferimento a una specifica sottorete dell'organizzazione.

Prima dell'adozione di CIDR, le parti di rete di un indirizzo IP dovevano essere lunghe 8, 16 o 24 bit. Tale schema di indirizzamento era noto come **classful addressing**, dato che le sottoreti con indirizzi di sottorete da 8, 16 e 24 bit erano note rispettivamente come reti di classe A, B e C. Il requisito che la parte di sottorete di un indirizzo IP fosse lungo esattamente 1, 2 o 3 byte si rivelò presto problematico nel supportare il numero di organizzazioni in rapida cre-

**BOX 4.2****TEORIA E PRATICA****Aggregazione di indirizzi**

Questo esempio relativo a un ISP che connette otto organizzazioni a Internet illustra come l'instradamento sia facilitato da indirizzi CIDR accuratamente allocati. Supponiamo, come mostrato nella Figura 4.21, che il provider (chiamato Scappo-Con-La-Cassa-ISP) comunichi che gli debbano essere inviati tutti i datagrammi i cui primi 20 bit di indirizzo corrispondano a 200.23.16.0/20. Il resto del mondo non ha bisogno di sapere che all'interno del blocco di indirizzi 200.23.16.0/20 esistono di fatto altre otto organizzazioni, ciascuna con la propria sottorete. Questa possibilità di usare un singolo prefisso per sottendere più reti viene spesso detta **aggregazione di indirizzi** (*address aggregation*, *route aggregation* o *route summarization*).

L'aggregazione di indirizzi funziona molto bene quando gli indirizzi sono allocati in blocchi agli ISP e da questi alle organizzazioni loro clienti. Ma che cosa avviene quando gli indirizzi non sono allocati in modo gerarchico? Che cosa succederebbe, per esempio, se Scappo-Con-La-Cassa-ISP acquisisse un provider chiamato ISP-Siamo-Noi e poi facesse connettere l'Organizzazione 1 a Internet attraverso la propria sussidiaria ISP-Siamo-Noi? Come mostrato

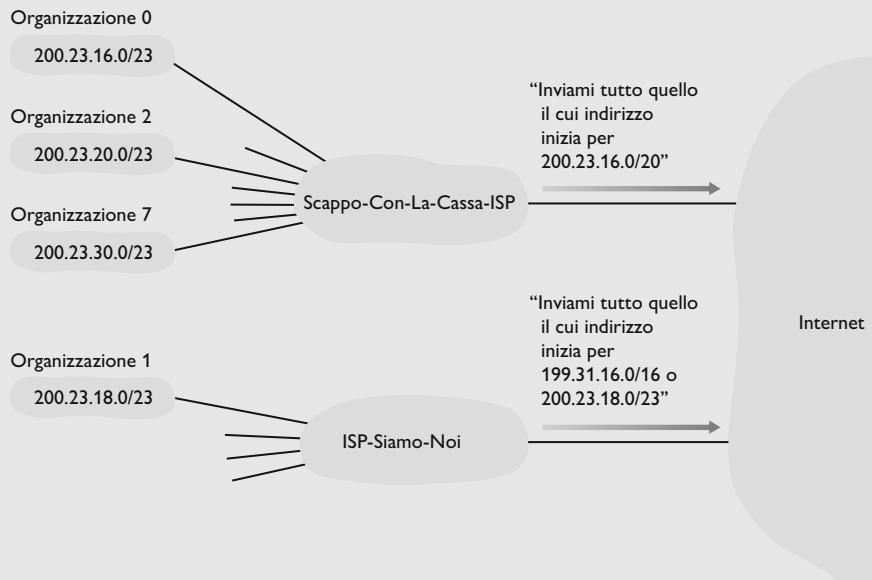


**Figura 4.21** Indirizzamento gerarchico e aggregazione di indirizzi.

scita con sottoreti di piccole e medie dimensioni. Una sottorete di classe C (/24) poteva ospitare solo fino a  $2^8 - 2 = 254$  host (due dei 256 indirizzi sono riservati per usi speciali): troppo pochi per molte organizzazioni. D'altro canto, alcune sottoreti di classe B (/16), che possono avere 65.634 host, risultavano sovrdimensionate. Con il *classful addressing*, a un'organizzazione con 2000 host veniva allocato un indirizzo di sottorete di classe B (/16). Ciò portò a un rapido esaurimento degli indirizzi della classe B e a uno scarso utilizzo dello spazio di indirizzamento assegnato. Per esempio, l'ente che usava indirizzi di classe B per i suoi 2000 host si vedeva allocare spazio a sufficienza per 65.534 interfacce, lasciando più di 63.000 indirizzi inutilizzati.

Costituirebbe una lacuna non menzionare un altro tipo di indirizzo IP, il cosiddetto indirizzo IP broadcast 255.255.255.255. Quando un host emette un da-

nella Figura 4.21, la sussidiaria possiede il blocco di indirizzi 199.31.0.0/16, ma gli indirizzi IP dell'Organizzazione 1 sfortunatamente non appartengono a tale blocco. Che cosa si deve fare in questi casi? Di certo, l'Organizzazione 1 potrebbe rinumerare tutti i propri router e host al fine di presentare indirizzi all'interno del blocco di ISP-Siamo-Noi. Si tratta tuttavia di una soluzione costosa e l'Organizzazione 1 in futuro potrebbe essere riassegnata a un'altra sottorete. La soluzione tipica consiste nel tenere gli indirizzi IP dell'Organizzazione 1 in 200.23.18.0/23. In questo caso, come mostrato nella Figura 4.22, Scappo-Con-La-Cassa-ISP continua a mostrare il blocco di indirizzi 200.23.16.0/20 e ISP-Siamo-Noi continua a mostrare 199.31.0.0/16. Però, ISP-Siamo-Noi ora mostra anche il blocco di indirizzi dell'Organizzazione 1, 200.23.18.0/23. Quando altri router di Internet vedono i blocchi di indirizzi 200.23.16.0/20 (da Scappo-Con-La-Cassa-ISP) e 200.23.18.0/23 (da ISP-Siamo-Noi) e vogliono instradare su un indirizzo nel blocco 200.23.18.0/23, useranno la corrispondenza a prefisso più lungo (Paragrafo 4.2.1) e instraderanno verso ISP-Siamo-Noi, dato che quest'ultimo mostra il prefisso più lungo (più specifico) che corrisponde all'indirizzo di destinazione.



**Figura 4.22** ISP-Siamo-Noi presenta un percorso più specifico verso Organizzazione 1.

tagramma con destinazione 255.255.255.255, il messaggio viene consegnato a tutti gli host sulla stessa sottorete. I router possono inoltrare il messaggio alle sottoreti confinanti (sebbene solitamente non lo facciano).

Dopo aver studiato l'indirizzamento IP, abbiamo ora bisogno di sapere come gli host e le sottoreti acquisiscano i propri indirizzi. Per prima cosa vedremo come un'organizzazione ottenga un blocco di indirizzi e quindi studieremo come questi siano assegnati ai dispositivi.

### Come ottenere un blocco di indirizzi

Per ottenere un blocco di indirizzi IP da usare in una sottorete, un amministratore di rete deve innanzitutto contattare il proprio ISP, che potrebbe fornire degli indirizzi attingendo da un blocco più grande che gli è già stato allocato. Un provider,

al quale, a titolo di esempio, sia stato allocato il blocco di indirizzi 200.23.16.0/20, potrebbe a sua volta dividerlo in otto blocchi uguali di indirizzi contigui e fornirne uno a ciascuna delle otto organizzazioni che supporta, come mostrato di seguito. Per comodità abbiamo sottolineato la parte di sottorete di questi indirizzi.

|                  |                |                                            |
|------------------|----------------|--------------------------------------------|
| Blocco dell'ISP  | 200.23.16.0/20 | <u>11001000 00010111 00010000 00000000</u> |
| Organizzazione 0 | 200.23.16.0/23 | <u>11001000 00010111 00010000 00000000</u> |
| Organizzazione 1 | 200.23.18.0/23 | <u>11001000 00010111 00010010 00000000</u> |
| Organizzazione 2 | 200.23.20.0/23 | <u>11001000 00010111 00010100 00000000</u> |
| ...              | ...            | ...                                        |
| Organizzazione 7 | 200.23.30.0/23 | <u>11001000 00010111 00011110 00000000</u> |

Rivolgersi a un ISP è solo un modo per ottenere un blocco di indirizzi, ma non è l'unico, anche perché lo stesso ISP deve richiedere, a sua volta, un blocco di indirizzi. Esiste un'autorità globale che ha la responsabilità ultima di gestire lo spazio di indirizzamento IP e di allocare i blocchi di indirizzi: l'*Internet Corporation for Assigned Names and Numbers* (ICANN) [ICANN 2020], che opera sulla base delle linee guida tracciate in [RFC 7020]. Il ruolo di ICANN, un'organizzazione senza scopo di lucro, non è solo allocare indirizzi IP, ma anche gestire i root DNS server. Ha anche il compito, assai controverso, di assegnare e risolvere dispute sui nomi di dominio. ICANN alloca indirizzi ai registri Internet regionali (per esempio, ARIN, RIPE, APNIC e LACNIC, che formano insieme l'*Address Supporting Organization* di ICANN [ASO-ICANN 2020]), i quali si occupano dell'allocazione e della gestione degli indirizzi all'interno delle rispettive regioni.

### Come ottenere l'indirizzo di un host: DHCP

Un'organizzazione che ha ottenuto un blocco di indirizzi li può assegnare individualmente alle interfacce di host e router nella propria struttura. Per gli indirizzi delle interfacce dei router, l'amministratore di sistema configura manualmente gli indirizzi IP nel router (spesso da remoto, tramite uno strumento di gestione della rete). Gli indirizzi degli host possono essere configurati manualmente, ma più spesso questo compito è svolto utilizzando il **Dynamic Host Configuration Protocol (DHCP)** [RFC 2131]. DHCP consente a un host di ottenere un indirizzo IP in modo automatico, così come di apprendere informazioni aggiuntive, quali la sua maschera di sottorete, l'indirizzo del router per uscire dalla sottorete (spesso detto *router di default* o *gateway*) e l'indirizzo del suo DNS server locale. L'amministratore di rete può configurare DHCP di modo che un dato host riceva un indirizzo IP persistente, in modo che ogni volta che l'host entra in rete gli venga assegnato sempre lo stesso indirizzo IP, oppure in modo da assegnare a ciascun host che si connette un **indirizzo IP temporaneo**, che sarà diverso tutte le volte che l'host si collega alla rete.

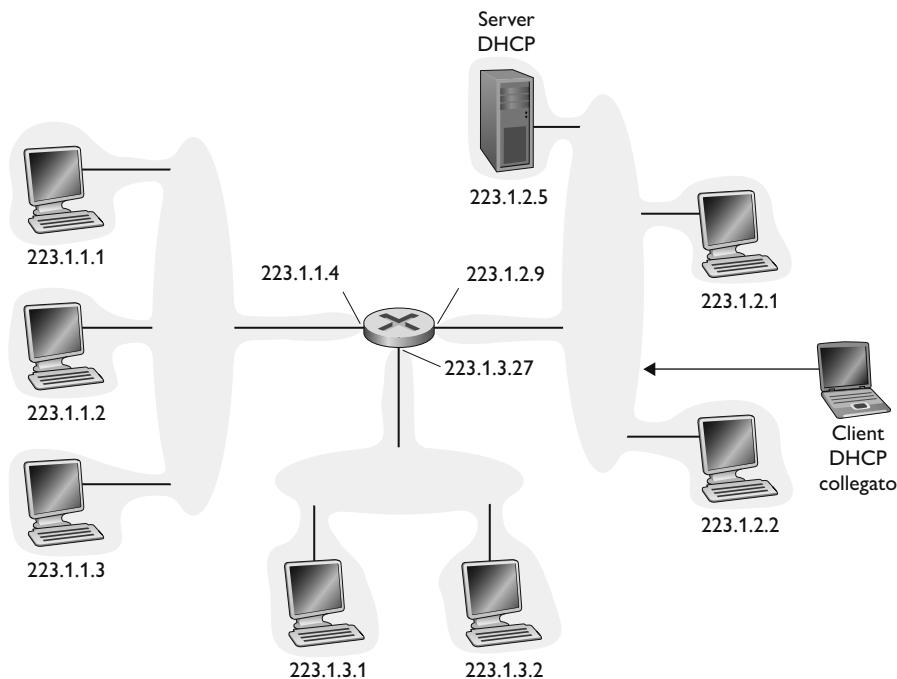
DHCP viene spesso detto protocollo **plug-and-play** o **zero-conf** (*zero-configuration*) per la sua capacità di automatizzare la connessione degli host alla rete. Questa peculiarità lo rende molto interessante per gli amministratori di rete che, altrimenti, dovrebbero svolgere questi compiti manualmente. DHCP è anche largamente usato nelle reti residenziali di accesso a Internet e nelle LAN wireless, dove gli host arrivano e se ne vanno con estrema frequenza dalla rete. Consideriamo, per esempio, uno studente che sposta il portatile dalla propria stanza alla biblioteca e poi in classe: è probabile che in ogni luogo lo studente

sia collegato a una nuova sottorete e quindi abbia bisogno di un nuovo indirizzo IP. DHCP è perfettamente adatto a questa situazione nella quale ci sono molti utenti che vanno e vengono e gli indirizzi sono necessari solo per una quantità di tempo limitata. L'importanza della caratteristica di plug-and-play del DHCP è chiara, considerando il fatto che l'alternativa consiste nel configurare manualmente l'indirizzo IP dell'host. Non è immaginabile che si debba riconfigurare il portatile a ogni nuova connessione e che gli studenti (eccetto quelli che studiano reti) siano in grado di configurare i loro computer portatili.

DHCP è un protocollo client-server. Un client è di solito un host appena connesso che desidera ottenere informazioni sulla configurazione della rete, non soltanto su uno specifico indirizzo IP. Nel caso più semplice, ogni sottorete (nel senso dell'indirizzamento nella Figura 4.20) dispone di un server DHCP. In caso contrario, è necessario un agente di relay DHCP (generalmente implementato in un router), che conosca l'indirizzo di un server DHCP per quella rete. La Figura 4.23 mostra un server DHCP collegato alla sottorete 223.1.2/24, con il router che opera da agente di relay per i client collegati nelle sottoreti 223.1.1/24 e 223.1.3/24. Nella trattazione seguente ipotizzeremo che nella sottorete sia disponibile un DHCP server.

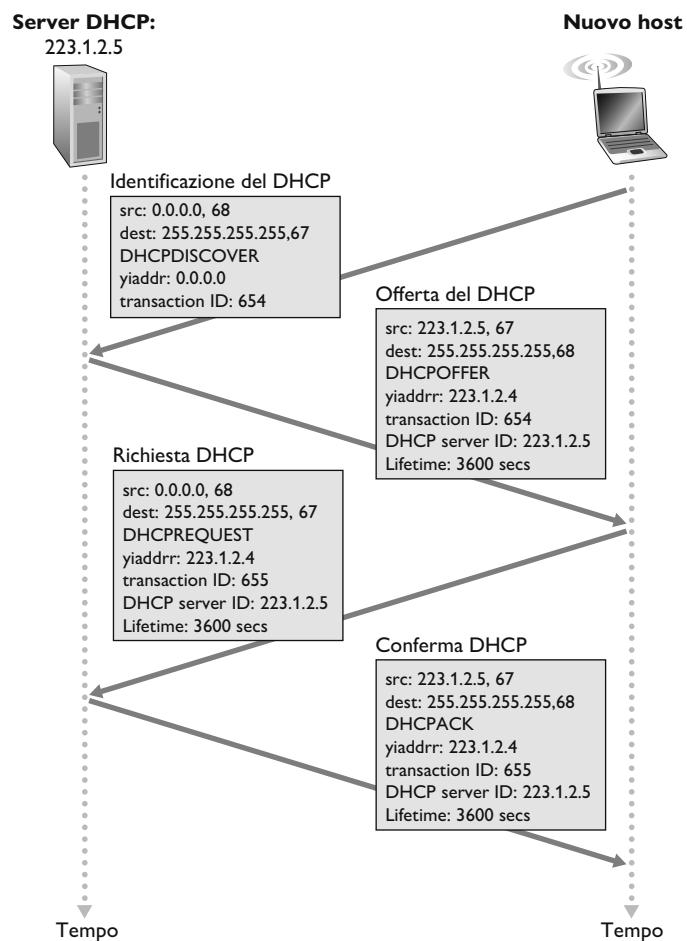
Per i nuovi host, il protocollo DHCP si articola in quattro punti (Figura 4.24 per le impostazioni di rete illustrate nella Figura 4.23). In questa figura, *yiaddr* (che sta per *your Internet address*, il tuo indirizzo Internet) indica l'indirizzo assegnato al client appena connesso. I quattro passi sono:

- **Individuazione del server DHCP.** Il primo compito di un host appena collegato è l'identificazione del server DHCP con il quale interagire. Questa operazione è svolta utilizzando un messaggio **DHCP discover**, che un client invia in un pacchetto UDP attraverso la porta 67. Il pacchetto UDP è encapsulato



**Figura 4.23** Scenario del protocollo DHCP client-server.

**Figura 4.24** Interazione client–server DHCP.



in un datagramma IP. Ma a chi dovrebbe essere inviato questo datagramma? L'host non conosce ancora l'indirizzo IP della rete alla quale è collegato e ancor meno l'indirizzo di un server DHCP per quella rete. Detto ciò, il client DHCP crea un datagramma IP contenente il suo messaggio DHCP con l'indirizzo IP di destinazione broadcast di 255.255.255.255 e l'indirizzo IP sorgente di 0.0.0.0, cioè “questo host”. Il client DHCP inoltra il datagramma IP al suo livello di collegamento, che invia il frame in broadcast a tutti i nodi collegati alla sottorete. Vedremo i dettagli dell'invio in broadcast a livello di collegamento nel Paragrafo 6.4.

- **Offerta del server DHCP.** Un server DHCP, che riceve un messaggio di identificazione, risponde al client con un messaggio **DHCP offer**, che viene inviato in broadcast a tutti i nodi della sottorete, usando di nuovo l'indirizzo IP broadcast 255.255.255.255 (dovreste chiedervi come mai anche la risposta del server deve essere in broadcast). Dato che in una sottorete possono essere presenti diversi server DHCP, il client dovrebbe trovarsi nell'invidiabile posizione di essere in condizione di scegliere tra le diverse “offerte” disponibili. Ciascun messaggio di offerta server contiene l'ID di transazione del messaggio di identificazione ricevuto, l'indirizzo IP proposto al client, la maschera

di sottorete e la durata della concessione (**lease time**) dell’indirizzo IP (il lasso di tempo durante il quale l’indirizzo IP sarà valido). Tale valore è comune-mente dell’ordine delle ore o dei giorni [Droms 2002].

- **Richiesta DHCP.** Il client appena collegato sceglierà tra le offerte dei server e risponderà con un messaggio **DHCP request**, che riporta i parametri di configurazione.
- **Conferma DHCP.** Il server risponde al messaggio di richiesta DHCP con un messaggio **DHCP ACK**, che conferma i parametri richiesti.

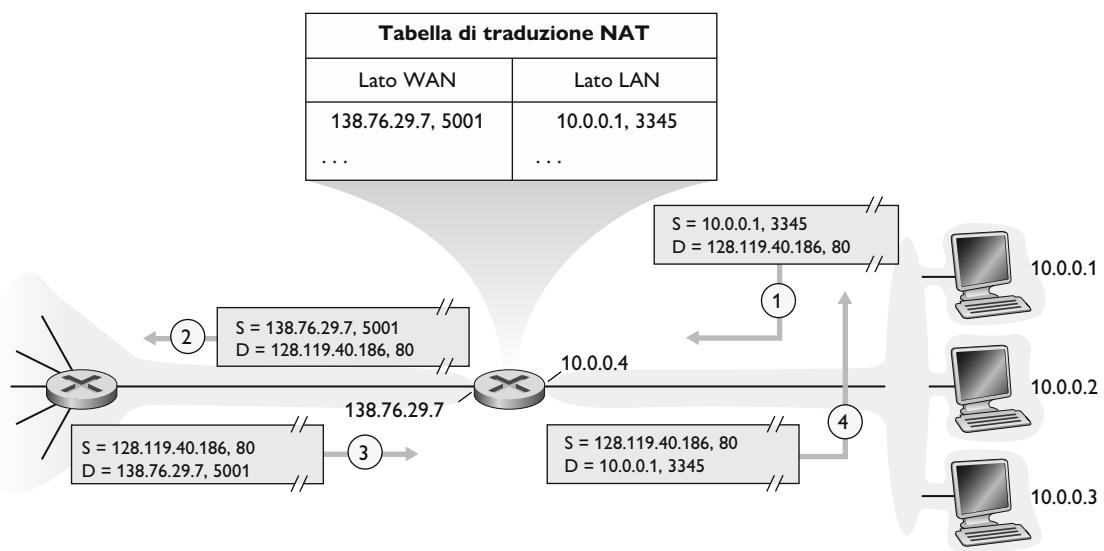
Quando il client riceve il DHCP ACK, l’interazione è completata e il client può utilizzare l’indirizzo IP fornito da DHCP per la durata della concessione. Dato che un client potrebbe voler utilizzare il proprio indirizzo IP oltre la durata della sua concessione, DHCP fornisce anche un meccanismo che consente ai client di rinnovare la concessione di un indirizzo IP.

Tuttavia, dal punto di vista della mobilità, DHCP non è privo di difetti. Quando un nodo si connette a una nuova sottorete, DHCP gli rilascia un nuovo indirizzo IP; non si può quindi mantenere una connessione TCP a un’applica-zione remota, spostandosi il nodo mobile da una sottorete a un’altra. Nel Capitolo 7, disponibile in inglese sulla piattaforma MyLab, vedremo come le reti cellulari consentano a un nodo mobile di conservare lo stesso indirizzo permanente e le connessioni TCP anche spostandosi tra base station diverse. Ulteriori dettagli su DHCP si possono trovare in [Droms 2002] e [dhc 2020]. Un’implementazione di riferimento di DHCP che mette a disposizione il codice sorgente si può ricavare da ISC (Internet System Consortium) [ISC 2020].

### 4.3.3 NAT (Network Address Translation)

Dopo la nostra trattazione sugli indirizzi Internet e sul formato dei datagrammi IPv4, siamo ora ben consci del fatto che ogni dispositivo IP richiede un indirizzo. La proliferazione di sottoreti “small office, home office” (o SOHO, piccoli uffici in ambiente domestico), sembrerebbe implicare che ogni volta che si voglia installarne una, l’ISP debba allocare un intervallo di indirizzi per coprire tutte le macchine della sottorete (tra le quali smartphone, tablet, console di gioco, TV IP, stampanti etc). Di conseguenza, al crescere della LAN dovrebbe esserne allo-cato un blocco maggiore di indirizzi. Ma che cosa avverrebbe se l’ISP avesse già allocato la parte contigua all’intervallo di indirizzi dell’attuale rete SOHO? E che cosa dovrebbe sapere il normale utente per gestire gli indirizzi IP? Fortunatamente, esiste un approccio più semplice e sempre più usato: il **NAT** (*Network Address Translation*), [RFC 2663; RFC 3022; Huston 2004; Zhang 2007; Houston 2017].

La Figura 4.25 mostra l’attività di un router abilitato al NAT, con un’inter-faccia che fa parte della rete domestica (sulla destra della figura). Come visto in precedenza, le quattro interfacce della rete domestica hanno lo stesso indirizzo di sottorete, 10.0.0.0/24. Lo spazio di indirizzamento 10.0.0.0/8 è una delle tre parti dello spazio di indirizzi IP riservato alle **reti private** [RFC 1918], o **realm** (*realm*) **con indirizzi privati**: ossia, una rete i cui indirizzi hanno significato solo per i dispositivi interni. In effetti, esistono centinaia di migliaia di reti pri-  
vate, molte delle quali usano un identico spazio di indirizzamento, 10.0.0.0/24, per scambiare pacchetti fra i loro dispositivi. Ovviamen-te, quelli inviati

**Figura 4.25** NAT.

sull'Internet globale non possono utilizzare questi indirizzi come sorgente o destinazione. Ma se gli indirizzi privati hanno significato solo all'interno di una data rete, come viene gestito l'indirizzamento dei pacchetti relativi all'Internet globale, in cui gli indirizzi sono necessariamente univoci? La risposta è il NAT.

I router abilitati al NAT non appaiono come router al mondo esterno, ma si comportano come un *unico* dispositivo con un *unico* indirizzo IP. Nella figura tutto il traffico che lascia il router domestico verso Internet ha l'indirizzo IP di origine 138.76.29.7 e tutto il traffico in entrata deve avere lo stesso indirizzo come destinazione. In sostanza, il router abilitato al NAT nasconde i dettagli della rete domestica al mondo esterno. Contestualmente, ci si potrebbe chiedere dove i calcolatori della rete domestica ottengano i propri indirizzi e dove il router acquisisca il proprio indirizzo IP. Spesso la risposta è DHCP. Il router ottiene il proprio indirizzo dal server DHCP dell'ISP e manda in esecuzione un server DHCP per fornire gli indirizzi ai calcolatori all'interno dello spazio di indirizzamento della rete domestica.

Se tutti i datagrammi in arrivo al router NAT dalla rete geografica hanno lo stesso indirizzo IP di destinazione (nello specifico, quello dell'interfaccia sul lato WAN del router NAT), allora come apprende il router a quale host interno dovrebbe essere inoltrato un determinato datagramma? Il trucco consiste nell'utilizzare una **tavola di traduzione NAT** (NAT translation table) nel router NAT e nell'includere nelle righe di tale tabella i numeri di porta oltre che gli indirizzi IP.

Facciamo ancora riferimento alla Figura 4.25 e supponiamo che un utente che si trovi nella rete domestica dietro l'host 10.0.0.1 richieda una pagina web da un server (porta 80) con indirizzo IP 128.119.40.186. L'host 10.0.0.1 assegna il numero di porta di origine (arbitrario) 3345 e invia il datagramma nella rete locale. Il router NAT riceve il datagramma, genera per esso un nuovo numero di porta di origine 5001, sostituisce l'indirizzo IP sorgente con il proprio indirizzo IP sul lato WAN 138.76.29.7 e sostituisce il numero di porta di origine iniziale 3345 con il nuovo numero 5001. Quando genera un nuovo numero di porta di

origine, il router NAT può selezionare qualsiasi numero di porta che attualmente non si trova nella tabella di traduzione NAT. Notiamo che, essendo il campo numero di porta lungo 16 bit, il protocollo NAT può supportare più di 60.000 connessioni simultanee con un solo indirizzo IP sul lato WAN relativo al router. Il NAT del router aggiunge inoltre una riga alla propria tabella di traduzione NAT. Il web server, ignaro della manipolazione subita dal datagramma in arrivo con la richiesta HTTP, risponde con un datagramma con l'indirizzo IP del router NAT come destinazione e il cui numero di porta di destinazione è 5001. Quando questo datagramma arriva al router NAT, quest'ultimo consulta la tabella di traduzione NAT usando l'indirizzo IP di destinazione e il numero di porta di destinazione per ottenere l'appropriato l'indirizzo IP (10.0.0.1) e il corretto numero di porta di destinazione (3345) del browser nella rete domestica. Il router, quindi, riscrive l'indirizzo di destinazione del datagramma e il suo numero di porta di destinazione e inoltra il datagramma nella rete domestica.

NAT ha conosciuto un'ampia diffusione negli ultimi anni; tuttavia è giusto menzionare che ha anche molti detrattori. Innanzitutto, si può argomentare che i numeri di porta sono stati concepiti per indirizzare processi, non per individuare gli host. La cosa può infatti causare problemi ai server in esecuzione su reti domestiche dato che i processi server attendono richieste in ingresso su numeri di porta prestabiliti (Capitolo 2). Anche i peer in un protocollo P2P devono accettare connessioni quando agiscono da server. Sono state proposte alcune soluzioni, come l'**attraversamento del NAT** (*NAT traversal*) [RFC 5389, RFC 5128, Ford 2005].

Argomentazioni più filosofiche sono state sollevate dai “puristi” delle reti. I router dovrebbero elaborare i pacchetti solo fino al livello 3. NAT viola il cosiddetto *principio end-to-end*: gli host dovrebbero comunicare tra loro direttamente, senza intromissione di nodi né modifica di indirizzi IP e di numeri di porta. Torneremo su questo dibattito nel Paragrafo 4.5 dove tratteremo i dispositivi middlebox.

#### 4.3.4 IPv6

Nei primi anni '90 l'Internet Engineering Task Force diede inizio allo sviluppo del successore di IPv4. Una prima motivazione di tale sforzo era legata alla considerazione che lo spazio di indirizzamento IP a 32 bit stava cominciando a esaurirsi, dato che nuove sottoreti e nuovi nodi IP venivano connessi a Internet (e si vedevano allocare indirizzi IP univoci) con spasmatica frequenza. Per soddisfare l'esigenza di un grande spazio di indirizzamento venne sviluppato un nuovo protocollo: IPv6. Sulla base dell'esperienza accumulata, i progettisti colsero l'opportunità per apportare migliorie e modifiche rispetto alla precedente versione.

La stima del momento in cui gli indirizzi IPv4 sarebbero stati esauriti (e di conseguenza non sarebbe stato possibile aggiungere nuove sottoreti a Internet) fu oggetto di un interessante dibattito. Sulla base delle tendenze dell'epoca nell'allocazione di indirizzi, due figure di spicco del gruppo di lavoro Address Lifetime Expectations di IETF stimarono che l'esaurimento degli indirizzi si sarebbe verificato, rispettivamente, nel 2008 e nel 2018 [Solensky 1996]. Nel febbraio 2011 IANA allocò a un'autorità regionale l'ultimo blocco di indirizzi IPv4 rimasto. Benché i registri regionali avessero ancora a disposizione indirizzi IPv4 nei loro blocchi, una volta esauriti non avrebbero più potuto avere altri

**BOX 4.3** **FOCUS SULLA SICUREZZA****Firewall e sistemi di rilevamento delle intrusioni**

Supponete che vi sia affidato il compito di amministrare una rete domestica, quella di un dipartimento, di un'università o di un'azienda. Gli aggressori (*attackers*), conoscendo l'intervallo di indirizzi IP della vostra rete, possono facilmente inviare datagrammi IP a indirizzi interni a quell'intervallo. Questi datagrammi possono causare qualsiasi tipo di danno, compresa la rilevazione della struttura della vostra rete, ricerche tramite ping e scansioni delle porte, mandare in blocco host vulnerabili con pacchetti malformati, inondare i server con una quantità enorme di pacchetti ICMP e infettare gli host includendo malware nei pacchetti. Come amministratore di rete, che cosa fareste per contrastare questi malintenzionati? Due comuni meccanismi di protezione dagli attacchi con pacchetti malevoli sono i *firewall* e i *sistemi di rilevamento delle intrusioni* (IDS, *intrusion detection system*).

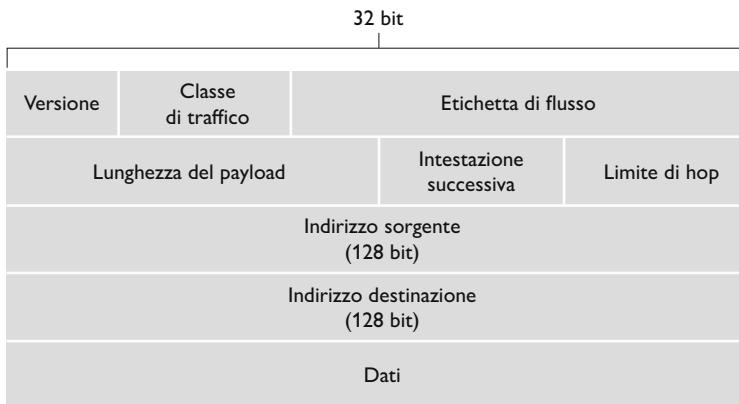
Come amministratore di rete, in primo luogo, potete provare a installare un firewall tra la vostra rete e Internet. Molti degli odierni router di accesso hanno le funzionalità dei firewall. I firewall controllano i datagrammi e i campi dell'intestazione dei segmenti in essi contenuti, impedendo l'accesso all'interno della rete ai datagrammi sospetti. Un firewall può essere per esempio configurato per bloccare tutti i pacchetti ICMP di richiesta di echo (Paragrafo 5.6), impedendo così a un aggressore di effettuare una ricerca tramite ping nel vostro intervallo di indirizzi IP. I firewall possono anche bloccare pacchetti in base agli indirizzi IP e numeri di porta sorgente e destinazione. Inoltre, possono essere configurati per tener traccia delle connessioni TCP, garantendo l'accesso solo ai datagrammi appartenenti a connessioni approvate.

Una protezione aggiuntiva può essere fornita con un IDS, tipicamente posizionato ai confini della rete, che effettua un “ispezione approfondita del pacchetto”, esaminando non solo i campi dell'intestazione, ma anche il payload nei datagrammi (compresi i dati a livello applicativo). Un IDS ha un database di firme di pacchetti, che sono noti far parte di attacchi tipici. Quando viene scoperto un nuovo attacco, questa base di dati è automaticamente aggiornata. Quando i pacchetti passano attraverso l'IDS, esso tenta di far corrispondere i campi dell'intestazione e il payload alle firme presenti nella base di dati. Se viene trovata una corrispondenza, viene creato un allarme. Un *sistema di prevenzione delle intrusioni* (IPS, *intrusion prevention system*) è simile a un IDS, eccetto che blocca effettivamente i pacchetti, oltre a creare allarmi. Nel Paragrafo 4.5 e nel Capitolo 8, disponibile in inglese sulla piattaforma MyLab, esamineremo firewall e IDS in maggior dettaglio.

Ma firewall e IDS sono in grado di proteggervi completamente da tutti gli attacchi? La risposta è chiaramente no, in quanto gli aggressori trovano continuamente nuove strategie di attacco. Tuttavia, i firewall e gli IDS tradizionali basati sulle firme sono utili per proteggere la vostra rete da attacchi noti.

blocchi [Huston 2011a]. Si consulti [Richter 2015] e [Houston 2019] per una recente trattazione dell'argomento.

Sebbene a metà degli anni '90 le stime suggerissero una considerevole distanza temporale dall'esaurimento completo dello spazio di indirizzamento IPv4, si pensò che sarebbe stato necessario un tempo notevole per il rilascio di una nuova tecnologia su scala tanto estesa. Pertanto, si diede inizio alla progettazione dell'IP versione 6 [RFC 1752], (IPv6) [RFC 2460]. Una domanda frequente riguarda la sorte di IPv5. Inizialmente si pensò che il protocollo ST-2 sarebbe diventato IPv5, ma questo venne successivamente scartato. Un'eccellente fonte d'informazione su IPv6 è [Huitema 1998].



**Figura 4.26** Formato dei datagrammi IPv6.

## Formato dei datagrammi IPv6

La Figura 4.26 mostra il formato dei datagrammi e illustra i cambiamenti più significativi.

- **Indirizzamento esteso.** IPv6 aumenta la dimensione dell'indirizzo IP da 32 a 128 bit, in modo che gli indirizzi IP diventino praticamente inesauribili. Ogni granello di sabbia del pianeta potrà avere il suo indirizzo IP. Oltre agli indirizzi unicast e multicast, IPv6 supporta anche indirizzi anycast, che consentono di consegnare un datagramma a un qualsiasi host all'interno di un gruppo. Questa caratteristica potrebbe essere usata, per esempio, per inviare una GET HTTP al più vicino di una serie di siti contenenti un dato documento.
- **Intestazione ottimizzata di 40 byte.** Alcuni campi IPv4 sono stati eliminati o resi opzionali. La risultante intestazione a 40 byte e a lunghezza fissa consente una più rapida elaborazione dei datagrammi IP, mentre una nuova codifica delle opzioni ne consente l'elaborazione in maniera più flessibile.
- **Etichettatura dei flussi.** IPv6 presenta una definizione elusiva di **flusso** (*flow*). L'RFC 2460 afferma che ciò consente "l'etichettatura di pacchetti che appartengono a flussi particolari per i quali il mittente richiede una gestione speciale, come una qualità di servizio diversa da quella di default o un servizio in tempo reale". La trasmissione audio e video potrebbe essere trattata per esempio come un flusso, così come il traffico ad alta priorità di un utente che paga per avere un servizio preferenziale; ma non sono considerate come flusso le applicazioni più tradizionali, quali il trasferimento file e la posta elettronica. Anche se non hanno esattamente determinato il significato da attribuire al termine flusso, appare chiaro che i progettisti di IPv6 prevedono l'esigenza di una differenziazione del traffico in diverse tipologie.

Un confronto tra le Figure 4.26 e 4.17 evidenzia la struttura più semplice e più efficiente dei datagrammi del protocollo IPv6 in cui sono definiti i seguenti campi.

- **Versione.** Campo a 4 bit che identifica il numero di versione IP (che, come si può ben immaginare, per IPv6 è 6). Porre 4 in questo campo non è però sufficiente a creare un datagramma IPv4 valido. Più avanti tratteremo il problema della transizione da IPv4 a IPv6.
- **Classe di traffico.** Campo a 8 bit, simile al campo TOS di IPv4. Può essere utilizzato per attribuire priorità a determinati datagrammi all'interno di un

flusso o provenienti da specifiche applicazioni (per esempio, voice-over-IP) rispetto a quelli di altri servizi (per esempio SMTP).

- **Etichetta di flusso.** Campo a 20 bit utilizzato per identificare un flusso di datagrammi.
- **Lunghezza del payload.** Questo valore a 16 bit è trattato come un intero senza segno e indica il numero di byte nel datagramma IPv6 che seguono l'intestazione a lunghezza fissa di 40 byte.
- **Intestazione successiva.** Campo che identifica il protocollo a cui verranno consegnati i contenuti (campo dati) del datagramma, per esempio TCP o UDP. Utilizza gli stessi valori del campo protocollo nell'intestazione IPv4.
- **Limite di hop.** Il contenuto di questo campo è decrementato di 1 da ciascun router che inoltra il datagramma. Quando il suo valore raggiunge 0, il datagramma viene eliminato.
- **Indirizzi sorgente e destinazione.** I diversi formati degli indirizzi IPv6 a 128 bit sono descritti nell'RFC 4291.
- **Dati.** Payload che viene passato al protocollo specificato nel campo di intestazione successiva quando il datagramma IPv6 raggiunge la sua destinazione.

Confrontando il formato IPv6 (Figura 4.26) con IPv4 (Figura 4.17) notiamo che sono stati eliminati vari campi.

- **Frammentazione/riassemblaggio.** IPv6 non consente frammentazione né riassemblaggio sui router intermedi; queste operazioni possono essere effettuate soltanto da sorgente o destinazione. Se un router riceve un datagramma IPv6 che risulta troppo grande per essere inoltrato sul collegamento di uscita, non fa altro che eliminarlo e inviare al mittente un messaggio d'errore ICMP “Pacchetto troppo grande” (Paragrafo 5.6). Il mittente può quindi inviare nuovamente i dati, con una dimensione di datagramma IP inferiore. La frammentazione e il riassemblaggio sono operazioni che consumano tempo; trasferire l'onere di questa funzionalità dai router ai sistemi periferici rende assai più rapido l'instradamento IP all'interno della rete.
- **Checksum dell'intestazione.** Dal momento che i protocolli Internet a livello di trasporto (per esempio, TCP e UDP) e di collegamento (per esempio, Ethernet) calcolano un loro checksum, i progettisti di IP hanno probabilmente ritenuto questa funzionalità talmente ridondante nel livello di rete da decidere di rimuoverla. Ancora una volta la principale preoccupazione è stata l'elaborazione rapida dei pacchetti IP. Richiamando la nostra discussione di IPv4 nel Paragrafo 4.3.1, ricordiamo che l'intestazione IPv4 contiene un campo TTL (simile al campo limite di hop in IPv6), ragion per cui il checksum dell'intestazione in IPv4 doveva essere ricalcolato da ogni router. Come per la frammentazione e il riassemblaggio, quest'operazione è stata considerata troppo onerosa.
- **Opzioni.** Il campo opzioni non fa più parte dell'intestazione IP standard, anche se non è del tutto scomparso. Infatti, è una delle possibili intestazioni successive cui punta l'intestazione IPv6. In altre parole, proprio come le intestazioni del protocollo TCP o UDP possono rappresentare l'intestazione successiva all'interno di un pacchetto IP, lo stesso accade anche per il campo Opzioni. Grazie all'eliminazione di tale campo, la lunghezza dell'intestazione IP è fissata a 40 byte.

## Passaggio da IPv4 a IPv6

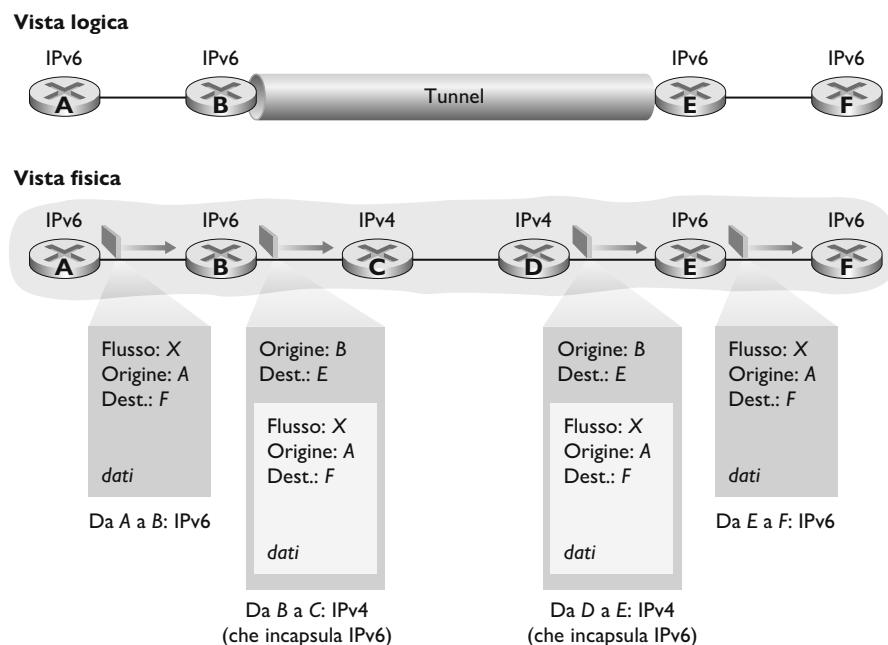
Ora che abbiamo visto i dettagli tecnici di IPv6, passiamo a un argomento pratico: il passaggio di Internet da IPv4 a IPv6. Il problema è che, mentre i nuovi sistemi IPv6 sono retrocompatibili, ossia sono in grado d'inviare, instradare e ricevere datagrammi IPv4, i sistemi IPv4 esistenti non sono in grado di gestire datagrammi IPv6. Per ovviare a questo handicap esistono diverse opzioni [Huston 2011b, RFC 4213].

Una via sarebbe quella di dichiarare una “giornata campale” in cui tutte le macchine Internet vengano spente e aggiornate da IPv4 a IPv6. L’ultima, rilevante transizione tecnologica ebbe luogo circa 25 anni fa con il passaggio da NCP a TCP per il servizio di trasporto affidabile. Ma se, perfino allora [RFC 801], quando Internet era piccola e amministrata da un manipolo di “maghi”, ci si rese conto che questa soluzione non era percorribile, tanto più appare improponibile al giorno d’oggi, con centinaia di milioni di macchine e milioni di amministratori e utenti di rete.

L’approccio alla transizione da IPv4 a IPv6 più diffusamente adottato è noto come **tunneling** [RFC 4213]. L’idea alla base del tunneling, un concetto chiave con applicazioni in molti altri scenari al di là della transizione da IPv4 a IPv6, tra cui un ampio uso nelle reti cellulari all-IP che tratteremo nel Capitolo 7, disponibile in inglese sulla piattaforma MyLab, è la seguente. Supponiamo che due nodi IPv6 (per esempio, B ed E nella Figura 4.27) vogliano utilizzare datagrammi IPv6, ma siano connessi da un insieme di router intermedi IPv4, che chiameremo **tunnel** (Figura 4.27). Il Nodo B, al lato di invio del tunnel, prende l’*intero* datagramma IPv6 pervenutogli da A e lo pone nel campo dati di un datagramma IPv4. Quest’ultimo viene quindi indirizzato al Nodo E, al lato di ricezione del tunnel e inviato al primo nodo nel tunnel (C). I router IPv4 intermedi instradano il datagramma IPv4, come farebbero per qualsiasi altro datagramma, ignari che questo contenga un datagramma IPv6 completo. Il nodo IPv6, sul lato di ricezione del tunnel, riceverà quindi il datagramma IPv4, determinerà che questo ne contiene uno IPv6 osservando che il valore del campo numero di protocollo nel pacchetto IPv4 è 41 [RFC 4213] corrispondente a payload IPv4, lo estrarrà e lo instraderà esattamente come se l’avesse ricevuto da un nodo IPv6 adiacente.

Concludiamo questo paragrafo notando che l’adozione di IPv6 ha avuto un avvio lento [Lawton 2001; Huston 2008b] e ha preso slancio solo recentemente. NIST riporta che più di un terzo dei domini governativi di secondo livello negli Stati Uniti sono abilitati a IPv6 [NIST IPv6 2020]. D’altro canto Google riporta che circa solo il 25% dei client che si connettono ai suoi servizi utilizzano IPv6 [Google IPv6 2020]; tuttavia recenti misurazioni indicano che l’adozione di IPv6 sta accelerando [Csyz 2014]. La proliferazione di dispositivi quali telefoni IP e altri apparecchi portatili fornisce impulso per il rilascio su larga scala di IPv6. Il Third Generation Partnership Program europeo [3GPP 2020] ha indicato IPv6 come standard per il sistema di indirizzamento per la distribuzione di dati multimediali su terminali mobili.

Un’importante lezione che possiamo apprendere dall’esperienza IPv6 è l’enorme difficoltà nel variare i protocolli a livello di rete. Dai primi anni ’90 numerosi nuovi protocolli a livello di rete sono stati annunciati come la prossima rivoluzione di Internet, ma la maggior parte di essi ha finora avuto una pe-

**Figura 4.27** Tunneling.

netrazione molto modesta. Tra questi protocolli includiamo IPv6, i protocolli multicast e quelli di prenotazione delle risorse (si veda la piattaforma MyLab del volume per il materiale supplementare in inglese relativamente a queste ultime due classi). Infatti introdurre nuovi protocolli nel livello di rete è come sostituire le fondamenta di una casa; è difficile farlo senza far crollare l'intero edificio o almeno traslocare temporaneamente gli abitanti. D'altro canto, Internet è stata testimone di un rapido sviluppo di nuovi protocolli a livello di applicazione. I classici esempi, ovviamente, sono il Web, la messaggistica istantanea e la condivisione di file P2P. Altri esempi includono lo *streaming* audio e video, i giochi on-line e i social media. Introdurre nuovi protocolli a livello di applicazione è come aggiungere un nuovo strato di vernice a una casa: è un compito relativamente facile da svolgere e, scegliendo colori attraenti, altri vicini vi copieranno. Riassumendo, in futuro ci possiamo aspettare cambiamenti nel livello di rete Internet, ma questi probabilmente si verificheranno su una scala di tempo assai più lenta rispetto ai cambiamenti al livello di applicazione.

## 4.4 Inoltro generalizzato e SDN

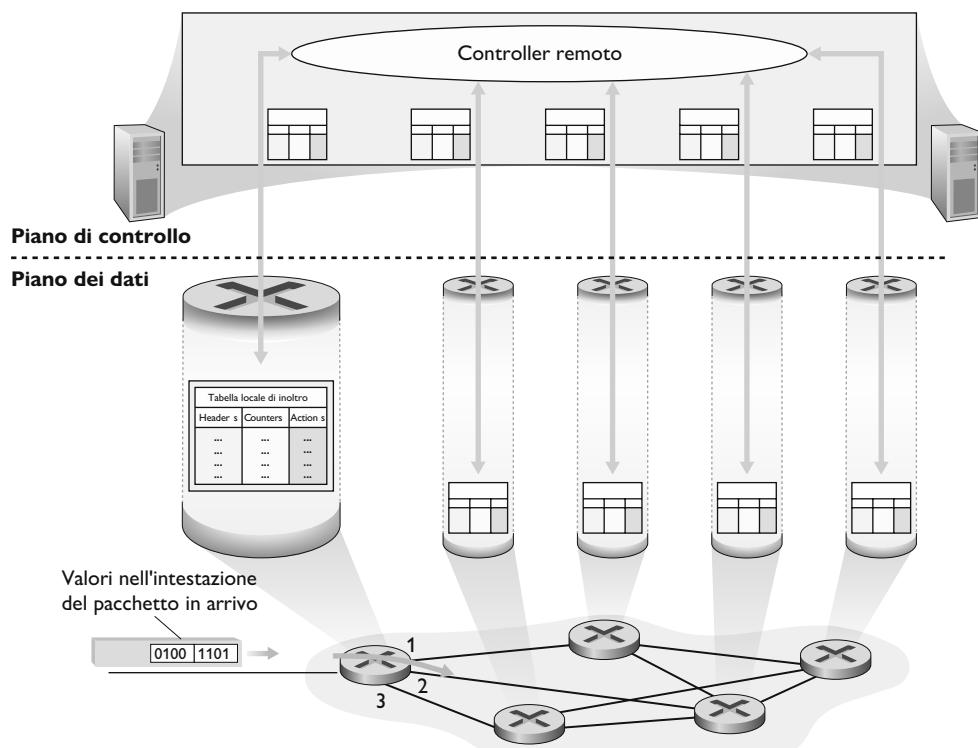
Ricordiamo dal Paragrafo 4.2.1 che l'inoltro basato sulla destinazione è caratterizzato da due passi: ricerca dell'indirizzo IP di destinazione (“match”), e poi invio del pacchetto attraverso la struttura di commutazione a una specifica porta di uscita (“action”). Consideriamo ora un paradigma molto più generale, il paradigma “match-action” (*corrispondenza e azione*), dove il “match” può essere effettuato su più campi dell'intestazione associati a differenti protocolli corrispondenti a diversi livelli della pila dei protocolli. La “action” può includere l'inoltro del pacchetto a una o più porte di uscita (come l'inoltro basato sulla destinazione), il bilanciamento di carico dei pacchetti attraverso più interfacce di uscita che portano a un servizio (*load-balancing*), la riscrittura dei valori del-

l'intestazione (come nel NAT), il blocco/scarto di un pacchetto (come in un firewall), l'invio di un pacchetto a un server speciale per ulteriori elaborazioni e azioni (come in DPI), e altro ancora.

Nell'inoltro generalizzato, una tabella match-action generalizza il concetto di tabella di inoltro basata sulla destinazione trattata nel Paragrafo 4.2.1. Poiché le decisioni di inoltro possono essere effettuate utilizzando indirizzi di sorgente e destinazione del livello di rete e/o del livello di collegamento, i dispositivi di inoltro mostrati nella Figura 4.28 sono denominati più accuratamente "packed switch" piuttosto che "router" di livello 3 o "switch" di livello 2. Quindi, nel seguito di questo paragrafo e nel Paragrafo 5.5, adotteremo tale terminologia che sta rapidamente diffondendosi nella letteratura sulle SDN.

La Figura 4.28 mostra una tabella match-action in un packed switch, calcolata, installata e aggiornata da un controller remoto. Si noti che, anche se è possibile che i componenti di controllo di un packed switch interagiscano tra di loro (come si vede nella Figura 4.2), nella pratica sono implementati da un controller remoto che calcola, installa e aggiorna le tabelle. Prendetevi un minuto per paragonare le Figure 4.2, 4.3 e 4.28: quali somiglianze e quali differenze notate tra l'inoltro basato sulla destinazione mostrato nelle Figure 4.2 e 4.3 e l'inoltro generalizzato mostrato nella Figura 4.28?

La trattazione seguente sull'inoltro generalizzato sarà basata su OpenFlow [McKeown 2008; ONF 2020; Tourrilhes 2014], uno standard di successo, pionieri del paradigma match-action così come in generale della rivoluzione SDN [Fteamster 2013]. Considereremo dapprima OpenFlow 1.0 che introduce le fun-



**Figura 4.28** Inoltro generalizzato: ogni packet switch ha una tabella match-action calcolata e distribuita da un controller remoto.

zionalità e i concetti chiave SDN in modo chiaro e conciso. Le versioni successive di OpenFlow hanno introdotto funzionalità supplementari; tutte le versioni degli standard OpenFlow possono essere consultate in [ONF 2020].

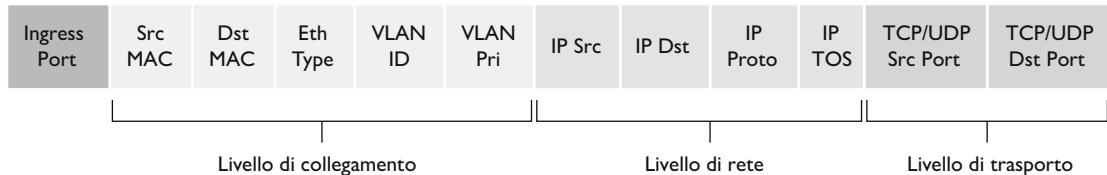
Ogni entry (occorrenza, voce) corrispondente a una riga in una tabella di inoltro match-action, nota come **tabella dei flussi** (*flow table*) in OpenFlow, contiene quanto segue.

- *Un insieme di valori dei campi dell'intestazione* con i quali il pacchetto entrante viene confrontato. Come nel caso dell'inoltro basato sulla destinazione, il confronto implementato a livello hardware è effettuato molto più rapidamente nelle memorie TCAM, che permettono più di un milione di occorrenze di indirizzi di destinazione [Bosshart 2013]. Un pacchetto che non abbia riscontro in alcuna occorrenza della tabella dei flussi può essere scaricato o inviato a un controller remoto per ulteriori elaborazioni. Nella pratica, una tabella dei flussi può essere implementata attraverso molteplici tabelle per ragioni di prestazioni o di costo [Bosshart 2013], ma qui ci focalizzeremo sull'astrazione di una singola tabella dei flussi.
- *Un insieme di contatori* che vengono aggiornati quando i pacchetti vengono associati a un'occorrenza nella tabella dei flussi. Tali contatori possono contenere il numero di pacchetti associati a tale occorrenza e l'informazione temporale sull'ultimo aggiornamento dell'occorrenza.
- *Un insieme di azioni* che devono essere intraprese quando un pacchetto è associato a un'occorrenza della tabella dei flussi. Tali azioni potrebbero essere l'inoltro di un pacchetto a una data porta di uscita, lo scarto del pacchetto, l'effettuazione delle copie del pacchetto e il loro invio a più porte di uscita e/o la riscrittura di alcuni campi dell'intestazione.

Tratteremo più dettagliatamente il paradigma match-action nei Paragrafi 4.4.1 e 4.4.2. Studieremo quindi come l'insieme delle regole di corrispondenza possa essere usato per implementare moltissime funzioni, tra cui il routing, lo switching di livello 2, il firewalling, il load balancing, le reti virtuali e molto altro nel Paragrafo 4.4.3. Infine, si noti che una tabella dei flussi è essenzialmente un'API, la cui astrazione permette di programmare il comportamento di un singolo packet switch; vedremo nel Paragrafo 4.4.3 che i comportamenti dell'intera rete possono essere programmati in modo simile, programmando/configurando nel modo appropriato tali tabelle nell'insieme dei packet switch della rete [Casado 2014].

#### 4.4.1 Match

La Figura 4.29 mostra gli undici campi dell'intestazione del pacchetto e l'ID della porta di ingresso che possono essere confrontati in una regola match-action di OpenFlow 1.0. Si ricordi dal Paragrafo 1.5.2 che un frame di livello 2 che arriva a un packet switch contiene un datagramma di livello 3 come suo carico; quest'ultimo, a sua volta, tipicamente contiene un segmento di livello 4. Come prima osservazione notiamo che l'astrazione del match di OpenFlow permette che un match venga effettuato sui campi delle intestazioni di tre livelli di protocollo (in contraddizione con la stratificazione dei protocolli vista nel Paragrafo 1.5). Anche se non abbiamo ancora trattato il livello di collegamento, possiamo almeno dire che gli indirizzi MAC di sorgente e destinazione mostrati nella Figura 4.29 sono indirizzi di livello 2 associati alle interfacce di invio e ri-



**Figura 4.29** Campi dell'intestazione del pacchetto per l'operazione di match in una tabella dei flussi OpenFlow 1.0.

cezione dei frame. Compiendo la funzione di inoltro sulla base dell'indirizzo Ethernet piuttosto che dell'indirizzo IP, possiamo dire che un dispositivo abilitato a OpenFlow può funzionare sia come un router, dispositivo di livello 3 che inoltra datagrammi, sia come uno switch, dispositivo di livello 2 che inoltra frame. Il campo Ethernet "Type" corrisponde al protocollo del livello sovrastante (per esempio IP) al quale il carico del frame viene inviato tramite demultiplexing; i campi VLAN riguardano invece le cosiddette reti virtuali locali che tratteremo nel Capitolo 6. L'insieme dei 12 valori che possono essere confrontati nelle specifiche di OpenFlow 1.0 è cresciuto fino a 41 valori nelle specifiche più recenti [Bosshart 2014].

La porta di ingresso si riferisce alla porta in entrata al packet switch sul quale il pacchetto viene ricevuto. L'indirizzo IP sorgente del pacchetto, l'indirizzo IP di destinazione, il campo IP relativo al protocollo e il campo IP riferito al tipo di servizio sono stati discussi nel Paragrafo 4.3.1. Anche i campi numero di porta della sorgente e della destinazione possono essere utilizzati nel match.

Le occorrenze della tabella dei flussi possono essere anche delle wildcard (sequenze nelle quali alcuni bit possono assumere qualunque valore). Per esempio un indirizzo IP 128.119.\*.\* potrà trovare corrispondenza in qualsiasi datagramma che abbia 128.119 come primi 16 bit del suo indirizzo. Una priorità viene associata a tutte le occorrenze della tabella dei flussi. Se un pacchetto corrisponde a più occorrenze in una tabella dei flussi, verrà scelta quella con la priorità più elevata.

Si osservi infine che, in funzione della specifica tecnologia SDN, non tutti i campi dell'intestazione IP possono essere confrontati; per esempio, OpenFlow non permette il confronto sulla base del campo TTL o lunghezza del datagramma. Ma perché alcuni campi possono essere confrontati e altri no? La risposta è che bisogna raggiungere un compromesso tra funzionalità e complessità. L'arte di scegliere un'astrazione significa fornire sufficienti funzionalità da poter effettuare un compito (in questo caso implementare, configurare e gestire un vasto assortimento di funzioni di rete precedentemente implementate attraverso diversi dispositivi di rete), senza appesantire l'astrazione con così tanti dettagli da renderla inutilizzabile. Citando Butler Lampson [Lampson 1983]:

*"Fai una cosa alla volta, ma falla bene. Un'interfaccia dovrebbe catturare il minimo essenziale di un'astrazione. Non generalizzare; le generalizzazioni sono usualmente sbagliate."*

Dato il successo di OpenFlow, evidentemente i suoi progettisti hanno scelto bene l'astrazione. Una trattazione più dettagliata del matching di OpenFlow può essere trovata in [ONF 2020].

#### 4.4.2 Action

Come mostrato nella Figura 4.28, ogni occorrenza della tabella dei flussi ha una lista di azioni che determinano l’elaborazione da effettuare su un pacchetto che le corrisponde. In caso vi siano molteplici azioni, vengono effettuate nell’ordine specificato nella lista.

Di seguito sono elencate alcune delle più importanti azioni possibili.

- **Inoltro (forwarding).** Un pacchetto in entrata può essere inoltrato a una particolare porta di uscita, inviato in broadcast a tutte le porte eccetto quella da cui è entrato o inviato in multicast a un insieme selezionato di porte. Il pacchetto può essere incapsulato e inviato al controller remoto del dispositivo. Il controller può quindi effettuare alcune azioni sul pacchetto, quali installare nuove occorrenze nella tabella dei flussi o restituire il pacchetto al dispositivo per effettuare l’inoltro in base alle regole aggiornate della tabella dei flussi.
- **Scarto (dropping).** Un’occorrenza della tabella dei flussi senza azioni indica che il pacchetto dovrebbe essere scartato.
- **Modifica dei campi (modify-field).** I valori nei dieci campi dell’intestazione del pacchetto (tutti i campi di livello 2, 3, 4 mostrati nella Figura 4.29 tranne il campo protocollo IP) possono essere riscritti prima che il pacchetto venga inoltrato alla porta di uscita selezionata.

#### 4.4.3 Esempi del paradigma match-action in OpenFlow

Applichiamo ora l’inoltro generalizzato nel contesto della rete di esempio mostrata nella Figura 4.30. La rete ha 6 host (h1, h2, h3, h4, h5, h6) e 3 packet switch (s1, s2 e s3), ognuno con 4 interfacce locali numerate da 1 a 4.

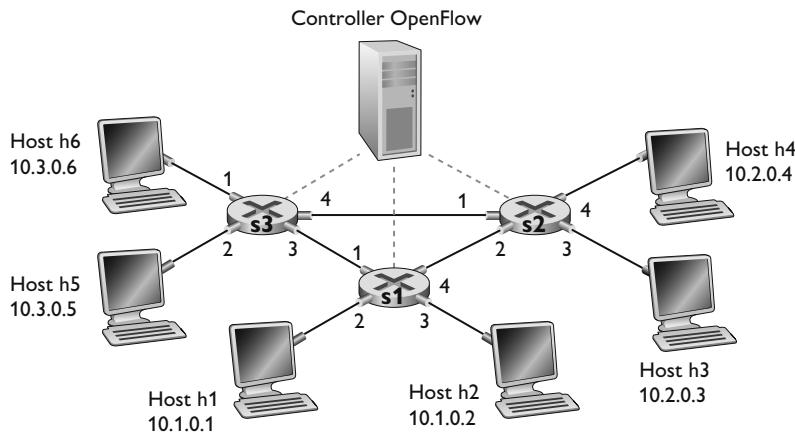
##### Un primo esempio: inoltro semplice

Come primo semplice esempio supponete che i pacchetti da h5 o h6 destinati a h3 o h4 debbano essere inoltrati da s3 a s1 e quindi da s1 a s2, evitando quindi l’uso del collegamento tra s3 e s2. L’occorrenza della tabella dei flussi in s1 sarebbe:

| s1 Flow Table (Example 1)                                |            |
|----------------------------------------------------------|------------|
| Match                                                    | Action     |
| Ingress Port = 1 ; IP Src = 10.3.*.* ; IP Dst = 10.2.*.* | Forward(4) |
| ...                                                      | ...        |

Ovviamente abbiamo anche bisogno di un’occorrenza della tabella dei flussi in s3 in modo che i datagrammi da h5 o h6 siano inoltrati a s1 sull’interfaccia di uscita 3:

| s3 Flow Table (Example 1)             |            |
|---------------------------------------|------------|
| Match                                 | Action     |
| IP Src = 10.3.*.* ; IP Dst = 10.2.*.* | Forward(3) |
| ...                                   | ...        |



**Figura 4.30** Rete OpenFlow match-action con tre packet switch, 6 host e un controller OpenFlow.

Infine, abbiamo bisogno di un'occorrenza della tabella dei flussi in **s2** in modo che i datagrammi inviati da **s1** siano inviati alla loro destinazione, **h3** o **h4**:

#### s2 Flow Table (Example 1)

| Match                                | Action     |
|--------------------------------------|------------|
| Ingress port = 2 ; IP Dst = 10.2.0.3 | Forward(3) |
| Ingress port = 2 ; IP Dst = 10.2.0.4 | Forward(4) |
| ...                                  | ...        |

#### Un secondo esempio: load balancing

Come secondo esempio si consideri uno scenario di bilanciamento del carico, nel quale i datagrammi da **h3** destinati a **10.1.\*.\*** debbano essere inoltrati sul collegamento da **s2** a **s1**, mentre i datagrammi da **h4** destinati a **10.1.\*.\*** debbano essere inoltrati sul collegamento tra **s2** e **s3** e quindi da **s3** a **s1**. Si noti che tale comportamento non potrebbe essere effettuato con l'inoltro basato sulla destinazione IP. In questo caso la tabella dei flussi in **s2** sarebbe:

#### s2 Flow Table (Example 2)

| Match                               | Action     |
|-------------------------------------|------------|
| Ingress port = 3; IP Dst = 10.1.*.* | Forward(2) |
| Ingress port = 4; IP Dst = 10.1.*.* | Forward(1) |
| ...                                 | ...        |

Le occorrenze della tabella dei flussi sono necessarie anche in **s1** in modo da inoltrare i datagrammi ricevuti da **s2** a **h1** o **h2**; anche in **s3** è necessaria una tabella dei flussi per inoltrare i datagrammi ricevuti sull'interfaccia 4 da **s3** a **s1** tramite l'interfaccia 3. Provate a scrivere voi le occorrenze della tabella dei flussi in **s1** e **s3**.

### Un terzo esempio: firewalling

Come terzo esempio si consideri uno scenario di firewall in cui s2 desideri ricevere su qualunque sua interfaccia il traffico inviato da host attaccati a s3.

| s2 Flow Table (Example 3)           |            |
|-------------------------------------|------------|
| Match                               | Action     |
| IP Src = 10.3.*.* IP Dst = 10.2.0.3 | Forward(3) |
| IP Src = 10.3.*.* IP Dst = 10.2.0.4 | Forward(4) |
| ...                                 | ...        |

Se non ci fossero altre occorrenze nella tabella dei flussi in s2, solo il traffico da 10.3.\*.\* verrebbe inoltrato agli host attaccati a s2.

Sebbene abbiamo considerato solo pochi scenari di base risulta chiaro quale siano la versatilità e i vantaggi dell'inoltro generalizzato. Nei problemi a fine capitolo esploreremo come le tabelle dei flussi possano essere usate per creare molti comportamenti logici differenti, come le reti virtuali (due o più reti separate logicamente, ognuna con il suo comportamento di inoltro indipendente e distinto), che usano lo stesso insieme fisico di packet switch e collegamenti. Nel Paragrafo 5.5 rituneremo sulle tabelle dei flussi quando studieremo i controller SDN che calcolano e distribuiscono le tabelle dei flussi e tratteremo il protocollo usato per comunicare tra i packet switch e i loro controller.

Le tabelle di flusso match-plus-action che abbiamo visto in questo paragrafo sono in realtà una forma limitata di programmabilità, che specifica come un router debba inoltrare e manipolare (per esempio, modificare un campo di intestazione) un datagramma, in base alla corrispondenza tra i valori di intestazione della sua intestazione e le condizioni di corrispondenza. Si potrebbe immaginare una forma ancora più ricca di programmabilità: un linguaggio di programmazione con costrutti ad alto livello come variabili, operazioni aritmetiche, operazioni booleane, funzioni e istruzioni condizionali, nonché costrutti pensati specificamente per l'elaborazione di datagrammi al tasso di linea. Un linguaggio di questo tipo è P4 (*Programming Protocol - independent Packet Processors*) [P4 2020] che ha riscosso un notevole interesse sin dalla sua introduzione [Bosshart 2014].

## 4.5 I dispositivi middlebox

I router sono i cavalli di battaglia del livello di rete e in questo capitolo abbiamo imparato come svolgono il loro lavoro quotidiano di inoltro di datagrammi IP alla loro destinazione. Ma in questo capitolo, e nei capitoli precedenti, abbiamo incontrato anche altri dispositivi di rete (“boxes”) che si trovano sul percorso dei dati ed eseguono funzioni diverse dall'inoltro: le web cache nel Paragrafo 2.2.5 (“Web caching”), il TCP splitting nel Paragrafo 3.7 – Box 3.5 –, la traduzione dell'indirizzo di rete (NAT) – Paragrafo 4.3.3 –, i firewall e i sistemi di rilevamento delle intrusioni nel Paragrafo 4.3.4. Abbiamo imparato nel Paragrafo 4.4 che l'inoltro generalizzato consente a un router moderno di eseguire in modo semplice le operazioni di firewall e di bilanciamento del carico con operazioni di “match + action” generalizzato.

Negli ultimi vent'anni abbiamo assistito a un'enorme crescita di tali middlebox, che RFC 3234 definisce come:

*“Qualsiasi dispositivo di rete (box) intermediario che svolga funzioni diverse dalle normali funzioni standard di un router IP sul percorso dati tra un host di origine e un host di destinazione”.*

Possiamo identificare a grandi linee tre tipi di servizi eseguiti dai middlebox:

- *Traduzione NAT.* Come abbiamo visto nel Paragrafo 4.3.4, i box NAT implementano un indirizzamento di rete privato, la riscrittura degli indirizzi IP e dei numeri di porta dell'intestazione del datagramma.
- *Servizi di sicurezza.* I firewall bloccano il traffico in base ai valori del campo di intestazione o reindirizzano i pacchetti per ulteriori elaborazioni, come nel caso del *Deep Packet Inspection* (DPI). Gli *Intrusion Detection Systems* (IDS) sono in grado di rilevare caratteristiche predeterminate e quindi di filtrare i pacchetti. I filtri di posta elettronica a livello di applicazione bloccano i messaggi di posta elettronica considerati spazzatura, phishing o che comunque costituiscono una minaccia per la sicurezza.
- *Miglioramento delle prestazioni.* Questi middlebox eseguono servizi come la compressione, il caching dei contenuti e il bilanciamento del carico delle richieste di servizio (per esempio, una richiesta HTTP o di un motore di ricerca) a uno dei server in grado di fornire il servizio desiderato.

Molti altri middlebox [RFC 3234] forniscono funzionalità che appartengono a questi tre tipi di servizi, nelle reti cellulari sia cablate sia wireless [Wang 2011].

Con la proliferazione dei middlebox sorge la necessità di operare, gestire e aggiornare questi dispositivi.

Dispositivi hardware specializzati, stack software e funzionalità operative/gestionali separati si traducono in significativi costi operativi e di capitale. Non sorprende quindi che i ricercatori stiano esplorando l'uso di hardware generale (tipicamente basato su CPU) per il networking, computing e storage con software specializzato costruito su uno stack software comune: esattamente l'approccio adottato da SDN dieci anni fa per implementare questi servizi. Tale approccio è noto come *Network Function Virtualization* (NFV) [Mijumbi 2016]. Una volta trasformate in software, le funzionalità dei middlebox possono anche essere esternalizzate nel cloud [Sherry 2012].

Per molti anni, l'architettura di Internet ha avuto una netta separazione tra i livelli di rete e i livelli di trasporto/applicazione. Ai “bei vecchi tempi”, il livello di rete consisteva in router che operavano all'interno del *core* della rete per inoltrare i datagrammi verso le loro destinazioni utilizzando solo i campi nell'intestazione del datagramma IP. I livelli di trasporto e di applicazione erano implementati negli host che operano all'*edge* della rete. Gli host si scambiavano pacchetti tra di loro nei segmenti del livello di trasporto e nei messaggi a livello di applicazione. I middlebox di oggi violano chiaramente questa separazione: un NAT, posto tra un router e un host, riscrive gli indirizzi IP a livello di rete e i numeri di porta a livello di trasporto; un firewall in rete blocca i datagrammi sospetti utilizzando i campi dell'intestazione a livello di applicazione (come i campi HTTP), a livello di trasporto e a livello di rete; i gateway di sicurezza della posta elettronica vengono posti tra il mittente del messaggio di posta elettronica, sia che siano malevoli oppure no, e il destinatario dell'e-mail, filtrando

così i messaggi di posta elettronica a livello di applicazione in base alle white-list/blacklist di indirizzi IP e al contenuto dei messaggi di posta elettronica.

Mentre alcuni considerano tali middlebox un po' come dei mostri architettonici [Garfinkel 2003], altri hanno adottato la filosofia che i middlebox “esistono per ragioni importanti e permanenti”, che rispondono a una necessità rilevante e che avremo più, non meno, middlebox in futuro [Walsh 2004].

Per un'ottica leggermente diversa con cui vedere la questione di dove collocare la funzionalità di servizio in una rete si veda la sezione dedicata a *L'argomento end-to-end* all'interno del Box 4.4.

#### BOX 4.4

#### TEORIA E PRATICA

### PRINCIPI DELL'ARCHITETTURA DI INTERNET

Dato il fenomenale successo di Internet è naturale chiedersi quali siano i principi architettonici che hanno guidato lo sviluppo di quello che è probabilmente il più grande e complesso sistema ingegneristico mai costruito dall'umanità. La RFC 1958, intitolata “Architectural Principles of the Internet” suggerisce che questi principi, se effettivamente esistono, sono veramente minimali:

“Molti membri della comunità di Internet sostengono che non esista un'architettura, ma solo una tradizione, mai messa per iscritto per i primi 25 anni (o almeno non dallo IAB). Tuttavia, in termini molto generali, la comunità crede che l'obiettivo sia la connettività, lo strumento sia il protocollo Internet e che l'intelligenza risieda più nel paradigma end-to-end che nasconde all'interno della rete.” [RFC 1958].

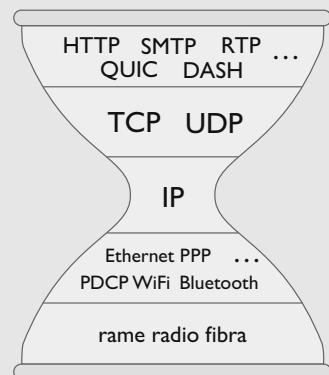
Quindi ce l'abbiamo! L'obiettivo era fornire connettività, ci sarebbe stato un solo protocollo a livello di rete, il celebre protocollo IP che abbiamo studiato in questo capitolo, e “intelligenza” (si potrebbe dire la “complessità”) sarebbe stata collocata alla periferia della rete, piuttosto che nel nucleo della rete. Esaminiamo queste ultime due considerazioni più in dettaglio.

### LA CLESSIDRA IP

Ormai conosciamo bene lo stack di protocolli Internet a cinque livelli che abbiamo incontrato per la prima volta nella Figura 1.23. Un'altra visualizzazione di questo stack, mostrata nella Figura 4.31 e nota anche come **clessidra IP** (*IP hourglass*), illustra la forma a “vita stretta” dell'architettura a livelli di Internet. Sebbene Internet abbia molti protocolli nei livelli fisico, di collegamento, di trasporto, e applicativo, esiste un solo protocollo a livello di rete: il protocollo IP. Questo è l'unico protocollo che deve essere implementato da ciascuno dei miliardi di dispositivi connessi a Internet. Questa forma a “vita stretta” ha svolto un ruolo fondamentale nella incredibile crescita di Internet. La relativa semplicità del protocollo IP e il fatto che sia l'unico requisito universale per la connettività Internet hanno permesso che una ricca varietà di reti, con tecnologie di livello di collegamento molto diverse - che variano da Ethernet al WiFi e alle reti cellulari e ottiche - diventassero parte di Internet.

[Clark 1997] nota che il ruolo della “vita stretta”, a cui si riferisce come uno “spanning layer (strato di copertura)”, significhi “... nascondere i dettagli delle differenze tra queste varie tecnologie (sostanziali) e presentare alle applicazioni che stanno sopra un'interfaccia di servizio uniforme”. In particolare, per il livello IP: “Come fa il livello IP a ottenere il suo scopo? Definisce un insieme di servizi di base, progettato con cura, in modo che sia realizzato da un'ampia gamma di tecnologie di rete sottostanti. Il software, come parte del livello Internet [cioè, di rete], traduce ciò che ciascuna di queste tecnologie di livello inferiore offre nel servizio comune del livello Internet.”

Per una discussione sulla “vita stretta”, che comprenda anche esempi al di fuori di Internet, si veda [Beck 2019; Akhshabi 2011].



**Figura 4.31** La clessidra IP a “vita stretta”.

## 4.6 Riepilogo

In questo capitolo abbiamo trattato le funzioni del piano dei dati del livello di rete: le funzioni che ogni router implementa per determinare come i pacchetti che arrivano a uno dei collegamenti in entrata del router vengano inoltrati a uno dei collegamenti in uscita del router. Abbiamo iniziato guardando in dettaglio le operazioni interne di un router, studiando le funzionalità delle porte di ingresso e di uscita e l'inoltro basato sulla destinazione, il meccanismo di commutazione interno al router, la gestione delle code dei pacchetti e molto altro. Abbiamo

Si noti che, poiché l'architettura di Internet sta entrando nella sua mezza età (l'età di 40-50 la qualifica certamente come mezza età!), la sua "vita stretta" potrebbe effettivamente allargarsi un po' (come spesso accade nella mezza età!) a causa del diffondersi dei middlebox.

### L'ARGOMENTO END-TO-END

Il terzo principio della RFC 1958, cioè che "l'intelligenza sta nel paradigma end-to-end piuttosto che nascosta all'interno della rete" si riferisce al posizionamento delle funzionalità all'interno della rete. Abbiamo visto che fino alla recente diffusione dei middlebox, la maggior parte delle funzionalità Internet era effettivamente collocata alla periferia della rete. Vale la pena notare che, in netto contrasto con la rete telefonica del ventesimo secolo, che aveva endpoint "stolti" (cioè non programmabili) e switch intelligenti, Internet ha sempre avuto endpoint intelligenti (computer programmabili), consentendo di collocare funzionalità complesse in tali endpoint. Un argomento più centrato sui principi per il posizionamento della funzionalità agli endpoint è stato proposto in un articolo che ha suscitato grande interesse [Saltzer 1984], dove si sviluppa l'"argomento end-to-end" (*the end-to-end argument*), affermando:

"... esiste un insieme di funzioni, ognuna delle quali potrebbe essere implementata in molti modi diversi: dal sottosistema di comunicazione, dal suo client, come una joint venture, o forse in modo ridondante, ciascuno eseguendone la propria versione. Nel ragionare su questa scelta, i requisiti dell'applicazione forniscono la base per una classe di argomenti, come segue:

La funzione in questione può essere implementata completamente e correttamente solo con la conoscenza e l'aiuto dell'applicazione che sta nell'endpoint del sistema di comunicazione. Pertanto non è possibile fornire tale funzione messa in discussione come una caratteristica del sistema di comunicazione stesso (a volte una versione incompleta della funzione fornita dal sistema di comunicazione può risultare utile come potenziamento delle prestazioni).

Questa linea di pensiero contraria all'implementazione delle funzioni nei livelli bassi viene chiamata "end-to-end argument".

Un esempio che illustra l'argomento end-to-end è quello del trasferimento dati affidabile. Poiché i pacchetti possono essere persi all'interno della rete (per esempio, un router con un pacchetto in coda potrebbe bloccarsi anche in assenza di un buffer overflow o una parte della rete in cui un pacchetto è in coda potrebbe scollegarsi a causa di errori di collegamento), gli endpoint devono eseguire il controllo degli errori, in questo caso tramite il protocollo TCP. Come vedremo nel Capitolo 6, alcuni protocolli a livello di collegamento eseguono effettivamente un controllo locale degli errori, che però è "incompleto" e non sufficiente a fornire un trasferimento dati affidabile end-to-end. Quindi il trasferimento dati affidabile deve essere implementato end to end.

La RFC 1958 include deliberatamente solo due riferimenti, entrambi "articoli fondamentali sull'architettura di Internet". Uno di questi è lo stesso articolo sull'end-to-end [Saltzer 1984]; il secondo articolo [Clark 1988] discute la filosofia di progettazione dei protocolli Internet DARPA. Entrambi sono letture altamente consigliate per chiunque sia interessato all'architettura Internet. Il seguito è rappresentato da [Clark 1988] e [Blumenthal 2001; Clark 2005] che riconsiderano l'architettura di Internet alla luce dell'ambiente molto più complesso in cui deve funzionare l'Internet di oggi.

trattato sia l'inoltro IP tradizionale, basato sull'indirizzo di destinazione del datagramma, sia l'inoltro generalizzato, nel quale l'inoltro e le altre funzioni possono essere effettuate utilizzando il valore di alcuni campi dell'intestazione del datagramma; abbiamo anche visto la versatilità di questo ultimo approccio. Abbiamo anche studiato nel dettaglio i protocolli IPv4 e IPv6 e l'indirizzamento Internet, scoprendo quanto sia molto più profondo, delicato e interessante di quanto ci aspettassimo. Abbiamo completato la trattazione con lo studio dei dispositivi middlebox e con un'ampia discussione dell'architettura di Internet.

Siamo ora pronti per immergerci nel Capitolo 5 nel piano di controllo del livello di rete!

## Domande e problemi

### Domande di revisione

#### PARAGRAFO 4.1

- R1.** Ripassiamo la terminologia adottata nel testo. Il pacchetto a livello di trasporto è detto *segmento* e a livello di collegamento *frame*. Qual è il nome a livello di rete? Si ricordi che sia i router che gli switch di livello 2 sono chiamati *packet switch*. Qual è la differenza fondamentale tra router e switch di livello di collegamento?
- R2.** Abbiamo visto che le funzionalità del livello di rete possono essere divise in funzionalità del piano dei dati e funzionalità del piano di controllo. Quali sono le principali funzioni del piano dei dati? Quali sono le principali funzioni del piano di controllo?
- R3.** Abbiamo fatto una distinzione tra la funzione di inoltro e la funzione di instradamento effettuate nel livello di rete. Quali sono le differenze chiave tra instradamento e inoltro (*routing and forwarding*)?
- R4.** Qual è il ruolo della tabella di inoltro in un router?

- R5.** Abbiamo detto che il modello di servizio del livello di rete “definisce le caratteristiche del trasporto end-to-end di pacchetti tra host di invio e di ricezione”. Qual è il modello di servizio del livello di rete di Internet? Quali garanzie offre il modello di servizio di Internet riguardo la consegna di datagrammi host-to-host?

#### PARAGRAFO 4.2

- R6.** Nel Paragrafo 4.2 abbiamo visto che un router consiste di porte di ingresso, porte di uscita, struttura di commutazione e processore di instradamento. Quali di questi sono implementati in hardware e quali in software? Perché? Tornando alla nozione del piano dei dati e del piano di controllo nel livello di rete, quali sono implementati in hardware e quali in software? Perché?
- R7.** Perché nelle porte di ingresso di un router ad alta velocità è memorizzata una copia della tabella di inoltro?
- R8.** Che cosa si intende per inoltro basato sulla destinazione? Come differisce dall'inoltro generalizzato? Quale dei due approcci viene adottato nelle SDN?

**R9.** Supponiamo che un pacchetto in entrata abbia corrispondenza con due o più voci nella tabella di inoltro di un router. Quale regola applica un router per determinare la porta di uscita nell'inoltro tradizionale basato sulla destinazione?

**R10.** Descrivete sinteticamente i tre tipi di struttura di commutazione presentati nel Paragrafo 4.2. Quale fra questi, se esiste, può inviare più pacchetti in parallelo attraverso la struttura di commutazione?

**R11.** Spiegate come si può verificare la perdita di pacchetti alle porte di ingresso di un router e come questo fenomeno possa essere eliminato (senza utilizzare buffer infiniti).

**R12.** Descrivete come si può verificare perdita di pacchetti alle porte di uscita di un router. Si può evitare tale perdita aumentando la velocità della struttura di commutazione?

**R13.** Che cos'è il blocco HOL? Si verifica sulle porte di ingresso o di uscita?

**R14.** Nel Paragrafo 4.2 abbiamo trattato diversi scheduling dei pacchetti: FIFO, con priorità, Round Robin (RR) e WFQ. Quali di questi garantisce che i pacchetti partano nello stesso ordine in cui sono arrivati?

**R15.** Fornite un esempio che mostri perché un operatore di rete dovrebbe voler dare priorità a una classe di pacchetti rispetto a un'altra classe di pacchetti.

**R16.** Qual è la differenza essenziale tra lo scheduling dei pacchetti RR e WFQ? Esiste un caso in cui si comportano esattamente allo stesso modo? Suggerimento: considerate i pesi WFQ.

#### PARAGRAFO 4.3

**R17.** Supponete che l'Host A mandi all'host B un segmento TCP incapsulato in un datagramma IP. Come può sapere il livello di rete dell'Host B di dover passare il segmento (ossia il payload del datagramma) a TCP anziché a UDP o a un altro protocollo di livello superiore?

**R18.** Quale campo nell'intestazione IP può essere utilizzato per garantire che il pacchetto non venga inoltrato per più di  $N$  router?

**R19.** La checksum di Internet viene usata sia nei segmenti a livello di trasporto sia nei datagrammi a livello di rete (rispettivamente nelle intestazioni UDP e TCP mostrate nelle Figure 3.7 e 3.29). Considerate un segmento a livello di trasporto encapsulato in un datagramma IP (intestazione IP, Figura 4.17). Le checksum dell'intestazione del segmento e in quella del datagramma sono calcolate su qualche bit comune nel datagramma IP? Motivate la risposta.

**R20.** Quando un datagramma grande viene frammentato in datagrammi più piccoli, questi ultimi dove vengono riassemblati in un unico datagramma?

**R21.** I router hanno indirizzi IP? Se sì, quanti?

**R22.** Qual è l'equivalente binario dei 32 bit dell'indirizzo IP 223.1.3.27?

**R23.** Accedete a un host che utilizza DHCP per ottenere la maschera di rete, il router di default e il suo indirizzo IP e quello del server DNS locale. Elencate questi valori.

**R24.** Ipotizzate che ci siano tre router tra un host sorgente e uno di destinazione. Ignorando la frammentazione, quante interfacce attraverserà un datagramma IP? Quante tabelle di inoltro saranno consultate per recapitare il datagramma?

**R25.** Supponete che un'applicazione generi blocchi di 40 byte di dati ogni 20 ms e che i blocchi siano encapsulati in un segmento TCP e poi in un datagramma IP. Quale percentuale di datagramma sarà costituita da overhead e quale da dati applicativi?

**R26.** Ipotizzate: (1) di comprare un router wireless e di collegarlo al vostro modem; (2) che il vostro ISP assegna dinamicamente un indirizzo IP al vostro router wireless; (3) di avere in casa 5 PC che usano 802.11 per connettersi in modalità wireless al vostro router. Quanti indi-

rizzi IP vengono assegnati ai 5 PC? Il router wireless utilizza NAT? Perché?

**R27.** Che cosa si intende per "aggregazione di indirizzi"? Quale ne è l'utilità?

**R28.** Che cosa si intende per protocollo "plug-and-play" o "zero-conf"?

**R29.** Che cos'è un indirizzo di una rete privata? Dovrebbe essere visibile all'Internet pubblica? Discutete tale problematica.

**R30.** Confrontate ed evidenziate differenze e similitudini nei campi dell'intestazione IPv4 e IPv6.

**R31.** Quando un datagramma IPv6 viene trasportato tramite tunneling attraverso router IPv4, IPv6 tratta i tunnel IPv4 come protocolli a livello di collegamento? Motivate la risposta.

#### PARAGRAFO 4.4

**R32.** In che modo l'inoltro generalizzato differisce dall'inoltro basato sulla destinazione?

**R33.** Qual è la differenza tra la tabella di inoltro dell'inoltro basato sulla destinazione trattato nel Paragrafo 4.1 e la tabella dei flussi di OpenFlow trattata nel Paragrafo 4.4?

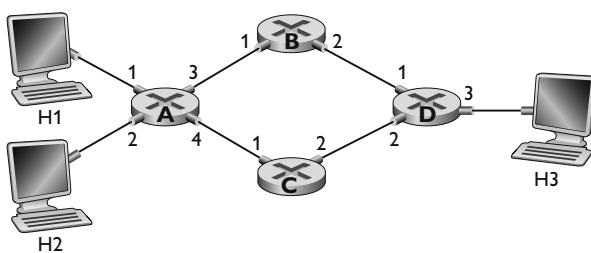
**R34.** Che cosa si intende per operazione di "match-action" di un router o di uno switch? Nel caso di inoltro basato sulla destinazione, che cosa viene confrontato e quale azione viene intrapresa? Nel caso di SDN, nominate tre campi che possono essere confrontati e tre azioni che possono essere intraprese.

**R35.** Nominate tre campi dell'intestazione di un datagramma IP che possono essere confrontati nell'inoltro generalizzato di OpenFlow 1.0. Quali sono tre campi dell'intestazione del datagramma IP che non possono essere confrontati in OpenFlow?

## Problemi

**P1.** Considerate la rete rappresentata di seguito.

- (a) Mostrate la tabella di inoltro del router A progettata in modo che tutto il traffico destinato all'Host H3 venga inoltrato attraverso l'interfaccia 3.
- (b) Scrivete una tabella di inoltro del router A tale che tutto il traffico da H1 destinato all'host H3 sia inoltrato attraverso l'interfaccia 3, mentre quello da H2 destinato all'host H3 sia inoltrato attraverso l'interfaccia 4.



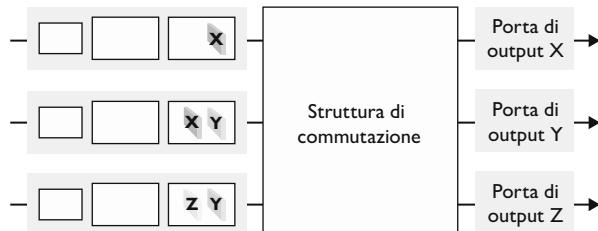
**P2.** Supponete che due pacchetti arrivino a due diverse porte di input di un router, esattamente nello stesso momento. Supponete inoltre che non vi siano altri pacchetti nel router.

- (a) Supponete che i due pacchetti debbano essere inoltrati a due diverse porte di output. È possibile inoltrare i due pacchetti attraverso la struttura di commutazione nello stesso istante se essa usa un bus condiviso?
- (b) Supponete che i due pacchetti debbano essere inoltrati a due diverse porte di output. È possibile inoltrare i due pacchetti attraverso la struttura di commutazione nello stesso istante se essa usa la memoria?
- (c) Supponete che i due pacchetti debbano essere inoltrati alla stessa porta di output. È possibile inoltrare i due pacchetti attraverso la struttura di commutazione nello stesso istante se essa usa un crossbar?

**P3.** Come visto nel Paragrafo 4.2.4, se  $R_{switch}$  è  $N$  volte più veloce di  $R_{line}$  l'accodamento sulle porte di input è trascurabile anche quando tutti i pacchetti devono es-

sere inoltrati sulla stessa porta di output. Supponiamo ora che  $R_{switch} = R_{line}$ , ma che tutti i pacchetti debbano essere inoltrati su porte di output differenti. Sia  $D$  il tempo di trasmissione del pacchetto. Esprimete il ritardo massimo di accodamento in input del pacchetto se la struttura di commutazione è (a) a memoria, (b) a bus o (c) a crossbar.

- P4.** Considerate lo switch mostrato di seguito. Supponete che tutti i datagrammi abbiano la stessa lunghezza fissa, che lo switch operi in modo sincrono e a slot temporali e che in uno slot un datagramma possa essere trasferito da una porta di input a una di output. La struttura di comunicazione è a crossbar, tale che solo un datagramma per volta possa essere trasferito a una data porta di output in uno slot temporale ma che, in tale slot, porte di output diverse possono ricevere datagrammi da parte di input differenti. Qual è il numero minimo di slot temporali necessari per trasferire i pacchetti mostrati dalle porte di input a quelle di output, assumendo l'ordine di scheduling che volete sulla coda di input, vale a dire che non ha bisogno di un blocco HOL? Qual è il massimo numero di slot necessari nel caso peggiore di scheduling, assumendo che una coda in ingresso non vuota non sia mai bloccata?



- P5.** Supponete che la policy di pianificazione WFQ venga applicata a un buffer che supporta tre classi i cui pesi siano 0,5, 0,25 e 0,25.

- Supponiamo che ogni classe abbia un numero elevato di pacchetti nel buffer. In quale sequenza potrebbero essere servite le tre classi per ottenere tali pesi WFQ? (Per la programmazione round robin, una sequenza naturale è 123123123...).
- Supponiamo che le classi 1 e 2 abbiano un gran numero di pacchetti nel buffer e non vi siano pacchetti di classe 3. In quale sequenza potrebbero essere servite le tre classi per ottenere tali pesi WFQ?

- P6.** Considerate la figura in fondo alla pagina a fianco e rispondete alle seguenti domande.

- Considerate un servizio FIFO, indicate il tempo in cui i pacchetti da 2 a 12 lasciano la coda. Indicate qual è il ritardo tra l'arrivo e l'inizio dello slot in cui viene trasmesso ogni pacchetto e il ritardo medio calcolato su tutti i 12 pacchetti.
- Considerate ora un servizio prioritario e supponete che i pacchetti numerati con numero dispari abbiano priorità alta e i pacchetti numerati con numero pari abbiano priorità bassa. Indicate il tempo in cui i pacchetti da 2 a 12 lasciano la coda. Indicate qual è il ritardo tra l'arrivo e l'inizio dello slot in cui viene trasmesso ogni pacchetto e il ritardo medio calcolato su tutti i 12 pacchetti.

- Considerate ora un servizio round robin. Supponete che i pacchetti 1, 2, 3, 6, 11 e 12 provengano dalla classe 1 e i pacchetti 4, 5, 7, 8, 9 e 10 provengano dalla classe 2. Indicate il tempo in cui i pacchetti da 2 a 12 lasciano la coda. Indicate qual è il ritardo tra l'arrivo e l'inizio dello slot in cui viene trasmesso ogni pacchetto e il ritardo medio calcolato su tutti i 12 pacchetti.

- Considerate ora il servizio Weighted Fair Queuing (WFQ). Supponete che i pacchetti con numeri dispari provengano dalla classe 1, quelli con numeri pari provengano dalla classe 2. La classe 1 ha un peso WFQ di 2, mentre la classe 2 ha un peso WFQ di 1. Poiché potrebbe non essere possibile ottenere una pianificazione WFQ idealizzata come descritta nel testo, indicate perché avete scelto che uno specifico pacchetto entri nel servizio per ogni time slot. Indicate qual è il ritardo tra l'arrivo e l'inizio dello slot in cui viene trasmesso ogni ritardo e il ritardo medio calcolato su tutti i 12 pacchetti.
- Che cosa notate sul ritardo medio in tutti e quattro i casi (FIFO, RR, priorità e WFQ)?

- P7.** Considerate ancora la figura in fondo alla pagina a fianco relativa al problema P6.

- Assumete un servizio prioritario, con i pacchetti 1, 4, 5, 6 e 11 ad alta priorità, mentre i pacchetti rimanenti hanno una priorità bassa. Indicate gli slot temporali in cui i pacchetti da 2 a 12 escono dalla coda.
- Supponete ora che venga utilizzato il servizio round robin, con i pacchetti 1, 4, 5, 6 e 11 appartenenti a una classe di traffico e i restanti pacchetti appartenenti alla seconda classe di traffico. Indicate gli slot temporali in cui i pacchetti da 2 a 12 escono dalla coda.
- Supponete ora che venga utilizzato il servizio WFQ, con i pacchetti 1, 4, 5, 6 e 11 appartenenti a una classe di traffico e i restanti pacchetti appartenenti alla seconda classe di traffico. La classe 1 ha un peso WFQ di 1, mentre la classe 2 ha un peso WFQ di 2 (si noti che questi pesi sono diversi rispetto alla precedente domanda). Indicate gli slot temporali in cui i pacchetti da 2 a 12 escono dalla coda.

- P8.** Considerate una rete con indirizzi degli host a 32 bit. Supponete che un router abbia quattro collegamenti, 0, 1, 2 e 3, e che i pacchetti debbano essere inoltrati verso le interfacce di collegamento come segue:

| Intervallo<br>degli indirizzi<br>di destinazione                                                              | Interfaccia<br>di collegamento |
|---------------------------------------------------------------------------------------------------------------|--------------------------------|
| da 11100000 00000000 00000000 00000000                                                                        | 0                              |
| a 11100000 00111111 11111111 11111111                                                                         |                                |
| da 11100000 01000000 00000000 00000000                                                                        | 1                              |
| a 11100000 01000000 11111111 11111111                                                                         |                                |
| da 11100000 01000001 00000000 00000000                                                                        | 2                              |
| a 11100001 01111111 11111111 11111111                                                                         |                                |
| altrimenti                                                                                                    | 3                              |
| (a) Scrivete una tabella di inoltro con quattro righe, che utilizza il confronto a prefisso più lungo e inol- |                                |

tra correttamente i pacchetti verso le interfacce di collegamento.

- (b) Descrivete come la vostra tabella di inoltro determini l'interfaccia di collegamento corretta per i datagrammi con i seguenti indirizzi di destinazione:

11001000 10010001 01010001 01010101  
 11100001 01000000 11000011 00111100  
 11100001 10000000 00010001 01110111

- P9.** Considerate una rete con indirizzi degli host a 8 bit. Supponete che un router utilizzi il confronto a prefisso più lungo e abbia la seguente tabella di inoltro:

| Prefisso | Interfaccia |
|----------|-------------|
| 00       | 0           |
| 010      | 1           |
| 011      | 2           |
| 10       | 2           |
| 11       | 3           |

Per ciascuna delle quattro interfacce, fornite l'intervallo di indirizzi degli host di destinazione e il numero di indirizzi nell'intervallo.

- P10.** Considerate una rete con indirizzi degli host a 8 bit. Supponete che un router utilizzi il confronto a prefisso più lungo e abbia la seguente tabella di inoltro:

| Prefisso   | Interfaccia |
|------------|-------------|
| 1          | 0           |
| 10         | 1           |
| 111        | 2           |
| altrimenti | 3           |

Per ciascuna delle quattro interfacce fornite l'intervallo associato di indirizzi host di destinazione e il numero di indirizzi nell'intervallo.

- P11.** Considerate un router che connette tre sottoreti: sottorete 1, sottorete 2 e sottorete 3. Supponete che tutte le interfacce nelle sottoreti abbiano il prefisso 223.1.17/24

e che la sottorete 1 debba servire almeno 60 interfacce, la sottorete 2 debba servire almeno 90 interfacce, mentre la sottorete 3 debba servire almeno 12 interfacce. Scrivete tre indirizzi di rete (nella forma a.b.c.d/x) che soddisfino tali vincoli.

- P12.** Nel Paragrafo 4.2.2 viene fornito un esempio di tabella di inoltro (che utilizza il confronto a prefisso più lungo). Riscrivete questa tabella con la notazione a.b.c.d/x invece di quella binaria.

- P13.** Nel Problema P8 vi è stato chiesto di scrivere una tabella di inoltro (con confronto a prefisso più lungo). Riscrivete la tabella con la notazione a.b.c.d/x invece di quella binaria.

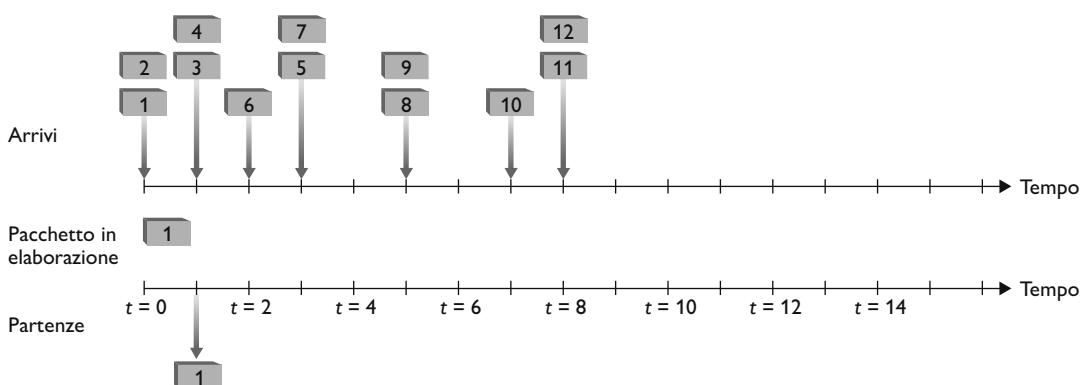
- P14.** Considerate una sottorete con prefisso 128.119.40.128/26 e fornite un indirizzo IP (di forma xxx.xxx.xxx.xxx) che può essere assegnato a questa rete. Supponete che un ISP possieda il blocco di indirizzi 128.119.40.64/26 a partire dal quale voglia creare quattro sottoreti, e che ciascun blocco abbia lo stesso numero di indirizzi IP. Quali sono i prefissi (di forma a.b.c.d/x) per le quattro sottoreti?

- P15.** Considerate la topologia della Figura 4.20. Denotate in senso orario le tre sottoreti con host come reti A, B e C. Chiamate D, E e F le sottoreti senza host.

- (a) Assegnate indirizzi di rete alle sei sottoreti, con i seguenti vincoli: (1) tutti gli indirizzi devono essere allocati da 214.97.254/23; (2) la sottorete A deve avere indirizzi a sufficienza per supportare 250 interfacce; (3) le sottoreti B e C devono avere indirizzi a sufficienza per supportarne 120 ciascuna. Ovviamente, le sottoreti D, E e F dovrebbero essere in grado di supportare 2 interfacce. Per ciascuna sottorete, l'assegnamento deve assumere la forma a.b.c.d/x oppure a.b.c.d/x – e.f.g.h/y.

- (b) Utilizzando la vostra risposta alla parte (a), scrivete le tabelle di inoltro (usando il confronto a prefisso più lungo) per i tre router.

- P16.** Usate il servizio whois dell'American Registry for Internet Numbers (<http://www.arin.net/whois>) per determinare il blocco di indirizzi IP di tre università americane.<sup>2</sup> È possibile utilizzare i servizi whois per determinare con certezza la posizione geografica di un indirizzo IP specifico? Usateli per determinare la posizione del web server di ognuna di queste università.



- P17.** Ipotizzate che i datagrammi siano limitati a 1500 byte (intestazioni incluse) tra l'host sorgente *A* e la destinazione *B*. Assumendo un'intestazione IP da 20 byte, quanti datagrammi sono necessari per inviare un MP3 da 5 milioni di byte? Spiegate come siete giunti alla risposta.
- P18.** Considerate lo schema di rete della Figura 4.25. Supponete che l'ISP assegna al router l'indirizzo 24.34.112.235 e che l'indirizzo della rete domestica sia 192.168.1/24.
- Assegnate indirizzi a tutte le interfacce nella rete domestica.
  - Supponete che gli host abbiano due connessioni TCP in uscita, verso la porta 80 dell'host 128.119.40.86. Fornite le sei righe corrispondenti nella tabella di traduzione NAT.
- P19.** Supponete di essere interessati a rilevare il numero di host dietro a un NAT. Osservate che il livello IP marca ogni pacchetto con un numero identificativo sequenziale. Il numero di identificazione del primo pacchetto IP generato da un host è un numero casuale, mentre i numeri identificativi dei pacchetti IP successivi sono assegnati in modo sequenziale. Supponete che tutti pacchetti IP generati dagli host dietro al NAT vengano inviati alla rete esterna.
- Sulla base di questa osservazione e assumendo che possiate spiare i pacchetti inviati dal NAT verso la rete esterna, sapete descrivere una semplice tecnica in grado di rilevare il numero di host dietro a un NAT?
  - La vostra tecnica funzionerebbe anche se i numeri di identificazione fossero assegnati non in modo sequenziale, ma in modo casuale? Giustificate la vostra risposta.
- P20.** In questo problema esploreremo l'impatto del NAT sulle applicazioni P2P. Ipotizzate che: (1) un peer con nome utente Alice scopra attraverso le query che un peer con nome utente Bob ha un file che vuole scaricare; (2) Alice e Bob si trovano dietro a un router NAT; cercate di individuare una procedura che gli consenta di stabilire una connessione TCP con Bob senza una configurazione NAT specifica per l'applicazione. Se incontrate delle difficoltà, spiegatene il motivo.
- P21.** Considerate la rete SDN OpenFlow rappresentata nella Figura 4.30. Supponete che il comportamento di inoltro desiderato per i datagrammi che arrivano a *s2* sia il seguente:
- i datagrammi che arrivano alla porta 1 provenienti dagli host *h5* o *h6* e aventi come destinazione gli host *h1* o *h2* devono essere inoltrati alla porta di uscita 2;
  - i datagrammi che arrivano alla porta 2 provenienti dagli host *h1* o *h2* e aventi come destinazione gli host *h5* o *h6* devono essere inoltrati alla porta di uscita 1;
  - i datagrammi che arrivano alla porta 1 o 2 provenienti dagli host *h1* o *h2* e aventi come destinazione gli host *h3* o *h4* devono essere inoltrati all'host specificato;
  - gli host *h3* o *h4* devono potersi scambiare datagrammi.
- Specificate le occorrenze della tabella dei flussi di *s2* in modo da implementare tale comportamento.
- P22.** Considerate nuovamente la rete SDN OpenFlow mostrata nella Figura 4.30. Supponete che il comportamento di inoltro desiderato per i datagrammi che arrivano a *s2* da *h3* o *h4* sia il seguente:
- i datagrammi provenienti dall'host *h3* e aventi come destinazione gli host *h1*, *h2*, *h5* e *h6* devono essere inoltrati in senso orario nella rete;
  - i datagrammi provenienti dall'host *h4* e aventi come destinazione gli host *h1*, *h2*, *h5* e *h6* devono essere inoltrati in senso antiorario nella rete;
- Specificate le occorrenze della tabella dei flussi di *s2* in modo da implementare tale comportamento.
- P23.** Considerate lo stesso scenario dell'esercizio P21. Scrivete le occorrenze della tabella dei flussi ai packet switch *s1* e *s3* in modo che i datagrammi aventi come sorgente *h3* o *h4* vengano inoltrati direttamente all'host di destinazione specificato nel campo indirizzo IP dell'intestazione del datagramma. Suggerimento: le vostre regole nella tabella di inoltro dovrebbero considerare i casi in cui un datagramma in arrivo sia destinato a un host attaccato direttamente o che debba essere inoltrato a un router vicino per essere infine consegnato.
- P24.** Considerate la rete SDN OpenFlow rappresentata nella Figura 4.30. Supponete che *s2* debba funzionare come firewall. Specificate, per ognuno dei seguenti comportamenti, la tabella dei flussi di *s2* che lo implementa per consegnare i datagrammi destinati a *h3* e *h4*. Non è necessario che specificiate il comportamento di inoltro in *s2* per datagrammi destinati ad altri router.
- Solo il traffico da *h1* e *h6* va consegnato agli host *h3* e *h4*, bloccando in questo modo il traffico da *h2* e *h5*.
  - Solo il traffico TCP è autorizzato a essere consegnato agli host *h3* e *h4*, bloccando così il traffico UDP.
  - Solo il traffico destinato a *h3* deve essere consegnato, bloccando così tutto il traffico diretto a *h4*.
  - Solo il traffico UDP da *h1* a *h3* va consegnato, bloccando così tutto il resto del traffico.
- P25.** Considerate la pila dei protocolli Internet mostrata nelle Figure 1.23 e 4.31. Secondo voi il protocollo ICMP è un protocollo del livello di rete o del livello di trasporto? Giustificate la risposta.

## Esercitazione Wireshark: IP

Sulla piattaforma MyLab del testo trovate le esercitazioni di laboratorio sul protocollo IP e il formato dei datagrammi IP in particolare.

# Livello di rete: il piano di controllo

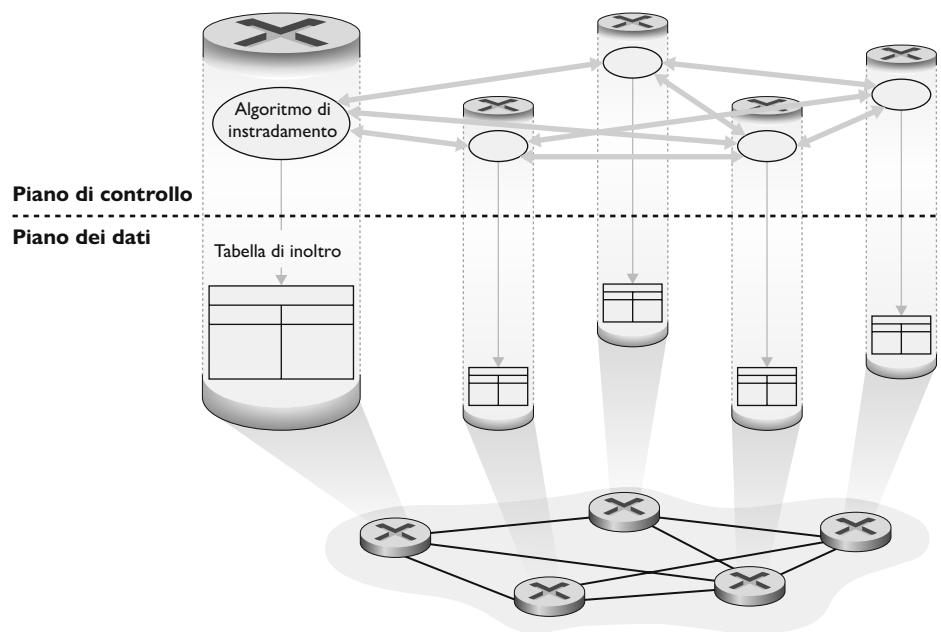
In questo capitolo completeremo il nostro viaggio attraverso il livello di rete trattando il **piano di controllo** (*control plane*), la logica di rete globale che controlla non solo come i datagrammi vengono inoltrati tra i router lungo i percorsi end-to-end dall'host sorgente all'host destinatario, ma anche come le componenti e i servizi del livello di rete vengono configurati e gestiti. Nel Paragrafo 5.2 tratteremo gli algoritmi di instradamento tradizionali per calcolare i percorsi a costo minimo in un grafo; tali algoritmi sono alla base dei due protocolli di instradamento di Internet: OSPF e BGP, che tratteremo rispettivamente nei Paragrafi 5.3 e 5.4. Come vedremo, OSPF è il protocollo di instradamento che opera all'interno della rete di un singolo ISP. BGP è il protocollo di instradamento che serve a interconnettere tutte le reti in Internet; BGP è quindi visto come la colla che tiene insieme Internet. Tradizionalmente, i protocolli di instradamento del piano di controllo sono stati implementati insieme alle funzioni di inoltro del piano dei dati, in modo monolitico, all'interno di ogni router. Come abbiamo visto nell'introduzione del Capitolo 4, SDN opera una chiara separazione tra il piano dei dati e il piano di controllo, implementando le funzioni di quest'ultimo in un controller separato e remoto rispetto alle componenti di inoltro dei router che controlla. Tratteremo i controller SDN nel Paragrafo 5.5.

Nei Paragrafi 5.6 e 5.7 tratteremo i meccanismi di gestione di una rete IP: ICMP (*Internet Control Message Protocol*) e SNMP (*Simple Network Management Protocol*).

## 5.1 Introduzione

Rivediamo brevemente il contesto nel quale trattare il piano di controllo di una rete riconsiderando le Figure 4.2 e 4.3. Avevamo visto che le tabelle di inoltro, nel caso di inoltro basato sulla destinazione, e le tabelle dei flussi, nel caso di inoltro generalizzato, sono gli elementi principali che collegano il piano dei dati e il piano di controllo del livello di rete. Abbiamo imparato che tali tabelle specificano il comportamento di inoltro nel piano dei dati locale di un router. Abbiamo inoltre visto che, nel caso di inoltro generalizzato, le azioni intraprese (Paragrafo 4.4.2) potrebbero includere non solo l'inoltro di un pacchetto verso una porta di uscita del router, ma anche lo scarso del pacchetto, la duplicazione del pacchetto e la riscrittura dei campi dell'intestazione del pacchetto di livello 2, 3 e 4.

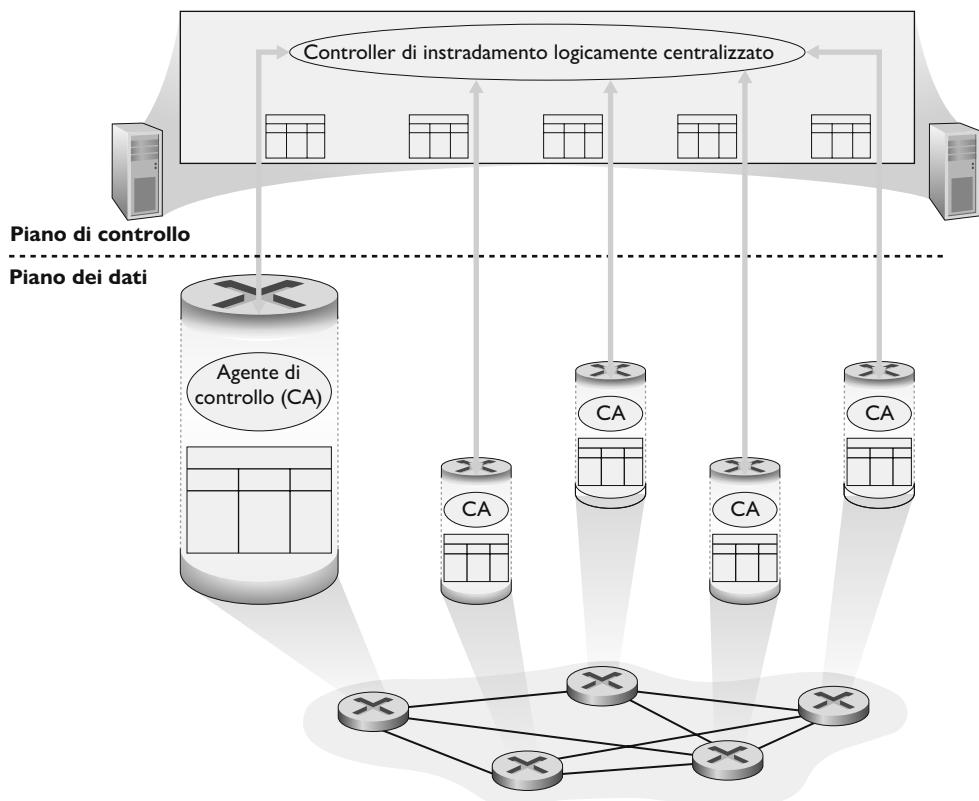
**Figura 5.1** Controllo per router: componenti individuali che eseguono l'algoritmo di instradamento interagiscono nel piano di controllo.



In questo capitolo studieremo come le tabelle di inoltro e dei flussi vengono calcolate, mantenute e installate. Nella nostra introduzione al livello di rete nel Paragrafo 4.1 abbiamo imparato che esistono due possibili approcci.

- *Controllo per router*. La Figura 5.1 illustra il caso in cui l'algoritmo di instradamento viene eseguito su ogni singolo router, all'interno del quale vengono effettuate sia le funzioni di inoltro (piano dei dati) che quelle di instradamento (piano di controllo). Ogni router ha una componente di instradamento che comunica con le componenti di instradamento degli altri router per calcolare la propria tabella di inoltro. Questo approccio di controllo locale per router è stato usato in Internet per decenni; i protocolli OSPF e BGP, che studieremo nei Paragrafi 5.3 e 5.4, sono basati su tale approccio.
- *Controllo logicamente centralizzato*. La Figura 5.2 illustra il caso in cui un controller logicamente centralizzato calcola e distribuisce le tabelle di inoltro che devono essere utilizzate da ogni router. Come visto nei Paragrafi 4.4 e 4.5 l'astrazione del match-action permette che il router effettui l'inoltro IP tradizionale e molte altre funzioni (distribuzione del carico, firewalling e NAT) che prima erano implementate in middlebox separati.

Il controller interagisce con l'agente di controllo (CA, *Control Agent*) in ogni router tramite un protocollo che configura e gestisce la tabella dei flussi del router. Tipicamente, il CA ha funzionalità minime: comunica con il controller ed esegue quello che il controller gli ordina. Al contrario degli algoritmi di instradamento della Figura 5.1, gli agenti di controllo non interagiscono direttamente tra di loro e non partecipano attivamente all'elaborazione della tabella di inoltro. Questa è una distinzione chiave tra il controllo locale e il controllo logicamente centralizzato.



**Figura 5.2** Controllo logicamente centralizzato: un controller distinto e remoto interagisce con gli agenti di controllo locali (CA, control agent).

Per controllo logicamente centralizzato [Levin 2012] intendiamo un servizio di controllo dell’instradamento a cui si accede come se fosse un singolo punto centrale di servizio, anche se il servizio probabilmente viene implementato su più server per ragioni di resistenza ai guasti e scalabilità delle prestazioni. Come vedremo nel Paragrafo 5.5, SDN adotta il controller logicamente centralizzato, un approccio sempre più diffuso. Google usa SDN per controllare i router della sua rete WAN interna, B4, che interconnette i suoi data center [Jain 2013]. SWAN [Hong 2013] di Microsoft Research usa un controllo logicamente centralizzato per gestire l’instradamento e l’inoltro tra una WAN e una rete di data center. Gli ISP più importanti, tra i quali ActiveCore di COMCAST e Access 4.0 di Deutsche Telecom sono impegnati nell’integrazione di SDN nelle loro reti e come vedremo nel Capitolo 8 (disponibile in inglese sulla piattaforma MyLab), il controllo SDN è centrale anche nelle reti cellulari 4G/5G. [AT&T 2019] osserva: “... SDN, non è una visione, un obiettivo o una promessa. È una realtà. Entro la fine del prossimo anno, il 75% delle funzioni della nostra rete saranno completamente virtualizzate e controllate da software”. China Telecom e China Unicom utilizzano SDN sia all’interno dei data center sia tra i data center [Li 2015].

## 5.2 Algoritmi di instradamento

In questo paragrafo studieremo gli **algoritmi di instradamento** (*routing algorithm*), il cui scopo è determinare i percorsi o cammini<sup>1</sup>, tra le sorgenti e i destinatari, attraverso la rete di router. Tipicamente, il percorso migliore è quello che ha costo minimo (che in verità possono essere più di uno a ugual costo). Tuttavia, vedremo che nella pratica giocano un ruolo anche problematiche di interesse concreto quali le policy (per esempio una regola potrebbe essere che il router  $x$ , appartenente all’organizzazione  $Y$ , non debba inoltrare pacchetti che abbiamo come sorgente la rete appartenente all’organizzazione  $Z$ ). Si noti che è sempre necessario avere una sequenza ben definita di router che il pacchetto attraversa viaggiando dall’host sorgente all’host destinatario, sia che il piano di controllo adotti un approccio per router sia che adotti un approccio logicamente centralizzato. Quindi, gli algoritmi di instradamento che elaborano tali percorsi sono di importanza basilare, e possiamo considerarli come candidati nella lista dei 10 concetti di fondamentale importanza nel networking.

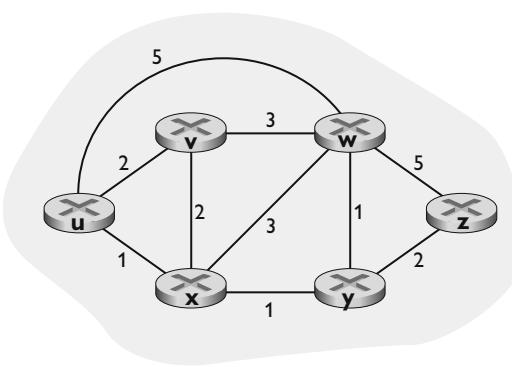
Per formulare i problemi di instradamento si utilizza un **grafo**. Ricordiamo che un grafo  $G = (N, E)$  è un insieme  $N$  di nodi e un insieme  $E$  di archi (*edge*), ove ciascun arco collega una coppia di nodi di  $N$ . Nel contesto dell’instradamento a livello di rete, i nodi del grafo rappresentano i router – che prendono decisioni sull’inoltro dei pacchetti – e gli archi che connettono tali nodi e rappresentano i collegamenti fisici tra i router (Figura 5.3). Per prendere visione di alcuni grafi che rappresentano reali mappe di rete si veda [CAIDA 2020]; per una trattazione sulla modellazione di Internet tramite grafi, si veda [Zegura 1997; Faloutsos 1999; Li 2004].

Come mostrato nella Figura 5.3, a un arco è anche associato un valore che ne indica il costo. In genere, ciò può riflettere la lunghezza fisica del collegamento corrispondente (per esempio, un collegamento transoceanico può avere costo superiore rispetto a un collegamento terrestre a corto raggio), la velocità del collegamento o il suo prezzo. Per i nostri scopi, considereremo i costi degli archi come dati, senza preoccuparci di come siano stati determinati (spesso viene usato il termine metrica per indicare i costi associati agli archi di una rete). Per ogni arco  $(x, y)$  tra i nodi  $x$  e  $y$  denotiamo con  $c(x, y)$  il suo costo. Se la coppia  $(x, y)$  non appartiene a  $E$ , poniamo  $c(x, y) = +\infty$ . Inoltre, consideriamo sempre e solo grafi non orientati, cioè con archi bidirezionali, ragion per cui l’arco  $(x, y)$  è uguale all’arco  $(y, x)$  e  $c(x, y) = c(y, x)$ . Inoltre, un nodo  $y$  viene detto **adiacente** o **vicino** (*neighbor*) a un nodo  $x$  se  $(x, y)$  è un arco in  $E$ .

Dato che agli archi del grafo sono assegnati dei costi, obiettivo naturale di un algoritmo di instradamento è l’individuazione dei percorsi meno costosi tra sorgenti e destinazioni. Per rendere questo problema più preciso, ricordiamo che un **percorso** in un grafo  $G = (N, E)$  è una sequenza di nodi  $(x_1, x_2, \dots, x_p)$  tali che ciascuna delle coppie  $(x_1, x_2), (x_2, x_3), \dots, (x_{p-1}, x_p)$  sia un arco appartenente a  $E$ . Il costo di un percorso  $(x_1, x_2, \dots, x_p)$  è semplicemente la somma di tutti i costi degli archi lungo il percorso, ossia  $c(x_1, x_2) + c(x_1, x_3) + \dots + c(x_{p-1}, x_p)$ . Dati

---

<sup>1</sup> Nella versione in italiano e nel contesto dell’instradamento i termini “percorso” e “cammino” sono usati come sinonimi.



**Figura 5.3** Modello astratto di grafo di una rete di calcolatori.

due nodi qualsiasi  $x$  e  $y$ , esistono più percorsi che li congiungono, ciascuno con il proprio costo. Uno o più di tali percorsi rappresenta un **percorso a costo minimo** (*least-cost path*). Il problema è quindi chiaro: determinare un percorso tra l'origine e la destinazione che abbia il costo minimo. Nella Figura 5.3, per esempio, il percorso a costo minimo tra il nodo origine  $u$  e il nodo destinazione  $w$  è  $(u, x, y, w)$ , il cui costo è 3. Si noti che se tutti gli archi del grafo hanno lo stesso costo, il percorso a costo minimo rappresenta anche il **percorso più breve** (*shortest path*), ossia il percorso con il minor numero di collegamenti tra sorgente e destinazione.

Come semplice esercizio, cercate di determinare il percorso a costo minimo tra i nodi  $u$  e  $z$  nella Figura 5.3 e riflettete per un momento su come l'avete calcolato. Nella maggior parte dei casi, avrete trovato il percorso esaminando la Figura 5.3, tracciando qualche percorso da  $u$  a  $z$  e convincendovi che il percorso scelto abbia costo minore tra tutti quelli possibili. Avete controllato tutti i 17 percorsi possibili tra  $u$  e  $z$ ? Probabilmente no. Tale calcolo è un esempio di algoritmo di instradamento centralizzato: l'algoritmo viene eseguito “in una locazione”, il vostro cervello, che ha informazione completa sulla rete. In generale, gli algoritmi di instradamento sono classificabili come centralizzati o decentralizzati.

- Un **algoritmo di instradamento centralizzato** calcola il percorso a costo minimo tra una sorgente e una destinazione avendo una conoscenza globale e completa della rete. In altre parole, l'algoritmo riceve in ingresso tutti i collegamenti tra i nodi e i loro costi. Ciò richiede che l'algoritmo in qualche modo ottenga tale informazione prima di effettuare il vero e proprio calcolo, che può essere svolto in una sola locazione quale potrebbe essere un controller logicamente centralizzato (Figura 5.2), o replicato in ognuno dei router della rete (Figura 5.1). La caratteristica distintiva, tuttavia, è che un algoritmo globale ha informazioni complete su connettività e costi. In pratica, gli algoritmi con informazioni di stato globali sono spesso detti **algoritmi link-state** (LS, o con stato del collegamento), dato che l'algoritmo deve essere consci del costo di ciascun collegamento della rete (Paragrafo 5.2.1).
- In un **algoritmo di instradamento decentralizzato** il percorso a costo minimo viene calcolato in modo distribuito e iterativo. Nessun nodo possiede informazioni complete sul costo di tutti i collegamenti di rete. Inizialmente i nodi conoscono soltanto i costi dei collegamenti a loro incidenti. Poi, attraverso

un processo iterativo e lo scambio di informazioni con i nodi adiacenti, un nodo gradualmente calcola il percorso a costo minimo verso una destinazione o un insieme di destinazioni. L'algoritmo di instradamento decentralizzato che studieremo nel Paragrafo 5.2.2 è chiamato algoritmo distance-vector (DV, o con vettore delle distanze), poiché ogni nodo elabora un vettore di stima dei costi (distanze) verso tutti gli altri nodi nella rete. Tali algoritmi decentralizzati, che prevedono scambi interattivi tra router vicini, possono essere implementati nei piani di controllo nei quali i router interagiscono direttamente, come nella Figura 5.1, mentre hanno poco senso nel caso di controllo centralizzato.

Un secondo criterio per classificare gli algoritmi di instradamento riguarda il fatto di essere statici o dinamici. Negli **algoritmi di instradamento statici** i percorsi cambiano molto raramente, spesso come risultato di un intervento umano (per esempio, la modifica manuale di una tabella di inoltro di un router). Gli **algoritmi di instradamento dinamici**, invece, determinano gli instradamenti al variare del volume di traffico o della topologia della rete. Un algoritmo dinamico può essere eseguito sia periodicamente o come conseguenza diretta di un cambiamento nella topologia o nel costo di un collegamento. Gli algoritmi dinamici rispondono meglio ai cambiamenti della rete, ma sono anche maggiormente soggetti a problemi quali l'instradamento in loop e l'oscillazione dei percorsi.

Un terzo modo per classificare gli algoritmi di instradamento si basa sull'essere più o meno sensibili al carico della rete (*load-sensitive* o *load-insensitive*). In un **algoritmo sensibile al carico** i costi dei collegamenti variano dinamicamente per riflettere il livello corrente di congestione. Se si associa un alto costo a un collegamento in una situazione di traffico, un algoritmo di instradamento tenderà a evitare di usarlo. Quando i primi algoritmi di instradamento di ARPAnet erano sensibili al carico [McQuillan 1980], si sperimentarono numerose difficoltà [Huitema 1998]. Gli attuali algoritmi di instradamento Internet (quali RIP, OSPF e BGP) sono **algoritmi insensibili al carico**, dato che il costo di un collegamento non riflette esplicitamente il suo attuale (o recente) livello di congestione.

### 5.2.1 Instradamento “link-state” (LS)

In un instradamento link-state la topologia di rete e tutti i costi dei collegamenti sono noti, ossia disponibili in input all'algoritmo. Ciò si ottiene facendo inviare a ciascun nodo pacchetti sullo stato dei suoi collegamenti a *tutti* gli altri nodi della rete. Questi pacchetti contengono identità e costi dei collegamenti connessi al nodo che li invia. In pratica (per esempio, con il protocollo OSPF trattato nel Paragrafo 5.3), questo viene spesso ottenuto tramite un algoritmo di **link-state broadcast** [Perlman 1999]. In tal modo tutti i nodi dispongono di una vista identica e completa della rete e ciascun nodo che esegue l'algoritmo LS ottiene gli stessi risultati.

L'algoritmo di calcolo dei percorsi che presentiamo associato all'instradamento link-state è noto come **algoritmo di Dijkstra**, dal nome del suo ideatore. Un algoritmo strettamente collegato è quello di Prim: si veda [Cormen 2001] per una discussione generale degli algoritmi sui grafi. L'algoritmo di Dijkstra

calcola il percorso a costo minimo da un nodo (l'origine, che chiameremo  $u$ ) a tutti gli altri nodi nella rete, è iterativo e ha le seguenti proprietà: dopo la  $k$ -esima iterazione, i percorsi a costo minimo sono noti a  $k$  nodi di destinazione e, tra i percorsi a costo minimo verso tutti i nodi di destinazione, questi  $k$  percorsi hanno i  $k$  costi più bassi. Adottiamo la seguente notazione.

- $D(v)$ : costo minimo del percorso dal nodo origine alla destinazione  $v$  per quanto concerne l'iterazione corrente dell'algoritmo.
- $p(v)$ : immediato predecessore di  $v$  lungo il percorso a costo minimo dall'origine a  $v$ .
- $N'$ : sottoinsieme di nodi contenente tutti (e solo) i nodi  $v$  per cui il percorso a costo minimo dall'origine a  $v$  è definitivamente noto.

L'algoritmo di instradamento centralizzato consiste in un passo di inizializzazione seguito da un ciclo che viene eseguito una volta per ogni nodo del grafo. Quando termina, l'algoritmo avrà calcolato il cammino minimo dal nodo origine  $u$  a tutti gli altri nodi.

### Algoritmo Dijkstra dal nodo origine $u$

```

1  Inizializzazione:
2       $N' = \{u\}$ 
3      per tutti i nodi  $v$ 
4          se  $v$  è adiacente a  $u$ 
5              allora  $D(v) = c(u,v)$ 
6          altrimenti  $D(v) = \infty$ 
7
8  Ciclo
9      determina un  $w$  non in  $N'$  tale che  $D(w)$  sia minimo
10     aggiungi  $w$  a  $N'$ 
11     aggiorna  $D(v)$  per ciascun nodo  $v$  adiacente a  $w$  e non in
N':
12          $D(v) = \min(D(v), D(w) + c(w,v))$ 
13     /* il nuovo costo verso  $v$  è il vecchio costo verso  $v$ 
oppure
14         il costo del percorso minimo noto verso  $w$  più il
costo da  $w$  a  $v$  */
15     ripeti in ciclo finché non si verifica che  $N' = N$ 

```

Consideriamo per esempio la rete nella Figura 5.3 e calcoliamo i percorsi a costo minimo da  $u$  a tutte le destinazioni. I passi dell'algoritmo sono mostrati nella Tabella 5.1, le cui righe forniscono i valori delle variabili dell'algoritmo al termine dell'iterazione. Analizziamo in dettaglio i primissimi passi.

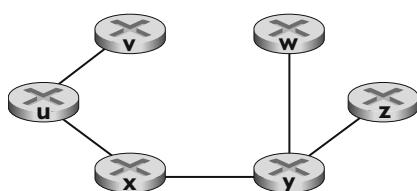
- Nel passo di inizializzazione i valori dei percorsi a costo minimo noti da  $u$  ai suoi nodi adiacenti,  $v$ ,  $w$  e  $x$ , sono posti rispettivamente a 2, 5 e 1. In particolare il costo verso  $w$  vale 5 (anche se vedremo presto che di certo esiste un percorso a costo inferiore) dato che questo è il costo del collegamento diretto (un solo hop) da  $u$  a  $w$ . I costi verso  $y$  e  $z$  sono posti a infinito dato che tali nodi non sono adiacenti a  $u$ .

**Tabella 5.1** Esecuzione dell'algoritmo Dijkstra sulla rete della Figura 5.3.

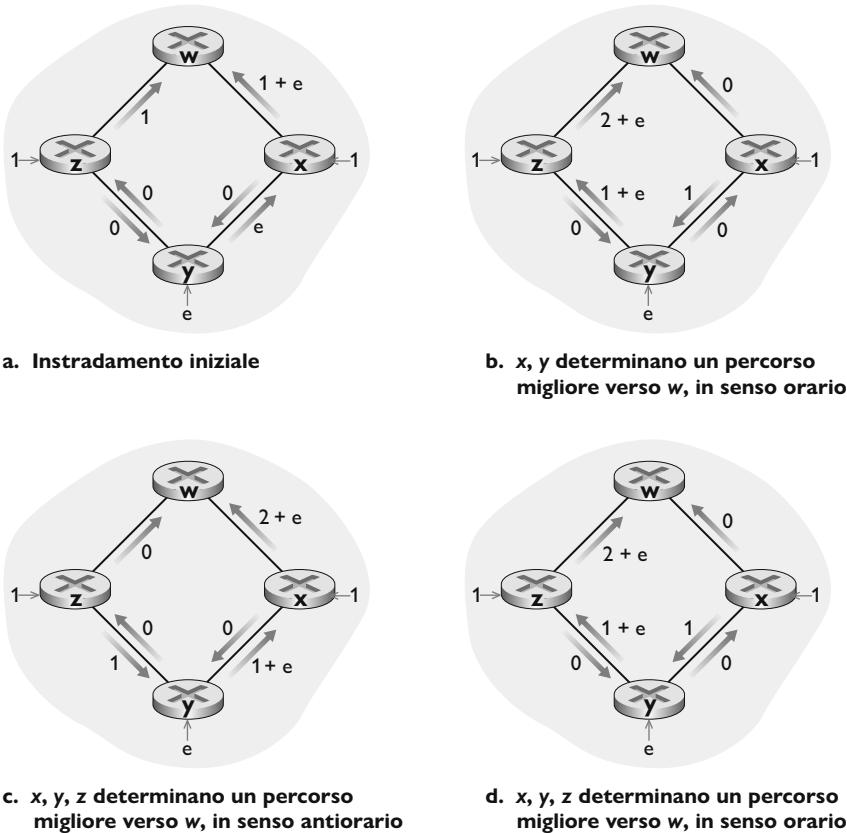
| Passo | $N'$  | $D(v), p(v)$ | $D(w), p(w)$ | $D(x), p(x)$ | $D(y), p(y)$ | $D(z), p(z)$ |
|-------|-------|--------------|--------------|--------------|--------------|--------------|
| 0     | u     | 2, u         | 5, u         | 1, u         | $\infty$     | $\infty$     |
| 1     | ux    | 2, u         | 4, x         |              | 2, x         | $\infty$     |
| 2     | uxy   | 2, u         | 3, y         |              |              | 4, y         |
| 3     | uxyv  |              | 3, y         |              |              | 4, y         |
| 4     | uxyw  |              |              |              |              | 4, y         |
| 5     | uxywz |              |              |              |              |              |

- Nella prima iterazione prendiamo in considerazione i nodi non ancora aggiunti all'insieme  $N'$  e determiniamo il nodo a costo minimo come alla fine della precedente iterazione. Tale nodo è  $x$ , di costo 1, e pertanto  $x$  viene aggiunto all'insieme  $N'$ . Viene poi eseguita la riga 12 dell'algoritmo Dijkstra per aggiornare  $D(v)$  per tutti i nodi, ottenendo i risultati mostrati nella seconda riga (Passo 1) della Tabella 5.1. Il costo del percorso verso  $v$  non è cambiato. Il costo del percorso verso  $w$  (che era 5 al termine dell'inizializzazione) passando per il nodo  $x$  è diventato 4. Viene quindi selezionato questo percorso a costo inferiore e il predecessore di  $w$  lungo il percorso minimo da  $u$  diventa  $x$ . Analogamente, il costo verso  $y$  (attraverso  $x$ ) viene calcolato essere 2, e la tabella viene aggiornata di conseguenza.
- Nella seconda iterazione si trova che i nodi  $v$  e  $y$  hanno percorsi a costo minimo (2), ne scegliamo arbitrariamente uno e aggiungiamo  $y$  all'insieme  $N'$  che ora conterrà  $u$ ,  $x$  e  $y$ . I costi verso i nodi rimanenti non ancora in  $N'$ , ossia  $v$ ,  $w$  e  $z$ , sono aggiornati dalla riga 12 dell'algoritmo Dijkstra, ottenendo i risultati mostrati nella terza riga della tabella.
- E così via...

Quando l'algoritmo Dijkstra termina, abbiamo per ciascun nodo il suo predecessore lungo il percorso a costo minimo dal nodo origine. Per ciascun predecessore abbiamo il rispettivo predecessore, e in questo modo riusciamo a costruire l'intero percorso dall'origine a tutte le destinazioni. La tabella di inoltro in un nodo, diciamo  $u$ , può pertanto essere costruita da queste informazioni memorizzando, per ciascuna destinazione, il nodo del successivo hop sul percorso a costo minimo da  $u$  alla destinazione. La Figura 5.4 mostra i percorsi a costo minimo risultanti e la tabella di inoltro di  $u$  per la rete della Figura 5.3.

**Figura 5.4** Percorso a costo minimo e tabella di inoltro per il nodo  $u$ .

| Destinazione | Link   |
|--------------|--------|
| v            | (u, v) |
| w            | (u, x) |
| x            | (u, x) |
| y            | (u, x) |
| z            | (u, x) |



**Figura 5.5** Oscillazioni con instradamento sensibile alla congestione.

Qual è la complessità computazionale di questo algoritmo? In altre parole, dati  $n$  nodi (senza contare l'origine), quanti calcoli occorre fare, nel caso peggiore, per determinare i percorsi a costo minimo dall'origine a tutte le destinazioni? Nella prima iterazione, dobbiamo cercare su tutti gli  $n$  nodi per determinare quello  $w$  non in  $N'$  avente il costo minimo. Nella seconda iterazione dobbiamo controllare  $n - 1$  nodi per determinare il costo minimo; nella terza iterazione i nodi sono  $n - 2$ , e così via. Complessivamente il numero totale di nodi da cercare in tutte le iterazioni è  $n(n + 1)/2$  e, di conseguenza, diciamo che la precedente implementazione dell'algoritmo Dijkstra ha, nel caso peggiore, una complessità di ordine quadratico:  $O(n^2)$ . Un'implementazione più sofisticata di questo algoritmo, utilizzando una struttura dati nota come heap, riesce a determinare il minimo nella riga 9 in un tempo logaritmico anziché lineare, riducendo la complessità.

Prima di completare la nostra trattazione dell'algoritmo Dijkstra, consideriamo una situazione “patologica” che può manifestarsi. La Figura 5.5 mostra una topologia di rete in cui i costi dei collegamenti sono uguali al carico trasportato sul collegamento, il che riflette il ritardo che si verificherebbe. In questo esempio, i costi dei collegamenti non sono simmetrici; in altre parole,  $c(u,v)$  risulta uguale a  $c(v,u)$  solo se il carico trasportato in entrambe le direzioni del collegamento  $(u,v)$  è lo stesso. Inoltre, il nodo  $z$  e quello  $x$  danno origine a un'unità

di traffico ciascuno verso  $w$ , mentre  $y$  invia una quantità di traffico pari a  $e$ , anche questo verso  $w$ . L'instradamento iniziale viene mostrato nella Figura 5.5(a), dove i costi dei collegamenti corrispondono alla quantità di traffico trasportato.

Nella successiva iterazione dell'algoritmo, il nodo  $y$  determina, sulla base dei costi dei collegamenti indicati nella Figura 5.5(a), che il percorso in senso orario verso  $w$  ha costo 1, mentre il percorso in senso antiorario ha costo pari a  $1 + e$ . Pertanto, il percorso a costo minimo di  $y$  verso  $w$  ora è quello in senso orario. Analogamente,  $x$  determina che il proprio nuovo percorso a costo minimo verso  $w$  è quello in senso orario, il che porta ai risultati sui costi mostrati nella Figura 5.5(b). Quando l'algoritmo viene nuovamente eseguito,  $x$ ,  $y$  e  $z$  instradano tutti il proprio traffico su percorsi in senso antiorario. La volta successiva che l'algoritmo viene eseguito, tutti i nodi  $x$ ,  $y$  e  $z$  instradano il loro traffico in senso orario.

Che cosa si può fare per evitare tali oscillazioni (che si possono verificare in qualsiasi algoritmo, non solo Dijkstra, che utilizza una metrica dei collegamenti basata sulla congestione o sul ritardo)? Una soluzione consiste nello stabilire che i costi dei collegamenti non dipendono dalla quantità di traffico trasportato: ipotesi inaccettabile dato che uno scopo dell'instradamento è evitare collegamenti troppo congestionati (per esempio, ad alto ritardo). Un'altra soluzione consiste nell'assicurarsi che non tutti i router lancino l'esecuzione dell'algoritmo nello stesso istante. Questa sembra essere una soluzione più ragionevole, dato che vorremmo che l'istanza in esecuzione dell'algoritmo non fosse la stessa su ciascun nodo anche se i router eseguissero l'algoritmo con la stessa periodicità. Aspetto interessante: i ricercatori hanno scoperto che i router in Internet si possono auto-sincronizzare tra loro [Floyd Synchronization 1994]. In altre parole, anche se mandassero inizialmente in esecuzione l'algoritmo con la stessa frequenza ma in diversi istanti, l'istanza di esecuzione dell'algoritmo potrebbe alla fine diventare e rimanere sincronizzata tra i router. Un modo per evitare tale auto-sincronizzazione è determinare su ciascun router in modo casuale l'istante di emissione dell'avviso di collegamento.

Studiamo ora l'altra fondamentale modalità di instradamento usata in pratica oggi: il distance-vector.

### 5.2.2 Instradamento “distance-vector” (DV)

Mentre l'instradamento LS usa informazioni globali, quello **distance-vector** è iterativo, asincrono e distribuito. È *distribuito* nel senso che ciascun nodo riceve parte dell'informazione da uno o più dei suoi vicini direttamente connessi, a cui, dopo aver effettuato il calcolo, restituisce i risultati. È *iterativo* nel senso che questo processo si ripete fino a quando non avviene ulteriore scambio informativo tra vicini. Aspetto interessante, l'algoritmo è anche auto-terminante: non vi è alcun segnale che il calcolo debba fermarsi, semplicemente si blocca. È *asincrono* nel senso che non richiede che tutti i nodi operino al passo con gli altri.

Prima di presentare l'algoritmo usato dall'instradamento DV sarà bene discutere un'importante relazione esistente tra costi e percorsi a costo minimo. Sia  $d_x(y)$  il costo del percorso a costo minimo dal nodo  $x$  al nodo  $y$ . Allora i costi minimi sono correlati dalla nota **formula di Bellman-Ford**:

$$d_x(y) = \min_v \{ c(x,v) + d_v(y) \} \quad (5.1)$$

dove  $\min_v$  riguarda tutti i vicini di  $x$ . Tale relazione è piuttosto intuitiva. Infatti se, dopo aver viaggiato da  $x$  a  $v$ , consideriamo il percorso a costo minimo da  $v$  a  $y$ , il costo del percorso sarà  $c(x,v) + d_v(y)$ . Dato che dobbiamo iniziare viaggiando verso qualche vicino  $v$ , il costo minimo da  $x$  a  $y$  è il minimo di  $c(x,v) + d_v(y)$  calcolato su tutti i nodi adiacenti  $v$ .

Per coloro che fossero scettici sulla validità dell'equazione, verifichiamola per il nodo origine  $u$  e per il nodo destinazione  $z$  (Figura 5.3). Il nodo origine  $u$  è adiacente ai nodi  $v$ ,  $x$  e  $w$ . Attraversando i diversi percorsi del grafo, è facile constatare che  $d_v(z) = 5$ ,  $d_x(z) = 3$  e  $d_w(z) = 3$ . Ponendo tali valori nell'Equazione 5.1 e facendo altrettanto con i costi  $c(u,v) = 2$ ,  $c(u,x) = 5$  e  $c(u,w) = 1$ , il risultato dà  $d_u(z) = \min\{2 + 5, 5 + 3, 1 + 3\} = 4$ , il che è ovviamente vero e coincide con il risultato dell'algoritmo di Dijkstra per la stessa rete.

La formula di Bellman-Ford non rappresenta solo una curiosità intellettuale, ma ha anche una significativa importanza pratica, in quanto fornisce le righe della tabella di inoltro nel nodo  $x$ . Per rendersene conto, sia  $v^*$  qualsiasi nodo vicino che minimizza l'Equazione 5.1. Allora, se il nodo  $x$  vuole inviare un pacchetto al nodo  $y$  lungo il percorso a costo minimo, dovrebbe per prima cosa inoltrarlo al nodo  $v^*$ . Pertanto, la tabella di inoltro al nodo  $x$  specificherebbe il nodo  $v^*$  come router successivo per la destinazione finale  $y$ . Un altro importante contributo pratico della formula è suggerire la forma della comunicazione tra vicini che avrà luogo nell'algoritmo usato dal DV, detto appunto **algoritmo di Bellman-Ford**.

L'idea di base è la seguente. Ciascun nodo  $x$  inizia con  $D_x(y)$ , una stima del costo del percorso a costo minimo da sé stesso al nodo  $y$ , per tutti i nodi in  $N$ . Sia  $\mathbf{D}_x = [D_x(y): y \in N]$  il vettore delle distanze del nodo  $x$ , che è il vettore delle stime dei costi da  $x$  a tutti gli altri nodi,  $y$ , in  $N$ . Con l'algoritmo Bellman-Ford, ciascun nodo  $x$  mantiene i seguenti dati di instradamento.

- Per ciascun vicino  $v$ , il costo  $c(x,v)$  da  $x$  al vicino  $v$ .
- Il vettore delle distanze del nodo  $x$ , che è  $\mathbf{D}_x = [D_x(y): y \in N]$ , contenente la stima presso  $x$  del costo verso tutte le destinazioni,  $y$ , in  $N$ .
- I vettori delle distanze di ciascuno dei suoi vicini, ossia  $\mathbf{D}_v = [D_v(y): y \in N]$ , per ciascun vicino  $y$  di  $x$ .

In questo algoritmo distribuito e asincrono, ogni tanto un nodo invia una copia del proprio vettore delle distanze a ciascuno dei suoi vicini. Quando un nodo  $x$  riceve un nuovo vettore da qualcuno dei suoi vicini  $v$ , lo salva e quindi usa la formula di Bellman-Ford per aggiornare il proprio vettore come segue:

$$D_x(y) = \min_v \{ c(x,v) + D_v(y) \} \text{ per ciascun nodo } y \text{ in } N.$$

Se il vettore delle distanze del nodo  $x$  è cambiato per via di tale passo di aggiornamento, il nodo  $x$  manderà il proprio vettore aggiornato a ciascuno dei suoi vicini, i quali a loro volta aggiorneranno il proprio vettore. Cosa piuttosto miracolosa, finché tutti i nodi continuano a cambiare i propri vettori delle distanze in maniera asincrona, ciascuna stima dei costi  $D_x(y)$  converge a  $d_x(y)$ , l'effettivo costo del percorso a costo minimo dal nodo  $x$  al nodo  $y$  [Bertsekas 1991]!

## Algoritmo Bellman-Ford

A ciascun nodo  $x$ :

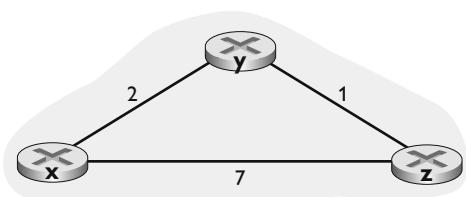
```

1  Inizializzazione:
2    per tutte le destinazioni  $y$  in  $N$ :
3       $D_x(y) = c(x,y)$  /* se  $y$  non è adiacente, allora  $c(x,y)$ 
   =  $\infty$  */
4    per ciascun vicino  $w$ 
5       $D_w(y) = ?$  per tutte le destinazioni  $y$  in  $N$ 
6    per ciascun vicino  $w$ 
7      invia il vettore delle distanze  $D_x = [D_x(y): y \in N]$ 
      a  $w$ 
8
9  ciclo
10   attendi (finché vedi cambiare il costo di un
     collegamento verso
11     qualche vicino  $w$  o finché ricevi un vettore delle
     distanze
12     da qualche vicino  $w$ )
13   per ogni  $y$  in  $N$ :
14      $D_x(y) = \min_v \{c(x,v) + D_v(y)\}$ 
15
16   se  $D_x(y)$  è cambiato per qualche destinazione  $y$ 
17     invia il vettore delle distanze  $D_x = [D_x(y): y \in N]$ 
     a tutti i vicini
18
19  ripeti il ciclo indefinitamente

```

L'algoritmo Bellman-Ford mostra come un nodo  $x$  aggiorni la propria stima del vettore delle distanze quando vede il cambiamento di costo in uno dei collegamenti direttamente connessi o quando riceve da qualche vicino un vettore aggiornato. Tuttavia, per aggiornare la propria tabella di inoltro per una data destinazione  $y$ , ciò che il nodo  $x$  ha davvero bisogno di sapere non è la distanza sul percorso minimo verso  $y$ , bensì il nodo vicino  $v^*(y)$  che rappresenta il router successivo lungo il percorso più breve verso  $y$ . Come ci si potrebbe aspettare, il router  $v^*(y)$  è il vicino  $v$  che ottiene il valore minimo nella riga 14 dell'algoritmo. Se esistono più vicini  $v$  che ottengono il minimo, allora  $v^*(y)$  può essere uno qualunque tra questi. Pertanto, nelle righe 13 e 14, per ciascuna destinazione  $y$  il nodo  $x$  determina anche  $v^*(y)$  e aggiorna la propria tabella di inoltro per la destinazione  $y$ .

Ricordiamo che l'instradamento LS è centralizzato nel senso che richiede a ciascun nodo di ottenere innanzitutto una mappa completa della rete prima di mandare in esecuzione l'algoritmo di Dijkstra. L'algoritmo Bellman-Ford è invece *decentralizzato* e non usa tale informazione globale. Infatti, le sole informazioni detenute dal nodo sono il costo dei collegamenti verso i vicini direttamente connessi e quelle ricevute da questi vicini. Ogni nodo attende aggiornamenti dai suoi vicini (righe 10, 11 e 12), quando riceve un aggiornamento calcola il proprio nuovo vettore delle distanze (riga 14) e lo distribuisce ai suoi vicini (righe 16 e 17). L'instradamento DV viene utilizzato in molti protocolli reali tra cui RIP e BGP per Internet, ISO IDRP, IPX di Novell e ARPAnet originale.



**Figura 5.6**  
Instradamento distance vector (DV).

**Tabella del nodo x**

|    |   | costo verso |   |   |
|----|---|-------------|---|---|
|    |   | x           | y | z |
| da | x | 0           | 2 | 7 |
|    | y | •           | • | • |
| z  |   | •           | • | • |

|    |   | costo verso |   |   |
|----|---|-------------|---|---|
|    |   | x           | y | z |
| da | x | 0           | 2 | 3 |
|    | y | 2           | 0 | 1 |
| z  |   | 7           | 1 | 0 |

|    |   | costo verso |   |   |
|----|---|-------------|---|---|
|    |   | x           | y | z |
| da | x | 0           | 2 | 3 |
|    | y | 2           | 0 | 1 |
| z  |   | 3           | 1 | 0 |

**Tabella del nodo y**

|    |   | costo verso |   |   |
|----|---|-------------|---|---|
|    |   | x           | y | z |
| da | x | •           | • | • |
|    | y | 2           | 0 | 1 |
| z  |   | •           | • | • |

|    |   | costo verso |   |   |
|----|---|-------------|---|---|
|    |   | x           | y | z |
| da | x | 0           | 2 | 7 |
|    | y | 2           | 0 | 1 |
| z  |   | 7           | 1 | 0 |

|    |   | costo verso |   |   |
|----|---|-------------|---|---|
|    |   | x           | y | z |
| da | x | 0           | 2 | 3 |
|    | y | 2           | 0 | 1 |
| z  |   | 3           | 1 | 0 |

**Tabella del nodo z**

|    |   | costo verso |   |   |
|----|---|-------------|---|---|
|    |   | x           | y | z |
| da | x | •           | • | • |
|    | y | •           | • | • |
| z  |   | 7           | 1 | 0 |

|    |   | costo verso |   |   |
|----|---|-------------|---|---|
|    |   | x           | y | z |
| da | x | 0           | 2 | 7 |
|    | y | 2           | 0 | 1 |
| z  |   | 3           | 1 | 0 |

|    |   | costo verso |   |   |
|----|---|-------------|---|---|
|    |   | x           | y | z |
| da | x | 0           | 2 | 3 |
|    | y | 2           | 0 | 1 |
| z  |   | 3           | 1 | 0 |

Tempo

La Figura 5.6 illustra il funzionamento dell'instradamento DV in una semplice rete a tre nodi. Il funzionamento dell'algoritmo viene mostrato in modo sincrono, in quanto tutti i nodi ricevono simultaneamente i vettori delle distanze dai propri vicini, calcolano i rispettivi nuovi vettori e informano i vicini degli eventuali cambiamenti. Dopo aver studiato questo esempio vi dovrete convincere che l'algoritmo opera correttamente anche in modo asincrono, con calcoli ai nodi, generazione e ricezione di aggiornamenti in qualsiasi istante.

La colonna di sinistra della figura mostra tre **tabelle di instradamento** (*routing table*) iniziali per ciascuno dei tre nodi (per esempio, in alto a sinistra è indicata quella del nodo  $x$ ). All'interno di una specifica tabella di instradamento le righe rappresentano i vettori delle distanze: più precisamente, la tabella di instradamento di ciascun nodo include il proprio vettore delle distanze e quello dei suoi vicini. Pertanto, la prima riga nella tabella di instradamento iniziale del nodo  $x$  è  $\mathbf{D}_x = [D_x(x), D_x(y), D_x(z)] = [0, 2, 7]$ . La seconda e la terza riga di questa tabella rappresentano i vettori delle distanze ricevuti più di recente rispettivamente dai nodi  $y$  e  $z$ . Dato che al momento dell'inizializzazione il nodo  $x$  non ha ricevuto nulla dai nodi  $y$  e  $z$ , i valori della seconda e della terza riga sono posti a infinito.

Dopo l'inizializzazione, ciascun nodo invia il proprio vettore ai suoi vicini come mostrato dalle frecce dalla prima alla seconda colonna delle tabelle. Per esempio, il nodo  $x$  invia il suo vettore delle distanze  $\mathbf{D}_x = [0, 2, 7]$  ai nodi  $y$  e  $z$ . Dopo aver ricevuto gli aggiornamenti, i nodi ricalcolano il vettore delle distanze. Per esempio, il nodo  $x$  calcola

$$\begin{aligned} D_x(x) &= 0 \\ D_x(y) &= \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} = \min\{2 + 0, 7 + 1\} = 2 \\ D_x(z) &= \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} = \min\{2 + 1, 7 + 0\} = 3 \end{aligned}$$

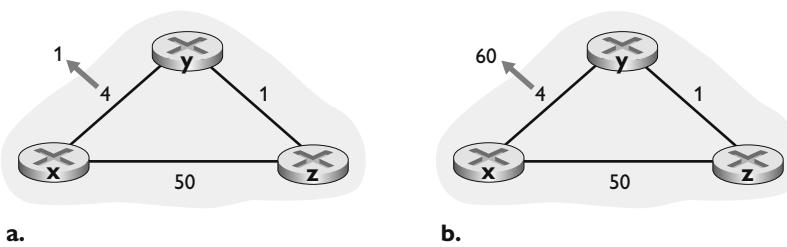
La seconda colonna pertanto mostra, per ciascun nodo, il nuovo vettore delle distanze del nodo e i vettori delle distanze appena ricevuti dai suoi vicini. Notiamo, per esempio, che la stima del nodo  $x$  per il costo minore verso il nodo  $z$ ,  $D_x(z)$ , è passata da 7 a 3, e che il nodo  $y$  ottiene il valore minimo alla riga 14 dell'algoritmo Bellman-Ford; quindi, a questo stadio dell'algoritmo, per il nodo  $x$ ,  $v^*(y) = y$  e  $v^*(z) = y$ .

Dopo aver ricalcolato i rispettivi vettori delle distanze, i nodi li inviano in versione aggiornata ai propri vicini (ammesso che siano cambiati), procedura indicata nella figura con le frecce dalla seconda colonna alla terza. Notiamo che solo i nodi  $x$  e  $z$  inviano aggiornamenti: il vettore delle distanze del nodo  $y$  non è cambiato e pertanto non è stato spedito. Dopo la ricezione degli aggiornamenti, i nodi ricalcolano i propri vettori delle distanze e aggiornano le tabelle di instradamento (terza colonna).

Il processo di ricezione dei vettori aggiornati, di ricalcolo delle righe nella tabella di instradamento e di informazione ai vicini sui costi cambiati nel percorso a costo minimo verso la destinazione continua finché non viene più inviato alcun messaggio di aggiornamento. A questo punto, dato che non vengono più inviati messaggi di aggiornamento, non si verificano ulteriori computazioni nella tabella di instradamento e l'algoritmo entra in uno stato quiescente. In altre parole, tutti i nodi eseguono l'attesa nelle righe 10 e 11 dell'algoritmo Bellman-Ford. L'algoritmo rimane in tale stato finché non cambia il costo di un collegamento.

### Instradamento distance-vector: modifica dei costi e guasti dei collegamenti

Quando un nodo che esegue l'instradamento DV rileva un cambiamento nel costo dei collegamenti con un vicino (righe 10-11-12) aggiorna il proprio vettore delle distanze (riga 13-14) e, se si verifica un cambiamento nel costo del percorso a costo minimo, trasmette ai suoi vicini (righe 16-17) il proprio nuovo vettore.



**Figura 5.7** Variazioni nel costo dei collegamenti.

tore delle distanze. La Figura 5.7(a) mostra uno scenario in cui il costo del collegamento da  $y$  a  $x$  passa da 4 a 1. In questa sede ci concentriamo solo su  $y$  e sulle righe della tabella delle distanze di  $z$  verso la destinazione  $x$ . L'algoritmo provoca la seguente sequenza di eventi.

- All'istante  $t_0$ ,  $y$  rileva il cambiamento nel costo del collegamento (passato da 4 a 1), aggiorna il proprio vettore delle distanze e informa i vicini del cambiamento.
- All'istante  $t_1$ ,  $z$  riceve l'aggiornamento da  $y$  e aggiorna la propria tabella, calcola un nuovo costo minimo verso  $x$  (che passa da 5 a 2) e invia il nuovo vettore delle distanze ai vicini.
- All'istante  $t_2$ ,  $y$  riceve l'aggiornamento di  $z$  e aggiorna la propria tabella delle distanze. I costi minimi di  $y$  non cambiano e  $y$  non manda alcun messaggio a  $z$ . L'algoritmo entra in uno stato quiescente.

Pertanto, dopo due iterazioni l'algoritmo raggiunge uno stato di quiete. Le buone notizie sul costo diminuito tra  $x$  e  $y$  si sono propagate rapidamente nella rete.

Prendiamo in considerazione ora che cosa può avvenire quando il costo di un collegamento aumenta. Supponiamo che il costo del collegamento tra  $x$  e  $y$  passi da 4 a 60, come mostrato nella Figura 5.7(b).

1. Prima che il costo del collegamento cambi,  $D_y(x) = 4$ ,  $D_y(z) = 1$ ,  $D_z(y) = 1$  e  $D_z(x) = 5$ . All'istante  $t_0$ ,  $y$  rileva che il costo del collegamento è passato da 4 a 60 e calcola il suo nuovo percorso a costo minimo verso  $x$  con la formula

$$D_y(x) = \min \{ c(y,x) + D_x(x), c(y,z) + D_z(x) \} = \min\{60 + 0, 1 + 5\} = 6$$

Ovviamente, con la nostra visione globale della rete, possiamo rilevare che questo nuovo costo attraverso  $z$  è *errato*. Ma l'unica informazione che il nodo  $y$  possiede è che il costo diretto verso  $x$  è 60 e che  $z$  ha ultimamente detto a  $y$  di essere in grado di giungere a  $x$  con un costo di 5. Pertanto al fine di arrivare a  $x$ ,  $y$  ora farebbe passare il percorso per  $z$ , aspettandosi che questo sia in grado di giungere a  $x$  con un costo pari a 5. All'istante  $t_1$ , abbiamo un **instradamento ciclico**: al fine di giungere a  $x$ ,  $y$  fa passare il percorso per  $z$  e  $z$  lo fa passare per  $y$ . Un ciclo in un instradamento assomiglia a un buco nero: un pacchetto destinato a  $x$  che arriva a  $y$  o  $z$  all'istante  $t_1$  rimbalzerà avanti e indietro tra questi due nodi per sempre (a meno che non vengano cambiate le tabelle di inoltro).

2. Dato che il nodo  $y$  ha calcolato un nuovo costo minimo verso  $x$ , informa  $z$  del suo nuovo vettore delle distanze all'istante  $t_1$ .

3. In un istante successivo a  $t_1$ ,  $z$  riceve il nuovo vettore delle distanze di  $y$ , che indica che il costo minimo di  $y$  verso  $x$  è 6, sa che può giungere a  $y$  a costo 1 e quindi calcola un nuovo costo minimo verso  $x$  pari a  $D_z(x) = \min\{50 + 0, 1 + 6\} = 7$ . Dato che il costo minimo di  $z$  verso  $x$  è aumentato,  $z$  informa  $y$  del suo nuovo vettore delle distanze al tempo  $t_2$ .
4. Analogamente, dopo aver ricevuto il nuovo vettore delle distanze di  $z$ ,  $y$  determina  $D_y(x) = 8$  e invia a  $z$  il suo nuovo vettore delle distanze.  $z$  allora determina  $D_z(x) = 9$  e invia a  $y$  il suo nuovo vettore delle distanze, e così via.

Il ciclo proseguirà per 44 iterazioni (scambi di messaggi tra  $y$  e  $z$ ), fino a quando  $z$  considera il costo del proprio percorso attraverso  $y$  maggiore di 50. A questo punto,  $z$  determina che il percorso a costo minimo verso  $x$  passa attraverso la connessione diretta a  $x$ , e  $y$  instradera verso  $x$  passando per  $z$ . Il risultato delle cattive notizie sull'incremento del costo dei collegamenti ha viaggiato senza dubbio assai lentamente. Che cosa sarebbe successo se il costo del collegamento  $c(y,x)$  fosse passato da 4 a 10.000 e il costo  $c(z,x)$  avesse avuto valore 9999? A causa di questi scenari, il processo che abbiamo descritto viene talvolta detto **problema di conteggio all'infinito** (*count-to-infinity problem*).

### Instrandamento distance-vector: aggiunta dell'inversione avvelenata

Lo specifico scenario con cicli appena descritto può essere evitato utilizzando una tecnica nota come **inversione avvelenata** (*poisoned reverse*). L'idea è semplice: se  $z$  in strada tramite  $y$  per giungere alla destinazione  $x$ , allora  $z$  avverrà  $y$  che la sua distanza verso  $x$  è infinita, ossia  $z$  comunicherà a  $y$  che  $D_z(x) = +\infty$ , anche se in realtà  $z$  sa che  $D_z(x) = 5$ , e continuerà a dire questa piccola bugia fintanto che in strada verso  $x$  passando per  $y$ . Dato che  $y$  crede che  $z$  non abbia un percorso verso  $x$ , non tenterà mai di instradare verso  $x$  passando per  $z$ , per tutto il tempo in cui  $z$  continua a instradare verso  $x$  passando per  $y$  (e mente a riguardo).

Vediamo ora come l'inversione avvelenata risolva il problema dei cicli che abbiamo incontrato nella Figura 5.5(b). Come risultato dell'inversione avvelenata, la tabella di distanza di  $y$  indica  $D_z(x) = +\infty$ . Quando il costo del collegamento  $(x,y)$  cambia da 4 a 60 all'istante  $t_0$ ,  $y$  aggiorna la propria tabella, continua a instradare direttamente verso  $x$ , nonostante il costo più alto pari a 60, e informa  $z$  del suo nuovo costo verso  $x$ , ossia  $D_y(x) = 60$ . Una volta ricevuto l'aggiornamento all'istante  $t_1$ ,  $z$  cambia immediatamente il proprio percorso verso  $x$  facendolo passare attraverso il collegamento diretto  $(x,y)$  al costo 50. Dato che questo è il nuovo percorso a costo minimo verso  $x$  e dato che non passa più attraverso  $y$ ,  $z$  ora informa  $y$  che all'istante  $t_2$   $D_z(x) = 50$ . Dopo aver ricevuto l'aggiornamento da  $z$ ,  $y$  adegua la propria tabella di distanza ponendo  $D_y(x) = 51$ . Ancora, dato che  $z$  si trova ora sul percorso a costo minimo di  $y$  verso  $x$ ,  $y$  avvelena il percorso inverso da  $z$  a  $x$  informando  $z$  all'istante  $t_3$  che  $D_y(x) = +\infty$ , anche se in realtà  $y$  sa che  $D_y(x) = 51$ .

L'inversione avvelenata risolve il problema generale del conteggio all'infinito? La risposta è negativa: i cicli che non riguardano semplicemente due nodi adiacenti non verranno rilevati dalla tecnica dell'inversione avvelenata.

### Confronto tra gli instradamenti LS e DV

I due meccanismi di instradamento studiati utilizzano approcci complementari nel calcolare i percorsi. Nel DV ciascun nodo dialoga *solo* con i vicini direttamente connessi, informandoli delle stime a costo minimo da sé stesso a *tutti* i nodi (che conosce) nella rete. Nel LS ciascun nodo dialoga con *tutti* gli altri nodi via broadcast, ma comunica loro *solo* i costi dei collegamenti direttamente connessi. Concludiamo la nostra trattazione degli instradamenti LS e DV con un veloce confronto di alcuni dei rispettivi attributi. Ricordiamo che  $N$  rappresenta l'insieme dei nodi (router) ed  $E$  l'insieme degli archi (collegamenti).

- **Complessità dei messaggi.** LS richiede che ciascun nodo conosca il costo di ogni collegamento nella rete. Ciò implica l'invio di  $O(|N| \cdot |E|)$  messaggi. Inoltre, ognqualvolta cambia il costo di un collegamento, il nuovo costo deve essere comunicato a tutti i nodi. A ogni iterazione l'instradamento DV richiede scambi di messaggi tra nodi adiacenti. Abbiamo visto che il tempo richiesto perché l'algoritmo converga può dipendere da molti fattori. Quando cambiano i costi dei collegamenti, l'instradamento DV propaga i risultati dei costi cambiati se il nuovo costo ha causato la variazione del percorso a costo minimo per uno o più nodi connessi a tale collegamento.
- **Velocità di convergenza.** Abbiamo visto che la nostra implementazione di LS è un algoritmo  $O(|N|^2)$  che richiede  $O(|N| \cdot |E|)$  messaggi. L'algoritmo del DV può convergere lentamente e può presentare cicli di instradamento. DV presenta anche il problema del conteggio all'infinito.
- **Robustezza.** Che cosa avviene se un router si guasta, funziona male o viene sabotato? Con LS, un router può comunicare via broadcast un costo sbagliato per uno dei suoi collegamenti connessi (ma non per altri). Un nodo può anche alterare o eliminare i pacchetti ricevuti in broadcast LS. Ma i nodi LS si occupano di calcolare soltanto le proprie tabelle di inoltro, e gli altri nodi effettuano calcoli simili per quanto li riguarda. Ciò significa che i calcoli di instradamento sono in qualche modo isolati, il che fornisce un certo grado di robustezza. Con DV un nodo può comunicare percorsi a costo minimo errati a tutte le destinazioni. In effetti, nel 1997 un router malfunzionante di un piccolo ISP ha comunicato informazioni di instradamento errate a router della dorsale nazionale americana. Ciò ha indotto gli altri router a sommersere di traffico il router malfunzionante e per svariate ore grandi porzioni di Internet hanno perso connettività [Neumann 1997]). Più in generale notiamo che, a ogni iterazione, il calcolo di DV in un nodo viene comunicato ai suoi vicini e quindi ai vicini dei vicini alla successiva iterazione. In questo senso, un calcolo errato su un nodo si può diffondere per l'intera rete.

In conclusione, nessuno dei due meccanismi di instradamento è nettamente superiore all'altro ed entrambi vengono utilizzati in Internet.

## 5.3 Instradamento interno ai sistemi autonomi: OSPF

Nel nostro studio degli algoritmi di instradamento abbiamo visto la rete semplicemente come una collezione di router interconnessi. Ciascun router era indistinguibile dagli altri nel senso che tutti eseguivano lo stesso algoritmo per

calcolare l'instradamento attraverso la rete. In pratica, questo modello con la sua visione omogenea dei router risulta un po' semplicistico per almeno due importanti motivi.

- **Scalabilità.** Al crescere del numero di router, il tempo richiesto per calcolare, memorizzare e comunicare le informazioni di instradamento diventa proibitivo. Attualmente, Internet è costituita da centinaia di milioni di host. Archiviare le informazioni di instradamento su ciascuno di essi richiederebbe chiaramente un'enorme quantità di memoria, e il traffico generato dalla comunicazione broadcast degli aggiornamenti non lascerebbe banda per i pacchetti di dati. Un algoritmo distance-vector con interazioni tra un così grande numero di router non convergerebbe mai. Ovviamente, si deve fare qualcosa per ridurre la complessità del calcolo dei percorsi nelle reti di grandi dimensioni.
- **Autonomia amministrativa.** Internet è una rete di ISP che generalmente desiderano gestire autonomamente i propri router (per esempio, scegliendo l'algoritmo di instradamento preferito) o nascondere all'esterno aspetti dell'organizzazione interna della rete. L'ideale sarebbe che ciascuno fosse in grado di amministrare la propria rete nel modo desiderato, pur mantenendo la possibilità di connetterla alle reti esterne.

Questi problemi possono essere risolti organizzando i router in **sistemi autonomi** (AS, *Autonomous System*), generalmente composti da gruppi di router posti sotto lo stesso controllo amministrativo. A volte i router e i collegamenti di un ISP formano un unico AS, gigante nel caso di ISP tier-1, altre volte gli ISP partizionano la loro rete in più AS. Un AS è identificato da un numero di sistema autonomo (ASN) [RFC 1930] univoco che viene assegnato da ICANN [ICANN 2020], esattamente come gli indirizzi IP.

I router di un AS eseguono lo stesso algoritmo di instradamento e gli uni hanno informazioni sugli altri. L'algoritmo di instradamento in esecuzione in un AS è detto **protocollo di instradamento interno al sistema autonomo (intra-AS routing protocol)**.

### **OSPF (Open Shortest Path First)**

OSPF e il suo stretto parente, IS-IS, sono generalmente impiegati negli ISP di livello superiore. Il termine *open* nell'acronimo indica che le specifiche del protocollo sono pubblicamente disponibili, cosa che non era inizialmente avvenuta per il protocollo EIGRP di Cisco, reso open solo dopo 20 anni dalla sua introduzione [Savage 2015]. La versione più recente di OSPF, la 2, è definita nell'RFC 2328, che è un documento pubblico.

OSPF è un protocollo link-state che utilizza il **flooding (inondazione)** per inviare in broadcast le informazioni riguardo lo stato dei collegamenti e l'algoritmo di Dijkstra per la determinazione del percorso a costo minimo. In OSPF, un router costruisce una mappa topologica, cioè un grafo, dell'intero sistema autonomo e manda in esecuzione (locale) l'algoritmo di Dijkstra per determinare un albero dei cammini minimi verso tutte le sottoreti (albero in cui il router stesso rappresenta il nodo radice). I costi dei collegamenti vengono fissati dall'amministratore di rete (si veda il Box 5.1, “Come impostare i pesi dei collegamenti OSPF”). L'amministratore può scegliere di impostare i costi di tutti i collegamenti a 1 (ottenendo di conseguenza l'instradamento con il minimo numero

**BOX 5.1****TEORIA E PRATICA****Come impostare i pesi dei collegamenti OSPF**

Nella nostra trattazione dell'instradamento link-state abbiamo implicitamente assunto che siano stati impostati i pesi dei collegamenti, che venga eseguito un algoritmo di instradamento come OSPF e che il traffico fluisca secondo le tabelle di instradamento calcolate dall'algoritmo. In termini di cause ed effetti, i pesi dei collegamenti sono dati e determinano (tramite l'algoritmo di Dijkstra) i percorsi a costo minimo complessivo. In quest'ottica, i pesi riflettono il costo di utilizzo dei collegamenti (per esempio, se sono inversamente proporzionali alla capacità, l'uso di collegamenti con elevate prestazioni comporterebbe pesi inferiori e pertanto risulterebbe più interessante per quanto riguarda l'instradamento) e l'algoritmo di Dijkstra serve a minimizzare il costo complessivo.

In pratica, questa relazione causa-effetto può essere invertita, per cui gli operatori di rete possono configurare i pesi dei collegamenti per ottenere percorsi di instradamento che raggiungono determinati obiettivi d'ingegnerizzazione del traffico [Fortz 2000; Fortz 2002]. Supponiamo per esempio che un operatore di rete possieda una stima del flusso di traffico nei punti d'ingresso e in quelli di uscita della rete. Questi può voler attuare uno specifico instradamento del flusso che minimizza l'utilizzo massimo dei collegamenti della rete. Ma con un algoritmo d'instradamento quale OSPF, le principali "manopole" dell'operatore per mettere a punto l'instradamento dei flussi attraverso la rete sono i pesi dei collegamenti. Di conseguenza, per minimizzare il massimo utilizzo dei collegamenti, l'operatore deve trovare adeguati pesi dei collegamenti. Si tratta di un'inversione della relazione causa-effetto: l'instradamento desiderato dei flussi è noto e i pesi dei collegamenti OSPF devono essere determinati in modo che OSPF abbia come risultato l'instradamento desiderato.

di hop), oppure in modo inversamente proporzionale alla capacità del collegamento (al fine di scoraggiare l'uso di collegamenti con poca banda). OSPF non impone una politica sulla scelta dei pesi dei collegamenti (lavoro che compete all'amministratore di rete), ma fornisce invece i meccanismi per determinare l'instradamento con percorso a costo minimo per un dato insieme di pesi dei collegamenti.

In OSPF, ogni qualvolta si verifica un cambiamento nello stato di un collegamento (per esempio, una variazione di costo o un cambiamento di disponibilità), il router manda informazioni di instradamento via broadcast a *tutti* gli altri router nel sistema autonomo. Inoltre, invia periodicamente lo stato dei collegamenti (almeno ogni 30 minuti), anche se questo non è cambiato. L'RFC 2328 afferma che "l'aggiornamento periodico degli annunci sullo stato dei collegamenti aggiunge robustezza all'algoritmo link-state". Gli annunci OSPF sono contenuti in messaggi OSPF che vengono trasportati direttamente da IP come un protocollo di livello superiore con identificativo 89. Quindi, il protocollo OSPF deve implementare funzionalità quali il trasferimento affidabile dei messaggi e il broadcast dello stato dei collegamenti. Inoltre, controlla che i collegamenti siano operativi (tramite messaggio HELLO inviato a un vicino connesso) e consente ai router OSPF di accedere ai database sullo stato dei collegamenti della rete, contenuti nei router confinanti.

Tra i vantaggi di OSPF ricordiamo i seguenti.

- **Sicurezza.** Gli scambi tra router OSPF (come gli aggiornamenti sullo stato dei collegamenti) possono essere autenticati; di conseguenza, soltanto router fidati possono prendere parte al protocollo OSPF in un sistema autonomo, evitando che malintenzionati (o studenti di networking forti delle loro nuove conoscenze) immettano informazioni errate nelle tabelle dei router. Come comportamento predefinito, i pacchetti OSPF tra router non sono autenticati e potrebbero essere contraffatti. Si possono configurare due tipi di autenticazione: semplice e tramite MD5 (Capitolo 8, disponibile in inglese sulla piattaforma MyLab). Nel caso di autenticazione semplice, la meno sicura, si configura su tutti i router la stessa password che deve essere inclusa, in chiaro, nei pacchetti OSPF a essi inviati. L'autenticazione MD5 si basa su chiavi segrete condivise, configurate in ogni router. Per ogni pacchetto OSPF che mandano, i router calcolano l'hash MD5 del contenuto a cui è stata aggiunta la chiave segreta (per i codici di autenticazione del messaggio si veda il Capitolo 8, disponibile in inglese sulla piattaforma MyLab) e includono il risultato nel pacchetto OSPF. Il router ricevente calcola la funzione *hash* MD5 del pacchetto e, usando la chiave segreta preconfigurata, la confronta con il valore che il pacchetto trasporta, per verificarne l'autenticità. Con l'autenticazione MD5 vengono usati anche numeri di sequenza come protezione contro i *replay attacks*.
- **Percorsi con lo stesso costo.** Quando più percorsi verso una destinazione hanno lo stesso costo, OSPF consente di usarli senza doverne scegliere uno per trasportare tutto il traffico.
- **Supporto integrato per l'instradamento unicast e multicast.** Per consentire l'instradamento multicast viene impiegata una semplice estensione di OSPF, MOSPF (multicast OSPF) [RFC 1584], che utilizza il database dei collegamenti OSPF e aggiunge un nuovo tipo di annuncio sullo stato dei collegamenti al meccanismo di broadcast.
- **Supporto alle gerarchie in un dominio di instradamento.** La novità forse più significativa di OSPF è la possibilità di strutturare i sistemi autonomi in modo gerarchico. Un sistema autonomo OSPF può essere configurato in aree che eseguono diversi algoritmi di instradamento OSPF: ciascun router in un'area invia lo stato dei suoi collegamenti agli altri router nell'area. In ogni area, uno o più **router di confine d'area** (*area border router*) si fa carico dell'instradamento dei pacchetti indirizzati all'esterno. Un'area di un sistema autonomo OSPF è configurata per essere l'**area di dorsale** (*backbone area*), il cui ruolo principale è quello di instradare il traffico tra le altre aree nel sistema autonomo. La dorsale contiene sempre tutti i router di confine del sistema autonomo, ma non necessariamente soltanto quelli. L'instradamento nell'area di un sistema autonomo richiede che i pacchetti siano instradati dapprima verso il proprio router di confine (instradamento intra-area) e da questi, attraverso la dorsale, al router di confine dell'area di destinazione, che provvede a farli pervenire alla destinazione finale.

OSPF è un protocollo relativamente complesso e la nostra trattazione è stata ovviamente schematica. Per ulteriori dettagli potete consultare [Huitema 1998; Moy 1998; RFC 2328].

## 5.4 Instradamento tra ISP: BGP

Abbiamo appena appreso come gli ISP utilizzino OSPF per determinare i percorsi ottimali per le coppie sorgente-destinazione interne a un sistema autonomo. Per determinare i percorsi per le coppie sorgente-destinazione che interessano più sistemi autonomi, come uno smartphone a Timbuktu e un server in un data center della Silicon Valley, è tuttavia necessario un **protocollo di instradamento inter-AS** (*inter-autonomous system routing protocol*) che coordini più AS. Il **border gateway protocol**, detto **BGP** [RFC 4271; Stewart 1999], rappresenta l'attuale standard *de facto* dei protocolli di instradamento tra sistemi autonomi in Internet.

BGP è forse il più importante dei protocolli di Internet, al pari di IP (Paragrafo 4.3), in quanto è il collante che tiene insieme le migliaia di ISP che formano Internet. È un protocollo di tipo distance-vector decentralizzato e asincrono. BGP è estremamente complesso ma, data la sua importanza, è necessario acquisire quantomeno una comprensione rudimentale del suo modo di operare.

### 5.4.1 Il ruolo di BGP

Per capire il ruolo di BGP, si consideri un AS e un suo router qualsiasi. Si ricordi che ogni router ha una tabella di inoltro, fondamentale per il processo di inoltro dei pacchetti. Abbiamo visto che le occorrenze della tabella di inoltro corrispondenti a destinazioni interne all'AS vengono determinate dal protocollo di instradamento intra-AS, ma chi determina quelle esterne all'AS? Questo è precisamente il ruolo di BGP.

In BGP, i pacchetti non vengono instradati verso uno specifico indirizzo, ma piuttosto verso **prefissi** CIDR che rappresentano una sottorete o una collezione di sottoreti. Nel mondo di BGP, una destinazione potrebbe avere la forma 138.16.68/22, che include 1024 indirizzi IP. Quindi le occorrenze delle tabelle di inoltro hanno forma  $(x, I)$  dove  $x$  è un prefisso di rete e  $I$  è un numero di interfaccia del router.

BGP mette a disposizione di ciascun router un modo per:

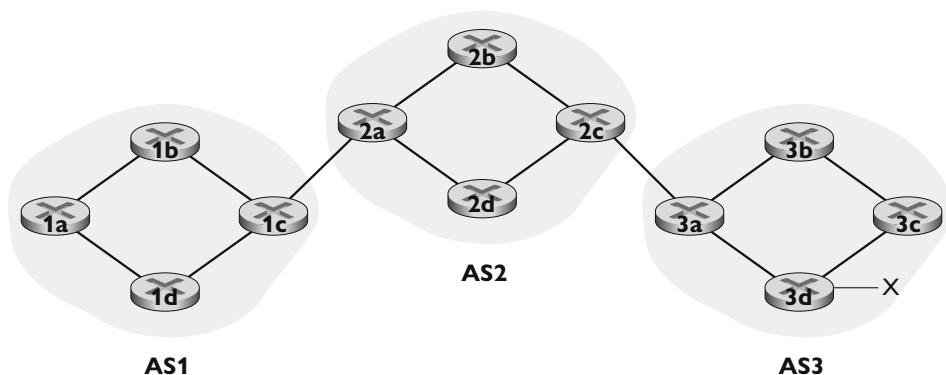
1. *Ottenere informazioni sulla raggiungibilità dei prefissi di sottorete* da parte dei *sistemi confinanti*. In particolare, BGP consente a ciascuna sottorete di comunicare la propria esistenza al resto di Internet. Basta che una sottorete urli “Esisto, son qui” e BGP si assicura che lo sappiano tutti i router di Internet. Se non fosse per BGP, ogni sottorete sarebbe isolata, sola, sconosciuta e irraggiungibile dal resto di Internet.
2. *Determinare i percorsi “ottimi” verso le sottoreti*. Un router può venire a conoscenza di più cammini verso uno specifico prefisso; per determinare il migliore, esegue localmente BGP, sulla base delle informazioni di raggiungibilità e delle politiche del sistema.

Ora vedremo come BGP svolga queste due operazioni.

### 5.4.2 Distribuzione delle informazioni dei percorsi in BGP

Si consideri la rete della Figura 5.8 con tre AS: AS1, AS2 e AS3, quest'ultimo con una sottorete di prefisso  $x$ . Ogni router, in ogni AS, funge sia da **router gateway**, vale a dire un router di bordo direttamente connesso a uno o più router

**Figura 5.8** Rete con tre AS. AS3 include una sottorete con prefisso x.

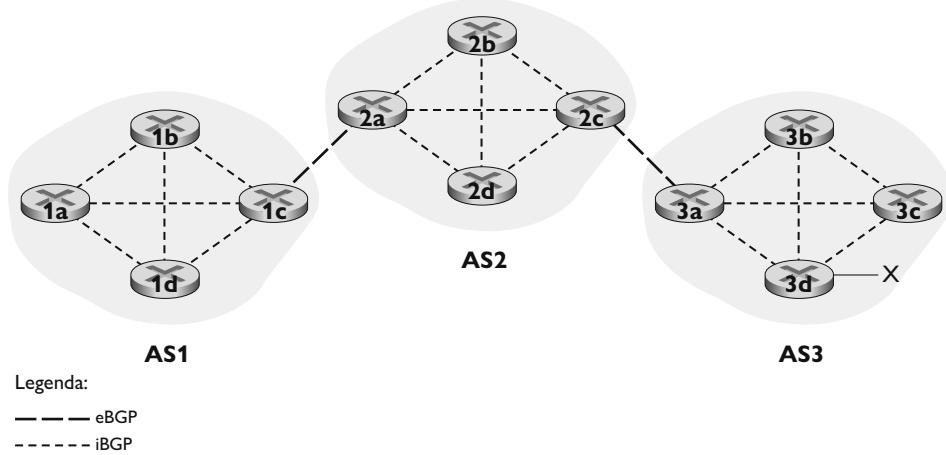


di altri AS, sia da **router interno**, connesso cioè solo a host e router interni all'AS. In AS1, per esempio, il router 1c è un router gateway, mentre 1a, 1b e 1d sono router interni.

Esaminiamo ora come BGP distribuirebbe le informazioni di raggiungibilità del prefisso x a tutti i router della Figura 5.8. In linea di principio è semplice. AS3 invia un messaggio BGP ad AS2, con l'annuncio dell'esistenza in AS3 di x; denotiamo tale messaggio con “AS3 x”. Quindi AS2 invia un messaggio BGP a AS1 con l'annuncio che x esiste ed è raggiungibile passando prima da AS2 per poi arrivare a AS3: “AS2 AS3 x”. In questo modo ogni AS verrà a conoscenza non solo dell'esistenza di x, ma anche dei percorsi per raggiungerlo.

Benché questa sia l'idea generale, in realtà sono i router e non gli AS a inviare gli annunci. Riesaminiamo l'esempio della Figura 5.8. In BGP, coppie di router si scambiano informazioni di instradamento su connessioni TCP semi-permanenti usando la porta 179. Le connessioni TCP semi-permanenti per la rete della Figura 5.8 sono mostrate nella Figura 5.9. Ogni connessione TCP, con tutti i messaggi BGP che vi vengono inviati, è detta **sessione BGP**. Nel caso in cui questa coinvolga due sistemi autonomi viene detta **sessione BGP esterna** (**eBGP**), mentre quella tra router dello stesso sistema autonomo è chiamata **sessione BGP interna** (**iBGP**). Generalmente, in BGP c'è una

**Figura 5.9** Connessioni eBGP e iBGP.



connessione TCP di questo tipo per ciascun collegamento che connette direttamente due router in due diversi sistemi autonomi; nella figura vediamo una connessione sessione eBGP tra i gateway 2a e 1c e tra i router 2c e 3a.

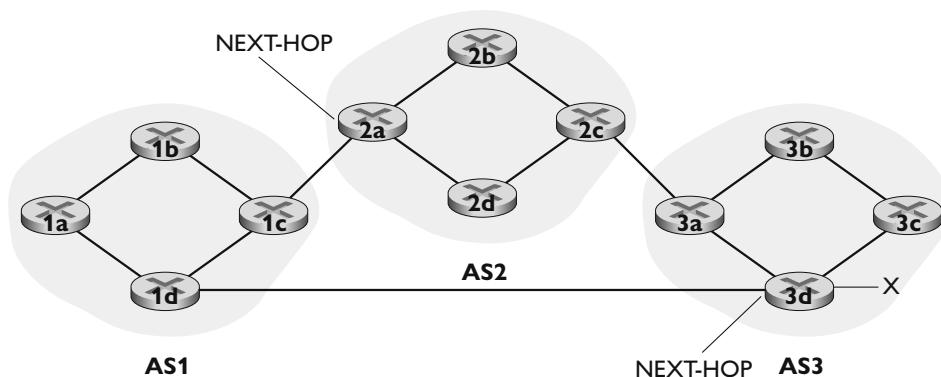
Esistono anche sessioni BGP interne tra router interni ai sistemi autonomi. La figura mostra la configurazione comune di una sessione BGP per ciascuna coppia di router interni a un sistema autonomo, creando una rete “a maglia completa” di connessioni TCP. Nella Figura 5.9 le sessioni eBGP sono evidenziate con tratteggio lungo e quelle iBGP con tratteggio breve. Notiamo che le linee di sessione BGP non sempre corrispondono ai collegamenti fisici.

Esaminiamo ora come BGP distribuirebbe le informazioni di raggiungibilità del prefisso x ad AS1 e AS2. In router gateway 3a invia un messaggio eBGP “AS3 x” al router gateway 2c che a sua volta lo invia su una sessione iBGP a tutti i router di AS2 compreso il gateway 2a. Quest’ultimo quindi invia un messaggio eBGP “AS2 AS3 x” al router gateway 1c, che infine usa iBGP per inviare il messaggio “AS2 AS3 x” a tutti i router di AS1. Completato tale processo, ogni router di AS1 e AS2 è a conoscenza dell’esistenza di x e del percorso per raggiungerlo.

Ovviamente in una rete reale esistono più percorsi da un router a una data destinazione che attraversano AS diversi. Per esempio, nella rete rappresentata nella Figura 5.10, ottenuta da quella della Figura 5.8 aggiungendo un collegamento tra i router 1d e 3d, esistono 2 percorsi da AS1 a x: il percorso “AS2 AS3 x” attraverso il router 1c e il nuovo percorso “AS3 x” attraverso il router 1d.

### 5.4.3 Selezione delle rotte migliori

Ma come fa un router a scegliere il percorso migliore tra le dozzine che riceve attraverso gli annunci di raggiungibilità? Prima di rispondere introduciamo un po’ di terminologia BGP. Quando un router annuncia un prefisso per una sessione BGP, include anche un certo numero di **attributi BGP**. In gergo, un prefisso assieme ai suoi attributi è detto **rotta** (*route*). Due dei più importanti attributi sono **AS-PATH** e **NEXT-HOP**. **AS-PATH** elenca i sistemi autonomi attraverso i quali è passato l’annuncio del prefisso. Quando un prefisso attraversa un sistema autonomo, questi aggiunge il proprio ASN all’attributo **AS-PATH**. Per esempio, consideriamo la Figura 5.10 con due rotte da AS1 a x: una usa l’AS-PATH “AS2 AS3”, l’altra “AS3”. I router usano tale attributo per ri-



**Figura 5.10** Rete con aggiunta del collegamento tra AS1 e AS3.

levare ed evitare gli annunci reiterati. Nello specifico, se un router vede che il proprio AS è contenuto nella lista di percorsi, rifiuta l'annuncio.

Nel fornire il delicato collegamento tra i protocolli di instradamento intra-AS e inter-AS, l'attributo **NEXT-HOP** ha un ruolo sottile, ma importante. In NEXT-HOP è riportata l'interfaccia del router che inizia l'AS-PATH. Per comprenderlo, facciamo di nuovo riferimento alla Figura 5.10. L'attributo NEXT-HOP per la rotta “AS2 AS3 x” da AS1 a x passando per AS2 è l'indirizzo IP dell'interfaccia a sinistra del router 2a; l'attributo NEXT-HOP per la rotta “AS3 x” da AS1 a x che salta AS2 è l'indirizzo IP dell'interfaccia più a sinistra del router 3d. Quindi ogni router di AS1 conosce ora due rotte per il prefisso x:

L'indirizzo IP dell'interfaccia più a sinistra del router 2a: AS2 AS3; x

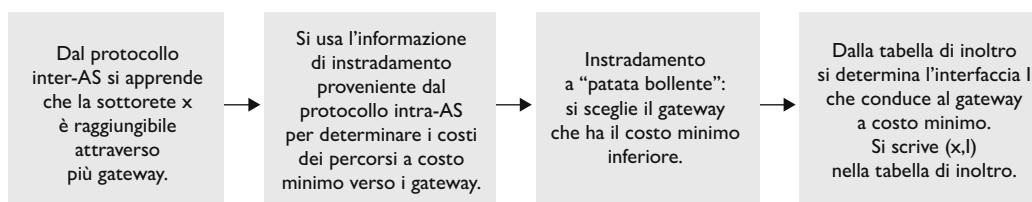
L'indirizzo IP dell'interfaccia più a sinistra del router 3d: AS3; x

Ogni rotta è quindi scritta come una lista di tre elementi: NEXT-HOP, AS-PATH, prefisso x. In realtà include anche altri attributi che per ora trascuriamo. Si noti che NEXT-HOP è l'indirizzo IP di un router che non appartiene ad AS1, ma a una sottorete collegata direttamente ad AS1.

### Instradamento hot potato

Siamo ora finalmente in grado di parlare di algoritmi di routing BGP in modo preciso. Inizieremo con uno dei più semplici algoritmi di routing, vale a dire, l'**instradamento a patata bollente** (*hot potato routing*). Consideriamo il router 1b nella rete della Figura 5.10. Come appena descritto, questo router impara due possibili percorsi BGP verso il prefisso x. Nell'instradamento a patata bollente, il percorso scelto (tra tutti i percorsi possibili) è quello con il minor costo per il router NEXT-HOP che lo inizia. In questo esempio, il router 1b consulterà le sue informazioni di routing intra-AS per trovare il percorso intra-AS a costo minimo verso il router NEXT-HOP 2a e il percorso a costo minimo verso il router NEXT-HOP 3d, e tra questi seleziona quello a costo minimo. Per esempio, supponiamo che il costo sia definito come il numero di collegamenti attraversati; dal router 1b al router 2a il costo è 2, dal router 1b al router 2d è 3: verrebbe quindi selezionato il router 2a. Il router 1b consulterebbe quindi la sua tabella di inoltro (configurata dal suo algoritmo intra-AS) per trovare l'interfaccia I che si trova sul percorso a costo minimo per il router 2a. Quindi aggiunge (x, I) alla sua tabella di inoltro.

La procedura per inserire un prefisso fuori-AS nella tabella di inoltro di un router usando l'instradamento a patata bollente è riassunta nella Figura 5.11. È importante notare che quando si aggiunge un prefisso fuori-AS a una tabella



**Figura 5.11** Passi per aggiungere una destinazione esterna al sistema autonomo in una tabella di inoltro di un router.

di inoltro vengono utilizzati entrambi i protocolli inter-AS (BGP) e intra-AS (per esempio, OSPF).

L'idea alla base è che il router 1b butti fuori i pacchetti dal suo AS quanto prima (più specificamente, con il minor costo possibile) senza preoccuparsi del costo delle restanti tratte del percorso al di fuori del suo AS; da qui il nome “patata bollente”: un pacchetto è analogo a una patata bollente che sta bruciandoti nelle mani e che vuoi passare a un'altra persona (un altro AS) il più rapidamente possibile. È quindi un algoritmo egoista che cerca di ridurre i costi per il proprio AS, non badando a quello degli altri. Si noti che due router nello stesso AS potrebbero selezionare diversi percorsi per lo stesso prefisso. Abbiamo per esempio appena visto che il router 1b invierebbe i pacchetti per  $x$  attraverso AS2, mentre il router 1d li invierebbe direttamente attraverso AS3.

### **Algoritmo di selezione delle rotte**

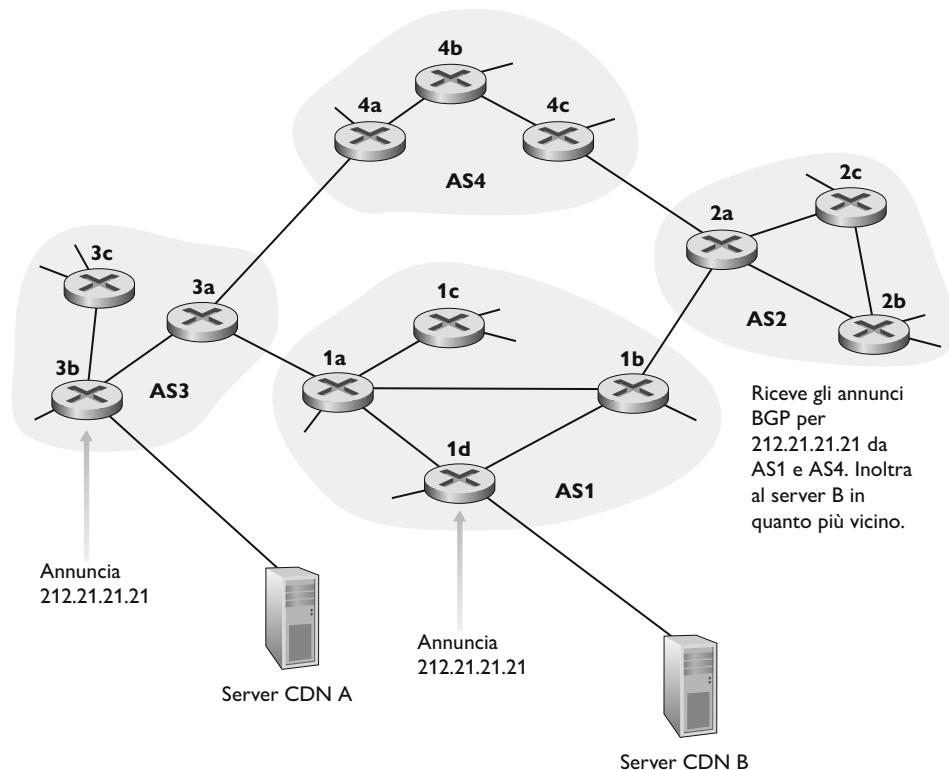
In realtà BGP utilizza un algoritmo più complesso di quello “hot potato”. L'input del processo di selezione è l'insieme di tutte le rotte apprese e accettate dal router. Se ne esistono due o più verso lo stesso prefisso, BGP invoca in sequenza le seguenti regole di eliminazione fino a individuare un'unica possibilità.

1. Alle rotte viene assegnato come attributo un valore di **preferenza locale**, che potrebbe esser stato impostato direttamente dal router o appreso da un altro router nello stesso AS. Si tratta di una scelta che è lasciata all'amministratore di rete del sistema autonomo. Si selezionano quindi le rotte con i più alti valori di preferenza locale.
2. Tra le rotte con lo stesso valore di preferenza locale, si seleziona quella con AS-PATH più breve. Se questa fosse l'unica regola di selezione, allora BGP utilizzerebbe un algoritmo Bellman-Ford per la determinazione del percorso, con metrica sulla distanza data dal numero di hop tra sistemi autonomi anziché il numero di hop tra router.
3. Tra le rotte con lo stesso valore di preferenza locale e la stessa lunghezza AS-PATH si seleziona quella il cui router di NEXT-HOP è più vicino. In questo caso, con “router più vicino” s'intende quello che presenta il percorso con costo minore, determinato dall'algoritmo intra-AS (instradamento a patata bollente).
4. Se rimane ancora più di una rotta, il router si basa sugli identificatori BGP [Stewart 1999].

Come esempio, si consideri il router 1b della Figura 5.10 e si ricordi che esistono due rotte BGP verso il prefisso  $x$ : una che passa da AS2 e una che lo evita; l'algoritmo hot potato sceglierebbe la prima. Al contrario qui la regola 2 verrebbe applicata prima della 3 e quindi verrebbe scelta la seconda rotta avente AS-PATH più breve. Tale politica BGP non è quindi egoista e riduce anche il ritardo end-to-end.

Come evidenziato, BGP è lo standard *de facto* per l'instradamento tra sistemi autonomi in Internet. I contenuti di svariate (assai grandi) tabelle di instradamento BGP tratte da router di ISP di livello 1, sono visibili in <http://www.roulevies.org>. Le tabelle di instradamento BGP contengono più di mezzo milione di rotte (prefisso e attributi). Infine, [Huston 2019b] riporta statistiche su dimensioni e caratteristiche delle tabelle di instradamento BGP.

**Figura 5.12** Utilizzo di anycast IP per instradare i client al cluster CDN più vicino.



#### 5.4.4 Anycast IP

Il protocollo BGP, oltre a essere il protocollo inter-AS di Internet, viene spesso utilizzato anche per implementare il servizio IP anycast [RFC 1546, RFC 7094], comunemente utilizzato in DNS. Per capire l'utilità di anycast si consideri che in molte applicazioni si è interessati a (1) replicare lo stesso contenuto su server differenti sparsi in diverse aree geografiche e (2) che un utente acceda al contenuto dal server che gli è più vicino. Per esempio, una CDN può replicare i video e gli altri oggetti sui suoi server in differenti paesi. Allo stesso modo, il sistema DNS può replicare i record DNS sui server DNS sparsi in tutto il mondo. È auspicabile che un utente che voglia accedere a un contenuto utilizzi il server a lui più vicino. L'algoritmo di selezione delle rotte di BGP fornisce un meccanismo semplice e naturale per tale operazione.

Facciamo ora un caso concreto di come una CDN possa utilizzare l'anycast IP. Come mostrato nella Figura 5.12, durante la configurazione dell'anycast IP, la CDN assegna lo stesso indirizzo IP a ognuno dei suoi server e utilizza lo standard BGP per annunciare tale indirizzo IP da ognuno dei suoi server. Quando un router BGP riceve l'annuncio di più percorsi da questo indirizzo IP, lo tratta come se fossero forniti diversi percorsi verso la stessa area fisica, anche se nella realtà sono percorsi diversi verso luoghi diversi. Ogni router, quando configura la sua tabella di inoltro, utilizza localmente l'algoritmo di selezione dei cammini BGP per scegliere il percorso migliore (per esempio il più vicino secondo il conteggio degli hop AS) per quell'indirizzo IP. Per esempio, se un cammino BGP, corrispondente a un luogo, è solo un hop AS lontano dal router, e tutti gli altri

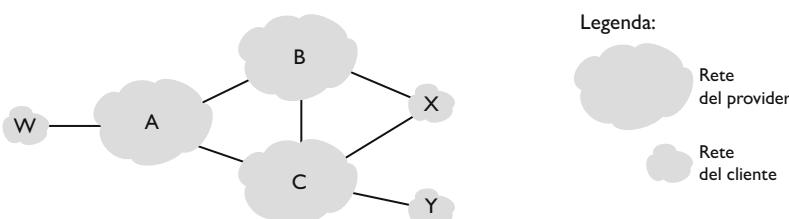
percorsi BGP corrispondenti ad altri luoghi sono distanti due o più hop AS, il router BGP sceglierrebbe di inviare i pacchetti al luogo distante un hop. Dopo la fase iniziale di annuncio degli indirizzi BGP, la CDN può dedicarsi a distribuire contenuti. Quando un client richiede in video, la CDN gli restituisce l'indirizzo IP usato da tutti i suoi server dispersi geograficamente, indipendentemente dal luogo in cui si trova il client. Quando il client invia una richiesta a quell'indirizzo IP, i router Internet inoltrano il pacchetto richiesto al server più vicino come definito dall'algoritmo BGP.

Sebbene tale esempio di CDN illustri bene come utilizzare l'anycast IP, nella pratica le CDN non lo utilizzano perché cambiamenti nell'instradamento BGP potrebbero portare ad avere sulla stessa connessione TCP pacchetti differenti che arrivano a differenti istanze del server web. Tuttavia, l'anycast IP viene diffusamente utilizzato dal sistema DNS per dirigere le richieste DNS al server root più vicino. Come visto nel Paragrafo 2.4, vi sono attualmente 13 indirizzi IP per i server root DNS, ma a ognuno di essi sono associati più server root, alcuni corrispondenti a più di 100 server distribuiti in tutto il mondo. Quando una richiesta DNS viene inviata a uno di questi 13 indirizzi IP, l'anycast IP viene utilizzato per instradare la richiesta al server root più vicino, responsabile per quell'indirizzo. [Li 2018] mostra alcune recenti misure dell'uso, delle prestazioni e delle sfide dell'anycast IP.

#### 5.4.5 Politiche di instradamento

Quando un router sceglie una rotta, le politiche di instradamento dell'AS possono intervenire facendo saltare qualsiasi altra considerazione riguardante per esempio la brevità del percorso o la procedura "hot potato".

Presentiamo alcuni concetti base dell'instradamento BGP ricorrendo a un semplice esempio. Si osservi la Figura 5.13, che mostra sei sistemi autonomi interconnessi: A, B, C, W, X e Y. È importante notare che si tratta di sistemi autonomi e non di router. Ipotizziamo che W, X e Y siano **reti stub** e che A, B e C siano reti di provider di dorsale. Ipotizziamo anche che A, B e C siano peer l'uno dell'altro e forniscano informazioni BGP complete alle reti dei loro clienti. Tutto il traffico che entra in una **rete di accesso di un ISP** è destinato a tale rete e tutto il traffico in uscita ha origine in tale rete. W e Y rappresentano chiaramente reti di accesso. X è un **ISP di accesso multi-homed**, dato che è connessa al resto della rete attraverso due diversi provider (uno scenario che è sempre più comune). In ogni caso, come W e Y, anche X deve essere sorgente o destinazione di tutto il traffico in uscita o ingresso attraverso X. Ma come sarà implementato e imposto tale comportamento? Come si impedirà a X di smaltire traffico tra B e C? Una risposta risiede nel controllo degli annunci delle rotte



**Figura 5.13**  
Un semplice scenario BGP.

BGP. In particolare,  $X$  opererà come rete stub se annuncia ai suoi vicini  $B$  e  $C$  di non avere percorsi verso altre destinazioni tranne sé stessa. In altre parole, anche se  $X$  fosse a conoscenza di un percorso del tipo  $XY$ , non l'annuncerà a  $B$ . Dato che questo ignora che  $X$  ha un percorso verso  $Y$ , non inoltrerà mai traffico destinato a  $Y$  (né a  $C$ ) attraverso  $X$ . Questo semplice esempio mostra come la politica selettiva di annuncio del percorso possa gestire relazioni di instradamento tra cliente e provider.

Concentriamoci ora sulla rete di un provider, per esempio il sistema autonomo  $B$ , e supponiamo che questo abbia appreso (da  $A$ ) che  $A$  presenta un percorso  $AW$  verso  $W$ . Allora  $B$  può installare il percorso  $BAW$  nella sua struttura dati per l'instradamento. Chiaramente,  $B$  vuole anche annunciare il percorso  $BAW$  al suo cliente  $X$ , per permettergli di raggiungere  $W$  tramite  $B$ . Ma dovrebbe forse  $B$  annunciarlo anche a  $C$ ? In questo caso,  $C$  potrebbe instradare il traffico verso  $W$  via  $CBAW$ . Se  $A$ ,  $B$  e  $C$  sono provider di dorsale, allora  $B$  potrebbe giustamente pensare di non dover sopportare il peso (e il costo) del traffico in transito tra  $A$  e  $C$  e che sia compito di  $A$  e  $C$  assicurarsi che  $C$  possa instradare da e verso i clienti di  $A$  tramite una connessione diretta tra  $A$  e  $C$ . Attualmente non esistono standard ufficiali per gestire l'instradamento reciproco degli ISP di dorsale. Una regola pratica seguita dagli ISP commerciali è quella per cui tutto il traffico che fluisce attraverso la rete di dorsale di un ISP deve avere origine e/o destinazione in una sua rete cliente. Accordi individuali di **peering** (che possono risolvere problemi del genere) vengono negoziati tra coppie di ISP, sovente in via confidenziale. Per approfondimenti in merito a questo tipo di accordi si può consultare [Huston 1999a; Houston 2012]. Una descrizione dettagliata dell'influenza delle politiche di instradamento sulle relazioni commerciali tra ISP è invece presente in [Gao 2001; Dimitropoulos 2007]. Per una trattazione delle politiche di instradamento BGP dal punto di vista degli ISP si veda [Caesar 2005b].

Con questo si completa la nostra breve introduzione a BGP. La comprensione dell'argomento è importante dato che BGP gioca un ruolo determinante per Internet. Per ulteriori approfondimenti vi suggeriamo di consultare [Stewart 1999; Huston 2019a; Labovitz 1997; Halabi 2000; Huitema 1998; Gao 2001; Feamster 2004; Caesar 2005b; Li 2007].

#### 5.4.6 Retrospettiva: come essere presenti in Internet

Questo paragrafo, pur non trattando strettamente BGP, metterà insieme molti protocolli e concetti che abbiamo visto finora, tra cui l'indirizzamento IP, DNS e BGP.

Supponete d'aver appena creato una piccola azienda con un certo numero di server, tra cui un server web che espone i prodotti e i servizi dell'azienda, un mail server per gestire le e-mail aziendali e un server DNS. Naturalmente volete che tutto il mondo sia in grado di visitare il vostro sito web e che i dipendenti possano scambiare e-mail con i potenziali clienti di tutto il mondo.

Per raggiungere tali obiettivi avete innanzitutto bisogno di essere connessi a Internet tramite un ISP locale. La vostra azienda avrà così un router gateway che verrà connesso a uno dei router del vostro ISP locale con una connessione DSL tramite l'infrastruttura telefonica già esistente, una linea dedicata verso il router dell'ISP o una qualsiasi delle altre soluzioni di accesso descritte nel

**BOX 5.2** **TEORIA E PRATICA****Protocolli di instradamento inter-AS e intra-AS**

Avendo studiato alcuni dettagli dei protocolli di instradamento inter-AS e intra-AS utilizzati in Internet prendiamo ora in considerazione la domanda principale riguardo questi protocolli: perché si usano algoritmi di instradamento diversi?

La risposta concerne i differenti obiettivi dell'instradamento all'interno di un AS e tra AS.

- **Politiche.** A causa delle questioni amministrative che dominano gli AS, potrebbe risultare determinante che il traffico originato da uno non passi attraverso un altro; oppure un AS vorrebbe poter controllare il traffico di altri sistemi che transita attraverso le sue reti. Abbiamo visto che BGP trasporta attributi di percorso e si occupa della distribuzione controllata delle informazioni di instradamento, per consentire decisioni di instradamento basate su politiche. All'interno di un AS, esiste un solo controllo amministrativo e di conseguenza le questioni legate a politiche hanno un ruolo molto meno importante nello scegliere i percorsi interni.
- **Scalabilità.** La capacità degli algoritmi di gestire l'instradamento verso e tra un gran numero di reti rappresenta un problema critico per gli AS. All'interno di un sistema, la scalabilità è una questione assai meno importante. Se un singolo dominio amministrativo diventa troppo grande, è sempre possibile ripartirlo ed effettuare instradamento tra i sottosistemi. Ricordiamo che OSPF consente la costruzione di una simile gerarchia dividendo gli AS in aree.
- **Prestazioni.** Dato che l'instradamento tra AS è principalmente governato dalle politiche, le prestazioni dei percorsi utilizzati sono spesso di secondaria importanza. In altre parole, un percorso più lungo o più costoso che soddisfi determinati criteri potrebbe essere preferibile a un percorso più breve, ma che non rispetta i canoni richiesti. Infatti, abbiamo anche visto che tra sistemi autonomi non esiste neanche la nozione di costo del percorso (al di là del conteggio degli hop). In ogni caso, nel singolo AS, questo tipo di questioni riveste minore importanza, per cui l'instradamento può concentrarsi maggiormente sulle prestazioni ottenute sui percorsi.

Capitolo 1. Il vostro ISP locale vi fornirà anche un blocco di indirizzi IP, per esempio un /24 consistente di 256 indirizzi. Una volta che avrete ottenuto connettività fisica e il blocco di indirizzi IP, potrete assegnare uno dei vostri indirizzi IP al vostro server web, uno al mail server, uno al DNS server, uno al router gateway e tutti gli altri indirizzi IP ai server e ai dispositivi in rete della vostra azienda.

Dovrete anche registrare il nome del dominio della vostra azienda presso un ente di registrazione, come descritto nel Capitolo 2. Per esempio, se la vostra azienda si chiama Xanadu Inc., cercherete naturalmente di ottenere il nome di dominio xanadu.com. La vostra azienda deve anche essere registrata nel sistema DNS: affinché da fuori sia possibile contattare il vostro server DNS per ottenere gli indirizzi IP dei vostri server, dovete fornire l'indirizzo IP del vostro server DNS all'ente di registrazione, che quindi immetterà un record del vostro server DNS (nome del dominio e indirizzo IP corrispondente) nel server TLD .com, come descritto nel Capitolo 2. A questo punto chiunque conosca il vostro nome di dominio, xanadu.com, potrà ottenere l'indirizzo IP del vostro server DNS dal sistema DNS.

Affinché le persone possano scoprire gli indirizzi IP del vostro server web, dovrete inserire nel vostro server DNS un record che mappi il nome dell'host del server web (per esempio www.xanadu.com) al suo indirizzo IP. Dovrete inoltre inserire record simili per altri server disponibili pubblicamente, quali il vostro mail server. In questo modo, se Alice volesse visitare il vostro server web, il sistema DNS contatterebbe il vostro server DNS, troverebbe l'indirizzo IP del vostro server web e lo invierebbe ad Alice che quindi potrà stabilire una connessione TCP direttamente con il vostro server web.

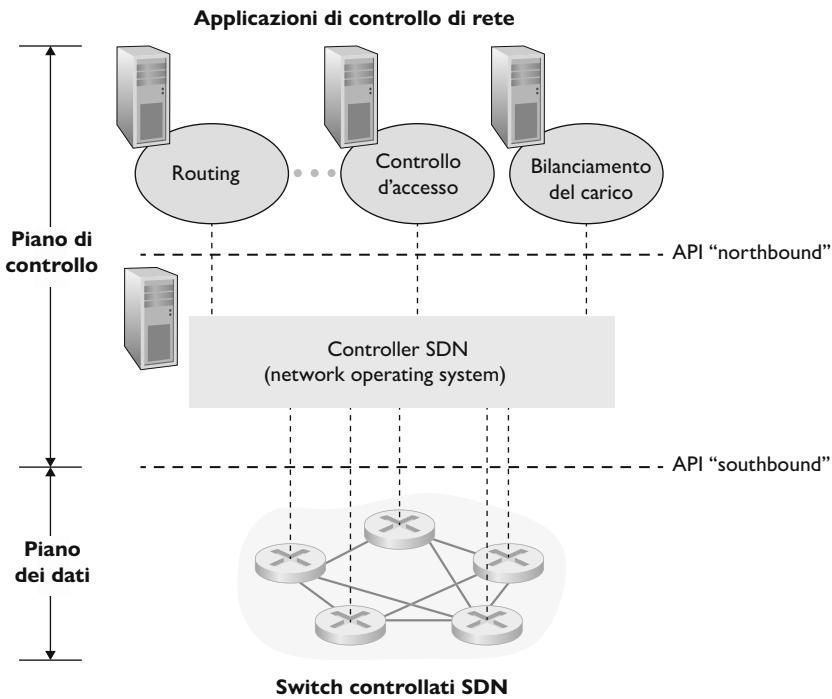
Rimane tuttavia un altro passo fondamentale per permettere di accedere al vostro server web dal mondo esterno. Considerate che cosa succede quando Alice, che conosce l'indirizzo IP del vostro server web, invia un datagramma IP (per esempio un segmento TCP SYN) a quell'indirizzo IP. Tale datagramma verrà instradato attraverso Internet passando una serie di router in molti differenti AS e infine raggiungerà il vostro server web. Ogni router che riceve il datagramma cerca una corrispondenza nella sua tabella di inoltro per determinare la porta di uscita. Quindi ogni router deve conoscere il prefisso /24 della vostra azienda, o almeno un'occorrenza aggregata. Ma come fa il router a venire a conoscenza del prefisso della vostra azienda? Come abbiamo già visto, tramite BGP! Quando la vostra azienda sigla un contratto con l'ISP locale e ottiene un prefisso, l'ISP locale usa BGP per annunciare il vostro prefisso agli ISP a cui è connesso, che a loro volta propagheranno tale annuncio usando BGP. Alla fine tutti i router di Internet conosceranno il vostro prefisso e saranno in grado di inoltrare i datagrammi destinati ai vostri server.

## 5.5 Il piano di controllo SDN

In questo paragrafo approfondiremo il piano di controllo SDN, la logica globale di rete che controlla l'inoltro dei pacchetti tra i dispositivi di una rete SDN, così come la configurazione e la gestione di tali dispositivi e dei loro servizi. La trattazione si baserà sulla discussione sull'inoltro generalizzato dei Paragrafi 4.4 e 5.1. Anche qui useremo la terminologia adottata nella letteratura su SDN e chiameremo i dispositivi di rete abilitati all'inoltro con il termine “packet switch” (commutatore di pacchetto), o anche solo “switch”, in quanto le decisioni di inoltro possono essere prese sulla base degli indirizzi di sorgente/destinazione a livello di rete o a livello di collegamento, così come sulla base dei valori dei campi dell'intestazione del livello di trasporto, rete e collegamento.

In un'architettura SDN possono essere identificate le seguenti quattro caratteristiche fondamentali [Kreutz 2015].

- *Inoltro basato sui flussi.* L'inoltro dei pacchetti può essere effettuato da uno switch SDN sulla base del valore dei campi dell'intestazione a livello di trasporto, di rete e di collegamento. Abbiamo visto nel Paragrafo 4.4 che l'estrazione di OpenFlow permette di effettuare l'inoltro sulla base del valore di undici campi dell'intestazione. Questa modalità è in netto contrasto con l'approccio tradizionale dell'inoltro dei router che abbiamo visto nei Paragrafi 5.2 – 5.4, dove l'inoltro dei datagrammi IP era basato esclusivamente sull'indirizzo IP di destinazione del datagramma. Si ricordi dalla Figura 5.2 che le regole di inoltro dei pacchetti sono specificate nella tabella dei flussi degli



**Figura 5.14**  
Componenti dell'architettura SDN:  
switch controllati,  
controller, applicazioni di  
controllo di rete.

switch; è compito del piano di controllo SDN calcolare, gestire e installare le occorrenze della tabella dei flussi in tutti gli switch della rete.

- *Separazione del piano dei dati e del piano di controllo.* Tale separazione è mostrata chiaramente nelle Figure 5.2 e 5.14. Il piano dei dati consiste di switch di rete, dispositivi relativamente semplici, ma veloci, che eseguono le regole “match-action” nelle loro tabelle dei flussi. Il piano di controllo consiste di server e software che determinano e gestiscono le tabelle dei flussi degli switch.
- *Funzioni di controllo di rete: esterne agli switch del piano dei dati.* Dato che la “S” in SDN sta per “software” non è sorprendente che il piano di controllo SDN sia implementato in software. Al contrario dei router tradizionali, tuttavia, tale software viene eseguito su server distinti e remoti rispetto agli switch della rete. Come mostrato nella Figura 5.14, lo stesso piano di controllo consiste di due componenti: un controller SDN (o sistema operativo di rete [Gude 2008]) e un insieme di applicazioni di controllo di rete. Il controller mantiene informazioni di stato della rete quali per esempio lo stato dei collegamenti, degli switch e degli host remoti; fornisce tali informazioni alle applicazioni di controllo di rete eseguite nel piano di controllo e fornisce il mezzo attraverso il quale tali applicazioni possono monitorare, programmare e controllare i sottostanti dispositivi di rete. Anche se il controller è mostrato nella Figura 5.14 come un unico server centralizzato, nella pratica è centralizzato solo dal punto di vista logico, ma tipicamente è implementato su più server.
- *Una rete programmabile.* La rete è programmabile attraverso le applicazioni di controllo di rete che vengono eseguite nel piano di controllo. Tali applicazioni rappresentano il “cervello” del piano di controllo SDN e usano le API

fornite dal controller SDN per specificare e controllare il piano dei dati nei dispositivi di rete. Per esempio, l'applicazione di instradamento potrebbe determinare i percorsi tra sorgente e destinazione (per esempio, eseguendo l'algoritmo di Dijkstra usando le informazioni di stato mantenute dal controller SDN). Un'altra applicazione di rete potrebbe effettuare il controllo di accesso, determinando quali pacchetti debbano essere bloccati a uno switch, come nel terzo esempio del Paragrafo 4.4.3. Un'altra applicazione potrebbe inoltrare i pacchetti in modo da effettuare il bilanciamento di carico dei server, come visto nel secondo esempio del Paragrafo 4.4.3.

Dalla nostra trattazione si evince che le SDN rappresentano una separazione delle funzionalità di rete: gli switch del piano dei dati, i controller SDN e le applicazioni di controllo di rete sono entità distinte che possono essere fornite da differenti fornitori e organizzazioni. Tutto ciò contrasta con il modello precedente SDN, nel quale uno switch/router, insieme al suo software di controllo e alle implementazioni di protocollo, era monolitico, integrato verticalmente e venduto da un singolo fornitore. Tale separazione delle funzionalità di rete nelle SDN è stata paragonata alla precedente evoluzione dai computer mainframe (nei quali l'hardware, il software di sistema e le applicazioni erano forniti da un singolo fornitore) ai personal computer aventi hardware, sistema operativo e applicazioni separate. La separazione dell'hardware, del software di sistema e delle applicazioni ha probabilmente portato a un ecosistema ricco e aperto guidato dall'innovazione in tutte queste tre aree; speriamo che sia così anche per le SDN.

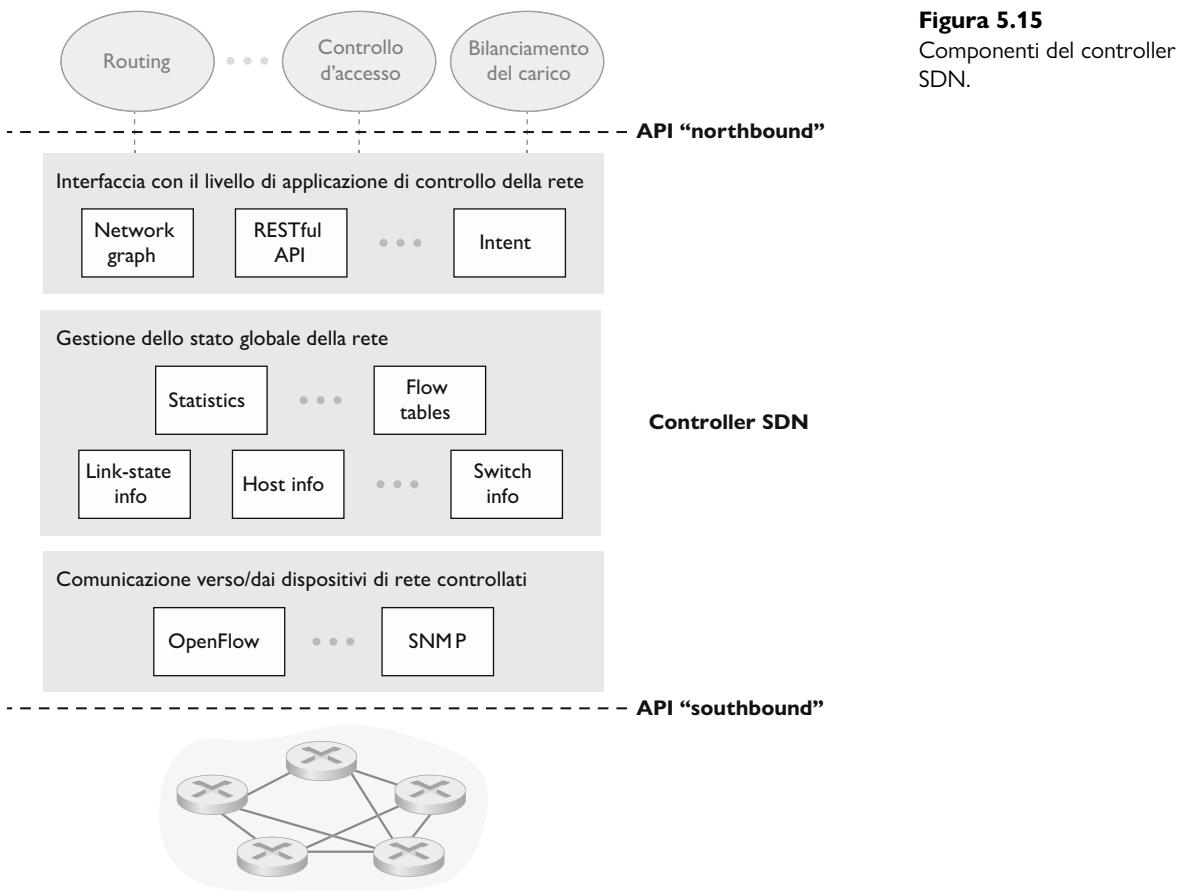
Osservando la Figura 5.14 riguardante l'architettura SDN, sorgono molte domande. Come e dove vengono realmente calcolate le tabelle dei flussi? Come tali tabelle vengono aggiornate in risposta a eventi che avvengono nei dispositivi SDN, quali per esempio un collegamento che si disconnette o si connette? E come vengono coordinate le occorrenze delle tabelle dei flussi di più switch in modo che siano coerenti a livello di tutta la rete (per esempio i percorsi end-to-end dei pacchetti inoltrati da sorgenti a destinazioni o i firewall distribuiti)? Tali abilità fanno parte del ruolo del piano di controllo SDN.

### **5.5.1 Il piano di controllo SDN: controller SDN e applicazioni di controllo**

Cominciamo a trattare il piano di controllo SDN dal punto di vista astratto, considerando le capacità generiche che esso deve fornire. Come vedremo tale approccio astratto porterà a un'architettura che riflette come il piano di controllo SDN è poi implementato in pratica.

Come già visto, il piano di controllo SDN si divide a grandi linee in due componenti: il controller SDN e le applicazioni di controllo SDN. Cominciamo dal primo [Gude 2008; Kreutz 2015]. La Figura 5.15 fornisce una descrizione dettagliata di un generico controller SDN. Le sue funzionalità possono essere divise in tre livelli che descriviamo ora con un approccio bottom-up:

- *Un livello di comunicazione*: effettua le comunicazioni tra il controller SDN e i dispositivi di rete controllati. Chiaramente, se un controller SDN deve controllare le operazioni di uno switch, un host o un altro dispositivo da remoto ha bisogno di un protocollo che trasferisca le informazioni tra di essi. Inoltre un dispositivo deve essere in grado di comunicare al controller eventi



che osserva localmente, quali un messaggio che indichi che un collegamento è stato connesso o disconnesso, un dispositivo è stato unito alla rete o un dispositivo è diventato operativo. Tali eventi forniscono al controller SDN una visione aggiornata dello stato della rete. Tale protocollo costituisce il livello più basso dell'architettura del controller, come mostrato nella Figura 5.15. La comunicazione tra il controller e i dispositivi controllati passa attraverso un'interfaccia del controller nota come interfaccia "southbound". Nel Paragrafo 5.5.2 studieremo OpenFlow, un protocollo specifico che fornisce tale funzionalità di comunicazione, implementato nella maggior parte dei controller SDN, se non in tutti.

- *Un livello di gestione dello stato globale della rete.* Le decisioni di controllo finali prese dal piano di controllo SDN richiedono che il controller abbia informazioni aggiornate sullo stato di host, link, switch e altri dispositivi SDN controllati della rete. Una tabella dei flussi di uno switch contiene contatori i cui valori possono essere utilizzati proficuamente dalle applicazioni di controllo di rete e pertanto devono essere disponibili. Poiché lo scopo finale del piano di controllo è determinare le tabelle dei flussi dei vari dispositivi controllati, un controller potrebbe anche mantenere una copia di tali tabelle. Tali informazioni costituiscono insieme un esempio dello stato globale della rete mantenuto da un controller SDN.

- *L’interfaccia con il livello di applicazione di controllo della rete.* Il controller interagisce con le applicazioni di controllo di rete attraverso la sua interfaccia “northbound”. Tali API permettono alle applicazioni di leggere/scrivere lo stato della rete e le tabelle dei flussi nel livello di gestione dello stato di rete. Le applicazioni possono richiedere una notifica quando avviene un evento di cambiamento di stato, in modo che possano intraprendere azioni in risposta alle notifiche di evento di rete inviate da un dispositivo controllato. Possono essere forniti diversi tipi di API; i più comuni controller SDN comunicano tramite l’interfaccia REST [Fielding 2000].

Abbiamo già menzionato più volte come un controller SDN possa essere considerato “logicamente centralizzato”, in quanto può essere visto dall’esterno (per esempio dal punto di vista dei dispositivi controllati e delle applicazioni di controllo esterne) come un servizio singolo e monolitico. Tuttavia, tali servizi e i database usati per mantenere le informazioni di stato sono nella pratica implementati da un insieme distribuito di server per ragioni di tolleranza agli errori, alte disponibilità e prestazioni. In tal modo però bisogna tener conto della semantica delle operazioni interne dei controller, come mantenere il tempo logico degli eventi, la coerenza, il consenso e altre ancora [Panda 2013]. Tali problematiche sono comuni a molti sistemi distribuiti; si consulti [Lamport 1989; Lampson 1996] per una trattazione di alcune eleganti soluzioni a queste sfide. I controller moderni quali OpenDaylight [OpenDaylight 2020] e ONOS [ONOS 2020] pongono grande enfasi sulla progettazione di una piattaforma di controller logicamente centralizzata, ma fisicamente distribuita, che fornisca servizi scalabili ed elevata disponibilità ai dispositivi controllati e alle applicazioni di controllo.

L’architettura mostrata nella Figura 5.15 richiama l’architettura del controller originario NOX proposto nel 2008 [Gude 2008], così come quella dei moderni controller SDN [OpenDaylight 2020] e ONOS [ONOS 2020] (si veda il Box 5.3). Vedremo un esempio di operazione di un controller nel Paragrafo 5.5.3, mentre ora esaminiamo il protocollo OpenFlow del livello di comunicazione.

### 5.5.2 Il protocollo OpenFlow

Il protocollo OpenFlow [OpenFlow 2009; ONF 2020] opera tra un controller SDN e uno switch o un altro dispositivo che implementi le API OpenFlow viste nel Paragrafo 4.4. Il protocollo opera su TCP con numero di porta 6653.

Alcuni dei principali messaggi del protocollo inviati dal controller allo switch controllato sono i seguenti:

- **Configuration.** Questo messaggio permette al controller di interrogare e impostare i parametri di configurazione di uno switch.
- **Modify-State.** Questo messaggio viene usato dal controller per aggiungere/cancellare o modificare le occorrenze della tabella dei flussi dello switch e per impostare le proprietà delle porte dello switch.
- **Read-State.** Questo messaggio è usato dal controller per raccogliere le statistiche e i valori dei contatori nelle tabelle dei flussi e nelle porte dello switch.
- **Send-Packet.** Questo messaggio è usato dal controller per inviare un pacchetto specifico fuori da una specifica porta dello switch; il messaggio stesso contiene il pacchetto da inviare nel suo carico.

**BOX 5.3****TEORIA E PRATICA****La rete globale SDN di Google**

Come visto nel Paragrafo 2.6, Google ha allestito una sua WAN dedicata che interconnette i suoi data center e cluster di server (in IXP e ISP). Questa rete, chiamata B4, ha un piano di controllo SDN progettato da Google e costruito su OpenFlow. La rete di Google è in grado di utilizzare a circa il 70% (da due a tre volte il valore di utilizzazione tipico) i suoi collegamenti e di suddividere i flussi di applicazione tra percorsi multipli in base alla priorità e alle richieste dei flussi esistenti [Jain 2013].

Tale rete è particolarmente adatta a SDN: (1) Google controlla tutti i dispositivi, dai server della periferia di IXP e ISP fino ai router all'interno della rete; (2) le applicazioni che richiedono più banda sono duplicate su larga scala su siti che possono rinviare ad applicazioni interattive ad alta priorità quando sorge una congestione; (3) il controllo centralizzato è fattibile in quanto solo una dozzina di data center sono connessi.

La rete B4 di Google usa switch progettati ad hoc che implementano una versione di OpenFlow estesa nella quale un agente OpenFlow locale (OFA) è simile all'agente di controllo visto nella Figura 5.2. Ogni OFA si connette a un controller OpenFlow (OFC) nel server di controllo della rete (NCS) usando una rete separata fuori banda, distinta dalla rete che trasporta il traffico tra i data center. OFC quindi fornisce il servizio di comunicazione tra NCS e i suoi switch controllati, in modo simile al livello più basso dell'architettura SDN mostrata nella Figura 5.15. Inoltre OFC effettua le funzioni di gestione di stato, mantenendo lo stato dei nodi e dei link nel database NIB (Network Information Base). L'implementazione di Google dell'OFC è basata sul controller ONIX [Koponen 2010]. Sono implementati due protocolli di instradamento: BGP per l'instradamento tra data center e IS-IS, un protocollo simile a OSPF per l'instradamento all'interno dei data center. Paxos [Chandra 2007] viene usato per eseguire replicate delle componenti NCS per proteggere il sistema da guasti.

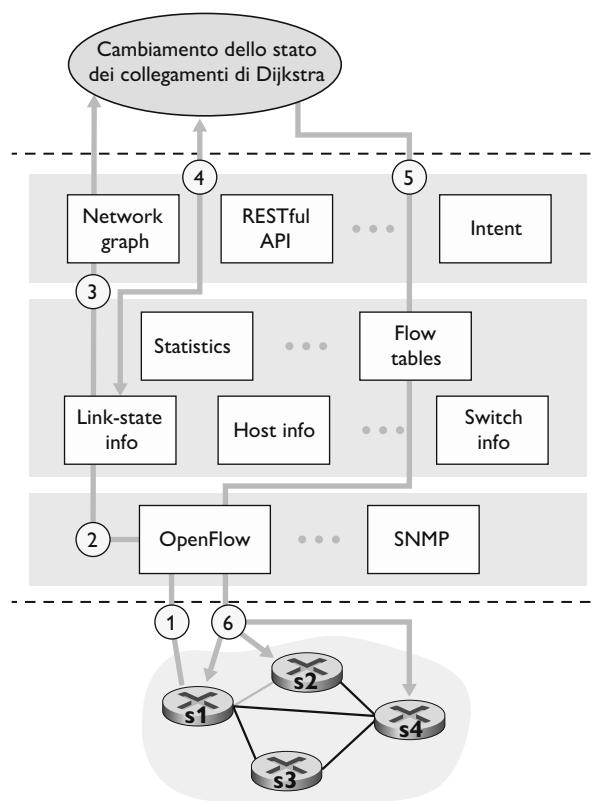
Un'applicazione di controllo del traffico, che si trova logicamente al di sopra dei server di controllo della rete, interagisce con tali server per fornire banda globale ai gruppi di flussi delle applicazioni. Con la rete B4, SDN ha fatto un importante salto in avanti verso una rete operativa di un fornitore di rete globale. Si veda [Jain 2013; Hong 2018] per una descrizione dettagliata della rete B4.

Alcuni dei più importanti messaggi del protocollo inviati dallo switch controllato al controller sono i seguenti:

- **Flow-Removed.** Questo messaggio informa il controller che un'occorrenza della tabella dei flussi è stata cancellata, per esempio per un evento di timeout o per un messaggio *modify-state* ricevuto.
- **Port-status.** Questo messaggio è usato dallo switch per informare il controller di un cambiamento nello stato di una porta.
- **Packed-in.** Come visto nel Paragrafo 4.4, un pacchetto in arrivo su una porta di uno switch che non abbia corrispondenza alcuna nella tabella dei flussi viene inviato al controller per ulteriori elaborazioni. Inoltre anche i pacchetti che abbiano trovato una corrispondenza possono essere inviati al controller in seguito a un'azione che deve essere intrapresa a seguito della corrispondenza. Il messaggio packed-in è usato per inviare tali pacchetti al controller.

Ulteriori messaggi OpenFlow sono definiti in [OpenFlow 2009; ONF 2020].

**Figura 5.16** Controller SDN con cambiamento dello stato dei collegamenti.



### 5.5.3 Interazione tra piano dei dati e piano di controllo: un esempio

Per consolidare quanto visto finora, si consideri l'esempio mostrato nella Figura 5.16, nel quale l'algoritmo di Dijkstra studiato nel Paragrafo 5.2 è usato per determinare i cammini più brevi. Lo scenario SDN della Figura 5.16 presenta due importanti differenze rispetto allo scenario riguardante il controllo per router visto nei Paragrafi 5.2.1 e 5.3, dove l'algoritmo di Dijkstra era implementato su tutti i router e gli aggiornamenti sui collegamenti erano inviati a tutti i router della rete:

- l'algoritmo di Dijkstra viene eseguito come un'applicazione separata, esterna agli switch;
- gli switch inviano gli aggiornamenti sui collegamenti solo al controller SDN e non li scambiano tra di loro.

In questo esempio si assume che il collegamento tra lo switch s1 e lo switch s2 cada; che si implementi l'instradamento basato sul cammino minimo e che le regole di inoltro dei flussi in entrata e in uscita in s1, s3 e s4 ne siano condizionati, mentre le operazioni in s2 rimangono inalterate. Si assume inoltre che OpenFlow venga utilizzato come protocollo a livello di comunicazione e che il piano di controllo effettui unicamente la funzione di instradamento sulla base dello stato dei collegamenti.

1. Lo switch s1 osserva la caduta del collegamento con s2 e notifica tale cambiamento nello stato del collegamento al controller SDN utilizzando un messaggio OpenFlow port-status.
2. Il controller SDN riceve il messaggio OpenFlow con il cambiamento dello stato del collegamento, lo notifica al gestore dello stato del collegamento che aggiorna il database degli stati dei collegamenti.
3. L'applicazione di controllo di rete che implementa l'algoritmo di Dijkstra riceve la notifica del cambiamento dello stato del collegamento.
4. L'applicazione di instradamento interagisce con il gestore dello stato del collegamento per ottenere lo stato aggiornato del collegamento; potrebbe anche consultare altre componenti del livello di gestione dello stato. Quindi computa i nuovi cammini minimi.
5. L'applicazione di instradamento interagisce quindi con il gestore della tabella dei flussi che determina quali tabelle debbano essere aggiornate.
6. Il gestore della tabella dei flussi usa quindi il protocollo OpenFlow per aggiornare le occorrenze della tabella dei flussi negli switch coinvolti: s1, che adesso instrada i pacchetti destinati a s2 attraverso s4, s2 che adesso inizia a ricevere i pacchetti da s1 attraverso s4, e s4 che adesso deve inoltrare i pacchetti da s1 destinati a s2.

Questo esempio, benché semplice, illustra come il piano di controllo SDN fornisca servizi di controllo prima implementati localmente su ogni router della rete. Si può facilmente apprezzare come un ISP abilitato SDN possa facilmente passare da un instradamento basato sui percorsi a minimo costo a un instradamento più personalizzato. Poiché il controller può scrivere le tabelle di flusso come vuole, può implementare qualunque forma di inoltro desideri, semplicemente cambiando il suo software di applicazione di controllo, mentre tradizionalmente era necessario cambiare il software di tutti i router dell'ISP, magari forniti da differenti aziende.

#### **5.5.4 SDN: il passato e il futuro**

Nonostante l'interesse nelle SDN sia un fenomeno relativamente recente, le basi della SDN e in particolare la separazione tra il piano dei dati e quello di controllo è più datata. Nel 2004 [Fteamster 2004; Lakshman 2004; RFC3746] trattavano la separazione tra i due piani. [van der Merwe 1998] descrive un'architettura di controllo per le reti ATM [Black 1995] con controller multipli, in cui ognuno controlla un certo numero di switch ATM. Il progetto Ethane [Casado 2007] fu un pioniere della nozione di una rete di switch Ethernet basati sui flussi con modalità match-action, un controller centralizzato che gestisca l'ammissione e l'instradamento dei flussi e l'inoltro dei pacchetti senza corrispondenza tra lo switch e il controller. Nel 2007 era operativa una rete con più di 300 switch Ethane. Il progetto Ethane si è evoluto velocemente nel progetto OpenFlow e il resto è storia!

Molte ricerche hanno lo scopo di sviluppare future architetture SDN. Come abbiamo visto, la rivoluzione SDN sta portando alla sostituzione degli switch e dei router dedicati con semplice hardware di commutazione e un piano di controllo sofisticato effettuato via software. Una generalizzazione delle SDN nota come **virtualizzazione delle funzioni di rete** (*NVF, network functions virtualization*)

**BOX 5.4****TEORIA E PRATICA****I controller SDN: OpenDaylight e ONOS**

All'inizio di SDN esisteva un solo protocollo SDN (OpenFlow [McKeown 2008; OpenFlow 2009]) e un solo controller SDN (NOX [Gude 2008]). Da allora sono stati sviluppati molti altri controller SDN [Kreutz 2015]. Alcuni controller sono proprietari, ma molti altri controller sono open-source e implementati in diversi linguaggi di programmazione [Erickson 2013]. Più recentemente il controller OpenDaylight [OpenDaylight 2020] e il controller ONOS [ONOS 2020] hanno ricevuto un considerevole supporto industriale. Sono entrambi open-source e sviluppati in partnership con Linux Foundation.

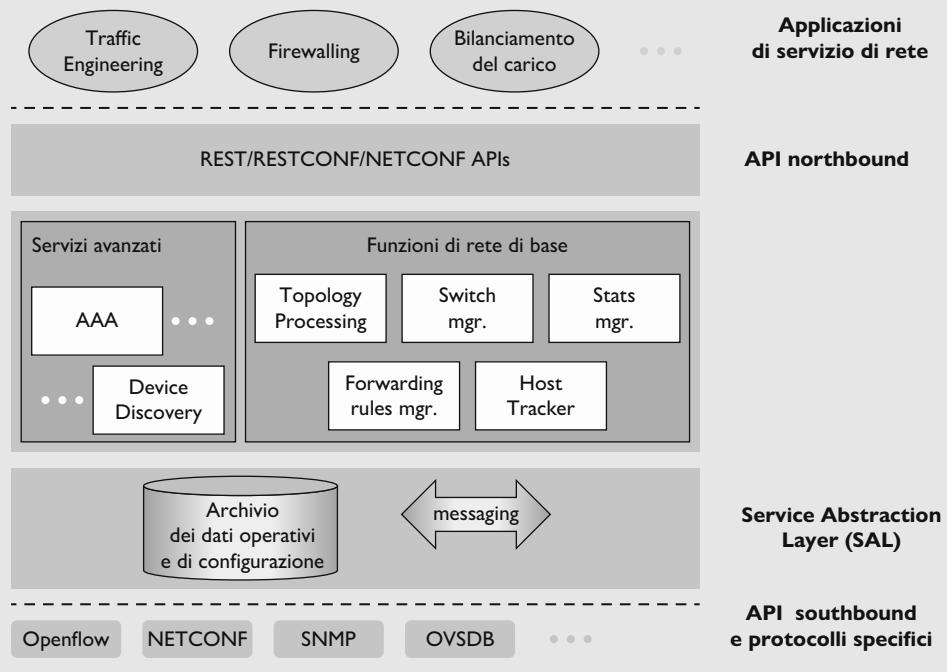
**Il controller OpenDaylight**

La Figura 5.17 mostra una versione semplificata del controller SDN OpenDaylight [ODL] [OpenDaylight 2020, Eckel 2017].

Le funzioni di base del servizio di rete ODL rappresentano il cuore del controller e corrispondono alle funzioni di gestione dello stato della rete globale viste nella Figura 5.15. Il SAL (Service Abstraction Layer) è il nervo centrale del controller che permette alle sue componenti e applicazioni di invocare reciprocamente i servizi, accedere ai dati di configurazione e operativi ed effettuare la sottoscrizione agli eventi. Fornisce anche un'interfaccia astratta uniforme a specifici protocolli che operano tra il controller ODL e i dispositivi, tra i quali OpenFlow, trattato nel Paragrafo 4.5, SNMP (Simple Network Management Protocol) e NETCONF che verranno trattati nel Paragrafo 5.7. OVSDP (Open vSwitch Database Management Protocol) è un protocollo usato per gestire gli switch dei data center, un'applicazione molto importante nella tecnologia SDN. Tratteremo le reti nei data center nel Capitolo 6.

Le applicazioni di servizio di rete sono le applicazioni che determinano come l'inoltro del piano dei dati e altri servizi come il firewalling e il bilanciamento di carico siano effettuati negli switch controllati. Il controller ODL ha due interfacce attraverso cui le applicazioni possono comunicare con i servizi del controller e tra di loro; nell'approccio basato sulle API (AD-SAL), mostrato nella Figura 5.17, le applicazioni comunicano con i moduli del controller usando le API REST eseguite sopra HTTP.

**Figura 5.17** Una visione semplificata del controller OpenDaylight.

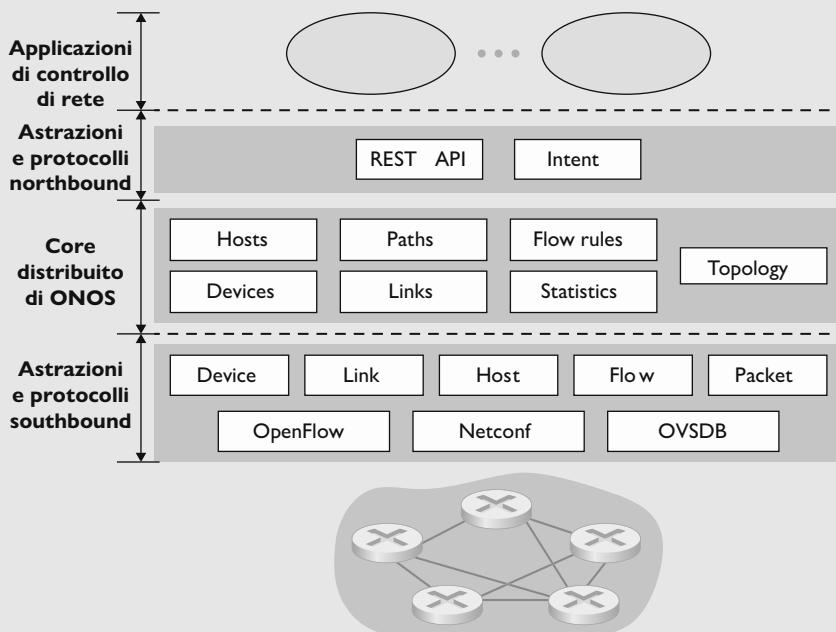


Le versioni iniziali del controller OpenDaylight fornivano solo AD-SAL ma, diventando ODL sempre più utilizzato per la configurazione e la gestione della rete, le versioni successive di ODL hanno introdotto un approccio Model-Driven (MD-SAL). Qui, il linguaggio di modellazione dei dati YANG [RFC 6020] definisce il modello dei dati di configurazione di rete e operativi. I dispositivi vengono quindi configurati e gestiti manipolando questi dati per mezzo del protocollo NETCONF.

## Il controller ONOS

La Figura 5.18 mostra una versione semplificata del controller ONOS [ONOS 2020]. Come nel controller canonico mostrato nella Figura 5.15, si possono identificare tre livelli:

- **Astrazioni e protocolli northbound.** Una caratteristica unica di ONOS è quella di permettere che un'applicazione richieda un servizio ad alto livello (per esempio di instaurare una connessione tra l'host A e l'host B o di non permettere ai due di comunicare) senza sapere come tale servizio venga effettuato. Le informazioni di stato vengono fornite alle applicazioni di controllo tramite le API *northbound* sia in modo sincrono, via interrogazione, sia in modo asincrono.
- **Core distribuito.** Lo stato di collegamenti, host e dispositivi è mantenuto nel core distribuito di ONOS. ONOS è allestito come un servizio su un insieme di server interconnessi, ognuno dei quali esegue una copia identica del software ONOS. Il core ONOS fornisce i meccanismi per replicare il servizio ed effettuare il coordinamento tra le istanze. In questo modo fornisce alle applicazioni sovrstanti e ai dispositivi di rete sottostanti l'astrazione di servizi di core logicamente centralizzati.
- **Astrazioni e protocolli southbound.** Le astrazioni *southbound* mascherano l'eterogeneità di host, collegamenti, switch e protocolli sottostanti permettendo al core distribuito di essere indipendente da dispositivi e protocolli. Data questa astrazione, l'interfaccia *southbound* sottostante al core distribuito sta logicamente più in alto del controller canonico della Figura 5.14 o del controller ODL della Figura 5.17.



ha un simile scopo: la sostituzione dei sofisticati middlebox come quelli con hardware dedicato e software proprietario per i servizi multimediali con semplici server, switch e memorie [Gember-Jacobson 2014]. Un’altra importante area di ricerca cerca di estendere i concetti SDN anche a scenari inter-AS [Gupta 2014].

## 5.6 ICMP (Internet Control Message Protocol)

Il protocollo ICMP (*Internet Control Message Protocol*) [RFC 792] viene usato da host e router per scambiarsi informazioni a livello di rete: il suo uso più tipico è la notifica degli errori. Durante l’esecuzione di una sessione HTTP potreste per esempio incontrare il messaggio “Rete di destinazione irraggiungibile” (*Destination network unreachable*). Questo messaggio ha origine da ICMP. Da qualche parte, un router IP non è stato in grado di trovare un percorso verso l’host specificato nella vostra applicazione Telnet (FTP, HTTP) e quindi ha composto e poi inviato al vostro host un messaggio ICMP di tipo 3 che indica l’errore.

ICMP è spesso considerato parte di IP, ma dal punto di vista dell’architettura si trova esattamente sopra IP, dato che i suoi messaggi vengono trasportati nei datagrammi IP: ossia, i messaggi ICMP vengono trasportati come payload di IP, esattamente come i segmenti TCP o UDP. Allo stesso modo, se un host riceve un datagramma IP, che specifica ICMP come protocollo di livello superiore, allora effettua il demultiplexing dei contenuti del datagramma a ICMP, esattamente come farebbe per contenuti TCP o UDP.

I messaggi ICMP hanno un campo tipo e un campo codice e contengono l’indirizzazione e i primi 8 byte del datagramma IP che ha provocato la generazione del messaggio, in modo che il mittente possa determinare il datagramma che ha causato l’errore. Alcuni tipi di messaggio ICMP sono mostrati nella Figura 5.19. Notiamo che i messaggi ICMP non vengono usati soltanto per segnalare condizioni d’errore.

Il noto programma *ping* invia un messaggio ICMP di tipo 8 e codice 0 verso l’host specificato. L’host destinazione, vedendo la richiesta di *echo*, risponde con un messaggio ICMP di tipo 0 e codice 0. La maggior parte delle implementazioni TCP/IP implementa un server ping direttamente nel sistema operativo; in altre parole, il server non è un processo. Il Capitolo 11 di [Stevens 1990] presenta il codice sorgente per un programma client ping. Notiamo che il programma client deve essere in grado di istruire il sistema operativo per generare un messaggio ICMP di tipo 8 e codice 0.

Un altro interessante messaggio ICMP è quello di riduzione del tasso di trasmissione. Lo scopo originario di questo messaggio, raramente usato, era un controllo di congestione che consentiva a un router congestionato di inviare un messaggio ICMP a un host, per farzelo a ridurre il proprio tasso di trasmissione. Abbiamo visto nel Capitolo 3 che TCP ha un meccanismo di controllo della congestione che opera a livello di trasporto e che i bit della notifica di congestione esplicita possono essere usati dai dispositivi del livello di rete per segnalare una congestione.

Nel corso del Capitolo 1 abbiamo introdotto traceroute, che consente di rilevare il percorso tra due host, utilizzando messaggi ICMP. Per determinare i

| Tipo ICMP | Codice | Descrizione                             |
|-----------|--------|-----------------------------------------|
| 0         | 0      | risposta echo (a ping)                  |
| 3         | 0      | rete destinazione irraggiungibile       |
| 3         | 1      | host destinazione irraggiungibile       |
| 3         | 2      | protocollo destinazione irraggiungibile |
| 3         | 3      | porta destinazione irraggiungibile      |
| 3         | 6      | rete destinazione sconosciuta           |
| 3         | 7      | host destinazione sconosciuto           |
| 4         | 0      | riduzione (controllo di congestione)    |
| 8         | 0      | richiesta echo                          |
| 9         | 0      | annuncio di un router                   |
| 10        | 0      | scoperta di un router                   |
| 11        | 0      | TTL scaduto                             |
| 12        | 0      | intestazione IP errata                  |

**Figura 5.19** Tipi di messaggio ICMP.

nomi e gli indirizzi dei router tra sorgente e destinazione, il programma invia una serie di datagrammi IP ordinari verso la destinazione, ciascuno dei quali trasporta un segmento UDP con un numero di porta improbabile, il primo con TTL pari a 1, il secondo pari a 2, il terzo pari a 3 e così via. Inoltre, la sorgente avvia un timer per ogni datagramma. Quando l' $n$ -esimo datagramma giunge all' $n$ -esimo router, quest'ultimo nota che TTL è appena scaduto. Secondo le regole del protocollo IP, il router lo scarta e invia alla sorgente un messaggio di errore ICMP (tipo 11 codice 0). Quando il messaggio giunge alla sorgente, questa ottiene il tempo di andata e ritorno a partire dal timer e dal nome e dall'indirizzo IP dell' $n$ -esimo router indicati da ICMP.

Come fa traceroute a sapere quando smettere di inviare segmenti UDP? Ricordiamo che la sorgente incrementa il campo TTL di ciascun datagramma inviato. Pertanto, uno di questi alla fine completerà il percorso verso l'host di destinazione. Dato che questo datagramma contiene un segmento UDP con un numero di porta improbabile, l'host di destinazione restituisce alla sorgente un messaggio ICMP di porta non raggiungibile (tipo 3 codice 3). L'host sorgente, quando riceve questo particolare messaggio ICMP, sa che non deve inviare ulteriori pacchetti. Il programma traceroute standard in effetti invia blocchi di tre pacchetti con identico TTL e di conseguenza offre tre risultati per ciascun TTL.

In questo modo, l'host sorgente apprende quanti e quali sono i router che lo separano dalla destinazione, nonché il tempo di andata e ritorno. Notiamo che il programma client traceroute deve essere in grado di istruire il sistema operativo per generare datagrammi UDP con valori specifici di TTL e deve anche essere in grado di recepire notifiche da parte del sistema operativo all'arrivo di messaggi ICMP.

Una nuova versione di ICMP è stata definita per Ipv6 [RFC 4443], in cui sono stati introdotti nuovi tipi e codici e ridefiniti quelli vecchi: per esempio, sono stati aggiunti i codici di errore “pacchetto troppo grande” e “opzioni non riconosciute”.

## 5.7 Gestione della rete e SNMP, NETCONF/YANG

A questo punto dovremmo essere ben consapevoli del fatto che una rete è costituita da molteplici e complessi componenti hardware e software che interagiscono tra loro: connessioni fisiche, switch, router, host e svariati dispositivi che costituiscono i componenti fisici della rete come pure molti protocolli che li controllano e li coordinano. Ovviamente, quando centinaia o migliaia di componenti vengono assemblati insieme per costruire una rete, il compito dell'amministratore di far funzionare la rete è sicuramente una sfida. Abbiamo visto nel Paragrafo 5.5 che il controller logicamente centralizzato può rappresentare un grande aiuto nel contesto SDN. Ma visto che l'amministratore di rete ha tale compito da prima della nascita di SDN, deve disporre di strumenti per il monitoraggio, la gestione e il controllo della rete, che studieremo in questo paragrafo insieme ad altri che si sono evoluti con SDN.

Una domanda che viene posta molto spesso è: “Che cosa vuol dire gestire una rete?”. La nostra discussione ne ha motivato la necessità e illustrato alcuni casi di utilizzo. Chiudiamo ora questo paragrafo con una definizione tratta da [Saydam 1996]:

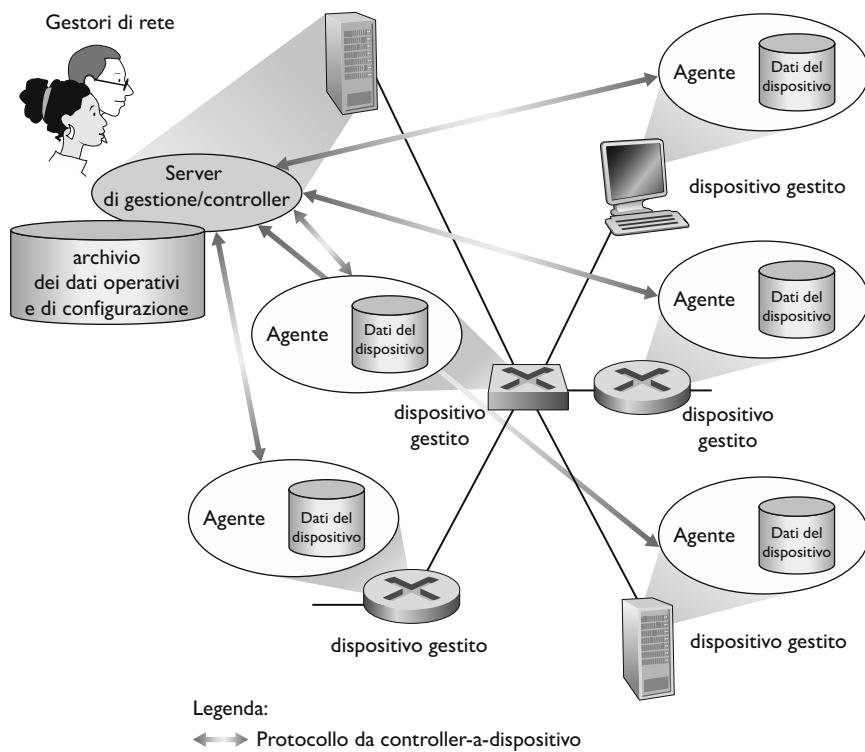
*“La gestione della rete comprende il funzionamento, l'integrazione e il coordinamento di hardware, software e personale tecnico per monitorare, verificare, configurare, analizzare, valutare e controllare le risorse della rete affinché soddisfino le funzionalità in tempo reale e i requisiti di qualità del servizio a un costo accettabile”.*

In questo capitolo esamineremo solo l'architettura, i protocolli e le informazioni di base utilizzati dai responsabili delle reti per svolgere i loro compiti. Non ci occuperemo dei processi di decisione come gli errori di identificazione [Labovitz 1997; Steinder 2002; Feamster 2005; Wu 2005; Teixeira 2006], la rilevazione di anomalie [Lakhina 2005; Barford 2009], la progettazione e l'ingegnerizzazione della rete in modo da adempiere al Service Level Agreements (SLA's) [Huston 1999a], e altro ancora. Il lettore interessato può consultare anche l'eccellente testo di [Subramanian 2000, Schonwalder 2010; Claise 2019] e trovare sul sito del libro ulteriori informazioni.

### 5.7.1 Infrastruttura di gestione

Nella Figura 5.20 sono mostrate le componenti chiave della gestione di rete:

- Il **server di gestione** è un'applicazione controllata da gestori di rete (*network managers*) ed eseguita da un calcolatore del NOC (*Network Operations Server*). Il suo compito è quello di sovraintendere alla raccolta, elaborazione, analisi e invio delle informazioni e dei comandi di gestione. È qui che vengono avviate le azioni per configurare, monitorare e controllare i dispositivi della rete. Nella pratica, una rete può avere più server di gestione di questo tipo.
- Un **dispositivo di rete gestito** è un componente hardware o software della rete (che nella nostra analogia equivale a una filiale), come un host, un router, un bridge, uno hub, una stampante o un modem. Il dispositivo stesso ha componenti da gestire (per esempio, una interfaccia di rete non è altro che



**Figura 5.20** Elementi della gestione di rete.

una componente di un host o di un router) e un insieme di parametri di configurazione per hardware e software (per esempio, un protocollo di instradamento intra-dominio come OSPF).

- **Dati.** A ogni oggetto da gestire sono associati dei dati, chiamati “stati”, di diverso tipo. I **dati di configurazione** sono informazioni sul dispositivo configurati esplicitamente dal gestore della rete quali, per esempio, un indirizzo IP configurato/assegnato dal gestore o la velocità per una interfaccia del dispositivo. I **dati operativi** sono informazioni che il dispositivo acquisisce mentre opera quali, per esempio, l’elenco dei vicini nel protocollo OSPF. Le **statistiche del dispositivo** sono indicatori di stato e contatori che vengono aggiornati come operatori del dispositivo (per esempio, il numero di pacchetti scartati su un’interfaccia o la velocità della ventola di raffreddamento del dispositivo). Il gestore della rete può interrogare i dati del dispositivo remoto e, in alcuni casi, controllarlo scrivendo il valore dei dati, come discusso di seguito. Come mostrato nella Figura 5.17, anche il server di gestione conserva la propria copia dei dati di configurazione, operativi e statistici dei dispositivi che gestisce, così come i dati a livello di rete (per esempio, la topologia della rete).
- L'**agente di gestione** è un processo software che comunica con il server di gestione ed esegue azioni locali sul dispositivo che gestisce sotto il comando e controllo del server di gestione, in modo simile all’agente di instradamento mostrato nella Figura 5.2.
- Il **protocollo di gestione di rete**. Questo protocollo sta tra il server di gestione e i dispositivi gestiti, consentendo al server di gestione di richiedere lo stato

dei dispositivi e agire su di essi attraverso gli agenti. Questi, a loro volta, utilizzano il protocollo di gestione per informare il server di gestione di eventuali anomalie, come guasti nei componenti o prestazioni sotto una soglia minima. A questo riguardo occorre fare una sottile, ma importante distinzione: il protocollo di gestione non gestisce esso stesso la rete, ma è soltanto uno strumento tramite il quale l'amministratore opera.

In pratica, ci sono tre modi comunemente usati da un operatore di rete per gestire la rete, utilizzando i componenti sopra descritti:

- **CLI.** Un operatore di rete può inviare direttamente **comandi CLI** (*Command Line Interface*) al dispositivo. Questi comandi possono essere digitati direttamente su una console del dispositivo, se l'operatore è fisicamente presente sul dispositivo, o su una connessione Telnet o Secure Shell (SSH), tramite script, tra il server/controller di gestione e il dispositivo. I comandi CLI sono specifici del fornitore e del dispositivo e possono essere piuttosto arcani. Mentre i maghi della rete potrebbero essere in grado di utilizzare la CLI per configurare in modo impeccabile i dispositivi di rete, l'uso della CLI è incline agli errori ed è difficile da automatizzare o scalare in modo efficiente per reti di grandi dimensioni. I dispositivi di uso comune, come il router domestico wireless, possono esportare un menu di gestione a cui l'utente (gestore di rete!) può accedere tramite HTTP per configurare il dispositivo. Sebbene questo approccio possa funzionare bene per dispositivi singoli e semplici e sia meno soggetto a errori rispetto alle interfacce CLI, non scala a reti di dimensioni maggiori.
- **SNMP/MIB.** In questo approccio l'operatore di rete può interrogare/impostare i dati contenuti negli **oggetti MIB** (*Management Information Base*, base di dati gestionale) di un dispositivo utilizzando il protocollo SNMP (*Simple Network Management Protocol*). Alcuni MIB sono specifici del fornitore e del dispositivo, mentre altri, quali il numero di datagrammi IP scartati in un router a causa di errori nell'intestazione o il numero di segmenti UDP ricevuti in un host, sono indipendenti dal dispositivo, fornendo così astrazione e generalità. Un operatore di rete userebbe più tipicamente questo approccio per interrogare e monitorare lo stato delle operazioni e le statistiche del dispositivo, e userebbe le CLI per controllare/configurare attivamente il dispositivo. È importante notare che entrambi gli approcci gestiscono i dispositivi individualmente. Tratteremo SNMP e MIB, che sono in uso dalla fine degli anni '80, nel Paragrafo 5.7.2, di seguito. Un seminario sulla gestione della rete organizzato dall'Internet Architecture Board nel 2002 [RFC 3535] ha sottolineato non solo il valore dell'approccio SNMP/MIB per il monitoraggio dei dispositivi, ma anche i suoi difetti, relativi soprattutto alla configurazione dei dispositivi e la gestione delle reti su larga scala, dando origine all'approccio più recente per la gestione della rete che utilizza NETCONF e YANG.
- **NETCONF/YANG.** L'approccio NETCONF/YANG richiede una visione più astratta, globale e olistica della rete con una enfasi maggiore sulla gestione della configurazione, tra cui la specifica dei vincoli di correttezza e la fornitura di operazioni di gestione fini su più dispositivi. **YANG** [RFC 6020] è un linguaggio di modellazione dei dati utilizzato per modellare dati di configurazione e operativi. Il protocollo **NETCONF** [RFC 6241] viene utilizzato per comunicare azioni e dati compatibili con YANG verso/da/tra dispositivi

remoti. Abbiamo già visto brevemente NETCONF e YANG nel nostro caso di studio dell'OpenDaylight Controller nella Figura 5.17; li tratteremo nel Paragrafo 5.7.3, di seguito.

### 5.7.2 SNMP (*Simple Network Management Protocol*) e MIB (*Management Information Base*)

**SNMPv3** è un protocollo di livello applicazione che trasporta informazioni fra server di gestione e agenti [RFC 3410]. La più comune modalità di utilizzo di SNMP è detta **richiesta-risposta**: un'entità di gestione SNMP invia una richiesta a un agente SNMP che, a seguito di questa, compie azioni e invia una risposta. Di solito queste istanze sono utilizzate per la lettura/scrittura di un parametro di un oggetto MIB associato a un dispositivo. Un altro tipico utilizzo di SNMP si verifica quando un agente, senza che gli sia pervenuta alcuna richiesta, invia un messaggio non sollecitato, detto **messaggio trap**, al server di gestione, in cui segnala il verificarsi di una situazione eccezionale che ha prodotto una variazione dei valori degli oggetti MIB.

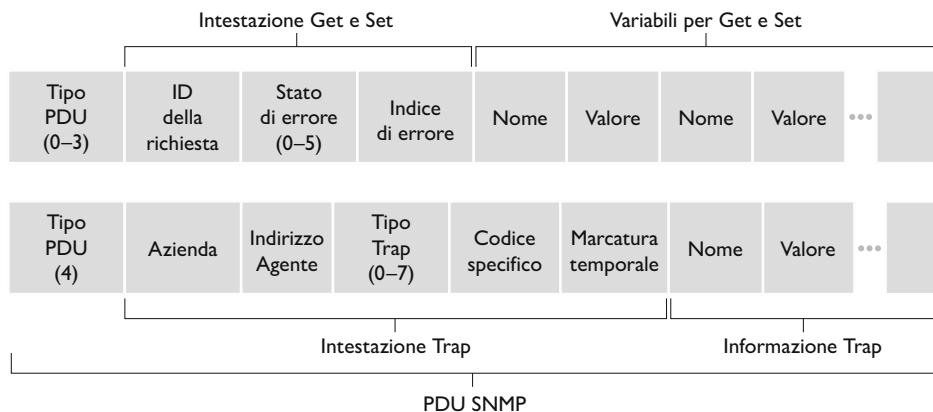
Gli oggetti MIB sono specificati in un linguaggio di descrizione dei dati noto come SMI (*Structure of Management Information*) [RFC 2578; RFC 2579; RFC 2580], un nome piuttosto strano che non contiene alcuna indicazione sulla sua funzionalità. Un linguaggio di definizione formale viene utilizzato per garantire che la sintassi e la semantica dei dati di gestione della rete siano ben definiti e inequivocabili. Gli oggetti MIB vengono raccolti in moduli MIB. Alla fine del 2019 esistono più di 400 RFC legate a MIB e un numero molto maggiore di moduli MIB (privati) specifici del fornitore.

SNMPv3 definisce sette tipi di messaggi, genericamente detti PDU (*Protocol Data Unit*, unità dati di protocollo) e riportati nella Tabella 5.2. Il formato delle PDU è mostrato nella Figura 5.21.

**Tabella 5.2** Tipi di PDU in SNMPv3.

| Tipo di PDU in SNMPv3 | Mittente-ricevente                   | Descrizione                                                                                                         |
|-----------------------|--------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| GetRequest            | manager – agente                     | Recupera il valore di una o più istanze di oggetto MIB                                                              |
| GetNextRequest        | manager – agente                     | Recupera il valore della prossima istanza di oggetto MIB (in una lista o in una tabella)                            |
| GetBulkRequest        | manager – agente                     | Recupera il valore di grandi blocchi di dati, per esempio valori in una grossa tabella                              |
| InformRequest         | manager – manager                    | Informa un'entità di gestione remota su dei valori MIB (che potrebbero non essere locali)                           |
| SetRequest            | manager – agente                     | Imposta il valore di una o più istanze di oggetti MIB                                                               |
| Response              | agente – manager o manager – manager | Generato in risposta a<br>GetRequest<br>GetNextRequest<br>GetBulkRequest<br>SetRequest PDU, oppure<br>InformRequest |
| SNMPv3-Trap           | agente – manager                     | Informa il manager di un evento inatteso                                                                            |

**Figura 5.21** Formato delle PDU SNMP.



- Le tre PDU GetRequest, GetNextRequest e GetBulkRequest sono inviate da un’entità di gestione all’agente del dispositivo da gestire per richiedere uno o più valori di oggetti MIB. Gli identificativi degli oggetti MIB i cui valori sono stati richiesti sono specificati nella porzione variabile obbligatoria della PDU. Queste tre PDU differiscono per la granularità dei dati richiesti. GetRequest può richiedere qualunque valore MIB, mentre per elenchi o tavole di oggetti si può utilizzare una sequenza di GetNextRequest. GetBulkRequest restituisce grandi gruppi di dati, evitando le ridondanze di una serie di GetRequest o di GetNextRequest. In tutti e tre i casi, l’agente risponde con una PDU Response, che contiene gli identificatori simbolici dell’oggetto e i rispettivi valori.
- Con la PDU SetRequest un server di gestione può impostare il valore di uno o più oggetti MIB in un dispositivo. Il server ricevente risponde con una PDU Response contenente “noError” in caso di effettiva modifica del parametro.
- La PDU InformRequest è utilizzata da un server di gestione per trasmettere informazioni MIB a un’altra entità di gestione. Quest’ultima risponde con una PDU Response con stato di errore impostato a “noError” a conferma della ricezione della PDU InformRequest.
- La PDU Response viene inviata dal dispositivo al server di gestione per restituire l’informazione richiesta.
- Il messaggio trap è un evento asincrono, cioè *non* è la risposta a una richiesta, ma viene emesso quando si verifica uno degli eventi di cui l’entità di gestione aveva richiesto la notifica. L’RFC 3418 definisce vari tipi di trap tra cui quelle relative all’avvio a freddo (senza procedura di reboot, in conseguenza a un malfunzionamento o un calo di tensione) o a caldo (con procedura di reboot, gestito dall’amministratore) di un dispositivo, la disponibilità o meno di un collegamento, la perdita di raggiungibilità di un vicino o una mancata autenticazione. Il server di gestione non è tenuto a fornire un riscontro ai messaggi trap ricevuti.

Vista la struttura a “domanda e risposta” di SNMPv3, va detto che, sebbene le PDU possano essere convogliate da qualunque protocollo di trasporto, di solito costituiscono il payload di un datagramma UDP. Infatti, l’RFC 3417 afferma che UDP è “il mezzo di trasporto preferito”. Dato che UDP è un protocollo di

trasporto inaffidabile, non c'è garanzia che una richiesta, o la sua risposta, raggiunga il destinatario. Il campo identificativo della richiesta (Request ID) delle PDU (Figura 5.21) è utilizzato dall'entità di gestione per numerare le sue richieste e dall'agente per la risposta. Quindi si può utilizzare il campo Request ID per rilevare la perdita di richieste o di risposte. È compito dell'entità di gestione decidere se ritrasmettere una richiesta non riscontrata entro un certo tempo. SNMP non prevede particolari procedure di ritrasmissione (e non specifica se deve esserci ritrasmissione), ma richiede solamente che l'entità di gestione "sia responsabile di frequenza e durata delle ritrasmissioni". Questo, naturalmente, spinge a chiederci quale sia il comportamento di un protocollo "responsabile".

SNMP si è evoluto in tre versioni. I suoi progettisti hanno dichiarato che "SNMPv3 può essere visto come SNMPv2 con aggiunte caratteristiche di sicurezza e amministrazione" [RFC 3410]. Infatti, nessuno dei cambiamenti di SNMPv3 rispetto a SNMPv2 è così evidente come quelli relativi ad amministrazione e sicurezza. La necessità di queste modifiche è dovuta al fatto che le versioni precedenti di SNMP venivano usate soprattutto per il monitoraggio piuttosto che per il controllo (per esempio, in SNMPv1, SetRequest è utilizzata raramente).

### **MIB (*Management Information Base*)**

Abbiamo appreso che nell'approccio SNMP/MIB alla gestione della rete i dati sullo stato operativo di un dispositivo gestito (e in una certa misura i suoi dati di configurazione) sono rappresentati come oggetti che vengono raccolti insieme in un MIB per quel dispositivo. Un oggetto MIB potrebbe essere un contatore, come il numero di datagrammi IP scartati su un router a causa di errori nell'intestazione di un datagramma, informazioni descrittive come la versione del software in esecuzione su un server DNS, informazioni sullo stato, quale per esempio se un particolare dispositivo funziona correttamente, o informazioni specifiche del protocollo, come un percorso di instradamento verso una destinazione. Gli oggetti MIB vengono raccolti in moduli MIB. Ci sono oltre 400 moduli MIB definiti in varie RFC IETC e ci sono molti altri MIB specifici dei fornitori. [RFC 4293] specifica il modulo MIB che definisce gli oggetti gestiti, tra cui ipSystemStatsInDelivers, per la gestione del protocollo Internet IP (Internet Protocol) e il relativo protocollo ICMP (Internet Control Message Protocol). [RFC 4022] specifica il modulo MIB per TCP e [RFC 4113] specifica il modulo MIB per UDP.

Mentre le RFC relative a MIB sono una lettura piuttosto noiosa, è invece istruttivo prendere in considerazione un esempio di oggetto MIB. La definizione del tipo di oggetto ipSystem-StatsInDelivers nella [RFC 4293] definisce un contatore di sola lettura a 32 bit che tiene traccia del numero di datagrammi IP ricevuti sul dispositivo gestito e consegnati a un protocollo di livello superiore. Nell'esempio seguente, Counter32 è uno dei tipi di dati di base definiti in SMI.

```
ipSystemStatsInDelivers OBJECT-TYPE
  SYNTAX Counter32
  MAX-ACCESS read-only
  STATUS current
  DESCRIPTION
```

“The total number of datagrams successfully delivered to IPUser-protocols (including ICMP).

When tracking interface statistics, the counter of the interface to which these datagrams were addressed is incremented. This interface might not be the same as the input interface for some of the datagrams.

Discontinuities in the value of this counter can occur at re-initialization of the management system, and at other times as indicated by the value of ipSystemStatsDiscontinuityTime.”

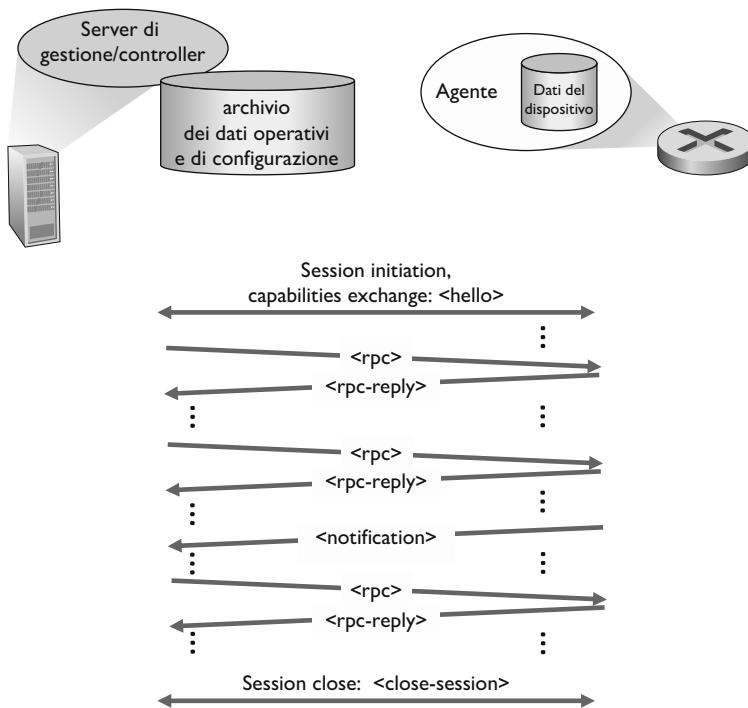
```
::= { ipSystemStatsEntry 18 }
```

### 5.7.3 NETCONF e YANG

Il protocollo NETCONF opera tra il server di gestione e i dispositivi di rete, fornendo messaggi per (i) recuperare, impostare e modificare i dati di configurazione; (ii) interrogare i dati operativi e le statistiche; e (iii) iscriversi alle notifiche generate dai dispositivi gestiti. Il server di gestione controlla attivamente un dispositivo gestito sia inviandogli le configurazioni, che sono specificate in un documento XML strutturato, sia attivando una configurazione. NETCONF utilizza un paradigma di procedura di chiamata remota (RPC, *Remote Procedure Call*), in cui anche i messaggi di protocollo sono codificati in XML e scambiati tra il server di gestione e un dispositivo gestito su una sessione orientata alla connessione sicura come il protocollo TLS (*Transport Layer Security*), discusso nel Capitolo 8, disponibile in inglese sulla piattaforma MyLab, su TCP.

La Figura 5.22 mostra un esempio di sessione NETCONF. Innanzitutto, il server di gestione stabilisce una connessione sicura al dispositivo gestito. Nel gergo NETCONF, il server di gestione viene indicato come “client” e il dispositivo gestito come “server”, in quanto è il server di gestione a stabilire la connessione verso il dispositivo, ma qui ignoreremo tale terminologia per coerenza con la usuale terminologia server/client mostrata nella Figura 5.20. Una volta stabilita una connessione sicura, il server di gestione e il dispositivo scambiano messaggi <hello>, dichiarando le loro “capabilities”: una funzionalità NETCONF che integra la specifica NETCONF di base in [RFC 6241]. Le interazioni tra il server di gestione e il dispositivo gestito assumono la forma di una chiamata di procedura remota, utilizzando i messaggi <rpc> e <rpc-response>. Questi messaggi vengono utilizzati per recuperare, impostare, interrogare e modificare le configurazioni, i dati operativi e le statistiche e iscriversi alle notifiche del dispositivo. Le notifiche del dispositivo vengono inviate in modo proattivo dal dispositivo gestito al server di gestione utilizzando i messaggi NETCONF <notification>. Una sessione viene chiusa con il messaggio <close-session>.

La Tabella 5.3 mostra alcune importanti operazioni NETCONF. Come nel caso di SNMP, ci sono operazioni per il recupero dei dati operativi (<get>) e per la notifica di eventi. Tuttavia, le operazioni <get-config>, <edit-config>, <lock> e <unlock> dimostrano la particolare enfasi di NETCONF sulla config-



**Figura 5.22** Una sessione NETCONF tra il server/controller di gestione e il dispositivo gestito.

gurazione del dispositivo. Utilizzando le operazioni di base mostrate nella Tabella 5.3 è anche possibile creare un insieme di transazioni di gestione di rete più sofisticate che operano su un insieme di parametri e di dispositivi. Tali transazioni multi-dispositivo, permettendo agli operatori di concentrarsi sulla configurazione della rete nel suo complesso piuttosto che su un singolo dispositivo, sono state un importante requisito presentato nella [RFC 3535].

**Tabella 5.3** Una selezione di operazioni NETCONF.

| Operazione NETCONF                    | Descrizione                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <get-config>                          | Recupera tutta o parte di una data configurazione. Un dispositivo può avere più configurazioni. C'è sempre una configurazione in esecuzione, <code>running</code> , che descrive la configurazione corrente (in esecuzione) dei dispositivi.                                                                                                                                                                                                                                                                                                             |
| <get>                                 | Recupera tutti o parte dei dati di configurazione e operativi.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <edit-config>                         | Modifica in tutto o in parte una configurazione specificata sul dispositivo gestito. Se viene specificata la configurazione <code>running</code> , quindi l'attuale configurazione in esecuzione sul dispositivo, tale configurazione verrà modificata. Se il dispositivo gestito è stato in grado di soddisfare la richiesta, viene inviato un <rpc-reply> contenente un elemento <ok>; altrimenti viene restituita la risposta <rpcerror>. In caso di errore, lo stato di configurazione del dispositivo può essere riportato al suo stato precedente. |
| <lock>, <unlock>                      | L'operazione <lock> (<unlock>) consente al server di gestione di bloccare (sbloccare) l'intero sistema di archivio dei dati di configurazione di un dispositivo gestito. I blocchi hanno una durata breve e consentono a un client di apportare una modifica senza timore di interazione con altri comandi NETCONF, SNMP o CLI da altre fonti.                                                                                                                                                                                                           |
| <create-subscription>, <notification> | Questa operazione avvia una sottoscrizione di notifica degli eventi che invierà una <notification> di evento asincrona per eventi di interesse specificati dal dispositivo al server di gestione, fino al termine della sottoscrizione.                                                                                                                                                                                                                                                                                                                  |

Una descrizione completa di NETCONF va oltre il nostro scopo; [RFC 6241, RFC5277, Claise 2019; Schonwalder 2010] forniscono una trattazione più approfondita.

Poiché tuttavia questa è la prima volta che vediamo messaggi di protocollo formattati come documenti XML, piuttosto che come messaggi tradizionali con campi di intestazione e corpo del messaggio come, per esempio, quello mostrato nella Figura 5.21 per PDU SNMP, concludiamo il nostro breve studio di NETCONF con due esempi.

Nel primo esempio, il documento XML inviato dal server di gestione al dispositivo gestito è un comando NETCONF <get> che richiede i dati di configurazione e operativi. Con questo comando il server apprende la configurazione del dispositivo.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <rpc message-id="101"
03   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
04   <get/>
05 </rpc>
```

Sebbene siano in pochi a essere in grado di analizzare direttamente un documento XML, possiamo vedere come NETCONF sia relativamente leggibile dall'uomo e ricordi molto più HTTP e HTML rispetto ai formati dei messaggi PDU SNMP della Figura 5.21. Il messaggio RPC è nelle righe 02–05 (abbiamo aggiunto i numeri di riga qui a scopo didattico). L'RPC ha un valore ID di messaggio di 101, dichiarato nella riga 02 e contiene un singolo comando NETCONF <get>. La risposta dal dispositivo contiene un numero ID corrispondente (101), tutti i dati di configurazione del dispositivo (in formato XML, ovviamente), a partire dalla riga 04 e infine la chiusura </rpc-reply>.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <rpc-reply message-id="101"
03   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
04   <!-- . . . all configuration data returned... -->
. .
</rpc-reply>
```

Nel secondo esempio di seguito, adattato da [RFC 6241], il documento XML inviato dal server di gestione al dispositivo gestito imposta l'unità di trasmissione massima (MTU) di un'interfaccia denominata “Ethernet0/0” a 1500 byte:

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <rpc message-id="101"
03   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
04   <edit-config>
05     <target>
06       <running/>
07     </target>
08     <config>
09       <top xmlns="http://example.com/schema/
1.2/config">
10         <interface>
```

```

11      <name>Ethernet0/0</name>
12      <mtu>1500</mtu>
13    </interface>
14  </top>
15 </config>
16 </edit-config>
17 </rpc>

```

Il messaggio RPC si estende sulle righe 02-17, ha un valore ID di messaggio di 101 e contiene un singolo comando NETCONF `<edit-config>`, che copre le righe 04-15. La linea 06 indica che la configurazione in esecuzione sul dispositivo gestito sarà cambiata. Le righe 11 e 12 specificano la dimensione MTU da impostare dell'interfaccia Ethernet0/0.

Il dispositivo, dopo aver modificato la dimensione MTU dell'interfaccia nella configurazione, risponde al server di gestione con una risposta OK (riga 04 qui di seguito), di nuovo all'interno di un documento XML:

```

01 <?xml version="1.0" encoding="UTF-8"?>
02 <rpc-reply message-id="101"
03   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
04     <ok/>
05   </rpc-reply>

```

## YANG

YANG è il linguaggio di modellazione dei dati utilizzato per specificare con precisione la struttura, la sintassi e la semantica dei dati di gestione della rete usati da NETCONF, più o meno allo stesso modo in cui SMI viene utilizzato per specificare i MIB in SNMP. Tutte le definizioni YANG sono contenute nei moduli, e un documento XML che descrive un dispositivo e le sue capacità può essere generato da un modulo YANG.

YANG presenta un piccolo insieme di tipi di dati incorporati (come nel caso di SMI) e consente a chi modella i dati di esprimere vincoli che devono essere soddisfatti da un configuratore NETCONF valido: un aiuto efficace per garantire che le configurazioni NETCONF soddisfino i vincoli di correttezza e coerenza specificati. Anche YANG specifica le notifiche NETCONF.

Una discussione più completa di YANG va oltre il nostro scopo. Per maggiori informazioni rimandiamo il lettore interessato all'ottimo libro [Claise 2019].

## 5.8 Riepilogo

In questi due capitoli abbiamo completato il nostro viaggio attraverso il livello di rete iniziato con la trattazione del piano dei dati nel Capitolo 4 e terminato esaminando il piano di controllo. Quest'ultimo è la logica di rete globale che controlla non solo come i datagrammi vengono inoltrati tra i router lungo i percorsi end-to-end dall'host sorgente all'host destinatario, ma anche come le componenti e i servizi del livello di rete vengono configurati e gestiti.

Abbiamo imparato che esistono due approcci per la costruzione di un piano di controllo: il controllo tradizionale per router (nel quale ogni router esegue un algoritmo di instradamento e comunica con tutti gli altri) e il *software-defined*

*networking* (SDN) (nel quale un controller logicamente centralizzato calcola e distribuisce le tabelle di inoltro che devono essere utilizzate da ogni singolo router). Abbiamo studiato nel Paragrafo 5.2 due algoritmi di routing fondamentali per il calcolo dei percorsi a costo minimo in un grafo: link-state e distance-vector, che trovano applicazione sia nell’approccio tradizionale che in quello SDN. Questi algoritmi sono alla base di due protocolli ampiamente diffusi in Internet: OSPF e BGP, trattati nei Paragrafi 5.3 e 5.4. Abbiamo trattato l’approccio SDN al piano di controllo nel Paragrafo 5.5, discutendone le applicazioni, il controller e il protocollo OpenFlow per la comunicazione tra controller e dispositivi controllati. Nei Paragrafi 5.6 e 5.7 abbiamo discusso i fondamenti della gestione delle reti IP: i protocolli ICMP e la gestione di rete usando SNMP e NET-CONF/YANG.

Avendo ora completato lo studio del livello di rete, possiamo scendere nello stack dei protocolli per trattare il livello di collegamento che, come quello di rete, è presente in tutti i dispositivi in rete. Tuttavia, nel prossimo capitolo vedremo che tale livello ha il compito molto più localizzato di spostare i pacchetti tra nodi sullo stesso collegamento o LAN. Benché tale compito appaia semplice, vedremo che in realtà ha a che fare con problematiche complesse e affascinanti che ci terranno impegnati a lungo.

## Domande e problemi

### Domande di revisione

#### PARAGRAFO 5.1

- R1.** Che cosa si intende per piano di controllo basato sul controllo per router? In questi casi, quando diciamo che il piano di controllo della rete e il piano dei dati sono implementati “monoliticamente,” che cosa intendiamo?
- R2.** Che cosa si intende per piano di controllo basato sul controllo logicamente centralizzato? In questi casi, il piano dei dati e il piano di controllo sono implementati all’interno dello stesso dispositivo o in dispositivi separati? Perché?

#### PARAGRAFO 5.2

- R3.** Confrontate le proprietà degli algoritmi di instradamento centralizzati e distribuiti e fornite esempi di protocolli che li utilizzano.
- R4.** Confrontate gli algoritmi di routing link-state e distance-vector.
- R5.** Qual è il problema del “conteggio all’infinito” negli algoritmi distance-vector?
- R6.** È necessario che ogni sistema autonomo utilizzi lo stesso algoritmo di instradamento intra-AS? Perché o perché no?

#### PARAGRAFI 5.3-5.4

- R7.** Perché i protocolli inter-AS e intra-AS utilizzati in Internet sono diversi?
- R8.** Vero o falso: quando un percorso OSPF invia le sue informazioni sullo stato dei collegamenti, le invia solo ai nodi direttamente a esso collegati. Motivate la risposta.
- R9.** Che cosa si intende per “area” in un sistema autonomo OSPF? Perché tale concetto è stato introdotto?
- R10.** Definite e paragonate i seguenti termini: *sottorete*, *prefisso* e *rotta BGP*.
- R11.** Come usa BGP gli attributi NEXT-HOP e AS-PATH?
- R12.** Descrivete come l’amministratore di rete di un ISP di livello superiore possa attuare una policy quando configura BGP.
- R13.** Vero o falso: un router BGP, quando riceve l’annuncio di un percorso da un suo vicino, deve aggiungere la propria identità e quindi inviare tale nuovo percorso a tutti i suoi vicini. Motivate la risposta.

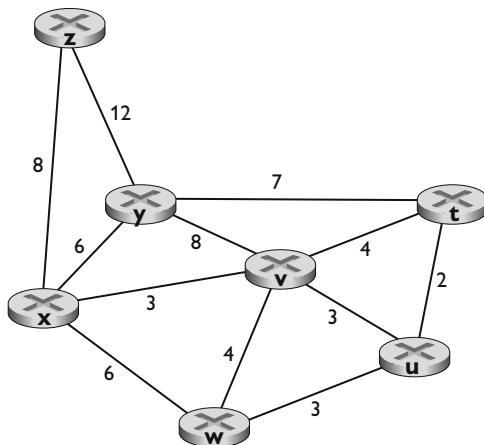
#### PARAGRAFO 5.5

- R14.** Descrivete il ruolo principale del livello di comunicazione, del livello di gestione dello stato globale di rete e del livello di applicazione della rete in un controller SDN.

- R15.** Supponete di voler implementare un nuovo protocollo di routing nel piano di controllo SDN. A quale livello si dovrebbe implementare tale protocollo? Motivate la risposta.
- R16.** Quali tipi di messaggi fluiscono attraverso le API northbound e southbound di un controller SDN? Chi è il destinatario di questi messaggi inviati dal controller attraverso l'interfaccia southbound, e chi invia messaggi al controller attraverso l'interfaccia northbound?
- R17.** Descrivete lo scopo di due tipi di messaggi OpenFlow (di vostra scelta) che vengono inviati da un dispositivo controllato al controller. Descrivete lo scopo dei due tipi di messaggi OpenFlow (di vostra scelta) che vengono inviati dai controller di un dispositivo controllato.
- R18.** Qual è lo scopo del livello di astrazione di servizio nel controller OpenDaylight SDN?

## Problemi

- P1.** Considerando la Figura 5.3, enumerate i percorsi da  $y$  a  $u$  che non contengono cicli.
- P2.** Ripetete il Problema P1 per i percorsi da  $x$  a  $z$ , da  $z$  a  $u$  e da  $z$  a  $w$ .
- P3.** Considerate la rete della figura illustrata di seguito. Con i costi per collegamento indicati, utilizzate l'algoritmo di Dijkstra per calcolare il percorso più breve da  $x$  a tutti i nodi della rete. Mostrate il funzionamento dell'algoritmo calcolando una tabella simile alla Tabella 5.1.

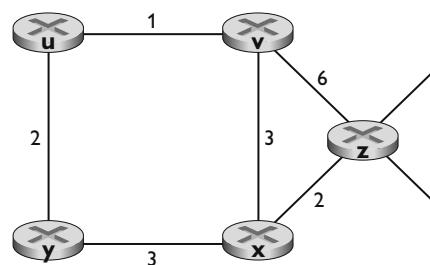


- P4.** Considerate la rete del Problema P3. Usando l'algoritmo di Dijkstra e mostrando il vostro lavoro con una tabella simile alla Tabella 5.1, calcolate il percorso più breve:
- da  $t$  a tutti i nodi della rete;
  - da  $u$  a tutti i nodi della rete;
  - da  $v$  a tutti i nodi della rete;
  - da  $w$  a tutti i nodi della rete;
  - da  $y$  a tutti i nodi della rete;
  - da  $z$  a tutti i nodi della rete.

## PARAGRAFI 5.6-5.7

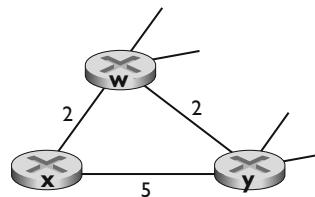
- R19.** Nominate quattro diversi tipi di messaggi ICMP.
- R20.** Quali due tipi di messaggi ICMP vengono ricevuti dall'host sorgente avente in esecuzione il programma Traceroute?
- R21.** Definite i seguenti termini nel contesto di SNMP: server di gestione, dispositivo gestito, agente di gestione e MIB.
- R22.** Qual è lo scopo dei messaggi SNMP GetRequest e SetRequest?
- R22.** Qual è lo scopo del messaggio trap SNMP?

- P5.** Esamine la rete mostrata di seguito e assumete che ciascun nodo inizialmente conosca il costo dei suoi archi incidenti. Considerate l'algoritmo distance-vector e mostrate le righe della tabella delle distanze presso il nodo  $z$ .



- P6.** Considerate una topologia generica (ossia, non la rete specifica mostrata in precedenza) e una versione sincrona dell'algoritmo distance-vector. Supponete che a ciascuna iterazione, un nodo scambi il proprio vettore delle distanze con i suoi vicini e riceva i loro. Assumendo che l'algoritmo cominci con ciascun nodo che conosce solo i costi dei suoi immediati vicini, qual è il massimo numero di iterazioni richieste prima che l'algoritmo distribuito converga? Giustificate la vostra risposta.

- P7.** Considerate il frammento di rete mostrato di seguito. Il nodo  $x$  è collegato solamente a  $w$  e  $y$ . Il primo ha un percorso minima verso la destinazione  $u$  (non mostrata) di costo 5 e  $y$  ha un percorso minima verso  $u$  di costo 6. I percorsi completi da  $w$  e  $y$  a  $u$  (e tra  $w$  e  $y$ ) non sono rappresentati. Tutti i costi dei collegamenti nella rete hanno valori interi positivi.



- (a) Fornite il vettore delle distanze di  $x$  per le destinazioni  $w, y$  e  $u$ .
- (b) Fornite una variazione nel costo dei collegamenti o per  $c(x,w)$  o per  $c(x,y)$  tale che  $x$  informi i suoi vicini di un nuovo percorso a costo minimo verso  $u$  come risultato dell'esecuzione dell'algoritmo distance-vector.
- (c) Fornite una variazione nel costo dei collegamenti o per  $c(x,w)$  o per  $c(x,y)$  tale che  $x$  non informi i suoi vicini di un nuovo percorso a costo minimo verso  $u$  come risultato dell'esecuzione dell'algoritmo distance-vector.

**P8.** Considerate il grafo con tre nodi della Figura 5.6. Anziché i costi indicati nella figura, supponete ora che i costi siano:  $c(x,y) = 3, c(y,z) = 6, c(z,x) = 4$ . Calcolate le tabelle delle distanze dopo l'inizializzazione e dopo ogni iterazione di una versione sincrona dell'algoritmo distance-vector (come abbiamo precedentemente fatto per la Figura 5.6).

**P9.** Considerate il problema del conteggio all'infinito nell'instradamento con distance-vector. Si verifica anche se diminuiamo il costo dei collegamenti? Perché? Che cosa succede se collegiamo due nodi prima non collegati?

**P10.** Discutete il fatto che nell'instradamento con distance-vector della Figura 5.6 tutti i valori del vettore delle distanze  $D(x)$  sono non crescenti e infine si stabilizzano in un numero finito di passi.

**P11.** Considerate la Figura 5.7. Supponete che un altro router  $w$  sia connesso ai router  $y$  e  $z$ . I costi dei collegamenti sono:  $c(x,y) = 4, c(x,z) = 50, c(y,w) = 1, c(z,w) = 1, c(y,z) = 3$ . Supponete di usare l'inversione avvelenata nell'algoritmo di instradamento distance-vector.

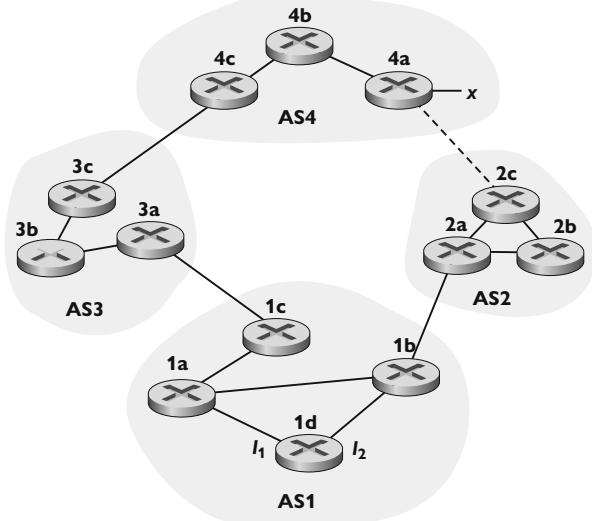
- (a) Quando l'algoritmo è stabilito, i router  $x, y$  e  $z$  si scambiano le loro distanze da  $x$ . Qual è il valore che riportano?
- (b) Supponete ora che il costo del collegamento tra  $x$  e  $y$  aumenti a 60. Si verificherà il problema del conteggio all'infinito anche usando l'inversione avvelenata? Spiegatene il motivo. Se esiste un problema del conteggio all'infinito, quante iterazioni sono necessarie perché l'instradamento distance-vector raggiunga di nuovo uno stato stazionario? Giustificate la risposta.
- (c) Come modifichereste  $c(y,z)$  in modo da non avere il problema del conteggio all'infinito se  $c(y,x)$  aumenta da 4 a 60?

**P12.** Descrivete come BGP possa rilevare percorsi ciclici.

**P13.** I router BGP scelgono sempre i percorsi privi di cicli con il percorso interno all'AS minimo? Giustificate la risposta.

**P14.** Considerate la rete riportata in seguito. Supponete che (1) AS3 e AS2 abbiano in esecuzione OSPF come protocollo di instradamento intra-AS, mentre AS1 e AS4 usino RIP, (2) eBGP e iBGP siano usati come protocolli di instradamento inter-AS. Inizialmente supponete che non vi siano collegamenti fisici tra AS2 e AS4.

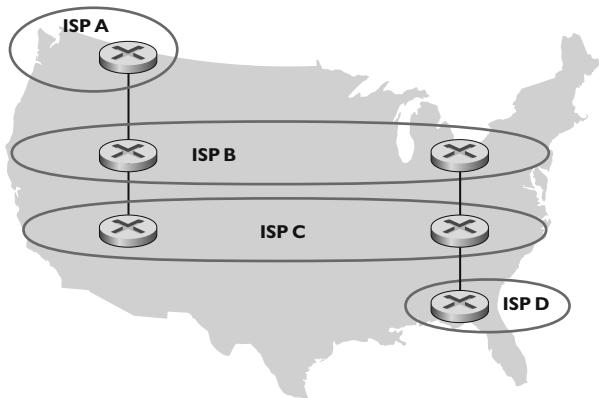
- (a) Da quale protocollo di instradamento, RIP, OSPF, eBGP o iBGP, il router 3c apprende il prefisso  $x$ ?
- (b) Da quale protocollo di instradamento il router 3a apprende il prefisso  $x$ ?
- (c) Da quale protocollo di instradamento il router 1c apprende il prefisso  $x$ ?
- (d) Da quale protocollo di instradamento il router 1d apprende il prefisso  $x$ ?



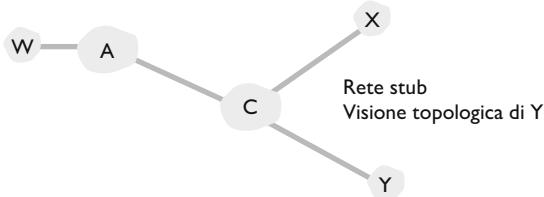
**P15.** Facendo riferimento al problema precedente, una volta che il router 1d apprende di  $x$ , mette una voce  $(x,I)$  nella sua tabella di inoltro.

- (a)  $I$  sarà uguale a  $l_1$  o  $l_2$  per questa voce? Spiegate perché con una sola frase.
- (b) Supponete ora che vi sia un collegamento fisico tra AS2 e AS4, mostrato dalla linea tratteggiata. Supponete che il router 1d apprenda che  $x$  è accessibile sia via AS2 che via AS4.  $I$  sarà impostato a  $l_1$  o  $l_2$ ? Spiegate perché con una sola frase.
- (c) Supponete ora che (1) un altro AS, chiamato AS5 (non mostrato nel diagramma), si trovi sul percorso tra AS2 e AS4 e che (2) il router 1d apprenda che  $x$  è accessibile via AS2, AS5 e AS4, come pure via AS3 e AS4.  $I$  sarà impostato a  $l_1$  o  $l_2$ ? Spiegate perché con una sola frase.

**P16.** Considerate la rete riportata di seguito. Gli ISP  $B$  e  $C$  forniscono rispettivamente un servizio di dorsale nazionale a quelli regionali  $A$  e  $D$ . Ciascun ISP corrisponde a un AS.  $B$  e  $C$  sono peer BGP l'uno dell'altro in due punti. Considerate il traffico che va da  $A$  a  $D$ .  $B$  preferirebbe gestire su  $C$  il traffico rivolto alla costa del Pacifico (di modo che  $C$  debba assorbire il costo del trasporto del traffico continentale), mentre  $C$  preferirebbe gestire il traffico con  $B$  attraverso il proprio punto di peering sulla costa atlantica (di modo che  $B$  debba trasportare il traffico continentale). Quale meccanismo BGP dovrebbe utilizzare  $C$ , cosicché  $B$  gestisca il traffico da  $A$  a  $D$  sul suo punto di peering sulla costa orientale? Per rispondere al quesito dovete addentrarvi nelle specifiche BGP.



**P17.** Nella Figura 5.13 considerate le informazioni di percorso per raggiungere le reti stub  $W$ ,  $X$  e  $Y$ . Sulla base delle informazioni disponibili presso  $W$  e  $X$ , quali sono le rispettive visioni della topologia di rete? Argomentate la vostra risposta. La visione topologica di  $Y$  è mostrata nella seguente figura.



**P18.** Considerate la Figura 5.13. In base all'instradamento BGP,  $B$  non inoltrerebbe mai il traffico diretto a  $Y$  attraverso  $X$ . Esistono tuttavia applicazioni molto diffuse i cui pacchetti passano prima da  $X$  e poi da  $Y$ . Identificate tali applicazioni e descrivete come possono i pacchetti seguire un percorso non dato da BGP.

**P19.** Nella Figura 5.13 supponete che esista un'altra rete stub  $V$ , cliente dell'ISP A. Supponete che  $A$  e  $B$  siano peer e che  $A$  sia cliente sia di  $B$  che di  $C$ . Supponete che  $A$  voglia che il traffico destinato a  $W$  arrivi solo da  $B$  e quello destinato a  $V$  sia da  $B$  che da  $C$ . Come dovrebbe annunciare  $A$  i suoi percorsi verso  $B$  e  $C$ ? Quali percorsi dell'AS riceve  $C$ ?

**P20.** Supponete che gli AS  $X$  e  $Z$  non siano direttamente connessi, ma siano collegati tramite l'AS  $Y$ , che  $X$  abbia un accordo di peering con  $Y$  e  $Y$  con  $Z$ . Supponete infine che  $Z$  voglia gestire il transito di tutto il traffico di  $Y$ , ma non di  $X$ . BGP permette a  $Z$  di implementare questa politica?

**P21.** Considerate i due tipi di comunicazione fra entità di gestione e dispositivi da gestire: richiesta-risposta e trap. Quali sono i pro e i contro dei due approcci in termini di (1) ridondanza, (2) tempi di notifica degli eventi eccezionali e (3) robustezza rispetto alla perdita di messaggi fra l'entità di gestione e il dispositivo gestito?

**P22.** Nel Paragrafo 5.7 abbiamo visto che si preferisce trasportare i messaggi SNMP in datagrammi UDP inaffidabili. Perché i progettisti di SNMP hanno scelto UDP invece che TCP?

## Esercizi di programmazione con le socket: Ping ICMP

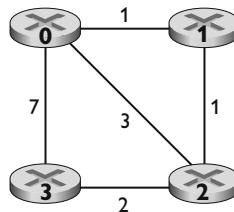
Alla fine del Capitolo 2 sono stati proposti quattro esercizi di programmazione con le socket; il quinto, proposto qui, impiega il protocollo ICMP.

Ping è una popolare applicazione usata per sondare da remoto se un host sia raggiungibile e funzionante. Viene usata anche per misurare la latenza tra due host. Funziona inviando pacchetti ICMP di “echo request” (o ping) all’host bersaglio e stando in ascolto delle risposte ICMP “echo reply” (o pong). Ping misura l’RTT, traccia le perdite di pacchetti e calcola statistiche su un campione di scambi ping-pong (minimo, media, massimo e deviazione standard dell’RTT). In questa esercitazione scrivrete la vostra applicazione Ping in Python. Per tenere l’implementazione semplice non seguite le specifiche ufficiali di RFC 1739; dovete scrivere solo la parte client, perché quella server è già contenuta nella maggior parte dei sistemi operativi. Trovate i dettagli dell’applicazione e alcuni importanti pezzi di codice sulla piattaforma MyLab di questo libro.

## Esercizi di programmazione: Routing

In questo compito di programmazione scriverete un insieme “distribuito” di procedure che implementano un instradamento distance-vector asincrono distribuito della rete mostrata di seguito.

Dovrete scrivere le seguenti routine che verranno eseguite in modo asincrono nell’ambiente simulato messo a disposizione per questo compito. Per il nodo 0, scriverete le seguenti routine.



- ***rtinit0()***. Questa procedura, che verrà chiamata una volta all’inizio della simulazione, non presenta argomenti e ha lo scopo di inizializzare la tabella delle distanze nel nodo 0 per riflettere i costi 1, 3 e 7 rispettivamente verso i nodi 1, 2 e 3. Nella figura precedente, i collegamenti sono bidirezionali e i costi nelle due direzioni sono identici. Dopo l’inizializzazione della tabella delle distanze e di ogni altra struttura dati richiesta dalle vostre routine al nodo 0, questa funzione dovrebbe inviare ai propri vicini (in questo caso, 1, 2 e 3) il costo dei suoi percorsi a costo minimo verso tutti gli altri nodi della rete. Tale informazione viene spedita in un pacchetto di aggiornamento dell’instradamento chiamando la routine *tolayer2()*, come mostrato nel testo completo del compito. Nel testo completo è descritto anche il formato del pacchetto di aggiornamento dell’instradamento.
- ***rtupdate0(struct rtpkt \*rcvdpkt)***. Questa routine verrà chiamata quando il nodo 0 riceverà un pacchetto di instradamento da uno dei suoi vicini. Il parametro *rcvdpkt* è un puntatore al pacchetto ricevuto. La routine *rtupdate0()* è il fulcro dell’algoritmo distance-vector. I valori che riceve in un pacchetto di aggiornamento dell’instradamento da qualche altro nodo *i* contengono gli attuali costi dei percorsi più brevi verso tutti gli altri nodi della rete. *rtupdate0()* utilizza i valori ricevuti per aggiornare la propria tabella delle distanze (come specificato dall’algoritmo distance-vector). Se il suo costo minimo verso un altro nodo cambia a causa dell’aggiornamento, il nodo 0 informa i suoi vicini direttamente connessi di tale cambiamento inviando loro un pacchetto di instradamento. Ricordiamo che nell’algoritmo distance-vector solo i nodi adiacenti scambiano pacchetti di instradamento. Pertanto, i nodi 1 e 2 comunicano tra loro, ma non i nodi 1 e 3.

Per i nodi 1, 2 e 3 vengono definite routine simili. Di conseguenza, in tutto scriverete otto procedure: *rtinit0()*, *rtinit1()*, *rtinit2()*, *rtinit3()*, *rtupdate0()*, *rtupdate1()*, *rtupdate2()* e *rtupdate3()*. Queste routine realizzano un calcolo distribuito e asincrono delle tabelle delle distanze per la topologia e i costi mostrati nella figura.

Troverete tutti i dettagli del compito di programmazione e il codice C necessario per creare l’ambiente hardware/software richiesto sulla piattaforma MyLab di questo libro.

È disponibile anche una versione Java del compito.

## Esercitazioni Wireshark: ICMP

Sulla piattaforma MyLab del volume troverete un’esercitazione Wireshark, che indaga l’uso del protocollo ICMP nei comandi ping e traceroute.

# Livello di collegamento e reti locali

Nei precedenti due capitoli abbiamo visto come il livello di rete fornisca un servizio di comunicazione tra due host qualsiasi della rete. I datagrammi attraversano una serie di collegamenti, cablati e wireless, che iniziano all'host sorgente, passano attraverso una serie di commutatori di pacchetto (switches), router e switch, e raggiungono la destinazione. Scendendo nella pila dei protocolli, dal livello di rete a quello di collegamento, viene naturale domandarsi come i pacchetti vengano instradati attraverso i singoli collegamenti che costituiscono il cammino di instradamento. Come sono incapsulati i datagrammi a livello di rete nei frame a livello di collegamento per la trasmissione lungo un cavo? È possibile utilizzare diversi protocolli a livello di collegamento lungo il cammino? Come vengono risolti i conflitti nei collegamenti broadcast? Esiste un indirizzamento a livello di collegamento? E, in tal caso, come si rapporta all'indirizzamento a livello di rete, appreso nel Capitolo 4? Qual è esattamente la differenza tra uno switch e un router? Risponderemo a queste e ad altre importanti domande nel corso del presente capitolo.

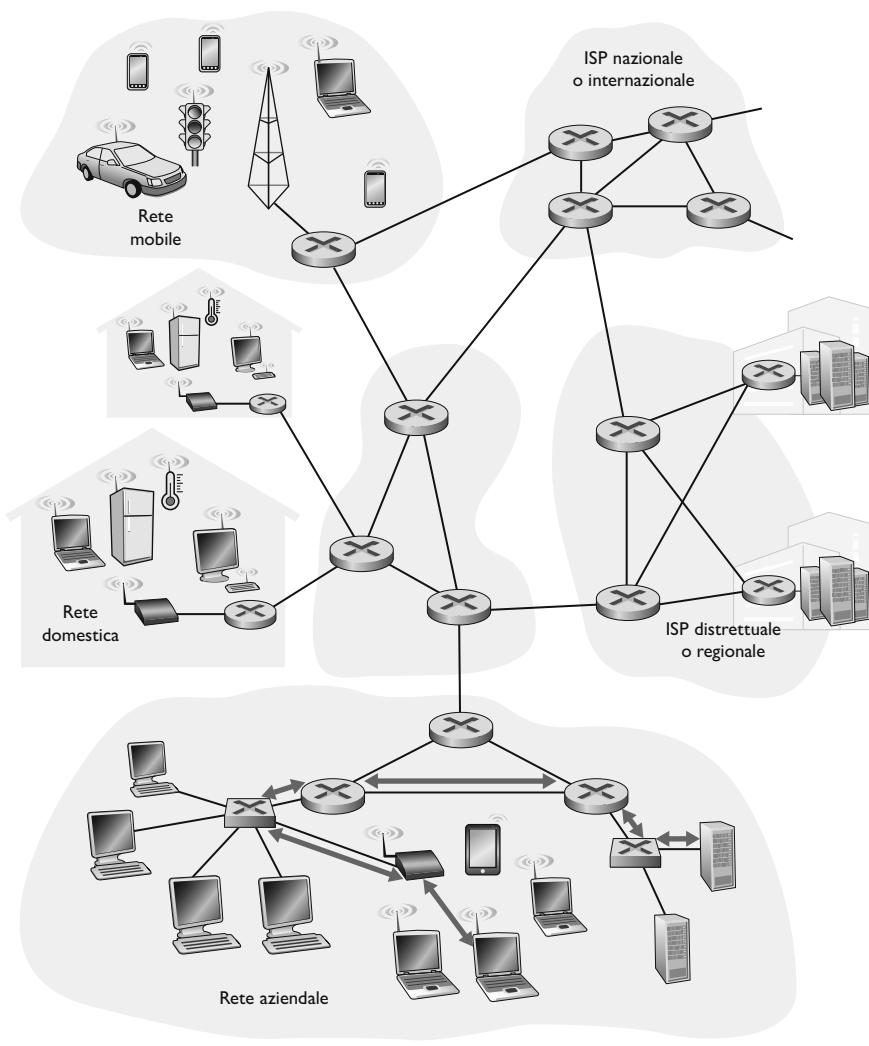
Analizzando il livello di collegamento scopriremo che esistono due tipologie fondamentali di canali. Al primo tipo appartengono i canali broadcast che collegano un gruppo di host nelle LAN wireless, nelle reti satellitari e nelle reti di accesso HFC (*hybrid fiber-coaxial cable*). Poiché più host sono connessi allo stesso canale di comunicazione broadcast, è necessario un protocollo di accesso al mezzo trasmissivo (*medium access protocol*) per coordinare la trasmissione dei frame. In alcuni casi è utilizzabile un *controllore* (*controller*) centralizzato per coordinare le trasmissioni, in altri sono gli host stessi che coordinano le trasmissioni. Al secondo tipo di canale a livello di collegamento appartiene il canale di comunicazione punto a punto, come quello che spesso si trova tra due router collegati da un canale a lunga distanza o tra il computer di un ufficio e lo switch Ethernet nelle vicinanze. Coordinare l'accesso a un collegamento punto a punto è più semplice; il materiale di riferimento sulla piattaforma MyLab riporta un'analisi dettagliata del protocollo PPP (*point-to-point protocol*) usato nei casi più svariati: dal dial-up su una linea telefonica al trasporto punto a punto di frame ad alta velocità su collegamenti in fibra ottica.

In questo capitolo tratteremo alcuni concetti e tecnologie importanti del livello di collegamento. In particolare approfondiremo l'argomento della rilevazione e correzione degli errori solo accennato nel Capitolo 3, i protocolli di accesso multiplo, le LAN commutate, tra cui Ethernet, le LAN virtuali e le reti dei data center. Alle reti WiFi e più in generale alle reti wireless, benché argomenti pertinenti il livello di collegamento, verrà dedicato l'intero Capitolo 7, disponibile in inglese sulla piattaforma MyLab.

## 6.1 Introduzione al livello di collegamento

Vediamo subito alcuni termini utili: faremo riferimento a qualunque dispositivo che opera a livello di collegamento (livello 2) indicandolo semplicemente come **nodo**. Ci riferiremo inoltre ai canali di comunicazione, che collegano nodi adiacenti lungo un cammino, come a **collegamenti** (*link*). Quindi i datagrammi che devono essere trasferiti da un host sorgente a uno di destinazione devono essere trasportati lungo ciascun collegamento nel percorso da un estremo all'altro. Consideriamo come esempio la rete aziendale mostrata in fondo alla Figura 6.1 e supponiamo di voler inviare un datagramma da uno degli host wireless a uno dei server. Il datagramma dovrà attraversare sei collegamenti: un collegamento WiFi tra l'host sorgente e l'access point WiFi; un collegamento Ethernet dall'access point allo switch a livello di collegamento; un collegamento tra lo switch a livello di collegamento e il router; un collegamento tra i due router; un collegamento Ethernet tra il router e lo switch a livello di collegamento e infine un collegamento Ethernet tra lo switch e il server. Su ogni collegamento, un nodo trasmittente incapsula il datagramma in un **frame del livello di collegamento** (*link-layer frame*) e lo trasmette lungo il collegamento stesso.

Per capire meglio il livello di collegamento e come questo si colleghi al livello di rete, consideriamo un'analogia del mondo dei trasporti. Consideriamo un'agenzia di viaggi che stia pianificando un viaggio per un turista che vuole andare da Battipaglia a Losanna in Svizzera. L'agenzia di viaggi decide che è più conveniente per il turista prendere una macchina da Battipaglia all'aeroporto di Capodichino a Napoli, quindi volare sull'aeroporto di Ginevra e infine prendere un treno dall'aeroporto di Ginevra alla stazione dei treni di Losanna. Nel momento in cui l'agenzia di viaggi effettua le tre prenotazioni, risulta che la compagnia di autonoleggio è responsabile di portare il turista da Battipaglia a Capodichino; è responsabilità della compagnia aerea trasportarlo da Capodichino a Ginevra e infine è responsabilità dei treni svizzeri trasportare il turista da Ginevra a Losanna. Ognuno dei tre tratti di viaggio è “diretto” tra due posizioni “adiacenti”; si noti che i tre segmenti di trasporto sono gestiti da aziende diverse che usano metodi di trasporto completamente differenti (macchina, aeroplano e treno). Benché i modi di trasporto siano differenti, ognuno fornisce il servizio basilare di muovere i passeggeri da un luogo a quello adiacente. In questa analogia sui trasporti, il turista è il datagramma, ogni tratto di viaggio è un collegamento, il modo di trasporto è un protocollo a livello di collegamento e l'agenzia di viaggi è il protocollo di instradamento.



**Figura 6.1** Sei passaggi a livello di collegamento tra host wireless e server.

### 6.1.1 Servizi offerti dal livello di collegamento

Sebbene il servizio di base del livello di collegamento sia il trasporto di datagrammi da un nodo a quello adiacente lungo un singolo canale di comunicazione, i dettagli dei servizi forniti possono variare da un protocollo all'altro. Tra i possibili servizi che possono essere offerti dai protocolli a livello di collegamento sono inclusi i seguenti.

- **Framing.** Quasi tutti i protocolli encapsulano i datagrammi del livello di rete all'interno di un frame a livello di collegamento, prima di trasmetterlo. I frame sono costituiti da un campo dati, nel quale è inserito il datagramma, e da vari campi di intestazione. La struttura del frame è specificata dal protocollo. I differenti formati di frame saranno analizzati nella seconda parte di questo capitolo, quando esamineremo gli specifici protocolli del livello di collegamento.
- **Accesso al collegamento.** Un protocollo che controlla l'accesso al mezzo trasmissivo (*MAC, medium access control*) specifica le regole con cui immettere

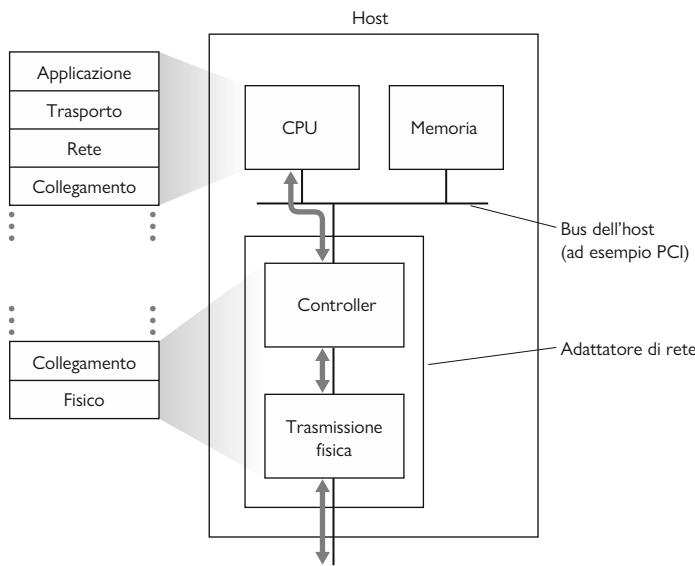
i frame nel collegamento. Nei collegamenti punto a punto, con un solo mittente e un solo destinatario, in cui il protocollo MAC è semplice (o non esiste), il mittente può inviare il frame quando il canale risulta libero. Un caso più interessante è quello in cui vari nodi condividono un singolo canale broadcast (il cosiddetto problema di accesso multiplo): in questo caso il protocollo MAC aiuta a coordinare la trasmissione dei frame da parte dei nodi.

- **Consegna affidabile.** I protocolli a livello di collegamento che forniscono un servizio di consegna affidabile garantiscono il trasporto senza errori di ciascun datagramma. Abbiamo già visto che anche alcuni protocolli di trasporto (come TCP) forniscono un servizio di consegna affidabile. Analogamente a quello del livello di trasporto, anche il servizio di consegna affidabile del livello di collegamento può essere realizzato attraverso acknowledgment e ritrasmissioni (Paragrafo 3.4). Il servizio di consegna affidabile è spesso utilizzato per i collegamenti soggetti a elevati tassi di errore (per esempio i collegamenti wireless), allo scopo di correggere l'errore localmente, sul collegamento dove è avvenuto l'errore, piuttosto che costringere i protocolli di trasporto o di applicazione a procedere alla ritrasmissione dei dati dalla sorgente alla destinazione. Tuttavia, la consegna affidabile del livello di collegamento può essere considerata non necessaria nei collegamenti che presentano un basso numero di errori sui bit, come nel caso di collegamenti con fibra ottica, cavo coassiale e doppino, ragion per cui molti protocolli del livello di collegamento non implementano questo servizio.
- **Rilevazione e correzione degli errori.** Il nodo ricevente può decidere erroneamente che un bit in un frame, trasmesso come un 1, sia uno 0 e viceversa. Gli errori di bit sono causati dall'attenuazione di segnale e dai disturbi elettromagnetici. Siccome non è utile inoltrare datagrammi contenenti errori, molti protocolli del livello di collegamento forniscono un meccanismo per rilevarne la presenza. Ciò è possibile grazie all'inserimento, da parte del nodo trasmittente, di bit di controllo di errore all'interno del frame e all'esecuzione di un semplice controllo da parte del nodo ricevente. Come abbiamo visto nei Capitoli 3 e 4, anche i livelli di trasporto e di rete di Internet forniscono una seppur limitata rilevazione degli errori: il checksum (*Internet checksum*). Il rilevamento degli errori a livello di collegamento è però solitamente più sofisticato, in quanto implementato in hardware. La correzione dell'errore è simile alla rilevazione degli errori, ma in più il nodo ricevente determina il punto preciso del frame in cui si è verificato l'errore (per poi correggerlo).

### 6.1.2 Dov'è implementato il livello di collegamento?

Prima di iniziare uno studio dettagliato del livello di collegamento, domandiamoci dove venga implementato. Ci concentreremo sul sistema periferico, in quanto abbiamo appreso dal Capitolo 4 che il livello di collegamento è implementato nelle *line card* dei router. Il livello di collegamento di un host è implementato in hardware o in software? Su una scheda o un chip separati? Come si interfaccia con le altre parti dell'hardware dell'host e con il suo sistema operativo?

La Figura 6.2 mostra la tipica architettura di un host. Per un dato collegamento, il protocollo del livello di collegamento è sostanzialmente realizzato da un **adattatore di rete** (*network adapter*), noto anche come **scheda di rete** (NIC,



**Figura 6.2** Adattatore di rete: le sue relazioni con gli altri componenti dell'host e con le funzionalità della pila di protocolli.

*network interface controller*). Il cuore della scheda di rete è il controller a livello di collegamento (*link layer controller*), di solito un chip dedicato, che implementa molti dei servizi a livello di collegamento (framing, accesso al collegamento, rilevazione degli errori) identificati nel paragrafo precedente. La maggior parte, quindi, delle funzionalità del controller è implementata in hardware. Per esempio, l'adattatore Intel 700 [Intel 2020] implementa il protocollo Ethernet (Paragrafo 6.5) e quello Atheros AR5006 [Atheros 2020] implementa i protocolli WiFi 802.11 (Capitolo 7, disponibile in inglese sulla piattaforma MyLab).

Lato mittente, il controller prende un datagramma, creato e memorizzato nella memoria dell'host dai livelli più alti della pila di protocolli, lo incapsula in un frame a livello di collegamento riempiendone i vari campi dell'intestazione, e lo trasmette sul canale di comunicazione, seguendo il protocollo di accesso al canale. Dall'altro lato, un controller riceve l'intero frame, estrae il datagramma e lo consegna al livello di rete. Se il protocollo del livello di collegamento fornisce il servizio di rilevazione degli errori, allora è il controller trasmittente a impostare i bit di rilevazione degli errori ed è quello ricevente che esegue il controllo.

La Figura 6.2 mostra anche che, mentre la maggior parte del livello di collegamento è implementato in hardware sulla scheda di rete, una parte è invece realizzata in software e viene eseguita dalla CPU dell'host. Le componenti software del livello di collegamento implementano tipicamente le funzionalità del livello più alto, come l'assemblaggio delle informazioni di indirizzamento e l'attivazione dell'hardware del controller. Lato ricevente, il software del livello di collegamento risponde agli interrupt del controller (per esempio, dovuti alla ricezione di uno o più frame), effettua la gestione di condizioni di errore e il passaggio del datagramma fino al livello di rete. Il livello di collegamento, quindi, è una combinazione di hardware e software: il luogo dove, nella pila di protocolli, il software incontra l'hardware. [Intel 2020] fornisce un'interessante pa-

noramica e anche una descrizione dettagliata del controller XL710 dal punto di vista della programmazione.

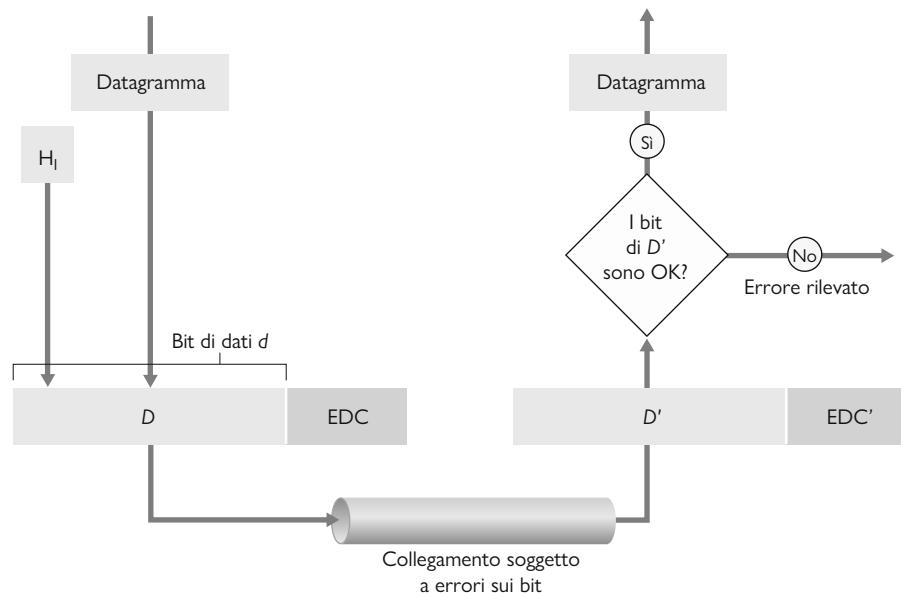
## 6.2 Tecniche di rilevazione e correzione degli errori

Nel paragrafo precedente abbiamo visto che **rilevamento e correzione degli errori sui bit** (quelli alterati nei frame scambiati tra nodi adiacenti) sono due servizi generalmente forniti dal livello di collegamento, ma abbiamo visto (Capitolo 3) che tali servizi sono spesso forniti anche dal livello di trasporto. Esamineremo ora alcune tra le tecniche più semplici utilizzabili per il trattamento degli errori sui bit. La teoria e l'implementazione di questo argomento rappresentano l'oggetto principale di molti testi (per esempio: [Schwartz 1980] o [Bertsekas 1991]). Ci limiteremo quindi a una sintetica esposizione dell'argomento, con l'obiettivo di fornire informazioni generali sulle possibilità offerte dalle tecniche di rilevazione e di correzione degli errori e di vedere come poche, semplici tecniche funzionino e vengano usate in pratica a livello di collegamento.

La Figura 6.3 schematizza lo scenario di riferimento. Al nodo trasmittente, ai dati  $D$  che devono essere protetti da errori vengono aggiunti dei bit detti *EDC* (*error-detection and -correction*). Generalmente, i dati che devono essere protetti non includono soltanto datagrammi trasferiti verso il basso dal livello di rete per la trasmissione, ma anche le informazioni relative agli indirizzi del livello di collegamento, i numeri di sequenza e altri campi nell'intestazione del frame. I dati  $D$  e i bit *EDC* sono inviati in un frame al nodo ricevente che legge una sequenza di bit  $D'$  ed *EDC'* che può essere diversa dall'originale, come risultato della modifica dei bit in transito.

Il nodo ricevente deve determinare se  $D'$  coincide con  $D$ , potendo contare soltanto su  $D'$  e su *EDC'*. È importante la terminologia relativa alle decisioni

**Figura 6.3** Scenario di rilevazione e correzione degli errori.



del nodo ricevente nella Figura 6.3 (ci chiediamo se il nodo rilevi l'errore, non se si è verificato un errore). Le attuali tecniche non sempre consentono al nodo ricevente di rilevare se si sono verificati errori nei bit. Anche con l'utilizzo dei bit di rilevazione degli errori è possibile che ci siano degli **errori non rilevati**; vale a dire che il nodo ricevente potrebbe non accorgersi che le informazioni ricevute contengono errori. Di conseguenza, il ricevente potrebbe consegnare un datagramma errato al livello di rete o ignorare che il contenuto di alcuni campi nell'intestazione del frame sia stato alterato. Dovremo quindi optare per uno schema per la rilevazione degli errori che riduca la probabilità di questo evento. Generalmente, le tecniche più sofisticate comportano un'elevata ridondanza, nel senso che sono necessari calcoli più complessi e la trasmissione di molti bit aggiuntivi.

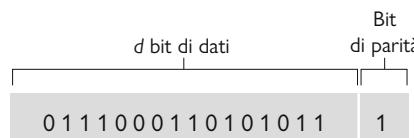
Vediamo ora tre tecniche per rilevare gli errori nei dati trasmessi: **controllo di parità** (*parity check*), per illustrare l'idea di base della rilevazione e correzione degli errori, **tecniche di checksum**, solitamente utilizzato nel livello di trasporto e **controllo a ridondanza ciclica** (*cyclic redundancy check*), in genere impiegato dalle schede di rete a livello di collegamento.

### 6.2.1 Controllo di parità

La forma più semplice di rilevamento degli errori è quella che utilizza un unico **bit di parità** (*parity bit*). Supponiamo che le informazioni da inviare,  $D$  nella Figura 6.4, siano costituite da  $d$  bit. In uno schema di parità pari, il mittente include un bit addizionale e sceglie il suo valore in modo da rendere pari il numero totale di bit 1 nei  $d + 1$  bit trasmessi (l'informazione originale più il bit di parità). Nello schema di parità dispari, il valore del bit di parità è scelto in modo che ci sia un numero dispari di bit 1. La Figura 6.4 mostra uno schema di parità pari, con un bit di parità inserito in un campo separato.

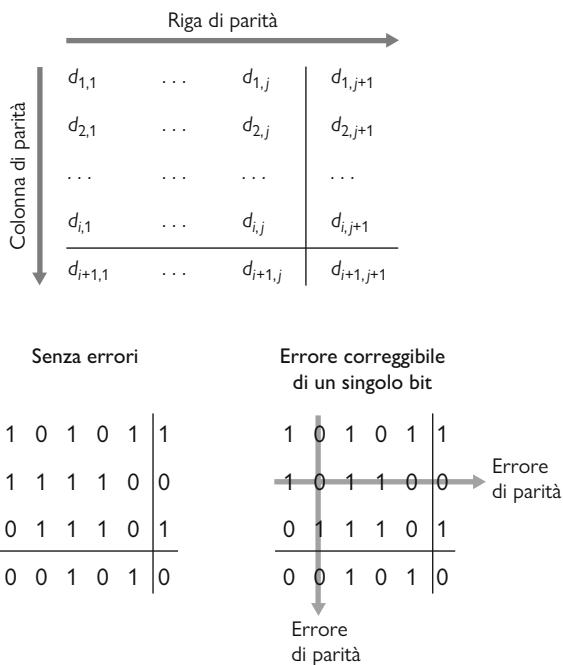
Con un solo bit di parità ciò che il ricevente deve fare è molto semplice: contare il numero di bit a 1 tra quelli ricevuti. Se trova un numero dispari di bit 1, sa che si è verificato almeno un errore in un bit (più precisamente, sa che si è verificato un numero *dispari* di errori nei bit).

Tuttavia che cosa accade se si è verificato un numero pari di errori nei bit? In questo caso ci troveremmo in presenza di un errore non rilevato. Se la probabilità di errori nei bit è bassa e si può assumere che gli errori siano indipendenti, l'eventualità di errori multipli in un pacchetto è estremamente ridotta e un solo bit di parità potrebbe risultare sufficiente. Tuttavia, è stato statisticamente rilevato che gli errori tendono generalmente a verificarsi a gruppi di bit consecutivi (*burst*) piuttosto che in modo indipendente. La probabilità che non vengano rilevati errori a burst in un frame protetto da un solo bit di parità può avvicinarsi al 50% [Spragins 1991]. È evidente che occorre adottare una strategia più efficiente per la rilevazione degli errori. Prima di analizzare gli schemi



**Figura 6.4** Parità pari a un bit.

**Figura 6.5** Parità pari bidimensionale.



adottati in pratica, consideriamo una semplice generalizzazione del bit di parità che ci fornirà un approccio alla tecnica di correzione dell’errore.

La Figura 6.5 illustra una generalizzazione bidimensionale dello schema di parità con un bit. In questo caso, i  $d$  bit del dato  $D$  sono suddivisi in  $i$  righe e  $j$  colonne per ognuna delle quali è stato calcolato un valore di parità. I risultanti  $i + j + 1$  bit di parità contengono bit per la rilevazione dell’errore nei frame a livello di collegamento.

Supponiamo ora che si verifichi un solo errore nei  $d$  bit originali. Con questo schema di **parità bidimensionale** i bit di parità della colonna e della riga contenenti il bit errato individueranno l’errore. Il ricevente può quindi non solo rilevare che si è verificato un errore, ma può utilizzare gli indici di colonna e di riga per identificare il bit alterato e correggerlo. La Figura 6.5 mostra un esempio in cui il bit a 1 in posizione (2,2) è “corrotto” ed è diventato 0 (errore rilevabile e correggibile dal ricevente). Nonostante la nostra esposizione sia stata centrata sui  $d$  bit originali, anche un errore negli stessi bit di parità è rilevabile e correggibile. Questo schema può rilevare (ma non correggere) qualsiasi combinazione di due errori in un pacchetto. Altre proprietà dello schema bidimensionale sono prese in considerazione nei problemi di fine capitolo.

La capacità del ricevente sia di rilevare sia di correggere gli errori è conosciuta come **forward error correction** (FEC, *correzione degli errori in avanti*). Queste tecniche sono comunemente utilizzate in dispositivi audio di registrazione e riproduzione, come i lettori CD audio. In una configurazione di rete, le tecniche FEC possono essere impiegate singolarmente o abbinate a tecniche ARQ (Capitolo 3). Le tecniche FEC sono molto utili, perché possono diminuire il numero di ritrasmissioni, ma è forse più importante il fatto che permettono al ricevente l’immediata correzione degli errori. Ciò consente di evitare un’at-

tesa equivalente all'RTT, necessaria affinché il trasmittente riceva un pacchetto NAK e il pacchetto ritrasmesso torni al ricevente (un importante vantaggio per le applicazioni in tempo reale) [Rubenstein 1998], o in collegamenti (come quelli nello spazio profondo) con grande ritardo di propagazione. Alcune trattazioni delle tecniche FEC nei protocolli di controllo dell'errore sono [Biersack 1992; Nonnenmacher 1998; Byers 1998; Shacham 1990].

### 6.2.2 Checksum

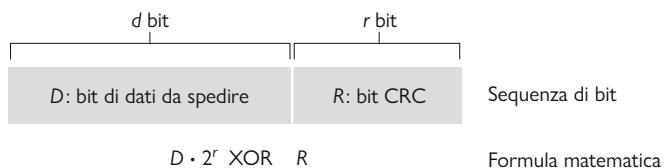
Nelle tecniche che utilizzano il checksum i  $d$  bit di dati della Figura 6.4 sono trattati come una sequenza di numeri interi da  $k$  bit. Un semplice metodo per eseguire il checksum è quello di sommare questi interi da  $k$  bit e usare i bit del risultato come bit per la rilevazione degli errori. Il **checksum di Internet** si basa su questo approccio: i dati sono trattati come interi di 16 bit e sommati. Il complemento a 1 di questa somma costituisce il checksum di Internet che viene trasposto nell'intestazione dei segmenti. Come detto nel Paragrafo 3.3, il ricevente controlla il checksum calcolando il complemento a 1 della somma dei dati ricevuti (compreso il checksum stesso) e verifica che tutti i bit del risultato siano 0. Se non è così, viene segnalato un errore. L'RFC 1071 tratta l'algoritmo del checksum di Internet e la sua implementazione. Nei protocolli TCP e UDP il checksum è calcolato per tutti i campi (intestazione e dati). In IP il checksum è calcolato solo sull'intestazione, perché i segmenti TCP/UDP hanno il proprio. In altri protocolli, per esempio XTP [Stayer 1992], si calcola un checksum sull'intestazione e un altro sull'intero pacchetto.

I metodi di checksum richiedono informazioni addizionali nel pacchetto relativamente piccole (TCP e UDP usano solo 16 bit), ma forniscono una prevenzione dagli errori piuttosto limitata in confronto alle tecniche di controllo a ridondanza ciclica (CRC) utilizzate nel livello di collegamento. A questo punto diventa ovvio chiedersi perché a livello di trasporto venga utilizzato il checksum e a livello di collegamento il CRC. Il livello di trasporto è generalmente eseguito dal software del sistema operativo dell'host. Dato che la rilevazione degli errori a livello di trasporto è implementata come software, risulta fondamentale poter disporre di schemi di rilevazione di errore semplici e veloci. D'altro lato, la rilevazione di errori al livello di collegamento è implementata, mediante hardware dedicato, nelle schede di rete che possono effettuare più rapidamente le complesse operazioni di CRC. Feldmeier [Feldmeier 1995] presenta tecniche per l'implementazione ad alte prestazioni non solo per checksum ponderato, ma anche per CRC e altri codici.

### 6.2.3 Controllo a ridondanza ciclica (CRC)

Una tecnica di rilevazione dell'errore largamente utilizzata nelle più recenti reti di calcolatori è basata sui **codici di controllo a ridondanza ciclica** (CRC, *cyclic redundancy check*). I codici CRC sono anche detti **codici polinomiali**, in quanto è possibile vedere la stringa di bit da trasmettere come un polinomio i cui coefficienti sono i bit della stringa, con le operazioni sulla stringa di bit interpretate come aritmetica polinomiale.

Vediamo come operano i codici CRC. Consideriamo  $d$  bit costituenti i dati  $D$  da trasmettere e supponiamo che sorgente e destinazione si siano accordate su una stringa di  $r + 1$  bit, conosciuta come **generatore**, che indicheremo con  $G$ .

**Figura 6.6** CRC.

È necessario che il bit più significativo (quello più a sinistra) di  $G$  sia 1. L’idea alla base del codice CRC è illustrata nella Figura 6.6. Dato un blocco di dati,  $D$ , il mittente sceglierà  $r$  bit addizionali,  $R$ , e li unirà a  $D$  in modo da ottenere una stringa di  $d + r$  bit che, interpretata come numero binario, sia esattamente divisibile per  $G$  (nell’aritmetica modulo 2). Il processo di controllo del CRC è semplice: se la divisione  $(d + r)/G$  ha un resto diverso da 0, il ricevente sa che si è verificato un errore; altrimenti i dati sono accettati come corretti.

Tutti i calcoli di CRC sono eseguiti in aritmetica modulo 2 senza riporti nelle addizioni e prestiti nelle sottrazioni. Questo significa che addizione e sottrazione sono operazioni identiche e che entrambe equivalgono a un’operazione di OR esclusivo (XOR) sui bit degli operandi. Per esempio:

$$\begin{aligned} 1011 \text{ XOR } 0101 &= 1110 \\ 1001 \text{ XOR } 1101 &= 0100 \end{aligned}$$

o analogamente

$$\begin{aligned} 1011 - 0101 &= 1110 \\ 1001 - 1101 &= 0100 \end{aligned}$$

Moltiplicazioni e divisioni sono eseguite come in base 2, ma sottrazioni e addizioni non hanno riporti. Come nella normale aritmetica binaria, la moltiplicazione di una stringa per  $2^k$  corrisponde allo slittamento a sinistra della stringa di  $k$  posizioni. Quindi, dati  $R$  e  $D$ , la quantità  $D \times 2^r$  XOR  $R$  fornisce come risultato la stringa di  $d + r$  bit mostrata nella Figura 6.6. Di seguito utilizzeremo questa caratterizzazione algebrica dello schema  $d + r$  bit della Figura 6.6.

Torniamo alla questione cruciale di come il trasmettente calcoli  $R$  e ricordiamo che vogliamo trovare  $R$  in modo che esista un valore di  $n$  tale che

$$D \times 2^r \text{ XOR } R = nG$$

Vale a dire, vogliamo scegliere  $R$  in modo che  $G$  sia divisibile per  $D \times 2^r$  XOR  $R$  senza resto. Se eseguiamo l’operazione di OR esclusivo (cioè sommiamo modulo 2, senza resto) di  $R$  con entrambi i membri dell’espressione sopra indicata otteniamo

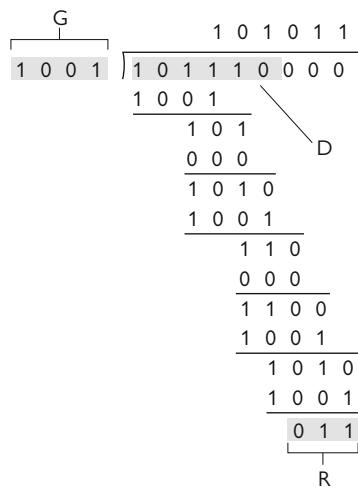
$$D \times 2^r = nG \text{ XOR } R$$

Quest’espressione ci mostra che se dividiamo  $D \times 2^r$  per  $G$ , il valore del resto è precisamente  $R$ . In altre parole possiamo calcolare  $R$  come

$$R = \text{resto di } (D \times 2^r / G)$$

La Figura 6.7 illustra questo calcolo per  $D = 101110$ ,  $d = 6$ ,  $G = 1001$ ,  $r = 3$ . I nuovi bit trasmessi in questo caso sono 101110011. Potete verificare il calcolo e controllare che in effetti:

$$D \times 2^r = 101011 \times G \text{ XOR } R.$$



**Figura 6.7** Esempio di calcolo di CRC.

Sono stati definiti dei generatori standard di 8, 12, 16 e 32 bit. Nelle celle ATM si utilizza una CRC a 8 bit per proteggere i 5 byte dell'intestazione. Lo standard CRC-32 a 32 bit, in numerosi protocolli IEEE del livello di collegamento, usa il generatore

$$G_{\text{CRC-32}} = 100000100110000010001110110110111$$

CRC può rilevare errori a burst inferiori a  $r + 1$  bit: ovvero tutti gli errori consecutivi di non oltre  $r$  bit saranno rilevati. Inoltre, in conformità con le appropriate assunzioni, la probabilità di avere un burst più lungo di  $r + 1$  bit è  $1 - 0,5^r$ . Inoltre, il CRC può rilevare qualsiasi numero dispari di errori nei bit. Si veda [Williams 1993] per una trattazione sull'implementazione dei controlli CRC. La teoria alla base dei codici CRC come di quelli più potenti va oltre gli obiettivi di questo testo. [Schwartz 1980] fornisce un'eccellente introduzione a questi argomenti.

## 6.3 Collegamenti e protocolli di accesso multiplo

Nell'introduzione di questo capitolo abbiamo osservato che esistono due tipi di collegamento di rete: punto a punto e broadcast. Il **collegamento punto a punto** è costituito da un trasmittente a un'estremità del collegamento e da un unico ricevente all'altra. Molti protocolli del livello di collegamento sono stati progettati per questi collegamenti, tra cui PPP (*point-to-point protocol*) e HDLC (*high-level data link control*). Il **collegamento broadcast** può avere più nodi trasmittenti e riceventi connessi allo stesso canale broadcast condiviso. Il termine broadcast indica che, quando un nodo trasmette un frame, il canale lo diffonde e tutti gli altri nodi ne ricevono una copia. Ethernet e Wireless LAN sono esempi di tecnologie con collegamenti broadcast. In questo paragrafo faremo un passo indietro rispetto agli specifici protocolli del livello di collegamento ed esamineremo per primo un problema di fondamentale importanza: come coordinare l'accesso di più nodi trasmittenti e riceventi in un canale bro-

adcast condiviso, ossia il **problema dell'accesso multiplo**. Al termine di questo paragrafo prenderemo in considerazione anche l'utilizzo dei canali ad accesso multiplo nelle LAN, cioè in reti geograficamente confinate in un singolo edificio, in un'azienda o in un campus universitario.

La nozione di broadcast è familiare a tutti, se non altro perché utilizzata per le trasmissioni televisive. A differenza di queste, però, i nodi su un canale broadcast di una rete di calcolatori possono sia trasmettere sia ricevere. Un'analogia più appropriata è quella di una festa dove molte persone sono in una stanza (l'aria fornisce il canale broadcast) parlando e ascoltandosi a vicenda. Una seconda analogia è quella di una classe nella quale insegnante e studenti condividono lo stesso singolo canale broadcast. Il problema principale in entrambi gli scenari descritti è determinare chi e quando debba parlare (cioè trasmettere). I protocolli impiegati sono estremamente vari. Ecco qualche esempio.

“Concedi a ciascuno la possibilità di parlare”.

“Non parlare finché non sei interrogato”.

“Non monopolizzare la conversazione”.

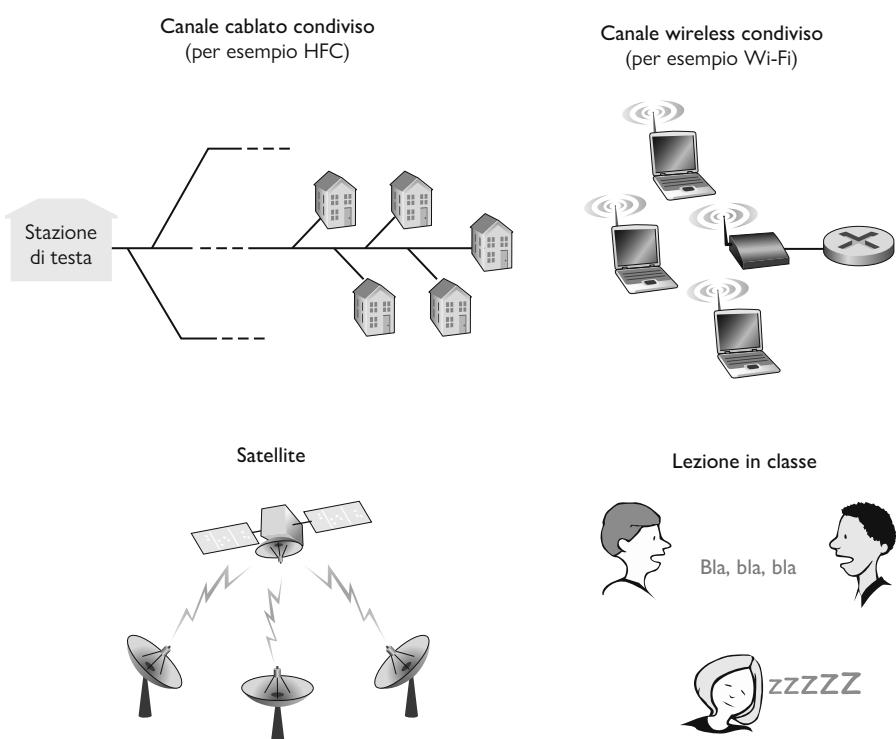
“Alza la mano se devi porre una domanda”.

“Non interrompere se qualcuno sta parlando”.

“Non addormentarti quando qualcuno parla”.

Analogamente, le reti di calcolatori usano protocolli simili – i cosiddetti **protocolli di accesso multiplo** – che fissano le modalità con cui i nodi regolano le loro trasmissioni sul canale condiviso. Come mostrato nella Figura 6.8 i protocolli

**Figura 6.8** Canali ad accesso multiplo.



di accesso multiplo sono richiesti in un'ampia varietà di configurazioni di rete, comprese le LAN cablate, quelle wireless e le reti satellitari. Anche se, tecnicamente, ciascun nodo accede al canale broadcast attraverso la propria scheda di rete, in questo paragrafo useremo il termine *nodo* per indicare il dispositivo che trasmette e riceve. In pratica, centinaia o anche migliaia di nodi possono comunicare direttamente su un canale broadcast.

Dato che tutti i nodi sono in grado di trasmettere frame, può accadere che due o più lo facciano nello stesso istante. In tal caso tutti i nodi ricevono contemporaneamente più frame. Tra questi si genera una **collisione** a causa della quale nessuno dei nodi riceventi riuscirà a interpretare i frame che, in un certo senso, risultano ingarbugliati tra loro. Di conseguenza con la collisione si verifica una perdita di frame, mentre il canale rimane inutilizzato. È chiaro che, se una situazione di questo genere dovesse ripetersi con eccessiva frequenza, gran parte della banda del canale risulterebbe sprecata.

Per far sì che il canale broadcast venga utilizzato in modo efficiente, occorre coordinare le trasmissioni dei nodi attivi. Compito, questo, che spetta ai protocolli di accesso multiplo in merito ai quali, negli ultimi 40 anni, sono stati scritti numerosi saggi e tesi di laurea. Un compendio dei primi 20 anni si trova in [Rom 1990]. Inoltre le ricerche sui protocolli di accesso multiplo sono sempre attive sotto la spinta di tipologie di collegamenti sempre nuove, in particolare quelli wireless.

In generale possiamo classificare praticamente tutti i protocolli di accesso multiplo in una di queste categorie: **protocolli a suddivisione del canale** (*channel partitioning protocol*), **protocolli ad accesso casuale** (*random access protocol*) e **protocolli a rotazione** (*taking-turn protocol*).

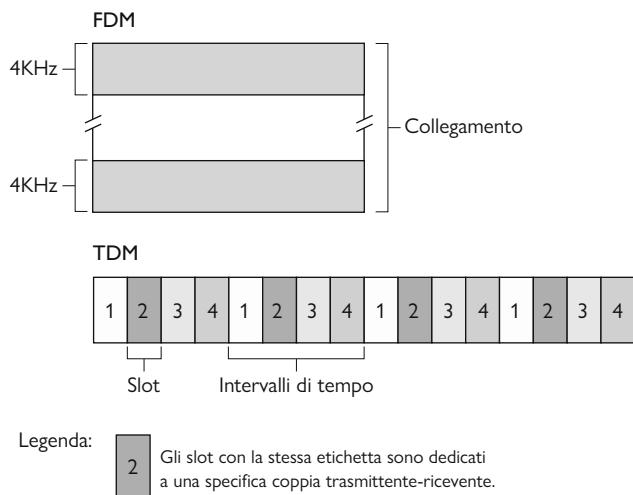
Concludiamo questa panoramica rilevando che, idealmente, un protocollo di accesso multiplo per un canale broadcast con velocità di  $R$  bit al secondo dovrebbe avere le seguenti caratteristiche:

1. Quando un solo nodo deve inviare dati, questo dispone di un throughput (velocità media di invio dati) pari a  $R$  bps.
2. Quando  $M$  nodi devono inviare dati, questi dispongono di un throughput pari a  $R/M$  bps. Ciò non implica necessariamente che ciascuno degli  $M$  nodi abbia sempre una velocità istantanea di trasmissione di  $R/M$ , ma che ciascun nodo dovrebbe avere una velocità di trasmissione media di  $R/M$  in un dato, appropriato intervallo di tempo.
3. Il protocollo è decentralizzato; in altre parole non ci sono nodi principali che, qualora non funzionassero correttamente, potrebbero bloccare l'intero sistema.
4. Il protocollo è semplice, in modo che risulti economico da implementare.

### 6.3.1 Protocolli a suddivisione del canale

Ricordiamo dal Paragrafo 1.3 che il multiplexing a divisione di tempo, TDM (*time-division multiplexing*) e quello a divisione di frequenza, FDM (*frequency division multiplexing*), sono due tecniche che possono essere utilizzate per suddividere la larghezza di banda di un canale broadcast fra i nodi che lo condividono. Per esempio, supponiamo che il canale supporti  $N$  nodi e che la sua velocità di trasmissione sia di  $R$  bps. TDM suddivide il tempo in **intervalli di tempo**

**Figura 6.9** Esempi di TDM e FDM con quattro nodi.



(*time frame*) e poi divide ciascun intervallo di tempo in  $N$  **slot temporali** (*time slot*). Si noti che il time frame di TDM non deve essere confuso con l'unità di dati a livello di collegamento che le schede di rete si scambiano, che è anch'essa chiamata frame. Per ridurre la confusione, in questo paragrafo ci riferiremo all'unità di dati del livello di collegamento come a un pacchetto. Ogni slot è quindi assegnato a uno degli  $N$  nodi. Ogni volta che un nodo ha un pacchetto da inviare, trasmette i bit del pacchetto durante lo slot di tempo assegnatogli. Generalmente, le dimensioni dello slot sono stabilite in modo da consentire la trasmissione di un singolo pacchetto. La Figura 6.9 mostra un esempio di TDM con quattro nodi. Ritornando alla nostra analogia con la festa, se questa è regolata da TDM, consente a un invitato di parlare per un intervallo di tempo stabilito. Lo stesso tempo spetterà quindi a un altro invitato, e così via. Una volta che tutti hanno avuto l'opportunità di parlare, la sequenza si ripete.

TDM viene utilizzato in quanto riesce a evitare le collisioni ed è perfettamente imparziale: ciascun nodo ottiene, durante ciascun intervallo di tempo, un tasso trasmissivo di  $R/N$  bps. In ogni caso, TDM presenta due specifici inconvenienti. In effetti anche quando non vi sono altri nodi che devono inviare un pacchetto, quello che vuole trasmettere è vincolato ad attendere il suo turno nella sequenza di trasmissione e a non superare il limite medio di  $R/N$  bps. Immaginiamo che ci sia un solo invitato che abbia qualcosa da dire e che questa sia una delle rare circostanze in cui tutti vogliono ascoltare. In questo particolare caso, è evidente che TDM risulta una pessima scelta di protocollo per l'accesso multiplo in questa particolare festa.

Mentre TDM suddivide il canale condiviso in intervalli di tempo, FDM lo divide in frequenze differenti (ciascuna con una larghezza di banda di  $R/N$ ) e assegna ciascuna frequenza a un nodo. Quindi, a partire da un canale da  $R$  bps, FDM crea  $N$  canali di  $R/N$  bps. FDM condivide alcuni vantaggi e alcuni svantaggi del precedente protocollo. Anche FDM evita le collisioni e divide equamente la larghezza di banda tra gli  $N$  nodi. Tuttavia, anche con FDM la larghezza di banda è limitata a  $R/N$ , pure quando vi è un solo nodo che ha un pacchetto da spedire.

Un terzo protocollo di suddivisione del canale è l'**accesso multiplo a divisione di codice** (CDMA, *code division multiple access*). Mentre TDM e FDM assegnano rispettivamente ai nodi slot di tempo e di frequenze, CDMA assegna loro un **codice**. Ciascun nodo, quindi, utilizza il proprio codice univoco per codificare i dati inviati. Se i codici sono scelti accuratamente, le reti CDMA avranno un'eccellente proprietà: consentiranno a nodi differenti di trasmettere simultaneamente e ai rispettivi destinatari di ricevere correttamente i bit dei dati codificati (assumendo che il ricevente conosca il codice) nonostante le interferenze derivanti dalle trasmissioni degli altri nodi. CDMA è stato utilizzato per un certo periodo da impianti militari (per le sue caratteristiche di robustezza alle interferenze) e ha poi trovato largo impiego in ambito civile, in particolare nella telefonia cellulare. Poiché è molto legato ai canali wireless, ne affronteremo i dettagli tecnici nel Capitolo 7, disponibile in inglese sulla piattaforma MyLab. Per ora, è sufficiente sapere che i codici CDMA, come gli slot TDM e le frequenze FDM, possono essere allocati agli utenti del canale ad accesso multiplo.

### 6.3.2 Protocolli ad accesso casuale

La seconda vasta classe di protocolli per l'accesso multiplo comprende quelli ad accesso casuale e concerne situazioni in cui un nodo trasmette sempre alla massima velocità consentita dal canale, cioè  $R$  bps. Quando si verifica una collisione, i nodi coinvolti ritrasmettono anche più volte i loro frame (cioè i pacchetti) fino a quando sono correttamente ricevuti dalla destinazione, senza collisioni. La ritrasmissione del frame non è immediata, ma il nodo attende per un periodo di tempo casuale (*random delay*); ogni nodo coinvolto in una collisione seleziona un ritardo casuale indipendente da quello degli altri nodi. Le differenti scelte arbitrarie del tempo di attesa operate dai diversi nodi possono consentire ai frame con ragionevole probabilità di non subire ulteriori collisioni.

In letteratura sono riportate decine, se non centinaia, di protocolli di questo tipo [Rom 1990; Bertsekas 1991]. In questo paragrafo ne descriveremo alcuni tra i più utilizzati: il protocollo ALOHA [Abramson 1970; Abramson 1985; Abramson 2009] e i protocolli di accesso multiplo con rilevazione della portante (CSMA, *carrier sense multiple access*) [Kleinrock 1975b]. L'accesso casuale utilizzato da Ethernet [Metcalfe 1976] non è altro che un famoso protocollo CSMA largamente utilizzato.

#### Slotted ALOHA

Iniziamo la nostra analisi dei protocolli di accesso casuale con uno tra i più semplici: lo slotted ALOHA. Nella nostra descrizione assumeremo che:

- tutti i frame consistano esattamente di  $L$  bit;
- il tempo sia suddiviso in slot di  $L/R$  secondi (cioè che uno slot equivalga al tempo di trasmissione di un pacchetto);
- i nodi comincino la trasmissione dei frame solo all'inizio degli slot;
- i nodi siano sincronizzati in modo che tutti sappiano quando iniziano gli slot;
- qualora in uno slot due o più frame collidano, tutti i nodi della rete rilevino l'evento prima del termine dello slot.

Indichiamo con  $p$  una probabilità, vale a dire un numero tra 0 e 1. Le operazioni dei nodi slotted ALOHA sono semplici.

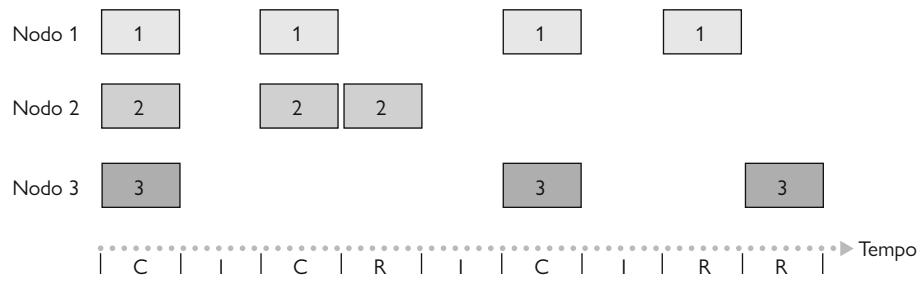
- Quando un nodo ha un nuovo frame da spedire, attende fino all'inizio dello slot successivo e poi trasmette l'intero frame.
- Se non si verifica una collisione, l'operazione ha avuto successo, quindi non occorre effettuare una ritrasmissione e il nodo può predisporre l'invio di un nuovo frame.
- Se si verifica una collisione, il nodo la rileva prima del termine dello slot e ritrasmette con probabilità  $p$  il suo frame durante gli slot successivi, fino a quando l'operazione non ha successo.

Per specificare che cosa intendiamo con “ritrasmissione con probabilità  $p$ ” possiamo immaginare che il nodo lanci una moneta truccata: l'evento testa corrisponde alla ritrasmissione, che si verifica con probabilità  $p$ , quello croce corrisponde a “salta questo slot e lancia di nuovo la moneta in quello successivo”. Questo accade con probabilità  $(1 - p)$ . I nodi coinvolti nella collisione lanciano le proprie monete indipendentemente gli uni dagli altri.

Questo protocollo sembra possedere grandi vantaggi. A differenza della suddivisione del canale, slotted ALOHA consente a un singolo nodo di trasmettere continuamente pacchetti alla massima velocità del canale, quando è il solo nodo attivo. Un nodo si dice attivo se ha un frame da trasmettere. Slotted ALOHA è anche fortemente decentralizzato, in quanto ciascun nodo rileva le collisioni e decide indipendentemente quando ritrasmettere, anche se è comunque necessario che gli slot siano sincronizzati ai nodi. Analizzeremo tra breve una versione di ALOHA che non fa uso di slot temporali e CSMA; entrambi non richiedono sincronizzazione e sono quindi completamente decentralizzati.

Slotted ALOHA funziona bene quando è attivo un solo nodo, ma è altrettanto efficiente in presenza di molti nodi attivi? Esistono in questo caso due possibili problemi. In primo luogo (Figura 6.10), una certa frazione degli slot presenterà collisioni e di conseguenza andrà “sprecata”. Il secondo problema è che una grande frazione degli slot risulterà vuota, perché tutti i nodi attivi terminano la trasmissione in conseguenza della politica di trasmissione probabilistica. I soli slot non sprecati saranno esattamente quelli utilizzati da un solo nodo per trasmettere. Lo slot in cui trasmette un solo nodo è chiamato **slot riuscito**

**Figura 6.10** I nodi 1, 2 e 3 collidono nel primo slot. Il nodo 2 riesce a trasmettere nel quarto slot, il nodo 1 nell'ottavo e il nodo 3 nel nono.



Tempo

| C | I | | C | R | I | | C | I | | R | R | |

Legenda:    C = Slot di collisione  
              I = Slot inutilizzato  
              R = Slot riuscito

(*successful slot*). L'*efficienza* di un protocollo di accesso multiplo che fa uso di slot temporali è definita come la frazione di slot riusciti in presenza di un elevato numero di nodi attivi che hanno sempre un elevato numero di pacchetti da spedire. Notiamo che se non venisse utilizzata alcuna forma di controllo dell'accesso e se ciascun nodo ritrasmettesse subito dopo che si è effettuata la collisione, l'*efficienza* sarebbe zero. È evidente che slotted ALOHA si comporta meglio, ma di quanto?

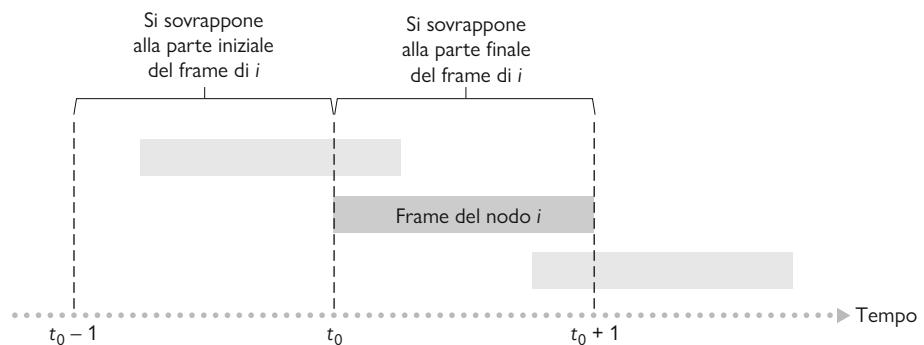
Ora deriveremo l'*efficienza* massima del protocollo. Per semplicità, modificiamo leggermente il protocollo e assumiamo che ciascun nodo tenti di trasmettere un frame in uno slot con probabilità  $p$ . Supponiamo, cioè, che ciascun nodo abbia sempre un frame da spedire e che il nodo trasmetta con probabilità  $p$  un nuovo frame o uno che ha già subito una collisione. In primo luogo, ipotizziamo di avere  $N$  nodi. In questo caso la probabilità che un dato slot sia vincente è data dalla probabilità che un solo nodo trasmetta, mentre i rimanenti  $N - 1$  rimangono inattivi. La probabilità che un dato nodo trasmetta è  $p$ ; la probabilità che i rimanenti nodi non trasmettano è  $(1 - p)^{N-1}$ . Quindi la probabilità di successo di un dato nodo è  $p(1 - p)^{N-1}$ . Poiché ci sono  $N$  nodi, la probabilità che un nodo arbitrario abbia successo è  $Np(1 - p)^{N-1}$ .

Di conseguenza, con  $N$  nodi attivi, l'*efficienza* dello slotted ALOHA è  $Np(1 - p)^{N-1}$ . Per ottenere la massima efficienza con  $N$  nodi attivi bisogna trovare il valore  $p^*$  che massimizza questa espressione. (Per una descrizione generale di questa derivazione si vedano i problemi in fondo al capitolo). Per ottenere la massima efficienza per un elevato numero di nodi attivi, ricaviamo il limite di  $Np^*(1 - p^*)^{N-1}$  per  $N$  che tende all'infinito. Dopo aver svolto questi calcoli, otterremo che l'*efficienza* massima del protocollo è data da  $1/e = 0,37$ . Vale a dire che quando un gran numero di nodi ha molti pacchetti da trasmettere, allora (nel caso migliore) solo il 37% degli slot compie lavoro utile. Quindi, l'*effettiva velocità* di trasmissione del canale non è  $R$  bps, ma soltanto  $0,37 R$  bps! Un'analisi simile dimostra anche che il 37% degli slot viaggia vuoto e che il 26% subisce collisioni. Immaginate il povero amministratore della rete che ha acquistato un sistema slotted ALOHA a 100 Mbps, aspettandosi di riuscire a utilizzare la rete per trasmettere dati a un grande numero di utenti a un tasso aggregato di, diciamo, 80 Mbps. Sebbene il canale abbia la capacità di trasmettere un frame alla velocità di 100 Mbps, a lungo termine il volume di dati trasmesso con successo in questo canale sarà inferiore a 37 Mbps.

## ALOHA

Slotted ALOHA richiede che tutti i nodi sincronizzino le loro trasmissioni a partire dall'inizio di uno slot. Il primo protocollo ALOHA [Abramson 1970] era in realtà un protocollo privo di slot, completamente decentralizzato. Nel protocollo ALOHA puro, appena arriva un frame (cioè, un datagramma del livello di rete raggiunge la scheda del nodo trasmittente), il nodo lo trasmette immediatamente e integralmente nel canale broadcast. Se un frame va in collisione, allora il nodo lo ritrasmette immediatamente (dopo aver completato la trasmissione del frame) con probabilità  $p$ . Altrimenti, attenderà il tempo di trasmissione del frame. Trascorso questo tempo, il nodo ritrasmetterà il frame con probabilità  $p$  o aspetterà, restando inattivo per un altro periodo di tempo, con probabilità  $1 - p$ .

**Figura 6.11** Interferenza delle trasmissioni in ALOHA puro.



Per determinare l'efficienza massima di ALOHA puro, analizziamo un singolo nodo. Partiremo dalle stesse assunzioni definite per la precedente analisi e, come unità di tempo, prenderemo il tempo di trasmissione di un frame. A ogni dato istante, la probabilità che un nodo stia trasmettendo è  $p$ . Supponiamo che la trasmissione di un frame inizi al tempo  $t_0$ . Come illustrato nella Figura 6.11, affinché questo frame sia trasmesso con esito positivo nessun altro nodo può cominciare la sua trasmissione nell'intervallo di tempo  $(t_0 - 1, t_0]$ , in quanto si sovrapporrebbe con l'inizio della trasmissione del frame del nodo  $i$ . La probabilità che tutti gli altri nodi non diano inizio a una trasmissione in questo intervallo è  $(1 - p)^{N-1}$ . Analogamente, nessun altro nodo può iniziare la trasmissione mentre il nodo  $i$  sta trasmettendo, in quanto si sovrapporrebbe all'ultima parte della trasmissione di quel frame. La probabilità che tutti gli altri nodi non comincino a trasmettere in questo intervallo è anch'essa  $(1 - p)^{N-1}$ . Di conseguenza, possiamo notare che la probabilità che un certo nodo abbia successo nella trasmissione è  $p(1 - p)^{2(N-1)}$ . Ricavando il limite come prima, otteniamo che l'efficienza massima del protocollo ALOHA puro è solo  $1/(2e)$  (esattamente la metà dello slotted ALOHA). Questo è il prezzo che il protocollo ALOHA deve pagare per la completa decentralizzazione.

### CSMA: accesso multiplo con rilevamento della portante

Nei due protocolli ALOHA i nodi prendono la decisione di trasmettere indipendentemente dall'attività degli altri nodi collegati al canale broadcast. In particolare, un nodo non presta attenzione al fatto che vi sia un altro nodo che sta trasmettendo né arresta la trasmissione se un altro nodo inizia a interferire con la sua trasmissione. Nella nostra analogia con il comportamento umano, i protocolli ALOHA corrispondono a un maleducato che continua a parlare senza preoccuparsi del fatto che altri stiano conversando tra loro. Certo, noi disponiamo di “protocolli” che ci consentono non solo di comportarci più educatamente, ma anche di diminuire il tempo di “collistione” tra le diverse conversazioni e, di conseguenza, di aumentare la quantità di dati che viene scambiata. In particolare, esistono due regole importanti per dialogare in modo civile, che sono le seguenti.

- *Ascoltare prima di parlare.* Se qualcun altro sta parlando, aspettare finché abbia concluso. Nel mondo delle reti, ciò è chiamato **rilevamento della portante** (*carrier sensing*): un nodo ascolta il canale prima di trasmettere. Se il

canale sta già trasmettendo un frame, il nodo aspetta finché rileva che il canale è libero per un intervallo di tempo e quindi inizia la trasmissione.

- *Se qualcun altro comincia a parlare insieme a voi, smettete di parlare.* Nel mondo delle reti, ciò si chiama **rilevamento della collisione** (*collision detection*): il nodo che sta trasmettendo rimane contemporaneamente in ascolto del canale. Se osserva che un altro nodo sta trasmettendo un frame che interferisce col suo, arresta la propria trasmissione, aspetta un intervallo di tempo casuale e poi ripete il processo.

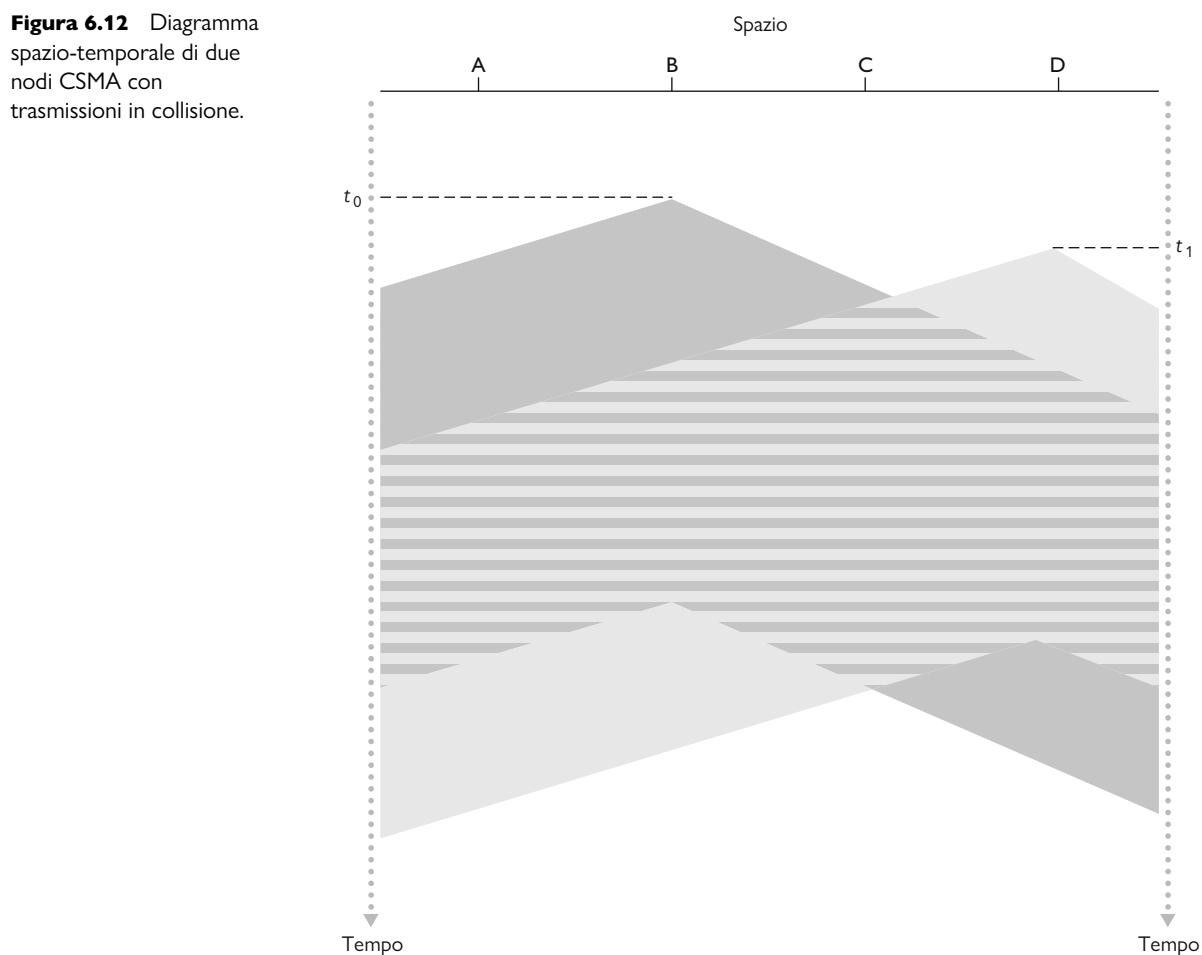
Queste due regole sono alla base dei **protocolli CSMA** (*carrier sense multiple access*, accesso multiplo con rilevamento della portante) e **CSMA/CD** (*CSMA with collision detection*, CSMA con rilevamento della collisione) [Kleinrock 1975b; Metcalfe 1976; Lam 1980; Rom 1990]. Sono state proposte molte variazioni di CSMA e CSMA/CD. Considereremo alcune tra le loro fondamentali caratteristiche.

La prima domanda è: perché, se tutti i nodi eseguono il rilevamento della portante, si verificano collisioni? Dopo tutto, i nodi rinunciano alla trasmissione quando rilevano che qualcun altro sta trasmettendo. La risposta alla domanda può essere meglio spiegata mediante diagrammi spazio-temporali [Molle 1987]. La Figura 6.12 ne illustra uno relativo a quattro nodi (A, B, C, D). L'asse orizzontale mostra la posizione dei nodi sul canale condiviso e quello verticale rappresenta il tempo.

All'istante  $t_0$ , il nodo B rileva che il canale è inattivo (nessun altro nodo sta trasmettendo), comincia la trasmissione e i suoi bit si propagano in entrambe le direzioni lungo il mezzo trasmisivo. La propagazione verso valle dei bit di B al crescere del tempo indica che è necessario un intervallo di tempo non nullo per l'effettiva propagazione dei bit di B lungo il canale (benché questa avvenga a una velocità paragonabile a quella della luce). Al tempo  $t_1 > t_0$ , il nodo D ha un frame da spedire. Sebbene al tempo  $t_1$  il nodo B stia ancora trasmettendo, i suoi bit non hanno ancora raggiunto D e quindi, in quell'istante, D ritiene libero il canale. Nel pieno rispetto del protocollo, D allora comincia a trasmettere il suo frame. Dopo un breve periodo, la trasmissione di B comincerà a interferire con quella di D. Dalla Figura 6.12 risulta evidente che il **ritardo di propagazione** da un estremo all'altro di un canale broadcast (il tempo richiesto da un segnale per propagarsi da un nodo all'altro) avrà un ruolo cruciale nel determinare le sue prestazioni. Maggiore è questo ritardo, maggiore sarà la possibilità che il nodo, pur attento a rilevare la portante, non si accorga che è già cominciata la trasmissione da parte di un altro nodo.

### **Accesso multiplo a rilevazione della portante con rilevamento delle collisioni (CSMA/CD)**

Nella Figura 6.12 i nodi non eseguono il rilevamento delle collisioni: B e D continuano a trasmettere i loro pacchetti anche se si è verificata una collisione. Quando un nodo rileva una collisione, cessa immediatamente la trasmissione. La Figura 6.13 mostra lo stesso scenario della precedente, ma ora i due nodi terminano la loro trasmissione poco dopo aver rilevato la collisione. È evidente che l'introduzione della rilevazione di collisione in un protocollo di accesso multiplo ne migliorerà le prestazioni, evitando l'inutile trasmissione dell'intero frame danneggiato dall'interferenza con un altro frame.

**Figura 6.12** Diagramma spazio-temporale di due nodi CSMA con trasmissioni in collisione.

Prima di analizzare il protocollo CSMA/CD riassumiamo le sue operazioni dal punto di vista di una scheda di rete collegata a un canale broadcast.

1. La scheda ottiene direttamente un datagramma dal livello di rete, prepara un frame a livello di collegamento e lo sistema in un suo buffer.
2. Quando riscontra che il canale è libero (cioè, non vi è energia di segnale che entri nella scheda dal canale), inizia la trasmissione del frame. Se il canale risulta occupato, resta in attesa sino al momento in cui non rileva più il segnale e solo allora inizia l'invio del frame.
3. Durante la trasmissione verifica la presenza di eventuali segnali provenienti da altre schede di rete sul canale broadcast.
4. La scheda di rete, se trasmette l'intero frame senza rilevare energia di segnale proveniente da altre schede, ha finito il suo lavoro; altrimenti, se riscontra energia di segnale durante la trasmissione, interrompe immediatamente la trasmissione del frame.
5. Dopo aver annullato la trasmissione, la scheda di rete aspetta per un tempo casuale e poi ritorna al passo 2.

**BOX 6.1****CASO DI STUDIO****Norm Abramson e ALOHAnet**

Norm Abramson era un ingegnere con la passione per il surf e un particolare interesse per la commutazione di pacchetto. Nel 1969 questa combinazione di interessi lo portò alla University of Hawaii. Le Hawaii sono un arcipelago costituito da molte isole montuose, che rendono difficoltosa l'installazione e il funzionamento delle reti terrestri. Quando non si dedicava al surf, Abramson pensava a come progettare una rete radio a commutazione di pacchetto. La rete che progettò aveva un host centrale e molti nodi secondari disseminati tra le isole. La rete aveva due canali, con diverse bande di frequenza. Il canale in downstream distribuiva i pacchetti in broadcast da un host centrale agli host secondari, mentre l'altro canale in upstream operava in senso inverso. Oltre all'invio di pacchetti d'informazione, l'host centrale trasmetteva anche una conferma per ciascun pacchetto ricevuto con successo dagli host secondari.

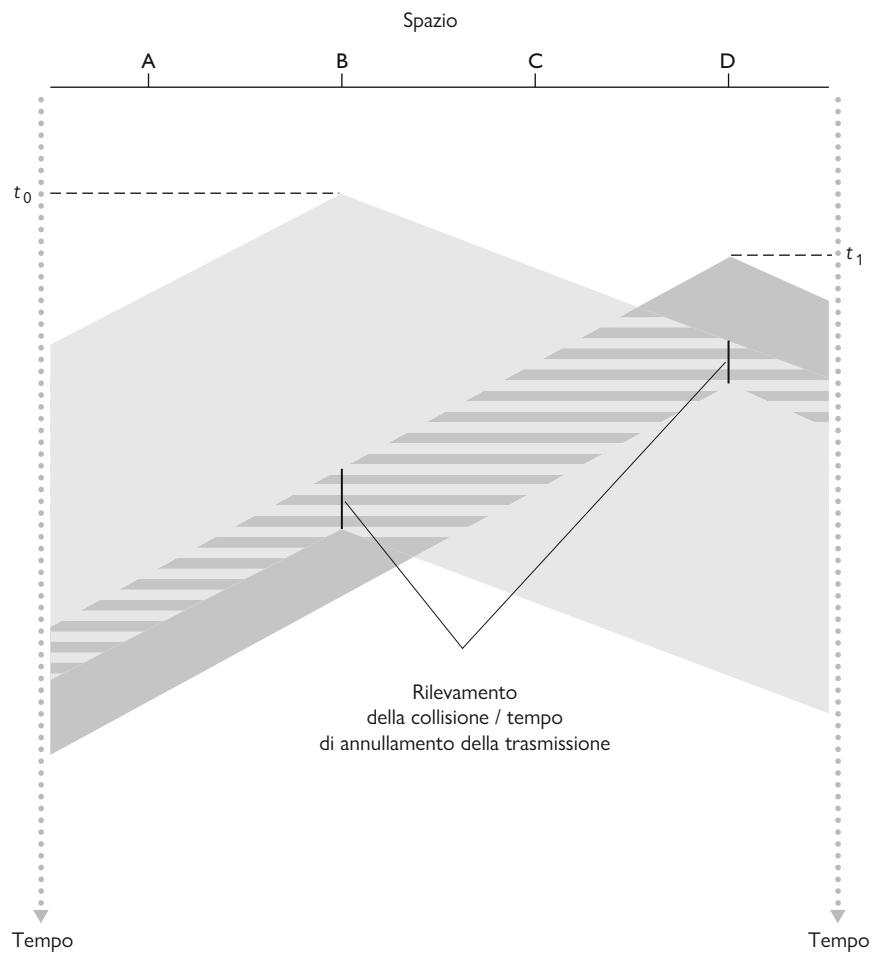
Dato che gli host secondari trasmettevano pacchetti in modo decentralizzato, era inevitabile che nel canale in upstream si verificassero delle collisioni. Queste osservazioni portarono Abramson a inventare il protocollo ALOHA puro, che abbiamo appena descritto. Nel 1970, grazie ai finanziamenti di ARPA, Abramson collegò ALOHAnet ad ARPAnet. Il lavoro di Abramson è importante non solo perché fu il primo esempio di una rete radio a commutazione di pacchetto, ma anche perché ispirò Bob Metcalfe. Alcuni anni dopo l'invenzione di Abramson, Metcalfe modificò il protocollo ALOHA per dare vita al protocollo CSMA/CD e alle LAN Ethernet.

La necessità di aspettare per un intervallo di tempo casuale piuttosto che fissato dovrebbe essere chiara: se due nodi trasmettono frame simultaneamente e quindi aspettano per lo stesso periodo di tempo, continueranno a entrare in collisione per sempre. Ma qual è un intervallo di tempo opportuno da scegliere per il tempo di attesa casuale (detto anche **tempo di backoff**)? Se l'intervallo è grande e il numero di nodi che collidono è piccolo, i nodi probabilmente aspetteranno per un lungo intervallo di tempo (mentre il canale rimarrà inattivo), prima di ripetere il ciclo. D'altro canto, se l'intervallo è piccolo e il numero di nodi che collidono è grande, sarà probabile che valori scelti casualmente siano simili e che quindi i nodi trasmittenti tornino a collidere. Idealmente occorrerebbe un intervallo di tempo piccolo, quando il numero di nodi in collisione è piccolo, e uno grande quando il numero di nodi è grande.

L'algoritmo di **binary exponential backoff** (*attesa binaria esponenziale*), usato sia in Ethernet sia nei protocolli di accesso multiplo delle reti HFC DOCSIS [DOCSIS 3.1 2014], risolve elegantemente tale problema. In particolare, quando il trasmettitore riscontra l' $n$ -esima collisione durante la trasmissione di un dato frame, stabilisce casualmente un valore  $K$  nell'insieme  $\{0, 1, 2, \dots, 2^n - 1\}$ . Quindi più alto è il numero di collisioni, più grande sarà l'intervallo da cui  $K$  viene estratto. In Ethernet, l'intervallo di tempo che un nodo deve aspettare è pari a quello necessario per inviare  $K$  volte 512 bit e il valore massimo che  $n$  può assumere è 10.

Vediamo un esempio. Supponiamo che la scheda di rete tenti per la prima volta di inviare un frame e che mentre lo trasmette rilevi una collisione. Allora sceglie  $K = 0$  con probabilità 0,5 o  $K = 1$  con probabilità 0,5. Se la scheda sceglie

**Figura 6.13** CSMA con rilevamento della collisione.



$K = 0$ , allora inizia immediatamente a rilevare la portante del canale. Se sceglie  $K = 1$ , aspetta il tempo necessario per inviare 512 bit (0,01 microsecondi per Ethernet a 100 Mbps) prima di ritornare al ciclo. Dopo una seconda collisione,  $K$  è scelto con uguale probabilità fra 0, 1, 2 e 3. Dopo tre collisioni,  $K$  è scelto con uguale probabilità fra 0, 1, 2, 3, 4, 5, 6, 7. Dopo dieci o più collisioni,  $K$  è scelto uniformemente fra 0, 1, 2, ..., 1023. Quindi la cardinalità dell'insieme da cui è scelto  $K$  cresce esponenzialmente con il numero delle collisioni raddoppiando il suo valore massimo a ognuna di esse. Per questa ragione l'algoritmo è detto di **attesa binaria esponenziale**.

A questo punto occorre rilevare che quando una scheda di rete predisponde un nuovo frame per la trasmissione, inizializza l'algoritmo CSMA/CD. In sostanza, la scheda di rete non prende in considerazione le collisioni avvenute precedentemente. È quindi possibile che con un nuovo frame la trasmissione abbia immediatamente successo, mentre le altre schede di rete rimangono nella fase di attesa esponenziale.

### Efficienza di CSMA/CD

Quando un solo nodo ha un frame da inviare può trasmettere alla massima velocità del canale (10 Mbps, 100 Mbps o 1 Gbps per Ethernet). Se, invece, i nodi che vogliono trasmettere sono numerosi, l'effettiva velocità di trasmissione sul canale può risultare notevolmente inferiore. Definiamo **efficienza di CSMA/CD** la frazione di tempo media durante la quale i frame sono trasferiti sul canale senza collisioni in presenza di un alto numero di nodi attivi, con un'elevata quantità di frame da inviare. Per presentare un'approssimazione in forma chiusa dell'efficienza di Ethernet, indichiamo con  $d_{prop}$  il tempo massimo che occorre al segnale per propagarsi fra una coppia di schede di rete. Dato  $d_{trasm}$ , ossia il tempo necessario per trasmettere un frame della maggior dimensione possibile (circa 1,2 ms per Ethernet a 10 Mbps), avremo la semplice approssimazione (se ne veda la derivazione esatta in [Lam 1980] e [Bertsekas 1991]):

$$\text{Efficienza} = \frac{1}{1 + 5d_{prop}/d_{trasm}}$$

Dalla formula si evince che quando  $d_{prop}$  tende a 0, l'efficienza tende a 1. Questo conferma la nostra ipotesi che quando il ritardo di propagazione è nullo, i nodi in cui si verifica una collisione interromperanno immediatamente la trasmissione senza sprecare la capacità del canale. Inoltre, al crescere di  $d_{trasm}$  l'efficienza tende a 1. Il ragionamento è intuitivo: nel momento in cui un frame si appropria del canale può trattenerlo per un periodo di tempo estremamente lungo; di conseguenza il canale svolge lavoro produttivo per la maggior parte del tempo.

### 6.3.3 Protocolli a rotazione

Ricordiamo che due proprietà auspicabili di un protocollo di accesso multiplo sono: (1) quando un solo nodo è attivo, deve avere un throughput di  $R$  bps, e (2) quando sono attivi  $M$  nodi, ciascuno deve avere un throughput vicino a  $R/M$  bps. I protocolli ALOHA e CSMA possiedono la prima proprietà, ma non la seconda. Questo ha incoraggiato i ricercatori a creare un'altra classe di protocolli: i **protocolli a rotazione** (*taking-turn protocol*). Esistono decine di questi protocolli, ciascuno con molte varianti; ne vedremo qui due tra le più importanti.

Il primo è il **protocollo polling** nel quale uno dei nodi, designato come principale, interpella “a turno” gli altri. In particolare, il nodo principale invia un messaggio al nodo 1, comunicandogli che può trasmettere fino a un dato numero massimo di frame. Dopo che il nodo 1 ha trasmesso dei frame, il nodo principale avvisa il nodo 2 che può iniziare a inviare un dato numero massimo di frame. Il nodo principale può determinare che un nodo ha terminato di inviare i propri frame, osservando la mancanza di segnale sul canale. La procedura prosegue in questo modo, con il nodo principale che interpella in modo ciclico tutti gli altri nodi.

Il protocollo polling elimina le collisioni e gli slot vuoti che si riscontrano nei protocolli ad accesso casuale e quindi può avere un'efficienza più elevata, ma presenta anche alcuni svantaggi. Il primo è l'introduzione del ritardo di polling: il tempo richiesto per notificare a un nodo il permesso di trasmettere. Se, per esempio, è attivo solo un nodo, allora questo trasmetterà a un tasso inferiore di  $R$  bps, in quanto il nodo principale deve contattare ciclicamente i nodi inattivi

ogni volta che quello attivo ha terminato l'invio del suo numero massimo di frame. Il secondo inconveniente, potenzialmente molto più serio, è che, se il nodo principale si guasta, l'intero canale diventa inattivo. Il protocollo Bluetooth che studieremo nel Paragrafo 7.3 (il Capitolo 7 è disponibile in inglese sulla piattaforma MyLab) è un esempio di protocolli polling.

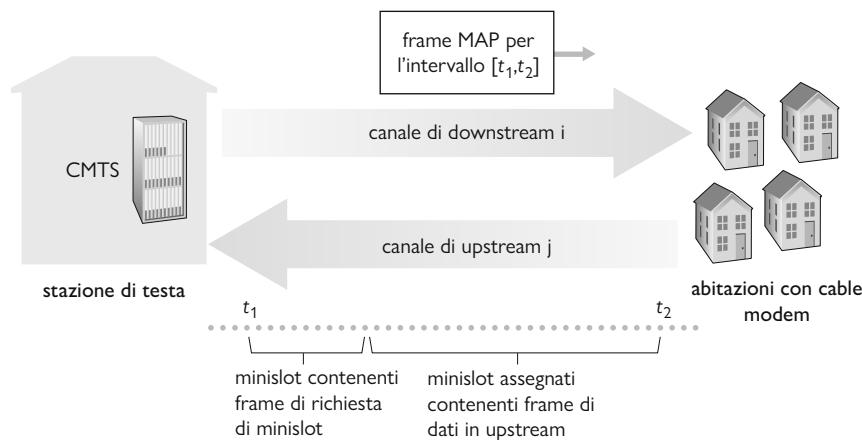
Il secondo protocollo a rotazione è il **protocollo token-passing** (a passaggio di gettone). In questo caso non esiste un nodo principale, ma un frame, un messaggio di controllo detto **token** (gettone), che circola fra i nodi seguendo un ordine prefissato. Per esempio: il nodo 1 deve sempre spedire il token al nodo 2 che deve inviarlo al nodo 3 e così via fino a quando il nodo  $N$  lo restituirà al nodo 1 che riprenderà la procedura. Se il nodo che riceve il token non ha pacchetti da inviare, lo inoltra immediatamente al successivo. Altrimenti, procede a trasmettere il numero massimo di frame consentito, prima di inoltrare il token al nodo successivo. Questo protocollo è decentralizzato e altamente efficiente, ma non è privo di problemi. Per esempio, il guasto di un nodo può mettere fuori servizio l'intero canale. Oppure, se un nodo non riesce a inoltrare il token, occorre invocare procedure di recupero, per rimetterlo in circolazione. Tutti i prodotti commerciali di questo tipo, tra i quali il protocollo FDDI [Jain 1994] e il token ring 802.5 [IEEE 802.5 2012], hanno dovuto affrontare questo e altre questioni delicate.

### 6.3.4 DOCSIS: il protocollo a livello di collegamento per reti di accesso a Internet HFC

Nei precedenti tre paragrafi abbiamo studiato tre grandi classi di protocollo di accesso multiplo: i protocolli a ripartizione di canale, i protocolli ad accesso casuale e i protocolli a rotazione. Una rete di accesso HFC sarà un eccellente caso di studio per vedere come si comporta ognuna delle tre classi di protocollo di accesso multiplo.

Ricordiamo dal Paragrafo 1.2.1 che la rete di accesso HFC connette normalmente alcune migliaia di cable modem residenziali a un apparato situato nella stazione di testa (CMTS, *cable modem termination system*). DOCSIS (*data-over-cable service interface specification*) [DOCSIS 3.1 2014; Hamzeh 2015]] specifica l'architettura della rete cablata e i suoi protocolli; usa FDM per dividere i segmenti di rete in downstream (dal CMTS al modem) da quelli in upstream (dal modem al CMTS) in canali a più frequenze. Ogni canale in downstream ha un'ampiezza in frequenza tra i 24 e i 192 MHz, con un massimo throughput pari circa a 1,6 Gbps per canale. Ogni canale di upstream ha una larghezza di banda massima tra i 6,4 e i 96 MHz e un throughput massimo di circa 1 Gbps per canale. I canali downstream e upstream sono tutti canali broadcast. I frame trasmessi sul canale di downstream dal CMTS vengono ricevuti da tutti i modem cablati che ricevono quel canale; tuttavia, poiché c'è un solo canale CMTS che trasmette sul canale di downstream, non ci sono problemi di accesso multiplo. La direzione di upstream è però più interessante e tecnicamente affascinante, perché più modem condividono lo stesso canale di upstream (frequenza) verso il CMTS e quindi, potenzialmente, possono avvenire delle collisioni.

Come mostrato nella Figura 6.14 ogni canale di upstream è diviso in intervalli di tempo (come in TDM), ognuno dei quali contiene una sequenza di mi-



**Figura 6.14** Canali di upstream e downstream tra CMTS e cable modem.

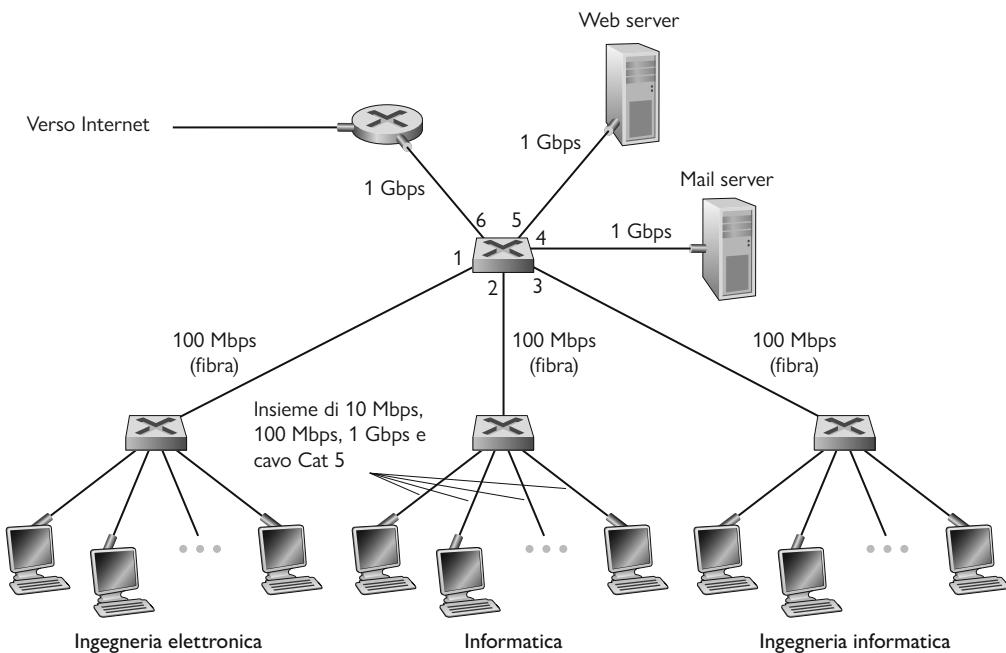
ni-slot durante i quali i modem possono trasmettere al CMTS, il quale dà esplicitamente l'autorizzazione a ogni modem di trasmettere durante uno specifico mini-slot. CMTS effettua tale operazione inviando un messaggio di controllo noto come messaggio MAP sul canale di downstream per specificare quale modem, avente dati da spedire, possa trasmettere durante quale mini-slot nell'intervallo di tempo specificato nel messaggio di controllo. Poiché i mini-slot sono esplicitamente allocati ai modem, CMTS può assicurare che non ci siano collisioni di trasmissione durante un mini-slot.

Tuttavia come fa CMTS a sapere quali modem hanno dati da inviare? I modem inviano richieste di mini-slot indirizzati al CMTS, durante uno specifico insieme di mini-slot temporali dedicati a questa funzione, come mostrato nella Figura 6.14. Questi frame di richiesta di mini-slot sono trasmessi in modo casuale e quindi possono collidere tra di loro. Un modem cablato non può sentire direttamente né quando un canale di upstream è occupato né se ci sono collisioni. Il modem cablato però inferisce che un frame di richiesta di mini-slot ha subito una collisione se non riceve risposta alla richiesta di allocazione nel messaggio di controllo in un frame successivo. Se deduce che c'è stata una collisione, un modem usa l'algoritmo di attesa binaria esponenziale per ritardare la ritrasmissione del suo mini-slot a uno slot temporale futuro. Se c'è poco traffico sul canale di upstream, il cable modem può trasmettere frame di dati durante i mini-slot temporali nominalmente assegnati ai frame di richiesta di mini-slot, evitando in questo modo di aspettare l'assegnazione di un mini-slot.

Una rete di accesso HFC è quindi un esempio perfetto di protocolli di accesso multiplo in azione: FDM, TDM, accesso casuale e allocazione centralizzata di slot temporali, tutto all'interno della stessa rete!

## 6.4 Reti locali commutate

Dopo aver trattato nel paragrafo precedente le reti broadcast e i protocolli di accesso multiplo, rivolgiamo ora la nostra attenzione alle reti locali commutate (*switched local area network*). La Figura 6.15 mostra una rete locale commutata che connette tre dipartimenti, due server e un router con 4 switch. Gli switch,

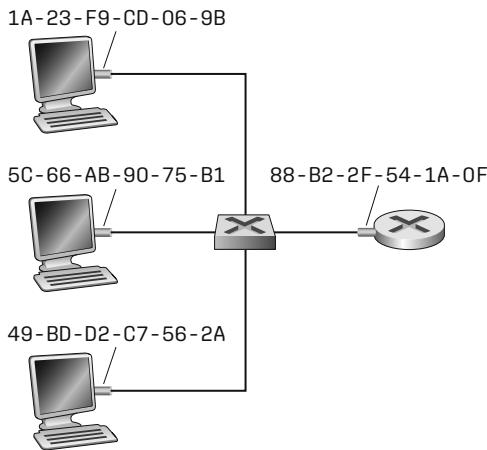


**Figura 6.15** Una rete istituzionale connessa da quattro switch.

operando a livello di collegamento, commutano frame a livello di collegamento e non datagrammi a livello di rete, non leggono gli indirizzi a livello di rete e non usano algoritmi di instradamento quali OSPF per determinare i percorsi attraverso la rete degli switch di livello 2. Al posto degli indirizzi IP, vedremo presto che usano indirizzi a livello di collegamento per inoltrare i frame attraverso la rete. Inizieremo il nostro studio delle LAN commutate trattando il problema dell'indirizzamento a livello di collegamento (Paragrafo 6.4.1). Esamineremo quindi il famoso protocollo Ethernet (Paragrafo 6.4.2). Vedremo quindi come operano gli switch a livello di collegamento (Paragrafo 6.4.3) e infine come tali switch siano spesso utilizzati per costruire LAN su larga scala (Paragrafo 6.4.4).

#### 6.4.1 Indirizzi a livello di collegamento e ARP

Host e router hanno indirizzi a livello di collegamento, il che potrebbe essere sorprendente visto che (Capitolo 4) i nodi hanno anche un indirizzo a livello di rete. Dovreste quindi domandarvi perché abbiamo la necessità di avere indirizzi sia a livello di rete sia a livello di collegamento. Oltre alla descrizione della sintassi e delle funzionalità degli indirizzi del livello di collegamento, in questo paragrafo cerchiamo di chiarire il motivo per cui gli indirizzi di entrambi i livelli sono utili e di fatto indispensabili. Inoltre, analizziamo il protocollo per la risoluzione degli indirizzi (**ARP**, *address resolution protocol*), che fornisce ai nodi un meccanismo per mappare indirizzi IP in indirizzi a livello di collegamento.



**Figura 6.16** Ogni interfaccia di una LAN ha un indirizzo MAC univoco.

### Indirizzi MAC

In realtà, non sono i nodi (cioè, gli host e i router) ad avere indirizzi a livello di collegamento, ma sono le loro interfacce (schede di rete) a possederli. Un host o un router con più interfacce di rete avrà più indirizzi a livello di collegamento associati a esse, esattamente come accade per gli indirizzi IP. Tuttavia è importante notare che gli switch a livello di collegamento non hanno indirizzi a livello di collegamento associati alle loro interfacce che connettono a host e router. Questo avviene perché il compito degli switch è trasportare i datagrammi tra host e router; uno switch effettua questa operazione in modo trasparente, in modo tale che gli host e i router non debbano esplicitamente porre l'indirizzo nei frame per lo switch che interviene. Tutto ciò è mostrato nella Figura 6.16. Gli indirizzi a livello di collegamento sono indicati con varie terminologie che vanno da **indirizzo fisico**, **indirizzo LAN** o **indirizzo MAC**. Poiché quest'ultimo sembra essere il termine più utilizzato, d'ora in avanti ci riferiremo a questi indirizzi come a indirizzi MAC. Per molte LAN (come Ethernet e 802.11), l'indirizzo MAC è lungo sei byte, il che consente di avere  $2^{48}$  possibili indirizzi. Come illustrato nella Figura 6.16, questi indirizzi sono generalmente espressi in notazione esadecimale, scrivendo due cifre esadecimali per ogni byte. Sebbene sia stato progettato per essere permanente è ora possibile cambiare l'indirizzo MAC della scheda di rete via software. Tuttavia, per il resto di questo paragrafo assumeremo che l'indirizzo MAC della scheda di rete sia permanente.

Un'interessante proprietà degli indirizzi MAC è che non esistono due schede di rete con lo stesso indirizzo. Ciò può sembrare sorprendente, dato che le schede di rete sono costruite in differenti paesi da diverse società. Come può un'azienda che costruisce schede a Taiwan essere sicura di usare indirizzi diversi da una che le costruisce in Belgio? La risposta è che la IEEE sovrintende alla gestione degli indirizzi MAC. In particolare, quando una società vuole costruire schede di rete, compra un blocco di spazio di indirizzi, costituito da  $2^{24}$  indirizzi, a un prezzo simbolico. IEEE riserva il blocco di  $2^{24}$  indirizzi, fissando i primi 24 bit dell'indirizzo e lasciando alla società il compito di assegnare a ciascuna scheda di rete una specifica combinazione dei 24 bit rimanenti.

L'indirizzo MAC di una scheda di rete ha una struttura piatta (cioè non gerarchica) e non cambia mai. Un portatile con una scheda Ethernet avrà sempre

lo stesso indirizzo MAC, indipendentemente dal luogo in cui il computer è utilizzato. Allo stesso modo uno smartphone dotato di un'interfaccia wireless 802.11 avrà sempre lo stesso indirizzo MAC. Ricordiamo che, al contrario, gli indirizzi IP hanno una struttura gerarchica (una parte per la rete e una parte per l'host) e che l'indirizzo IP di un nodo deve essere cambiato quando l'host si sposta, cioè quando cambia la rete al quale è collegato. L'indirizzo MAC di una scheda di rete è analogo al numero di codice fiscale di una persona: anch'esso ha una struttura piatta e non varia a seconda del luogo in cui la persona si trasferisce. Gli indirizzi IP invece sono analoghi all'indirizzo postale di una persona. Questi indirizzi hanno una struttura gerarchica e devono essere aggiornati quando la persona cambia residenza. Come a noi è utile possedere sia il codice fiscale sia l'indirizzo postale, così per i nodi è utile avere i due tipi di indirizzi.

Quando una scheda di rete vuole spedire un frame, vi inserisce l'indirizzo MAC di destinazione e lo immette nella LAN. Come vedremo tra breve, uno switch può occasionalmente fare broadcast di un frame in ingresso su tutte le sue interfacce in uscita, per cui una scheda di rete può ricevere un frame non indirizzato a lei. Nel Capitolo 7, disponibile in inglese sulla piattaforma MyLab, vedremo come anche la LAN 802.11 trasmetta frame. Ogni scheda controlla se l'indirizzo MAC di destinazione corrisponde al suo. In caso affermativo, la scheda di rete estrae il datagramma dal frame e lo passa verso l'alto nella pila dei protocolli del nodo cui appartiene. Altrimenti, la scheda di rete non fa altro che scartare il frame. Quindi, soltanto le schede di rete dei nodi destinatari generano degli interrupt ai propri nodi quando ricevono un frame.

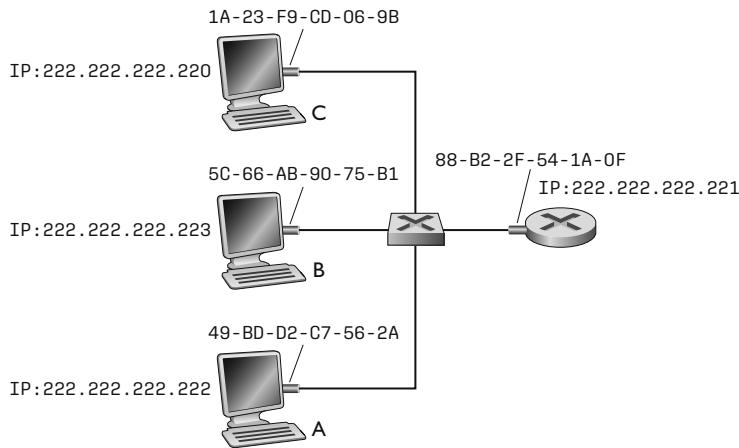
È tuttavia possibile che una scheda di rete voglia proprio che tutte le altre schede di rete ricevano e processino i frame che sta inviando. In questo caso, la scheda di rete inserisce uno speciale **indirizzo MAC broadcast** nel campo dell'indirizzo destinazione del frame. Per le LAN che utilizzano indirizzi a sei byte

#### BOX 6.2

#### TEORIA E PRATICA

##### Mantenere l'indipendenza dei livelli

Sussistono numerose ragioni perché host e interfacce di rete possiedano indirizzi MAC oltre agli indirizzi del livello di rete. In primo luogo, le LAN sono progettate per protocolli arbitrari e non solo per IP e Internet. Se agli adattatori fossero assegnati indirizzi IP, invece che "neutri" indirizzi MAC, questi non sarebbero in grado di supportare facilmente altri protocolli (per esempio, IPX o DECNet). Secondo, se gli adattatori usassero indirizzi del livello di rete, questi dovrebbero essere registrati nella loro RAM e riconfigurati ogni volta che l'adattatore venisse spostato (o riaccesso). Un'altra possibilità è quella di non utilizzare alcun indirizzo negli adattatori e far in modo che ciascuno passi i dati (di solito un datagramma IP) dei frame ricevuti alla pila protocollare del nodo sul quale risiede. Quest'ultimo può allora controllare la concordanza dell'indirizzo del livello di rete. Il problema di questa soluzione è che ogni nodo verrebbe interrotto da tutti i pacchetti inviati sulla LAN, anche da quelli destinati ad altri nodi. In sintesi, per poter assicurare la massima indipendenza dei livelli nell'architettura della rete, molti di essi hanno la necessità di avere propri schemi di indirizzamento. Abbiamo visto finora tre diversi tipi di indirizzi: i nomi degli host per il livello di applicazione, gli indirizzi IP per il livello di rete e gli indirizzi MAC per quello di collegamento.



**Figura 6.17** Ogni nodo di una LAN ha un indirizzo IP e l'adattatore di ciascun nodo ha un indirizzo MAC.

(come le LAN Ethernet e 802.11), l'indirizzo broadcast è una stringa i cui 48 bit sono 1 (cioè, FF-FF-FF-FF-FF FF in notazione esadecimale).

### Protocollo per la risoluzione degli indirizzi (ARP)

Dato che esistono sia indirizzi del livello di rete (come gli indirizzi IP di Internet) sia indirizzi del livello di collegamento (gli indirizzi MAC), si presenta la necessità della loro mappatura. Internet affida questo compito al **protocollo di risoluzione degli indirizzi** (ARP, *address resolution protocol*) [RFC 826].

Per comprendere la necessità di un protocollo come ARP, consideriamo la rete illustrata nella Figura 6.17. In questo semplice esempio, ciascun nodo ha un indirizzo IP e la scheda di ciascun nodo ha un indirizzo MAC. Come al solito, gli indirizzi IP sono rappresentati in notazione decimale puntata e gli indirizzi MAC in notazione esadecimale. Assumeremo in questa trattazione che lo switch invii in broadcast tutti i frame; ogni volta che lo switch riceve un frame su una interfaccia lo inoltra a tutte le altre. Nel paragrafo successivo forniremo una spiegazione più dettagliata di come operano gli switch.

Supponiamo ora che il nodo con l'indirizzo IP 222.222.222.220 voglia inviare un datagramma IP al nodo 222.222.222.222. In questo esempio i nodi sorgente e destinazione appartengono alla stessa sottorete, dal punto di vista dell'indirizzamento (Paragrafo 4.3.3). Per trasmettere un datagramma, il nodo trasmittente deve fornire alla sua scheda di rete non solo il datagramma IP, ma anche l'indirizzo MAC del nodo destinatario 222.222.222.222. Quando riceve il datagramma IP e l'indirizzo MAC, la scheda del nodo trasmittente deve costruire un frame contenente l'indirizzo MAC del nodo di destinazione e immetterlo nella LAN.

Un'importante problematica affrontata in questo paragrafo è come il nodo trasmittente riesca a determinare l'indirizzo MAC del nodo destinazione con indirizzo IP 222.222.222.222. Come potete immaginare, bisogna utilizzare ARP. Un modulo ARP nel nodo sorgente riceve in input un indirizzo IP della stessa LAN e restituisce l'indirizzo MAC corrispondente. Nell'esempio trattato, il nodo trasmittente passa al suo modulo ARP l'indirizzo IP 222.222.222.222 e ARP gli restituisce il corrispondente indirizzo MAC, cioè 49-BD-D2-C7-56-2A.

**Figura 6.18**

Una possibile tabella ARP per il nodo 222.222.222.220.

| Indirizzo IP    | Indirizzo MAC     | TTL      |
|-----------------|-------------------|----------|
| 222.222.222.221 | 88-B2-2F-54-1A-0F | 13:45:00 |
| 222.222.222.223 | 5C-66-AB-90-75-B1 | 13:52:00 |

Abbiamo quindi visto che un modulo ARP traduce un indirizzo IP in un indirizzo MAC. Per molti aspetti è simile al DNS (Paragrafo 2.5), che traduce i nomi degli host in indirizzi IP. Un’importante differenza fra i due traduttori consiste nel fatto che DNS esegue l’operazione per host localizzati in qualunque punto di Internet, mentre ARP risolve soltanto gli indirizzi IP per i nodi nella stessa sottorete. Se un nodo a Torino cerca di utilizzare ARP per risolvere l’indirizzo IP di un nodo di Venezia, ARP segnala un errore.

Ora che abbiamo visto che cosa fa ARP, cerchiamo di capire come funziona. Nella RAM dei nodi vi è una **tabella ARP** (*ARP table*) che contiene la corrispondenza tra indirizzi IP e MAC. La Figura 6.18 mostra come può apparire una tabella ARP nel nodo 222.222.222.220. La tabella contiene anche un valore relativo al **TTL** (*time-to-live*, tempo di vita), che indica quando bisognerà eliminare una data voce dalla tabella. Notiamo che la tabella non contiene necessariamente una voce per ciascun nodo della sottorete; alcuni nodi possono essere stati cancellati (perché scaduto il loro TTL), altri possono non essere mai stati inseriti. Il tipico TTL è di 20 minuti, da quando una voce viene inserita nella tabella ARP.

Supponiamo adesso che il nodo 222.222.222.220 voglia inviare un datagramma a un altro nodo della sottorete. Il nodo trasmittente ha la necessità di ottenere l’indirizzo MAC del nodo di destinazione partendo dall’indirizzo IP di quel nodo. Ciò è semplice se la tabella ARP del nodo trasmittente ha una voce per il nodo di destinazione, ma che cosa accade se la tabella al momento non la possiede? In particolare, supponiamo che il nodo 222.222.222.220 voglia inviare un datagramma al nodo 222.222.222.222. In questo caso, il nodo trasmittente utilizza il protocollo ARP per la conversione dell’indirizzo. Innanzitutto, il nodo trasmittente costruisce uno speciale pacchetto, chiamato **pacchetto ARP**, che ha molti campi, compresi quelli per gli indirizzi IP e MAC di chi spedisce e di chi riceve. I pacchetti ARP di richiesta e di risposta hanno lo stesso formato. Lo scopo di un pacchetto ARP di richiesta è interrogare tutti gli altri nodi della sottorete riguardo all’indirizzo MAC corrispondente all’indirizzo IP da risolvere.

Tornando al nostro esempio, il nodo 222.222.222.220 passa alla scheda di rete un pacchetto di richiesta ARP insieme all’indicazione di inviare il pacchetto all’indirizzo broadcast della rete, cioè FF-FF-FF-FF-FF-FF. La scheda di rete incapsula il pacchetto ARP in un frame, utilizza l’indirizzo broadcast per la destinazione del frame e lo trasmette nella sottorete. Rifacendoci all’analogia col codice fiscale e l’indirizzo postale, possiamo dire che una richiesta ARP è simile a una persona che in mezzo al corridoio dell’ufficio di una società, si mette a gridare: “Qual è il codice fiscale del collega il cui indirizzo postale è: stanza 112, Ditta Rossi, Via Verdi, Milano, Italia?” Il frame contenente la richiesta ARP è ricevuto da tutte le altre schede di rete sulla sottorete. Ciascuna di queste, poiché l’indirizzo è quello broadcast, trasferisce il pacchetto ARP al proprio nodo che controlla se il proprio indirizzo IP corrisponde a quello di destinazione indicato nel pacchetto ARP. L’unico nodo (se esiste) che ha l’indirizzo corrispon-

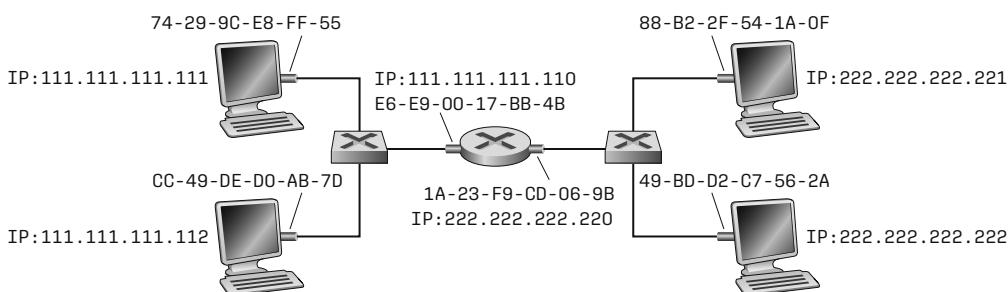
dente invia al nodo richiedente un frame di risposta ARP con la corrispondenza desiderata. Il nodo richiedente 222.222.222.220 può quindi aggiornare la sua tabella ARP e inviare il datagramma IP, incapsulato in un frame, il cui MAC di destinazione è quello del nodo che ha risposto alla richiesta ARP precedente.

Ci sono un paio di aspetti interessanti da notare in merito a questo protocollo. Innanzitutto, il messaggio di richiesta ARP è inviato in un frame broadcast, mentre il messaggio di risposta ARP è inviato in un frame standard. Prima di leggere la spiegazione, provate a pensare al motivo di questa scelta. In secondo luogo, ARP è plug-and-play, nel senso che la tabella ARP di un nodo si costruisce automaticamente e non deve essere configurata dall'amministratore del sistema; inoltre, se un nodo viene scollegato dalla sottorete, prima o poi il suo indirizzo verrà eliminato dalla tabella dei nodi della sottorete.

Viene spesso da chiedersi se ARP sia un protocollo a livello di collegamento o a livello di rete. Come abbiamo visto, un pacchetto ARP è incapsulato in un frame al livello di collegamento e quindi, dal punto di vista dell'architettura, giace sopra il livello di collegamento. Tuttavia, un pacchetto ARP ha campi che contengono gli indirizzi del livello di collegamento e quindi presumibilmente è un protocollo a livello di collegamento, ma contiene anche gli indirizzi di livello di rete e quindi presumibilmente è un protocollo di livello di rete. Alla fine è probabilmente meglio considerare ARP come un protocollo che sta al confine tra i livelli di collegamento e quelli di rete, il che non si adatta bene al semplice stack di protocolli che abbiamo studiato nel Capitolo 1. Questo è parte della complessità dei protocolli nel mondo reale!

### **Come inviare un datagramma a un nodo esterno alla sottorete**

Dovrebbe ora essere chiaro come opera ARP quando un nodo vuole inviare un datagramma a un altro nodo della stessa sottorete, ma diamo ora uno sguardo alla situazione più complicata in cui un nodo voglia inviare un datagramma a un host esterno alla propria sottorete (cioè, attraverso un router). Trattiamo il problema facendo riferimento alla Figura 6.19, che illustra una rete costituita da due sottoreti interconnesse da un router, in cui troviamo molti aspetti interessanti. In primo luogo, vediamo due tipi di nodi: host e router. Ciascun host ha esattamente un indirizzo IP e una scheda di rete. Ma, come abbiamo visto nel Capitolo 4, un router ha un indirizzo IP per ogni interfaccia che, a sua volta, dispone anche di propri moduli ARP (nel router) e di proprie schede di rete. Il router della figura ha due interfacce, e quindi avrà due indirizzi IP, due moduli



**Figura 6.19** Due sottoreti connesse da un router.

ARP e due schede di rete. Naturalmente, ogni scheda di rete ha il proprio indirizzo MAC.

Notiamo anche che la Sottorete 1 ha indirizzo di rete 111.111.111/24 e la Sottorete 2 ha indirizzo 222.222.222/24. Così tutte le interfacce collegate alla Sottorete 1 hanno indirizzo 111.111.111.xxx e tutte quelle collegate alla Sottorete 2 hanno indirizzo 222.222.222.xxx.

Esaminiamo ora come un host della Sottorete 1 possa inviare un datagramma a un host dell'altra sottorete. In particolare, supponiamo che l'host 111.111.111.111 voglia inviare un datagramma IP all'host 222.222.222.222. Come di consueto l'host trasmittente passa il datagramma alla sua scheda di rete, ma deve anche comunicargli un appropriato indirizzo MAC di destinazione. A questo punto, potreste essere tentati di ipotizzare che l'indirizzo MAC sia quello corrispondente alla scheda di rete dell'host 222.222.222.222, cioè 49-BD-D2-C7-56-2A, ma non è così. Se la scheda di rete trasmittente usasse questo indirizzo MAC, nessuna delle schede di rete sulla Sottorete 1 si preoccuperebbe di passare il datagramma IP al suo host, in quanto l'indirizzo di destinazione del frame non corrisponde ad alcuno degli indirizzi MAC delle schede di rete della Sottorete 1 e il datagramma andrebbe perso.

Se osserviamo con attenzione la Figura 6.19 notiamo che, affinché un datagramma possa passare da 111.111.111.111 a un nodo sulla Sottorete 2, deve prima essere inviato all'interfaccia del router 111.111.111.110. Quindi, l'indirizzo MAC appropriato è quello della scheda di rete del router 111.111.111.110, cioè E6-E9-00-17-BB-4B. Ma come può l'host trasmittente acquisire l'indirizzo MAC di 111.111.111.110? Ovviamente, tramite l'utilizzo del protocollo ARP. Dopo aver ottenuto l'indirizzo MAC, la scheda di rete trasmittente crea un frame e lo trasferisce nella Sottorete 1. La scheda di rete del router sulla Sottorete 1 riconosce che il pacchetto è indirizzato a lui e allora lo passa al livello di rete del router. A questo punto, il datagramma IP è stato trasportato con successo dall'host sorgente al router. Il router consulta quindi la propria tabella e individua la scheda cui va inoltrato il datagramma (Capitolo 4), ossia l'interfaccia 222.222.222.220, che a sua volta lo trasferisce alla sua scheda di rete; questa incapsula il datagramma in un nuovo frame e lo spedisce nella Sottorete 2. Adesso, l'indirizzo MAC di destinazione del frame è davvero quello finale, che il router ha ottenuto tramite ARP.

ARP per Ethernet è definito nell'RFC 826. Un'interessante introduzione ad ARP si trova nel tutorial su TCP/IP dell'RFC 1180. Affronteremo ulteriori dettagli su ARP nei problemi al termine del capitolo.

#### 6.4.2 Ethernet

Ethernet ha pressoché conquistato il mercato delle reti cablate. Negli anni '80 e '90 ha affrontato molte sfide da altre tecnologie, come token ring, FDDI e ATM. Alcune di esse sono riuscite a conquistarsi una fetta di mercato per pochi anni. Fin dalla sua ideazione a metà degli anni '70 Ethernet ha continuato a evolversi e a crescere e ha mantenuto la sua posizione dominante. Attualmente, Ethernet è di gran lunga la tecnologia per LAN cablate più diffusa, una situazione che sembra destinata a perdurare anche nel prossimo futuro. Si potrebbe dire che Ethernet è stata per le reti locali quello che Internet è stata per la rete globale.

Molte sono le ragioni alla base del suo successo. Innanzitutto, Ethernet è stata la prima LAN ad alta velocità con vasta diffusione. Quindi, gli amministratori di rete, dopo aver acquisito familiarità con Ethernet, si dimostrarono riluttanti al cambiamento quando apparvero sulla scena altre tecnologie. Secondariamente, token ring, FDDI e ATM sono molto più complesse e costose; aspetti che hanno costituito un ulteriore ostacolo alla loro affermazione. Inoltre, se il motivo più rilevante per passare a un'altra tecnologia LAN (come FDDI o ATM) poteva essere rappresentato da una più elevata velocità trasmissiva, Ethernet ha saputo contrastare la concorrenza, proponendo versioni sempre più veloci. Nei primi anni '90 è anche stata introdotta una tipologia di Ethernet commutata (*switched Ethernet*), con significativo incremento delle velocità trasmissive. Infine, la grande diffusione di Ethernet ha fatto sì che le sue componenti hardware (in particolare, schede di rete e switch) siano facilmente reperibili e particolarmente economiche.

Ideata a metà degli anni '70 da Bob Metcalfe e David Boggs, le prime LAN Ethernet usavano un cavo coassiale come bus per connettere i nodi; questa topologia per Ethernet è durata fino alla metà degli anni '90. Ethernet con topologia a bus è una LAN broadcast, in quanto tutti i frame trasmessi sono elaborati da tutte le schede di rete collegate al bus.

Alla fine degli anni '90, la maggior parte delle aziende e università aveva sostituito le proprie LAN con installazioni Ethernet usando una topologia a stella, basata su hub. In questo tipo di installazioni gli host e i router sono direttamente collegati a un hub tramite doppino intrecciato in rame. Un **hub** è un dispositivo a livello fisico che agisce sui singoli bit, piuttosto che sui frame. Quando un bit, rappresentato da 0 o 1, arriva a un'interfaccia, l'hub semplicemente rigenera il bit, amplifica la sua potenza e lo trasmette su tutte le altre interfacce. Ethernet con topologia a stella basata su hub, quindi, è anch'essa una LAN broadcast, poichéogniqualvolta l'hub riceve un bit da una delle sue interfacce, ne manda una copia a tutte le altre. In particolare, se l'hub riceve dei frame da due diverse interfacce contemporaneamente si verifica una collisione e i nodi che hanno creato i frame devono ritrasmetterli.

All'inizio del 2000, Ethernet ha sperimentato un altro cambiamento rivoluzionario. Le installazioni Ethernet continuavano a usare una topologia a stella, ma l'hub al centro veniva sostituito da uno **switch**. Esamineremo Ethernet commutata in dettaglio più avanti in questo capitolo, per ora accenniamo al fatto che uno switch non solo è "privo di collisioni", ma è anche un vero e proprio commutatore di pacchetti store-and-forward. Tuttavia, a differenza dei router che operano fino a livello 3, gli switch si limitano al livello 2.

### Struttura dei frame Ethernet

Possiamo imparare molto analizzando la struttura del frame Ethernet (Figura 6.20). Per concretezza, consideriamo l'invio di un datagramma IP da un host a un altro, sulla stessa LAN Ethernet (per esempio, quella nella Figura 6.17).



**Figura 6.20** Struttura di un frame Ethernet.

Sebbene il payload del frame Ethernet sia un datagramma IP, notiamo che Ethernet può anche trasportare altri tipi di pacchetti a livello di rete. Supponiamo che la scheda di rete trasmittente, A, abbia indirizzo MAC AA-AA-AA-AA-AA-AA e che il ricevente, B, abbia indirizzo MAC BB-BB-BB-BB-BB-BB. La scheda di rete A incapsula il datagramma IP in un frame Ethernet e lo passa al livello fisico. B ottiene il frame dal livello fisico, estrae il datagramma IP e lo passa al livello di rete. Esaminiamo adesso i sei campi del pacchetto di Ethernet (Figura 6.20).

- *Campo dati (da 46 a 1500 byte)*. Contiene il datagramma IP. Dato che l'**unità massima di trasmissione** (MTU, *maximum transfer unit*) per Ethernet è di 1500 byte, se il datagramma IP supera questo valore, l'host deve frammentare il datagramma (Paragrafo 4.3.2). Altrimenti, se il datagramma IP è inferiore alla dimensione minima del campo dati, equivalente a 46 byte, il campo dovrà essere “riempito” (*stuffed*) fino a raggiungere quel dato valore. Per cui, i dati trasferiti al livello di rete conterranno oltre al datagramma IP anche i byte di riempimento che verranno rimossi utilizzando il campo Lunghezza dell'intestazione del datagramma IP.
- *Indirizzo di destinazione (6 byte)*. Campo che contiene l'indirizzo MAC della scheda di rete di destinazione, vale a dire BB-BB-BB-BB-BB-BB. Quando B riceve un pacchetto Ethernet con tale indirizzo, o con l'indirizzo MAC broadcast, trasferisce il contenuto del campo dati del pacchetto al livello di rete. I pacchetti con altri indirizzi MAC vengono scartati.
- *Indirizzo sorgente (6 byte)*. Campo che include l'indirizzo MAC della scheda che trasmette il pacchetto (AA-AA-AA-AA-AA-AA).
- *Tipo (2 byte)*. Consente a Ethernet di supportare vari protocolli di rete. Per comprendere questo aspetto, occorre tener presente che, oltre a IP, gli host possono supportare vari protocolli di rete e utilizzare diversi protocolli per differenti applicazioni. Per questa ragione, la scheda di rete B deve sapere a quale protocollo di rete passare il contenuto del campo dati di ciascun frame ricevuto. IP e gli altri protocolli di rete (come IPX di Novell o AppleTalk) hanno ciascuno il proprio numero di tipo standardizzato. Inoltre, lo stesso discorso vale per il protocollo ARP (discusso nel paragrafo precedente), se il frame in arrivo ha per questo campo un valore 0806 in esadecimale. Notiamo che il campo tipo è assimilabile al campo protocollo nel datagramma del livello di rete e ai campi numero di porta nel segmento del livello di trasporto; tutti questi campi servono a far comunicare un protocollo di un livello con quello del livello superiore.
- *Controllo a ridondanza ciclica (CRC) (4 byte)*. Come abbiamo studiato nel Paragrafo 6.2.3, questo campo consente alla scheda di rete ricevente di rilevare la presenza di un errore nei bit del frame.
- *Preamble (8 byte)*. I frame Ethernet iniziano con un campo di otto byte: sette hanno i bit 10101010 e l'ultimo è 10101011. I primi sette byte del preamble servono per “risvegliare” le schede di rete dei riceventi e sincronizzare i loro clock con quello del trasmittente. Ciò avviene per una precisa ragione. In effetti, in funzione del tipo di LAN Ethernet, la scheda di rete A cercherà di trasmettere il pacchetto a 10 Mbps, 100 Mbps o 1 Gbps; si verificherà tuttavia sempre una variazione rispetto all'esatto tasso previsto, che

non è conosciuta a priori dalle altre schede di rete sulla LAN. Quindi, il ricevente può utilizzare i primi sette byte del preamble per sincronizzarsi con il clock della scheda di rete di A. Gli ultimi due bit degli otto byte del preamble (i primi due 1 consecutivi) avvisano la scheda di rete B che “le cose importanti” stanno per arrivare.

A livello di rete, tutte le tecnologie Ethernet forniscono un servizio senza connessione, nel senso che quando una scheda di rete vuole inviare un datagramma a un host della rete non fa altro che incapsularlo in un frame Ethernet e immetterlo nella LAN, senza alcuna forma di handshake preventivo con il destinatario. Questo servizio è analogo ai servizi senza connessione dei datagrammi IP a livello 3 e UDP a livello 4.

Tutte le tecnologie Ethernet forniscono un servizio non affidabile al livello di rete. In particolare, quando la scheda di rete B riceve un frame da A, lo sottopone a un controllo CRC, ma non invia un acknowledgement né se il frame supera il controllo né in caso contrario: semplicemente lo scarta. Quindi, A non sa se il frame trasmesso abbia superato il controllo CRC. Questa mancanza di affidabilità nel servizio di trasferimento (a livello di collegamento) aiuta a mantenere Ethernet semplice ed economica, ma significa anche che il flusso dei datagrammi che attraversano il livello di rete può presentare delle lacune.

Come può l'applicazione nell'host B rilevare le lacune dovute ai frame Ethernet scartati? Come abbiamo visto nel Capitolo 3, dipende se l'applicazione sta usando TCP o UDP. In effetti, se l'applicazione utilizza UDP, l'applicazione nell'host B vedrà delle lacune nei dati. Se invece l'applicazione sta usando TCP, l'host B non invia un acknowledgement per i dati contenuti nei frame scartati,

### BOX 6.3

### UN CASO DI STUDIO

#### **Bob Metcalfe ed Ethernet**

Nel periodo in cui era studente della Harvard University, nei primi anni '70, Bob Metcalfe ebbe occasione di lavorare ad ARPAnet al MIT e di conoscere l'attività di Abramson per ALOHA e i protocolli ad accesso casuale. Conseguito il dottorato, prima di essere assunto al Palo Alto Research Center della Xerox (Xerox PARC), Metcalfe frequentò per tre mesi Abramson e i suoi colleghi della University of Hawaii, assistendo personalmente al lavoro su ALOHAnet. Allo Xerox PARC, Metcalfe si occupò inizialmente degli elaboratori Alto, che per molti aspetti possono essere considerati i precursori dei personal computer realizzati negli anni '80, intuendo le potenzialità insite nella possibilità di collegare tra loro questi calcolatori in modo economico. Così, grazie alle sue conoscenze di ARPAnet, ALOHAnet e dei protocolli ad accesso casuale, Metcalfe con il supporto di David Boggs (suo collega presso lo Xerox PARC) ideò Ethernet.

La prima versione di Ethernet operava a 2,94 Mbps e connetteva fino a un massimo di 256 host in un raggio di 1,6 Km, consentendo a molti ricercatori dello Xerox PARC di comunicare tramite i loro calcolatori Alto. In seguito, Metcalfe stabilì un rapporto di collaborazione tra Xerox, Digital e Intel per definire lo standard di Ethernet a 10 Mbps, ratificato dall'IEEE. La Xerox non dimostrava però particolare interesse alla commercializzazione di Ethernet. Per questo, Metcalfe fondò nel 1979 la 3Com, con l'obiettivo di sviluppare le tecnologie per l'interconnessione delle reti, compresa Ethernet. Nei primi anni '80 3Com iniziò la distribuzione delle schede Ethernet per gli allora diffusissimi PC IBM.

forzando il TCP sull'host A a ritrasmetterli. Si noti che quando TCP ritrasmette delle informazioni, queste, prima o poi, ritorneranno alla scheda di rete che le ha scartate. Per cui, in un certo senso, Ethernet effettua una ritrasmissione, anche se non ha idea se si tratta di dati nuovi o già trasmessi.

### Tecnologie Ethernet

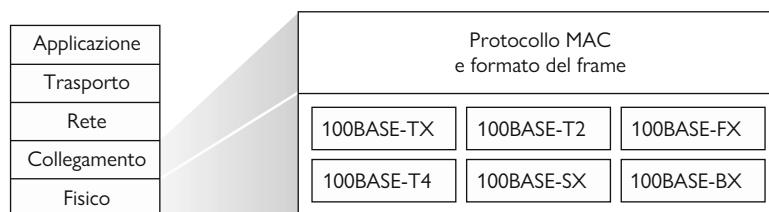
Nella precedente discussione abbiamo fatto riferimento a Ethernet come se si trattasse di un singolo standard di protocollo ma, in effetti, Ethernet compare in molte forme differenti con svariate denominazioni come: 10 BASE-T, 10BASE-2, 100BASE-T, 1000BASE-LX e 10GBASE-T. Queste e molte altre tecnologie Ethernet sono standardizzate dai gruppi di lavoro IEEE 802.3 CSMA/CD (Ethernet) [IEEE 802.3 2020]. Nonostante queste definizioni possano apparire a prima vista oscure, in realtà rispondono a una precisa logica. La prima parte fa infatti riferimento alla velocità dello standard (10, 100, 1000, o 10G), per 10 Mbps, 100 Mbps, 1 Gbps e 10 Gbps e 40 Gbps. “BASE” si riferisce a Ethernet in banda base, cioè che il mezzo fisico trasporta solo traffico Ethernet, ed è usata da quasi tutti gli standard 802.3. La parte finale dell'acronimo rimanda al mezzo fisico; Ethernet è una specifica sia del livello di collegamento sia di quello fisico e viene usata con un'ampia varietà di mezzi fisici, compresi cavi coassiali, fili di rame e fibre ottiche. Tipicamente, “T” si riferisce ai doppini in rame intrecciati.

Storicamente, Ethernet venne inizialmente concepita come un segmento di cavo coassiale. I primi standard 10BASE-2 e 10BASE-5 forniscono le specifiche per Ethernet a 10 Mbps su due tipi di cavi coassiali, ciascuno limitato a una lunghezza di 500 metri. Segmenti più lunghi si possono ottenere usando un **repeater** (ripetitore): un dispositivo a livello fisico che riceve un segnale sul lato di ingresso e lo rigenera sul lato di uscita. Un cavo coassiale corrisponde esattamente alla nostra visione di Ethernet come un mezzo trasmissivo broadcast: tutti i frame trasmessi da un'interfaccia di rete sono ricevuti dalle altre interfacce e il protocollo Ethernet CSMA/CD risolve bene il problema dell'accesso multiplo. I nodi si collegano semplicemente al cavo e si ottiene una rete locale.

Ethernet ha compiuto una serie di passi evolutivi nel corso degli anni e l'odierna Ethernet è molto diversa dal progetto originale con topologia a bus e cavo coassiale. Nella maggior parte delle installazioni odierne i nodi sono collegati a uno switch con segmenti punto a punto, realizzati con doppini in rame o fibre ottiche (Figure da 6.15 a 6.17).

Nella metà degli anni '90, Ethernet fu standardizzata a 100 Mbps, 10 volte più veloce di quella a 10 Mbps. Il protocollo MAC originale e il formato dei frame furono conservati, ma vennero definiti livelli fisici a più alta velocità per i cavi in rame (100BASE-T) e in fibra ottica (100BASE-FX, 100BASE-SX, 100BASE-BX), come riassunto nella Figura 6.21. Ethernet a 100 Mbps è limi-

**Figura 6.21** Standard Ethernet a 100 Mbps: un livello di collegamento in comune, differenti livelli fisici.



tata a 100 metri di distanza su doppino e a parecchi chilometri su fibra, consentendo agli switch Ethernet in edifici diversi di essere connessi tra loro.

Un'estensione dei famosi standard Ethernet a 10 e a 100 Mbps è costituita da Gigabit Ethernet. 40 Gigabit Ethernet è in grado di offrire un tasso trasmissivo lordo di 40000 Mbps e di mantenere la completa compatibilità all'indietro con i molti dispositivi Ethernet già installati. Lo standard Gigabit Ethernet, detto anche IEEE 802.3z, ha le seguenti funzionalità.

- Utilizza il formato del frame standard di Ethernet (Figura 6.20) ed è compatibile con le versioni 10BaseT e 100BaseT. Ciò consente una facile integrazione con le infrastrutture Ethernet già installate.
- Consente sia l'uso di collegamenti punto a punto sia l'uso del canale broadcast condiviso. I collegamenti punto a punto utilizzano switch, mentre i canali broadcast utilizzano hub. Nel linguaggio convenzionale di Gigabit Ethernet, gli hub sono definiti “distributori bufferizzati”.
- Utilizza CSMA/CD per i canali broadcast condivisi. Per riuscire a ottenere un livello accettabile di efficienza è necessario limitare la distanza tra i nodi.
- Su canali punto a punto può operare in modalità full-duplex a 40 Gbps in entrambe le direzioni.

Gigabit Ethernet funzionava inizialmente con la fibra ottica, ma ora è in grado di funzionare con cavi UTP di categoria 5 (per 1000BASE-T e 10GBASE-T).

Terminiamo la nostra discussione su Ethernet sollevando una questione che potrebbe aver già cominciato a turbarvi. Nei giorni in cui le topologie erano a bus o a stella con degli hub Ethernet si trattava chiaramente di un collegamento broadcast (così come definito nel Paragrafo 6.3) in cui le collisioni tra frame si verificavano quando i nodi trasmettevano contemporaneamente. Per gestire queste collisioni, lo standard Ethernet comprendeva il protocollo CSMA/CD, che è particolarmente efficace in LAN broadcast cablate che coprivano un'area limitata. Tuttavia, se l'uso prevalente di Ethernet è oggi con una topologia a stella e switch, usando la commutazione di pacchetto store-and-forward, è ancora realmente necessario avere un protocollo MAC? Come vedremo tra breve, uno switch coordina le proprie trasmissioni e non inoltra mai più di un frame sulla stessa interfaccia. Inoltre, i moderni switch sono full-duplex, in modo che uno switch e un nodo possano scambiarsi frame contemporaneamente senza interferire. In altre parole, in una LAN Ethernet basata su switch non ci sono collisioni e, quindi, non c'è bisogno di un protocollo MAC.

Come abbiamo visto, le odierni Ethernet sono molto differenti da quella originale, concepita da Metcalfe e Boggs oltre trent'anni fa: le velocità sono aumentate di tre ordini di grandezza. I frame Ethernet vengono trasportati da una varietà di mezzi trasmissivi, le Ethernet commutate sono diventate dominanti e ora anche il protocollo MAC è spesso inutile! Nonostante questo, si parla sempre di Ethernet. Inoltre, è interessante notare che, attraverso tutti questi cambiamenti, c'è un elemento che è rimasto invariato per 30 anni: il formato del frame Ethernet. Forse è questa la vera chiave di volta dello standard.

### 6.4.3 Switch a livello di collegamento

Fino a questo punto siamo stati vaghi su che cosa faccia effettivamente lo switch e su come funzioni. Il ruolo dello switch è ricevere i frame in ingresso e inoltrarli

sui collegamenti in uscita (vedremo tra breve, in dettaglio, la funzionalità di inoltro). Lo switch stesso è *trasparente* ai nodi; cioè, un nodo indirizza un frame a un altro nodo, piuttosto che indirizzarlo allo switch e invia il frame nella LAN, senza sapere che uno switch riceverà il frame e lo inoltrerà agli altri nodi. La velocità con la quale i frame giungono a una qualsiasi delle interfacce di uscita degli switch può temporaneamente eccedere la capacità del collegamento di quell’interfaccia. Per risolvere questo problema, le interfacce di uscita dello switch hanno dei buffer, analogamente alle interfacce di uscita dei router per i datagrammi. Vediamo ora da vicino come funzionano gli switch.

### Inoltro e filtraggio

Il **filtraggio** (*filtering*) è la funzionalità dello switch che determina se un frame debba essere inoltrato a una qualche interfaccia o scartato. L’**inoltro** (*forwarding*) consiste nell’individuazione dell’interfaccia verso cui il frame deve essere diretto e, quindi, nell’inviarlo a quell’interfaccia. Le operazioni di filtraggio e inoltro di uno switch sono eseguite mediante una **tavella di commutazione** (*switch table*). Tale tabella di commutazione è composta da voci (*entries*) – per alcuni, ma non necessariamente per tutti, i nodi sulla LAN – che contengono: (1) l’indirizzo MAC del nodo, (2) l’interfaccia dello switch che conduce al nodo, (3) il momento in cui la voce per quel nodo è stata inserita nella tabella. Un esempio di tabella di commutazione per lo switch più in alto nella Figura 6.15, è riportato nella Figura 6.22. La descrizione dell’inoltro del frame può apparire analoga a quella dell’instradamento dei datagrammi (Capitolo 4). Nella trattazione dell’inoltro generalizzato nel Paragrafo 4.4 abbiamo visto che molti switch possono essere utilizzati per effettuare l’inoltro in base all’indirizzo MAC o IP di destinazione. Tuttavia manterremo la distinzione tale per cui gli switch utilizzano gli indirizzi MAC e non quelli IP; inoltre, in un contesto tradizionale non SDN, la costruzione delle tabelle di commutazione è ottenuta con modalità differenti rispetto alle tabelle di inoltro nei router.

Per comprendere come lo switch esegua filtraggio e inoltro ipotizziamo che un frame con indirizzo di destinazione DD-DD-DD-DD-DD-DD giunga allo switch sull’interfaccia  $x$ . Lo switch cerca nella sua tabella l’indirizzo MAC DD-DD-DD-DD-DD-DD. I possibili casi sono tre.

- Non vi è una voce nella tabella per DD-DD-DD-DD-DD-DD; in questo caso lo switch inoltra copie del frame ai buffer di uscita di tutte le interfacce, eccetto  $x$ . In altre parole, se non vi è una voce per l’indirizzo di destinazione, lo switch manda il frame in broadcast.
- Vi è una voce nella tabella che associa DD-DD-DD-DD-DD-DD a  $x$ . In questo caso il frame proviene da un segmento di rete che contiene la scheda di rete DD-DD-DD-DD-DD-DD. Non occorre, quindi, inoltrare il frame a

**Figura 6.22** Parte di una tabella di commutazione dello switch più in alto della Figura 6.15.

| Indirizzo         | Interfaccia | Tempo |
|-------------------|-------------|-------|
| 62-FE-F7-11-89-A3 | 1           | 9:32  |
| 7C-BA-B2-B4-91-10 | 3           | 9:36  |
| ....              | ....        | ....  |

un'altra interfaccia e lo switch esegue la funzione di filtraggio, scartando il frame.

- Vi è una voce nella tabella che associa DD-DD-DD-DD-DD-DD a  $y \neq x$ . In questo caso il frame deve essere inoltrato al segmento di LAN collegato all'interfaccia  $y$ . Lo switch esegue l'inoltro ponendo il frame nel buffer dell'interfaccia  $y$ .

Consideriamo queste regole in relazione alla rete illustrata nella Figura 6.15 e alla tabella della Figura 6.22, supponendo che un frame con indirizzo di destinazione 62-FE-F7-11-89-A3 giunga all'interfaccia 1 dello switch. Dalla tabella esso evince che la destinazione è situata sul segmento collegato all'interfaccia 1, corrispondente alla LAN di Ingegneria elettronica. Di conseguenza, filtra (cioè, scarta) il frame, in quanto è stato già spedito sul segmento di LAN che contiene la destinazione. Ipotizziamo ora che un frame con identico indirizzo di destinazione arrivi all'interfaccia 2. Lo switch esamina ancora la tabella e desume che la destinazione corrisponda all'interfaccia 1. Quindi, inoltra il frame al buffer di uscita dell'interfaccia 1. A questo punto risulta evidente che, disponendo di una tabella completa e accurata, lo switch inoltra i frame a destinazione senza alcun invio broadcast.

In questo senso lo switch è “più intelligente” di un hub. Ma come viene configurata la tabella dello switch all'avvio? Esistono protocolli di instradamento a livello di collegamento, equivalenti a quelli a livello di rete, o l'amministratore di rete deve configurare manualmente le tabelle degli switch?

## Autoapprendimento

Uno switch ha la pregevole proprietà (particolarmente apprezzata dagli amministratori di rete, sempre sovraccarichi di lavoro) di costruire automaticamente, dinamicamente e in modo autonomo le proprie tabelle, ovvero senza l'intervento di un operatore o di un protocollo di configurazione. In altre parole, si potrebbe dire che gli switch *auto-apprendono*. Questa capacità è ottenuta nel seguente modo.

1. La tabella è inizialmente vuota.
2. Di ogni frame che riceve, lo switch archivia nella sua tabella (1) l'indirizzo MAC del campo indirizzo sorgente del frame, (2) l'interfaccia da cui arriva il frame, (3) il momento di arrivo, registrando in tal modo il segmento LAN su cui risiede il nodo trasmittente. Quando tutti i nodi nella LAN avranno inviato un frame, allora la tabella sarà completa.
3. Quando, dopo un dato periodo di tempo detto **aging time** (tempo di invecchiamento), lo switch non riceve frame da un determinato indirizzo sorgente, lo cancella dalla tabella. In questo modo se un calcolatore viene sostituito (da un altro, con una diversa scheda di rete), l'indirizzo MAC del precedente elaboratore viene eliminato automaticamente dalla tabella.

Soffermiamoci ora sulla proprietà di autoapprendimento prendendo in considerazione lo switch più in alto della Figura 6.15 e la corrispondente tabella di commutazione (Figura 6.22). Ipotizziamo che alle 9:39 giunga dall'interfaccia 2 un pacchetto con un indirizzo sorgente 01-12-23-34-45-56, che non è nella tabella; allora lo switch lo registra nella tabella (Figura 6.23). Se il tempo di invecchiamento di questo switch è di 60 minuti e nessun pacchetto con indirizzo

**Figura 6.23** Lo switch apprende la localizzazione di una scheda con indirizzo 01-12-23-34-45-56.

| Indirizzo         | Interfaccia | Tempo |
|-------------------|-------------|-------|
| 01-12-23-34-45-56 | 2           | 9:39  |
| 62-FE-F7-11-89-A3 | 1           | 9:32  |
| 7C-BA-B2-B4-91-10 | 3           | 9:36  |
| ....              | ....        | ....  |

sorgente 62-FE-F7-11-89-A3 arriva fra le 9:32 e le 10:32, alle 10:32 viene rimosso dalla tabella.

Gli switch sono **dispositivi plug-and-play**, in quanto non richiedono interventi dell'amministratore di rete o dell'utente. Basta semplicemente collegare i segmenti di LAN alle sue interfacce, senza dover configurare le tabelle al momento dell'installazione o quando un host è rimosso da un segmento LAN.

### Proprietà della commutazione a livello di collegamento

Dopo aver descritto le operazioni di base degli switch a livello di collegamento, consideriamo ora le loro caratteristiche e proprietà. Possiamo identificare svariati vantaggi relativi all'utilizzo degli switch piuttosto che dei collegamenti broadcast, come i bus o le topologie a stella basate su hub.

- *Eliminazione delle collisioni.* In una LAN costituita da switch e senza hub non vi è spreco di banda a causa delle collisioni. Gli switch mettono i frame nei buffer e non trasmettono più di un frame su ogni segmento di LAN in un certo istante. Come per i router con architettura basata su una rete di interconnessione, il massimo throughput aggregato di uno switch è la somma dei tassi trasmissivi di tutte le sue interfacce. Gli switch, quindi, forniscono un significativo miglioramento delle prestazioni su LAN con collegamenti broadcast.
- *Collegamenti eterogenei.* Dato che uno switch isola un collegamento da un altro, i diversi collegamenti nella LAN possono funzionare a velocità diverse e possono usare mezzi trasmissivi diversi. Lo switch più in alto della Figura 6.15, per esempio, può avere 3 collegamenti in rame da 1 Gbps 1000BASE-T, due collegamenti in fibra da 100 Mbps 100BASE-FX e un collegamento in rame 100BASE-T. Quindi, uno switch è ideale per combinare dispositivi già installati con altri nuovi.
- *Gestione.* Oltre a fornire una maggiore sicurezza uno switch facilita anche la gestione di rete. Per esempio, se una scheda di rete ha un malfunzionamento e manda continuamente frame Ethernet (si parla di *jabbering adapter*, letteralmente “scheda di rete blaterante”), uno switch può individuare il problema e disconnettere internamente la scheda di rete non funzionante. L'amministratore di rete non ha pertanto bisogno di essere sul posto per risolvere il problema. In modo analogo, il taglio di un cavo disconnette solo il nodo che stava usando quel cavo per collegarsi allo switch. Quando veniva usato il cavo coassiale, molti gestori di rete impiegavano ore “seguendo il cavo” (o, più realisticamente, “strisciando sul pavimento”), per trovare il cavo guasto che aveva interrotto la rete. Gli switch raccolgono anche statistiche sull'uso della banda, tasso di collisioni e tipi di traffico, e rendono queste informazioni disponibili ai gestori di rete. Queste informazioni possono essere

**BOX 6.4****FOCUS SULLA SICUREZZA****Intercettazioni in una LAN commutata: avvelenamento dello switch**

Quando un nodo viene connesso a uno switch, esso riceve solamente i frame che gli sono stati esplicitamente inviati. Considerate per esempio la LAN commutata nella Figura 6.17. Quando il nodo A invia un frame al nodo B e vi è una voce per il nodo B nella tabella dello switch, allora lo switch inoltrerà il frame solo al nodo B. Se sul nodo C è in esecuzione un packet sniffer il nodo C non sarà in grado di intercettare il frame da A a B. Quindi, in una LAN commutata, al contrario di una con collegamento broadcast (come le LAN 802.11 e quelle Ethernet basate su hub) è più difficile per un attaccante intercettare i frame. Tuttavia, dato che lo switch invia in broadcast i frame con indirizzo di destinazione che non è nella sua tabella, lo sniffer su C può ancora intercettare qualche frame che non sia esplicitamente indirizzato a C. Inoltre, uno sniffer potrà intercettare tutti i frame Ethernet con indirizzo di destinazione broadcast FF–FF–FF–FF–FF–FF. Un noto attacco contro gli switch, chiamato **avvelenamento dello switch** (*switch poisoning*), consiste nell'inviare grandi quantità di pacchetti allo switch con molti indirizzi MAC fasulli diversi tra loro, affollando così le tabelle degli switch con voci false, senza lasciare spazio per gli indirizzi MAC dei nodi legittimi. Questo obbliga lo switch a inviare in broadcast la maggior parte dei frame, che può essere così intercettata dallo sniffer [Skoudis 2006]. Poiché questo attacco è abbastanza impegnativo anche per un attaccante esperto, gli switch sono significativamente meno vulnerabili degli hub e delle LAN wireless.

usate per rilevare e correggere i problemi e per pianificare come la LAN dovrà evolvere nel futuro. La ricerca sta esplorando come aggiungere nuove funzionalità gestionali alle LAN Ethernet [Casado 2007, Koponen 2011].

**Switch e router a confronto**

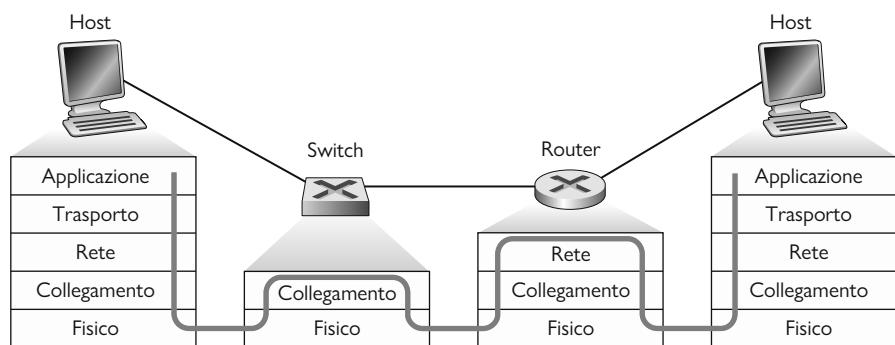
I router sono commutatori di pacchetti store-and-forward (Capitolo 4) che inoltrano pacchetti usando gli indirizzi a livello di rete. Anche se gli switch sono commutatori di pacchetto store-and-forward, si distinguono sostanzialmente dai router, in quanto inoltrano i pacchetti utilizzando indirizzi MAC. Mentre i router sono commutatori di pacchetto di livello 3, gli switch sono commutatori di pacchetto di livello 2. Si ricordi tuttavia che i moderni switch operanti in modalità “match-action” possono essere usati sia per inoltrare frame di livello 2 basandosi sull’indirizzo MAC di destinazione sia datagrammi di livello 3 usando l’indirizzo IP di destinazione. Inoltre gli switch con OpenFlow possono effettuare l’inoltro generalizzato come visto nel Paragrafo 4.4.

Nonostante questo, gli amministratori di rete devono molte volte operare una scelta fra i due quando installano un dispositivo di interconnessione. Così, per la rete della Figura 6.15, un amministratore di rete potrebbe decidere di utilizzare un router piuttosto che uno switch per collegare le LAN dei dipartimenti, i server e il gateway verso Internet, in quanto anche il primo consente di mantenere separate le comunicazioni tra i dipartimenti senza creare collisioni. Quali sono allora i pro e i contro che contraddistinguono i due dispositivi?

Prendiamo per primi in considerazione gli switch, dispositivi plug-and-play che possono avere capacità di filtraggio e inoltro dei pacchetti relativamente alte ma, come mostrato nella Figura 6.24, devono elaborare pacchetti solo fino

**Figura 6.24**

Elaborazione dei pacchetti negli switch, nei router e negli host.



al livello 2, mentre i router devono farlo fino al livello 3. D'altra parte, la topologia di una rete di switch è ristretta a un albero per evitare i cicli con i frame broadcast. Inoltre, una rete di switch molto grande richiederebbe delle grandi tabelle ARP nei nodi e nei router generando un considerevole traffico ARP e richiedendo molta elaborazione. Infine, gli switch non offrono alcuna protezione contro le tempeste di broadcast: se un host iniziasse a trasmettere un flusso ininterrotto di pacchetti broadcast, gli switch li inoltrerebbero, provocando il collasso della rete.

Rivolgiamo adesso la nostra attenzione ai router. Dato che gli indirizzi di rete sono sovente gerarchici (e non lineari come gli indirizzi MAC), generalmente i pacchetti non presentano cicli attraverso i router, anche nel caso in cui la rete presenti percorsi ridondanti. In ogni modo i pacchetti rischiano di percorrere dei cicli soltanto se le tabelle dei router sono configurate male. Però, come abbiamo appreso nel Capitolo 4, IP utilizza uno speciale campo di intestazione del datagramma per limitare la percorrenza dei cicli. Pertanto, i pacchetti non sono vincolati a un albero di copertura e possono utilizzare il miglior percorso fra sorgente e destinazione. Poiché i router non subiscono le restrizioni imposte dalla topologia ad albero, Internet ha potuto essere costruita con una più ricca topologia che ammette, per esempio, collegamenti multipli fra Europa e Nord America. Inoltre, contrariamente agli switch, i router proteggono dalle tempeste di broadcast a livello 2, ma non sono plug-and-play e quindi il loro indirizzo IP e quello degli host a essi collegati deve essere configurato. Inoltre, molte volte presentano un tempo di elaborazione per pacchetto più lungo di quello degli switch, in quanto devono eseguire l'elaborazione fino ai campi del livello 3. Infine, come amenità, ricordiamo che esistono due differenti modi di pronunciare router: “ruuter” o “rauter”. Così molti perdono un sacco di tempo nel tentativo di stabilire quale sia la corretta pronuncia [Perlman 1999].

Dato che switch e router presentano entrambi vantaggi e svantaggi (come riassunto nella Tabella 6.1), quando conviene usare gli uni e quando gli altri? Generalmente, per le reti costituite al massimo da alcune centinaia di host e da pochi segmenti risultano sufficienti gli switch, in quanto localizzano il traffico e incrementano il throughput aggregato senza richiedere la configurazione degli indirizzi IP. Le reti più grandi, costituite da migliaia di host, invece, includono tipicamente oltre agli switch anche dei router, che forniscono un più efficace isolamento del traffico, evitano le tempeste di broadcast, e utilizzano percorsi più funzionali fra gli host della rete.

**Tabella 6.1** Confronto tra le caratteristiche dei più diffusi dispositivi di interconnessione.

|                         | <b>Hub</b> | <b>Router</b> | <b>Switch</b> |
|-------------------------|------------|---------------|---------------|
| Isolamento del traffico | No         | Sì            | Sì            |
| Plug and play           | Sì         | No            | Sì            |
| Instradamento ottimale  | No         | Sì            | No            |

Per una più approfondita trattazione dei vantaggi e svantaggi delle reti facenti uso di switch o router e di come la tecnologia LAN commutata possa essere estesa per ampliare le odierne reti Ethernet di due ordini di grandezza, si vedano [Meyers 2004, Kim 2008].

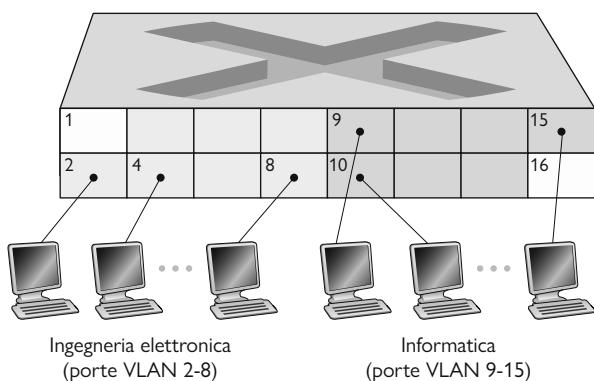
#### 6.4.4 LAN virtuali (VLAN)

Nella precedente discussione riguardante la Figura 6.15 abbiamo visto che le LAN istituzionali moderne sono spesso configurate in modo gerarchico, così che ogni gruppo (dipartimento) abbia la propria LAN commutata connessa alle LAN commutate degli altri gruppi attraverso una gerarchia di switch. Tale configurazione funziona bene in un mondo ideale, ma meno in quello reale. Possiamo identificare tre inconvenienti nella configurazione rappresentata dalla Figura 6.15.

- *Mancanza di isolamento del traffico.* Sebbene la gerarchia localizzi il traffico di un gruppo in un solo switch, il traffico broadcast (per esempio, frame che trasportano messaggi ARP e DHCP o messaggi la cui destinazione non sia ancora stata scoperta da uno switch in grado di auto-apprendere) deve ancora attraversare l'intera rete istituzionale. Le prestazioni della LAN aumenterebbero se si potesse limitare il campo di tale traffico broadcast e forse, ancora più importante, sarebbe limitarlo per ragioni di sicurezza e riservatezza. Per esempio, se un gruppo contiene il consiglio di amministrazione dell'azienda e un altro gruppo contiene impiegati scontenti che usano Wireshark per intercettare pacchetti, il gestore di rete potrebbe preferire che il traffico del consiglio non raggiunga mai gli host di tali impiegati. Questo tipo di isolamento potrebbe essere fornito sostituendo lo switch centrale della Figura 6.15 con un router. Vedremo tra breve che tale isolamento può essere ottenuto anche con un solo switch di livello 2.
- *Uso inefficiente degli switch.* Se invece di 3 gruppi, l'istituzione ne avesse 10, sarebbero necessari 10 switch di primo livello (quelli che raccolgono gruppi di host). Se ogni gruppo fosse piccolo, meno di 10 persone, un singolo switch a 96 porte sarebbe probabilmente sufficiente, ma non fornirebbe isolamento di traffico.
- *Gestione degli utenti.* Se un dipendente si muovesse tra più gruppi, sarebbe necessario cambiare la posatura della rete per connetterlo a un altro switch della Figura 6.15. La presenza di dipendenti che appartengono a più gruppi rende il problema ancora più ostico.

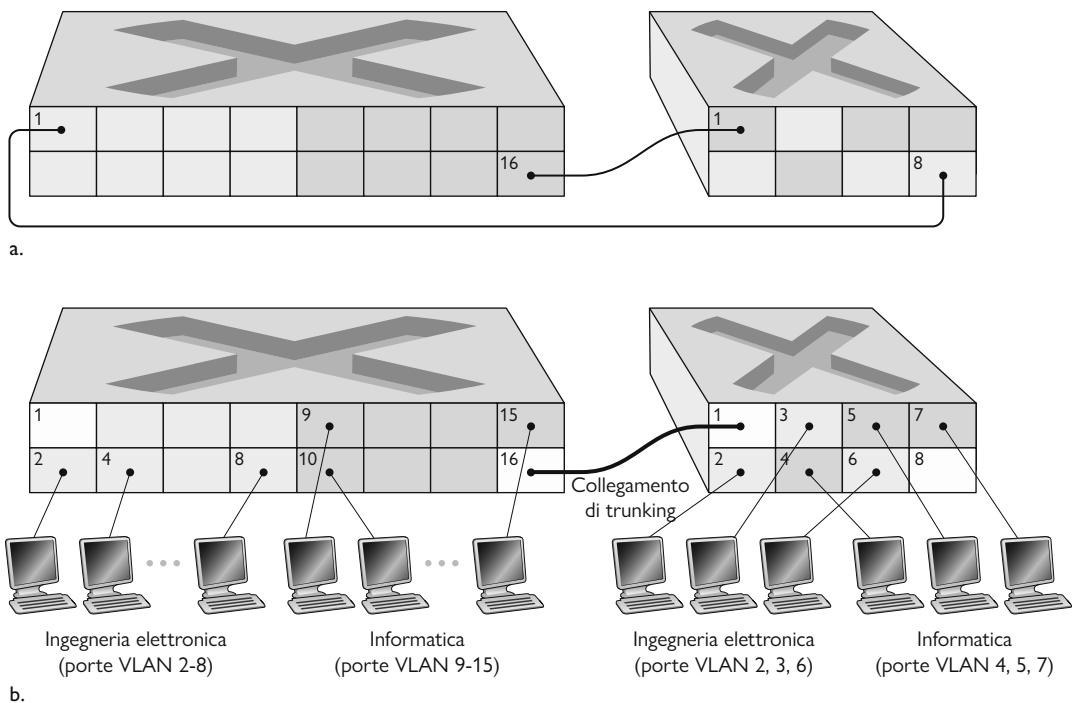
Fortunatamente, queste difficoltà possono essere superate utilizzando uno switch che supporti una **virtual local area network (VLAN o virtual LAN)**. Come suggerisce il nome, tale switch permette di definire più reti locali virtuali

**Figura 6.25** Un singolo switch con due VLAN configurate.



su una singola infrastruttura fisica di rete locale. Gli host all'interno di una VLAN comunicano tra loro come se fossero tutti (e nessun altro) connessi allo switch. In una VLAN basata sulle porte, le porte (interfacce) dello switch vengono divise dal gestore di rete in gruppi, ognuno dei quali costituisce una VLAN le cui porte formano un dominio broadcast (il traffico broadcast proveniente da una porta può raggiungere solo altre porte del gruppo). La Figura 6.25 mostra un singolo switch con 16 porte. Le porte dalla 2 alla 8 appartengono alla VLAN di Ingegneria elettronica, mentre le porte dalla 9 alla 15 appartengono alla VLAN di Informatica (le porte 1 e 16 non sono assegnate). Questa VLAN risolve tutti i problemi discussi precedentemente: i frame delle due VLAN sono isolati tra di loro, i due switch della Figura 6.15 sono stati sostituiti da un singolo switch e se l'utente della porta 8 si aggiunge al dipartimento di informatica, il gestore di rete semplicemente riconfigura il software di VLAN in modo che la porta 8 venga associata alla VLAN di informatica. È facile immaginarsi come lo switch sia configurato e funzioni: il gestore di rete dichiara che una porta appartiene a una determinata VLAN (le porte non dichiarate appartengono a una VLAN di default) utilizzando un software per la gestione dello switch. Una tabella di associazione tra porte e VLAN viene mantenuta all'interno dello switch; l'hardware dello switch si limita a consegnare frame tra porte appartenenti alla stessa VLAN.

Isolando completamente le due VLAN, abbiamo introdotto una nuova difficoltà: come trasmettere il traffico dal dipartimento di Ingegneria elettronica a quello di Informatica? Un modo sarebbe quello di connettere una porta dello switch VLAN (per esempio la porta 1 della Figura 6.25) a un router esterno e configurarla in modo che appartenga a entrambe le VLAN. In questo caso, anche se i due dipartimenti condividono lo stesso switch fisico, dal punto di vista logico la configurazione appare come se i due dipartimenti avessero switch separati connessi da un router. Un datagramma IP che debba transitare dal dipartimento di Ingegneria elettronica a quello di Informatica attraverserebbe prima la VLAN di Ingegneria per raggiungere il router che lo inoltrerebbe all'host destinazione sulla VLAN di Informatica. Fortunatamente, i vendori di switch semplificano la configurazione al gestore di rete fornendo un singolo dispositivo contenente sia uno switch sia un router, in modo che non sia necessario un router esterno separato. Uno dei problemi presentati a fine capitolo esplora con maggior dettaglio questo scenario.

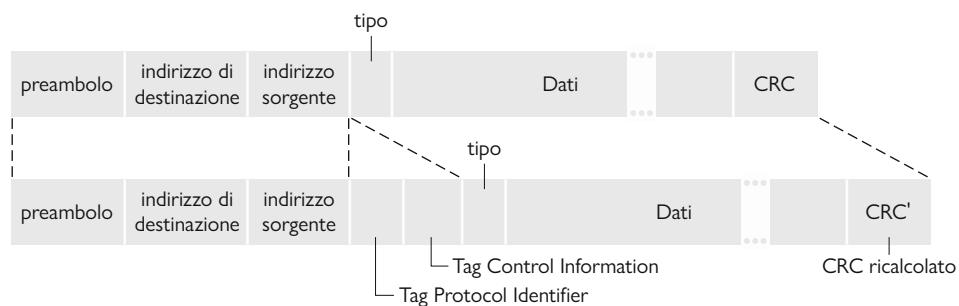


**Figura 6.26** Connessione di due switch abilitati VLAN a due VLAN (a) con due cavi (b) con il trunking.

Torniamo ora alla Figura 6.15 e supponiamo che i dipartimenti siano ospitati in molti edifici separati dove (naturalmente) debba essere fornito accesso alla rete e che (naturalmente) quest'ultimo faccia parte della VLAN del dipartimento. La Figura 6.26 mostra un secondo switch a 8 porte, ognuna definita come appartenente o alla VLAN di Ingegneria elettronica o a quella di Informatica. Ma come dovrebbero interconnettersi questi due switch? Una soluzione semplice consisterebbe nel definire che una porta appartenga alla VLAN di Informatica su ogni switch (e in modo simile per la VLAN di Ingegneria) e nel connettere tra di loro queste porte, come mostrato nella Figura 6.26(a). Tuttavia, tale soluzione non è scalabile, in quanto  $N$  VLAN richiederebbero  $N$  porte per ogni switch, semplicemente per interconnettere i due switch.

Un approccio più scalabile per l'interconnessione tra switch VLAN è noto come **VLAN trunking**. In questo approccio, mostrato nella Figura 6.26(b), una porta speciale per ogni switch (porta 16 sullo switch di sinistra e porta 1 sullo switch di destra) viene configurata come porta di trunking per interconnettere i due switch VLAN. La porta di trunking appartiene a tutte le VLAN e i frame inviati a qualunque VLAN vengono inoltrati attraverso il collegamento di trunking all'altro switch. Questo approccio tuttavia solleva un'altra questione: come fa uno switch a sapere che un frame che arriva a una porta di trunking appartiene a una VLAN particolare? IEEE ha definito un formato esteso di frame Ethernet nello standard 802.1Q, per frame che attraversano un trunk VLAN. Come mostrato della Figura 6.27 il frame 802.1Q consiste nel frame standard Ethernet con aggiunta una **etichetta VLAN** (o *tag VLAN*) di quattro byte nel-

**Figura 6.27** Frame Ethernet originale (in alto), frame Ethernet 802.1Q con etichetta VLAN (in basso).



l'intestazione che trasporta l'identità della VLAN a cui il frame appartiene. L'etichetta VLAN è aggiunta al frame dallo switch sul lato di trasmissione del trunk, mentre viene analizzata e rimossa dallo switch sul lato ricevente. L'etichetta VLAN consiste di un campo TPID (*tag protocol identifier*) di due byte (con un valore fisso esadecimale di 81-00) e un campo *tag control information* di due byte, contenente il campo di identificazione della VLAN a 12 bit e un campo di priorità a 3 bit, simile al campo TOS del datagramma IP.

In questa trattazione ci siamo concentrati sulle VLAN basate sulle porte, ma esse possono essere definite in molti altri modi. Nelle VLAN basate sull'indirizzo MAC, il gestore di rete specifica un insieme di indirizzi MAC che appartengono a ciascuna VLAN; quando un dispositivo viene collegato a una porta, la porta viene associata alla VLAN appropriata sulla base dell'indirizzo MAC del dispositivo. Le VLAN possono anche essere definite sulla base dei protocolli a livello di rete (per esempio, IPv4, IPv6 o AppleTalk) o sulla base di altri criteri. Le VLAN possono essere estese attraverso i router IP in modo che isole di LAN vengano connesse a formare un'unica VLAN che potrebbe estendersi sull'intero globo [YU 2011]. Per maggiori dettagli si veda lo standard 802.1Q [IEEE 802.1q 2005].

## 6.5 Canali virtuali: una rete come livello di collegamento

Dato che questo capitolo tratta i protocolli a livello di collegamento, riflettiamo sull'evoluzione del significato del termine collegamento. Abbiamo iniziato il capitolo considerando il collegamento come un cavo fisico che connette due host. Nello studio dei protocolli di accesso multiplo abbiamo visto che vari host possono essere connessi da un “cavo” condiviso, che può essere uno spettro radio o un altro mezzo. Questo ci induce a considerare il collegamento in maniera un po' più astratta come un canale, piuttosto che un cavo. Nei nostri studi su Ethernet (Figura 6.15) abbiamo visto che il mezzo di interconnessione potrebbe essere una complessa infrastruttura commutata. Attraverso questa evoluzione, gli host hanno comunque mantenuto la visione che il mezzo di interconnessione fosse un semplice canale del livello di collegamento che connettesse due o più host. Abbiamo osservato, per esempio, che un host Ethernet può ignorare se la sua connessione a un altro host sia confinata a un segmento di LAN (Figura 6.15) o si realizzzi tramite una LAN commutata e geograficamente estesa (Figura 6.26).

Nel caso di una connessione via modem in dial-up tra due host, il collegamento che connette gli host è una rete telefonica (una rete di telecomunicazione globale, separata logicamente, con i propri switch, collegamenti e pila dei protocolli per il trasferimento dei dati e per la segnalazione). Tuttavia, dal punto di vista del livello di collegamento di Internet, la connessione dial-up attraverso la rete telefonica è vista come un semplice cavo. In questo senso Internet è una rete telefonica virtuale, che considera la rete telefonica al pari di una tecnologia che fornisce connettività a livello di collegamento tra host Internet. Ricordiamo, dal Capitolo 2, che una rete di overlay vede Internet come un mezzo che fornisce connettività tra due nodi, cercando di usare Internet nello stesso modo in cui Internet usa la rete telefonica.

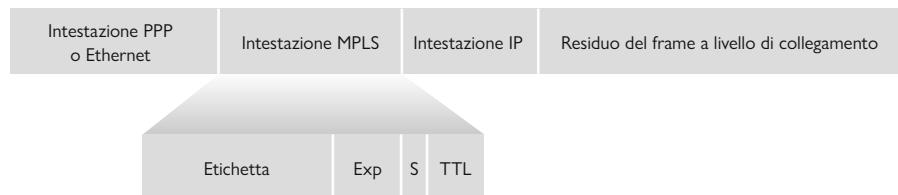
In questo paragrafo considereremo le reti MPLS (*multi-protocol label switching*). A differenza della rete telefonica a commutazione di circuito, MPLS utilizza la commutazione di pacchetto ed è una rete a circuito virtuale. Ha propri formati di pacchetto e propri metodi di invio. Quindi, in una prospettiva didattica, la discussione di MPLS ben si adatta allo studio dei livelli di rete e di collegamento. Dal punto di vista di Internet possiamo vedere MPLS sia come una rete telefonica sia come una rete Ethernet commutata sia al pari di un qualunque collegamento che interconnette dispositivi IP. Quindi, considereremo MPLS nella nostra discussione del livello di collegamento. Anche le reti Frame-Relay e ATM possono essere utilizzate per interconnettere dispositivi IP. Sono tecnologie un po' più “vecchie” (ma ancora impiegate) che non tratteremo qui. Per approfondimenti sull’argomento potete consultare il libro di facile lettura [Goralski 1999]. La nostra analisi di MPLS dovrà essere necessariamente sintetica, in quanto si potrebbe scrivere un intero trattato su queste reti [Davie 2000]. In questa parte ci concentreremo su come queste reti forniscano connessione ai dispositivi IP, sebbene andremo anche nel dettaglio delle tecnologie sottostanti.

### 6.5.1 Multiprotocol Label Switching (MPLS)

MPLS fu sviluppato nella seconda metà degli anni ’90 grazie agli sforzi di numerose aziende al fine di migliorare la velocità di trasferimento dei router IP, adottando il concetto chiave del mondo dei circuiti virtuali: un’etichetta di lunghezza stabilita. L’obiettivo non era l’abbandono dell’infrastruttura basata sulla destinazione dei datagrammi IP in favore di una basata su etichette di lunghezza stabilita e su canali virtuali, ma il miglioramento con datagrammi selettivamente etichettati e, quando possibile, con router che si occupassero dell’inoltro di datagrammi basati su etichette di lunghezza stabilita (invece che sugli indirizzi di destinazione IP). Queste tecniche lavorano a stretto contatto con IP, utilizzando l’indirizzamento e l’instradamento IP. IETF unificò questi sforzi nel protocollo MPLS [RFC 3031, RFC 3032], mischiando efficacemente le tecniche dei circuiti virtuali con le reti che instradano datagrammi.

Iniziamo la nostra analisi di MPLS considerando il formato dei frame, elaborato dai router MPLS. La Figura 6.28 mostra che un pacchetto del livello di collegamento trasmesso tra due dispositivi abilitati MPLS ha una breve intestazione aggiuntiva MPLS tra l’intestazione del livello 2 (Ethernet) e l’intestazione del livello 3 (cioè IP). L’RFC 3032 definisce il formato dell’intestazione per questi tipi di collegamenti; le intestazioni per le reti ATM e le reti a frame-

**Figura 6.28** Intestazione  
MPLS: collocata tra  
l'intestazione di livello  
di collegamento e quella  
di rete.

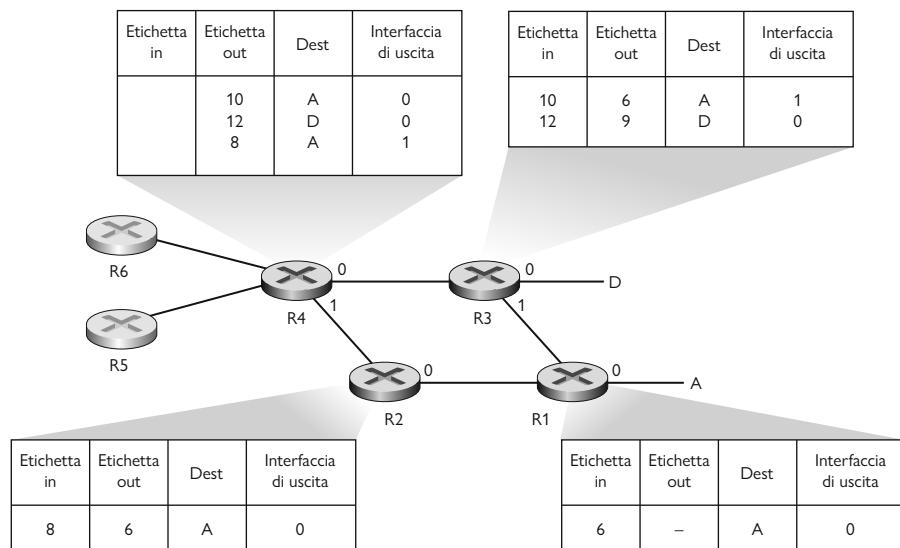


relay sono definite in altri RFC. Tra i campi dell'intestazione MPLS troviamo l'etichetta, 3 bit riservati per utilizzi sperimentali, un bit S, utilizzato per indicare il termine di una serie di intestazioni MPLS impilate (argomento avanzato, che non tratteremo qui) e un campo contenente il TTL.

Risulta immediatamente evidente dalla Figura 6.28 che un frame MPLS può solamente essere inviato tra router che operano su MPLS (dal momento che un router non MPLS potrebbe essere confuso se trova un'intestazione MPLS quando si aspetta un'intestazione IP). Spesso ci si riferisce ai router MPLS come a **router a commutazione di etichetta** (*label-switched router*), perché questi inviano i pacchetti MPLS cercando l'etichetta MPLS nella tabella di inoltro e passando immediatamente il datagramma all'appropriata interfaccia di uscita. Quindi, un router MPLS non ha la necessità di estrarre l'indirizzo di destinazione e di eseguire un controllo del prefisso più lungo nella tabella di inoltro. Ma come può un router sapere se il suo vicino conosce davvero MPLS? E come sa quale etichetta associare all'indirizzo di destinazione IP? Per rispondere a queste domande abbiamo bisogno di analizzare l'interazione tra gruppi di router MPLS.

Nell'esempio della Figura 6.29 i router da R1 fino a R4 sono MPLS, mentre R5 e R6 sono router IP standard. R1 ha avvertito R2 e R3 che può stabilire un percorso verso la destinazione A e che invierà a questa un frame ricevuto con etichetta MPLS 6. Il router R3 ha avvertito R4 che può stabilire un percorso verso A e D e che i frame che arrivano con etichetta MPLS 10 e 12 devono es-

## **Figura 6.29** Inoltro MPLS.



sere inviati a quelle destinazioni. Il router R2 ha anche avvertito il router R4 che può raggiungere A e che un frame ricevuto con etichetta MPLS 8 verrà inoltrato verso quella destinazione. Notiamo che R4 si trova ora in una posizione interessante, disponendo di due percorsi MPLS per raggiungere la destinazione A: attraverso l'interfaccia 0 con etichette MPLS 10 in uscita e attraverso l'interfaccia 1 con etichetta MPLS 8. Lo schema della Figura 6.29 mostra che i dispositivi IP R5, R6, A e D sono connessi tra loro attraverso un'infrastruttura MPLS (router MPLS R1, R2, R3 e R4) nello stesso modo in cui una LAN commutata o una rete ATM può connettere tra loro dispositivi IP. E come le LAN commutate o le reti ATM, i router R1, R2, R3 e R4 operano senza mai esaminare le intestazioni IP dei pacchetti.

Nel precedente esempio non abbiamo specificato il protocollo utilizzato per distribuire etichette tra i router MPLS, in quanto ciò va oltre lo scopo di questo libro. Notiamo, tuttavia, che il gruppo di lavoro IETF ha specificato in [RFC 3468] che il suo principale obiettivo è un'estensione del protocollo RSVP, conosciuto come RSVP-TE [RFC 3209], che i lettori interessati sono incoraggiati a consultare. Non abbiamo discusso né come MPLS effettivamente calcoli i percorsi dei pacchetti tra i router abilitati a MPLS né come raccolga informazioni sullo stato dei collegamenti (per esempio, la capacità di banda dei collegamenti non prenotata da MPLS) da usare nel calcolo dei percorsi. Gli algoritmi di instradamento link state (per esempio, OSPF) sono stati estesi per effettuare il flooding di queste informazioni ai router abilitati MPLS. È interessante notare che gli algoritmi di calcolo dei percorsi non sono standardizzati, ma attualmente sono specifici dei produttori di hardware.

L'attenzione è stata fin qui focalizzata sul fatto che MPLS esegue commutazioni basate sulle etichette senza dover considerare l'indirizzo IP dei pacchetti. Tuttavia, i reali vantaggi di MPLS e la ragione del nostro interesse non risiedono nel potenziale miglioramento della velocità di commutazione, ma piuttosto nella capacità di gestione del traffico. Come accennato, R4 ha due percorsi verso A. Se l'invio viene eseguito a livello IP sulla base degli indirizzi IP, il protocollo di instradamento IP (Capitolo 4) vorrebbe specificare solo il percorso più breve verso A. Pertanto, MPLS consente l'invio di pacchetti lungo percorsi che non potrebbero essere utilizzati con i protocolli di instradamento IP standard. Questa è una semplice forma di **ingegneria del traffico** (*traffic engineering*) [RFC 3346, RFC 3272, RFC 2702, Xiao 2000], in cui l'operatore di rete può ignorare il normale instradamento IP e fare in modo che parte del traffico indirizzato verso una data destinazione sia smistato su differenti percorsi. Ciò può avvenire per ragioni implementative, di prestazioni o per altri motivi.

MPLS risulta utile anche per molti altri scopi. Per esempio, può essere impiegato per implementare una rapida ricostruzione dei percorsi di invio MPLS, cioè per reindirizzare il traffico su un percorso alternativo pre-calcolato in risposta al malfunzionamento di un collegamento [Kar 2000, Huang 2002, RFC 3469]. Infine, notiamo che MPLS può essere utilizzato per implementare le così dette **reti private virtuali** (VPN, *virtual private network*). Nell'impostare le VPN per un cliente, un ISP utilizza la propria rete MPLS per connettere tra loro le reti dei clienti. Viene usato per isolare le risorse e l'indirizzamento delle VPN dei clienti da quelle degli altri utenti che utilizzano la rete del provider. Per ulteriori approfondimenti si veda [DeClercq 2002].

La nostra trattazione di MPLS è stata necessariamente breve. Incoraggiamo quindi il lettore a consultare i riferimenti bibliografici citati. Si noti che MPLS è salito alla ribalta prima dello sviluppo delle SDN studiate nel Capitolo 5 e che molte delle funzionalità di MPLS possono essere ottenute anche tramite SDN e il paradigma di inoltro generalizzato studiato nel Capitolo 4. Solo il futuro rivelerà se MPLS e SDN continueranno a coesistere o se le nuove tecnologie (come SDN) finiranno per sostituire MPLS.

## 6.6 Le reti dei data center

Le aziende il cui business è incentrato su Internet come Google, Microsoft, Amazon e Alibaba hanno costruito enormi data center, ognuno dei quali ospita da decine a centinaia di migliaia di host. I data center non solo sono connessi a Internet, ma hanno anche una loro rete interna complessa, chiamata **data center network**, che interconnette tra loro gli host interni. In questo paragrafo forniamo una breve introduzione alle reti dei data center per applicazioni di cloud computing.

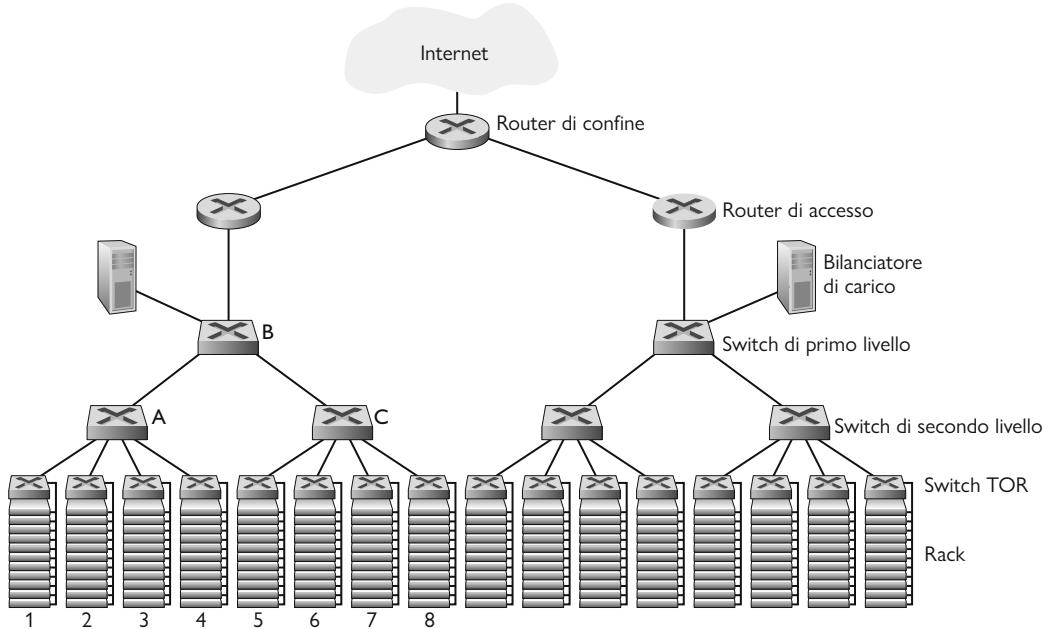
In generale, i data center hanno tre scopi. In primo luogo, forniscono contenuto come pagine web, risultati di ricerca, posta elettronica o video in streaming per gli utenti. In secondo luogo, servono come infrastrutture di calcolo massicciamente parallele per specifiche attività di elaborazione dei dati, come calcoli di indici distribuiti per i motori di ricerca. In terzo luogo, forniscono il cloud computing ad altre società. Oggi le aziende sempre più utilizzano un fornitore di servizi cloud come Amazon Web Services, Microsoft Azure e Alibaba Cloud per gestire essenzialmente tutte le loro esigenze IT.

### 6.6.1 Architetture dei data center

I progetti dei data center sono spesso segreti aziendali salvaguardati scrupolosamente, in quanto forniscono vantaggi competitivi alle principali aziende di cloud computing.

Il costo di un grande data center è molto elevato: supera i 12 milioni di dollari al mese per un data center che ospita 100.000 host [Greenberg 2009a]. Circa il 45% del costo è attribuito agli host, che devono essere sostituiti ogni 3-4 anni; il 25% all'infrastruttura che include trasformatori, gruppi di continuità (UPS), generatori in caso di mancanza di corrente per lungo tempo e sistemi di raffreddamento; il 15% per i costi legati ai consumi energetici e il 15% per la rete compresi gli apparati di rete (switch, router e bilanciatori di carico), i collegamenti esterni e i costi di traffico in transito. I costi per l'equipaggiamento sono ammortizzati in modo che una metrica di costo comune sia applicata sia ai singoli acquisti sia alle spese correnti (come l'energia elettrica). Sebbene la rete non sia il costo maggiore, l'innovazione di rete è la chiave per ridurre i costi complessivi e massimizzare le prestazioni [Greenberg 2009a].

In un data center le api operaie sono gli host. Gli host nei data center sono generalmente calcolatori dotati solo di CPU, memoria, e spazio disco; la forma più tipica con cui si trovano oggi in commercio prende il nome di **blade**. Gli host sono riposti in speciali armadi indicati generalmente con il termine *rack*, ognuno avente tipicamente dalle 20 alle 40 blade. Associato a ogni rack c'è uno switch (che, per abitudine, viene posto in cima al rack), chiamato **top of rack**.



**Figura 6.30** Un data center con topologia gerarchica.

**(TOR) switch**, che interconnette gli host del rack tra di loro e con gli altri switch del data center. Più in dettaglio ogni host nel rack ha una scheda di rete che lo connette al suo switch TOR e ogni switch TOR ha porte addizionali che possono essere connesse ad altri switch. Oggi giorno gli host hanno tipicamente connessioni Ethernet da 40 o 100 Gbps verso i loro switch TOR [FB 2019; Greenberg 2015; Roy 2015; Singh 2015]. Inoltre, a ogni host viene assegnato un indirizzo IP interno al data center.

La rete del data center supporta due tipi di traffico: il traffico tra client esterni e host interni e quello completamente interno. Per gestire i flussi di traffico tra client esterni e host interni la rete del data center ha uno o più **border router** (*router di confine*), che connettono la rete del data center a Internet. Perciò la rete del data center interconnette i rack tra di loro e con i border router. La Figura 6.30 mostra un esempio di una rete di un data center.

Il **data center network design**, l'arte di progettare la rete di interconnessione e i protocolli che connettono i rack tra di loro e con i border router, è diventata negli ultimi anni una parte importante della ricerca sulle reti di calcolatori.

### Bilanciamento del carico

Un data center per cloud computing, come quelli di Google, Microsoft, Amazon e Alibaba fornisce parallelamente molte applicazioni come la ricerca sul Web, l'e-mail e le applicazioni video. Per supportare le richieste da client esterni, a ogni applicazione è associato un indirizzo IP visibile pubblicamente al quale i client inviano le loro richieste e dal quale ricevono le risposte. All'interno del data center le richieste esterne vengono prima dirette a un **load balancer** (*bilanciatore di carico*) il cui compito è di distribuire le richieste agli host, bilanciando il lavoro in funzione del loro carico corrente [Patel 2013; Eisenbud

2016]. Un grande data center ha tipicamente molti load balancer, ognuno dei quali è dedicato a un particolare insieme di applicazioni. Un load balancer di questo tipo è spesso chiamato “switch di livello 4”, perché prende decisioni sulla base del numero di porta di destinazione (livello 4) e dell’indirizzo IP di destinazione del pacchetto. Il load balancer, quando riceve una richiesta per una particolare applicazione, la inoltra a uno degli host che gestisce tale applicazione. Un host può quindi richiedere i servizi di altri host che lo aiutino a processare la richiesta. L’host, quando finisce di elaborare la richiesta, invia la risposta al load balancer che a sua volta la ritrasmette al client esterno. Il load balancer non solo bilancia il carico di lavoro tra gli host, ma fornisce anche una funzionalità simile a quella del NAT, in quanto traduce gli indirizzi esterni pubblici IP negli indirizzi IP interni dell’host appropriato e quindi li ritraduce per i pacchetti che stanno viaggiando nella direzione opposta. Questa procedura fa in modo che i client non contattino direttamente gli host, fornendo così la sicurezza data dall’occultamento della struttura interna della rete e dal fatto che i client non possano interagire direttamente.

### Architettura gerarchica

Per un piccolo data center ospitante solo poche migliaia di host potrebbe essere sufficiente una rete con un border router, un load balancer e poche decine di rack tutti interconnessi da un singolo switch Ethernet. Tuttavia, per estendersi fino a decine e centinaia di migliaia di host, un data center deve spesso impiegare una **gerarchia di router e switch** come quella della topologia mostrata della Figura 6.30. In cima alla gerarchia il border router si connette ai router di accesso (nella Figura 6.30 ne sono mostrati solo 2, ma potrebbero essere molti di più). Al di sotto di ogni router di accesso ci sono tre livelli (*tiers*) di switch. Ogni router di accesso si connette a uno switch di primo livello (tier-1 switch) che a sua volta si connette a più switch di secondo livello (tier-2 switch) e a un load balancer. Gli switch di secondo livello a loro volta si connettono a più rack tramite gli switch TOR (switch di terzo livello). Tipicamente, tutti i collegamenti usano Ethernet per i protocolli a livello di collegamento e fisico, con cavi in fibra e rame. Con questa progettazione gerarchica è possibile che un data center possa estendersi fino a centinaia di migliaia di host.

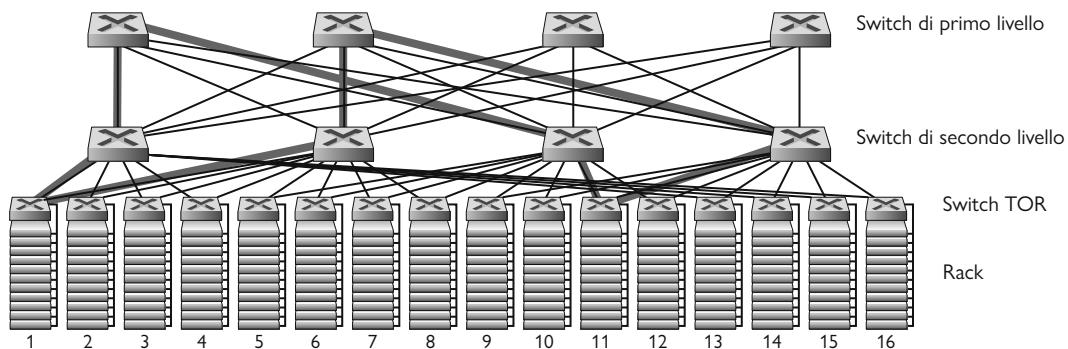
Dato che per un fornitore di applicazioni di cloud computing è critico offrire il servizio in modo continuo e con un alto tasso di disponibilità, i data center hanno collegamenti e apparati di rete ridondanti, non mostrati della Figura 6.30. Per esempio, ogni switch TOR può connettersi a due switch di livello 2 e ogni router di accesso, switch di livello 1 e switch di livello 2, può essere duplicato e integrato nella progettazione [Cisco 2012; Greenberg 2009b]. Nel progetto gerarchico mostrato nella Figura 6.30, gli host sotto i router di accesso formano una singola sottorete. Ognuna di queste sottoreti è ulteriormente partizionata in sottoreti VLAN più piccole, ognuna di poche centinaia di host, per localizzare il traffico broadcast di ARP [Greenberg 2009a].

L’architettura gerarchica convenzionale appena descritta, sebbene risolva il problema della scalabilità, soffre di una limitata capacità di comunicazione da host a host [Greenber 2009b]. Per capire tale limitazione si consideri di nuovo la Figura 6.30 e si supponga che ogni host si connetta al proprio switch TOR con un collegamento da 10 Gbps, mentre i collegamenti tra switch siano Ethernet a 100 Gbps. Due host nello stesso rack sfruttano sempre per comunicare tutta la

banda di 10 Gbps, in quanto sono limitati solo dalla velocità delle loro schede di rete. Tuttavia, se ci sono simultaneamente nella rete del data center molti flussi di traffico, il tasso trasmissivo tra due host in rack differenti può essere molto minore. Si consideri per esempio un traffico consistente di 40 flussi simultanei tra 40 coppie di host in rack diversi. Più specificatamente si supponga che ognuno dei 10 host nel rack 1 della Figura 6.30 invii un flusso a 1 host corrispondente nel rack 5. In modo simile vi sono 10 flussi simultanei tra coppie di host nei rack 2 e 6, 10 flussi simultanei tra i rack 3 e 7 e altrettanti tra i rack 4 e 8. Se ogni flusso condivide in modo paritario la capacità del collegamento con gli altri flussi, i 40 flussi che attraversano il collegamento da 100 Gbps tra A e B, così come quello tra B e C, riceveranno solo  $100 \text{ Gbps} / 40 = 2,5 \text{ Gbps}$ , che è significativamente minore della velocità della scheda di rete da 10 Gbps. Il problema diventa ancora più serio per i flussi tra host che devono attraversare la gerarchia verso l'alto.

Diverse possibili soluzioni a questa problematica sono le seguenti.

- Una possibile soluzione è quella di utilizzare switch e router a più alta velocità, ma questi sono estremamente costosi e aumenterebbero in modo significativo il costo del data center.
- Una seconda soluzione a questo problema, adattabile ogni volta sia possibile, consiste nel collocare servizi e dati correlati il più vicino possibile (per esempio, nello stesso rack o in un rack vicino) [Roy 2015; Singh 2015] per ridurre al minimo la comunicazione tra rack tramite switch di livello 2 e 1. Tuttavia con questa soluzione non si va molto lontano, perché il requisito chiave nei data center è la flessibilità nel posizionamento della capacità di calcolo e dei servizi [Greenberg 2009b; Farrington 2010]. Per esempio, un motore di ricerca Internet su larga scala può essere in esecuzione su migliaia di host distribuiti su più rack con requisiti di larghezza di banda significativi tra tutte le coppie di host. Allo stesso modo, un servizio di cloud computing (quali Amazon Web Services o Microsoft Azure) potrebbe voler posizionare le macchine virtuali che compongono il servizio sugli host con la capacità maggiore, indipendentemente dalla loro posizione nel data center. Se questi host fisici sono distribuiti in più rack, i colli di bottiglia della rete descritti precedentemente possono rendere le prestazioni insoddisfacenti.
- Una terza soluzione consiste nel fornire sempre più connettività tra gli switch TOR e gli switch di livello 2 e tra gli switch di livello 2 e quelli di livello 1. Per esempio, come mostrato nella Figura 6.31, ogni switch TOR si può connettere a due switch di secondo livello che a loro volta forniscono percorsi disgiunti e multipli tra i rack. Nella Figura 6.31 esistono 4 percorsi distinti tra il primo e il secondo switch di secondo livello, che insieme forniscono una capacità aggregata di 400 Gbps tra i primi due switch di livello 2. Tale progettazione non solo supera la limitazione di capacità trasmissiva da host a host, ma crea anche un ambiente più flessibile in cui la comunicazione tra coppie di rack non connessi allo stesso switch TOR è logicamente equivalente, indipendentemente dalla loro posizione nel data center. Aumentare il grado di connettività tra gli switch porta due vantaggi significativi: una maggiore capacità e una maggiore affidabilità (a causa della diversità del percorso) tra gli switch. Nei data center di Facebook [FB 2014; FB 2019] ogni TOR è collegato a quattro diversi switch di livello 2, ognuno collegato a quattro diversi switch di livello 1.



**Figura 6.31** Topologia di data center altamente interconnessa.

Una conseguenza diretta della maggiore connettività tra i livelli gerarchici (*tiers*) nella rete dei data center è che l'instradamento multi-percorso può diventare centrale in queste reti. I flussi sono a percorsi multipli (*multipath*) per impostazione predefinita. Uno schema molto semplice per realizzare il *routing multipath* è Equal Cost Multi Path (ECMP) [RFC 2992], che esegue una selezione casuale del nodo successivo lungo gli switch tra sorgente e destinazione. Sono stati proposti anche schemi avanzati che utilizzano un bilanciamento del carico più dettagliato [Alizadeh 2014; Noormohammadpour 2018]. Mentre questi schemi eseguono l'instradamento multi-percorso a livello di flusso, esistono anche progetti che instradano individualmente i pacchetti all'interno di un flusso tra più percorsi [He 2015; Raiciu 2010].

### 6.6.2 Tendenze nel networking dei data center

Il networking dei data center si sta evolvendo rapidamente e le nuove tendenze sono guidate da fattori quali riduzione dei costi, virtualizzazione, vincoli fisici, modularità e personalizzazione.

#### Riduzione dei costi

Al fine di ridurre il costo dei data center e allo stesso tempo migliorarne le prestazioni sul ritardo e sul throughput, nonché la facilità di espansione e distribuzione, i giganti del cloud implementano continuamente nuovi progetti di rete per data center. Sebbene alcuni di questi design siano proprietari, altri [FB 2019] sono esplicitamente open o descritti nella letteratura [Greenberg 2009b; Singh 2015]. Si possono identificare alcune importanti tendenze.

La Figura 6.31 illustra una delle tendenze principali nel networking dei data center: l'emergere di una rete gerarchica a più livelli che interconnette gli host del data center. Questa gerarchia ha concettualmente lo stesso scopo di una singola grande matrice di commutazione (*crossbar switch*) vista nel Paragrafo 4.2.2, che consente a qualsiasi host di un data center di comunicare con qualsiasi altro host. Ma come abbiamo visto, questa rete interconnessa a più livelli presenta molti vantaggi rispetto a una struttura di commutazione, tra cui i percorsi multipli tra sorgente e destinazione, la maggiore capacità (dovuta al routing a percorsi multipli) e l'affidabilità (dovuta ai percorsi multipli e con collegamenti disgiunti tra due host qualsiasi).

La rete di interconnessione del data center è composta da un gran numero di switch di piccole dimensioni. Per esempio, nel data center Jupiter di Google, una configurazione ha 48 collegamenti tra lo switch TOR e i suoi server e connessioni fino a 8 switch di livello 2; uno switch di livello 2 ha collegamenti a 256 switch TOR e collega fino a 16 switch di livello 1 [Singh 2015]. Nell'architettura del data center di Facebook, ogni switch TOR collega fino a quattro diversi switch di livello 2 (ciascuno in un diverso “piano di interconnessione” detto *spline*) e ogni switch di livello 2 si collega fino a 4 dei 48 switch di livello 1 nel piano spline; i piani spline sono quattro. Gli switch di livello 1 e 2 si collegano a un numero maggiore e più scalabile di switch di livello 2 o TOR, rispettivamente [FB 2019]. Alcuni dei principali operatori di data center, piuttosto che acquistare direttamente gli switch, li costruiscono internamente, acquisendo i componenti da produttori di microchip pronti all’uso (*merchant silicon*) [Greenberg 2009b; Roy 2015; Singh 2015].

Una rete di interconnessione multilivello (*tiered*) come quella rappresentata nella Figura 6.31 e come quelle implementate nelle architetture dei data center discusse precedentemente è nota come **rete Clos**, dal nome di Charles Clos, che ha studiato tali reti [Clos 1953] nel contesto della commutazione telefonica. Da allora è stata sviluppata una ricca teoria delle reti Clos, che ha trovato un uso aggiuntivo nel networking dei data center e nelle reti di interconnessione a multiprocessore.

### **Controllo e gestione SDN centralizzati**

Poiché un data center è gestito da una singola organizzazione è naturale che alcuni dei maggiori operatori di data center, tra cui Google, Microsoft e Facebook, stiano adottando l’approccio basato sul concetto di controllo logicamente centralizzato simile a SDN. Le loro architetture riflettono anche una netta separazione tra il piano dati (composto da switch relativamente semplici) e il piano di controllo basato su software, come abbiamo visto nel Paragrafo 5.5. A causa dell’immensa scala dei loro data center, risulta cruciale avere sia la configurazione che la gestione operativa automatizzate, come abbiamo visto nel Paragrafo 5.7.

### **Virtualizzazione**

La virtualizzazione ha rappresentato una forza trainante per gran parte della crescita del cloud computing e più in generale delle reti dei data center. Le macchine virtuali (VM) disaccoppiano le applicazioni software in esecuzione dall’hardware fisico. Questo disaccoppiamento consente anche la migrazione delle VM tra server fisici che potrebbero trovarsi su diversi rack. I protocolli Ethernet e IP standard presentano limitazioni nell’abilitazione dello spostamento delle VM mantenendo le connessioni di rete attive tra i server. Poiché tutte le reti dei data center sono gestite da un’unica autorità amministrativa, una soluzione elegante al problema è trattare l’intera rete del data center come un’unica rete di livello 2. Si ricordi che in una tipica rete Ethernet il protocollo ARP mantiene l’associazione tra l’indirizzo IP e l’indirizzo hardware (MAC).

Per emulare l’effetto della connessione di tutti gli host a un “singolo” switch il meccanismo ARP viene modificato per utilizzare un sistema di query in stile DNS invece di una trasmissione broadcast; la directory mantiene una mappatura dell’indirizzo IP assegnato a una VM e allo switch fisico a cui la VM è at-

tualmente connessa nella rete del data center. Alcuni schemi scalabili che implementano questo progetto di base sono proposti in [Mysore 2009; Greenberg 2009b] e implementati con successo nei moderni data center.

### Vincoli fisici

A differenza della WAN Internet, le reti dei data center operano in ambienti che non solo hanno capacità molto elevata (i collegamenti a 40 Gbps e 100 Gbps sono ormai comuni), ma hanno anche ritardi estremamente bassi (microsecondi). Di conseguenza, le dimensioni del buffer sono piccole e i protocolli di controllo della congestione come TCP e le sue varianti non scalano bene nei data center. Nei data center i protocolli di controllo della congestione devono reagire rapidamente e operare in regimi di perdita estremamente bassa, poiché il recupero delle perdite e i timeout possono portare a una estrema inefficienza. Sono stati proposti e implementati diversi approcci per affrontare questo problema, che vanno dalle varianti TCP specifiche dei data center [Alizadeh 2010] all'implementazione di tecnologie Remote Direct Memory Access (RDMA) su Ethernet standard [Zhu 2015; Moshref 2016; Guo 2016]. È stata anche applicata la teoria della pianificazione per sviluppare meccanismi che disaccoppiano la trasmissione dei flussi di pacchetti dal controllo della loro velocità, consentendo protocolli di controllo della congestione molto semplici pur mantenendo un elevato utilizzo dei collegamenti [Alizadeh 2013; Hong 2012].

### Modularità e personalizzazione hardware

Un'altra importante tendenza è l'impiego di data center modulari (MDC, *shipping container-based modular data center*) [YouTube 2009; Waldrop 2007]. In una MDC una fabbrica costruisce, all'interno di un container standard di 12 metri (uno “shipping container”, appunto), un mini data center e trasporta il container sul luogo dove è posto il data center. Ogni container ospita fino ad alcune migliaia di host, impilati in decine di rack, in una disposizione che non lascia spazi vuoti. Nel data center più container vengono interconnessi tra loro e a Internet. Il container prefabbricato, una volta installato nel data center, è spesso difficile da manutenere. Quindi ogni container è progettato per una tenue degradazione delle prestazioni: quando le componenti (server e switch) si rompono nel tempo, il container continua a funzionare, ma con prestazioni degradate. Quando molte componenti si sono rotte e le prestazioni sono scese sotto una determinata soglia, l'intero container viene rimosso e sostituito con uno nuovo.

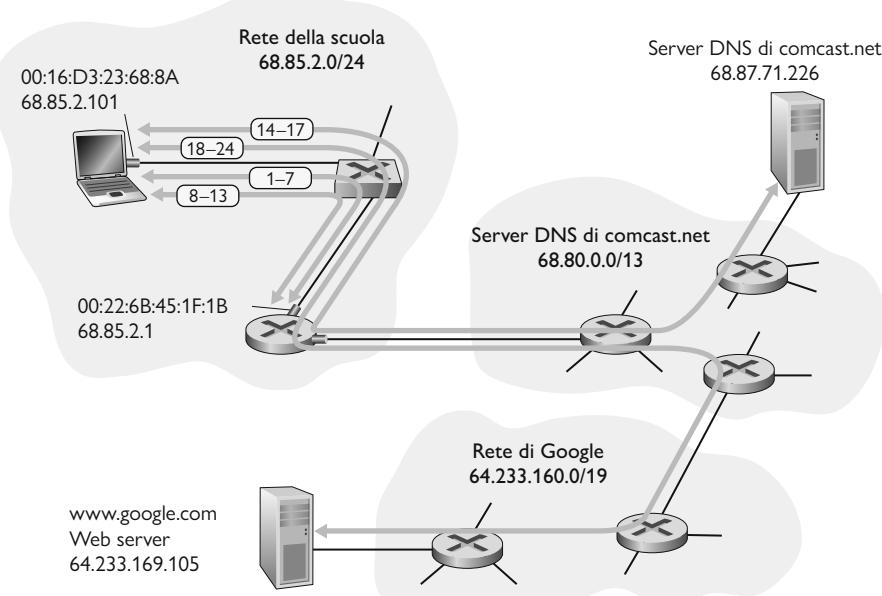
La costruzione di un data center tramite container crea nuove sfide di networking. All'interno di un MDC ci sono due tipi di rete: le reti interne al container e quelle che connettono tra loro i container [Guo 2009, Farrington 2010]. All'interno di ogni container, dove ci sono poche migliaia di host, è possibile costruire una rete completamente connessa dalla capacità di gigabit al secondo con degli switch Ethernet poco costosi. Tuttavia la progettazione di una rete che connetta centinaia o migliaia di container fornendo una grande larghezza di banda da host a host rimane una grande sfida. In [Farrington 2010] viene proposta un'architettura ibrida di switch elettrici/ottici per interconnettere i container.

I grandi fornitori di servizi cloud costruiscono o personalizzano sempre più quasi tutto ciò che si trova nei loro data center: adattatori, router, switch, TOR, software e protocolli di rete [Greenberg 2015, Singh 2015]. Un'altra tendenza,

lanciata da Amazon, è quella di migliorare l'affidabilità con *availability zones*, che replicano sostanzialmente data center distinti in più edifici vicini. Avendo gli edifici nelle vicinanze (a pochi chilometri di distanza), i dati transazionali possono essere sincronizzati attraverso le availability zones, fornendo al contempo la tolleranza ai guasti [Amazon 2014].

## 6.7 Retrospettiva: cronaca di una richiesta di una pagina web

Ora che abbiamo trattato il livello di collegamento in questo capitolo e quelli di rete, trasporto e applicazione nei capitoli precedenti, il nostro viaggio attraverso la pila di protocolli è completato! All'inizio di questo libro, nel Paragrafo 1.1, è stato scritto: "La maggior parte di questo libro riguarda i protocolli di rete" – e questo è sicuramente confermato per quanto riguarda i primi cinque capitoli! Prima di passare ai capitoli su argomenti particolari nella seconda parte del libro (i Capitoli 7 e 8 sono disponibili in inglese sulla piattaforma MyLab) vorremmo ricapitolare il nostro viaggio nella pila dei protocolli considerando una visione integrata e totale di quello che abbiamo imparato finora. Un modo per ottenere questa visione globale è identificare i tanti (tanti davvero!) protocolli coinvolti quando si soddisfa anche la più semplice delle richieste: scaricare una pagina web. La Figura 6.32 mostra il nostro esempio: uno studente, Bob, connette il proprio laptop a uno switch Ethernet della sua scuola e scarica una pagina web, per esempio, la home page di [www.google.com](http://www.google.com). Come già sappiamo, questa richiesta apparentemente semplice sottintende moltissime cose. Un laboratorio Wireshark alla fine del capitolo esamina i file delle tracce dei pacchetti coinvolti in questo esempio.



**Figura 6.32** Cronaca di una richiesta di una pagina web: impostazioni di rete e azioni.

### 6.7.1 Si comincia: DHCP, UDP, IP ed Ethernet

Supponiamo che Bob accenda il suo laptop e lo colleghi a un cavo Ethernet connesso allo switch Ethernet della scuola che a sua volta è connesso al router della scuola, come mostrato nella Figura 6.32. Il router della scuola è connesso a un ISP, in questo esempio Comcast.net, che fornisce il servizio DNS alla scuola. Quindi il server DNS risiede nella rete Comcast piuttosto che nella rete della scuola. Assumiamo che il server DHCP sia eseguito nel router, come avviene spesso.

Quando Bob connette il suo laptop alla rete, non può fare niente, nemmeno scaricare una pagina web, senza un indirizzo IP. Quindi la prima azione riguardante la rete intrapresa dal laptop di Bob è quella di eseguire il protocollo DHCP per ottenere un indirizzo IP, insieme ad altre informazioni, dal server DHCP locale.

1. Il sistema operativo del laptop di Bob crea un **messaggio DHCP request** (Paragrafo 4.3.3) e inserisce questo messaggio in un **segmento UDP** (Paragrafo 3.3) con porta di destinazione 67 (server DHCP) e porta sorgente 68 (client DHCP). Il segmento UDP viene quindi inserito all'interno di un **datagramma IP** (Paragrafo 4.3.1) con indirizzo IP di destinazione broadcast (255.255.255.255) e indirizzo IP sorgente 0.0.0.0, in quanto il laptop di Bob non ha ancora un indirizzo IP.
2. Il datagramma IP contenente il messaggio di richiesta DHCP è quindi posto in un **frame Ethernet** (Paragrafo 6.4.2). Il frame Ethernet ha indirizzo MAC di destinazione FF:FF:FF:FF:FF:FF in modo che il frame venga inviato in broadcast a tutti i dispositivi connessi allo switch, tra i quali si spera ci sia un server DHCP; l'indirizzo MAC della sorgente del frame è quello del laptop di Bob, 00:16:D3:23:68:8A.
3. Il frame Ethernet broadcast contenente la richiesta DHCP è il primo frame inviato dal laptop di Bob allo switch Ethernet. Lo switch invia in broadcast il frame in entrata a tutte le porte in uscita, compresa la porta che lo connette al router.
4. Il router riceve il frame broadcast Ethernet che contiene la richiesta DHCP sulla sua interfaccia con indirizzo MAC 00:22:6B:45:1F:1B e il datagramma IP è estratto dal frame Ethernet. L'indirizzo IP di destinazione del datagramma broadcast indica che tale datagramma IP dovrebbe essere elaborato dai protocolli di livello superiore su questo nodo; quindi sul payload del datagramma (un segmento UDP) viene effettuato un **demultiplexing** (Paragrafo 3.2) verso UDP e il messaggio di richiesta DHCP viene estratto dal segmento UDP. Ora il server DHCP ha il messaggio di richiesta DHCP.
5. Supponiamo ora che il server DHCP in esecuzione sul router possa allocare indirizzi IP nel blocco **CIDR** (Paragrafo 4.3.3) 68.85.2.0/24. In questo esempio tutti gli indirizzi IP usati all'interno della scuola sono contenuti nel blocco di indirizzi di Comcast. Supponiamo che il server DHCP allochi l'indirizzo 68.85.2.101 al laptop di Bob. Il server DHCP crea un **messaggio DHCP ACK** (Paragrafo 4.3.3) contenente tale indirizzo IP, insieme all'indirizzo IP del server DNS (68.87.71.226), l'indirizzo IP del gateway di default (68.85.2.1) e il blocco della sottorete (68.85.2.0/24 o equivalentemente la maschera di rete). Il messaggio DHCP è posto all'interno di un segmento UDP inserito in un

datagramma IP a sua volta posto in un frame Ethernet. Il frame Ethernet ha come indirizzo MAC sorgente quello dell’interfaccia del router sulla rete domestica (00:22:6B:45:1F:1B) e come indirizzo MAC di destinazione quello del laptop di Bob (00:16:D3:23:68:8A).

6. Il frame Ethernet contenente il messaggio DHCP ACK è inviato (in unicast) dal router allo switch. Lo switch, poiché è in grado di *auto-apprendere* (Paragrafo 6.4.3) e ha precedentemente ricevuto dal laptop di Bob un frame Ethernet contenente la richiesta DHCP, sa di dover inoltrare un frame indirizzato a 00:16:D3:23:68:8A solo alla porta di uscita verso il laptop di Bob.
7. Il laptop di Bob riceve il frame Ethernet contenente il messaggio DHCP ACK, estrae dal frame Ethernet il datagramma IP, estrae dal datagramma IP il segmento UDP ed estrae dal segmento UDP il messaggio DHCP ACK. Il client DHCP di Bob memorizza il suo indirizzo IP e quello del server DNS. Inoltre installa l’indirizzo del gateway di default nella sua **tabella di inoltro** (Paragrafo 4.1). Il laptop di Bob invierà tutti i datagrammi il cui indirizzo di destinazione è all’esterno della sua sottorete 68.85.2.0/24 al gateway di default. A questo punto, il laptop di Bob ha inizializzato le sue componenti di rete ed è pronto a iniziare a elaborare la lettura della pagina web. Si noti che solo gli ultimi due passi DHCP dei quattro presentati nel Capitolo 4 sono realmente necessari.

### 6.7.2 Siamo ancora all’inizio: DNS e ARP

Quando Bob scrive l’URL di www.google.com nel suo browser web inizia la lunga catena di eventi che alla fine porterà a visualizzare la home page di Google. Il web browser di Bob inizia il processo creando una **socket TCP** (Paragrafo 2.7) che verrà usata per inviare una **richiesta HTTP** (Paragrafo 2.2) a www.google.com. Il laptop di Bob, per creare la socket, deve conoscere l’indirizzo IP di www.google.com. Abbiamo visto nel Paragrafo 2.5 che il **protocollo DNS** viene usato per fornire il servizio di traduzione da nome a indirizzo IP.

8. Il sistema operativo del laptop di Bob crea un **messaggio di DNS query** (Paragrafo 2.5.3) inserendo la stringa “www.google.com” nella sezione riguardante la richiesta del messaggio DNS. Tale messaggio DNS è quindi posto all’interno di un segmento UDP con porta di destinazione 53 (server DNS). Il segmento UDP è quindi posto all’interno di un datagramma IP con indirizzo IP di destinazione 68.87.71.226 (l’indirizzo del server DNS restituito nel messaggio DHCP ACK al passo 5) e indirizzo IP sorgente 68.85.2.101.
9. Il laptop di Bob quindi inserisce il datagramma contenente il messaggio di richiesta DNS in un frame Ethernet che verrà inviato, con indirizzo a livello di collegamento, al gateway della rete della scuola di Bob. Tuttavia il laptop di Bob, pur conoscendo l’indirizzo IP del gateway della scuola (68.85.2.1) tramite il messaggio DHCP ACK del passo 5, non ne conosce l’indirizzo MAC, per ottenere il quale deve usare il **protocollo ARP** (Paragrafo 6.4.1).
10. Il laptop di Bob crea un messaggio di **ARP query** con indirizzo IP di destinazione 68.85.2.1 (il gateway di default), pone il messaggio ARP all’interno di un frame Ethernet con indirizzo di destinazione broadcast FF:FF:FF:FF:FF:FF e invia il frame Ethernet allo switch, che lo consegna a tutti i dispositivi connessi compreso il gateway.

11. Il router gateway riceve il frame contenente il messaggio di interrogazione ARP sull’interfaccia della rete della scuola e scopre che l’indirizzo IP 68.85.2.1 nel messaggio ARP corrisponde all’indirizzo IP della sua interfaccia. Il gateway prepara quindi un messaggio **ARP reply** che indica che il suo indirizzo MAC 00:22:6B:45:1F:1B corrisponde all’indirizzo IP 68.85.2.1. Pone il messaggio di risposta ARP in un frame Ethernet con l’indirizzo di destinazione 00:16:D3:23:68:8A (il laptop di Bob) e invia il frame allo switch che lo consegna al laptop di Bob.
12. Il laptop di Bob riceve il frame contenente il messaggio di risposta ARP ed estrae l’indirizzo MAC del gateway (00:22:6B:45:1F:1B) dal messaggio di risposta ARP.
13. Il laptop di Bob può ora (finalmente!) indirizzare il frame Ethernet contenente l’interrogazione DNS all’indirizzo MAC del gateway. Si noti che il datagramma IP in questo frame ha indirizzo IP di destinazione 68.87.71.226 (il server DNS), mentre il frame ha indirizzo di destinazione 00:22:6B:45:1F:1B (il router gateway). Il laptop di Bob invia questo frame allo switch che lo consegna al gateway.

### 6.7.3 Siamo ancora all’inizio: instradamento intra-dominio al server DNS

14. Il gateway riceve il frame ed estrae il datagramma IP contenente l’interrogazione DNS. Il router ricerca l’indirizzo di destinazione di tale datagramma (68.87.71.226) e determina sulla base della sua tabella di inoltro che il datagramma dovrebbe essere inviato al router più a sinistra nella rete Comcast mostrata nella Figura 6.32. Il datagramma IP è posto all’interno di un frame appropriato per il collegamento che connette il router della scuola al router Comcast più a sinistra; il frame viene trasmesso.
15. Il router più a sinistra nella rete Comcast riceve il frame, estrae il datagramma IP, esamina l’indirizzo di destinazione del datagramma (68.87.71.226) e determina l’interfaccia di uscita sulla quale inoltrare il datagramma al server DNS sulla base della sua tabella di inoltro, che è stata riempita dal protocollo intra-dominio di Comcast (come **RIP**, **OSPF** o **IS-IS**, Paragrafo 5.3) e dal **protocollo inter-dominio di Internet, BGP** (Paragrafo 5.4).
16. Infine, il datagramma IP contenente l’interrogazione DNS arriva al server DNS, che estrae il messaggio di interrogazione DNS, ricerca il nome www.google.com nel suo database DNS (Paragrafo 2.5) e trova il **record di risorsa DNS** che contiene l’indirizzo IP (64.233.169.105) di www.google.com (assumendo che sia nella cache del server DNS). Si ricordi che i dati nella cache sono originati dal **DNS server autoritativo** (Paragrafo 2.5.2) di google.com. Il server DNS scrive un messaggio **DNS reply** contenente la corrispondenza tra il nome dell’host e l’indirizzo IP e lo pone in un segmento UDP; infine pone il segmento all’interno di un datagramma IP indirizzato al laptop di Bob (68.85.2.101). Questo datagramma verrà restituito al router della scuola attraverso la rete Comcast e da qua al laptop di Bob tramite lo switch Ethernet.

17. Il laptop di Bob estrae l'indirizzo IP del server www.google.com dal messaggio DNS. Finalmente, dopo un sacco di lavoro, il laptop di Bob è pronto a contattare il server www.google.com!

#### 6.7.4 Interazione client-server: TCP e HTTP

18. Il laptop di Bob, ora che ha l'indirizzo IP di www.google.com, può creare la **socket TCP** (Paragrafo 2.7) che verrà usata per inviare un messaggio di **HTTP GET** (Paragrafo 2.2.3) a www.google.com. Quando Bob crea la socket TCP, per prima cosa il laptop di Bob effettua l'**handshake a tre vie** (Paragrafo 3.5.6) con TCP su www.google.com. Quindi il laptop di Bob crea un segmento **TCP SYN** con porta di destinazione 80 (per HTTP), pone il segmento TCP all'interno di un datagramma IP con indirizzo IP di destinazione 64.233.169.105 (www.google.com), pone il datagramma all'interno di un frame con indirizzo MAC di destinazione 00:22:6B:45:1F:1B (il gateway) e invia il frame allo switch.
19. I router nelle reti della scuola, di Comcast e di Google inoltrano il datagramma contenente TCP SYN a www.google.com, usando ognuno la propria tabella di inoltro, come spiegato nei passi da 14 a 16. Si ricordi che le righe delle tabelle di inoltro dei router che governano la procedura di inoltro dei pacchetti sul collegamento inter-dominio tra Comcast e Google sono determinate dal protocollo **BGP** (Capitolo 5).
20. Infine, il datagramma contenente il TCP SYN arriva a www.google.com. Il messaggio TCP SYN viene estratto dal datagramma e viene effettuato un demultiplexing verso la socket di benvenuto associata alla porta 80. Viene creata una socket di connessione (Paragrafo 2.7) per la connessione TCP tra il server HTTP di Google e il laptop di Bob. Un segmento TCP SYNACK (Paragrafo 3.5.6) viene generato, posto all'interno di un datagramma indirizzato al laptop di Bob e infine inserito in un frame appropriato al collegamento tra www.google.com e il primo router.
21. Il datagramma contenente il segmento TCP SYNACK viene inoltrato attraverso le reti di Google, Comcast e della scuola per arrivare infine al controller Ethernet del laptop di Bob. Viene effettuato un demultiplexing del datagramma all'interno del sistema operativo alla socket TCP creata al passo 18, che entra nello stato di connessione.
22. Con la socket sul laptop di Bob finalmente pronta per trasmettere byte a www.google.com, il browser di Bob crea il messaggio HTTP GET (Paragrafo 2.2.3) contenente l'URL da leggere. Quindi il messaggio HTTP GET viene scritto nella socket e diventa il payload di un segmento TCP. Il segmento TCP viene posto in un datagramma, inviato e consegnato a www.google.com come descritto dei passi da 18 a 20.
23. Il server HTTP di www.google.com legge dalla socket TCP il messaggio HTTP GET, crea un messaggio di **risposta HTTP** (Paragrafo 2.2), pone il contenuto della pagina web richiesta nel corpo del messaggio di risposta HTTP che invia alla socket TCP.

24. Il datagramma contenente il messaggio di risposta HTTP viene inoltrato attraverso le reti di Google, Comcast e della scuola al laptop di Bob. Il browser di Bob legge dalla socket la risposta HTTP, estrae il codice HTML della pagina web dal corpo della risposta HTTP e finalmente (finalmente!) visualizza la pagina web!

Con questo nostro esempio abbiamo coperto gran parte dei fondamenti delle reti! Anche se l'esempio potrebbe sembrarvi esageratamente dettagliato, in realtà sono stati omessi molti possibili protocolli aggiuntivi (per esempio, il NAT in esecuzione sul gateway della scuola, l'accesso wireless alla rete della scuola, i protocolli di sicurezza per accedere alla rete della scuola o per crittografare segmenti o datagrammi, i protocolli di gestione della rete) e considerazioni (le cache web, la gerarchia DNS) che si incontrano nella Internet pubblica. Tratteremo questi e altri argomenti nei prossimi capitoli di questo libro.

Infine sottolineiamo che il nostro esempio non solo fornisce una visione globale e dall'alto delle reti, ma anche una visione degli ingranaggi che formano molti dei protocolli studiati fin qui. L'esempio si è focalizzato più sul “come” che sul “perché”; per una visione più ampia e riflessiva della progettazione dei protocolli di rete è opportuno analizzare il contenuto del Box 4.4 – Paragrafo 4.5 – sui principi dell'architettura di Internet.

## 6.8 Riepilogo

Nel corso del capitolo abbiamo avuto modo di considerare il livello di collegamento e di studiarne i servizi e i principi che sono alla base delle sue operazioni insieme ai diversi protocolli che li utilizzano per implementare i servizi di trasmissione dati.

Abbiamo appreso che il servizio fondamentale del livello di collegamento è costituito dal trasporto di datagrammi del livello di rete tra nodi adiacenti e che i protocolli del livello di collegamento operano incapsulando i datagrammi di rete all'interno di un frame prima di trasferirlo sul collegamento. Inoltre, abbiamo visto come i differenti protocolli forniscono differenti servizi di accesso ai collegamenti, di consegna (affidabilità, rilevazione e correzione dell'errore), di controllo di flusso e trasmissione (full-duplex invece che half-duplex). La presenza di queste differenze è essenzialmente legata alla varietà delle tipologie di collegamento sui cui operano i protocolli del livello di collegamento. Un collegamento punto a punto ha un unico trasmittente e un solo ricevente che comunicano su un “cavo”, mentre un collegamento ad accesso multiplo viene condiviso da molti trasmittenti e riceventi; di conseguenza, il protocollo a livello di collegamento dispone a sua volta di uno specifico protocollo (il suo protocollo di accesso multiplo) per coordinare l'accesso al collegamento. Nel caso di MPLS, il collegamento che connette due nodi adiacenti può in realtà costituire una rete (per esempio, due router adiacenti, cioè che rappresentano l'uno per l'altro il successivo passo verso una certa destinazione). In effetti, considerare una rete come un collegamento presenta una sua specifica motivazione. Per esempio, un collegamento telefonico che connette un modem domestico con un modem remoto utilizza un percorso che attraversa una complessa e sofisticata rete telefonica.

Tra i principi basilari della comunicazione a livello di collegamento abbiamo esaminato la rilevazione degli errori e le tecniche di correzione, i protocolli di accesso multiplo, la virtualizzazione (VLAN), l'indirizzamento e la costruzione delle estensioni di reti di area locale attraverso switch e le reti dei data center. In questo momento l'attenzione a livello di collegamento è concentrata su queste reti commutate. Nel caso della rilevazione/correzione degli errori abbiamo esaminato come sia possibile inserire dei bit addizionali all'intestazione di un frame per rilevare e, in alcuni casi, correggere gli errori che possono verificarsi durante la trasmissione sul collegamento. Abbiamo trattato i semplici schemi di parità e checksum oltre che i più complessi controlli a ridondanza ciclica, per occuparci in seguito dei protocolli di accesso multiplo. Tre sono le tipologie di approccio esaminate per il coordinamento dell'accesso a un canale broadcast: la suddivisione del canale (TDM, FDM), l'accesso casuale (ALOHA e CSMA), e gli approcci a rotazione (polling e token passing). La necessità di fornire gli indirizzi dei nodi a livello di collegamento è stata vista come la conseguenza della presenza di molti nodi che condividono un singolo canale broadcast. Abbiamo quindi preso in esame la profonda differenza che esiste tra gli indirizzi fisici e quelli a livello di rete e visto come in Internet venga utilizzato il protocollo ARP (*address resolution protocol*) per muoversi fra questi due indirizzamenti. È stato poi evidenziato il modo in cui i nodi condividono un canale broadcast e come si possano collegare tra loro diverse reti locali per formare LAN più vaste (il tutto senza l'intervento del livello di rete). Abbiamo anche visto come si possa creare più LAN virtuali su una singola infrastruttura fisica.

Abbiamo concluso i nostri studi sul livello di collegamento focalizzando l'attenzione su come le reti MPLS forniscano servizi a livello di collegamento quando interconnettono router IP e una panoramica della progettazione di reti per grandi data center. Concludiamo questo capitolo, e in generale i primi cinque capitoli, identificando i tanti protocolli necessari a scaricare una semplice pagina web. Avendo analizzato il livello di collegamento, il nostro viaggio lungo la pila dei protocolli è ora concluso. Certamente, esiste anche il livello fisico, ma i dettagli su questo è forse meglio delegarli ad altri corsi. Abbiamo, tuttavia, toccato alcuni aspetti del livello fisico (Paragrafo 1.2). Considereremo ancora il livello fisico quando affronteremo le caratteristiche dei collegamenti wireless nel prossimo capitolo.

A questo punto il nostro *excursus* lungo la pila dei protocolli è terminato, ma lo studio delle reti e della loro interconnessione non si è ancora concluso. Nei successivi due capitoli (disponibili in inglese sulla piattaforma MyLab) affronteremo le reti wireless, le applicazioni multimediali in rete, la sicurezza e la gestione delle reti. Questi quattro temi (a volte indicati in altri testi come “argomenti avanzati”) non rientrano specificamente in alcun particolare livello, ma richiedono una solida conoscenza di tutta la pila dei protocolli.

## Domande e problemi

### Domande di revisione

#### PARAGRAFI 6.1-6.2

- R1.** Considerate l'analogia sui trasporti descritta nel Paragrafo 6.1.1. Se i passeggeri sono l'analogo di un datagramma, qual è l'analogo dei frame a livello di collegamento?
- R2.** Se tutti i collegamenti Internet fornissero un servizio di consegna affidabile, quello offerto da TCP risulterebbe ridondante? Perché?
- R3.** Quali servizi può offrire un protocollo di collegamento al livello di rete? E quali tra questi hanno servizi corrispondenti in IP? E in TCP?

#### PARAGRAFO 6.3

- R4.** Supponete che due nodi inizino a trasmettere nello stesso istante un pacchetto di lunghezza  $L$  su un canale broadcast con tasso trasmissivo  $R$ . Indichiamo il ritardo di propagazione fra quei due nodi con  $d_{prop}$ . Ci sarà collisione se  $d_{prop} < L/R$ ? Perché?
- R5.** Nel Paragrafo 6.3 abbiamo elencato quattro utili caratteristiche dei canali broadcast. Quali sono quelle possedute dallo slotted ALOHA? E quali dal “token passing”?
- R6.** In CSMA/CD, dopo la quinta collisione, qual è la probabilità che un nodo scelga  $K = 4$ ? Il risultato  $K = 4$  corrisponde a un ritardo di quanti secondi su una Ethernet a 10 Mbps?
- R7.** Descrivete i protocolli *polling* e *token passing* utilizzando un'analogia con il comportamento umano.
- R8.** Perché il protocollo token-ring risulterebbe inefficiente in una LAN con un perimetro molto esteso?

#### PARAGRAFO 6.4

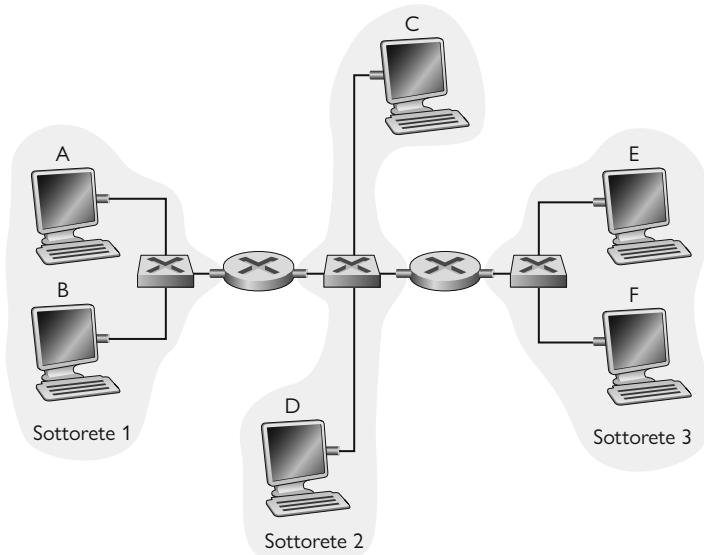
- R9.** Quanto è grande lo spazio degli indirizzi MAC? E quello di IPv4 e di IPv6?
- R10.** Supponete che i nodi A, B e C siano connessi alla stessa LAN broadcast (attraverso le loro schede di rete). Se A invia migliaia di datagrammi IP a B, ciascuno encapsulato in un frame destinato all'indirizzo MAC di B, la scheda di rete di C elaborerà questi frame? E in tal caso, passerà i datagrammi IP in essi contenuti al livello di rete di C? La risposta cambierebbe se A inviasse i frame con l'indirizzo MAC broadcast?
- R11.** Perché si inviano le richieste ARP all'interno di un frame broadcast? Perché il frame contenente una risposta ARP ha uno specifico indirizzo MAC di destinazione?
- R12.** Nella rete della Figura 6.19 il router ha due moduli ARP, ciascuno con la propria tabella ARP. È possibile che lo stesso indirizzo MAC compaia in entrambe le tabelle?
- R13.** In che cosa differisce la struttura dei pacchetti Ethernet 10Base-T, 100Base-T e Gigabit Ethernet?
- R14.** Considerate la Figura 6.15. Quante sottoreti ci sono, seguendo il discorso relativo all'indirizzamento del Paragrafo 4.3?
- R15.** Qual è il massimo numero di LAN che può essere configurato su uno switch che supporta il protocollo 802.1Q? Perché?
- R16.** Si supponga che  $N$  switch debbano supportare  $K$  VLAN e che gli switch siano connessi tramite trunking. Quante porte sono necessarie per connettere tra loro gli switch? Giustificate la risposta.

### Problemi

- P1.** Supponete che un pacchetto contenga la stringa 1110 0110 1001 0101 e che sia utilizzato uno schema di parità dispari. Quale sarebbe il valore del campo checksum per uno schema di parità bidimensionale? La vostra risposta dovrebbe essere tale da imporre una lunghezza minima al campo checksum.
- P2.** Mostrate (con un esempio diverso da quello della Figura 6.5) come il controllo di parità bidimensionale possa rilevare e correggere un singolo bit errato. Fate un esempio di doppio errore nei bit che possa essere rilevato, ma non corretto.
- P3.** Supponete che la porzione di informazioni in un pacchetto ( $D$  nella Figura 6.3) contenga 10 byte costituiti dalla rappresentazione binaria ASCII senza segno a otto bit della stringa “internet”. Calcolate il checksum di Internet per questi dati.
- P4.** Considerate il problema precedente, ma supponete che i 10 byte contengano:

- (a) la rappresentazione binaria dei numeri da 1 a 10;  
 (b) la rappresentazione ASCII delle lettere da B a K (maiuscole);  
 (c) la rappresentazione ASCII delle lettere da b a k (minuscole).
- P5.** Considerate un codice CRC con un generatore a sette bit  $G = 10011$  e supponete che  $D$  abbia il valore 1010101010. Qual è il valore di  $R$ ?
- P6.** Considerate il problema precedente, ma supponete che  $D$  abbia valore:  
 (a) 1000100101.  
 (b) 0101101010.  
 (c) 0110100011.
- P7.** In questo problema esploriamo alcune proprietà di CRC. Rispondete alle seguenti domande riguardanti il generatore  $G (=1001)$  dato nel Paragrafo 6.2.3.  
 (a) Perché è in grado di rilevare qualsiasi errore a un bit nei dati  $D$ ?

- (b) È in grado di rilevare qualsiasi errore a un numero dispari di bit? Perché?
- P8.** Nel Paragrafo 6.3 abbiamo fornito uno schema della derivazione dell'efficienza di slotted ALOHA. In questo problema completeremo la derivazione.
- Ricordate che quando ci sono  $N$  nodi attivi l'efficienza di slotted ALOHA è  $Np(1-p)^{N-1}$ . Trovate il valore di  $p$  che rende massima quest'espressione.
  - Usando il valore di  $p$  ricavato in (a), trovate l'efficienza di slotted ALOHA per  $N$  che tende all'infinito. Suggerimento:  $(1 - 1/N)^N$  tende a  $1/e$  quando  $N$  tende all'infinito.
- P9.** Mostrate che la massima efficienza di ALOHA puro è  $1/(2e)$ . Nota: questo problema risulta semplice se avete risolto quello precedente.
- P10.** Supponete che due nodi, A e B, siano in competizione per accedere a un canale che usa slotted ALOHA, che A abbia più dati da trasmettere di B e che la probabilità di ritrasmissione di A,  $p_A$ , sia più grande di quella di B,  $p_B$ .
- Calcolate l'espressione del valore medio del throughput di A. Qual è l'efficienza totale del protocollo con questi due nodi?
  - Se  $p_A = 2p_B$ , il throughput medio di A è il doppio di quello di B?
  - Supponete di avere in generale  $N$  nodi, tra cui A con probabilità di ritrasmissione  $p_A$  e gli altri con probabilità  $p$ . Calcolate l'espressione del valore medio del throughput di A e degli altri nodi.
- P11.** Supponete che quattro nodi attivi A, B, C e D siano in competizione per accedere a un canale che usa slotted ALOHA. Assumete che ciascun nodo abbia un numero infinito di pacchetti da inviare. Ciascun nodo tenta di trasmettere in ogni slot con probabilità  $p$ . Il primo slot è indicato con 1, il secondo con 2 e così via.
- (a) Qual è la probabilità che il nodo A abbia successo al primo tentativo di trasmettere nello slot 4?
- (b) Qual è la probabilità che un nodo abbia successo per trasmettere nello slot 5?
- (c) Qual è la probabilità che il primo successo si verifichi nello slot 4?
- (d) Qual è l'efficienza di questo sistema con quattro nodi?
- P12.** Riportate in un grafico l'efficienza di slotted ALOHA e ALOHA puro in funzione di  $p$  per i seguenti valori di  $N$ .
- $N = 10$ .
  - $N = 30$ .
  - $N = 50$ .
- P13.** Considerate un canale broadcast con  $N$  nodi e un tasso di trasmissione di  $R$  bps. Supponete che: (a) il canale broadcast utilizzzi il protocollo polling (con l'aggiunta di un nodo di controllo) per l'accesso multiplo; (b) il tempo da quando un nodo completa l'invio a quando quello successivo può trasmettere (cioè, il ritardo di polling), sia  $d_{poll}$ ; (c) all'interno di un ciclo di polling, a un dato nodo sia consentita la trasmissione di un massimo di  $Q$  bit. Qual è il massimo volume di dati (throughput) del canale broadcast?
- P14.** Considerate le tre LAN interconnesse da due router rappresentate nella Figura 6.33.
- Assegnate indirizzi IP a tutte le interfacce. Per la sottorete 1 usate indirizzi nella forma 192.168.1.xxx; per la 2 utilizzate gli indirizzi nella forma 192.168.2.xxx; per la 3 impiegate gli indirizzi nella forma 192.168.3.xxx.
  - Assegnate indirizzi MAC alle schede di rete.
  - Supponete di inviare un datagramma IP dall'host E a all'host B e che tutte le tabelle ARP siano aggiornate. Enumerate tutti i passi come nell'esempio del router singolo (Paragrafo 6.4.1).



**Figura 6.33**  
Tre sottoreti interconnesse da router.

- (d) Ripetete (c), assumendo ora che la tabella ARP nell'host trasmittente sia vuota (e che le altre siano aggiornate).

**P15.** Nella Figura 6.33 sostituite il router tra le sottoreti 1 e 2 con uno switch S1 e chiamate R1 il router tra le sottoreti 2 e 3.

- Se l'Host E invia un datagramma IP all'Host F, chiede aiuto al router R1 per inoltrarlo? Perché? Quali sono la sorgente e destinazione IP e l'indirizzo MAC nel frame Ethernet che contiene il datagramma?
- Supponete che E voglia inviare un datagramma IP a B e che la cache ARP di E non contenga l'indirizzo MAC di B. B effettuerà una richiesta ARP per trovare l'indirizzo ARP di B? Perché? Quali sono la sorgente e la destinazione IP e l'indirizzo MAC nel frame Ethernet che contiene il datagramma e viene consegnato al router R1?
- Supponete che A voglia inviare un datagramma IP a B, ma che né la cache ARP di A contiene l'indirizzo MAC di B, né la cache ARP di B contiene l'indirizzo MAC di A. Supponete inoltre che la tabella di inoltro dello switch S1 contiene solo le informazioni per l'Host B e il router R1. Quindi A effettua una richiesta ARP. Quale azione intraprende lo switch S1 una volta che ha ricevuto il messaggio di richiesta ARP? Anche il router R1 riceverà tale richiesta? In questo caso, R1 inoltrerà il messaggio alla sottoretina 3? L'Host B, una volta ricevuta la richiesta ARP, invierà ad A una risposta ARP. Ma invierà un messaggio di richiesta ARP per l'indirizzo MAC di A? Perché? Che cosa farà lo switch S1 quando riceverà il messaggio di risposta ARP dall'Host B?

**P16.** Considerate il problema precedente, ma ora supponete che sia il router tra le sottoreti 2 e 3 a essere sostituito da uno switch. Rispondete alle domande da (a) a (c) del problema precedente in questo nuovo contesto.

**P17.** Ricordiamo che, con il protocollo CSMA/CD, la scheda di rete attende il tempo equivalente a  $K \times 512$  bit dopo una collisione, dove  $K$  è casuale. Per  $K = 100$ , quanto deve aspettare una scheda di rete prima di tornare alla fase 2 per Ethernet a 100 Mbps? E a 1 Gbps?

**P18.** Supponete che: (a) i nodi A e B siano sullo stesso segmento Ethernet a 10 Mbps e che il ritardo di propagazione fra i due nodi sia di 325;<sup>3</sup> (b) il nodo A stia inviando un frame e che, prima che termini, B cominci a trasmettere. Il nodo A può concludere il trasferimento prima di rilevare che il nodo B sta trasmettendo? Perché? Se la risposta è affermativa, allora A crede, sbagliando, che il suo pacchetto sia stato trasmesso con successo, senza collisioni. Suggerimento: ipotizzate che A cominci a trasmettere un frame al tempo  $t = 0$ . Nel caso peggiore, A invia un frame con dimensioni minime di  $512 + 64$  bit. Così A finirebbe di trasmettere il frame all'istante  $t = 512 + 64$ . Allora, la risposta è negativa se il segnale di B raggiunge A prima dell'istante  $t = 512 + 64$ . Nel caso peggiore, in quale istante il segnale di B raggiunge A?

**P19.** Supponete che: (a) i nodi A e B siano sullo stesso segmento broadcast a 10 Mbps e che il ritardo di propagazione fra i due nodi sia di 245; (b) A e B inviano nello stesso istante frame che collidono e, quindi, A e B scelgano diversi valori di  $K$  nell'algoritmo CSMA/CD. Assumendo che nessun altro nodo sia attivo, le ritrasmissioni di A e B possono collidere? Per i nostri scopi, è sufficiente il seguente esempio. Ipotizzate che A e B comincino la trasmissione a  $t = 0$ . Entrambi rilevano la collisione a  $t = 245$ . Assumiamo  $K_A = 0$  e  $K_B = 1$ : in quale momento B programmerà la sua ritrasmissione? Quando A comincerà la ritrasmissione? (Nota: dopo il ritorno alla fase 2 i nodi devono aspettare che il canale sia libero, come stabilito protocollo.) In quale momento il segnale di A raggiunge B? B si astiene dal trasmettere al tempo che ha programmato?

**P20.** In questo problema dovrete derivare l'efficienza di un protocollo di accesso multiplo simile a CSMA/CD. In questo protocollo il tempo è suddiviso in slot e tutte le schede di rete sono sincronizzate con gli slot. A differenza dello slotted ALOHA, tuttavia, la durata di uno slot (in secondi) è molto inferiore al tempo di un frame (il tempo per trasmettere un frame). Indichiamo con  $S$  la lunghezza di uno slot. Supponiamo che tutti i frame abbiano lunghezza costante  $L = kRS$ , dove  $R$  è il tasso di trasmissione del canale e  $k$  è un intero grande, e che ci siano  $N$  nodi, ciascuno con un numero infinito di frame da spedire. Assumiamo anche che  $d_{prop} < S$ , così che tutti i nodi possano rilevare una collisione prima del termine di un tempo di slot. Il protocollo ha le seguenti caratteristiche.

- Se, per un dato slot, nessun nodo occupa il canale, tutti i nodi si contendono il canale; in particolare, ciascun nodo trasmette nello slot con probabilità  $p$ . Se un nodo trasmette nello slot, mantiene il possessore del canale per i seguenti  $k - 1$  slot e invia l'intero pacchetto.
- Se un nodo occupa il canale, tutti gli altri si astengono dal trasmettere fino a quando ha terminato l'invio del suo frame; dopo di che, gli altri nodi si contendono il canale.

Notate l'alternanza del canale fra due stati: uno stato “produttivo”, che dura esattamente  $k$  slot e uno stato “non produttivo”, che dura per un numero casuale di slot. Chiaramente, l'efficienza del canale è data dal rapporto  $k/(k + x)$ , dove  $x$  è il numero atteso di slot non produttivi consecutivi.

- Per  $N$  e  $p$  dati, determinate l'efficienza di questo protocollo.
- Per  $N$  dato, determinate  $p$  che rende massima l'efficienza.
- Utilizzando  $p$  (che è funzione di  $N$ ) trovato in (b), determinate l'efficienza quando  $N$  tende all'infinito.
- Dimostrate che questa efficienza tende a 1 quando la lunghezza del frame aumenta.

**P21.** Considerate la Figura 6.33 nel Problema P14. Fornite gli indirizzi MAC e IP per le interfacce dell'host A, dei due router e dell'host F. Supponete che l'host A invii un datagramma a F. Fornite gli indirizzi MAC di sorgente e di destinazione del frame che incapsula questo

- datagramma IP, quando viene trasmesso (i) da A al router di sinistra, (ii) dal router di sinistra a quello di destra, (iii) dal router di destra a F. Fornite anche gli indirizzi IP di sorgente e di destinazione del datagramma IP, incapsulato nel frame, in ciascuno dei tre punti.
- P22.** Supponete ora che il router più a sinistra nella Figura 6.3 sia sostituito da uno switch. Gli host A, B, C e D e il router a destra sono tutti collegati con una topologia a stella a questo switch. Fornite gli indirizzi MAC di sorgente e di destinazione del frame che incapsula questo datagramma IP, quando viene trasmesso (i) da A allo switch, (ii) dallo switch al router di destra, (iii) dal router di destra a F. Indicate anche gli indirizzi IP di sorgente e di destinazione del datagramma IP, incapsulato nel frame, in ciascuno dei tre punti.
- P23.** Considerate la Figura 6.15. Supponete che tutti i collegamenti siano a 1 Gbps. Qual è il massimo throughput totale aggregato che può essere raggiunto tra i 9 host e 2 server in questa rete, assumendo che tutti possano comunicare tra loro? Perché?
- P24.** Supponete che i tre switch dipartimentali della Figura 6.15 siano sostituiti da altrettanti hub. Tutti i collegamenti sono a 1 Gbps. Qual è il massimo throughput totale aggregato che può essere raggiunto tra i 9 host e 2 server in questa rete, assumendo che tutti possano comunicare tra loro? Perché?
- P25.** Supponete che *tutti* gli switch della Figura 6.15 siano sostituiti da altrettanti hub. Tutti i collegamenti sono a 1 Gbps. Qual è il massimo throughput totale aggregato che può essere raggiunto tra i 9 host e 2 server in questa rete, assumendo che tutti possano comunicare tra loro? Perché?
- P26.** Considerate le operazioni di apprendimento di uno switch nel contesto di una rete in cui 6 nodi etichettati da A a F sono connessi con una topologia a stella in una rete Ethernet. Supponete che: (i) B mandi un frame a E; (ii) E replichi con un frame a B; (iii) A mandi un frame a B; (iv) B replichi con un frame ad A. La tabella dello switch è inizialmente vuota. Mostrate lo stato della tabella dello switch prima e dopo ciascuno di questi eventi. Per ciascuno di questi eventi, identificate i collegamenti sui quali i frame trasmessi sono inoltrati e argomentate sinteticamente le vostre risposte.
- P27.** In questo problema esploriamo l'uso di pacchetti piccoli nelle applicazioni di VoIP. Uno degli inconvenienti legato alla piccola dimensione dei pacchetti è la larghezza di banda del collegamento consumata dai byte di intestazione. A questo scopo, supponete che il pacchetto consista di un numero  $L$  di byte e di 5 byte di intestazione.
- Supponete: (a) di inviare una sorgente vocale codificata digitalmente direttamente; (b) che la sorgente sia codificata a un tasso costante di 128 kbps; (c) che ciascun pacchetto sia interamente riempito prima che la sorgente lo invii in rete. Il tempo richiesto per riempire un pacchetto è il **ritardo di pacchettizzazione** (*packetization delay*). Determinate, in funzione di  $L$ , il ritardo di pacchettizzazione in millisecondi.
  - I ritardi di pacchettizzazione più grandi di 20 ms possono causare un'eco vistosa e spiacevole. Determinate il ritardo di pacchettizzazione per  $L = 1500$  byte (approssimativamente corrispondente a un pacchetto Ethernet di dimensione massima) e per  $L = 50$  (corrispondente a una cella ATM).
  - Calcolate il ritardo store-and-forward su un singolo switch per un collegamento con un tasso trasmissivo di  $R = 622$  Mbps usando  $L = 1500$  byte e  $L = 50$  byte.
  - Commentate i vantaggi dell'uso di pacchetti di piccole dimensioni.
- P28.** Considerate la VLAN con un solo switch mostrata nella Figura 6.25 e assumete che un router esterno sia connesso alla porta 1 dello switch. Assegnate indirizzi IP agli host e alle interfacce dei router di Ingegneria elettronica e di Informatica. Descrivete i passi intrapresi sia dal livello di rete che da quello di collegamento per trasferire un datagramma IP da un host di Ingegneria elettronica a uno di Informatica. (*Suggerimento:* rileggete nel testo la discussione riguardante la Figura 6.19).
- P29.** Considerate la rete MPLS nella Figura 6.29 e supponete che i router R5 e R6 abbiano funzionalità MPLS. Supponete che vogliono ingegnerizzare del traffico, in modo che i pacchetti da R6 destinati ad A siano commutati verso A via R6-R4-R3-R1 e che i pacchetti da R5 destinati ad A siano commutati verso A via R5-R4-R2-R1. Mostrate le tabelle MPLS di R5 e R6, come pure quella modificata di R4 che rende possibile questa configurazione.
- P30.** Considerate lo scenario del problema precedente, ma supponete che i pacchetti da R6 destinati a D siano commutati via R6-R4-R3, mentre quelli da R5 destinati a D siano commutati via R4-R2-R1-R3. Mostrate le tabelle MPLS di tutti i router che rendono possibile questa configurazione.
- P31.** In questo problema sfrutterete tutto quello che avete imparato sui protocolli Internet. Supponete di stare camminando in una stanza, di connettervi a Internet e di voler scaricare una pagina web. Quali sono i passi di protocollo che avvengono a partire da quando accendete il vostro PC a quando avete la pagina web? Supponete che nelle memorie cache del vostro DNS o del vostro browser non vi sia niente quando accendete il vostro PC. (*Suggerimento:* i passi includono l'uso dei protocolli Ethernet, DHCP, ARP, DNS, TCP e http). Indicate esplicitamente nei vostri passi come ottenete gli indirizzi IP e MAC del gateway.
- P32.** Considerate il data center con topologia gerarchica della Figura 6.30. Supponete che vi siano 80 coppie di flussi, di cui 10 tra il primo e il nono rack, 10 tra il secondo e il decimo rack, e così via. Supponete inoltre che tutti i collegamenti della rete siano a 10 Mbps, tranne i collegamenti tra gli host e gli switch TOR, che sono a 1 Gbps.
- Ogni flusso ha lo stesso tasso di invio dei dati; determinante il tasso trasmissivo massimo del flusso.
  - Con le stesse caratteristiche di traffico, determinate il tasso trasmissivo massimo del flusso per la to-

- pologia altamente interconnessa descritta nella Figura 6.31.
- (c) Con le stesse caratteristiche di traffico, ma coinvolgendo 20 host su ogni rack e 160 coppie di flussi, determinate il tasso trasmittivo massimo di ogni flusso per le due topologie.
- P33.** Considerate la rete gerarchica descritta nella Figura 6.30 e supponete che il data center debba supportare, tra varie applicazioni, anche la distribuzione di e-mail e video. Supponete che quattro rack siano riservati per le e-mail e altri quattro per il video. Per ognuna delle applicazioni tutti e quattro i rack devono giacere sotto un singolo switch di livello 2, perché i collegamenti dal livello 2 al livello 1 non hanno abbastanza banda per supportare il traffico intra-applicazione. Per le applicazioni e-mail supponete che per il 99,9% del tempo vengano usati solo tre rack e che l'applicazione video abbia le stesse caratteristiche di uso.
- (a) Per quale frazione di tempo l'applicazione e-mail necessita di usare un quarto rack? E per quanto riguarda l'applicazione video?
- (b) Supponendo che l'utilizzo video ed e-mail siano indipendenti, per quale frazione di tempo (o equivalentemente qual è la probabilità) entrambe le applicazioni necessitino del loro quarto rack?
- (c) Supponete che per un'applicazione sia accettabile avere scarsità di risorse (server) per lo 0,001 % di tempo o meno (causando rari periodi di degradazione delle prestazioni degli utenti).
- Discutete come sia possibile utilizzare la topologia descritta nella Figura 6.31 in modo che solo sette rack vengano assegnati collettivamente alle due applicazioni (supponendo che la topologia possa supportare tutto il traffico).

## Esercitazioni Wireshark: 802.11 Ethernet

---

Sulla piattaforma MyLab relativa a questo libro troverete un'esercitazione con Wireshark che esamina le operazioni del protocollo IEEE 802.3 e il formato dei frame Ethernet. Una seconda esercitazione esamina invece tracce di pacchetti in una rete domestica.

# Wireless and Mobile Networks

In the telephony world, the past 25 years have been the golden years of cellular telephony. The number of worldwide mobile cellular subscribers increased from 34 million in 1993 to 8.3 billion subscribers in 2019. There are now a larger number of mobile phone subscriptions than there are people on our planet. The many advantages of cell phones are evident to all—anywhere, anytime, untethered access to the global telephone network via a highly portable lightweight device. More recently, smartphones, tablets, and laptops have become wirelessly connected to the Internet via a cellular or WiFi network. And increasingly, devices such as gaming consoles, thermostats, home security systems, home appliances, watches, eye glasses, cars, traffic control systems and more are being wirelessly connected to the Internet.

From a networking standpoint, the challenges posed by networking these wireless and mobile devices, particularly at the link layer and the network layer, are so different from traditional wired computer networks that an individual chapter devoted to the study of wireless and mobile networks (*i.e.*, *this* chapter) is appropriate.

We'll begin this chapter with a discussion of mobile users, wireless links, and networks, and their relationship to the larger (typically wired) networks to which they connect. We'll draw a distinction between the challenges posed by the *wireless* nature of the communication links in such networks, and by the *mobility* that these wireless links enable. Making this important distinction—between wireless and mobility—will allow us to better isolate, identify, and master the key concepts in each area.

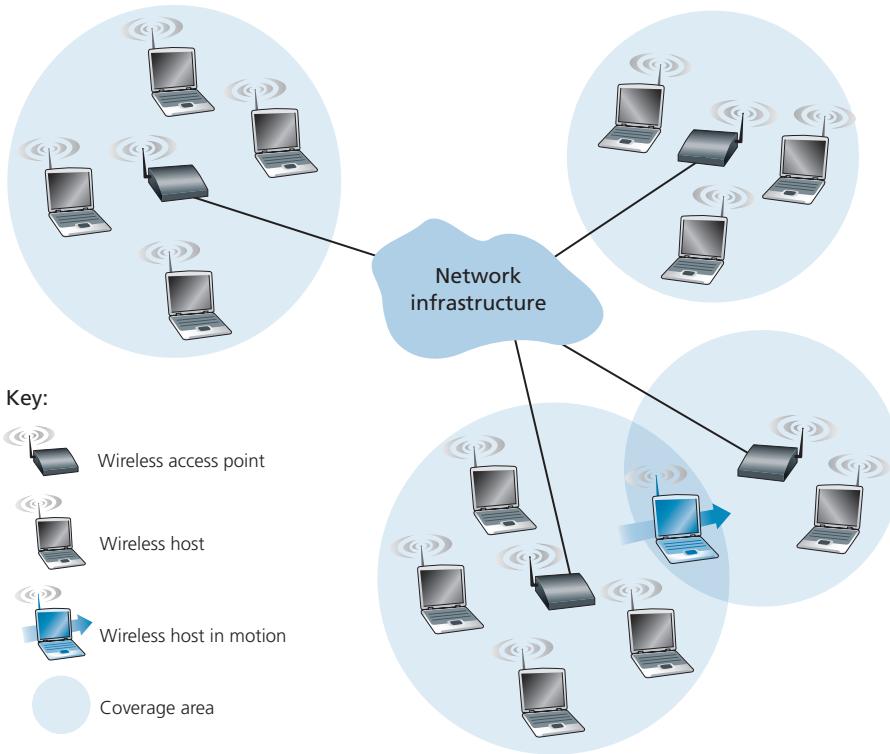
We will begin with an overview of wireless access infrastructure and associated terminology. We'll then consider the characteristics of this wireless link in

Section 7.2. We include a brief introduction to code division multiple access (CDMA), a shared-medium access protocol that is often used in wireless networks, in Section 7.2. In Section 7.3, we'll examine the link-level aspects of the IEEE 802.11 (WiFi) wireless LAN standard in some depth; we'll also say a few words about Bluetooth wireless personal area networks. In Section 7.4, we'll provide an overview of cellular Internet access, including 4G and emerging 5G cellular technologies that provide both voice and high-speed Internet access. In Section 7.5, we'll turn our attention to mobility, focusing on the problems of locating a mobile user, routing to the mobile user, and “handing over” the mobile user who dynamically moves from one point of attachment to the network to another. We'll examine how these mobility services are implemented in the 4G/5G cellular networks, and the in the Mobile IP standard in Section 7.6. Finally, we'll consider the impact of wireless links and mobility on transport-layer protocols and networked applications in Section 7.7.

## 7.1 Introduction

Figure 7.1 shows the setting in which we'll consider the topics of wireless data communication and mobility. We'll begin by keeping our discussion general enough to cover a wide range of networks, including both wireless LANs such as WiFi and 4G and 5G cellular networks; we'll drill down into a more detailed discussion of specific wireless architectures in later sections. We can identify the following elements in a wireless network:

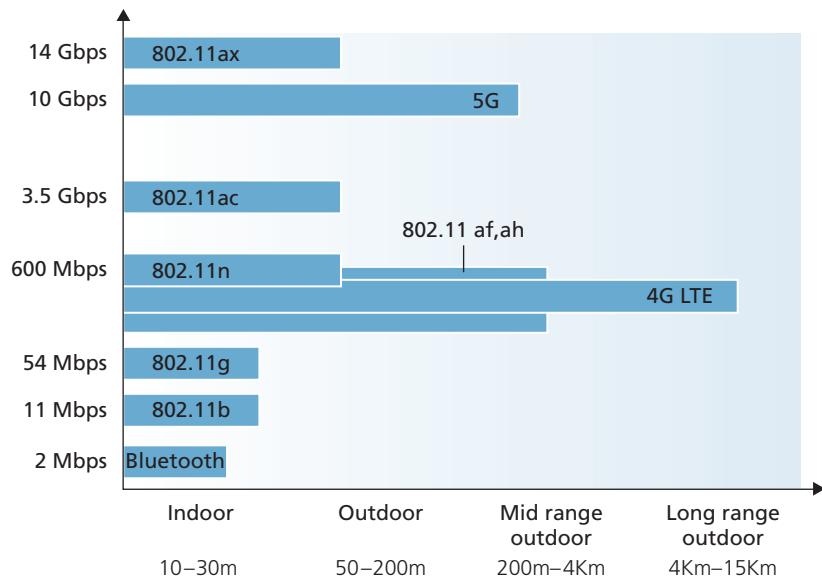
- *Wireless hosts.* As in the case of wired networks, hosts are the end-system devices that run applications. A **wireless host** might be a smartphone, tablet, or laptop, or it could be an Internet of Things (IoT) device such as a sensor, appliance, automobile, or any other of the myriad devices being connected to the Internet. The hosts themselves may or may not be mobile.
- *Wireless links.* A host connects to a base station (defined below) or to another wireless host through a **wireless communication link**. Different wireless link technologies have different transmission rates and can transmit over different distances. Figure 7.2 shows two key characteristics, link transmission rates and coverage ranges, of the more popular wireless network standards. (The figure is only meant to provide a rough idea of these characteristics. For example, some of these types of networks are only now being deployed, and some link rates can increase or decrease beyond the values shown depending on distance, channel conditions, and the number of users in the wireless network.) We'll cover these standards later in the first half of this chapter; we'll also consider other wireless link characteristics (such as their bit error rates and the causes of bit errors) in Section 7.2.



**Figure 7.1** ♦ Elements of a wireless network

In Figure 7.1, wireless links connect wireless hosts located at the edge of the network into the larger network infrastructure. We hasten to add that wireless links are also sometimes used *within* a network to connect routers, switches, and other network equipment. However, our focus in this chapter will be on the use of wireless communication at the network edge, as it is here that many of the most exciting technical challenges, and most of the growth, are occurring.

- **Base station.** The **base station** is a key part of the wireless network infrastructure. Unlike the wireless host and wireless link, a base station has no obvious counterpart in a wired network. A base station is responsible for sending and receiving data (e.g., packets) to and from a wireless host that is associated with that base station. A base station will often be responsible for coordinating the transmission of multiple wireless hosts with which it is associated. When we say a wireless host is “associated” with a base station, we mean that (1) the host is within the wireless communication distance of the base station, and (2) the host uses that base station to relay data between it (the host) and the larger network. **Cell towers** in cellular networks and **access points** in 802.11 wireless LANs are examples of base stations.



**Figure 7.2** ♦ Wireless transmission rates and range for WiFi, cellular 4G/5G and Bluetooth standards (note: axes are not linear)

In Figure 7.1, the base station is connected to the larger network (e.g., the Internet, corporate or home network), thus functioning as a link-layer relay between the wireless host and the rest of the world with which the host communicates.

Hosts associated with a base station are often referred to as operating in **infrastructure mode**, since all traditional network services (e.g., address assignment and routing) are provided by the network to which a host is connected via the base station. In **ad hoc networks**, wireless hosts have no such infrastructure with which to connect. In the absence of such infrastructure, the hosts themselves must provide for services such as routing, address assignment, DNS-like name translation, and more.

When a mobile host moves beyond the range of one base station and into the range of another, it will change its point of attachment into the larger network (i.e., change the base station with which it is associated)—a process referred to as **handoff** or **handover**. Such mobility raises many challenging questions. If a host can move, how does one find the mobile host's current location in the network so that data can be forwarded to that mobile host? How is addressing performed, given that a host can be in one of many possible locations? If the host moves *during* a TCP connection or phone call, how is data routed so that the connection

continues uninterrupted? These and many (many!) other questions make wireless and mobile networking an area of exciting networking research.

- *Network infrastructure.* This is the larger network with which a wireless host may wish to communicate.

Having discussed the “pieces” of a wireless network, we note that these pieces can be combined in many different ways to form different types of wireless networks. You may find a taxonomy of these types of wireless networks useful as you read on in this chapter, or read/learn more about wireless networks beyond this book. At the highest level we can classify wireless networks according to two criteria: (*i*) whether a packet in the wireless network crosses exactly *one wireless hop or multiple wireless hops*, and (*ii*) whether there is *infrastructure* such as a base station in the network:

- *Single-hop, infrastructure-based.* These networks have a base station that is connected to a larger wired network (e.g., the Internet). Furthermore, all communication is between this base station and a wireless host over a single wireless hop. The 802.11 networks you use in the classroom, café, or library; and the 4G LTE data networks that we will learn about shortly all fall in this category. The vast majority of our daily interactions are with single-hop, infrastructure-based wireless networks.
- *Single-hop, infrastructure-less.* In these networks, there is no base station that is connected to a wireless network. However, as we will see, one of the nodes in this single-hop network may coordinate the transmissions of the other nodes. Bluetooth networks (that connect small wireless devices such as keyboards, speakers, and headsets, and which we will study in Section 7.3.6) are single-hop, infrastructure-less networks.
- *Multi-hop, infrastructure-based.* In these networks, a base station is present that is wired to the larger network. However, some wireless nodes may have to relay their communication through other wireless nodes in order to communicate via the base station. Some wireless sensor networks and so-called **wireless mesh networks** deployed in homes fall in this category.
- *Multi-hop, infrastructure-less.* There is no base station in these networks, and nodes may have to relay messages among several other nodes in order to reach a destination. Nodes may also be mobile, with connectivity changing among nodes—a class of networks known as **mobile ad hoc networks (MANETs)**. If the mobile nodes are vehicles, the network is a **vehicular ad hoc network (VANET)**. As you might imagine, the development of protocols for such networks is challenging and is the subject of much ongoing research.

In this chapter, we’ll mostly confine ourselves to single-hop networks, and then mostly to infrastructure-based networks.

Let's now dig deeper into the technical challenges that arise in wireless and mobile networks. We'll begin by first considering the individual wireless link, deferring our discussion of mobility until later in this chapter.

## 7.2 Wireless Links and Network Characteristics

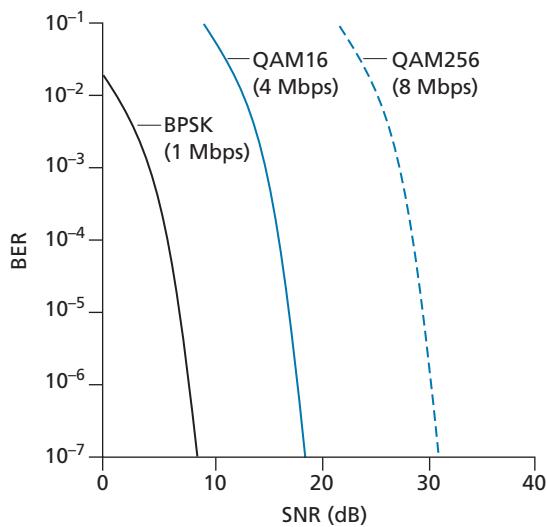
Wireless links differ from their wired counterparts in a number important ways:

- *Decreasing signal strength.* Electromagnetic radiation attenuates as it passes through matter (e.g., a radio signal passing through a wall). Even in free space, the signal will disperse, resulting in decreased signal strength (sometimes referred to as **path loss**) as the distance between sender and receiver increases.
- *Interference from other sources.* Radio sources transmitting in the same frequency band will interfere with each other. For example, 2.4 GHz wireless phones and 802.11b wireless LANs transmit in the same frequency band. Thus, the 802.11b wireless LAN user talking on a 2.4 GHz wireless phone can expect that neither the network nor the phone will perform particularly well. In addition to interference from transmitting sources, electromagnetic noise within the environment (e.g., a nearby motor, a microwave) can result in interference. For this reason, a number of more recent 802.11 standards operate in the 5GHz frequency band.
- *Multipath propagation.* **Multipath propagation** occurs when portions of the electromagnetic wave reflect off objects and the ground, taking paths of different lengths between a sender and receiver. This results in the blurring of the received signal at the receiver. Moving objects between the sender and receiver can cause multipath propagation to change over time.

For a detailed discussion of wireless channel characteristics, models, and measurements, see [Anderson 1995; Almers 2007].

The discussion above suggests that bit errors will be more common in wireless links than in wired links. For this reason, it is perhaps not surprising that wireless link protocols (such as the 802.11 protocol we'll examine in the following section) employ not only powerful CRC error detection codes, but also link-level reliable-data-transfer protocols that retransmit corrupted frames.

Having considered the impairments that can occur on a wireless channel, let's next turn our attention to the host receiving the wireless signal. This host receives an electromagnetic signal that is a combination of a degraded form of the original signal transmitted by the sender (degraded due to the attenuation and multipath propagation effects that we discussed above, among others) and background noise in the environment. The **signal-to-noise ratio (SNR)** is a relative measure of the strength of the received signal (i.e., the information being transmitted) and this noise. The SNR is typically measured in units of decibels (dB), a unit of measure that some think is used by

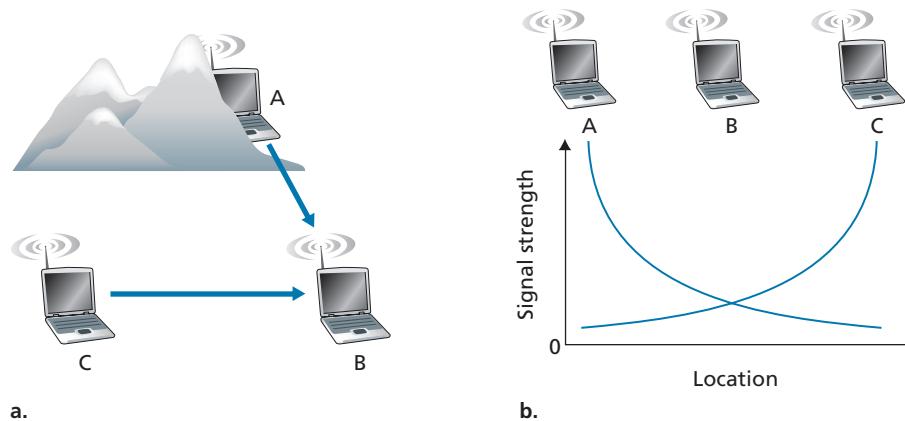


**Figure 7.3** ♦ Bit error rate, transmission rate, and SNR

electrical engineers primarily to confuse computer scientists. The SNR, measured in dB, is 20 times the ratio of the base-10 logarithm of the amplitude of the received signal to the amplitude of the noise. For our purposes here, we need only know that a larger SNR makes it easier for the receiver to extract the transmitted signal from the background noise.

Figure 7.3 (adapted from [Holland 2001]) shows the bit error rate (BER)—roughly speaking, the probability that a transmitted bit is received in error at the receiver—versus the SNR for three different modulation techniques for encoding information for transmission on an idealized wireless channel. The theory of modulation and coding, as well as signal extraction and BER, is well beyond the scope of this text (see [Schwartz 1980; Goldsmith 2005] for a discussion of these topics). Nonetheless, Figure 7.3 illustrates several physical-layer characteristics that are important in understanding higher-layer wireless communication protocols:

- *For a given modulation scheme, the higher the SNR, the lower the BER.* Since a sender can increase the SNR by increasing its transmission power, a sender can decrease the probability that a frame is received in error by increasing its transmission power. Note, however, that there is arguably little practical gain in increasing the power beyond a certain threshold, say to decrease the BER from  $10^{-12}$  to  $10^{-13}$ . There are also *disadvantages* associated with increasing the transmission power: More energy must be expended by the sender



**Figure 7.4** ♦ Hidden terminal problem caused by obstacle (a) and fading (b)

(an important concern for battery-powered mobile users), and the sender's transmissions are more likely to interfere with the transmissions of another sender (see Figure 7.4(b)).

- For a given SNR, a modulation technique with a higher bit transmission rate (whether in error or not) will have a higher BER. For example, in Figure 7.3, with an SNR of 10 dB, BPSK modulation with a transmission rate of 1 Mbps has a BER of less than  $10^{-7}$ , while with QAM16 modulation with a transmission rate of 4 Mbps, the BER is  $10^{-1}$ , far too high to be practically useful. However, with an SNR of 20 dB, QAM16 modulation has a transmission rate of 4 Mbps and a BER of  $10^{-7}$ , while BPSK modulation has a transmission rate of only 1 Mbps and a BER that is so low as to be (literally) “off the charts.” If one can tolerate a BER of  $10^{-7}$ , the higher transmission rate offered by QAM16 would make it the preferred modulation technique in this situation. These considerations give rise to the final characteristic, described next.
- Dynamic selection of the physical-layer modulation technique can be used to adapt the modulation technique to channel conditions. The SNR (and hence the BER) may change as a result of mobility or due to changes in the environment. Adaptive modulation and coding are used in the 802.11 WiFi and in 4G and 5G cellular data networks that we'll study in Sections 7.3 and 7.4. This allows, for example, the selection of a modulation technique that provides the highest transmission rate possible subject to a constraint on the BER, for given channel characteristics.

A higher and time-varying bit error rate is not the only difference between a wired and wireless link. Recall that in the case of wired broadcast links, all nodes

receive the transmissions from all other nodes. In the case of wireless links, the situation is not as simple, as shown in Figure 7.4. Suppose that Station A is transmitting to Station B. Suppose also that Station C is transmitting to Station B. With the so-called **hidden terminal problem**, physical obstructions in the environment (for example, a mountain or a building) may prevent A and C from hearing each other's transmissions, even though A's and C's transmissions are indeed interfering at the destination, B. This is shown in Figure 7.4(a). A second scenario that results in undetectable collisions at the receiver results from the **fading** of a signal's strength as it propagates through the wireless medium. Figure 7.4(b) illustrates the case where A and C are placed such that their signals are not strong enough to detect each other's transmissions, yet their signals *are* strong enough to interfere with each other at station B. As we'll see in Section 7.3, the hidden terminal problem and fading make multiple access in a wireless network considerably more complex than in a wired network.

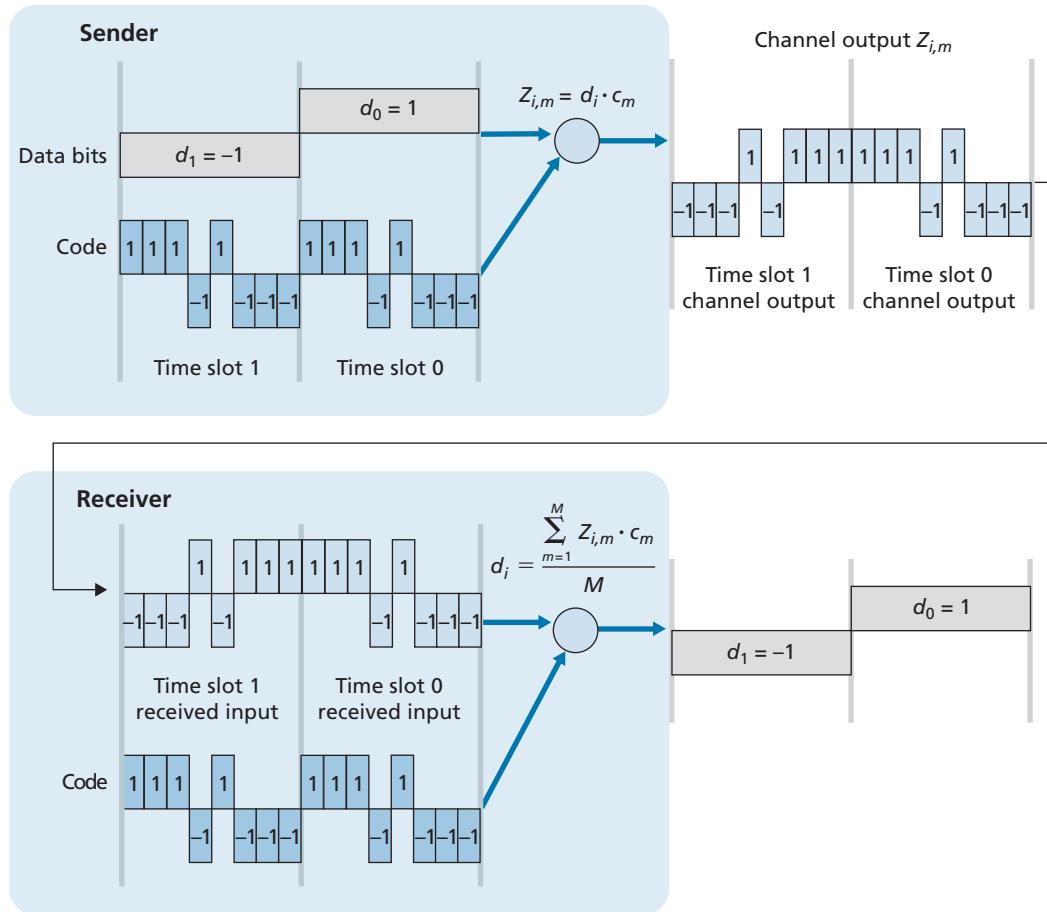
### 7.2.1 CDMA

Recall from Chapter 6 that when hosts communicate over a shared medium, a protocol is needed so that the signals sent by multiple senders do not interfere at the receivers. In Chapter 6, we described three classes of medium access protocols: channel partitioning, random access, and taking turns. Code division multiple access (CDMA) belongs to the family of channel partitioning protocols. It is prevalent in wireless LAN and cellular technologies. Because CDMA is so important in the wireless world, we'll take a quick look at CDMA now, before getting into specific wireless access technologies in the subsequent sections.

In a CDMA protocol, each bit being sent is encoded by multiplying the bit by a signal (the code) that changes at a much faster rate (known as the **chipping rate**) than the original sequence of data bits. Figure 7.5 shows a simple, idealized CDMA encoding/decoding scenario. Suppose that the rate at which original data bits reach the CDMA encoder defines the unit of time; that is, each original data bit to be transmitted requires a one-bit slot time. Let  $d_i$  be the value of the data bit for the  $i$ th bit slot. For mathematical convenience, we represent a data bit with a 0 value as  $-1$ . Each bit slot is further subdivided into  $M$  mini-slots; in Figure 7.5,  $M = 8$ , although in practice  $M$  is much larger. The CDMA code used by the sender consists of a sequence of  $M$  values,  $c_m$ ,  $m = 1, \dots, M$ , each taking a  $+1$  or  $-1$  value. In the example in Figure 7.5, the  $M$ -bit CDMA code being used by the sender is  $(1, 1, 1, -1, 1, -1, -1, -1)$ .

To illustrate how CDMA works, let us focus on the  $i$ th data bit,  $d_i$ . For the  $m$ th mini-slot of the bit-transmission time of  $d_i$ , the output of the CDMA encoder,  $Z_{i,m}$ , is the value of  $d_i$  multiplied by the  $m$ th bit in the assigned CDMA code,  $c_m$ :

$$Z_{i,m} = d_i \cdot c_m \quad (7.1)$$



**Figure 7.5** ♦ A simple CDMA example: Sender encoding, receiver decoding

In a simple world, with no interfering senders, the receiver would receive the encoded bits,  $Z_{i,m}$ , and recover the original data bit,  $d_i$ , by computing:

$$d_i = \frac{1}{M} \sum_{m=1}^M Z_{i,m} \cdot c_m \quad (7.2)$$

The reader might want to work through the details of the example in Figure 7.5 to see that the original data bits are indeed correctly recovered at the receiver using Equation 7.2.

The world is far from ideal, however, and as noted above, CDMA must work in the presence of interfering senders that are encoding and transmitting their data using a different assigned code. But how can a CDMA receiver recover a sender's original data bits when those data bits are being tangled with bits being transmitted by other senders? CDMA works under the assumption that the interfering transmitted bit signals are additive. This means, for example, that if three senders send a 1 value, and a fourth sender sends a  $-1$  value during the same mini-slot, then the received signal at all receivers during that mini-slot is a 2 (since  $1 + 1 + 1 - 1 = 2$ ). In the presence of multiple senders, sender  $s$  computes its encoded transmissions,  $Z_{i,m}^s$ , in exactly the same manner as in Equation 7.1. The value received at a receiver during the  $m$ th mini-slot of the  $i$ th bit slot, however, is now the *sum* of the transmitted bits from all  $N$  senders during that mini-slot:

$$Z_{i,m}^* = \sum_{s=1}^N Z_{i,m}^s$$

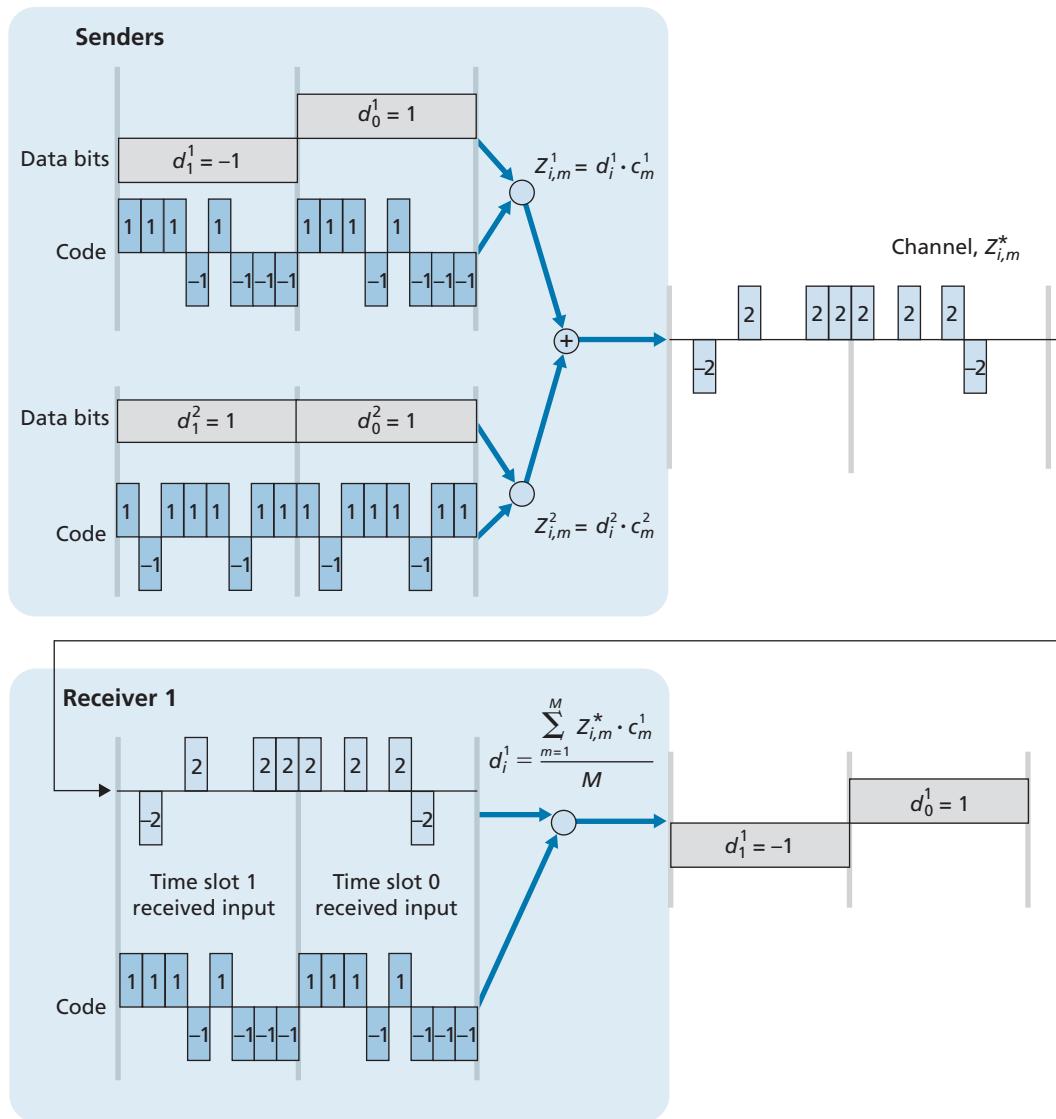
Amazingly, if the senders' codes are chosen carefully, each receiver can recover the data sent by a given sender out of the aggregate signal simply by using the sender's code in exactly the same manner as in Equation 7.2:

$$d_i = \frac{1}{M} \sum_{m=1}^M Z_{i,m}^* \cdot c_m \quad (7.3)$$

as shown in Figure 7.6, for a two-sender CDMA example. The  $M$ -bit CDMA code being used by the upper sender is  $(1, 1, 1, -1, 1, -1, -1, -1)$ , while the CDMA code being used by the lower sender is  $(1, -1, 1, 1, 1, -1, 1, 1)$ . Figure 7.6 illustrates a receiver recovering the original data bits from the upper sender. Note that the receiver is able to extract the data from sender 1 in spite of the interfering transmission from sender 2.

Recall our cocktail analogy from Chapter 6. A CDMA protocol is similar to having partygoers speaking in multiple languages; in such circumstances humans are actually quite good at locking into the conversation in the language they understand, while filtering out the remaining conversations. We see here that CDMA is a partitioning protocol in that it partitions the codespace (as opposed to time or frequency) and assigns each node a dedicated piece of the codespace.

Our discussion here of CDMA is necessarily brief; in practice a number of difficult issues must be addressed. First, in order for the CDMA receivers to be able to extract a particular sender's signal, the CDMA codes must be carefully chosen. Second, our discussion has assumed that the received signal strengths from various senders are the same; in reality, this can be difficult to achieve. There is a considerable body of literature addressing these and other issues related to CDMA; see [Pickholtz 1982; Viterbi 1995] for details.



**Figure 7.6** ♦ A two-sender CDMA example

### 7.3 WiFi: 802.11 Wireless LANs

Pervasive in the workplace, the home, educational institutions, cafés, airports, and street corners, wireless LANs are now one of the most important access network technologies in the Internet today. Although many technologies and standards for

wireless LANs were developed in the 1990s, one particular class of standards has clearly emerged as the winner: the **IEEE 802.11 wireless LAN**, also known as **WiFi**. In this section, we'll take a close look at 802.11 wireless LANs, examining its frame structure, its medium access protocol, and its internetworking of 802.11 LANs with wired Ethernet LANs.

As summarized in Table 7.1, there are several 802.11 standards [IEEE 802.11 2020]. The 802.11 b, g, n, ac, ax are successive generations of 802.11 technology aimed for wireless local area networks (WLANs), typically less than 70 m range in a home office, workplace, or business setting. The 802.11 n, ac, and ax standards have recently been branded as WiFi 4, 5 and 6, respectively—no doubt competing with 4G and 5G cellular network branding. The 802.11 af, ah standards operate over longer distances and are aimed at Internet of Things, sensor networks, and metering applications.

The different 802.11 b, g, n, ac, ax standards all share some common characteristics, including the 802.11 frame format that we will study shortly, and are backward compatible, meaning, for example, that a mobile capable only of 802.11 g may still interact with a newer 802.11 ac or 802.11 ax base station. They also all use the same medium access protocol, CSMA/CA, which we'll also discuss shortly, while also 802.11 ax also supports centralized scheduling by the base station of transmissions from associated wireless devices.

However, as shown in Table 7.1, the standards have some major differences at the physical layer. 802.11 devices operate in two different frequency ranges: 2.4–2.485 GHz (referred to as the 2.4 GHz range) and 5.1–5.8 GHz (referred to as the 5 GHz range). The 2.4 GHz range is an unlicensed frequency band, where 802.11 devices may compete for frequency spectrum with 2.4 GHz phones and appliances such as microwave ovens. At 5 GHz, 802.11 LANs have a shorter transmission distance for a given power level and suffer more from multipath propagation. The 802.11n, 802.11ac, and 802.11ax standards use multiple input multiple-output (MIMO) antennas; that is, two or more antennas on the sending side and two or more antennas on the receiving side that are transmitting/receiving different signals

| IEEE 802.11 standard | Year            | Max data rate | Range | Frequency                    |
|----------------------|-----------------|---------------|-------|------------------------------|
| 802.11 b             | 1999            | 11 Mbps       | 30 m  | 2.4 Ghz                      |
| 802.11 g             | 2003            | 54 Mbps       | 30 m  | 2.4 Ghz                      |
| 802.11 n (WiFi 4)    | 2009            | 600           | 70 m  | 2.4, 5 Ghz                   |
| 802.11 ac (WiFi 5)   | 2013            | 3.47 Gbps     | 70 m  | 5 Ghz                        |
| 802.11 ax (WiFi 6)   | 2020 (expected) | 14 Gbps       | 70 m  | 2.4, 5 Ghz                   |
| 802.11 af            | 2014            | 35–560 Mbps   | 1 Km  | unused TV bands (54–790 MHz) |
| 802.11 ah            | 2017            | 347 Mbps      | 1 Km  | 900 Mhz                      |

**Table 7.1** ◆ Summary of IEEE 802.11 standards

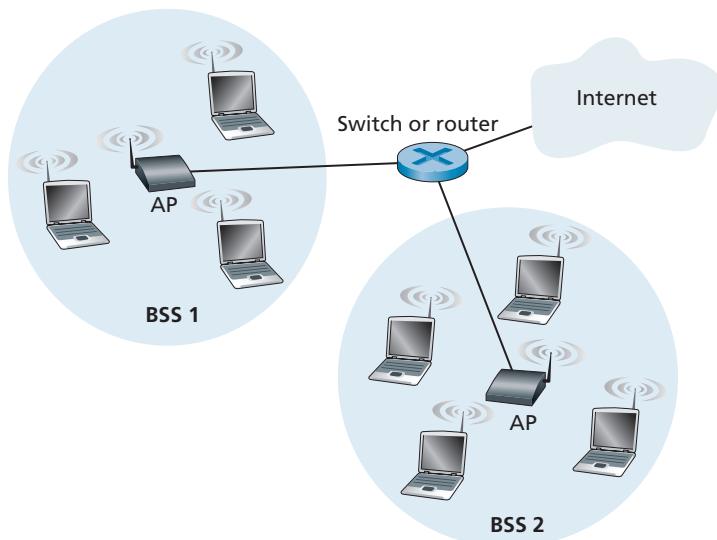
[Diggavi 2004]. 802.11ac and 802.11 ax base stations may transmit to multiple stations simultaneously, and use “smart” antennas to adaptively beamform to target transmissions in the direction of a receiver. This decreases interference and increases the distance reached at a given data rate. The data rates shown in Table 7.1 are for an idealized environment, for example, a receiver close to the base station, with no interference—a scenario that we’re unlikely to experience in practice! So as the saying goes, YMMV: Your Mileage (or in this case your wireless data rate) May Vary.

### 7.3.1 The 802.11 Wireless LAN Architecture

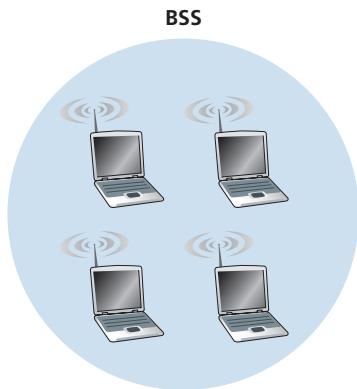
Figure 7.7 illustrates the principal components of the 802.11 wireless LAN architecture. The fundamental building block of the 802.11 architecture is the **basic service set (BSS)**. A BSS contains one or more wireless stations and a central **base station**, known as an **access point (AP)** in 802.11 parlance. Figure 7.7 shows the AP in each of two BSSs connecting to an interconnection device (such as a switch or router), which in turn leads to the Internet. In a typical home network, there is one AP and one router (typically integrated together as one unit) that connects the BSS to the Internet.

As with Ethernet devices, each 802.11 wireless station has a 6-byte MAC address that is stored in the firmware of the station’s adapter (that is, 802.11 network interface card). Each AP also has a MAC address for its wireless interface. As with Ethernet, these MAC addresses are administered by IEEE and are (in theory) globally unique.

As noted in Section 7.1, wireless LANs that deploy APs are often referred to as **infrastructure wireless LANs**, with the “infrastructure” being the APs along with the



**Figure 7.7** ♦ IEEE 802.11 LAN architecture



**Figure 7.8** ♦ An IEEE 802.11 ad hoc network

wired Ethernet infrastructure that interconnects the APs and a router. Figure 7.8 shows that IEEE 802.11 stations can also group themselves together to form an ad hoc network—a network with no central control and with no connections to the “outside world.” Here, the network is formed “on the fly,” by mobile devices that have found themselves in proximity to each other, that have a need to communicate, and that find no preexisting network infrastructure in their location. An ad hoc network might be formed when people with laptops get together (e.g., in a conference room, a train, or a car) and want to exchange data in the absence of a centralized AP. There has been tremendous interest in ad hoc networking, as communicating portable devices continue to proliferate. In this section, though, we’ll focus our attention on infrastructure wireless LANs.

### Channels and Association

In 802.11, each wireless station needs to associate with an AP before it can send or receive network-layer data. Although all of the 802.11 standards use association, we’ll discuss this topic specifically in the context of IEEE 802.11b, g, n, ac, ax.

When a network administrator installs an AP, the administrator assigns a one- or two-word **Service Set Identifier (SSID)** to the access point. (When you choose Wi-Fi under Setting on your iPhone, for example, a list is displayed showing the SSID of each AP in range.) The administrator must also assign a channel number to the AP. To understand channel numbers, recall that 802.11 operates in the frequency range of 2.4 GHz to 2.4835 GHz. Within this 85 MHz band, 802.11 defines 11 partially overlapping channels. Any two channels are non-overlapping if and only if they are separated by four or more channels. In particular, the set of channels 1, 6, and 11 is the only set of three non-overlapping channels. This means that an administrator could create a wireless LAN with an aggregate maximum transmission rate of three times the maximum transmission rate shown in Table 7.1 by installing three 802.11 APs at the same physical location, assigning channels 1, 6, and 11 to the APs, and interconnecting each of the APs with a switch.

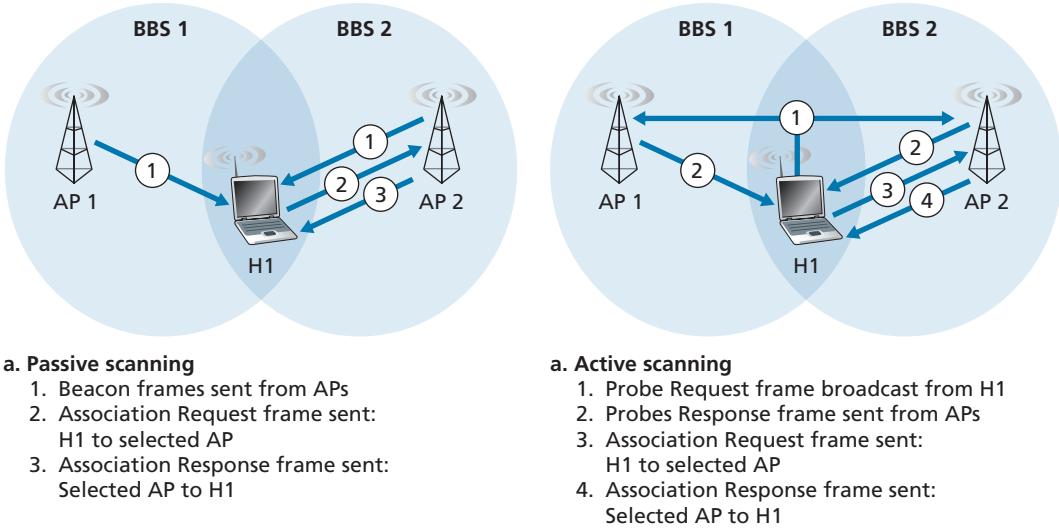
Now that we have a basic understanding of 802.11 channels, let's describe an interesting (and not completely uncommon) situation—that of a WiFi jungle. A **WiFi jungle** is any physical location where a wireless station receives a sufficiently strong signal from two or more APs. For example, in many cafés in New York City, a wireless station can pick up a signal from numerous nearby APs. One of the APs might be managed by the café, while the other APs might be in residential apartments near the café. Each of these APs would likely be located in a different IP subnet and would have been independently assigned a channel.

Now suppose you enter such a WiFi jungle with your smartphone, tablet, or laptop, seeking wireless Internet access and a blueberry muffin. Suppose there are five APs in the WiFi jungle. To gain Internet access, your wireless device needs to join exactly one of the subnets and hence needs to **associate** with exactly one of the APs. Associating means the wireless device creates a virtual wire between itself and the AP. Specifically, only the associated AP will send data frames (that is, frames containing data, such as a datagram) to your wireless device, and your wireless device will send data frames into the Internet only through the associated AP. But how does your wireless device associate with a particular AP? And more fundamentally, how does your wireless device know which APs, if any, are out there in the jungle?

The 802.11 standard requires that an AP periodically send **beacon frames**, each of which includes the AP's SSID and MAC address. Your wireless device, knowing that APs are sending out beacon frames, scans the 11 channels, seeking beacon frames from any APs that may be out there (some of which may be transmitting on the same channel—it's a jungle out there!). Having learned about available APs from the beacon frames, you (or your wireless device) select one of the APs for association.

The 802.11 standard does not specify an algorithm for selecting which of the available APs to associate with; that algorithm is left up to the designers of the 802.11 firmware and software in your wireless device. Typically, the device chooses the AP whose beacon frame is received with the highest signal strength. While a high signal strength is good (see, e.g., Figure 7.3), signal strength is not the only AP characteristic that will determine the performance a device receives. In particular, it's possible that the selected AP may have a strong signal, but may be overloaded with other affiliated devices (that will need to share the wireless bandwidth at that AP), while an unloaded AP is not selected due to a slightly weaker signal. A number of alternative ways of choosing APs have thus recently been proposed [Vasudevan 2005; Nicholson 2006; Sundaresan 2006]. For an interesting and down-to-earth discussion of how signal strength is measured, see [Bardwell 2004].

The process of scanning channels and listening for beacon frames is known as **passive scanning** (see Figure 7.9a). A wireless device can also perform **active scanning**, by broadcasting a probe frame that will be received by all APs within the wireless device's range, as shown in Figure 7.9b. APs respond to the probe request frame with a probe response frame. The wireless device can then choose the AP with which to associate from among the responding APs.



**Figure 7.9** ♦ Active and passive scanning for access points

After selecting the AP with which to associate, the wireless device sends an association request frame to the AP, and the AP responds with an association response frame. Note that this second request/response handshake is needed with active scanning, since an AP responding to the initial probe request frame doesn't know which of the (possibly many) responding APs the device will choose to associate with, in much the same way that a DHCP client can choose from among multiple DHCP servers (see Figure 4.21). Once associated with an AP, the device will want to join the subnet (in the IP addressing sense of Section 4.3.3) to which the AP belongs. Thus, the device will typically send a DHCP discovery message (see Figure 4.21) into the subnet via the AP in order to obtain an IP address on the subnet. Once the address is obtained, the rest of the world then views that device simply as another host with an IP address in that subnet.

In order to create an association with a particular AP, the wireless device may be required to authenticate itself to the AP. 802.11 wireless LANs provide a number of alternatives for authentication and access. One approach, used by many companies, is to permit access to a wireless network based on a device's MAC address. A second approach, used by many Internet cafés, employs usernames and passwords. In both cases, the AP typically communicates with an authentication server, relaying information between the wireless device and the authentication server using a protocol such as RADIUS [RFC 2865] or DIAMETER [RFC 6733]. Separating the authentication server from the AP allows one authentication server to serve many APs, centralizing the (often sensitive) decisions of authentication and access within the single server, and keeping

AP costs and complexity low. We'll see in chapter 8 that the new IEEE 802.11i protocol defining security aspects of the 802.11 protocol family takes precisely this approach.

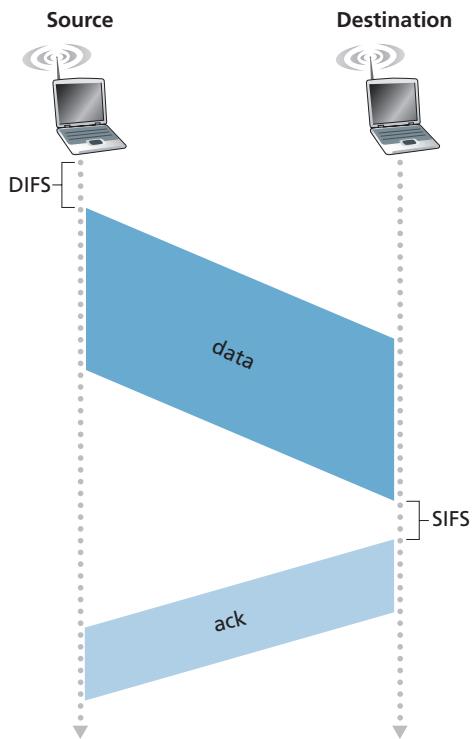
### 7.3.2 The 802.11 MAC Protocol

Once a wireless device is associated with an AP, it can start sending and receiving data frames to and from the access point. But because multiple wireless devices, or the AP itself may want to transmit data frames at the same time over the same channel, a multiple access protocol is needed to coordinate the transmissions. In the following, we'll refer to the devices or the AP as wireless "stations" that share the multiple access channel. As discussed in Chapter 6 and Section 7.2.1, broadly speaking there are three classes of multiple access protocols: channel partitioning (including CDMA), random access, and taking turns. Inspired by the huge success of Ethernet and its random access protocol, the designers of 802.11 chose a random access protocol for 802.11 wireless LANs. This random access protocol is referred to as **CSMA with collision avoidance**, or more succinctly as **CSMA/CA**. As with Ethernet's CSMA/CD, the "CSMA" in CSMA/CA stands for "carrier sense multiple access," meaning that each station senses the channel before transmitting, and refrains from transmitting when the channel is sensed busy. Although both Ethernet and 802.11 use carrier-sensing random access, the two MAC protocols have important differences. First, instead of using collision detection, 802.11 uses collision-avoidance techniques. Second, because of the relatively high bit error rates of wireless channels, 802.11 (unlike Ethernet) uses a link-layer acknowledgment/retransmission (ARQ) scheme. We'll describe 802.11's collision-avoidance and link-layer acknowledgment schemes below.

Recall from Sections 6.3.2 and 6.4.2 that with Ethernet's collision-detection algorithm, an Ethernet station listens to the channel as it transmits. If, while transmitting, it detects that another station is also transmitting, it aborts its transmission and tries to transmit again after waiting a small, random amount of time. Unlike the 802.3 Ethernet protocol, the 802.11 MAC protocol does *not* implement collision detection. There are two important reasons for this:

- The ability to detect collisions requires the ability to send (the station's own signal) and receive (to determine whether another station is also transmitting) at the same time. Because the strength of the received signal is typically very small compared to the strength of the transmitted signal at the 802.11 adapter, it is costly to build hardware that can detect a collision.
- More importantly, even if the adapter could transmit and listen at the same time (and presumably abort transmission when it senses a busy channel), the adapter would still not be able to detect all collisions, due to the hidden terminal problem and fading, as discussed in Section 7.2.

Because 802.11 wireless LANs do not use collision detection, once a station begins to transmit a frame, *it transmits the frame in its entirety*; that is, once a station



**Figure 7.10** ♦ 802.11 uses link-layer acknowledgments

gets started, there is no turning back. As one might expect, transmitting entire frames (particularly long frames) when collisions are prevalent can significantly degrade a multiple access protocol's performance. In order to reduce the likelihood of collisions, 802.11 employs several collision-avoidance techniques, which we'll shortly discuss.

Before considering collision avoidance, however, we'll first need to examine 802.11's **link-layer acknowledgment** scheme. Recall from Section 7.2 that when a station in a wireless LAN sends a frame, the frame may not reach the destination station intact for a variety of reasons. To deal with this non-negligible chance of failure, the 802.11 MAC protocol uses link-layer acknowledgments. As shown in Figure 7.10, when the destination station receives a frame that passes the CRC, it waits a short period of time known as the **Short Inter-frame Spacing (SIFS)** and then sends back an acknowledgment frame. If the transmitting station does not receive an acknowledgment within a given amount of time, it assumes that an error has occurred and retransmits the frame, using the CSMA/CA protocol to access the channel. If an acknowledgment is not received after some fixed number of retransmissions, the transmitting station gives up and discards the frame.

Having discussed how 802.11 uses link-layer acknowledgments, we’re now in a position to describe the 802.11 CSMA/CA protocol. Suppose that a station (wireless device or an AP) has a frame to transmit.

1. If initially the station senses the channel idle, it transmits its frame after a short period of time known as the **Distributed Inter-frame Space (DIFS)**; see Figure 7.10.
2. Otherwise, the station chooses a random backoff value using binary exponential backoff (as we encountered in Section 6.3.2) and counts down this value after DIFS when the channel is sensed idle. While the channel is sensed busy, the counter value remains frozen.
3. When the counter reaches zero (note that this can only occur while the channel is sensed idle), the station transmits the entire frame and then waits for an acknowledgment.
4. If an acknowledgment is received, the transmitting station knows that its frame has been correctly received at the destination station. If the station has another frame to send, it begins the CSMA/CA protocol at step 2. If the acknowledgment isn’t received, the transmitting station reenters the backoff phase in step 2, with the random value chosen from a larger interval.

Recall that under Ethernet’s CSMA/CD, multiple access protocol (Section 6.3.2), a station begins transmitting as soon as the channel is sensed idle. With CSMA/CA, however, the station refrains from transmitting while counting down, even when it senses the channel to be idle. Why do CSMA/CD and CDMA/CA take such different approaches here?

To answer this question, let’s consider a scenario in which two stations each have a data frame to transmit, but neither station transmits immediately because each senses that a third station is already transmitting. With Ethernet’s CSMA/CD, the two stations would each transmit as soon as they detect that the third station has finished transmitting. This would cause a collision, which isn’t a serious issue in CSMA/CD, since both stations would abort their transmissions and thus avoid the useless transmissions of the remainders of their frames. In 802.11, however, the situation is quite different. Because 802.11 does not detect a collision and abort transmission, a frame suffering a collision will be transmitted in its entirety. The goal in 802.11 is thus to avoid collisions whenever possible. In 802.11, if the two stations sense the channel busy, they both immediately enter random backoff, hopefully choosing different backoff values. If these values are indeed different, once the channel becomes idle, one of the two stations will begin transmitting before the other, and (if the two stations are not hidden from each other) the “losing station” will hear the “winning station’s” signal, freeze its counter, and refrain from transmitting until the winning station has completed its transmission. In this manner, a costly collision is avoided. Of course, collisions can still occur with 802.11 in this scenario: The two stations could be hidden from each other, or the two stations could choose random

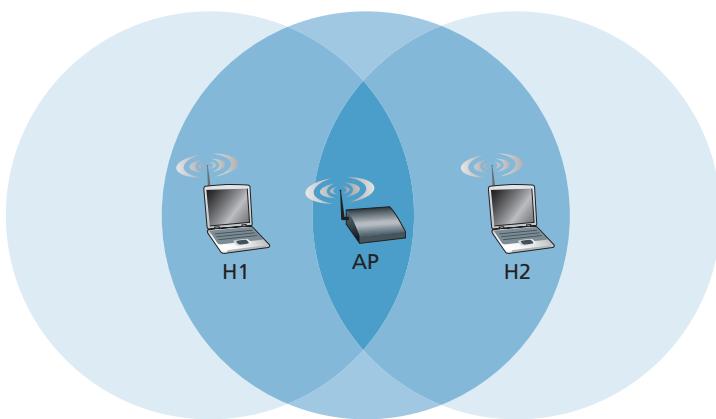
backoff values that are close enough that the transmission from the station starting first have yet to reach the second station. Recall that we encountered this problem earlier in our discussion of random access algorithms in the context of Figure 6.12.

### Dealing with Hidden Terminals: RTS and CTS

The 802.11 MAC protocol also includes a nifty (but optional) reservation scheme that helps avoid collisions even in the presence of hidden terminals. Let's investigate this scheme in the context of Figure 7.11, which shows two wireless stations and one access point. Both of the wireless stations are within range of the AP (whose coverage is shown as a shaded circle) and both have associated with the AP. However, due to fading, the signal ranges of wireless stations are limited to the interiors of the shaded circles shown in Figure 7.11. Thus, each of the wireless stations is hidden from the other, although neither is hidden from the AP.

Let's now consider why hidden terminals can be problematic. Suppose Station H1 is transmitting a frame and halfway through H1's transmission, Station H2 wants to send a frame to the AP. H2, not hearing the transmission from H1, will first wait a DIFS interval and then transmit the frame, resulting in a collision. The channel will therefore be wasted during the entire period of H1's transmission as well as during H2's transmission.

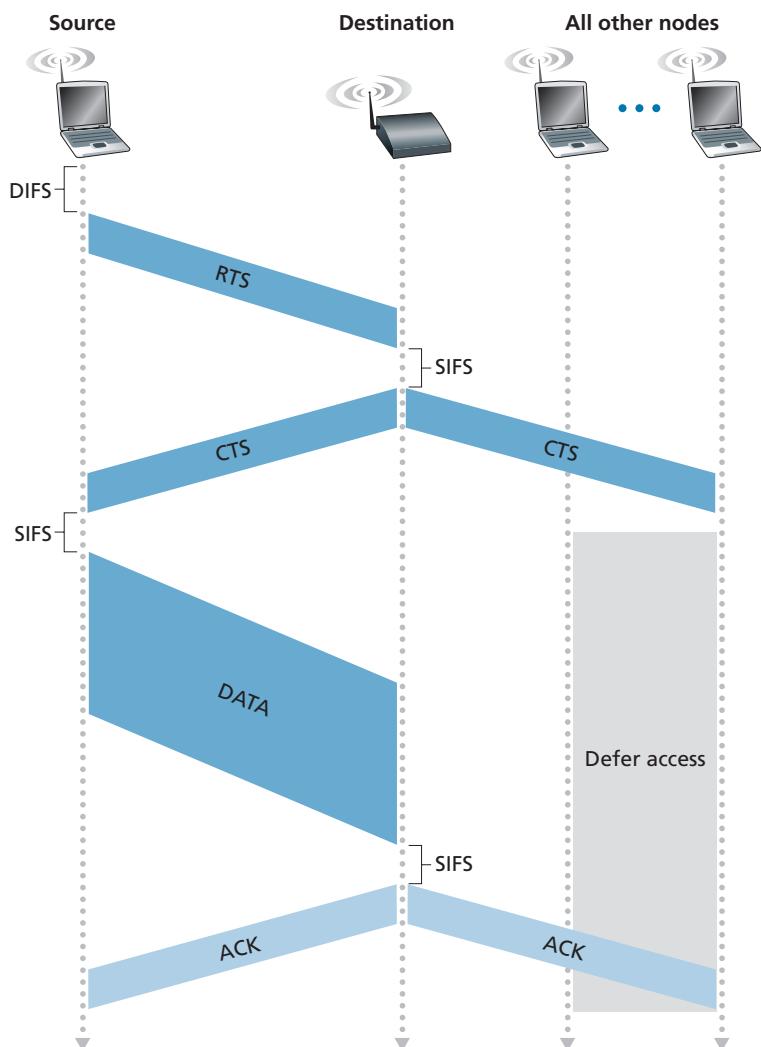
In order to avoid this problem, the IEEE 802.11 protocol allows a station to use a short **Request to Send (RTS)** control frame and a short **Clear to Send (CTS)** control frame to *reserve* access to the channel. When a sender wants to send a DATA frame, it can first send an RTS frame to the AP, indicating the total time required to transmit the DATA frame and the acknowledgment (ACK) frame. When the AP receives the RTS frame, it responds by broadcasting a CTS frame. This CTS frame



**Figure 7.11** ♦ Hidden terminal example: H1 is hidden from H2, and vice versa

serves two purposes: It gives the sender explicit permission to send and also instructs the other stations not to send for the reserved duration.

Thus, in Figure 7.12, before transmitting a DATA frame, H1 first broadcasts an RTS frame, which is heard by all stations in its circle, including the AP. The AP then responds with a CTS frame, which is heard by all stations within its range, including H1 and H2. Station H2, having heard the CTS, refrains from transmitting for the time specified in the CTS frame. The RTS, CTS, DATA, and ACK frames are shown in Figure 7.12.



**Figure 7.12** ♦ Collision avoidance using the RTS and CTS frames

The use of the RTS and CTS frames can improve performance in two important ways:

- The hidden station problem is mitigated, since a long DATA frame is transmitted only after the channel has been reserved.
- Because the RTS and CTS frames are short, a collision involving an RTS or CTS frame will last only for the duration of the short RTS or CTS frame. Once the RTS and CTS frames are correctly transmitted, the following DATA and ACK frames should be transmitted without collisions.

You are encouraged to check out the 802.11 animation in the textbook's Web site. This interactive animation illustrates the CSMA/CA protocol, including the RTS/CTS exchange sequence.

Although the RTS/CTS exchange can help reduce collisions, it also introduces delay and consumes channel resources. For this reason, the RTS/CTS exchange is only used (if at all) to reserve the channel for the transmission of a long DATA frame. In practice, each wireless station can set an RTS threshold such that the RTS/CTS sequence is used only when the frame is longer than the threshold. For many wireless stations, the default RTS threshold value is larger than the maximum frame length, so the RTS/CTS sequence is skipped for all DATA frames sent.

### Using 802.11 as a Point-to-Point Link

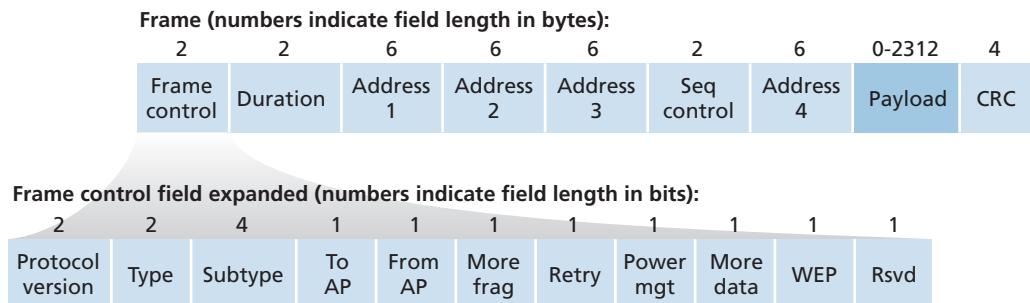
Our discussion so far has focused on the use of 802.11 in a multiple access setting. We should mention that if two nodes each have a directional antenna, they can point their directional antennas at each other and run the 802.11 protocol over what is essentially a point-to-point link. Given the low cost of commodity 802.11 hardware, the use of directional antennas and an increased transmission power allow 802.11 to be used as an inexpensive means of providing wireless point-to-point connections over tens of kilometers distance. [Raman 2007] describes one of the first such multi-hop wireless networks, operating in the rural Ganges plains in India using point-to-point 802.11 links.

### 7.3.3 The IEEE 802.11 Frame

Although the 802.11 frame shares many similarities with an Ethernet frame, it also contains a number of fields that are specific to its use for wireless links. The 802.11 frame is shown in Figure 7.13. The numbers above each of the fields in the frame represent the lengths of the fields in *bytes*; the numbers above each of the subfields in the frame control field represent the lengths of the subfields in *bits*. Let's now examine the fields in the frame as well as some of the more important subfields in the frame's control field.

#### Payload and CRC Fields

At the heart of the frame is the payload, which typically consists of an IP datagram or an ARP packet. Although the field is permitted to be as long as 2,312 bytes, it is



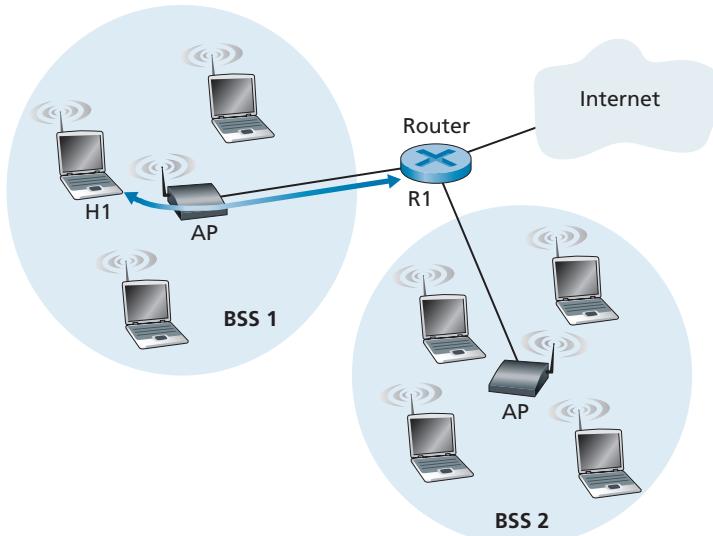
**Figure 7.13** ♦ The 802.11 frame

typically fewer than 1,500 bytes, holding an IP datagram or an ARP packet. As with an Ethernet frame, an 802.11 frame includes a 32-bit cyclic redundancy check (CRC) so that the receiver can detect bit errors in the received frame. As we've seen, bit errors are much more common in wireless LANs than in wired LANs, so the CRC is even more useful here.

### Address Fields

Perhaps the most striking difference in the 802.11 frame is that it has *four* address fields, each of which can hold a 6-byte MAC address. But why four address fields? Doesn't a source MAC field and destination MAC field suffice, as they do for Ethernet? It turns out that three address fields are needed for internetworking purposes—specifically, for moving the network-layer datagram from a wireless station through an AP to a router interface. The fourth address field is used when APs forward frames to each other in ad hoc mode. Since we are only considering infrastructure networks here, let's focus our attention on the first three address fields. The 802.11 standard defines these fields as follows:

- Address 2 is the MAC address of the station that transmits the frame. Thus, if a wireless station transmits the frame, that station's MAC address is inserted in the address 2 field. Similarly, if an AP transmits the frame, the AP's MAC address is inserted in the address 2 field.
- Address 1 is the MAC address of the wireless station that is to receive the frame. Thus if a mobile wireless station transmits the frame, address 1 contains the MAC address of the destination AP. Similarly, if an AP transmits the frame, address 1 contains the MAC address of the destination wireless station.
- To understand address 3, recall that the BSS (consisting of the AP and wireless stations) is part of a subnet, and that this subnet connects to other subnets via some router interface. Address 3 contains the MAC address of this router interface.



**Figure 7.14** ♦ The use of address fields in 802.11 frames: Sending frames between H1 and R1

To gain further insight into the purpose of address 3, let's walk through an inter-networking example in the context of Figure 7.14. In this figure, there are two APs, each of which is responsible for a number of wireless stations. Each of the APs has a direct connection to a router, which in turn connects to the global Internet. We should keep in mind that an AP is a link-layer device, and thus neither "speaks" IP nor understands IP addresses. Consider now moving a datagram from the router interface R1 to the wireless Station H1. The router is not aware that there is an AP between it and H1; from the router's perspective, H1 is just a host in one of the subnets to which it (the router) is connected.

- The router, which knows the IP address of H1 (from the destination address of the datagram), uses ARP to determine the MAC address of H1, just as in an ordinary Ethernet LAN. After obtaining H1's MAC address, router interface R1 encapsulates the datagram within an Ethernet frame. The source address field of this frame contains R1's MAC address, and the destination address field contains H1's MAC address.
- When the Ethernet frame arrives at the AP, the AP converts the 802.3 Ethernet frame to an 802.11 frame before transmitting the frame into the wireless channel. The AP fills in address 1 and address 2 with H1's MAC address and its own MAC address, respectively, as described above. For address 3, the AP inserts the MAC address of R1. In this manner, H1 can determine (from address 3) the MAC address of the router interface that sent the datagram into the subnet.

Now consider what happens when the wireless station H1 responds by moving a datagram from H1 to R1.

- H1 creates an 802.11 frame, filling the fields for address 1 and address 2 with the AP's MAC address and H1's MAC address, respectively, as described above. For address 3, H1 inserts R1's MAC address.
- When the AP receives the 802.11 frame, it converts the frame to an Ethernet frame. The source address field for this frame is H1's MAC address, and the destination address field is R1's MAC address. Thus, address 3 allows the AP to determine the appropriate destination MAC address when constructing the Ethernet frame.

In summary, address 3 plays a crucial role for internetworking the BSS with a wired LAN.

### Sequence Number, Duration, and Frame Control Fields

Recall that in 802.11, whenever a station correctly receives a frame from another station, it sends back an acknowledgment. Because acknowledgments can get lost, the sending station may send multiple copies of a given frame. As we saw in our discussion of the rdt2.1 protocol (Section 3.4.1), the use of sequence numbers allows the receiver to distinguish between a newly transmitted frame and the retransmission of a previous frame. The sequence number field in the 802.11 frame thus serves exactly the same purpose here at the link layer as it did in the transport layer in Chapter 3.

Recall that the 802.11 protocol allows a transmitting station to reserve the channel for a period of time that includes the time to transmit its data frame and the time to transmit an acknowledgment. This duration value is included in the frame's duration field (both for data frames and for the RTS and CTS frames).

As shown in Figure 7.13, the frame control field includes many subfields. We'll say just a few words about some of the more important subfields; for a more complete discussion, you are encouraged to consult the 802.11 specification [Held 2001; Crow 1997; IEEE 802.11 1999]. The *type* and *subtype* fields are used to distinguish the association, RTS, CTS, ACK, and data frames. The *to* and *from* fields are used to define the meanings of the different address fields. (These meanings change depending on whether ad hoc or infrastructure modes are used and, in the case of infrastructure mode, whether a wireless station or an AP is sending the frame.) Finally the WEP field indicates whether encryption is being used or not (WEP is discussed in Chapter 8).

#### 7.3.4 Mobility in the Same IP Subnet

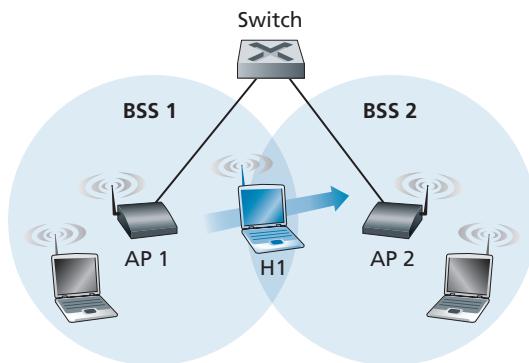
In order to increase the physical range of a wireless LAN, companies and universities will often deploy multiple BSSs within the same IP subnet. This naturally raises the issue of mobility among the BSSs—how do wireless stations seamlessly move from one BSS to another while maintaining ongoing TCP sessions? As we'll see in this subsection, mobility can be handled in a relatively straightforward manner when the BSSs are part

of the subnet. When stations move between subnets, more sophisticated mobility management protocols will be needed, such as those we'll study in Sections 7.5 and 7.6.

Let's now look at a specific example of mobility between BSSs in the same subnet. Figure 7.15 shows two interconnected BSSs with a host, H1, moving from BSS1 to BSS2. Because in this example the interconnection device that connects the two BSSs is *not* a router, all of the stations in the two BSSs, including the APs, belong to the same IP subnet. Thus, when H1 moves from BSS1 to BSS2, it may keep its IP address and all of its ongoing TCP connections. If the interconnection device were a router, then H1 would have to obtain a new IP address in the subnet in which it was moving. This address change would disrupt (and eventually terminate) any on-going TCP connections at H1. In Section 7.6, we'll see how a network-layer mobility protocol, such as mobile IP, can be used to avoid this problem.

But what specifically happens when H1 moves from BSS1 to BSS2? As H1 wanders away from AP1, H1 detects a weakening signal from AP1 and starts to scan for a stronger signal. H1 receives beacon frames from AP2 (which in many corporate and university settings will have the same SSID as AP1). H1 then disassociates with AP1 and associates with AP2, while keeping its IP address and maintaining its ongoing TCP sessions.

This addresses the handover problem from the host and AP viewpoint. But what about the switch in Figure 7.15? How does it know that the host has moved from one AP to another? As you may recall from Chapter 6, switches are "self-learning" and automatically build their forwarding tables. This self-learning feature nicely handles occasional moves (for example, when an employee gets transferred from one department to another); however, switches were not designed to support highly mobile users who want to maintain TCP connections while moving between BSSs. To appreciate the problem here, recall that before the move, the switch has an entry in its forwarding table that pairs H1's MAC address with the outgoing switch interface through which H1 can be reached. If H1 is initially in BSS1, then a datagram destined to H1 will be directed to H1 via AP1. Once H1 associates with BSS2, however, its frames should be directed to AP2. One solution (a bit of a hack, really) is for AP2 to send a broadcast Ethernet frame with H1's source address to the switch just after



**Figure 7.15** ♦ Mobility in the same subnet

the new association. When the switch receives the frame, it updates its forwarding table, allowing H1 to be reached via AP2. The 802.11f standards group is developing an inter-AP protocol to handle these and related issues.

Our discussion above has focused on mobility with the same LAN subnet. Recall that VLANs, which we studied in Section 6.4.4, can be used to connect together islands of LANs into a large virtual LAN that can span a large geographical region. Mobility among base stations within such a VLAN can be handled in exactly the same manner as above [Yu 2011].



## CASE HISTORY

### LOCATION DISCOVERY: GPS AND WIFI POSITIONING

Many of the most useful and important smartphone apps today are location-based mobile apps, including Foursquare, Yelp, Uber, Pokémon Go, and Waze. These software apps all make use of an API that allows them to extract their current geographical position directly from the smartphone. Have you ever wondered how your smartphone obtains its geographical position? Today, it is done by combining two systems, the **Global Positioning System (GPS)** and the **WiFi Positioning System (WPS)**.

The GPS, with a constellation of 30+ satellites, broadcasts satellite location and timing information, which in turn is used by each GPS receiver to estimate its geolocation. The United States government created the system, maintains it, and makes it freely accessible to anyone with a GPS receiver. The satellites have very stable atomic clocks that are synchronized with one another and with ground clocks. The satellites also know their locations with great precision. Each GPS satellite continuously broadcasts a radio signal containing its current time and position. If a GPS receiver obtains this information from at least four satellites, it can solve triangulation equations to estimate its position.

GPS, however, cannot always provide accurate geolocations if it does not have line-of-sight with at least four GPS satellites or when there is interference from other high-frequency communication systems. This is particularly true in urban environments, where tall buildings frequently block GPS signals. This is where WiFi positioning systems come to the rescue. WiFi positioning systems make use of databases of WiFi access points, which are independently maintained by various Internet companies, including Google, Apple, and Microsoft. Each database contains information about millions of WiFi access points, including each access point's SSID and an estimate of its geographic location. To understand how a WiFi positioning system makes use of such a database, consider an Android smartphone along with the Google location service. From each nearby access point, the smartphone receives and measures the signal strength of beacon signals (see Section 7.3.1), which contain the access point's SSID. The smartphone can therefore continually send messages to the Google location service (in the cloud) that include the SSIDs of nearby access points and the corresponding signal strengths. It will also send its GPS position (obtained via the satellite broadcast

signals, as described above) when available. Using the signal-strength information, Google will estimate the distance between the smartphone and each of the WiFi access points. Leveraging these estimated distances, it can then solve triangulation equations to estimate the smartphone's geolocation. Finally, this WiFi-based estimate is combined with the GPS satellite-based estimate to form an aggregate estimate, which is then sent back to the smartphone and used by the location-based mobile apps.

But you may still be wondering how Google (and Apple, Microsoft, and so on) obtain and maintain the database of access points, and in particular, the access point's geographic location? Recall that for a given access point, every nearby Android smartphone will send to the Google location service the strength of the signal received from the access point as well as the smartphone's estimated location. Given that thousands of smartphones may be passing by the access point during any single day, Google's location service will have *lots* of data at its disposition to use in estimating the access point's position, again by solving triangulation equations. Thus, the access points help the smartphones determine their locations, and in turn the smartphones help the access points determine their locations!

### 7.3.5 Advanced Features in 802.11

We'll wrap up our coverage of 802.11 with a short discussion of two advanced capabilities found in 802.11 networks. As we'll see, these capabilities are *not* completely specified in the 802.11 standard, but rather are made possible by mechanisms specified in the standard. This allows different vendors to implement these capabilities using their own (proprietary) approaches, presumably giving them an edge over the competition.

#### 802.11 Rate Adaptation

We saw earlier in Figure 7.3 that different modulation techniques (with the different transmission rates that they provide) are appropriate for different SNR scenarios. Consider, for example, a mobile 802.11 user who is initially 20 meters away from the base station, with a high signal-to-noise ratio. Given the high SNR, the user can communicate with the base station using a physical-layer modulation technique that provides high transmission rates while maintaining a low BER. This is one happy user! Suppose now that the user becomes mobile, walking away from the base station, with the SNR falling as the distance from the base station increases. In this case, if the modulation technique used in the 802.11 protocol operating between the base station and the user does not change, the BER will become unacceptably high as the SNR decreases, and eventually no transmitted frames will be received correctly.

For this reason, some 802.11 implementations have a rate adaptation capability that adaptively selects the underlying physical-layer modulation technique to use based on current or recent channel characteristics. If a node sends two frames in a row without receiving an acknowledgment (an implicit indication of bit errors on

the channel), the transmission rate falls back to the next lower rate. If 10 frames in a row are acknowledged, or if a timer that tracks the time since the last fallback expires, the transmission rate increases to the next higher rate. This rate adaptation mechanism shares the same “probing” philosophy as TCP’s congestion-control mechanism—when conditions are good (reflected by ACK receipts), the transmission rate is increased until something “bad” happens (the lack of ACK receipts); when something “bad” happens, the transmission rate is reduced. 802.11 rate adaptation and TCP congestion control are thus similar to the young child who is constantly pushing his/her parents for more and more (say candy for a young child, later curfew hours for the teenager) until the parents finally say “Enough!” and the child backs off (only to try again later after conditions have hopefully improved!). A number of other schemes have also been proposed to improve on this basic automatic rate-adjustment scheme [Kamerman 1997; Holland 2001; Lacage 2004].

### Power Management

Power is a precious resource in mobile devices, and thus the 802.11 standard provides power-management capabilities that allow 802.11 nodes to minimize the amount of time that their sense, transmit, and receive functions and other circuitry need to be “on.” 802.11 power management operates as follows. A node is able to explicitly alternate between sleep and wake states (not unlike a sleepy student in a classroom!). A node indicates to the access point that it will be going to sleep by setting the power-management bit in the header of an 802.11 frame to 1. A timer in the node is then set to wake up the node just before the AP is scheduled to send its beacon frame (recall that an AP typically sends a beacon frame every 100 msec). Since the AP knows from the set power-transmission bit that the node is going to sleep, it (the AP) knows that it should not send any frames to that node, and will buffer any frames destined for the sleeping host for later transmission.

A node will wake up just before the AP sends a beacon frame, and quickly enter the fully active state (unlike the sleepy student, this wakeup requires only 250 microseconds [Kamerman 1997]!). The beacon frames sent out by the AP contain a list of nodes whose frames have been buffered at the AP. If there are no buffered frames for the node, it can go back to sleep. Otherwise, the node can explicitly request that the buffered frames be sent by sending a polling message to the AP. With an inter-beacon time of 100 msec, a wakeup time of 250 microseconds, and a similarly small time to receive a beacon frame and check to ensure that there are no buffered frames, a node that has no frames to send or receive can be asleep 99% of the time, resulting in a significant energy savings.

#### 7.3.6 Personal Area Networks: Bluetooth

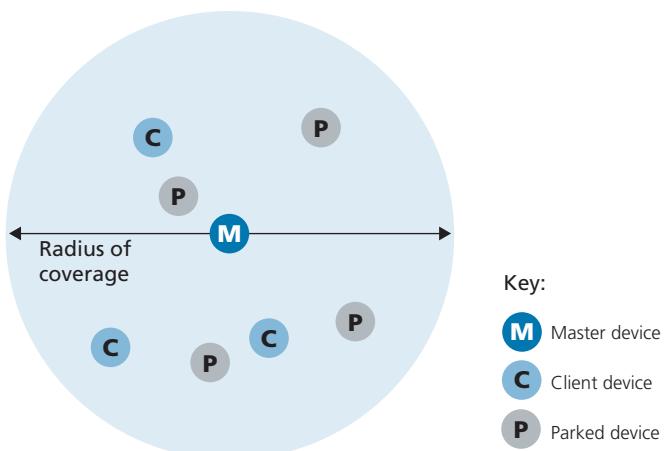
**Bluetooth** networks seem to have quickly become part of everyday life. Perhaps you’ve used a Bluetooth network as a “cable replacement” technology to interconnect

your computer with a wireless keyboard, mouse, or other peripheral device. Or perhaps you've used a Bluetooth network to connect your wireless earbuds, speaker, watch, or health monitoring band to your smartphone or to connect your smartphone to a car's audio system. In all of these cases, Bluetooth operates over short ranges (tens of meters or less), at low power, and at low cost. For this reason, Bluetooth networks are sometimes referred to as **wireless personal area networks (WPANs)** or **piconets**.

Although Bluetooth networks are small and relatively simple by design, they're packed with many of the link-level networking techniques that we've studied earlier including time division multiplexing (TDM) and frequency division (Section 6.3.1), randomized backoff (Section 6.3.2), polling (Section 6.3.3), error detection *and* correction (Section 6.2), reliable data transfer via ACKs and NAKS (Section 3.4.1). And that's just considering Bluetooth's link layer!

Bluetooth networks operate in the unlicensed 2.4 GHz Industrial, Scientific and Medical (ISM) radio band along with other home appliances such as microwaves, garage door openers, and cordless phones. As a result, Bluetooth networks are designed explicitly with noise and interference in mind. The Bluetooth wireless channel is operated in a TDM manner, with time slots of 625 microseconds. During each time slot, a sender transmits on one of 79 channels, with the channel (frequency) changing in a known but pseudo-random manner from slot to slot. This form of channel hopping, known as **frequency-hopping spread spectrum (FHSS)**, is used so that interference from another device or appliance operating in the ISM band will only interfere with Bluetooth communications in at most a subset of the slots. Bluetooth data rates can reach up to 3 Mbps.

Bluetooth networks are ad hoc networks—no network infrastructure (e.g., an access point) is needed. Instead, Bluetooth devices must organize *themselves* into a piconet of up to eight active devices, as shown in Figure 7.16. One of these devices



**Figure 7.16** ♦ A Bluetooth piconet

is designated as the master, with the remaining devices acting as clients. The master node truly rules the piconet—its clock determines time in the piconet (e.g., determines TDM slot boundaries), it determine the slot-to-slot frequency hopping sequence, it controls entry of client devices into the piconet, it controls the power (100 mW, 2.5mW, or 1 mW) at which client devices transmit; and uses polling to grant clients permission to transmit once admitted to the network. In addition to the active devices, there can also be up to 255 “parked” devices in the piconet. These parked devices are often in some form of “sleep mode” to conserve energy (as we saw with 802.11 power management) and will awaken periodically, according to the master’s schedule, to receive beacon messages from the master. A parked device cannot communicate until its status has been changed from parked to active by the master node.

Because Bluetooth ad hoc networks must be **self-organizing**, it’s worth looking into how they bootstrap their network structure. When a master node wants to form a Bluetooth network, it must first determine which other Bluetooth devices are within range; this is the **neighbor discovery** problem. The master does this by broadcasting a series of 32 inquiry messages, each on a different frequency channel, and repeats the transmission sequence for up to 128 times. A client device listens on its chosen frequency, hoping to hear one of the master’s inquiry messages on this frequency. When it hears an inquiry message, it backs off a random amount of time between 0 and 0.3 seconds (to avoid collisions with other responding nodes, reminiscent of Ethernet’s binary backoff) and then responds to the master with a message containing its device ID.

Once the Bluetooth master has discovered all of the potential clients within range, it then invites those clients that it wishes to join the piconet. This second phase is known as **Bluetooth paging**, and is reminiscent of 802.11 clients associating with a base station. Through the paging process, the master will inform the client of the frequency-hopping pattern to be used, and the sender’s clock. The master begins the paging process by again sending 32 identical paging invitation messages, each now addressed to a specific client, but again using different frequencies, since that client has yet to learn the frequency-hopping pattern. Once the client replies with an ACK message to the paging invitation message, the master sends frequency-hopping information, clock synchronization information and an active member address to the client, and then finally polls the client, now using the frequency-hopping pattern, to ensure that the client is connected into the network.

In our discussion above, we have only touched on Bluetooth’s wireless networking. Higher level protocols provide for reliable data packet transfer, circuit-like streaming of audio and video, changing transmission power levels, changing active/parked state (and other states), and more. More recent versions of Bluetooth have addressed low energy and security considerations. For more information about Bluetooth, the interested reader should consult [Bisdikian 2001, Colbach 2017, and Bluetooth 2020].

## 7.4 Cellular Networks: 4G and 5G

In the previous section, we examined how a host can access the Internet when within the vicinity of an 802.11 WiFi access point (AP). But as we've seen, APs have small coverage areas, and a host certainly will not be able to associate with every AP it encounters. As a result, WiFi access is hardly ubiquitous for a user on the move.

By contrast, 4G cellular network access has rapidly become pervasive. A recent measurement study of more than one million US mobile cellular network subscribers found that they can find 4G signals more than 90% of the time, with download speeds of 20 Mbps and higher. Users of Korea's three major cellular carriers are able to find a 4G signal between 95 and 99.5% of the time [Open Signal 2019]. As a result, it is now commonplace to stream HD videos or participate in videoconferences while on the move in a car, bus, or high-speed train. The ubiquity of 4G Internet access has also enabled myriad new IoT applications such as Internet-connected shared bike and scooter systems, and smartphone applications such as mobile payments (commonplace in China since 2018) and Internet-based messaging (WeChat, WhatsApp, and more).

The term *cellular* refers to the fact that the region covered by a cellular network is partitioned into a number of geographic coverage areas, known as **cells**. Each cell contains a **base station** that transmits signals to, and receives signals from, the **mobile devices** currently in its cell. The coverage area of a cell depends on many factors, including the transmitting power of the base station, the transmitting power of the devices, obstructing buildings in the cell, and the height and type of the base station antennas.

In this section, we provide an overview of the current 4G and emerging 5G cellular networks. We'll consider the wireless first hop between the mobile device and the base station, as well as the cellular carrier's all-IP core network that connects the wireless first hop into the carrier's network, other carrier networks, and the larger Internet. Perhaps surprisingly (given the origins of mobile cellular networks in the telephony world, which had a *very* different network architecture from the Internet), we'll encounter many of the architectural principles in 4G networks that we encountered in our Internet-focused studies in Chapters 1–6, including protocol layering, an edge/core distinction, the interconnection of multiple provider networks to form a global "network of networks," and the clear separation of data and control planes with logically centralized control. We'll now see these principles through the lens of mobile cellular networks (rather than through an Internet lens) and thus see these principles instantiated in different ways. And of course, with a carrier's network having an all-IP core, we'll also encounter many of the Internet protocols that we now know well. We'll cover additional 4G topics—mobility management in Section 7.6, and 4G security in Section 8.8—later, after developing the basic principles needed for these topics.

Our discussion here of 4G and 5G networks will be relatively brief. Mobile cellular networking is an area with great breadth and depth, with many universities offering several courses on the topic. Readers seeking a deeper understanding are encouraged to see [Goodman 1997; Kaaranen 2001; Lin 2001; Korhonen 2003;

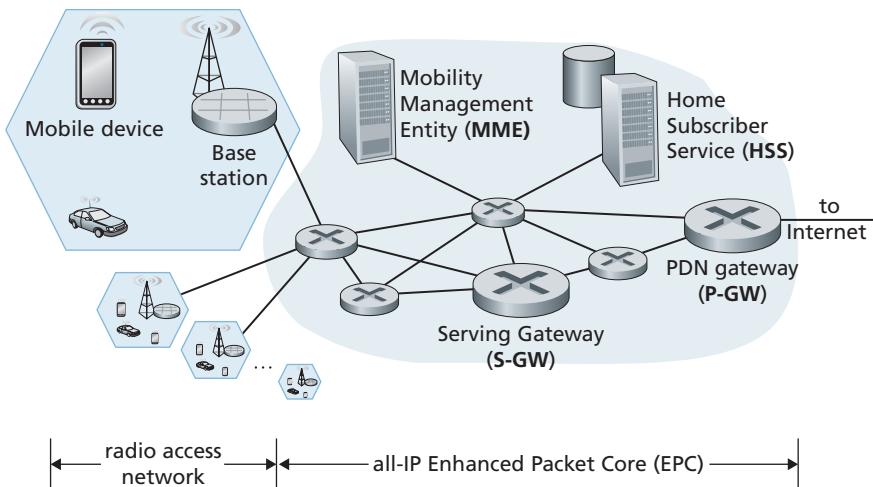
Schiller 2003; Palat 2009; Scourias 2012; Turner 2012; Akyildiz 2010], as well as the particularly excellent and exhaustive books [Mouly 1992; Sauter 2014].

Just as Internet RFCs define Internet-standard architecture and protocols, 4G and 5G networks are also defined by standards documents known as Technical Specifications. These documents are freely available online at [3GPP 2020]. Just like RFCs, technical specifications can make for rather dense and detailed reading. But when you have a question, they are the definitive source for answers!

### 7.4.1 4G LTE Cellular Networks: Architecture and Elements

The 4G networks that are pervasive as of this writing in 2020 implement the 4G Long-Term Evolution standard, or more succinctly **4G LTE**. In this section, we'll describe 4G LTE networks. Figure 7.17 shows the major elements of the 4G LTE network architecture. The network broadly divides into the radio network at the cellular network's edge and the core network. All network elements communicate with each other using the IP protocol we studied in Chapter 4. As with earlier 2G and 3G networks, 4G LTE is full of rather obtuse acronyms and element names. We'll try to cut through that jumble by first focusing on element functions and how the various elements of a 4G LTE network interact with each other in both the data and the control planes:

- **Mobile Device.** This is a smartphone, tablet, laptop, or IoT device that connects into a cellular carrier's network. This is where applications such as web browsers, map apps, voice and videoconference apps, mobile payment apps, and so much more are run. The mobile device typically implements the full 5-layer Internet protocol stack, including the transport and application layers, as we saw with hosts at the Internet's network edge. The mobile device is a network endpoint, with an IP address (obtained through NAT, as we'll see). The mobile device also has a globally unique 64-bit identifier called the **International Mobile Subscriber Identity (IMSI)**, which is stored on its SIM (Subscriber Identity Module) card. The IMSI identifies the subscriber in the worldwide cellular carrier network system, including the country and home cellular carrier network to which the subscriber belongs. In some ways, the IMSI is analogous to a MAC address. The SIM card also stores information about the services that the subscriber is able to access and encryption key information for that subscriber. In the official 4G LTE jargon, the mobile device is referred to as **User Equipment (UE)**. However, in this textbook, we'll use the more reader-friendly term "mobile device" throughout. We also note here that a mobile device is not always mobile; for example, the device might be a fixed temperature sensor or a surveillance camera.
- **Base Station.** The base station sits at the "edge" of the carrier's network and is responsible for managing the wireless radio resources and the mobile devices with its coverage area (shown as a hexagonal cell in Figure 7.17). As we'll see, a mobile device will interact with a base station to attach to the carrier's network. The base station coordinates device authentication and allocation of resources



**Figure 7.17** ♦ Elements of the 4G LTE architecture

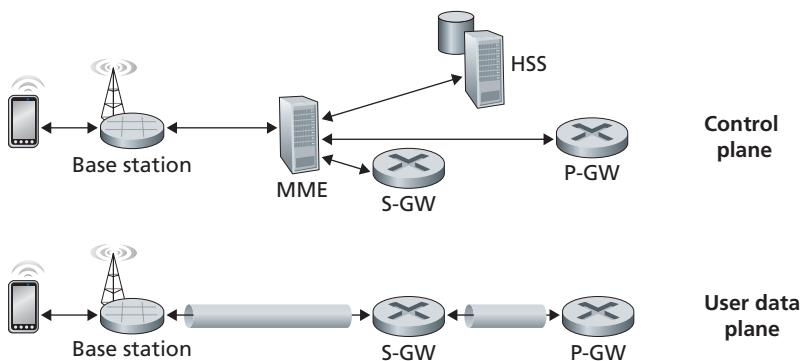
(channel access) in the radio access network. In this sense, cellular base station functions are comparable (but by no means identical) to those of APs in wireless LANs. But cellular base stations have several other important roles not found in wireless LANs. In particular, base stations create device-specific IP tunnels from the mobile device to gateways and interact among themselves to handle device mobility among cells. Nearby base stations also coordinate among themselves to manage the radio spectrum to minimize interference between cells. In the official 4G LTE terminology, the base station is referred to as an “**eNode-B**,” which is rather opaque and non-descriptive. In this textbook, we will instead use the reader-friendlier term “base station” throughout.

As an aside, if you find LTE terminology a bit opaque, you aren’t alone! The etymology of “eNode-B” is rooted in earlier 3G terminology, where network function points were referred to as “nodes,” with “B” harkening back to earlier “Base Station (BS)” 1G terminology or “Base Transceiver Station (BTS)” in 2G terminology. 4G LTE is an “e”volution over 3G, and hence, an “e” now precedes “Node-B” in 4G LTE terminology. This name opaqueness shows no signs in stopping! In 5G systems, eNode-B functions are now referred to as “ng-eNB”; perhaps you can guess what that acronym stands for!

- **Home Subscriber Server (HSS).** As shown in Figure 7.18, the HSS is a control-plane element. The HSS is a database, storing information about the mobile devices for which the HSS’s network is their home network. It is used in conjunction with the MME (discussed below) for device authentication.
- **Serving Gateway (S-GW), Packet Data Network Gateway (P-GW), and other network routers.** As shown in Figure 7.18, the Serving Gateway and the Packet Data Network Gateway are two routers (often collocated in practice) that

lie on the data path between the mobile device and the Internet. The PDN Gateway also provides NAT IP addresses to mobile devices and performs NAT functions (see Section 4.3.4). The PDN Gateway is the last LTE element that a datagram originating at a mobile device encounters before entering the larger Internet. To the outside world, the P-GW looks like any other gateway router; the mobility of the mobile nodes within the cellular carrier's LTE network is hidden from the outside world behind the P-GW. In addition to these gateway routers, a cellular carrier's all-IP core will have additional routers whose role is similar to that of traditional IP routers—to forward IP datagrams among themselves along paths that will typically terminate at elements of the LTE core network.

- **Mobility Management Entity (MME).** The MME is also a control-plane element, as shown in Figure 7.18. Along with the HSS, it plays an important role in authenticating a device wanting to connect into its network. It also sets up the tunnels on the data path from/to the device and the PDN Internet gateway router, and maintains information about an active mobile device's cell location within the carrier's cellular network. But, as shown in Figure 7.18, it is not in the forwarding path for the mobile device's datagrams being sent to and from the Internet.
  - *Authentication.* It is important for the network and the mobile device attaching to the network to *mutually* authenticate each other—for the network to know that the attaching device is indeed the device associated with a given IMSI, and for the mobile device to know that the network to which it is attaching is also a legitimate cellular carrier network. We will cover authentication in Chapter 8 and cover 4G authentication in Section 8.8. Here, we simply note that the MME plays a middleman role between the mobile and Home Subscriber Service (HSS) in the mobile's home network. Specifically, after receiving an attach request from mobile device, the local MME contacts the HSS in the mobile's home network. The mobile's home HSS then returns enough encrypted information to the local MME to prove to the mobile device that the home HSS is performing authentication through this MME, and for the mobile device to prove to the MME that it is indeed the mobile associated with that IMSI. When a mobile device is attached to its home network, the HSS to be contacted during authentication is located within that same home network. However, when a mobile device is roaming on a visited network operated by a different cellular network carrier, the MME in that roaming network will need to contact the HSS in the mobile device's home network.
  - *Path setup.* As shown in the bottom half of Figure 7.18, the data path from the mobile device to the carrier's gateway router consists of a wireless first hop between the mobile device and the base station, and concatenated IP tunnels between the base station and the Serving Gateway, and the Serving Gateway and the PDN Gateway. Tunnels are setup under the control of the MME and used for data forwarding (rather than direct forwarding among network routers) to facilitate device mobility—when a device moves, only the tunnel endpoint



**Figure 7.18** ♦ LTE data-plane and control-plane elements

terminating at the base station needs to be changed, while other tunnel endpoints, and the Quality of Service associated with a tunnel, remain unchanged.

- **Cell location tracking.** As the device moves between cells, the base stations will update the MME on the device's location. If the mobile device is in a sleep mode but nonetheless moving between cells, the base stations can no longer track the device's location. In this case, it will be the responsibility of the MME to locate the device for wakeup, through a process known as **paging**.

Table 7.2 summarizes the key LTE architectural elements that we have discussed above and compares these functions with those we encountered in our study of WiFi wireless LANs (WLANs).

| LTE Element                                | Description                                                                                                                | Similar WLAN function(s)                                                                    |
|--------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| Mobile device (UE: User equipment)         | End user's IP-capable wireless/mobile device (e.g., smartphone, tablet, laptop)                                            | Host, end-system                                                                            |
| Base Station (eNode-B)                     | Network side of wireless access link into LTE network                                                                      | Access point (AP), although the LTE base station performs many functions not found in WLANs |
| The Mobility Management Entity (MME)       | Coordinator for mobile device services: authentication, mobility management                                                | Access point (AP), although the MME performs many functions not found in WLANs              |
| Home Subscriber Server (HSS)               | Located in a mobile device's <i>home</i> network, providing authentication, access privileges in home and visited networks | No WLAN equivalent                                                                          |
| Serving Gateway (S-GW), PDN-Gateway (P-GW) | Routers in a cellular carrier's network, coordinating forwarding to outside of the carrier's network                       | iBGP and eBGP routers in access ISP network                                                 |
| Radio Access Network                       | Wireless link between mobile device and a base station                                                                     | 802.11 wireless link between mobile and AP                                                  |

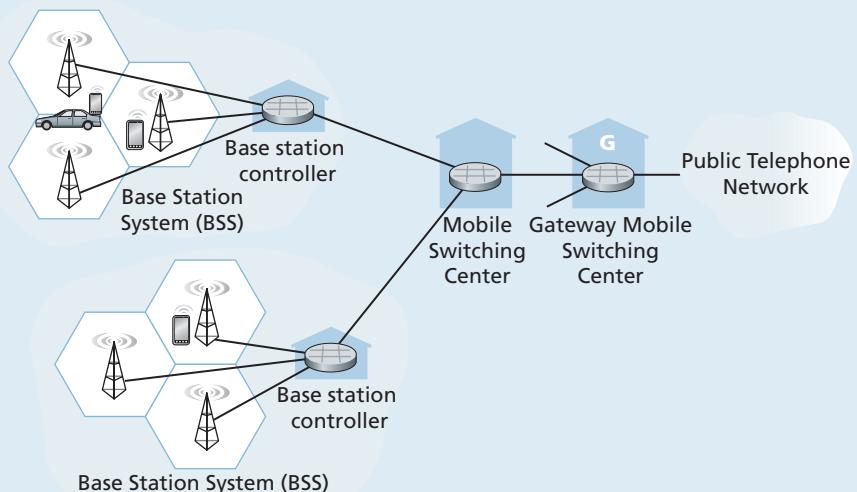
**Table 7.2** ♦ LTE Elements, and similar WLAN (WiFi) functions

## CASE HISTORY

### THE ARCHITECTURAL EVOLUTION FROM 2G TO 3G TO 4G

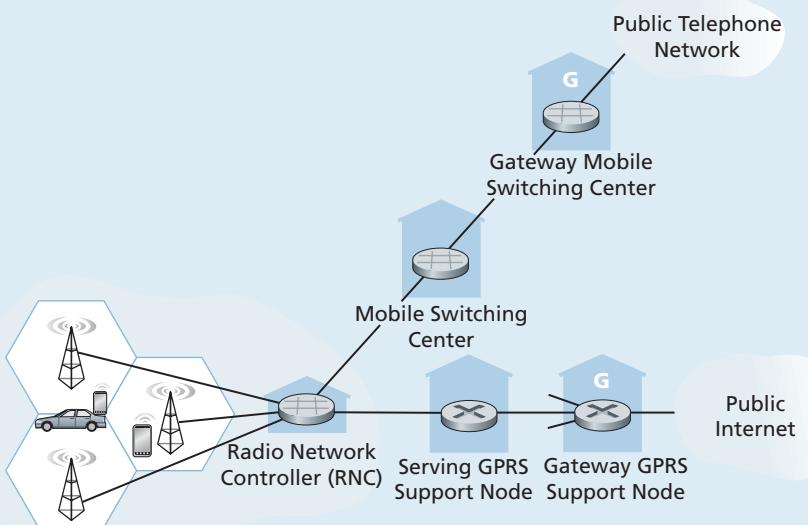
In a relatively short span of 20 years, cellular carrier networks have undergone an astonishing transition from being almost exclusively circuit-switched telephone networks to being all-IP packet-switched data networks which include voice as just one of many applications. How did this transition happen from an architectural standpoint? Was there a “flag day,” when the previous telephony-oriented networks were turned “off” and the all-IP cellular network was turned “on”? Or did elements in the previous telephony-oriented networks begin taking on dual circuit (legacy) and packet (new) functionality, as we saw with the IPv4-to-IPv6 transition in Section 4.3.5?

Figure 7.19 is taken from the earlier 7th edition of this textbook, which covered both 2G and 3G cellular networks. (We have retired this historical material, which is still available on this book’s website, in favor of a deeper coverage of 4G LTE in this 8th edition). Although the 2G network is a circuit-switched mobile telephone network, a comparison of Figures 7.17 and 7.19 illustrates a similar conceptual structure, albeit for voice rather than for data services—a wireless edge controlled by a base station, a gateway from the carrier’s network to the outside world, and aggregation points between the base stations and the gateway.



**Figure 7.19** ♦ Elements of the 2G cellular architecture, supporting circuit-switched voice service with the carrier’s core network

Figure 7.20 (also taken from the 7th edition of this textbook) shows the main architectural components of the 3G cellular architecture, which supports both circuit-switched voice service *and* packet-switched data services. Here, the transition from a voice-only network to a combined voice and data network is clear: the existing core 2G cellular voice network elements remained untouched. However, *additional cellular data functionality was added in parallel to, and functioned independently from, the existing core voice network at that time*. As shown in Figure 7.20, the splitting point into these two separate core voice and data networks happened at the network edge, at the base station in the radio access network. The alternative—integrating new data services directly into the core elements of the existing cellular voice network—would have raised the same challenges encountered in integrating new (IPv6) and legacy (IPv4) technologies in the Internet. The carriers also wanted to leverage and exploit their considerable investment of existing infrastructure (and profitable services!) in their existing cellular voice network.



**Figure 7.20** ♦ 3G system architecture: supporting separate circuit-switched voice service and packet-switched data service with the carrier's core network

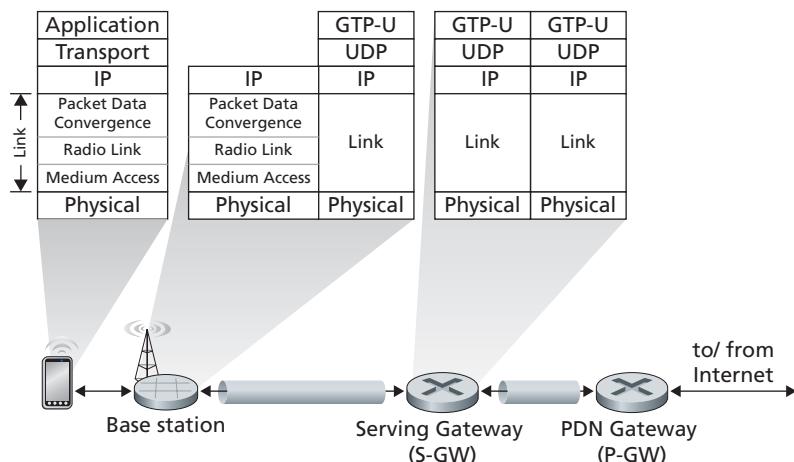
### 7.4.2 LTE Protocols Stacks

Since the 4G LTE architecture is an all-IP architecture, we're already very familiar with the higher-layer protocols in the LTE protocol stack, in particular IP, TCP, UDP, and various application layer protocols, from our studies in Chapters 2 through 5. Consequently, the new LTE protocols that we'll focus on here are primarily at the link and physical layers, and in mobility management.

Figure 7.21 shows the user-plane protocol stacks at the LTE mobile node, the base station, and the serving gateway. We'll touch on several of LTE's control-plane protocols later when we study LTE mobility management (Section 7.6) and security (Section 8.8). As we can see from Figure 7.21, most of the new and interesting user-plane protocol activity is happening at the wireless radio link between the mobile device and the base station.

LTE divides the mobile device's link layer into three sublayers:

- *Packet Data Convergence*. This uppermost sublayer of the link layer sits just below IP. The Packet Data Convergence Protocol (PDCP) [3GPP PDCP 2019] performs IP header/compression in order to decrease the number of bits sent over the wireless link, and encryption/decryption of the IP datagram using keys that were established via signaling messages between the LTE mobile device and the Mobility Management Entity (MME) when the mobile device first attached to the network; we'll cover aspects of LTE security in Section 8.8.2.
- *Radio Link Control*. The Radio Link Control (RLC) Protocol [3GPP RLCP 2018] performs two important functions: (*i*) fragmenting (on the sending side) and reassembly (on the receiving) of IP datagrams that are too large to fit into



**Figure 7.21** ◆ LTE data-plane protocol stacks

the underlying link-layer frames, and (ii) link-layer reliable data transfer at the through the use of an ACK/NAK-based ARQ protocol. Recall the we've studied the basic elements of ARQ protocols in Section 3.4.1.

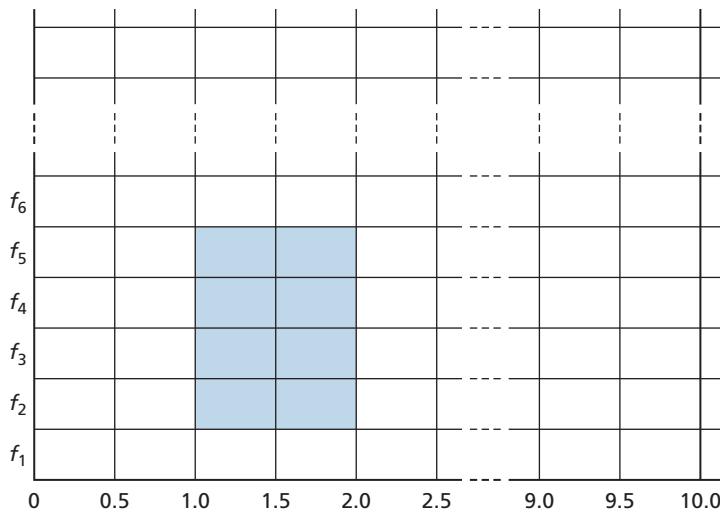
- *Medium Access Control (MAC).* The MAC layer performs transmission scheduling, that is, the requesting and use of the radio transmission slots described in Section 7.4.4. The MAC sublayer also performs additional error detection/correction functions, including the use of redundant bit transmission as a forward error-correction technique. The amount of redundancy can be adapted to channel conditions.

Figure 7.21 also shows the use of tunnels in the user data path. As discussed above, these tunnels are established, under MME control, when the mobile device first attaches to the network. Each tunnel between two endpoints has a unique tunnel endpoint identifier (TEID). When the base station receives datagrams from the mobile device, it encapsulates them using the GPRS Tunneling Protocol [3GPP GTPv1-U 2019], including the TEID, and sends them in UDP segments to the Serving Gateway at the other end of the tunnel. On the receiving side, the base station decapsulates tunneled UDP datagrams, extracts the encapsulated IP datagram destined for the mobile device, and forwards that IP datagram over the wireless hop to the mobile device.

### 7.4.3 LTE Radio Access Network

LTE uses a combination of frequency division multiplexing and time division multiplexing on the downstream channel, known as orthogonal frequency division multiplexing (OFDM) [Hwang 2009]. (The term “orthogonal” comes from the fact the signals being sent on different frequency channels are created so that they interfere very little with each other, even when channel frequencies are tightly spaced). In LTE, each active mobile device is allocated one or more 0.5 ms time slots in one or more of the channel frequencies. Figure 7.22 shows an allocation of eight time slots over four frequencies. By being allocated increasingly more time slots (whether on the same frequency or on different frequencies), a mobile device is able to achieve increasingly higher transmission rates. Slot (re)allocation among mobile devices can also be used to change the transmission rate; see our earlier discussion of Figure 7.3 and dynamic selection of modulation schemes in WiFi networks.

The particular allocation of time slots to mobile devices is not mandated by the LTE standard. Instead, the decision of which mobile devices will be allowed to transmit in a given time slot on a given frequency is determined by the scheduling algorithms provided by the LTE equipment vendor and/or the network operator. With opportunistic scheduling [Bender 2000; Kolding 2003; Kulkarni 2005], matching the physical-layer protocol to the channel conditions between the sender and receiver and choosing the receivers to which packets will be sent based on channel conditions allow the base station to make best use of the wireless medium. In addition, user



**Figure 7.22** ♦ Twenty 0.5-ms slots organized into 10 ms frames at each frequency. An eight-slot allocation is shown shaded.

priorities and contracted levels of service (e.g., silver, gold, or platinum) can be used in scheduling downstream packet transmissions. In addition to the LTE capabilities described above, LTE-Advanced allows for downstream bandwidths of hundreds of Mbps by allocating aggregated channels to a mobile device [Akyildiz 2010].

#### 7.4.4 Additional LTE Functions: Network Attachment and Power Management

Let's conclude our study of 4G LTE here by considering two additional important LTE functions: (*i*) the process with which a mobile device first attaches to the network and (*ii*) the techniques used by the mobile device, in conjunction with core network elements, to manage its power use.

##### Network Attachment

The process by which a mobile device attaches to the cellular carrier's network divides broadly into three phases:

- *Attachment to a Base Station.* This first phase of device attachment is similar in purpose to, but quite different in practice from, the 802.11 association protocol that we studied in Section 7.31. A mobile device wishing to attach to a cellular carrier network will begin a bootstrap process to learn about, and then associate with, a nearby base station. The mobile device initially searches all channels in all frequency bands for a primary synchronization signal that is periodically broadcast

every 5 ms by a base station. Once this signal is found, the mobile device remains on this frequency and locates the secondary synchronization signal. With information found in this second signal, the device can locate (following several further steps) additional information such as channel bandwidth, channel configurations, and the cellular carrier information of that base station. Armed with this information, the mobile device can select a base station to associate with (preferentially attaching to its home network, if available) and establish a control-plane signaling connection across the wireless hop with that base station. This mobile-to-base-station channel will be used through the remainder of the network attachment process.

- *Mutual Authentication.* In our earlier description of the Mobility Management Entity (MME) in Section 7.4.1, we noted that the base station contacts the local MME to perform mutual authentication—a process that we’ll study in further detail in Section 8.8.2. This is the second phase of network attachment, allowing the network to know that the attaching device is indeed the device associated with a given IMSI, and the mobile device to know that the network to which it is attaching is also a legitimate cellular carrier network. Once this second phase of network attachment is complete, the MME and mobile device have mutually authenticated each other, and the MME also knows the identity of the base station to which the mobile is attached. Armed with this information, the MME is now ready to configure the Mobile-device-to-PDN-gateway data path.
- *Mobile-device-to-PDN-gateway Data Path Configuration.* The MME contacts the PDN gateway (which also provides a NAT address for the mobile device), the Serving gateway, and the base station to establish the two tunnels shown in Figure 7.21. Once this phase is complete, the mobile device is able to send/receive IP datagrams via the base station through these tunnels to and from the Internet!

### Power Management: Sleep Modes

Recall in our earlier discussion of advanced features in 802.11 (Section 7.3.5) and Bluetooth (Section 7.3.6) that a radio in a wireless device may enter a sleep state to save power when it is not transmitting or receiving in order to minimize the amount of time that the mobile device’s circuitry needs to be “on” for sending/receiving data, and for channel sensing. In 4G LTE, a sleeping mobile device can be in one of two different sleep states. In the discontinuous reception state, which is typically entered after several hundred milliseconds of inactivity [Sauter 2014], the mobile device and the base station will schedule periodic times in advance (typically several hundred milliseconds apart) at which the mobile device will wake up and actively monitor the channel for downstream (base station to mobile device) transmissions; apart from these scheduled times, however, the mobile device’s radio will be sleeping.

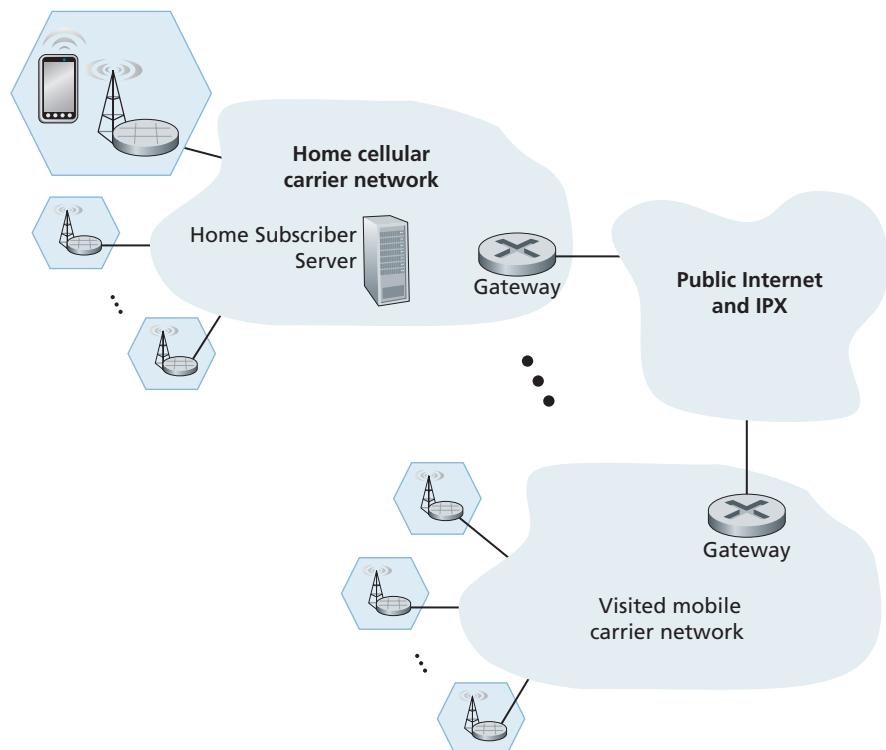
If the discontinuous reception state might be considered a “light sleep,” the second sleep state—the Idle state—which follows even longer periods of 5 to 10 seconds of inactivity, might be thought of as a “deep sleep.” While in this deep sleep, the mobile device’s radio wakes up and monitors the channel even less frequently. Indeed, this sleep is so deep that if the mobile device moves into a new cell in the carrier’s network

while sleeping, it need not inform the base station with which it was previous associated. Thus, when waking up periodically from this deep sleep, the mobile device will need to re-establish an association with a (potentially new) base station in order to check for paging messages broadcast by the MME to base stations nearby the base station with which the mobile was last associated. These control-plane paging messages, which are broadcast by these base stations to all mobile devices in their cells, indicate which mobile devices should fully wake up and re-establish a new data-plane connection to a base station (see Figure 7.18) in order to receive incoming packets.

#### 7.4.5 The Global Cellular Network: A Network of Networks

Having now studied the 4G cellular network architecture, let's take a step back at take a look at how the global cellular network—itself a “network of networks” like the Internet—is organized.

Figure 7.23 shows a user's mobile smartphone connected via a 4G base station into its **home network**. The user's home mobile network is operated by a cellular



**Figure 7.23** ♦ The global cellular data network: a network of networks.

carrier such as Verizon, AT&T, T-Mobile, or Sprint in the United States; Orange in France; or SK Telecom in Korea. The user's home network, in turn, is connected to the networks of other cellular carriers and to the global Internet, though one or more gateway routers in the home network, as shown in Figure 7.23. The mobile networks themselves interconnect with each other either via the public Internet or via an Internet Protocol Packet eXchange (IPX) Network [GSMA 2018a]. An IPX is a managed network specifically for interconnecting cellular carriers, similar to Internet eXchange Points (see Figure 1.15) for peering among ISPs. From Figure 7.23, we can see that the global cellular network is indeed a "network of networks"—just like the Internet (recall Figure 1.15 and Section 5.4). 4G networks can also peer with 3G cellular voice/data networks and earlier voice-only networks.

We'll return shortly to additional 4G LTE topics—mobility management in Section 7.6, and 4G security in Section 8.8.2—later, after developing the basic principles needed for these topics. Let's now take a quick look at the emerging 5G networks.

#### 7.4.6 5G Cellular Networks

The ultimate wide-area data service would be one with ubiquitous gigabit connection speeds, extremely low latency, and unrestricted limitations on the number of users and devices that could be supported in any region. Such a service would open the door to all kinds of new applications, including pervasive augmented reality and virtual reality, control of autonomous vehicles via wireless connections, control of robots in factories via wireless connections, and replacement of residential access technologies, such as DSL and cable, with fixed wireless Internet services (that is, residential wireless connections from base stations to modems in homes).

It is expected that 5G, for which progressively improved versions are likely to be rolled out in the 2020 decade, will make a big step towards achieving the goals of the ultimate wide-area data service. It is predicted that 5G will provide roughly a 10x increase in peak bitrate, a 10x decrease in latency, and a 100x increase in traffic capacity over 4G [Qualcomm 2019].

Principally, 5G refers to "5G NR (New Radio)," which is the standard adopted by 3GPP. Other 5G technologies besides NR do exist, however. For example, Verizon's proprietary 5G TF network operates on 28 and 39 GHz frequencies and is used only for fixed wireless Internet service, not in smartphones.

5G standards divide frequencies into two groups: FR1 (450 MHz–6 GHz) and FR2 (24 GHz–52 GHz). Most early deployments will be in the FR1 space, although there are early deployments as of 2020 in the FR2 space for fixed Internet residential access as mentioned just above. Importantly, the physical layer (that is, wireless) aspects of 5G are *not* backward-compatible with 4G mobile communications systems such as LTE: in particular, it can't be delivered to existing smartphones by deploying base station upgrades or software updates. Therefore, in the transition to 5G, wireless carriers will need to make substantial investments in physical infrastructure.

FR2 frequencies are also known as **millimeter wave frequencies**. While millimeter wave frequencies allow for much faster data speeds, they come with two major drawbacks:

- Millimeter wave frequencies have much shorter range from base station to receivers. This makes millimeter wave technology unsuitable in rural areas and requires denser deployments of base stations in urban areas.
- Millimeter wave communication is highly susceptible to atmospheric interference. Nearby foliage and rain can cause problems for outdoor use.

5G is not one cohesive standard, but instead consists of three co-existing standards [Dahlman 2018]:

- *eMBB (Enhanced Mobile Broadband)*. Initial deployments of 5G NR have focused on eMBB, which provides for increased bandwidth for higher download and upload speeds, as well as a moderate reduction in latency when compared to 4G LTE. eMBB enables rich media applications, such as mobile augmented reality and virtual reality, as well as mobile 4K resolution and 360° video streaming.
- *URLLC (Ultra Reliable Low-Latency Communications)*. URLLC is targeted towards applications that are highly latency-sensitive, such as factory automation and autonomous driving. URLLC is targeting latencies of 1msec. As of this writing, technologies that enable URLLC are still being standardized.
- *mMTC (Massive Machine Type Communications)*. mMTC is a narrowband access type for sensing, metering, and monitoring applications. One priority for the design of 5G networks is to lower barriers for network connectivity for IoT devices. In addition to lowering latency, emerging technologies for 5G networks are focusing on reducing power requirements, making the use of IoT devices more pervasive than has been with 4G LTE.

## 5G and Millimeter Wave Frequencies

Many 5G innovations will be a direct result of working in the millimeter wave frequencies in the 24 GHz–52 GHz band. For example, these frequencies offer the potential of achieving 100x increase in capacity over 4G. To get some insight into this, capacity can be defined as the product of three terms [Björnson 2017]:

$$\text{capacity} = \text{cell density} \times \text{available spectrum} \times \text{spectral efficiency}$$

where cell density is in units of cells/km<sup>2</sup>, available spectrum is in units of Hertz, and spectral efficiency is a measure of how efficiently each base station can communicate with users and is in units of bps/Hz/cell. By multiplying these units out, it is easy to see that capacity is in units of bps/km<sup>2</sup>. For each of these three terms, the values will be larger for 5G than for 4G:

- Because millimeter frequencies have much shorter range than 4G LTE frequencies, more base stations are required, which in turn increases the cell density.
- Because 5G FR2 operates in a much larger frequency band ( $52 - 24 = 28$  GHz) than 4G LTE (up to about 2 GHz), it has more available spectrum.
- With regard to spectral efficiency, information theory says that if you want to double spectral efficiency, a 17-fold increase in power is needed [Björnson 2017]. Instead of increasing power, 5G uses MIMO-technology (the same technology we encountered in our study of 802.11 networks in Section 7.3), which uses multiple antennas at each base station. Rather than broadcasting signals in all directions, each MIMO antenna employs **beam forming** and directs the signal at the user. MIMO technology allows a base station to send to 10–20 users at the same time in the same frequency band.

By increasing all three terms in the capacity equation, 5G is expected to provide a 100x increase in capacity in urban areas. Similarly, owing to the much wider frequency band, 5G is expected to provide peak download rates of 1 Gbps or higher.

Millimeter wave signals are, however, easily blocked by buildings and trees. **Small cell stations** are needed to fill in coverage gaps between base stations and users. In a highly populous region, the distance between two small cells could vary from 10 to 100 meters [Dahlman 2018].

## 5G Core Network

The **5G Core network** is the data network that manages all of the 5G mobile voice, data and Internet connections. The 5G Core network is being redesigned to better integrate with the Internet and cloud-based services, and also includes distributed servers and caches across the network, thereby reducing latency. Network function virtualization (as discussed in Chapters 4 and 5), and network slicing for different applications and services, will be managed in the core.

The new 5G Core specification introduces major changes in the way mobile networks support a wide variety of services with varied performance. As in the case of the 4G core network (recall Figures 7.17 and 7.18), the 5G core relays data traffic from end devices, authenticates devices, and manages device mobility. The 5G core also contains all of the network elements that we encountered in Section 7.4.2—the mobile devices, the cells, the base stations, and the Mobility Management Entity (now divided into two sub-elements, as discussed below), the HSS, and the Serving and PDN gateways.

Although the 4G and 5G core networks perform similar functions, there are some major differences in that the new 5G core architecture. The 5G Core is designed for complete control and user-plane separation (see Chapter 5). The 5G Core consists purely of virtualized software-based network functions. This new architecture will give

operators the flexibility to meet the diverse requirements of the different 5G applications. Some of the new 5G core network functions include [Rommer 2019]:

- *User-Plane Function (UPF)*. Control and user-plane separation (see Chapter 5) allows packet processing to be distributed and pushed to the network edge.
- *Access and Mobility Management Function (AMF)*. The 5G Core essentially decomposes the 4G Mobility Management Entity (MME) into two functional elements: AMF and SMF. The AMF receives all the connection and session information from end-user equipment but only handles connection and mobility management tasks.
- *Session Management Function (SMF)*. Session management is handled by the Session Management Function (SMF). The SMF is responsible for interacting with the decoupled data plane. The SMF also performs IP address management and plays the role of DHCP.

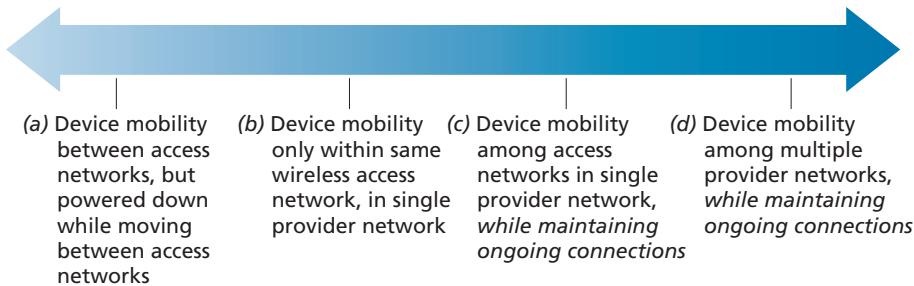
As of this writing (2020), 5G is in its early stages of deployment, and many 5G standards have yet to be finalized. Only time will tell whether 5G will become a pervasive broadband wireless service, whether it will successfully compete with WiFi for indoor wireless service, whether it will become a critical component of factory automation and the autonomous vehicle infrastructure, and whether it will take us a big step forward toward the ultimate wide-area wireless service.

## 7.5 Mobility Management: Principles

Having covered the wireless nature of the communication links in a wireless network, it's now time to turn our attention to the mobility that these wireless links enable. In the broadest sense, a mobile device is one that changes its point of attachment into the network over time. Because the term mobility has taken on many meanings in both the computer and telephony worlds, it will serve us well first to carefully consider forms of mobility.

### 7.5.1 Device Mobility: a Network-layer Perspective

From the network layer's standpoint, a physically mobile device will present a very different set of challenges to the network layer, depending on how active the device is as it moves between points of attachment to the network. At the one end of the spectrum, scenario (a) in Figure 7.24 is the mobile user who himself/herself physically moves between networks, but powers down the mobile device when moving. For example, a student might disconnect from a wireless classroom network and power down his/her device, head to the dining commons and connect to the wireless access



**Figure 7.24** ♦ Various degrees of mobility, from a network-layer perspective

network there while eating, and then disconnect and power down from the dining commons network, walk to the library, and connect to the library’s wireless network while studying. From a networking perspective, this device is *not* mobile—it attaches to an access network and remains in that access network while on. In this case, the device serially associates with, and later disassociates from, each wireless access network encountered. This case of device (non-)mobility can be completely handled using the networking mechanisms we’ve already studied in Sections 7.3 and 7.4.

In scenario (b) in Figure 7.24, the device is physically mobile but remains attached to the same access network. This device is also *not* mobile from a network-layer perspective. Additionally, if the device remains associated with the same 802.11 AP or LTE base station, the device is not even mobile from a link-layer perspective.

From a network standpoint, our interest in device mobility really starts with case (c), where a device changes its access network (e.g., 802.11 WLAN or LTE cell) while continuing to send and receiving IP datagrams, and while maintaining higher-level (e.g., TCP) connections. Here, the network will need to provide **handover**—a transfer of responsibility for forwarding datagrams to/from one AP or base station to the mobile device—as the device moves among WLANs or among LTE cells. We’ll cover handover in detail in Section 7.6. If the handover occurs within access networks belonging to a single network provider, that provider can orchestrate handover on its own. When a mobile device roams between multiple provider networks, as in scenario (d), the providers must orchestrate handover together, which considerably complicates the handover process.

### 7.5.2 Home Networks and Roaming on Visited Networks

As we learned in our discussions of cellular 4G LTE networks in Section 7.4.1, every subscriber has a “home” with some cellular provider. We learned that the Home Subscriber Service (HSS) stores information about each of its subscribers, including a globally unique device ID (embedded in a subscriber’s SIM card), information about services that the subscriber may access, cryptographic keys to be used for

communication, and billing/charging information. When a device is connected to a cellular network, other than its **home network**, that device is said to be **roaming** on a **visited network**. When a mobile device attaches to, and roams on, a visited network, coordination will be required between the home network and the visited network.

The Internet does not have a similarly strong notion of a home network or a visited network. In practice, a student's home network might be the network operated by his/her school; for mobile professionals, their home network might be their company network. The visited network might be the network of a school or a company they are visiting. But there is no notion of a home/visited network deeply embedded in the Internet's architecture. The Mobile IP protocol [Perkins 1998, RFC 5944], which we will cover briefly in Section 7.6, was a proposal that strongly incorporated the notion of home/visited networks. But Mobile IP has seen limited deployment/use in practice. There are also activities underway that are built on top of the existing IP infrastructure to provide authenticated network access across visited IP networks. Eduroam [Eduroam 2020] is one such activity.

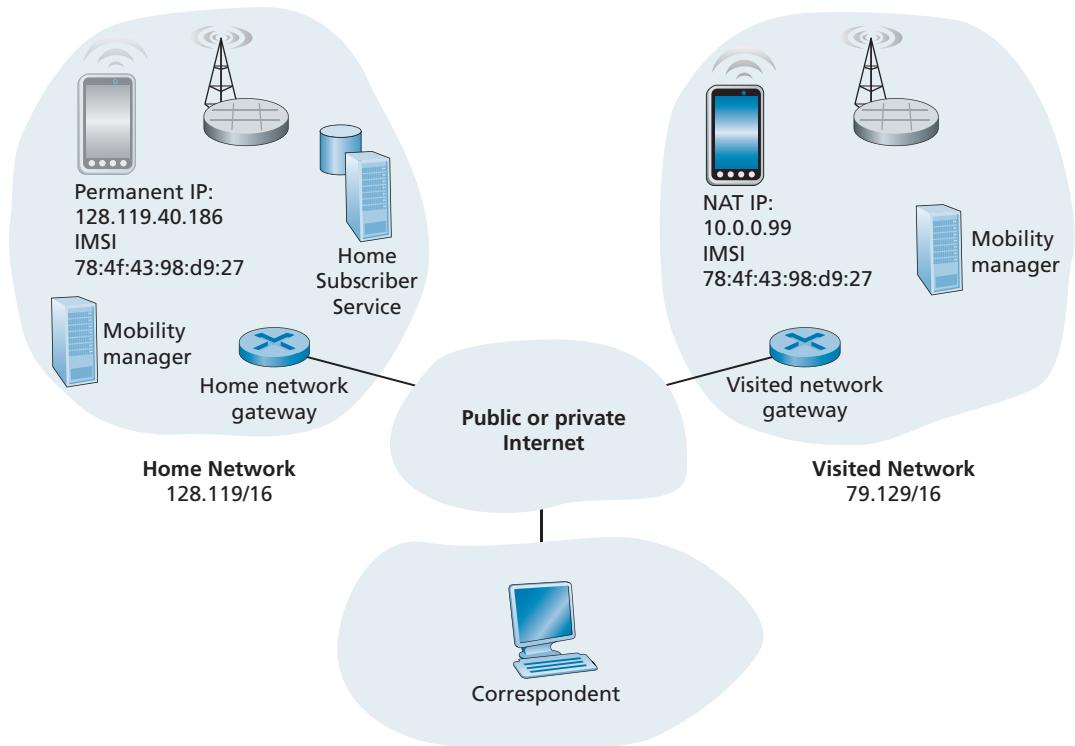
The notion of a mobile device having a home network provides two important advantages: the home network provides a single location where information about that device can be found, and (as we will see) it can serve as a coordination point for communication to/from a roaming mobile device.

To appreciate the potential value of the central point of information and coordination, consider the human analogy of a 20-something adult Bob moving out of the family home. Bob becomes mobile, living in a series of dormitories and apartments, and often changing addresses. If an old friend Alice wants to get in touch, how can Alice find the current address of Bob? One common way is to contact the family, since a mobile 20-something adult will often register his or her current address with the family (if for no other reason than so that the parents can send money to help pay the rent!). The family home becomes that unique location that others can go to as a first step in communicating with Bob. Additionally, later postal communication from Alice may be either *indirect* (e.g., with mail being sent first to Bob's family home and then forwarded to Bob) or *direct* (e.g., with Alice using the address obtained from Bob's parents to send mail directly to Bob).

### 7.5.3 Direct and Indirect Routing to/from a Mobile Device

Let us now consider the conundrum faced by the Internet-connected host (that we will refer to as a *correspondent*) in Figure 7.25 wishing to communicate with a mobile device that might be located within that mobile device's cellular home network, or might be roaming in a visited network. In our development below, we'll adopt a 4G/5G cellular network perspective, since these networks have such a long history of supporting device mobility. But as we'll see, the fundamental challenges and basic solution approaches for supporting device mobility are equally applicable in both cellular networks and in the Internet.

As shown in Figure 7.25, we'll assume that the mobile device has a globally unique identifier associated with it. In 4G, LTE cellular networks (see Section 7.4),



**Figure 7.25** ♦ Elements of a mobile network architecture

this would be the International Mobile Subscriber Identity (IMSI) and an associated phone number, stored on a mobile device's SIM card. For mobile Internet users, this would be a permanent IP address in the IP address range of its home network, as in the case of the Mobile IP architecture.

What approaches might be used in a mobile network architecture that would allow a datagram sent by the correspondent to reach that mobile device? Three basic approaches can be identified and are discussed below. As we will see, the latter two of these are adopted in practice.

### Leveraging the Existing IP Address Infrastructure

Perhaps the simplest approach to routing to a mobile device in a visited network is to simply use the existing IP addressing infrastructure—to add nothing new to the architecture. What could be easier!

Recall from our discussion of Figure 4.21 that an ISP uses BGP to advertise routes to destination networks by enumerating the CIDRized address ranges of reachable networks. A visited network could thus advertise to all other networks that a

particular mobile device is resident in its network simply by advertising a highly specific address—the mobile device’s full 32-bit IP permanent address—essentially informing other networks that it has the path to be used to forward datagrams to that mobile device. These neighboring networks would then propagate this routing information throughout the network as part of the normal BGP procedure of updating routing information and forwarding tables. Since datagrams will always be forwarded to the router advertising the most specific destination for that address (see Section 4.3), all datagrams addressed to that mobile device will be forwarded to the visited network. If the mobile device leaves one visited network and joins another, the new visited network can advertise a new, highly specific route to the mobile device, and the old visited network can withdraw its routing information regarding the mobile device.

This solves two problems at once, and does so without making changes to the network-layer infrastructure! Other networks know the location of the mobile device, and it is easy to route datagrams to the mobile device, since the forwarding tables will direct datagrams to the visited network. The killer drawback, however, is that of scalability—network routers would have to maintain forwarding table entries for potentially billions of mobile devices, and update a device’s entry each time it roams to a different network. Clearly, this approach would not work in practice. Some additional drawbacks are explored in the problems at the end of this chapter.

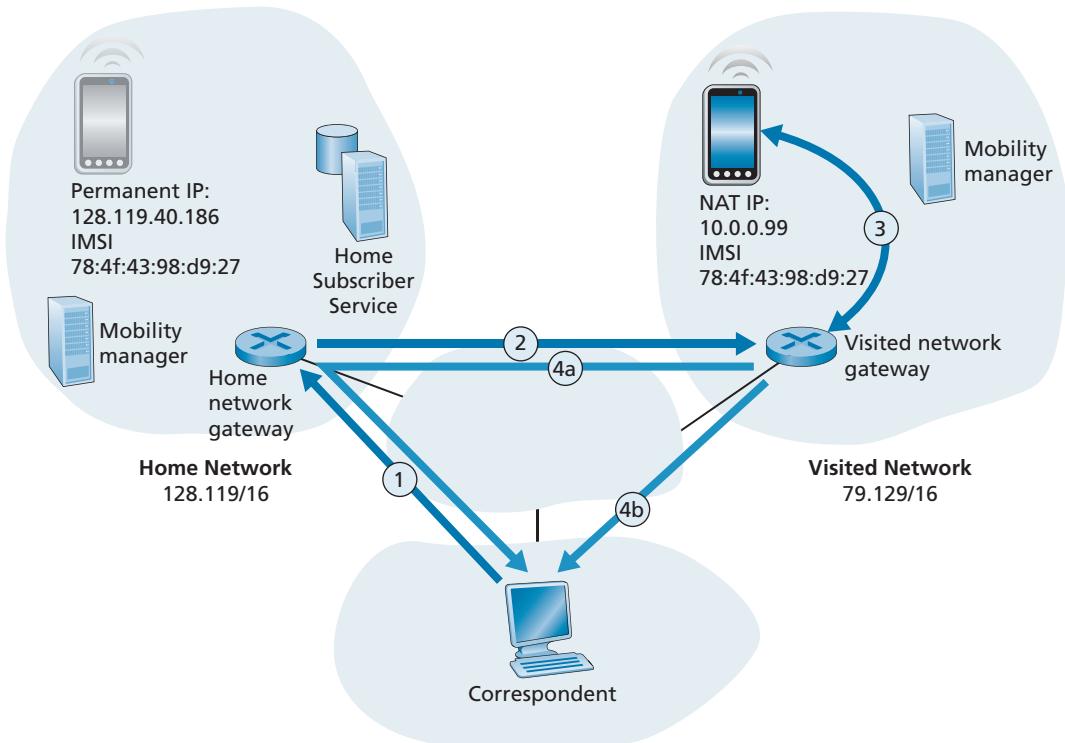
An alternative, more practical, approach (and one that has been adopted in practice) is to push mobility functionality from the network core to the network edge—a recurring theme in our study of Internet architecture. A natural way to do this is via the mobile device’s home network. In much the same way that parents of the mobile 20-something adult track their child’s location, a mobility management entity (MME) in the mobile device’s home network could track the visited network in which the mobile device resides. This information might reside in a database, shown as the HSS database in Figure 7.25. A protocol operating between the visited network and the home network will be needed to update the network in which the mobile device resides. You might recall that we encountered the MME and HSS elements in our study of 4G LTE. We’ll reuse their element names here, since they are so descriptive, and also because they are pervasively deployed in 4G networks.

Let’s next consider the visited network elements shown in Figure 7.25 in more detail. The mobile device will clearly need an IP address in the visited network. The possibilities here include using a permanent address associated with the mobile device’s home network, allocating a new address in the address range of the visited network, or providing an IP address via NAT (see Section 4.3.4). In the latter two cases, a mobile device has a transient identifier (a newly allocated IP address) in addition to its permanent identifiers stored in the HSS in its home network. These cases are analogous to a writer addressing a letter to the address of the house in which our mobile 20-something adult is currently living. In the case of a NAT address, datagrams destined to the mobile device would eventually reach the NAT gateway router in the visited network, which would then perform NAT address translation and forward the datagram to the mobile device.

We have now seen a number of elements of a solution to the correspondent's dilemma in Figure 7.24: home and visited networks, the MME and HSS, and mobile device addressing. But how should datagrams be addressed and forwarded to the mobile device? Since only the HSS (and not network-wide routers) knows the location of the mobile device, the correspondent cannot simply address a datagram to the mobile device's permanent address and send it into the network. Something more must be done. Two approaches can be identified: indirect and direct routing.

### Indirect Routing to a Mobile Device

Let's again consider the correspondent that wants to send a datagram to a mobile device. In the **indirect routing** approach, the correspondent simply addresses the datagram to the mobile device's permanent address and sends the datagram into the network, blissfully unaware of whether the mobile device is resident in its home network or in a visited network; mobility is thus completely transparent to the correspondent. Such datagrams are first routed, as usual, to the mobile device's home network. This is illustrated in step 1 in Figure 7.26.



**Figure 7.26** ♦ Indirect routing to a mobile device

Let's now turn our attention to the HSS, which is responsible for interacting with visited networks to track the mobile device's location, and the home network's gateway router. One job of this gateway router is to be on the lookout for an arriving datagram addressed to a device whose home is in that network, but that currently resides in a visited network. The home network gateway intercepts this datagram, consults with the HSS to determine the visited network where the mobile device is resident, and forwards the datagram toward the visited network gateway router—step 2 in Figure 7.26. The visited network gateway router then forwards the datagram toward the mobile device—step 3 in Figure 7.26. If NAT translation is used, as in Figure 7.26, the visited network gateway router performs NAT translation.

It is instructive to consider the rerouting at the home network in bit more detail. Clearly, the home network gateway will need to forward the arriving datagram to the gateway router in the visited network. On the other hand, it is desirable to leave the correspondent's datagram intact, since the application receiving the datagram should be unaware that the datagram was forwarded via the home network. Both goals can be satisfied by having the home gateway encapsulate the correspondent's original complete datagram within a new (larger) datagram. This larger datagram is then addressed and delivered to the visited network's gateway router, which will decapsulate the datagram—that is, remove the correspondent's original datagram from within the larger encapsulating datagram—and forward (step 3 in Figure 7.26) the original datagram to the mobile device. The sharp reader will note that the encapsulation/decapsulation described here is precisely the notion of tunneling, discussed in Section 4.3 in the context of IPv6; indeed, we also discussed the use of tunneling in the context of Figure 7.18, when we introduced the 4G LTE data plane.

Finally, let's consider how the mobile device sends datagrams to the correspondent. In the context of Figure 7.26, the mobile device will clearly need to forward the datagram through the visited gateway router, in order to perform NAT translation. But how then should the visited gateway router forward the datagram to the correspondent? As shown in Figure 7.26, there are two options here: (4a) the datagram could be tunneled back to the home gateway router, and sent to the correspondent from there, or (4b) the datagram could be transmitted from the visited network directly to the correspondent—an approach known as **local breakout** [GSMA 2019a] in LTE.

Let's summarize our discussion of indirect routing by reviewing the new network-layer functionality required to support mobility.

- *A mobile-device-to-visited-network association protocol.* The mobile device will need to associate with the visited network, and will similarly need to disassociate when leaving the visited network.
- *A visited-network-to-home-network-HSS registration protocol.* The visited network will need to register the mobile device's location with the HSS in the home network, and perhaps use information obtained from the HSS in performing device authentication.

- A datagram tunneling protocol between the home network gateway and the visited network gateway router. The sending side performs encapsulation and forwarding of the correspondent's original datagram; on the receiving side, the gateway router performs decapsulation, NAT translation, and forwarding of the original datagram to the mobile device.

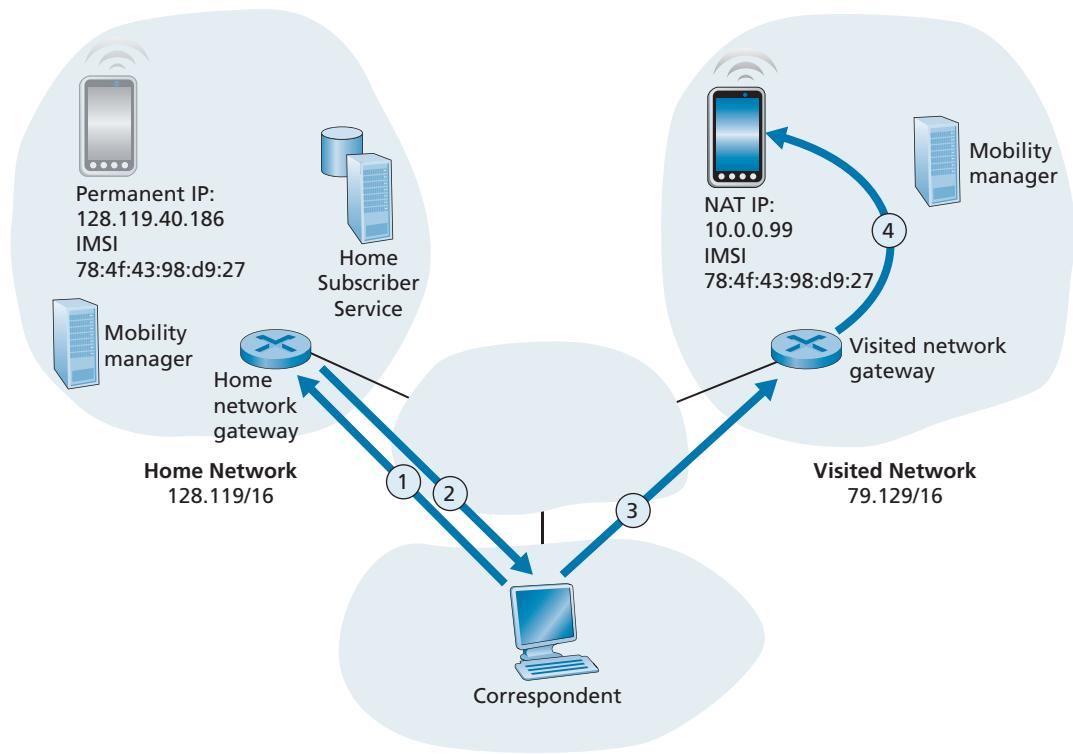
The previous discussion provides all the needed elements for a mobile device to maintain an ongoing connection with a correspondent as the device moves among networks. When a device roams from one visited network to another, the new visited network information needs to be updated in the home network HSS, and the home-gateway-router-to-visited-gateway-router tunnel endpoint needs to be moved. But will the mobile device see an interrupted flow of datagrams as it moves between networks? As long as the time between the mobile device disconnection from one visited network and its attachment to the next visited network is small, few datagrams will be lost. Recall from Chapter 3 that end-to-end connections can experience datagram loss due to network congestion. Hence, occasional datagram loss within a connection when a device moves between networks is by no means a catastrophic problem. If loss-free communication is required, upper-layer mechanisms will recover from datagram loss, whether such loss results from network congestion or from device mobility.

Our discussion above has been purposefully somewhat generic. An indirect routing approach is used in the mobile IP standard [RFC 5944], as well as in 4G LTE networks [Sauter 2014]. Their details, in particular the tunneling procedures employed, differ just a bit from our generic discussion above.

### Direct Routing to a Mobile Device

The indirect routing approach illustrated in Figure 7.26 suffers from an inefficiency known as the **triangle routing problem**—datagrams addressed to the mobile device must be forwarded first to the home network and then to the visited network, even when a much more efficient route exists between the correspondent and the roaming mobile device. In the worst case, imagine a mobile user who is roaming on the same network that is the home network for an overseas colleague who our mobile user is visiting. The two are sitting side-by-side and exchanging data. Datagrams between the mobile user and his overseas colleague will be forwarded to the mobile user's home network and then back again to the visited network!

**Direct routing** overcomes the inefficiency of triangle routing, but does so at the cost of additional complexity. In the direct routing approach, shown in Figure 7.27, the correspondent first discovers the visited network in which the mobile is resident. This is done by querying the HSS in the mobile device's home network, assuming (as in the case of indirect routing) that the mobile device's visited network is registered in the HSS. This is shown as steps 1 and 2 in Figure 7.27. The correspondent then tunnels datagrams from its network *directly* to the gateway router in the mobile device's visited network.



**Figure 7.27** ♦ Direct routing to a mobile device

While direct routing overcomes the triangle routing problem, it introduces two important additional challenges:

- A mobile-user location protocol is needed for the correspondent to query the HSS to obtain the mobile device's visited network (steps 1 and 2 in Figure 7.27). This is in addition to the protocol needed for the mobile device to register its location with its HSS.
- When the mobile device moves from one visited network to another, how will the correspondent know to now forward datagrams to the new visited network? In the case of indirect routing, this problem was easily solved by updating the HSS in the home network, and changing the tunnel endpoint to terminate at the gateway router of the new visited network. However, with direct routing, this change in visited networks is not so easily handled, as the HSS is queried by the correspondent only at the beginning of the session. Thus, additional protocol mechanisms would be required to proactively update the correspondent each time the mobile device moves. Two problems at the end of this chapter explore solutions to this problem.

## 7.6 Mobility Management in Practice

In the previous section, we identified key fundamental challenges and potential solutions in developing a network architecture to support device mobility: the notions of home and visited networks; the home network's role as a central point of information and control for mobile devices subscribed to that home network; control-plane functions needed by a home network's mobility management entity to track a mobile device roaming among visited networks; and data-plane approaches of direct and indirect routing to enable a correspondent and a mobile device to exchange datagrams. Let's now look at how these principles are put into practice! In Section 7.2.1, we'll study mobility management in 4G/5G networks; in Section 7.2.1, we'll look at Mobile IP, which has been proposed for the Internet.

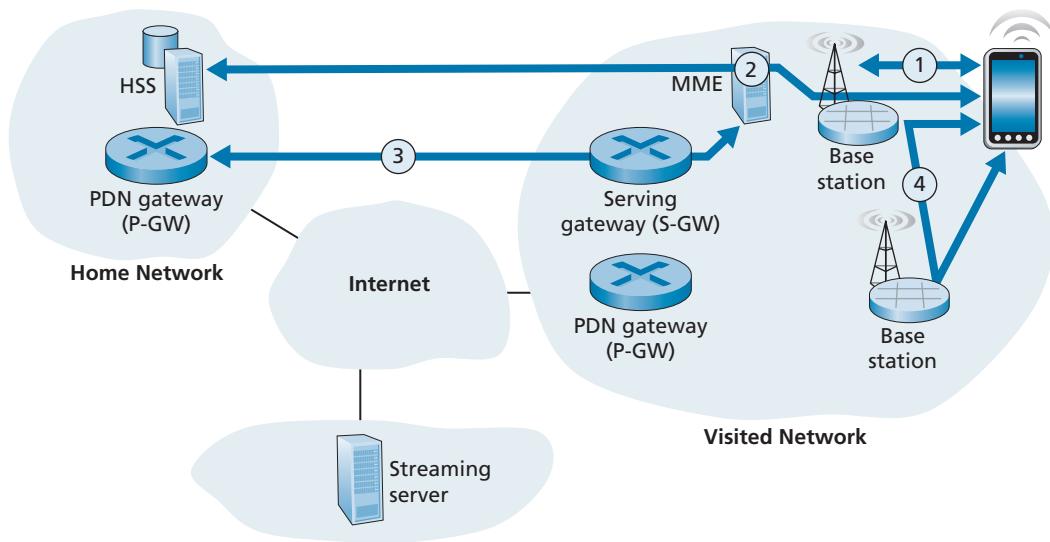
### 7.6.1 Mobility Management in 4G/5G Networks

Our earlier study of 4G and emerging 5G architectures in Section 7.4 acquainted us with all of the network elements that play a central role in 4G/5G mobility management. Let's now illustrate how those elements interoperate with each other to provide mobility services in today's 4G/5G networks [Sauter 2014; GSMA 2019b], which have their roots in earlier 3G cellular voice and data networks [Sauter 2014], and even earlier 2G voice-only networks [Mouly 1992]. This will help us synthesize what we've learned so far, allow us to introduce a few more advanced topics as well, and provide a lens into what might be in store for 5G mobility management.

Let's consider a simple scenario in which a mobile user (e.g., a passenger in a car), with a smartphone attaches to a visited 4G/5G network, begins streaming a HD video from a remote server, and then moves from the cell coverage of one 4G/5G base station to another. The four major steps in this scenario are shown in Figure 7.28:

1. *Mobile device and base station association.* The mobile device associates with a base station in the visited network.
2. *Control-plane configuration of network elements for the mobile device.* The visited and home networks establish control-plane state indicating that the mobile device is resident in the visited network.
3. *Data-plane configuration of forwarding tunnels for the mobile device.* The visited network and the home network establish tunnels through which the mobile device and streaming server can send/receive IP datagrams, using indirect routing through the home network's Packet Data Network gateway (P-GW).
4. *Mobile device handover from one base station to another.* The mobile device changes its point of attachment to the visited network, via handover from one base station to another.

Let's now consider each of these four steps in more detail.



**Figure 7.28** ♦ An example 4G/5G mobility scenario

**1. Base station association.** Recall that in Section 7.4.2, we studied the procedures by which a mobile device associates with a base station. We learned that the mobile device listens on all frequencies for primary signals being transmitted by base stations in its area. The mobile device acquires progressively more information about these base stations, ultimately selecting the base station with which to associate, and bootstrapping a control-signaling channel with that base station. As part of this association, the mobile device provides the base station with its International Mobile Subscriber Identity (IMSI), which uniquely identifies the mobile device as well as its home network and other additional subscriber information.

**2. Control-plane configuration of LTE network elements for the mobile device.** Once the mobile-device-to-base-station signaling channel has been established, the base station can contact the MME in the visited network. The MME will consult and configure a number of 4G/5G elements in both the home and visited networks to establish state on behalf of the mobile node:

- The MME will use the IMSI and other information provided by the mobile device to retrieve authentication, encryption, and available network service information for that subscriber. That information might be in the MME's local cache, retrieved from another MME that the mobile device had recently contacted, or retrieved from the HSS in the mobile device's home network. The mutual authentication process (which we will cover in more detail in Section 8.8) ensures that

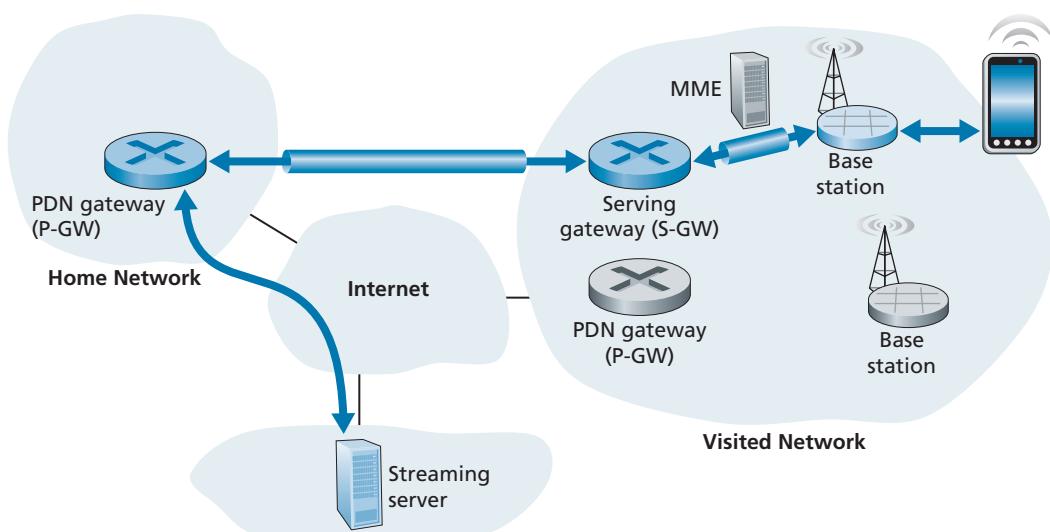
the visited network is sure about the identity of the mobile device and that the device can authenticate the network to which it is attaching.

- The MME informs the HSS in the mobile device's home network that the mobile device is now resident in the visited network, and the HSS updates its database.
- The base station and the mobile device select parameters for the data-plane channel to be established between the mobile device and the base station (recall that a control plane signaling channel is already in operation).

### 3. Data-plane configuration of forwarding tunnels for the mobile device.

The MME next configures the data plane for the mobile device, as shown in Figure 7.29. Two tunnels are established. One tunnel is between the base station and a Serving Gateway in the visited network. The second tunnel is between that Serving Gateway and the PDN Gateway router *in the mobile device's home network*. 4G LTE implements this form of symmetric indirect routing—all traffic to/from the mobile device will be tunneled through the device's home network. 4G/5G tunnels use the GPRS Tunneling Protocol (GTP), specified in [3GPP GTPv1-U 2019]. The Tunnel Endpoint ID (TEID) in the GTP header indicates which tunnel a datagram belongs, allowing multiple flows to be multiplexed and de-multiplexed by GTP between tunnel endpoints.

It is instructive to compare the configuration of tunnels in Figure 7.29 (the case of mobile roaming in a visited network) with that of Figure 7.18 (the case of mobility



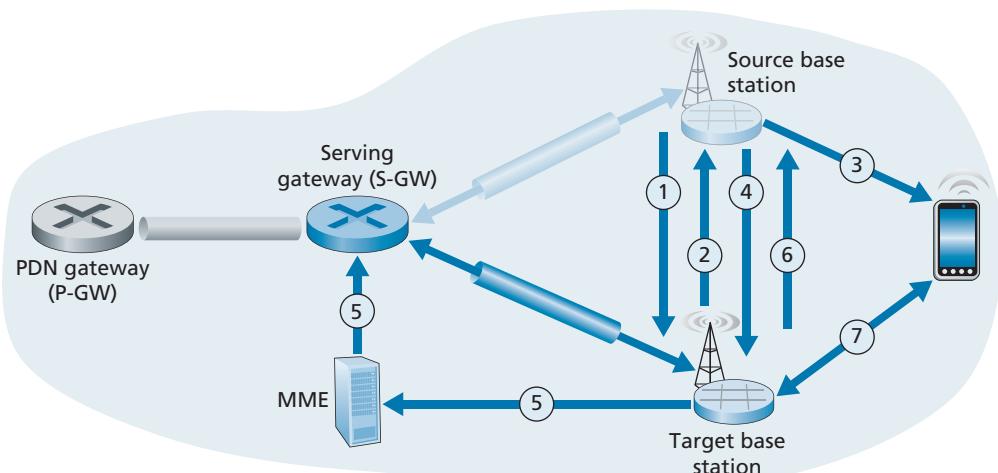
**Figure 7.29** ♦ Tunneling in 4G/5G networks between the Serving Gateway in the visited network and the PDN gateway in the home network

only within the mobile device's home network). We see that in both cases, the Serving Gateway is co-resident in the same network as the mobile device, but PDN Gateway (which is always the PDN Gateway in the mobile device's home network) may be in a different network than the mobile device. This is precisely indirect routing. An alternative to indirect routing, known as **local breakout** [GSMA 2019a] has been specified in which the Serving Gateway establishes a tunnel to the PDN Gateway in the local, visited network. In practice, however, local breakout is not widely used [Sauter 2014].

Once the tunnels have been configured and activated, the mobile device can now forward packets to/from the Internet via the PDN gateway in its home network!

**4. Handover management.** A **handover** occurs when a mobile device changes its association from one base station to another. The handover process described below is the same, regardless of whether the mobile device is resident in its home network, or is roaming in a visited network.

As shown in Figure 7.30, datagrams to/from the device are initially (before handover) forwarded to the mobile through one base station (which we'll refer to as the *source* base station), and after handover are routed to the mobile device through another base station (which we'll refer to as the *target* base station). As we will see, a handover between base stations results not only in the mobile device transmitting/receiving to/from a new base station but also in a change of the base-station side of the Serving-Gateway-to-base-station tunnel in Figure 7.29. In the simplest case of



**Figure 7.30** ♦ Steps in handing over a mobile device from the source base station to the target base station

handover, when the two base stations are near each other and in the same network, all changes occurring as a result of handover are thus relatively local. In particular, the PDN gateway being used by the Serving Gateway remains blissfully unaware of device mobility. Of course, more complicated handoff scenarios will require the use of more complex mechanisms [Sauter 2014; GSMA 2019a].

There may be several reasons for handover to occur. For example, the signal between the current base station and the mobile may have deteriorated to such an extent that communication is severely impaired. Or a cell may have become overloaded, handling a large amount of traffic; handing over mobile devices to less congested nearby cells may alleviate this congestion. A mobile device periodically measures characteristics of a beacon signal from its current base station as well as signals from nearby base stations that it can “hear.” These measurements are reported once or twice a second to the mobile device’s current (source) base station. Based on these measurements, the current loads of mobiles in nearby cells, and other factors, the source base station may choose to initiate a handover. The 4G/5G standards do not specify a specific algorithm to be used by a base station to determine whether or not to perform handover, or which target base station to choose; this is an active area of research [Zheng 2008; Alexandris 2016].

Figure 7.30 illustrates the steps involved when a source base station decides to hand over a mobile device to the target base station.

1. The current (source) base station selects the target base station, and sends a Handover Request message to the target base station.
2. The target base station checks whether it has the resources to support the mobile device and its quality of service requirements. If so, it pre-allocates channel resources (e.g., time slots) on its radio access network and other resources for that device. This pre-allocation of resources frees the mobile device from having to go through the time-consuming base-station association protocol discussed earlier, allowing handover to be executed as fast as possible. The target base station replies to the source base station with a Handover Request Acknowledge message, containing all the information at the target base station that the mobile device will need to associate with the new base station.
3. The source base station receives the Handover Request Acknowledgement message and informs the mobile device of the target base station’s identity and channel access information. At this point, the mobile device can begin sending/receiving datagrams to/from the new target base station. From the mobile device’s point of view, handover is now complete! However, there is still a bit of work to be done within the network.
4. The source base station will also stop forwarding datagrams to the mobile device and instead forward any tunneled datagrams it receives to the target base station, which will later forward these datagrams to the mobile device.
5. The target base station informs the MME that it (the target base station) will be the new base station servicing the mobile device. The MME, in turn, signals

to the Serving Gateway and the target base station to reconfigure the Serving-Gateway-to-base-station tunnel to terminate at the target base station, rather than at the source base station.

6. The target base station confirms back to the source base station that the tunnel has been reconfigured, allowing the source base station to release resources associated with that mobile device.
7. At this point, the target base station can also begin delivering datagrams to the mobile device, including datagrams forwarded to the target base station by the source base station during handover, as well as datagrams newly arriving on the reconfigured tunnel from the Serving Gateway. It can also forward outgoing datagrams received from the mobile device into the tunnel to the Serving Gateway.

The roaming configurations in today's 4G LTE networks, such as that discussed above, will also be used in future emerging 5G networks [GSMA 2019c]. Recall, however, from our discussion in Section 7.4.6 that the 5G networks will be denser, with significantly smaller cell sizes. This will make handover an even more critically important network function. In addition, low handover latency will be critical for many real-time 5G applications. The migration of the cellular network control plane to the SDN framework that we studied earlier in Chapter 5 [GSMA 2018b; Condoluci 2018] promises to enable implementations of a higher-capacity, lower-latency 5G cellular network control plane. The application of SDN in a 5G context is the subject of considerable research [Giust 2015; Ordonez-Lucena 2017; Nguyen 2016].

### 7.6.2 Mobile IP

Today's Internet does not have any widely deployed infrastructure that provides the type of services for "on the go" mobile users that we encountered for 4G/5G cellular networks. But this is certainly not due to the lack of technical solutions for providing such services in an Internet setting! Indeed, the Mobile IP architecture and protocols [RFC 5944] that we will briefly discuss below have been standardized by Internet RFCs for more than 20 years, and research has continued on new, more secure and more generalized mobility solutions [Venkataramani 2014].

Instead, it has perhaps been the lack of motivating business and use cases [Arkko 2012] and the timely development and deployment of alternative mobility solutions in cellular networks that has blunted the deployment of Mobile IP. Recall that 20 years ago, 2G cellular networks had already provided a solution for mobile voice services (the "killer app" for mobile users); additionally, next generation 3G networks supporting voice *and* data were on the horizon. Perhaps the dual technology solution—mobile services via cellular networks when we are truly mobile and "on the go" (i.e., the rightmost side of the mobility spectrum in Figure 7.24) and Internet services via 802.11 networks or wireline networks when we are stationary or moving

locally (i.e., the leftmost side of the mobility spectrum in Figure 7.24)—that we had 20 years ago and still have today will persist into the future.

It will nonetheless be instructive to briefly overview the Mobile IP standard here, as it provides many of the same services as cellular networks and implements many of the same basic mobility principles. Earlier editions of this textbook have provided a more in-depth study of Mobile IP than we will provide here; the interested reader can find this retired material on this textbook’s website. The Internet architecture and protocols for supporting mobility, collectively known as Mobile IP, are defined primarily in RFC 5944 for IPv4. Mobile IP, like 4G/5G, is a complex standard, and would require an entire book to describe in detail; indeed one such book is [Perkins 1998b]. Our modest goal here is to provide an overview of the most important aspects of Mobile IP.

The overall architecture and elements of Mobile IP are strikingly similar to that of cellular provider networks. There is a strong notion of a home network, in which a mobile device has a permanent IP address, and visited networks (known as “foreign” networks in Mobile IP), where the mobile device will be allocated a care-of-address. The home agent in Mobile IP has a similar function to the LTE HSS: it tracks the location of a mobile device by receiving updates from foreign agents in foreign networks visited by that mobile device, just as the HSS receives updates from Mobility Management Entities (MMEs) in visited networks in which a 4G mobile device resides. And both 4G/5G and Mobile IP use indirect routing to a mobile node, using tunnels to connect the gateway routers in the home and visited/foreign networks. Table 7.3 summarizes the elements of the Mobile IP architecture, along with a comparison with similar elements in 4G/5G networks

| 4G/5G element                                                                                                                                                                  | Mobile IP element                                                                                         | Discussion                                            |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------|-------------------------------------------------------|
| Home network                                                                                                                                                                   | Home network                                                                                              |                                                       |
| Visited network                                                                                                                                                                | Foreign network                                                                                           |                                                       |
| IMSI identifier                                                                                                                                                                | Permanent IP address                                                                                      | Globally unique routable address information          |
| Home Subscriber Service (HSS)                                                                                                                                                  | Home agent                                                                                                |                                                       |
| Mobility Management Entity (MME)                                                                                                                                               | Foreign agent                                                                                             |                                                       |
| Data plane: indirect forwarding via the home network, with tunneling between the home and visited network, and tunneling within the network in which the mobile device resides | Data plane: indirect forwarding via the home network, with tunneling between the home and visited network |                                                       |
| Base station (eNode-B)                                                                                                                                                         | Access Point (AP)                                                                                         | No specific AP technology is specified in Mobile IP   |
| Radio Access Network                                                                                                                                                           | WLAN                                                                                                      | No specific WLAN technology is specified in Mobile IP |

**Table 7.3** ♦ Commonalities between 4G/5G and Mobile IP architectures

The mobile IP standard consists of three main pieces:

- *Agent discovery.* Mobile IP defines the protocols used by a foreign agent to advertise its mobility services to a mobile device that wishes to attach to its network. Those services will include providing a care-of-address to the mobile device for use in the foreign network, registration of the mobile device with the home agent in the mobile device's home network, and forwarding of datagrams to/from the mobile device, among other services.
- *Registration with the home agent.* Mobile IP defines the protocols used by the mobile device and/or foreign agent to register and deregister a care-of-address with a mobile device's home agent.
- *Indirect routing of datagrams.* Mobile IP also defines the manner in which datagrams are forwarded to mobile devices by a home agent, including rules for forwarding datagrams and handling error conditions, and several forms of tunneling [RFC 2003, RFC 2004].

Again, our coverage here of Mobile IP has been intentionally brief. The interested reader should consult the references in this section, or more-detailed discussions of Mobile IP in earlier editions of this textbook.

## 7.7 Wireless and Mobility: Impact on Higher-Layer Protocols

In this chapter, we've seen that wireless networks differ significantly from their wired counterparts at both the link layer (as a result of wireless channel characteristics such as fading, multipath, and hidden terminals) and at the network layer (as a result of mobile users who change their points of attachment to the network). But are there important differences at the transport and application layers? It's tempting to think that these differences will be minor, since the network layer provides the same best-effort delivery service model to upper layers in both wired and wireless networks. Similarly, if protocols such as TCP or UDP are used to provide transport-layer services to applications in both wired and wireless networks, then the application layer should remain unchanged as well. In one sense, our intuition is right—TCP and UDP can (and do) operate in networks with wireless links. On the other hand, transport protocols in general, and TCP in particular, can sometimes have very different performance in wired and wireless networks, and it is here, in terms of performance, that differences are manifested. Let's see why.

Recall that TCP retransmits a segment that is either lost or corrupted on the path between sender and receiver. In the case of mobile users, loss can result from either network congestion (router buffer overflow) or from handover (e.g., from delays in rerouting segments to a mobile's new point of attachment to the network). In all

cases, TCP's receiver-to-sender ACK indicates only that a segment was not received intact; the sender is unaware of whether the segment was lost due to congestion, during handover, or due to detected bit errors. In all cases, the sender's response is the same—to retransmit the segment. TCP's congestion-control response is *also* the same in all cases—TCP decreases its congestion window, as discussed in Section 3.7. By unconditionally decreasing its congestion window, TCP implicitly assumes that segment loss results from congestion rather than corruption or handover. We saw in Section 7.2 that bit errors are much more common in wireless networks than in wired networks. When such bit errors occur or when handover loss occurs, there's really no reason for the TCP sender to decrease its congestion window (and thus decrease its sending rate). Indeed, it may well be the case that router buffers are empty and packets are flowing along the end-to-end path unimpeded by congestion.

Researchers realized in the early to mid 1990s that given high bit error rates on wireless links and the possibility of handover loss, TCP's congestion-control response could be problematic in a wireless setting. Three broad classes of approaches are possible for dealing with this problem:

- *Local recovery.* Local recovery protocols recover from bit errors when and where (e.g., at the wireless link) they occur, for example, the 802.11 ARQ protocol we studied in Section 7.3, or more sophisticated approaches that use both ARQ and FEC [Ayanoglu 1995] that we saw in use in 4G/5G networks in Section 7.4.2.
- *TCP sender awareness of wireless links.* In the local recovery approaches, the TCP sender is blissfully unaware that its segments are traversing a wireless link. An alternative approach is for the TCP sender and receiver to be aware of the existence of a wireless link, to distinguish between congestive losses occurring in the wired network and corruption/loss occurring at the wireless link, and to invoke congestion control only in response to congestive wired-network losses. [Liu 2003] investigates techniques for distinguishing between losses on the wired and wireless segments of an end-to-end path. [Huang 2013] provides insights on developing transport protocol mechanisms and applications that are more LTE-friendly.
- *Split-connection approaches.* In a split-connection approach [Bakre 1995], the end-to-end connection between the mobile user and the other end point is broken into two transport-layer connections: one from the mobile host to the wireless access point, and one from the wireless access point to the other communication end point (which we'll assume here is a wired host). The end-to-end connection is thus formed by the concatenation of a wireless part and a wired part. The transport layer over the wireless segment can be a standard TCP connection [Bakre 1995], or a specially tailored error recovery protocol on top of UDP. [Yavatkar 1994] investigates the use of a transport-layer selective repeat protocol over the wireless connection. Measurements reported in [Wei 2006] indicate that split TCP connections have been widely used in cellular data networks, and that significant improvements can indeed be made through the use of split TCP connections.

Our treatment of TCP over wireless links has been necessarily brief here. In-depth surveys of TCP challenges and solutions in wireless networks can be found in [Hanabali 2005; Leung 2006]. We encourage you to consult the references for details of this ongoing area of research.

Having considered transport-layer protocols, let us next consider the effect of wireless and mobility on application-layer protocols. Because of the shared nature of the wireless spectrum, applications that operate over wireless links, particularly over cellular wireless links, must treat bandwidth as a scarce commodity. For example, a Web server serving content to a Web browser executing on a 4G smartphone will likely not be able to provide the same image-rich content that it gives to a browser operating over a wired connection. Although wireless links do provide challenges at the application layer, the mobility they enable also makes possible a rich set of location-aware and context-aware applications [Baldauf 2007]. More generally, wireless and mobile networks will continue to play a key role in realizing the ubiquitous computing environments of the future [Weiser 1991]. It's fair to say that we've only seen the tip of the iceberg when it comes to the impact of wireless and mobile networks on networked applications and their protocols!

## 7.8 Summary

Wireless and mobile networks first revolutionized telephony and are now having an increasingly profound impact in the world of computer networks as well. With their anytime, anywhere, untethered access into the global network infrastructure, they are not only making network access more ubiquitous, they are also enabling an exciting new set of location-dependent services. Given the growing importance of wireless and mobile networks, this chapter has focused on the principles, common link technologies, and network architectures for supporting wireless and mobile communication.

We began this chapter with an introduction to wireless and mobile networks, drawing an important distinction between the challenges posed by the *wireless* nature of the communication links in such networks, and by the *mobility* that these wireless links enable. This allowed us to better isolate, identify, and master the key concepts in each area. We focused first on wireless communication, considering the characteristics of a wireless link in Section 7.2. In Sections 7.3 and 7.4, we examined the link-level aspects of the IEEE 802.11 (WiFi) wireless LAN standard, Bluetooth, and 4G/5G cellular networks. We then turned our attention to the issue of mobility. In Section 7.5, we identified several forms of mobility, with points along this spectrum posing different challenges and admitting different solutions. We considered the problems of locating and routing to a mobile user, as well as approaches for handing over the mobile user who dynamically moves from one point of attachment to the network to another. We examined how these issues were addressed in 4G/5G networks and in the

Mobile IP standard. Finally, we considered the impact of wireless links and mobility on transport-layer protocols and networked applications in Section 7.7.

Although we have devoted an entire chapter to the study of wireless and mobile networks, an entire book (or more) would be required to fully explore this exciting and rapidly expanding field. We encourage you to delve more deeply into this field by consulting the many references provided in this chapter.

## Homework Problems and Questions

---

### Chapter 7 Review Questions

#### SECTION 7.1

- R1. What does it mean for a wireless network to be operating in “infrastructure mode”? If the network is not in infrastructure mode, what mode of operation is it in, and what is the difference between that mode of operation and infrastructure mode?
- R2. What are the four types of wireless networks identified in our taxonomy in Section 7.1? Which of these types of wireless networks have you used?

#### SECTION 7.2

- R3. What are the differences between the following types of wireless channel impairments: path loss, multipath propagation, interference from other sources?
- R4. As a mobile node gets farther and farther away from a base station, what are two actions that a base station could take to ensure that the loss probability of a transmitted frame does not increase?

#### SECTION 7.3

- R5. Describe the role of the beacon frames in 802.11.
- R6. True or false: Before an 802.11 station transmits a data frame, it must first send an RTS frame and receive a corresponding CTS frame.
- R7. Why are acknowledgments used in 802.11 but not in wired Ethernet?
- R8. True or false: Ethernet and 802.11 use the same frame structure.
- R9. Describe how the RTS threshold works.
- R10. Suppose the IEEE 802.11 RTS and CTS frames were as long as the standard DATA and ACK frames. Would there be any advantage to using the CTS and RTS frames? Why or why not?
- R11. Section 7.3.4 discusses 802.11 mobility, in which a wireless station moves from one BSS to another within the same subnet. When the APs are interconnected with a switch, an AP may need to send a frame with a spoofed MAC address to get the switch to forward the frame properly. Why?

- R12. What are the differences between a master device in a Bluetooth network and a base station in an 802.11 network?
- R13. What is the role of the base station in 4G/5G cellular architecture? With which other 4G/5G network elements (mobile device, MME, HSS, Serving Gateway Router, PDN Gateway Router) does it *directly* communicate with in the control plane? In the data plane?
- R14. What is an International Mobile Subscriber Identity (IMSI)?
- R15. What is the role of the Home Subscriber Service (HSS) in 4G/5G cellular architecture? With which other 4G/5G network elements (mobile device, base station, MME, Serving Gateway Router, PDN Gateway Router) does it *directly* communicate with in the control plane? In the data plane?
- R16. What is the role of the Mobility Management Entity (MME) in 4G/5G cellular architecture? With which other 4G/5G network elements (mobile device, base station, HSS, Serving Gateway Router, PDN Gateway Router) does it *directly* communicate with in the control plane? In the data plane?
- R17. Describe the purpose of two tunnels in the data plane of the 4G/5G cellular architecture. When a mobile device is attached to its own home network, at which 4G/5G network element (mobile device, base station, HSS, MME, Serving Gateway Router, PDN Gateway Router) does each end of each of the two tunnels terminate?
- R18. What are the three sublayers in the link layer in the LTE protocol stack? Briefly describe their functions.
- R19. Does the LTE wireless access network use FDMA, TDMA, or both? Explain your answer.
- R20. Describe the two possible sleep modes of a 4G/5G mobile device. In each of these sleep modes, will the mobile device remain associated with the same base station between the time it goes to sleep and the time it wakes up and first sends/receives a new datagram?
- R21. What is meant by a “visited network” and a “home network” in 4G/5G cellular architecture?
- R22. List three important differences between 4G and 5G cellular networks.

#### SECTION 7.5

- R23. What does it mean that a mobile device is said to be “roaming?”
- R24. What is meant by “hand over” of a network device?
- R25. What is the difference between direct and indirect routing of datagrams to/from a roaming mobile host?
- R26. What does “triangle routing” mean?

**SECTION 7.6**

- R27. Describe the similarity and differences in tunnel configuration when a mobile device is resident in its home network, versus when it is roaming in a visited network.
- R28. When a mobile device is handed over from one base station to another in a 4G/5G network, which network element makes the decision to initiate that handover? Which network element chooses the target base station to which the mobile device will be handed over?
- R29. Describe how and when the forwarding path of datagrams entering the visited network and destined to the mobile device changes before, during, and after hand over.
- R30. Consider the following elements of the Mobile IP architecture: the home network, foreign network permanent IP address, home agent, foreign agent, data plane forwarding, Access Point (AP), and WLANs at the network edge. What are the closest equivalent elements in the 4G/5G cellular network architecture?

**SECTION 7.7**

- R31. What are three approaches that can be used to avoid having a single wireless link degrade the performance of an end-to-end transport-layer TCP connection?

**Problems**

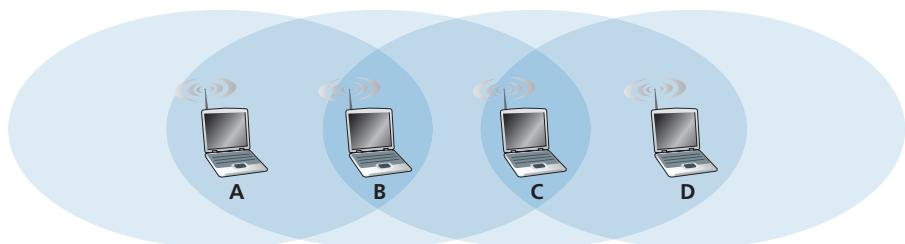
---

- P1. Consider the single-sender CDMA example in Figure 7.5. What would be the sender's output (for the 2 data bits shown) if the sender's CDMA code were  $(1, -1, 1, -1, 1, 1, 1, -1)$ ?
- P2. Consider sender 2 in Figure 7.6. What is the sender's output to the channel (before it is added to the signal from sender 1),  $Z^2_{i,m}$ ?
- P3. Suppose that the receiver in Figure 7.6 wanted to receive the data being sent by sender 2. Show (by calculation) that the receiver is indeed able to recover sender 2's data from the aggregate channel signal by using sender 2's code.
- P4. For the two-sender, two-receiver example, give an example of two CDMA codes containing 1 and 21 values that do not allow the two receivers to extract the original transmitted bits from the two CDMA senders.
- P5. Suppose there are two ISPs providing WiFi access in a particular café, with each ISP operating its own AP and having its own IP address block.
- Further suppose that by accident, each ISP has configured its AP to operate over channel 11. Will the 802.11 protocol completely break down in this situation? Discuss what happens when two stations, each associated with a different ISP, attempt to transmit at the same time.
  - Now suppose that one AP operates over channel 1 and the other over channel 11. How do your answers change?

- P6. In step 4 of the CSMA/CA protocol, a station that successfully transmits a frame begins the CSMA/CA protocol for a second frame at step 2, rather than at step 1. What rationale might the designers of CSMA/CA have had in mind by having such a station not transmit the second frame immediately (if the channel is sensed idle)?
- P7. Suppose an 802.11b station is configured to always reserve the channel with the RTS/CTS sequence. Suppose this station suddenly wants to transmit 1,500 bytes of data, and all other stations are idle at this time. As a function of SIFS and DIFS, and ignoring propagation delay and assuming no bit errors, calculate the time required to transmit the frame and receive the acknowledgment.
- P8. Consider the scenario shown in Figure 7.31, in which there are four wireless nodes, A, B, C, and D. The radio coverage of the four nodes is shown via the shaded ovals; all nodes share the same frequency. When A transmits, it can only be heard/received by B; when B transmits, both A and C can hear/receive from B; when C transmits, both B and D can hear/receive from C; when D transmits, only C can hear/receive from D.

Suppose now that each node has an infinite supply of messages that it wants to send to each of the other nodes. If a message's destination is not an immediate neighbor, then the message must be relayed. For example, if A wants to send to D, a message from A must first be sent to B, which then sends the message to C, which then sends the message to D. Time is slotted, with a message transmission time taking exactly one time slot, e.g., as in slotted Aloha. During a slot, a node can do one of the following: (i) send a message, (ii) receive a message (if exactly one message is being sent to it), (iii) remain silent. As always, if a node hears two or more simultaneous transmissions, a collision occurs and none of the transmitted messages are received successfully. You can assume here that there are no bit-level errors, and thus if exactly one message is sent, it will be received correctly by those within the transmission radius of the sender.

- a. Suppose now that an omniscient controller (i.e., a controller that knows the state of every node in the network) can command each node to do whatever it (the omniscient controller) wishes, that is, to send a message, to receive a



**Figure 7.31** ♦ Scenario for problem P8

- message, or to remain silent. Given this omniscient controller, what is the maximum rate at which a data message can be transferred from C to A, given that there are no other messages between any other source/destination pairs?
- b. Suppose now that A sends messages to B, and D sends messages to C. What is the combined maximum rate at which data messages can flow from A to B and from D to C?
  - c. Suppose now that A sends messages to B, and C sends messages to D. What is the combined maximum rate at which data messages can flow from A to B and from C to D?
  - d. Suppose now that the wireless links are replaced by wired links. Repeat questions (a) through (c) again in this wired scenario.
  - e. Now suppose we are again in the wireless scenario, and that for every data message sent from source to destination, the destination will send an ACK message back to the source (e.g., as in TCP). Also suppose that each ACK message takes up one slot. Repeat questions (a)–(c) above for this scenario.
- P9. Describe the format of the Bluetooth frame. You will have to do some reading outside of the text to find this information. Is there anything in the frame format that inherently limits the number of active nodes in a network to eight active nodes? Explain.
- P10. Consider the following idealized LTE scenario. The downstream channel (see Figure 7.22) is slotted in time, across F frequencies. There are four nodes, A, B, C, and D, reachable from the base station at rates of 10 Mbps, 5 Mbps, 2.5 Mbps, and 1 Mbps, respectively, on the downstream channel. These rates assume that the base station utilizes all time slots available on all F frequencies to send to just one station. The base station has an infinite amount of data to send to each of the nodes, and can send to any one of these four nodes using any of the F frequencies during any time slot in the downstream sub-frame.
- a. What is the maximum rate at which the base station can send to the nodes, assuming it can send to any node it chooses during each time slot? Is your solution fair? Explain and define what you mean by “fair.”
  - b. If there is a fairness requirement that each node must receive an equal amount of data during each one second interval, what is the average transmission rate by the base station (to all nodes) during the downstream sub-frame? Explain how you arrived at your answer.
  - c. Suppose that the fairness criterion is that any node can receive at most twice as much data as any other node during the sub-frame. What is the average transmission rate by the base station (to all nodes) during the sub-frame? Explain how you arrived at your answer.
- P11. In Section 7.5, one proposed solution that allowed mobile users to maintain their IP addresses as they moved among foreign networks was to have a foreign network advertise a highly specific route to the mobile user and use the existing

routing infrastructure to propagate this information throughout the network. We identified scalability as one concern. Suppose that when a mobile user moves from one network to another, the new foreign network advertises a specific route to the mobile user, and the old foreign network withdraws its route. Consider how routing information propagates in a distance-vector algorithm (particularly for the case of interdomain routing among networks that span the globe).

- a. Will other routers be able to route datagrams immediately to the new foreign network as soon as the foreign network begins advertising its route?
  - b. Is it possible for different routers to believe that different foreign networks contain the mobile user?
  - c. Discuss the timescale over which other routers in the network will eventually learn the path to the mobile users.
- P12. In 4G/5G networks, what effect will handoff have on end-to-end delays of datagrams between the source and destination?
- P13. Consider a mobile device that powers on and attaches to an LTE visited network *A*, and assume that indirect routing to the mobile device from its home network *H* is being used. Subsequently, while roaming, the device moves out of range of visited network *A* and moves into range of an LTE visited network *B*. You will design a handover process from a base station *BS.A* in visited network *A* to a base station *BS.B* in visited network *B*. Sketch the series of steps that would need to be taken, taking care to identify the network elements involved (and the networks to which they belong), to accomplish this handover. Assume that following handover, the tunnel from the home network to the visited network will terminate in visiting network *B*.
- P14. Consider again the scenario in Problem P13. But now assume that the tunnel from home network *H* to visited network *A* will continue to be used. That is, visited network *A* will serve as an anchor point following handover. (Aside: this is actually the process used for routing circuit-switched voice calls to a roaming mobile phone in 2G GSM networks.) In this case, additional tunnel(s) will need to be built to reach the mobile device in its resident visited network *B*. Once again, sketch the series of steps that would need to be taken, taking care to identify the network elements involved (and the networks to which they belong), to accomplish this handover.

What are one advantage and one disadvantage of this approach over the approach taken in your solution to Problem P13?

## Wireshark Lab: WiFi

---

At the Web site for this textbook, [www.pearsonhighered.com/cs-resources](http://www.pearsonhighered.com/cs-resources), also mirrored on the instructors' website, [http://gaia.cs.umass.edu/kurose\\_ross](http://gaia.cs.umass.edu/kurose_ross), you'll find a Wireshark lab for this chapter that captures and studies the 802.11 frames exchanged between a wireless laptop and an access point.

## AN INTERVIEW WITH...

### Deborah Estrin

Deborah Estrin is a Professor of Computer Science and Associate Dean for Impact at Cornell Tech in New York City and a Professor of Public Health at Weill Cornell Medical College. She received her Ph.D. (1985) in Computer Science from M.I.T. and her B.S. (1980) from UC Berkeley. Estrin's early research focused on the design of network protocols, including multicast and inter-domain routing. In 2002 Estrin founded the NSF-funded Science and Technology Center at UCLA, Center for Embedded Networked Sensing (CENS <http://cens.ucla.edu>). CENS launched new areas of multi-disciplinary computer systems research from sensor networks for environmental monitoring, to participatory sensing and mobile health. As described in her 2013 TEDMED talk, she explores how individuals can benefit from the pervasive data byproducts of digital and IoT interactions for health and life management. Professor Estrin is an elected member of the American Academy of Arts and Sciences (2007), the National Academy of Engineering (2009), and the National Academy of Medicine (2019). She is a Fellow of the IEEE, ACM, and AAAS. She was selected as the first ACM-W Athena Lecturer (2006), awarded the Anita Borg Institute's Women of Vision Award for Innovation (2007), inducted into the WITI hall of fame (2008), received honorary doctorates from EPFL (2008) and Uppsala University (2011), and was selected as a MacArthur Fellow (2018).



Courtesy of Deborah Estrin

#### Please describe a few of the most exciting projects you have worked on during your career. What were the biggest challenges?

In the mid-90s at USC and ISI, I had the great fortune to work with the likes of Steve Deering, Mark Handley, and Van Jacobson on the design of multicast routing protocols (in particular, PIM). I tried to carry many of the architectural design lessons from multicast into the design of ecological monitoring arrays, where for the first time I really began to take applications and multidisciplinary research seriously. The need for jointly innovating in the social and technological space is what interests me so much about my latest area of research, mobile health. The challenges in multicast routing, environmental sensing and

mobile health are as diverse as the problem domains, but what they have in common is the need to keep our eyes open to whether we have the problem definition right as we iterate between design and deployment, prototype and pilot. None of these are problems that could be solved solely analytically, or with simulation or even in constructed laboratory experiments. They challenged our ability to retain clean architectures in the presence of messy problems and contexts, and they required extensive collaboration.

**What changes and innovations do you see happening in wireless networks and mobility in the future?**

In a prior edition of this interview I said that I have never put much faith into predicting the future, but I did go on to speculate that we might see the end of feature phones (i.e., those that are not programmable and are used only for voice and text messaging) as smart phones become more and more powerful and the primary point of Internet access for many—and now not so many years later that is clearly the case. I also predicted that we would see the continued proliferation of embedded SIMs by which all sorts of devices have the ability to communicate via the cellular network at low data rates. While that has occurred, we see many devices and “Internet of Things” that use embedded WiFi and other lower power, shorter range, forms of connectivity to local hubs. I did not anticipate at that time the emergence of a large consumer wearables market or interactive voice agents like Siri and Alexa. By the time the next edition is published I expect broad proliferation of personal applications that leverage data from IoT and other digital traces.

**Where do you see the future of networking and the Internet?**

Again I think it's useful to look both back and forward. Previously I commented that the efforts in named data and software-defined networking would emerge to create a more manageable, evolvable, and richer infrastructure and more generally represent moving the role of architecture higher up in the stack. In the beginnings of the Internet, architecture was layer 4 and below, with applications being more siloed/monolithic, sitting on top. Now data and analytics dominate transport. The adoption of SDN (which I was really happy to see introduced into the 7th edition of this book) has been well beyond what I ever anticipated. That said, new challenges have emerged from higher up in the stack. Machine Learning based systems and services favor scale, particularly when they rely on continuous consumer engagement (clicks) for financial viability. The resulting information ecosystem has become far more monolithic than in earlier decades. This is a challenge for networking, the Internet, and frankly our society.

**What people inspired you professionally?**

There are three people who come to mind. First, Dave Clark, the secret sauce and under-sung hero of the Internet community. I was lucky to be around in the early days to see him act as the “organizing principle” of the IAB and Internet governance; the priest of rough consensus and running code. Second, Scott Shenker, for his intellectual brilliance, integrity, and persistence. I strive for, but rarely attain, his clarity in defining problems and solutions. He is always the first person I e-mail for advice on matters large and small. Third, my sister Judy Estrin, who had the creativity and commitment to spend the first half of her career bringing ideas and concepts to market; and now has the courage to study, write, and advise on how to rebuild it to support a healthier democracy.

**What are your recommendations for students who want careers in computer science and networking?**

First, build a strong foundation in your academic work, balanced with any and every real-world work experience you can get. As you look for a working environment, seek opportunities in problem areas you really care about and with smart teams that you can learn from and work with to build things that matter.





# Security in Computer Networks

Way back in Section 1.6, we described some of the more prevalent and damaging classes of Internet attacks, including malware attacks, denial of service, sniffing, source masquerading, and message modification and deletion. Although we have since learned a tremendous amount about computer networks, we still haven't examined how to secure networks from those attacks. Equipped with our newly acquired expertise in computer networking and Internet protocols, we'll now study in-depth secure communication and, in particular, how computer networks can be defended from those nasty bad guys.

Let us introduce Alice and Bob, two people who want to communicate and wish to do so "securely." This being a networking text, we should remark that Alice and Bob could be two routers that want to exchange routing tables securely, a client and server that want to establish a secure transport connection, or two e-mail applications that want to exchange secure e-mail—all case studies that we will consider later in this chapter. Alice and Bob are well-known fixtures in the security community, perhaps because their names are more fun than a generic entity named "A" that wants to communicate securely with a generic entity named "B." Love affairs, wartime communication, and business transactions are the commonly cited human needs for secure communications; preferring the first to the latter two, we're happy to use Alice and Bob as our sender and receiver, and imagine them in this first scenario.

We said that Alice and Bob want to communicate and wish to do so "securely," but what precisely does this mean? As we will see, security (like love) is a many-splendored thing; that is, there are many facets to security. Certainly, Alice and Bob would like for the contents of their communication to remain secret from an eavesdropper. They probably would also like to make sure that when they are

communicating, they are indeed communicating with each other, and that if their communication is tampered with by an eavesdropper, that this tampering is detected. In the first part of this chapter, we'll cover the fundamental cryptography techniques that allow for encrypting communication, authenticating the party with whom one is communicating, and ensuring message integrity.

In the second part of this chapter, we'll examine how the fundamental cryptography principles can be used to create secure networking protocols. Once again taking a top-down approach, we'll examine secure protocols in each of the (top four) layers, beginning with the application layer. We'll examine how to secure e-mail, how to secure a TCP connection, how to provide blanket security at the network layer, and how to secure a wireless LAN. In the third part of this chapter we'll consider operational security, which is about protecting organizational networks from attacks. In particular, we'll take a careful look at how firewalls and intrusion detection systems can enhance the security of an organizational network.

## 8.1 What Is Network Security?

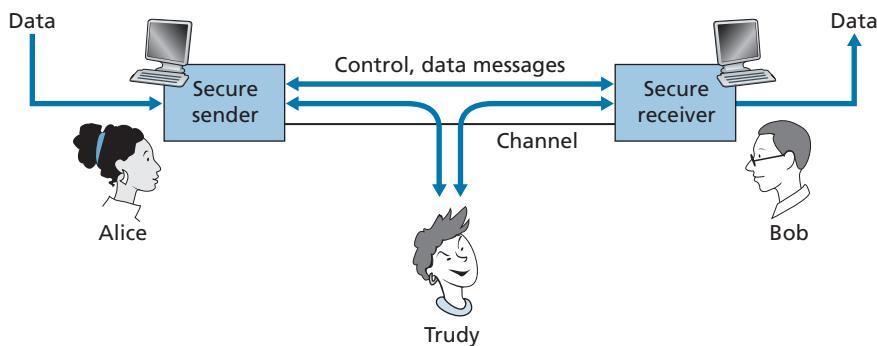
Let's begin our study of network security by returning to our lovers, Alice and Bob, who want to communicate "securely." What precisely does this mean? Certainly, Alice wants only Bob to be able to understand a message that she has sent, even though they *are* communicating over an insecure medium where an intruder (Trudy, the intruder) may intercept whatever is transmitted from Alice to Bob. Bob also wants to be sure that the message he receives from Alice was indeed sent by Alice, and Alice wants to make sure that the person with whom she is communicating is indeed Bob. Alice and Bob also want to make sure that the contents of their messages have not been altered in transit. They also want to be assured that they can communicate in the first place (i.e., that no one denies them access to the resources needed to communicate). Given these considerations, we can identify the following desirable properties of **secure communication**.

- *Confidentiality.* Only the sender and intended receiver should be able to understand the contents of the transmitted message. Because eavesdroppers may intercept the message, this necessarily requires that the message be somehow **encrypted** so that an intercepted message cannot be understood by an interceptor. This aspect of confidentiality is probably the most commonly perceived meaning of the term *secure communication*. We'll study cryptographic techniques for encrypting and decrypting data in Section 8.2.
- *Message integrity.* Alice and Bob want to ensure that the content of their communication is not altered, either maliciously or by accident, in transit. Extensions to the checksumming techniques that we encountered in reliable transport

and data link protocols can be used to provide such message integrity. We will study message integrity in Section 8.3.

- *End-point authentication.* Both the sender and receiver should be able to confirm the identity of the other party involved in the communication—to confirm that the other party is indeed who or what they claim to be. Face-to-face human communication solves this problem easily by visual recognition. When communicating entities exchange messages over a medium where they cannot see the other party, authentication is not so simple. When a user wants to access an inbox, how does the mail server verify that the user is the person he or she claims to be? We study end-point authentication in Section 8.4.
- *Operational security.* Almost all organizations (companies, universities, and so on) today have networks that are attached to the public Internet. These networks therefore can potentially be compromised. Attackers can attempt to deposit worms into the hosts in the network, obtain corporate secrets, map the internal network configurations, and launch DoS attacks. We'll see in Section 8.9 that operational devices such as firewalls and intrusion detection systems are used to counter attacks against an organization's network. A firewall sits between the organization's network and the public network, controlling packet access to and from the network. An intrusion detection system performs "deep packet inspection," alerting the network administrators about suspicious activity.

Having established what we mean by network security, let's next consider exactly what information an intruder may have access to, and what actions can be taken by the intruder. Figure 8.1 illustrates the scenario. Alice, the sender, wants to send data to Bob, the receiver. In order to exchange data securely, while meeting the requirements of confidentiality, end-point authentication, and message integrity, Alice and Bob will exchange control messages and data messages (in much the same way that TCP senders and receivers exchange control segments and data segments).



**Figure 8.1** ♦ Sender, receiver, and intruder (Alice, Bob, and Trudy)

All or some of these messages will typically be encrypted. As discussed in Section 1.6, an intruder can potentially perform

- *eavesdropping*—sniffing and recording control and data messages on the channel.
- *modification, insertion, or deletion* of messages or message content.

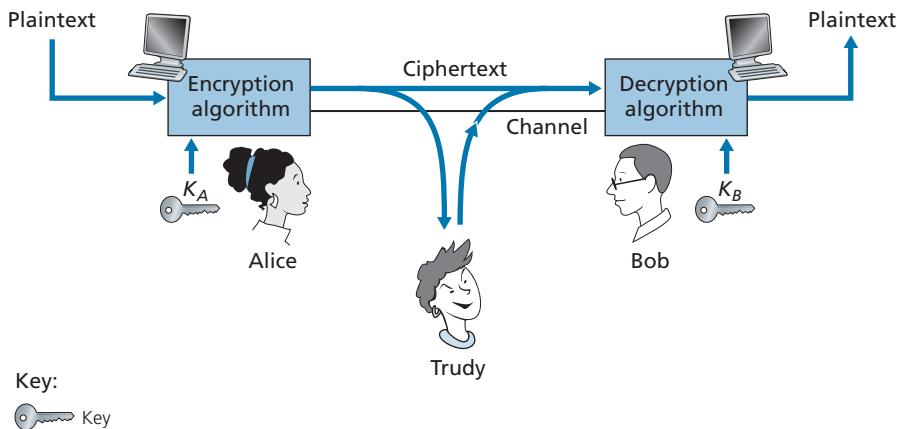
As we'll see, unless appropriate countermeasures are taken, these capabilities allow an intruder to mount a wide variety of security attacks: snooping on communication (possibly stealing passwords and data), impersonating another entity, hijacking an ongoing session, denying service to legitimate network users by overloading system resources, and so on. A summary of reported attacks is maintained at the CERT Coordination Center [CERT 2020].

Having established that there are indeed real threats loose in the Internet, what are the Internet equivalents of Alice and Bob, our friends who need to communicate securely? Certainly, Bob and Alice might be human users at two end systems, for example, a real Alice and a real Bob who really do want to exchange secure e-mail. They might also be participants in an electronic commerce transaction. For example, a real Bob might want to transfer his credit card number securely to a Web server to purchase an item online. Similarly, a real Alice might want to interact with her bank online. The parties needing secure communication might themselves also be part of the network infrastructure. Recall that the domain name system (DNS, see Section 2.4) or routing daemons that exchange routing information (see Chapter 5) require secure communication between two parties. The same is true for network management applications, a topic we examined in Chapter 5). An intruder that could actively interfere with DNS lookups (as discussed in Section 2.4), routing computations (Sections 5.3 and 5.4), or network management functions (Sections 5.5 and 5.7) could wreak havoc in the Internet.

Having now established the framework, a few of the most important definitions, and the need for network security, let us next delve into cryptography. While the use of cryptography in providing confidentiality is self-evident, we'll see shortly that it is also central to providing end-point authentication and message integrity—making cryptography a cornerstone of network security.

## 8.2 Principles of Cryptography

Although cryptography has a long history dating back at least as far as Julius Caesar, modern cryptographic techniques, including many of those used in the Internet, are based on advances made in the past 30 years. Kahn's book, *The Codebreakers* [Kahn 1967], and Singh's book, *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography* [Singh 1999], provide a fascinating look at the



**Figure 8.2** ♦ Cryptographic components

long history of cryptography. A complete discussion of cryptography itself requires a complete book [Bishop 2003; Kaufman 2002; Schneier 2015] and so we only touch on the essential aspects of cryptography, particularly as they are practiced on the Internet. We also note that while our focus in this section will be on the use of cryptography for confidentiality, we'll see shortly that cryptographic techniques are inextricably woven into authentication, message integrity, nonrepudiation, and more.

Cryptographic techniques allow a sender to disguise data so that an intruder can gain no information from the intercepted data. The receiver, of course, must be able to recover the original data from the disguised data. Figure 8.2 illustrates some of the important terminology.

Suppose now that Alice wants to send a message to Bob. Alice's message in its original form (e.g., "Bob, I love you. Alice") is known as **plaintext**, or **cleartext**. Alice encrypts her plaintext message using an **encryption algorithm** so that the encrypted message, known as **ciphertext**, looks unintelligible to any intruder. Interestingly, in many modern cryptographic systems, including those used in the Internet, the encryption technique itself is *known*—published, standardized, and available to everyone (e.g., [RFC 1321; RFC 3447; RFC 2420; NIST 2001]), even a potential intruder! Clearly, if everyone knows the method for encoding data, then there must be some secret information that prevents an intruder from decrypting the transmitted data. This is where keys come in.

In Figure 8.2, Alice provides a **key**,  $K_A$ , a string of numbers or characters, as input to the encryption algorithm. The encryption algorithm takes the key and the plaintext message,  $m$ , as input and produces ciphertext as output. The notation  $K_A(m)$  refers to the ciphertext form (encrypted using the key  $K_A$ ) of the plaintext message,  $m$ . The actual encryption algorithm that uses key  $K_A$  will be evident from the context. Similarly, Bob will provide a key,  $K_B$ , to the **decryption algorithm**

that takes the ciphertext and Bob’s key as input and produces the original plaintext as output. That is, if Bob receives an encrypted message  $K_A(m)$ , he decrypts it by computing  $K_B(K_A(m)) = m$ . In **symmetric key systems**, Alice’s and Bob’s keys are identical and are secret. In **public key systems**, a pair of keys is used. One of the keys is known to both Bob and Alice (indeed, it is known to the whole world). The other key is known only by either Bob or Alice (but not both). In the following two subsections, we consider symmetric key and public key systems in more detail.

### 8.2.1 Symmetric Key Cryptography

All cryptographic algorithms involve substituting one thing for another, for example, taking a piece of plaintext and then computing and substituting the appropriate ciphertext to create the encrypted message. Before studying a modern key-based cryptographic system, let us first get our feet wet by studying a very old, very simple symmetric key algorithm attributed to Julius Caesar, known as the **Caesar cipher** (a cipher is a method for encrypting data).

For English text, the Caesar cipher would work by taking each letter in the plaintext message and substituting the letter that is  $k$  letters later (allowing wraparound; that is, having the letter  $z$  followed by the letter  $a$ ) in the alphabet. For example, if  $k = 3$ , then the letter  $a$  in plaintext becomes  $d$  in ciphertext;  $b$  in plaintext becomes  $e$  in ciphertext, and so on. Here, the value of  $k$  serves as the key. As an example, the plaintext message “bob, i love you. Alice” becomes “ere, l oryh brx. dolfh” in ciphertext. While the ciphertext does indeed look like gibberish, it wouldn’t take long to break the code if you knew that the Caesar cipher was being used, as there are only 25 possible key values.

An improvement on the Caesar cipher is the **monoalphabetic cipher**, which also substitutes one letter of the alphabet with another letter of the alphabet. However, rather than substituting according to a regular pattern (e.g., substitution with an offset of  $k$  for all letters), any letter can be substituted for any other letter, as long as each letter has a unique substitute letter, and vice versa. The substitution rule in Figure 8.3 shows one possible rule for encoding plaintext.

The plaintext message “bob, i love you. Alice” becomes “nkn, s gktc wky. Mgsbc.” Thus, as in the case of the Caesar cipher, this looks like gibberish. A monoalphabetic cipher would also appear to be better than the Caesar cipher in that there are  $26!$  (on the order of  $10^{26}$ ) possible pairings of letters rather than 25 possible pairings. A brute-force approach of trying all  $10^{26}$  possible pairings

|                    |                                                     |
|--------------------|-----------------------------------------------------|
| Plaintext letter:  | a b c d e f g h i j k l m n o p q r s t u v w x y z |
| Ciphertext letter: | m n b v c x z a s d f g h j k l p o i u y t r e w q |

**Figure 8.3** ♦ A monoalphabetic cipher

would require far too much work to be a feasible way of breaking the encryption algorithm and decoding the message. However, by statistical analysis of the plain-text language, for example, knowing that the letters *e* and *t* are the most frequently occurring letters in typical English text (accounting for 13 percent and 9 percent of letter occurrences), and knowing that particular two-and three-letter occurrences of letters appear quite often together (for example, “in,” “it,” “the,” “ion,” “ing,” and so forth) make it relatively easy to break this code. If the intruder has some knowledge about the possible contents of the message, then it is even easier to break the code. For example, if Trudy the intruder is Bob’s wife and suspects Bob of having an affair with Alice, then she might suspect that the names “bob” and “alice” appear in the text. If Trudy knew for certain that those two names appeared in the ciphertext and had a copy of the example ciphertext message above, then she could immediately determine seven of the 26 letter pairings, requiring  $10^9$  fewer possibilities to be checked by a brute-force method. Indeed, if Trudy suspected Bob of having an affair, she might well expect to find some other choice words in the message as well.

When considering how easy it might be for Trudy to break Bob and Alice’s encryption scheme, one can distinguish three different scenarios, depending on what information the intruder has.

- *Ciphertext-only attack.* In some cases, the intruder may have access only to the intercepted ciphertext, with no certain information about the contents of the plain-text message. We have seen how statistical analysis can help in a **ciphertext-only attack** on an encryption scheme.
- *Known-plaintext attack.* We saw above that if Trudy somehow knew for sure that “bob” and “alice” appeared in the ciphertext message, then she could have determined the (plaintext, ciphertext) pairings for the letters *a*, *l*, *i*, *c*, *e*, *b*, and *o*. Trudy might also have been fortunate enough to have recorded all of the ciphertext transmissions and then found Bob’s own decrypted version of one of the transmissions scribbled on a piece of paper. When an intruder knows some of the (plaintext, ciphertext) pairings, we refer to this as a **known-plaintext attack** on the encryption scheme.
- *Chosen-plaintext attack.* In a **chosen-plaintext attack**, the intruder is able to choose the plaintext message and obtain its corresponding ciphertext form. For the simple encryption algorithms we’ve seen so far, if Trudy could get Alice to send the message, “The quick brown fox jumps over the lazy dog,” she could completely break the encryption scheme. We’ll see shortly that for more sophisticated encryption techniques, a chosen-plaintext attack does not necessarily mean that the encryption technique can be broken.

Five hundred years ago, techniques improving on monoalphabetic encryption, known as **polyalphabetic encryption**, were invented. The idea behind polyalphabetic encryption is to use multiple monoalphabetic ciphers, with a specific

|                          |                                                     |
|--------------------------|-----------------------------------------------------|
| Plaintext letter:        | a b c d e f g h i j k l m n o p q r s t u v w x y z |
| C <sub>1</sub> (k = 5):  | f g h i j k l m n o p q r s t u v w x y z a b c d e |
| C <sub>2</sub> (k = 19): | t u v w x y z a b c d e f g h i j k l m n o p q r s |

**Figure 8.4** ♦ A polyalphabetic cipher using two Caesar ciphers

monoalphabetic cipher to encode a letter in a specific position in the plaintext message. Thus, the same letter, appearing in different positions in the plaintext message, might be encoded differently. An example of a polyalphabetic encryption scheme is shown in Figure 8.4. It has two Caesar ciphers (with  $k = 5$  and  $k = 19$ ), shown as rows. We might choose to use these two Caesar ciphers, C<sub>1</sub> and C<sub>2</sub>, in the repeating pattern C<sub>1</sub>, C<sub>2</sub>, C<sub>2</sub>, C<sub>1</sub>, C<sub>2</sub>. That is, the first letter of plaintext is to be encoded using C<sub>1</sub>, the second and third using C<sub>2</sub>, the fourth using C<sub>1</sub>, and the fifth using C<sub>2</sub>. The pattern then repeats, with the sixth letter being encoded using C<sub>1</sub>, the seventh with C<sub>2</sub>, and so on. The plaintext message “bob, i love you.” is thus encrypted “ghu, n etox dhz.” Note that the first b in the plaintext message is encrypted using C<sub>1</sub>, while the second b is encrypted using C<sub>2</sub>. In this example, the encryption and decryption “key” is the knowledge of the two Caesar keys ( $k = 5, k = 19$ ) and the pattern C<sub>1</sub>, C<sub>2</sub>, C<sub>2</sub>, C<sub>1</sub>, C<sub>2</sub>.

### Block Ciphers

Let us now move forward to modern times and examine how symmetric key encryption is done today. We focus on block ciphers, which are used in many secure Internet protocols, including PGP (for secure e-mail), TLS (for securing TCP connections), and IPsec (for securing the network-layer transport).

In a block cipher, the message to be encrypted is processed in blocks of  $k$  bits. For example, if  $k = 64$ , then the message is broken into 64-bit blocks, and each block is encrypted independently. To encode a block, the cipher uses a one-to-one mapping to map the  $k$ -bit block of cleartext to a  $k$ -bit block of ciphertext. Let’s look at an example. Suppose that  $k = 3$ , so that the block cipher maps 3-bit inputs (cleartext) to 3-bit outputs (ciphertext). One possible mapping is given in Table 8.1. Notice that this is a one-to-one mapping; that is, there is a different output for each input. This block cipher breaks the message up into 3-bit blocks and encrypts each block according to the above mapping. You should verify that the message 010110001111 gets encrypted into 101000111001.

Continuing with this 3-bit block example, note that the mapping in Table 8.1 is just one mapping of many possible mappings. How many possible mappings are there? To answer this question, observe that a mapping is nothing more than a permutation of all the possible inputs. There are  $2^3$  (= 8) possible inputs (listed under the

| input | output | input | output |
|-------|--------|-------|--------|
| 000   | 110    | 100   | 011    |
| 001   | 111    | 101   | 010    |
| 010   | 101    | 110   | 000    |
| 011   | 100    | 111   | 001    |

**Table 8.1** ♦ A specific 3-bit block cipher

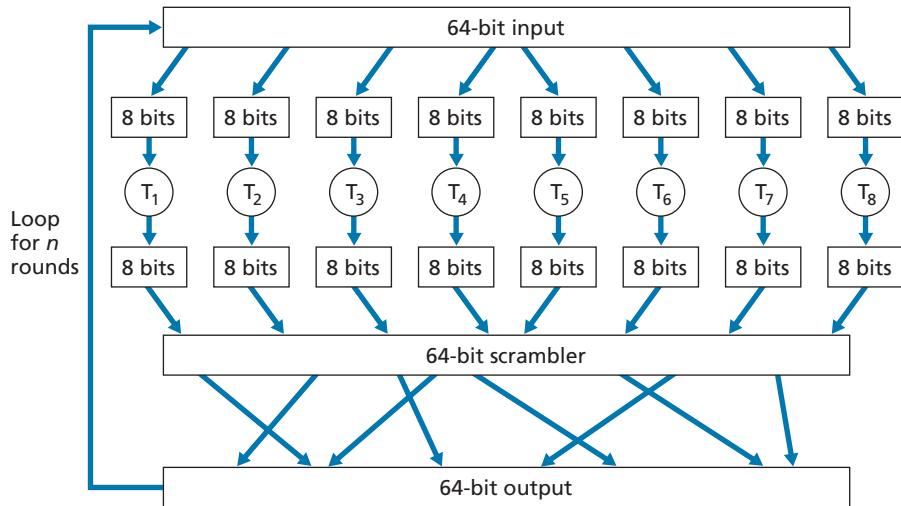
input columns). These eight inputs can be permuted in  $8! = 40,320$  different ways. Since each of these permutations specifies a mapping, there are 40,320 possible mappings. We can view each of these mappings as a key—if Alice and Bob both know the mapping (the key), they can encrypt and decrypt the messages sent between them.

The brute-force attack for this cipher is to try to decrypt ciphertext by using all mappings. With only 40,320 mappings (when  $k = 3$ ), this can quickly be accomplished on a desktop PC. To thwart brute-force attacks, block ciphers typically use much larger blocks, consisting of  $k = 64$  bits or even larger. Note that the number of possible mappings for a general  $k$ -block cipher is  $2^k!$ , which is astronomical for even moderate values of  $k$  (such as  $k = 64$ ).

Although full-table block ciphers, as just described, with moderate values of  $k$  can produce robust symmetric key encryption schemes, they are unfortunately difficult to implement. For  $k = 64$  and for a given mapping, Alice and Bob would need to maintain a table with  $2^{64}$  input values, which is an infeasible task. Moreover, if Alice and Bob were to change keys, they would have to each regenerate the table. Thus, a full-table block cipher, providing predetermined mappings between all inputs and outputs (as in the example above), is simply out of the question.

Instead, block ciphers typically use functions that simulate randomly permuted tables. An example (adapted from [Kaufman 2002]) of such a function for  $k = 64$  bits is shown in Figure 8.5. The function first breaks a 64-bit block into 8 chunks, with each chunk consisting of 8 bits. Each 8-bit chunk is processed by an 8-bit to 8-bit table, which is of manageable size. For example, the first chunk is processed by the table denoted by  $T_1$ . Next, the 8 output chunks are reassembled into a 64-bit block. The positions of the 64 bits in the block are then scrambled (permuted) to produce a 64-bit output. This output is fed back to the 64-bit input, where another cycle begins. After  $n$  such cycles, the function provides a 64-bit block of ciphertext. The purpose of the rounds is to make each input bit affect most (if not all) of the final output bits. (If only one round were used, a given input bit would affect only 8 of the 64 output bits.) The key for this block cipher algorithm would be the eight permutation tables (assuming the scramble function is publicly known).

Today there are a number of popular block ciphers, including DES (standing for Data Encryption Standard), 3DES, and AES (standing for Advanced Encryption



**Figure 8.5** ♦ An example of a block cipher

Standard). Each of these standards uses functions, rather than predetermined tables, along the lines of Figure 8.5 (albeit more complicated and specific to each cipher). Each of these algorithms also uses a string of bits for a key. For example, DES uses 64-bit blocks with a 56-bit key. AES uses 128-bit blocks and can operate with keys that are 128, 192, and 256 bits long. An algorithm's key determines the specific “mini-table” mappings and permutations within the algorithm's internals. The brute-force attack for each of these ciphers is to cycle through all the keys, applying the decryption algorithm with each key. Observe that with a key length of  $n$ , there are  $2^n$  possible keys. NIST [NIST 2001] estimates that a machine that could crack 56-bit DES in one second (that is, try all  $2^{56}$  keys in one second) would take approximately 149 trillion years to crack a 128-bit AES key.

### Cipher-Block Chaining

In computer networking applications, we typically need to encrypt long messages or long streams of data. If we apply a block cipher as described by simply chopping up the message into  $k$ -bit blocks and independently encrypting each block, a subtle but important problem occurs. To see this, observe that two or more of the cleartext blocks can be identical. For example, the cleartext in two or more blocks could be “HTTP/1.1”. For these identical blocks, a block cipher would, of course, produce the same ciphertext. An attacker could potentially guess the cleartext when it sees identical ciphertext blocks and may even be able to decrypt the entire message by identifying identical ciphertext blocks and using knowledge about the underlying protocol structure [Kaufman 2002].

To address this problem, we can mix some randomness into the ciphertext so that identical plaintext blocks produce different ciphertext blocks. To explain this idea, let  $m(i)$  denote the  $i$ th plaintext block,  $c(i)$  denote the  $i$ th ciphertext block, and  $a \oplus b$  denote the exclusive-or (XOR) of two bit strings,  $a$  and  $b$ . (Recall that the  $0 \oplus 0 = 1 \oplus 1 = 0$  and  $0 \oplus 1 = 1 \oplus 0 = 1$ , and the XOR of two bit strings is done on a bit-by-bit basis. So, for example,  $10101010 \oplus 11110000 = 01011010$ .) Also, denote the block-cipher encryption algorithm with key  $S$  as  $K_S$ . The basic idea is as follows. The sender creates a random  $k$ -bit number  $r(i)$  for the  $i$ th block and calculates  $c(i) = K_S(m(i) \oplus r(i))$ . Note that a new  $k$ -bit random number is chosen for each block. The sender then sends  $c(1), r(1), c(2), r(2), c(3), r(3)$ , and so on. Since the receiver receives  $c(i)$  and  $r(i)$ , it can recover each block of the plaintext by computing  $m(i) = K_S(c(i)) \oplus r(i)$ . It is important to note that, although  $r(i)$  is sent in the clear and thus can be sniffed by Trudy, she cannot obtain the plaintext  $m(i)$ , since she does not know the key  $K_S$ . Also note that if two plaintext blocks  $m(i)$  and  $m(j)$  are the same, the corresponding ciphertext blocks  $c(i)$  and  $c(j)$  will be different (as long as the random numbers  $r(i)$  and  $r(j)$  are different, which occurs with very high probability).

As an example, consider the 3-bit block cipher in Table 8.1. Suppose the plaintext is 010010010. If Alice encrypts this directly, without including the randomness, the resulting ciphertext becomes 101101101. If Trudy sniffs this ciphertext, because each of the three cipher blocks is the same, she can correctly surmise that each of the three plaintext blocks are the same. Now suppose instead Alice generates the random blocks  $r(1) = 001$ ,  $r(2) = 111$ , and  $r(3) = 100$  and uses the above technique to generate the ciphertext  $c(1) = 100$ ,  $c(2) = 010$ , and  $c(3) = 000$ . Note that the three ciphertext blocks are different even though the plaintext blocks are the same. Alice then sends  $c(1), r(1), c(2)$ , and  $r(2)$ . You should verify that Bob can obtain the original plaintext using the shared key  $K_S$ .

The astute reader will note that introducing randomness solves one problem but creates another: namely, Alice must transmit twice as many bits as before. Indeed, for each cipher bit, she must now also send a random bit, doubling the required bandwidth. In order to have our cake and eat it too, block ciphers typically use a technique called **Cipher Block Chaining (CBC)**. The basic idea is to send only *one random value along with the very first message, and then have the sender and receiver use the computed coded blocks in place of the subsequent random number*. Specifically, CBC operates as follows:

1. Before encrypting the message (or the stream of data), the sender generates a random  $k$ -bit string, called the **Initialization Vector (IV)**. Denote this initialization vector by  $c(0)$ . The sender sends the IV to the receiver *in cleartext*.
2. For the first block, the sender calculates  $m(1) \oplus c(0)$ , that is, calculates the exclusive-or of the first block of cleartext with the IV. It then runs the result through the block-cipher algorithm to get the corresponding ciphertext block; that is,  $c(1) = K_S(m(1) \oplus c(0))$ . The sender sends the encrypted block  $c(1)$  to the receiver.
3. For the  $i$ th block, the sender generates the  $i$ th ciphertext block from  $c(i) = K_S(m(i) \oplus c(i - 1))$ .

Let's now examine some of the consequences of this approach. First, the receiver will still be able to recover the original message. Indeed, when the receiver receives  $c(i)$ , it decrypts it with  $K_S$  to obtain  $s(i) = m(i) \oplus c(i - 1)$ ; since the receiver also knows  $c(i - 1)$ , it then obtains the cleartext block from  $m(i) = s(i) \oplus c(i - 1)$ . Second, even if two cleartext blocks are identical, the corresponding ciphertexts (almost always) will be different. Third, although the sender sends the IV in the clear, an intruder will still not be able to decrypt the ciphertext blocks, since the intruder does not know the secret key,  $S$ . Finally, the sender only sends one overhead block (the IV), thereby negligibly increasing the bandwidth usage for long messages (consisting of hundreds of blocks).

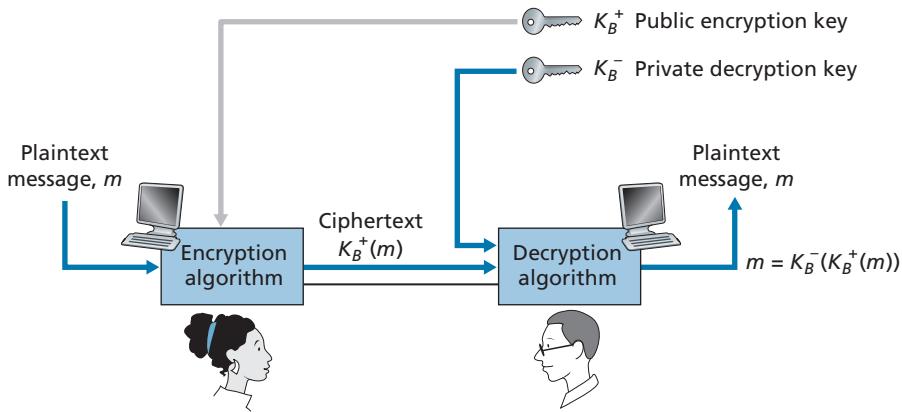
As an example, let's now determine the ciphertext for the 3-bit block cipher in Table 8.1 with plaintext 010010010 and IV =  $c(0) = 001$ . The sender first uses the IV to calculate  $c(1) = K_S(m(1) \oplus c(0)) = 100$ . The sender then calculates  $c(2) = K_S(m(2) \oplus c(1)) = K_S(010 \oplus 100) = 000$ , and  $c(3) = K_S(m(3) \oplus c(2)) = K_S(010 \oplus 000) = 101$ . The reader should verify that the receiver, knowing the IV and  $K_S$  can recover the original plaintext.

CBC has an important consequence when designing secure network protocols: we'll need to provide a mechanism within the protocol to distribute the IV from sender to receiver. We'll see how this is done for several protocols later in this chapter.

### 8.2.2 Public Key Encryption

For more than 2,000 years (since the time of the Caesar cipher and up to the 1970s), encrypted communication required that the two communicating parties share a common secret—the symmetric key used for encryption and decryption. One difficulty with this approach is that the two parties must somehow agree on the shared key; but to do so in itself requires secure communication. Perhaps the parties could first meet and agree on the key in person (for example, two of Caesar's centurions might meet at the Roman baths) and thereafter communicate with encryption. In a networked world, however, communicating parties may never meet and may never converse except over the network.

Is it possible for two parties to communicate with encryption without having a shared secret key that is known in advance? In 1976, Diffie and Hellman [Diffie 1976] demonstrated an algorithm (known now as Diffie-Hellman Key Exchange) to do just that—a radically different and marvelously elegant approach toward secure communication that has led to the development of today's public key cryptography systems. We'll see shortly that public key cryptography systems also have several wonderful properties that make them useful not only for encryption, but for authentication and digital signatures as well. Interestingly, it has come to light that ideas similar to those in [Diffie 1976] and [RSA 1978] had been independently developed in the early 1970s in a series of secret reports by researchers at the Communications-Electronics Security Group in the United Kingdom [Ellis 1987].



**Figure 8.6** ♦ Public key cryptography

As is often the case, great ideas can spring up independently in many places; fortunately, public key advances took place not only in private, but also in the public view, as well.

The use of public key cryptography is conceptually quite simple. Suppose Alice wants to communicate with Bob. As shown in Figure 8.6, rather than Bob and Alice sharing a single secret key (as in the case of symmetric key systems), Bob (the recipient of Alice’s messages) instead has two keys—a **public key** that is available to *everyone* in the world (including Trudy the intruder) and a **private key** that is known only to Bob. We will use the notation  $K_B^+$  and  $K_B^-$  to refer to Bob’s public and private keys, respectively. In order to communicate with Bob, Alice first fetches Bob’s public key. Alice then encrypts her message,  $m$ , to Bob using Bob’s public key and a known (for example, standardized) encryption algorithm; that is, Alice computes  $K_B^+(m)$ . Bob receives Alice’s encrypted message and uses his private key and a known (for example, standardized) decryption algorithm to decrypt Alice’s encrypted message. That is, Bob computes  $K_B^-(K_B^+(m))$ . We will see below that there are encryption/decryption algorithms and techniques for choosing public and private keys such that  $K_B^-(K_B^+(m)) = m$ ; that is, applying Bob’s public key,  $K_B^+$ , to a message,  $m$  (to get  $K_B^+(m)$ ), and then applying Bob’s private key,  $K_B^-$ , to the encrypted version of  $m$  (that is, computing  $K_B^-(K_B^+(m))$ ) gives back  $m$ . This is a remarkable result! In this manner, Alice can use Bob’s publicly available key to send a secret message to Bob without either of them having to distribute any secret keys! We will see shortly that we can interchange the public key and private key encryption and get the same remarkable result—that is,  $K_B^-(K_B^+(m)) = K_B^+(K_B^-(m)) = m$ .

Although public-key cryptography is appealing, one concern immediately springs to mind. Since Bob’s encryption key is public, anyone can send an encrypted

message to Bob, including Alice or someone *pretending* to be Alice. In the case of a single shared secret key, the fact that the sender knows the secret key implicitly identifies the sender to the receiver. In the case of public key cryptography, however, this is no longer the case since anyone can send an encrypted message to Bob using Bob's publicly available key. A digital signature, a topic we will study in Section 8.3, is needed to bind a sender to a message.

## RSA

While there may be many algorithms that address these concerns, the **RSA algorithm** (named after its founders, Ron Rivest, Adi Shamir, and Leonard Adleman) has become almost synonymous with public key cryptography. Let's first see how RSA works and then examine why it works.

RSA makes extensive use of arithmetic operations using modulo- $n$  arithmetic. So let's briefly review modular arithmetic. Recall that  $x \bmod n$  simply means the remainder of  $x$  when divided by  $n$ ; so, for example,  $19 \bmod 5 = 4$ . In modular arithmetic, one performs the usual operations of addition, multiplication, and exponentiation. However, the result of each operation is replaced by the integer remainder that is left when the result is divided by  $n$ . Adding and multiplying with modular arithmetic is facilitated with the following handy facts:

$$[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$$

$$[(a \bmod n) - (b \bmod n)] \bmod n = (a - b) \bmod n$$

$$[(a \bmod n) \cdot (b \bmod n)] \bmod n = (a \cdot b) \bmod n$$

It follows from the third fact that  $(a \bmod n)^d \bmod n = a^d \bmod n$ , which is an identity that we will soon find very useful.

Now suppose that Alice wants to send to Bob an RSA-encrypted message, as shown in Figure 8.6. In our discussion of RSA, let's always keep in mind that a message is nothing but a bit pattern, and every bit pattern can be uniquely represented by an integer number (along with the length of the bit pattern). For example, suppose a message is the bit pattern 1001; this message can be represented by the decimal integer 9. Thus, when encrypting a message with RSA, it is equivalent to encrypting the unique integer number that represents the message.

There are two interrelated components of RSA:

- The choice of the public key and the private key
- The encryption and decryption algorithm

To generate the public and private RSA keys, Bob performs the following steps:

1. Choose two large prime numbers,  $p$  and  $q$ . How large should  $p$  and  $q$  be? The larger the values, the more difficult it is to break RSA, but the longer it takes

to perform the encoding and decoding. RSA Laboratories recommends that the product of  $p$  and  $q$  be on the order of 1,024 bits. For a discussion of how to find large prime numbers, see [Caldwell 2020].

2. Compute  $n = pq$  and  $z = (p - 1)(q - 1)$ .
3. Choose a number,  $e$ , less than  $n$ , that has no common factors (other than 1) with  $z$ . (In this case,  $e$  and  $z$  are said to be relatively prime.) The letter  $e$  is used since this value will be used in encryption.
4. Find a number,  $d$ , such that  $ed - 1$  is exactly divisible (that is, with no remainder) by  $z$ . The letter  $d$  is used because this value will be used in decryption. Put another way, given  $e$ , we choose  $d$  such that

$$ed \bmod z = 1$$

5. The public key that Bob makes available to the world,  $K_B^+$ , is the pair of numbers  $(n, e)$ ; his private key,  $K_B^-$ , is the pair of numbers  $(n, d)$ .

The encryption by Alice and the decryption by Bob are done as follows:

- Suppose Alice wants to send Bob a bit pattern represented by the integer number  $m$  (with  $m < n$ ). To encode, Alice performs the exponentiation  $m^e$ , and then computes the integer remainder when  $m^e$  is divided by  $n$ . In other words, the encrypted value,  $c$ , of Alice's plaintext message,  $m$ , is

$$c = m^e \bmod n$$

The bit pattern corresponding to this ciphertext  $c$  is sent to Bob.

- To decrypt the received ciphertext message,  $c$ , Bob computes

$$m = c^d \bmod n$$

which requires the use of his private key  $(n, d)$ .

As a simple example of RSA, suppose Bob chooses  $p = 5$  and  $q = 7$ . (Admittedly, these values are far too small to be secure.) Then  $n = 35$  and  $z = 24$ . Bob chooses  $e = 5$ , since 5 and 24 have no common factors. Finally, Bob chooses  $d = 29$ , since  $5 \cdot 29 - 1$  (that is,  $ed - 1$ ) is exactly divisible by 24. Bob makes the two values,  $n = 35$  and  $e = 5$ , public and keeps the value  $d = 29$  secret. Observing these two public values, suppose Alice now wants to send the letters *l*, *o*, *v*, and *e* to Bob. Interpreting each letter as a number between 1 and 26 (with *a* being 1, and *z* being 26), Alice and Bob perform the encryption and decryption shown in Tables 8.2 and 8.3, respectively. Note that in this example, we consider each of the four letters as a distinct message. A more realistic example would be to convert the four letters into their 8-bit ASCII representations and then encrypt the integer corresponding to the resulting 32-bit bit pattern. (Such a realistic example generates numbers that are much too long to print in a textbook!)

| Plaintext Letter | $m$ : numeric representation | $m^e$   | Ciphertext $c = m^e \bmod n$ |
|------------------|------------------------------|---------|------------------------------|
| I                | 12                           | 248832  | 17                           |
| O                | 15                           | 759375  | 15                           |
| V                | 22                           | 5153632 | 22                           |
| E                | 5                            | 3125    | 10                           |

**Table 8.2** ♦ Alice’s RSA encryption,  $e = 5$ ,  $n = 35$ 

Given that the “toy” example in Tables 8.2 and 8.3 has already produced some extremely large numbers, and given that we saw earlier that  $p$  and  $q$  should each be several hundred bits long, several practical issues regarding RSA come to mind. How does one choose large prime numbers? How does one then choose  $e$  and  $d$ ? How does one perform exponentiation with large numbers? A discussion of these important issues is beyond the scope of this book; see [Kaufman 2002] and the references therein for details.

### Session Keys

We note here that the exponentiation required by RSA is a rather time-consuming process. As a result, RSA is often used in practice in combination with symmetric key cryptography. For example, if Alice wants to send Bob a large amount of encrypted data, she could do the following. First Alice chooses a key that will be used to encode the data itself; this key is referred to as a **session key**, and is denoted by  $K_S$ . Alice must inform Bob of the session key, since this is the shared symmetric key they will use with a symmetric key cipher (e.g., with DES or AES). Alice encrypts the session key using Bob’s public key, that is, computes  $c = (K_S)^e \bmod n$ . Bob receives the RSA-encrypted session key,  $c$ , and decrypts it to obtain

| Ciphertext $c$ | $c^d$                                        | $m = c^d \bmod n$ | Plaintext Letter |
|----------------|----------------------------------------------|-------------------|------------------|
| 17             | 4819685721067509150915091411825223071697     | 12                | I                |
| 15             | 127834039403948858939111232757568359375      | 15                | O                |
| 22             | 851643319086537701956194499721106030592      | 22                | V                |
| 10             | 10000000000000000000000000000000000000000000 | 5                 | E                |

**Table 8.3** ♦ Bob’s RSA decryption,  $d = 29$ ,  $n = 35$

the session key,  $K_S$ . Bob now knows the session key that Alice will use for her encrypted data transfer.

### Why Does RSA Work?

RSA encryption/decryption appears rather magical. Why should it be that by applying the encryption algorithm and then the decryption algorithm, one recovers the original message? In order to understand why RSA works, again denote  $n = pq$ , where  $p$  and  $q$  are the large prime numbers used in the RSA algorithm.

Recall that, under RSA encryption, a message (uniquely represented by an integer),  $m$ , is exponentiated to the power  $e$  using modulo- $n$  arithmetic, that is,

$$c = m^e \bmod n$$

Decryption is performed by raising this value to the power  $d$ , again using modulo- $n$  arithmetic. The result of an encryption step followed by a decryption step is thus  $(m^e \bmod n)^d \bmod n$ . Let's now see what we can say about this quantity. As mentioned earlier, one important property of modulo arithmetic is  $(a \bmod n)^d \bmod n = a^d \bmod n$  for any values  $a$ ,  $n$ , and  $d$ . Thus, using  $a = m^e$  in this property, we have

$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$$

It therefore remains to show that  $m^{ed} \bmod n = m$ . Although we're trying to remove some of the magic about why RSA works, to establish this, we'll need to use a rather magical result from number theory here. Specifically, we'll need the result that says if  $p$  and  $q$  are prime,  $n = pq$ , and  $z = (p - 1)(q - 1)$ , then  $x^y \bmod n$  is the same as  $x^{(y \bmod z)} \bmod n$  [Kaufman 2002]. Applying this result with  $x = m$  and  $y = ed$  we have

$$m^{ed} \bmod n = m^{(ed \bmod z)} \bmod n$$

But remember that we have chosen  $e$  and  $d$  such that  $ed \bmod z = 1$ . This gives us

$$m^{ed} \bmod n = m^1 \bmod n = m$$

which is exactly the result we are looking for! By first exponentiating to the power of  $e$  (that is, encrypting) and then exponentiating to the power of  $d$  (that is, decrypting), we obtain the original value,  $m$ . Even *more* wonderful is the fact that if we first exponentiate to the power of  $d$  and then exponentiate to the power of  $e$ —that is, we reverse the order of encryption and decryption, performing the decryption operation first and then applying the encryption operation—we also obtain the original value,  $m$ . This wonderful result follows immediately from the modular arithmetic:

$$(m^d \bmod n)^e \bmod n = m^{de} \bmod n = m^{ed} \bmod n = (m^e \bmod n)^d \bmod n$$

The security of RSA relies on the fact that there are no known algorithms for quickly factoring a number, in this case the public value  $n$ , into the primes  $p$  and  $q$ . If one knew  $p$  and  $q$ , then given the public value  $e$ , one could easily compute the secret key,  $d$ . On the other hand, it is not known whether or not there *exist* fast algorithms for factoring a number, and in this sense, the security of RSA is not guaranteed. With recent advances in quantum computing, and published fast factoring algorithms for quantum computers, there are concerns that RSA may not be secure forever [MIT TR 2019]. But the practical realization of these algorithms still appears to be far in the future.

Another popular public-key encryption algorithm is the Diffie-Hellman algorithm, which we will briefly explore in the homework problems. Diffie-Hellman is not as versatile as RSA in that it cannot be used to encrypt messages of arbitrary length; it can be used, however, to establish a symmetric session key, which is in turn used to encrypt messages.

## 8.3 Message Integrity and Digital Signatures

In the previous section, we saw how encryption can be used to provide confidentiality to two communicating entities. In this section, we turn to the equally important cryptography topic of providing **message integrity** (also known as message authentication). Along with message integrity, we will discuss two related topics in this section: digital signatures and end-point authentication.

We define the message integrity problem using, once again, Alice and Bob. Suppose Bob receives a message (which may be encrypted or may be in plaintext) and he believes this message was sent by Alice. To authenticate this message, Bob needs to verify:

1. The message indeed originated from Alice.
2. The message was not tampered with on its way to Bob.

We'll see in Sections 8.4 through 8.7 that this problem of message integrity is a critical concern in just about all secure networking protocols.

As a specific example, consider a computer network using a link-state routing algorithm (such as OSPF) for determining routes between each pair of routers in the network (see Chapter 5). In a link-state algorithm, each router needs to broadcast a link-state message to all other routers in the network. A router's link-state message includes a list of its directly connected neighbors and the direct costs to these neighbors. Once a router receives link-state messages from all of the other routers, it can create a complete map of the network, run its least-cost routing algorithm, and configure its forwarding table. One relatively easy attack on the routing algorithm is for Trudy to distribute bogus link-state messages with incorrect link-state information. Thus, the need for message integrity—when router B receives a link-state message from router A, router B should verify that router A actually created the message and, further, that no one tampered with the message in transit.

In this section, we describe a popular message integrity technique that is used by many secure networking protocols. But before doing so, we need to cover another important topic in cryptography—cryptographic hash functions.

### 8.3.1 Cryptographic Hash Functions

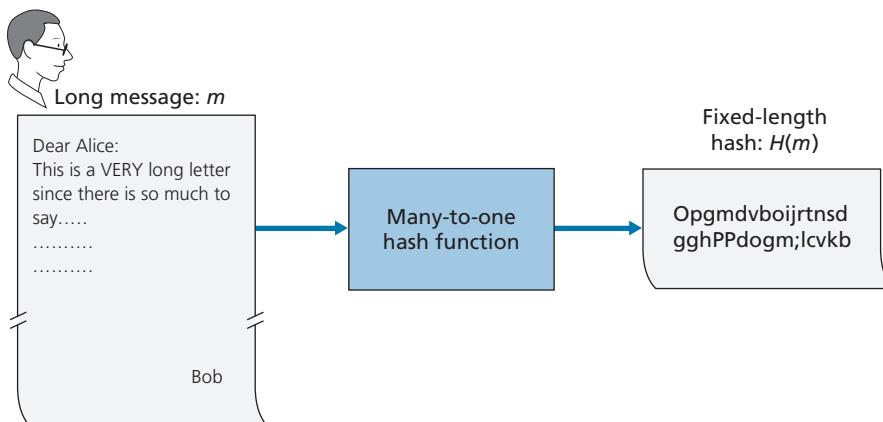
As shown in Figure 8.7, a hash function takes an input,  $m$ , and computes a fixed-size string  $H(m)$  known as a hash. The Internet checksum (Chapter 3) and CRCs (Chapter 6) meet this definition. A **cryptographic hash function** is required to have the following additional property:

- It is computationally infeasible to find any two different messages  $x$  and  $y$  such that  $H(x) = H(y)$ .

Informally, this property means that it is computationally infeasible for an intruder to substitute one message for another message that is protected by the hash function. That is, if  $(m, H(m))$  are the message and the hash of the message created by the sender, then an intruder cannot forge the contents of another message,  $y$ , that has the same hash value as the original message.

Let's convince ourselves that a simple checksum, such as the Internet checksum, would make a poor cryptographic hash function. Rather than performing 1s complement arithmetic (as in the Internet checksum), let us compute a checksum by treating each character as a byte and adding the bytes together using 4-byte chunks at a time. Suppose Bob owes Alice \$100.99 and sends an IOU to Alice consisting of the text string “IOU100.99BOB.” The ASCII representation (in hexadecimal notation) for these letters is 49, 4F, 55, 31, 30, 30, 2E, 39, 39, 42, 4F, 42.

Figure 8.8 (top) shows that the 4-byte checksum for this message is B2 C1 D2 AC. A slightly different message (and a much more costly one for Bob)



**Figure 8.7** ♦ Hash functions

| ASCII   |                |    |          |
|---------|----------------|----|----------|
| Message | Representation |    |          |
| I O U 1 | 49             | 4F | 55 31    |
| 0 0 . 9 | 30             | 30 | 2E 39    |
| 9 B O B | 39             | 42 | 4F 42    |
|         | B2             | C1 | D2 AC    |
|         |                |    | Checksum |

| ASCII   |                |    |          |
|---------|----------------|----|----------|
| Message | Representation |    |          |
| I O U 9 | 49             | 4F | 55 39    |
| 0 0 . 1 | 30             | 30 | 2E 31    |
| 9 B O B | 39             | 42 | 4F 42    |
|         | B2             | C1 | D2 AC    |
|         |                |    | Checksum |

**Figure 8.8** ♦ Initial message and fraudulent message have the same checksum!

is shown in the bottom half of Figure 8.8. The messages “IOU100.99BOB” and “IOU900.19BOB” have the *same* checksum. Thus, this simple checksum algorithm violates the requirement above. Given the original data, it is simple to find another set of data with the same checksum. Clearly, for security purposes, we are going to need a more powerful hash function than a checksum.

The MD5 hash algorithm of Ron Rivest [RFC 1321] is in wide use today. It computes a 128-bit hash in a four-step process consisting of a padding step (adding a one followed by enough zeros so that the length of the message satisfies certain conditions), an append step (appending a 64-bit representation of the message length before padding), an initialization of an accumulator, and a final looping step in which the message’s 16-word blocks are processed (mangled) in four rounds. For a description of MD5 (including a C source code implementation) see [RFC 1321].

The second major hash algorithm in use today is the Secure Hash Algorithm (SHA-1) [FIPS 1995]. This algorithm is based on principles similar to those used in the design of MD4 [RFC 1320], the predecessor to MD5. SHA-1, a US federal standard, is required for use whenever a cryptographic hash algorithm is needed for federal applications. It produces a 160-bit message digest. The longer output length makes SHA-1 more secure.

### 8.3.2 Message Authentication Code

Let’s now return to the problem of message integrity. Now that we understand hash functions, let’s take a first stab at how we might perform message integrity:

1. Alice creates message  $m$  and calculates the hash  $H(m)$  (for example, with SHA-1).
2. Alice then appends  $H(m)$  to the message  $m$ , creating an extended message  $(m, H(m))$ , and sends the extended message to Bob.
3. Bob receives an extended message  $(m, h)$  and calculates  $H(m)$ . If  $H(m) = h$ , Bob concludes that everything is fine.

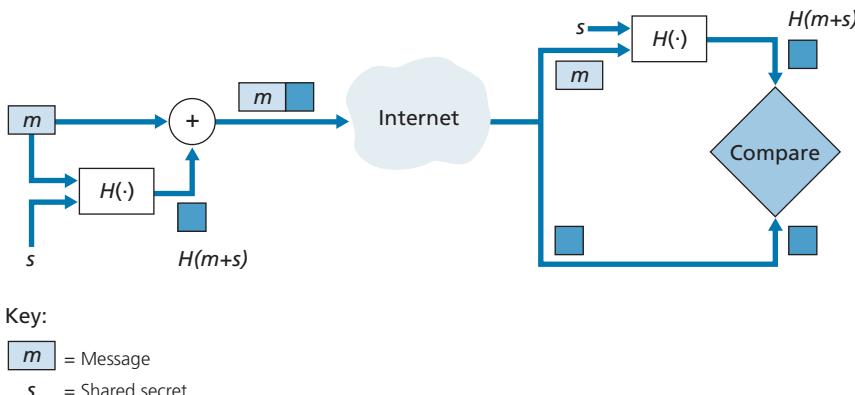
This approach is obviously flawed. Trudy can create a bogus message  $m'$  in which she says she is Alice, calculate  $H(m')$ , and send Bob  $(m', H(m'))$ . When Bob receives the message, everything checks out in step 3, so Bob doesn't suspect any funny business.

To perform message integrity, in addition to using cryptographic hash functions, Alice and Bob will need a shared secret  $s$ . This shared secret, which is nothing more than a string of bits, is called the **authentication key**. Using this shared secret, message integrity can be performed as follows:

1. Alice creates message  $m$ , concatenates  $s$  with  $m$  to create  $m + s$ , and calculates the hash  $H(m + s)$  (for example, with SHA-1).  $H(m + s)$  is called the **message authentication code (MAC)**.
2. Alice then appends the MAC to the message  $m$ , creating an extended message  $(m, H(m + s))$ , and sends the extended message to Bob.
3. Bob receives an extended message  $(m, h)$  and knowing  $s$ , calculates the MAC  $H(m + s)$ . If  $H(m + s) = h$ , Bob concludes that everything is fine.

A summary of the procedure is shown in Figure 8.9. Readers should note that the MAC here (standing for “message authentication code”) is not the same MAC used in link-layer protocols (standing for “medium access control”)!

One nice feature of a MAC is that it does not require an encryption algorithm. Indeed, in many applications, including the link-state routing algorithm described earlier, communicating entities are only concerned with message integrity and are



**Figure 8.9** ♦ Message authentication code (MAC)

not concerned with message confidentiality. Using a MAC, the entities can authenticate the messages they send to each other without having to integrate complex encryption algorithms into the integrity process.

As you might expect, a number of different standards for MACs have been proposed over the years. The most popular standard today is **HMAC**, which can be used either with MD5 or SHA-1. HMAC actually runs data and the authentication key through the hash function twice [Kaufman 2002; RFC 2104].

There still remains an important issue. How do we distribute the shared authentication key to the communicating entities? For example, in the link-state routing algorithm, we would somehow need to distribute the secret authentication key to each of the routers in the autonomous system. (Note that the routers can all use the same authentication key.) A network administrator could actually accomplish this by physically visiting each of the routers. Or, if the network administrator is a lazy guy, and if each router has its own public key, the network administrator could distribute the authentication key to any one of the routers by encrypting it with the router's public key and then sending the encrypted key over the network to the router.

### 8.3.3 Digital Signatures

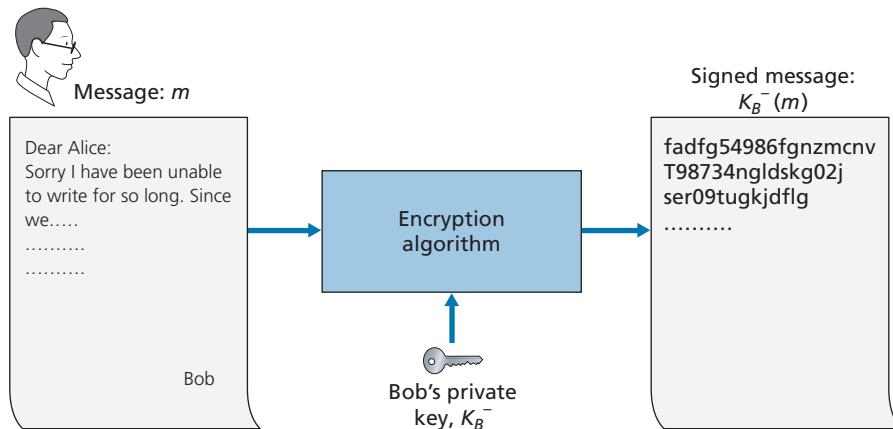
Think of the number of the times you've signed your name to a piece of paper during the last week. You sign checks, credit card receipts, legal documents, and letters. Your signature attests to the fact that you (as opposed to someone else) have acknowledged and/or agreed with the document's contents. In a digital world, one often wants to indicate the owner or creator of a document, or to signify one's agreement with a document's content. A **digital signature** is a cryptographic technique for achieving these goals in a digital world.

Just as with handwritten signatures, digital signing should be done in a way that is verifiable and nonforgeable. That is, it must be possible to prove that a document signed by an individual was indeed signed by that individual (the signature must be verifiable) and that *only* that individual could have signed the document (the signature cannot be forged).

Let's now consider how we might design a digital signature scheme. Observe that when Bob signs a message, Bob must put something on the message that is unique to him. Bob could consider attaching a MAC for the signature, where the MAC is created by appending his key (unique to him) to the message, and then taking the hash. But for Alice to verify the signature, she must also have a copy of the key, in which case the key would not be unique to Bob. Thus, MACs are not going to get the job done here.

Recall that with public-key cryptography, Bob has both a public and private key, with both of these keys being unique to Bob. Thus, public-key cryptography is an excellent candidate for providing digital signatures. Let us now examine how it is done.

Suppose that Bob wants to digitally sign a document,  $m$ . We can think of the document as a file or a message that Bob is going to sign and send. As shown in Figure 8.10, to sign this document, Bob simply uses his private key,  $K_B^-$ , to compute



**Figure 8.10** ♦ Creating a digital signature for a document

$K_B^-(m)$ . At first, it might seem odd that Bob is using his private key (which, as we saw in Section 8.2, was used to decrypt a message that had been encrypted with his public key) to sign a document. But recall that encryption and decryption are nothing more than mathematical operations (exponentiation to the power of  $e$  or  $d$  in RSA; see Section 8.2) and recall that Bob’s goal is not to scramble or obscure the contents of the document, but rather to sign the document in a manner that is verifiable and nonforgeable. Bob’s digital signature of the document is  $K_B^-(m)$ .

Does the digital signature  $K_B^-(m)$  meet our requirements of being verifiable and nonforgeable? Suppose Alice has  $m$  and  $K_B^-(m)$ . She wants to prove in court (being litigious) that Bob had indeed signed the document and was the only person who could have possibly signed the document. Alice takes Bob’s public key,  $K_B^+$ , and applies it to the digital signature,  $K_B^-(m)$ , associated with the document,  $m$ . That is, she computes  $K_B^+(K_B^-(m))$ , and voilà, with a dramatic flurry, she produces  $m$ , which exactly matches the original document! Alice then argues that only Bob could have signed the document, for the following reasons:

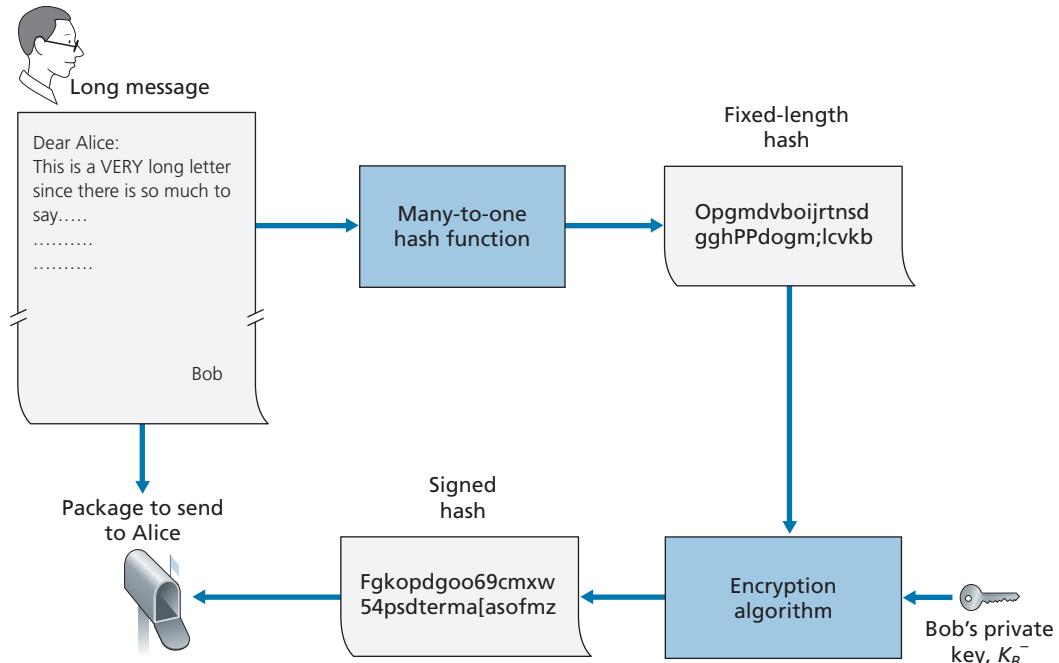
- Whoever signed the message must have used the private key,  $K_B^-$ , in computing the signature  $K_B^-(m)$ , such that  $K_B^+(K_B^-(m)) = m$ .
- The only person who could have known the private key,  $K_B^-$ , is Bob. Recall from our discussion of RSA in Section 8.2 that knowing the public key,  $K_B^+$ , is of no help in learning the private key,  $K_B^-$ . Therefore, the only person who could know  $K_B^-$  is the person who generated the pair of keys,  $(K_B^+, K_B^-)$ , in the first place, Bob. (Note that this assumes, though, that Bob has not given  $K_B^-$  to anyone, nor has anyone stolen  $K_B^-$  from Bob.)

It is also important to note that if the original document,  $m$ , is ever modified to some alternate form,  $m'$ , the signature that Bob created for  $m$  will not be valid for  $m'$ ,

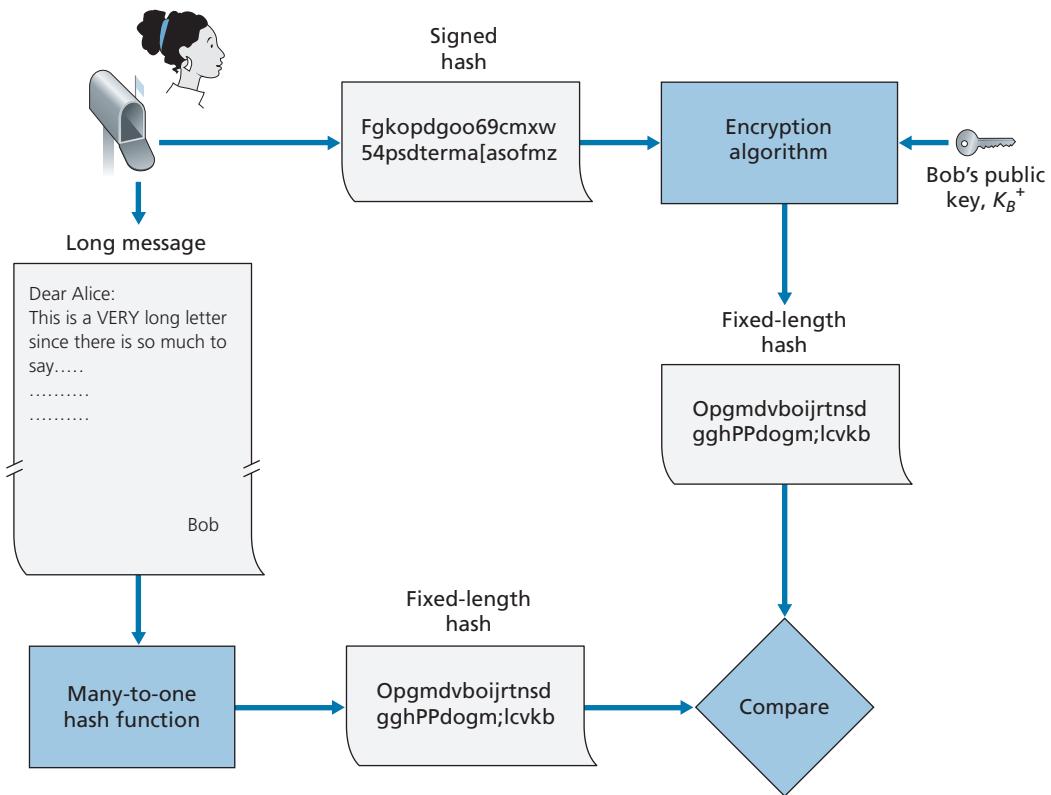
since  $K_B^+(K_B^-(m))$  does not equal  $m'$ . Thus, we see that digital signatures also provide message integrity, allowing the receiver to verify that the message was unaltered as well as the source of the message.

One concern with signing data by encryption is that encryption and decryption are computationally expensive. Given the overheads of encryption and decryption, signing data via complete encryption/decryption can be overkill. A more efficient approach is to introduce hash functions into the digital signature. Recall from Section 8.3.2 that a hash algorithm takes a message,  $m$ , of arbitrary length and computes a fixed-length “fingerprint” of the message, denoted by  $H(m)$ . Using a hash function, Bob signs the hash of a message rather than the message itself, that is, Bob calculates  $K_B^-(H(m))$ . Since  $H(m)$  is generally much smaller than the original message  $m$ , the computational effort required to create the digital signature is substantially reduced.

In the context of Bob sending a message to Alice, Figure 8.11 provides a summary of the operational procedure of creating a digital signature. Bob puts his original long message through a hash function. He then digitally signs the resulting hash with his private key. The original message (in cleartext) along with the digitally signed message digest (henceforth referred to as the digital signature) is then sent to Alice. Figure 8.12 provides a summary of the operational procedure of the signature. Alice applies the sender’s public key to the message to obtain a hash result. Alice also



**Figure 8.11** ♦ Sending a digitally signed message



**Figure 8.12** ♦ Verifying a signed message

applies the hash function to the cleartext message to obtain a second hash result. If the two hashes match, then Alice can be sure about the integrity and author of the message.

Before moving on, let's briefly compare digital signatures with MACs, since they have parallels, but also have important subtle differences. Both digital signatures and MACs start with a message (or a document). To create a MAC out of the message, we append an authentication key to the message, and then take the hash of the result. Note that neither public key nor symmetric key encryption is involved in creating the MAC. To create a digital signature, we first take the hash of the message and then encrypt the message with our private key (using public key cryptography). Thus, a digital signature is a “heavier” technique, since it requires an underlying Public Key Infrastructure (PKI) with certification authorities as described below. We'll see in Section 8.4 that PGP—a popular secure e-mail system—uses digital signatures for message integrity. We've seen already that OSPF uses MACs for message integrity. We'll see in Sections 8.5 and 8.6 that MACs are also used for popular transport-layer and network-layer security protocols.

## Public Key Certification

An important application of digital signatures is **public key certification**, that is, certifying that a public key belongs to a specific entity. Public key certification is used in many popular secure networking protocols, including IPsec and TLS.

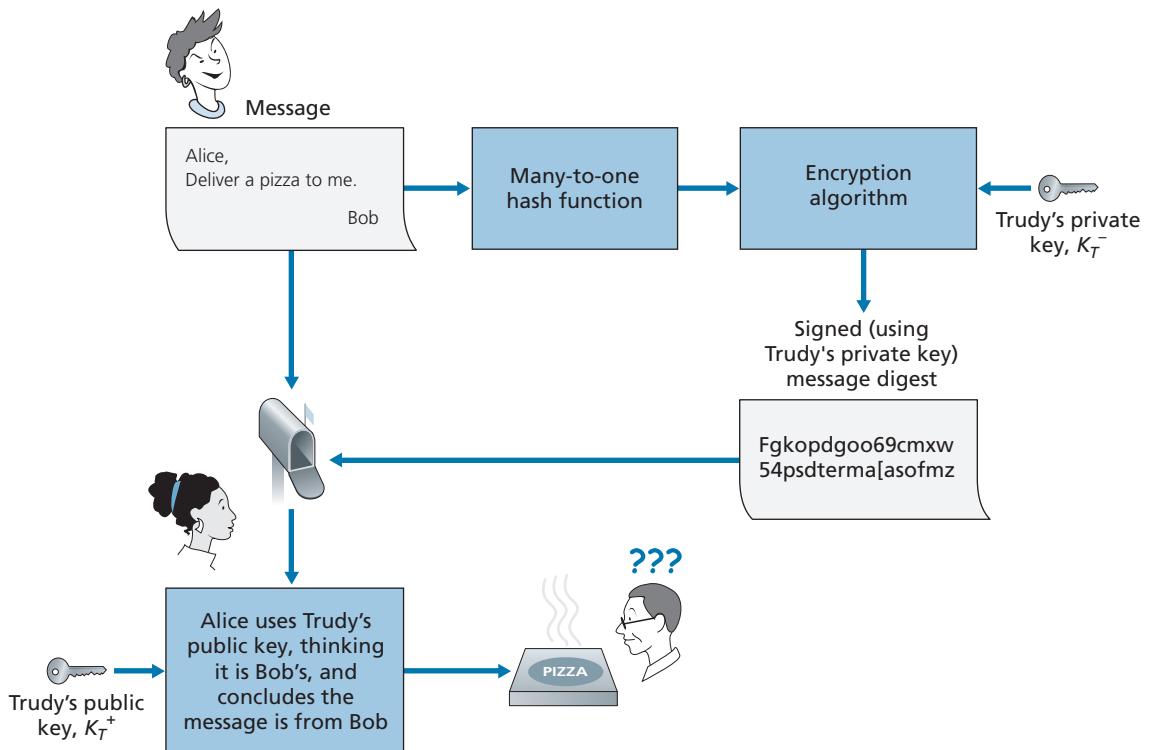
To gain insight into this problem, let's consider an Internet-commerce version of the classic “pizza prank.” Alice is in the pizza delivery business and accepts orders over the Internet. Bob, a pizza lover, sends Alice a plaintext message that includes his home address and the type of pizza he wants. In this message, Bob also includes a digital signature (that is, a signed hash of the original plaintext message) to prove to Alice that he is the true source of the message. To verify the signature, Alice obtains Bob's public key (perhaps from a public key server or from the e-mail message) and checks the digital signature. In this manner she makes sure that Bob, rather than some adolescent prankster, placed the order.

This all sounds fine until clever Trudy comes along. As shown in Figure 8.13, Trudy is indulging in a prank. She sends a message to Alice in which she says she is Bob, gives Bob's home address, and orders a pizza. In this message she also includes her (Trudy's) public key, although Alice naturally assumes it is Bob's public key. Trudy also attaches a digital signature, which was created with her own (Trudy's) private key. After receiving the message, Alice applies Trudy's public key (thinking that it is Bob's) to the digital signature and concludes that the plaintext message was indeed created by Bob. Bob will be very surprised when the delivery person brings a pizza with pepperoni and anchovies to his home!

We see from this example that for public key cryptography to be useful, you need to be able to verify that you have the actual public key of the entity (person, router, browser, and so on) with whom you want to communicate. For example, when Alice wants to communicate with Bob using public key cryptography, she needs to verify that the public key that is supposed to be Bob's is indeed Bob's.

Binding a public key to a particular entity is typically done by a **Certification Authority (CA)**, whose job is to validate identities and issue certificates. A CA has the following roles:

1. A CA verifies that an entity (a person, a router, and so on) is who it says it is. There are no mandated procedures for how certification is done. When dealing with a CA, one must trust the CA to have performed a suitably rigorous identity verification. For example, if Trudy were able to walk into the Fly-by-Night CA and simply announce “I am Alice” and receive certificates associated with the identity of Alice, then one shouldn't put much faith in public keys certified by the Fly-by-Night CA. On the other hand, one might (or might not!) be more willing to trust a CA that is part of a federal or state program. You can trust the identity associated with a public key only to the extent to which you can trust a CA and its identity verification techniques. What a tangled web of trust we spin!
2. Once the CA verifies the identity of the entity, the CA creates a **certificate** that binds the public key of the entity to the identity. The certificate contains

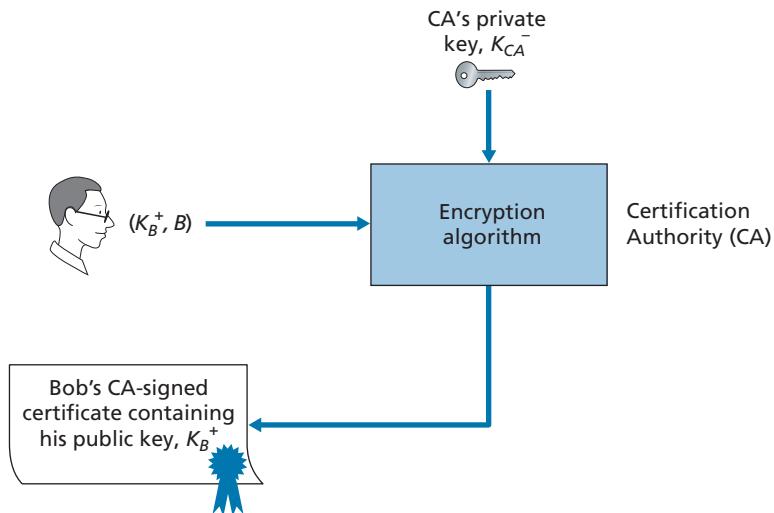


**Figure 8.13** ♦ Trudy masquerades as Bob using public key cryptography

the public key and globally unique identifying information about the owner of the public key (for example, a human name or an IP address). The certificate is digitally signed by the CA. These steps are shown in Figure 8.14.

Let us now see how certificates can be used to combat pizza-ordering pranksters, like Trudy, and other undesirables. When Bob places his order he also sends his CA-signed certificate. Alice uses the CA's public key to check the validity of Bob's certificate and extract Bob's public key.

Both the International Telecommunication Union (ITU) and the IETF have developed standards for CAs. ITU X.509 [ITU 2005a] specifies an authentication service as well as a specific syntax for certificates. [RFC 1422] describes CA-based key management for use with secure Internet e-mail. It is compatible with X.509 but goes beyond X.509 by establishing procedures and conventions for a key management architecture. Table 8.4 describes some of the important fields in a certificate.



**Figure 8.14** ♦ Bob has his public key certified by the CA

| Field Name         | Description                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------|
| Version            | Version number of X.509 specification                                                                                       |
| Serial number      | CA-issued unique identifier for a certificate                                                                               |
| Signature          | Specifies the algorithm used by CA to sign this certificate                                                                 |
| Issuer name        | Identity of CA issuing this certificate, in distinguished name (DN) [RFC 4514] format                                       |
| Validity period    | Start and end of period of validity for certificate                                                                         |
| Subject name       | Identity of entity whose public key is associated with this certificate, in DN format                                       |
| Subject public key | The subject's public key as well indication of the public key algorithm (and algorithm parameters) to be used with this key |

**Table 8.4** ♦ Selected fields in an X.509 and RFC 1422 public key

## 8.4 End-Point Authentication

**End-point authentication** is the process of one entity proving its identity to another entity over a computer network, for example, a user proving its identity to an e-mail server. As humans, we authenticate each other in many ways: We recognize each other's faces when we meet, we recognize each other's voices on the telephone, we are authenticated by the customs official who checks us against the picture on our passport.

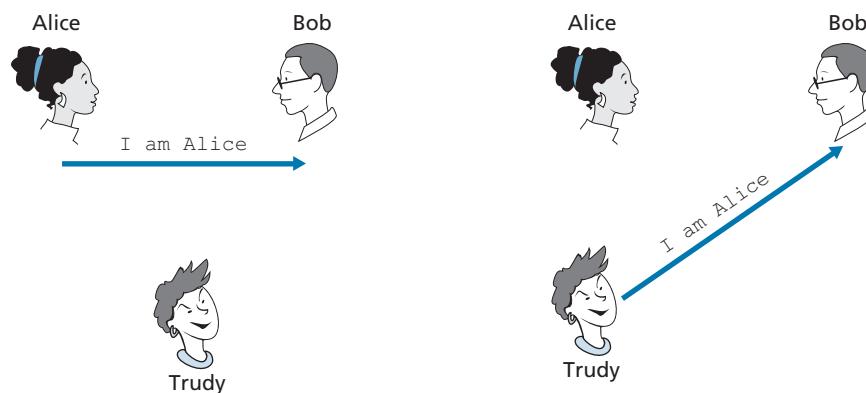
In this section, we consider how one party can authenticate another party when the two are communicating over a network. We focus here on authenticating a “live” party, at the point in time when communication is actually occurring. A concrete example is a user authenticating him or herself to an e-mail server. This is a subtly different problem from proving that a message received at some point in the past did indeed come from that claimed sender, as studied in Section 8.3.

When performing authentication over the network, the communicating parties cannot rely on biometric information, such as a visual appearance or a voiceprint. Indeed, we will see in our later case studies that it is often network elements such as routers and client/server processes that must authenticate each other. Here, authentication must be done solely on the basis of messages and data exchanged as part of an **authentication protocol**. Typically, an authentication protocol would run *before* the two communicating parties run some other protocol (for example, a reliable data transfer protocol, a routing information exchange protocol, or an e-mail protocol). The authentication protocol first establishes the identities of the parties to each other’s satisfaction; only after authentication do the parties get down to the work at hand.

As in the case of our development of a reliable data transfer (rdt) protocol in Chapter 3, we will find it instructive here to develop various versions of an authentication protocol, which we will call **ap** (authentication protocol), and poke holes in each version as we proceed. (If you enjoy this stepwise evolution of a design, you might also enjoy [Bryant 1988], which recounts a fictitious narrative between designers of an open-network authentication system, and their discovery of the many subtle issues involved.)

Let’s assume that Alice needs to authenticate herself to Bob.

Perhaps the simplest authentication protocol we can imagine is one where Alice simply sends a message to Bob saying she is Alice. This protocol is shown in Figure 8.15. The flaw here is obvious—there is no way for Bob actually to know that the person sending the message “I am Alice” is indeed Alice. For example, Trudy (the intruder) could just as well send such a message.



**Figure 8.15** ♦ Protocol *ap 1.0* and a failure scenario

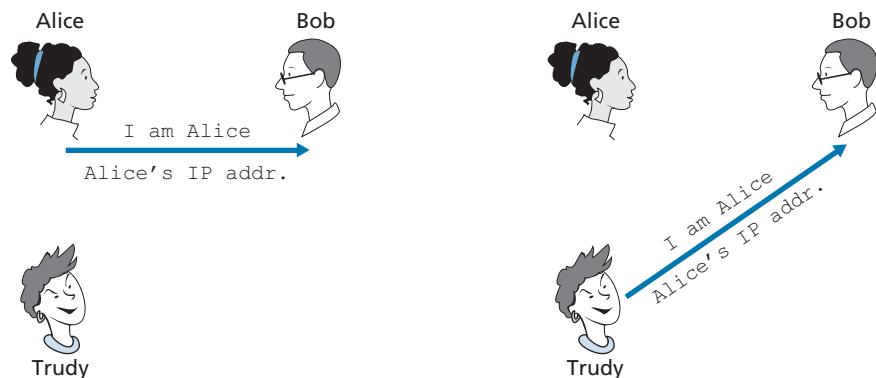
### Authentication Protocol *ap2.0*

If Alice has a well-known network address (e.g., an IP address) from which she always communicates, Bob could attempt to authenticate Alice by verifying that the source address on the IP datagram carrying the authentication message matches Alice's well-known address. In this case, Alice would be authenticated. This might stop a very network-naive intruder from impersonating Alice, but it wouldn't stop the determined student studying this book, or many others!

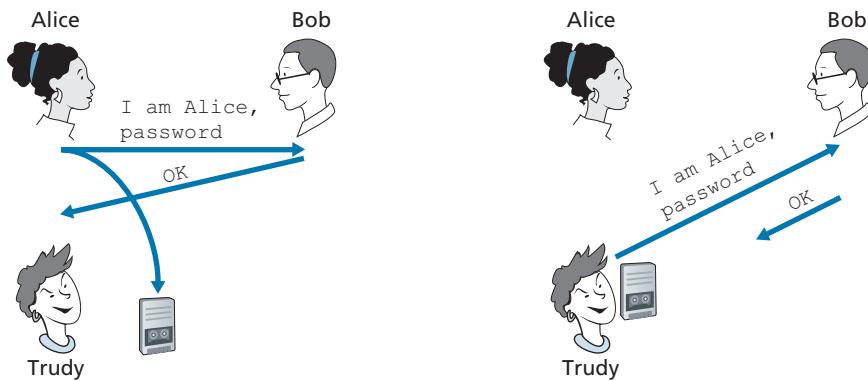
From our study of the network and data link layers, we know that it is not that hard (for example, if one had access to the operating system code and could build one's own operating system kernel, as is the case with Linux and several other freely available operating systems) to create an IP datagram, put whatever IP source address we want (for example, Alice's well-known IP address) into the IP datagram, and send the datagram over the link-layer protocol to the first-hop router. From then on, the incorrectly source-addressed datagram would be dutifully forwarded to Bob. This approach, shown in Figure 8.16, is a form of IP spoofing. IP spoofing can be avoided if Trudy's first-hop router is configured to forward only datagrams containing Trudy's IP source address [RFC 2827]. However, this capability is not universally deployed or enforced. Bob would thus be foolish to assume that Trudy's network manager (who might be Trudy herself) had configured Trudy's first-hop router to forward only appropriately addressed datagrams.

### Authentication Protocol *ap3.0*

One classic approach to authentication is to use a secret password. The password is a shared secret between the authenticator and the person being authenticated. Gmail, Facebook, telnet, FTP, and many other services use password authentication. In protocol *ap3.0*, Alice thus sends her secret password to Bob, as shown in Figure 8.17.



**Figure 8.16** ♦ Protocol *ap2.0* and a failure scenario



Key:  
 Tape recorder

**Figure 8.17** ♦ Protocol *ap3.0* and a failure scenario

Since passwords are so widely used, we might suspect that protocol *ap3.0* is fairly secure. If so, we'd be wrong! The security flaw here is clear. If Trudy eavesdrops on Alice's communication, then she can learn Alice's password. Lest you think this is unlikely, consider the fact that when you Telnet to another machine and log in, the login password is sent unencrypted to the Telnet server. Someone connected to the Telnet client or server's LAN can possibly sniff (read and store) all packets transmitted on the LAN and thus steal the login password. In fact, this is a well-known approach for stealing passwords (see, for example, [Jimenez 1997]). Such a threat is obviously very real, so *ap3.0* clearly won't do.

### Authentication Protocol *ap3.1*

Our next idea for fixing *ap3.0* is naturally to encrypt the password. By encrypting the password, we can prevent Trudy from learning Alice's password. If we assume that Alice and Bob share a symmetric secret key,  $K_{A-B}$ , then Alice can encrypt the password and send her identification message, "I am Alice," and her encrypted password to Bob. Bob then decrypts the password and, assuming the password is correct, authenticates Alice. Bob feels comfortable in authenticating Alice since Alice not only knows the password, but also knows the shared secret key value needed to encrypt the password. Let's call this protocol *ap3.1*.

While it is true that *ap3.1* prevents Trudy from learning Alice's password, the use of cryptography here does not solve the authentication problem. Bob is subject

to a **playback attack**: Trudy need only eavesdrop on Alice’s communication, record the encrypted version of the password, and play back the encrypted version of the password to Bob to pretend that she is Alice. The use of an encrypted password in *ap3.1* doesn’t make the situation manifestly different from that of protocol *ap3.0* in Figure 8.17.

### Authentication Protocol *ap4.0*

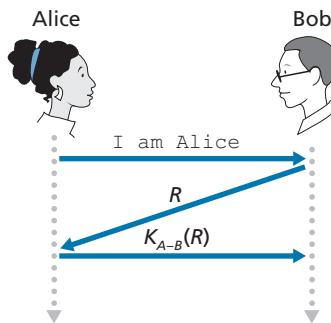
The failure scenario in Figure 8.17 resulted from the fact that Bob could not distinguish between the original authentication of Alice and the later playback of Alice’s original authentication. That is, Bob could not tell if Alice was live (that is, was currently really on the other end of the connection) or whether the messages he was receiving were a recorded playback of a previous authentication of Alice. The very (*very*) observant reader will recall that the three-way TCP handshake protocol needed to address the same problem—the server side of a TCP connection did not want to accept a connection if the received SYN segment was an old copy (retransmission) of a SYN segment from an earlier connection. How did the TCP server side solve the problem of determining whether the client was really live? It chose an initial sequence number that had not been used in a very long time, sent that number to the client, and then waited for the client to respond with an ACK segment containing that number. We can adopt the same idea here for authentication purposes.

A **nonce** is a number that a protocol will use only once in a lifetime. That is, once a protocol uses a nonce, it will never use that number again. Our *ap4.0* protocol uses a nonce as follows:

1. Alice sends the message “I am Alice” to Bob.
2. Bob chooses a nonce,  $R$ , and sends it to Alice.
3. Alice encrypts the nonce using Alice and Bob’s symmetric secret key,  $K_{A-B}$ , and sends the encrypted nonce,  $K_{A-B}(R)$ , back to Bob. As in protocol *ap3.1*, it is the fact that Alice knows  $K_{A-B}$  and uses it to encrypt a value that lets Bob know that the message he receives was generated by Alice. The nonce is used to ensure that Alice is live.
4. Bob decrypts the received message. If the decrypted nonce equals the nonce he sent Alice, then Alice is authenticated.

Protocol *ap4.0* is illustrated in Figure 8.18. By using the once-in-a-lifetime value,  $R$ , and then checking the returned value,  $K_{A-B}(R)$ , Bob can be sure that Alice is both who she says she is (since she knows the secret key value needed to encrypt  $R$ ) and live (since she has encrypted the nonce,  $R$ , that Bob just created).

The use of a nonce and symmetric key cryptography forms the basis of *ap4.0*. A natural question is whether we can use a nonce and public key cryptography (rather than symmetric key cryptography) to solve the authentication problem. This issue is explored in the problems at the end of the chapter.



**Figure 8.18** ♦ Protocol *ap4.0* and a failure scenario

## 8.5 Securing E-Mail

In previous sections, we examined fundamental issues in network security, including symmetric key and public key cryptography, end-point authentication, key distribution, message integrity, and digital signatures. We are now going to examine how these tools are being used to provide security in the Internet.

Interestingly, it is possible to provide security services in any of the top four layers of the Internet protocol stack. When security is provided for a specific application-layer protocol, the application using the protocol will enjoy one or more security services, such as confidentiality, authentication, or integrity. When security is provided for a transport-layer protocol, all applications that use that protocol enjoy the security services of the transport protocol. When security is provided at the network layer on a host-to-host basis, all transport-layer segments (and hence all application-layer data) enjoy the security services of the network layer. When security is provided on a link basis, then the data in all frames traveling over the link receive the security services of the link.

In Sections 8.5 through 8.8, we examine how security tools are being used in the application, transport, network, and link layers. Being consistent with the general structure of this book, we begin at the top of the protocol stack and discuss security at the application layer. Our approach is to use a specific application, e-mail, as a case study for application-layer security. We then move down the protocol stack. We'll examine the TLS protocol (which provides security at the transport layer), IPsec (which provides security at the network layer), and the security of the IEEE 802.11 wireless LAN protocol.

You might be wondering why security functionality is being provided at more than one layer in the Internet. Wouldn't it suffice simply to provide the security functionality at the network layer and be done with it? There are two answers to this question. First, although security at the network layer can offer "blanket coverage" by encrypting all the data in the datagrams (that is, all the transport-layer segments)

and by authenticating all the source IP addresses, it can't provide user-level security. For example, a commerce site cannot rely on IP-layer security to authenticate a customer who is purchasing goods at the commerce site. Thus, there is a need for security functionality at higher layers as well as blanket coverage at lower layers. Second, it is generally easier to deploy new Internet services, including security services, at the higher layers of the protocol stack. While waiting for security to be broadly deployed at the network layer, which is probably still many years in the future, many application developers "just do it" and introduce security functionality into their favorite applications. A classic example is Pretty Good Privacy (PGP), which provides secure e-mail (discussed later in this section). Requiring only client and server application code, PGP was one of the first security technologies to be broadly used in the Internet.

### 8.5.1 Secure E-Mail

We now use the cryptographic principles of Sections 8.2 through 8.3 to create a secure e-mail system. We create this high-level design in an incremental manner, at each step introducing new security services. When designing a secure e-mail system, let us keep in mind the racy example introduced in Section 8.1—the love affair between Alice and Bob. Imagine that Alice wants to send an e-mail message to Bob, and Trudy wants to intrude.

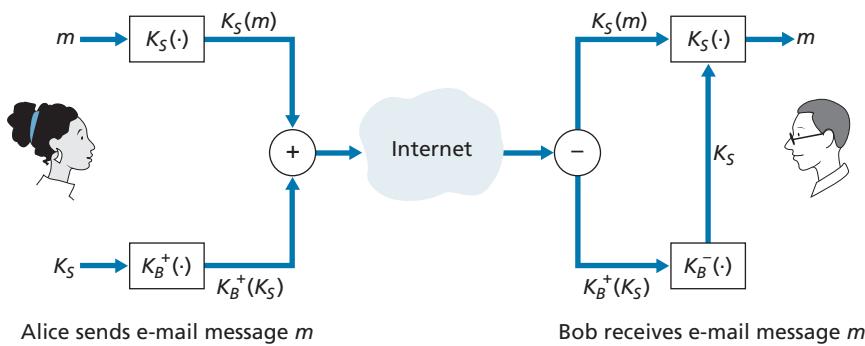
Before plowing ahead and designing a secure e-mail system for Alice and Bob, we should consider which security features would be most desirable for them. First and foremost is *confidentiality*. As discussed in Section 8.1, neither Alice nor Bob wants Trudy to read Alice's e-mail message. The second feature that Alice and Bob would most likely want to see in the secure e-mail system is *sender authentication*. In particular, when Bob receives the message "I don't love you anymore. I never want to see you again. Formerly yours, Alice," he would naturally want to be sure that the message came from Alice and not from Trudy. Another feature that the two lovers would appreciate is *message integrity*, that is, assurance that the message Alice sends is not modified while en route to Bob. Finally, the e-mail system should provide *receiver authentication*; that is, Alice wants to make sure that she is indeed sending the letter to Bob and not to someone else (for example, Trudy) who is impersonating Bob.

So let's begin by addressing the foremost concern, confidentiality. The most straightforward way to provide confidentiality is for Alice to encrypt the message with symmetric key technology (such as DES or AES) and for Bob to decrypt the message on receipt. As discussed in Section 8.2, if the symmetric key is long enough, and if only Alice and Bob have the key, then it is extremely difficult for anyone else (including Trudy) to read the message. Although this approach is straightforward, it has the fundamental difficulty that we discussed in Section 8.2—distributing a symmetric key so that only Alice and Bob have copies of it. So we naturally consider an alternative approach—public key cryptography (using, for example, RSA). In the

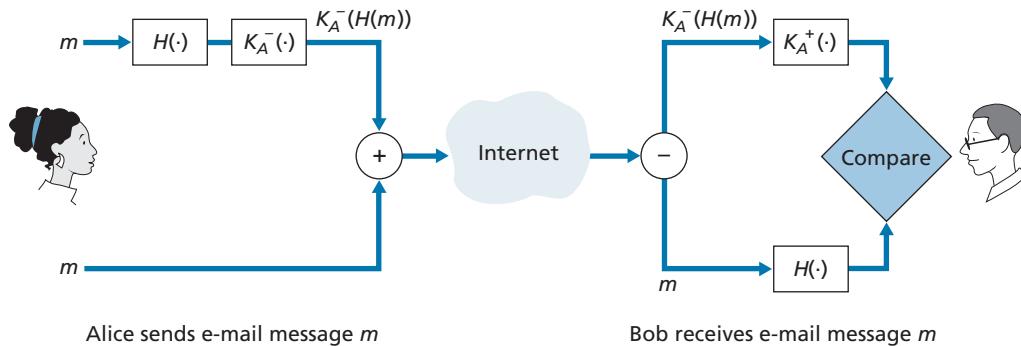
public key approach, Bob makes his public key publicly available (e.g., in a public key server or on his personal Web page). Alice encrypts her message with Bob's public key, and she sends the encrypted message to Bob's e-mail address. When Bob receives the message, he simply decrypts it with his private key. Assuming that Alice knows for sure that the public key is Bob's public key, this approach is an excellent means to provide the desired confidentiality. One problem, however, is that public key encryption is relatively inefficient, particularly for long messages.

To overcome the efficiency problem, let's make use of a session key (discussed in Section 8.2.2). In particular, Alice (1) selects a random symmetric session key,  $K_S$ , (2) encrypts her message,  $m$ , with the symmetric key, (3) encrypts the symmetric key with Bob's public key,  $K_B^+$ , (4) concatenates the encrypted message and the encrypted symmetric key to form a "package," and (5) sends the package to Bob's e-mail address. The steps are illustrated in Figure 8.19. (In this and the subsequent figures, the circled "+" represents concatenation and the circled "-" represents deconcatenation.) When Bob receives the package, he (1) uses his private key,  $K_B^-$ , to obtain the symmetric key,  $K_S$ , and (2) uses the symmetric key  $K_S$  to decrypt the message  $m$ .

Having designed a secure e-mail system that provides confidentiality, let's now design another system that provides both sender authentication and message integrity. We'll suppose, for the moment, that Alice and Bob are no longer concerned with confidentiality (they want to share their feelings with everyone!), and are concerned only about sender authentication and message integrity. To accomplish this task, we use digital signatures and message digests, as described in Section 8.3. Specifically, Alice (1) applies a hash function,  $H$  (e.g., MD5), to her message,  $m$ , to obtain a message digest, (2) signs the result of the hash function with her private key,  $K_A^-$ , to create a digital signature, (3) concatenates the original (unencrypted) message with the signature to create a package, and (4) sends the package to Bob's e-mail address. When Bob receives the package, he (1) applies Alice's public key,  $K_A^+$ , to the signed



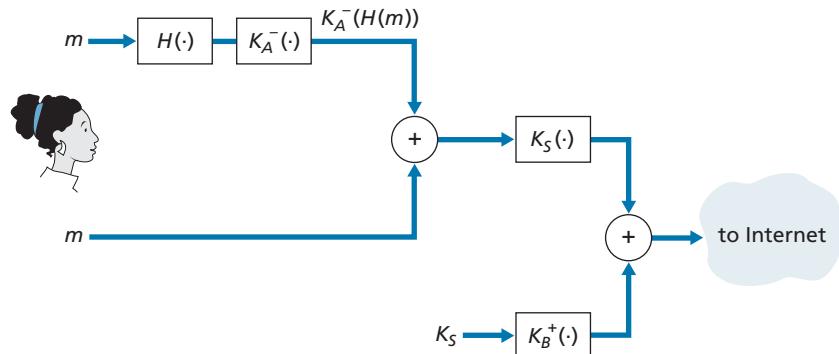
**Figure 8.19** ♦ Alice used a symmetric session key,  $K_S$ , to send a secret e-mail to Bob



**Figure 8.20** ♦ Using hash functions and digital signatures to provide sender authentication and message integrity

message digest and (2) compares the result of this operation with his own hash,  $H$ , of the message. The steps are illustrated in Figure 8.20. As discussed in Section 8.3, if the two results are the same, Bob can be pretty confident that the message came from Alice and is unaltered.

Now let's consider designing an e-mail system that provides confidentiality, sender authentication, *and* message integrity. This can be done by combining the procedures in Figures 8.19 and 8.20. Alice first creates a preliminary package, exactly as in Figure 8.20, that consists of her original message along with a digitally signed hash of the message. She then treats this preliminary package as a message in itself and sends this new message through the sender steps in Figure 8.19, creating a new package that is sent to Bob. The steps applied by Alice are shown in Figure 8.21. When Bob receives the package, he first applies his side of Figure 8.19 and then his



**Figure 8.21** ♦ Alice uses symmetric key cryptography, public key cryptography, a hash function, and a digital signature to provide secrecy, sender authentication, and message integrity

side of Figure 8.20. It should be clear that this design achieves the goal of providing confidentiality, sender authentication, and message integrity. Note that, in this scheme, Alice uses public key cryptography twice: once with her own private key and once with Bob's public key. Similarly, Bob also uses public key cryptography twice—once with his private key and once with Alice's public key.

The secure e-mail design outlined in Figure 8.21 probably provides satisfactory security for most e-mail users for most occasions. However, there is still one important issue that remains to be addressed. The design in Figure 8.21 requires Alice to obtain Bob's public key, and requires Bob to obtain Alice's public key. The distribution of these public keys is a nontrivial problem. For example, Trudy might masquerade as Bob and give Alice her own public key while saying that it is Bob's public key, enabling her to receive the message meant for Bob. As we learned in Section 8.3, a popular approach for securely distributing public keys is to *certify* the public keys using a CA.

### 8.5.2 PGP

Written by Phil Zimmermann in 1991, **Pretty Good Privacy (PGP)** is a nice example of an e-mail encryption scheme [PGP 2020]. The PGP design is, in essence, the same as the design shown in Figure 8.21. Depending on the version, the PGP software uses MD5 or SHA for calculating the message digest; CAST, triple-DES, or IDEA for symmetric key encryption; and RSA for the public key encryption.

When PGP is installed, the software creates a public key pair for the user. The public key can be posted on the user's Web site or placed in a public key server. The private key is protected by the use of a password. The password has to be entered every time the user accesses the private key. PGP gives the user the option of digitally signing the message, encrypting the message, or both digitally signing and encrypting. Figure 8.22 shows a PGP signed message. This message appears after the MIME header. The encoded data in the message is  $K_A^-(H(m))$ , that is, the digitally signed message digest. As we discussed above, in order for Bob to verify the integrity of the message, he needs to have access to Alice's public key.

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1
Bob:
Can I see you tonight?
Passionately yours, Alice
-----BEGIN PGP SIGNATURE-----
Version: PGP for Personal Privacy 5.0
Charset: noconv
yHJRhhGJGhgg/12EpJ+lo8gE4vB3mqJhFEvZP9t6n7G6m5Gw2
-----END PGP SIGNATURE-----
```

**Figure 8.22** ♦ A PGP signed message

```
-----BEGIN PGP MESSAGE-----
Version: PGP for Personal Privacy 5.0
u2R4d+/jKmn8Bc5+hgDsqAewsDfrGdszX681iKm5F6Gc4sDfcXyt
RfdS10juHgbcfDssWe7/K=1KhnMikLo0+1/BvcX4t==Ujk9PbcD4
Thdf2awQfgHbnmK1ok8iy6gThlp
-----END PGP MESSAGE
```

**Figure 8.23** ♦ A secret PGP message

Figure 8.23 shows a secret PGP message. This message also appears after the MIME header. Of course, the plaintext message is not included within the secret e-mail message. When a sender (such as Alice) wants both confidentiality and integrity, PGP contains a message like that of Figure 8.23 within the message of Figure 8.22.

PGP also provides a mechanism for public key certification, but the mechanism is quite different from the more conventional CA. PGP public keys are certified by a *web of trust*. Alice herself can certify any key/username pair when she believes the pair really belong together. In addition, PGP permits Alice to say that she trusts another user to vouch for the authenticity of more keys. Some PGP users sign each other's keys by holding key-signing parties. Users physically gather, exchange public keys, and certify each other's keys by signing them with their private keys.

## 8.6 Securing TCP Connections: TLS

In the previous section, we saw how cryptographic techniques can provide confidentiality, data integrity, and end-point authentication to a specific application, namely, e-mail. In this section, we'll drop down a layer in the protocol stack and examine how cryptography can enhance TCP with security services, including confidentiality, data integrity, and end-point authentication. This enhanced version of TCP is commonly known as **Transport Layer Security (TLS)**, which has been standardized by the IETF [RFC 4346]. An earlier and similar version of this protocol is SSL version 3.

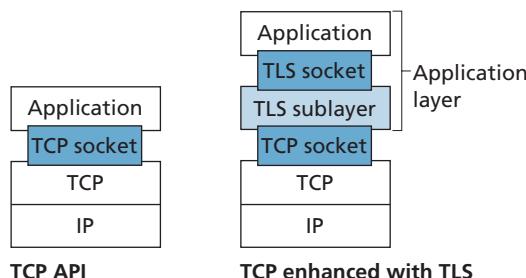
The SSL protocol was originally designed by Netscape, but the basic ideas behind securing TCP had predicated Netscape's work (for example, see Woo [Woo 1994]). Since its inception, SSL and its successor TLS have enjoyed broad deployment. TLS is supported by all popular Web browsers and Web servers, and it is used by Gmail and essentially all Internet commerce sites (including Amazon, eBay, and TaoBao). Hundreds of billions of dollars are spent over TLS every year. In fact, if you have ever purchased anything over the Internet with your credit card, the communication between your browser and the server for this purchase almost certainly went over TLS. (You can identify that TLS is being used by your browser when the URL begins with https: rather than http.)

To understand the need for TLS, let's walk through a typical Internet commerce scenario. Bob is surfing the Web and arrives at the Alice Incorporated site, which is selling perfume. The Alice Incorporated site displays a form in which Bob is supposed to enter the type of perfume and quantity desired, his address, and his payment card number. Bob enters this information, clicks on Submit, and expects to receive (via ordinary postal mail) the purchased perfumes; he also expects to receive a charge for his order in his next payment card statement. This all sounds good, but if no security measures are taken, Bob could be in for a few surprises.

- If no confidentiality (encryption) is used, an intruder could intercept Bob's order and obtain his payment card information. The intruder could then make purchases at Bob's expense.
- If no data integrity is used, an intruder could modify Bob's order, having him purchase ten times more bottles of perfume than desired.
- Finally, if no server authentication is used, a server could display Alice Incorporated's famous logo when in actuality the site maintained by Trudy, who is masquerading as Alice Incorporated. After receiving Bob's order, Trudy could take Bob's money and run. Or Trudy could carry out an identity theft by collecting Bob's name, address, and credit card number.

TLS addresses these issues by enhancing TCP with confidentiality, data integrity, server authentication, and client authentication.

TLS is often used to provide security to transactions that take place over HTTP. However, because TLS secures TCP, it can be employed by any application that runs over TCP. TLS provides a simple Application Programmer Interface (API) with sockets, which is similar and analogous to TCP's API. When an application wants to employ TLS, the application includes SSL classes/libraries. As shown in Figure 8.24, although TLS technically resides in the application layer, from the developer's perspective it is a transport protocol that provides TCP's services enhanced with security services.



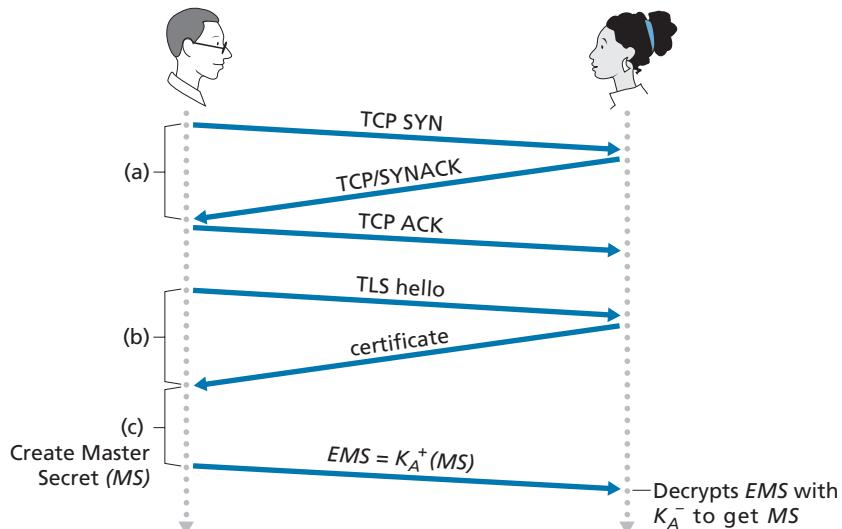
**Figure 8.24** ♦ Although TLS technically resides in the application layer, from the developer's perspective it is a transport-layer protocol

### 8.6.1 The Big Picture

We begin by describing a simplified version of TLS, one that will allow us to get a big-picture understanding of the *why* and *how* of TLS. We will refer to this simplified version of TLS as “almost-TLS.” After describing almost-TLS, in the next subsection we’ll then describe the real TLS, filling in the details. Almost-TLS (and TLS) has three phases: *handshake*, *key derivation*, and *data transfer*. We now describe these three phases for a communication session between a client (Bob) and a server (Alice), with Alice having a private/public key pair and a certificate that binds her identity to her public key.

#### Handshake

During the handshake phase, Bob needs to (a) establish a TCP connection with Alice, (b) verify that Alice is *really* Alice, and (c) send Alice a master secret key, which will be used by both Alice and Bob to generate all the symmetric keys they need for the TLS session. These three steps are shown in Figure 8.25. Note that once the TCP connection is established, Bob sends Alice a hello message. Alice then responds with her certificate, which contains her public key. As discussed in Section 8.3, because the certificate has been certified by a CA, Bob knows for sure that the public key in the certificate belongs to Alice. Bob then generates a Master Secret (MS) (which will only be used for this TLS session), encrypts the MS with Alice’s public key to create



**Figure 8.25** ♦ The almost-TLS handshake, beginning with a TCP connection

the Encrypted Master Secret (EMS), and sends the EMS to Alice. Alice decrypts the EMS with her private key to get the MS. After this phase, both Bob and Alice (and no one else) know the master secret for this TLS session.

### Key Derivation

In principle, the MS, now shared by Bob and Alice, could be used as the symmetric session key for all subsequent encryption and data integrity checking. It is, however, generally considered safer for Alice and Bob to each use different cryptographic keys, and also to use different keys for encryption and integrity checking. Thus, both Alice and Bob use the MS to generate four keys:

- $E_B$  = session encryption key for data sent from Bob to Alice
- $M_B$  = session HMAC key for data sent from Bob to Alice, where HMAC [RFC 2104] is a standardized hashed message authentication code (MAC) that we encountered in section 8.3.2
- $E_A$  = session encryption key for data sent from Alice to Bob
- $M_A$  = session HMAC key for data sent from Alice to Bob

Alice and Bob each generate the four keys from the MS. This could be done by simply slicing the MS into four keys. (But in *reality* TLS it is a little more complicated, as we'll see.) At the end of the key derivation phase, both Alice and Bob have all four keys. The two encryption keys will be used to encrypt data; the two HMAC keys will be used to verify the integrity of the data.

### Data Transfer

Now that Alice and Bob share the same four session keys ( $E_B$ ,  $M_B$ ,  $E_A$ , and  $M_A$ ), they can start to send secured data to each other over the TCP connection. Since TCP is a byte-stream protocol, a natural approach would be for TLS to encrypt application data on the fly and then pass the encrypted data on the fly to TCP. But if we were to do this, where would we put the HMAC for the integrity check? We certainly do not want to wait until the end of the TCP session to verify the integrity of all of Bob's data that was sent over the entire session! To address this issue, TLS breaks the data stream into records, appends an HMAC to each record for integrity checking, and then encrypts the record+HMAC. To create the HMAC, Bob inputs the record data along with the key  $M_B$  into a hash function, as discussed in Section 8.3. To encrypt the package record+HMAC, Bob uses his session encryption key  $E_B$ . This encrypted package is then passed to TCP for transport over the Internet.

Although this approach goes a long way, it still isn't bullet-proof when it comes to providing data integrity for the entire message stream. In particular, suppose Trudy is a woman-in-the-middle and has the ability to insert, delete, and replace segments in the stream of TCP segments sent between Alice and Bob. Trudy, for

example, could capture two segments sent by Bob, reverse the order of the segments, adjust the TCP sequence numbers (which are not encrypted), and then send the two reverse-ordered segments to Alice. Assuming that each TCP segment encapsulates exactly one record, let's now take a look at how Alice would process these segments.

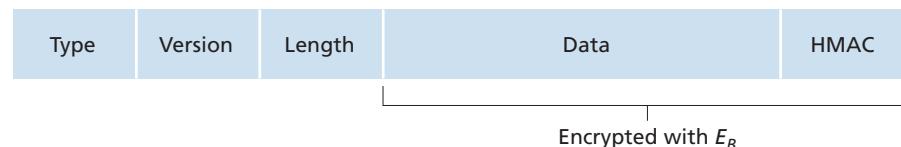
1. TCP running in Alice would think everything is fine and pass the two records to the TLS sublayer.
2. TLS in Alice would decrypt the two records.
3. TLS in Alice would use the HMAC in each record to verify the data integrity of the two records.
4. TLS would then pass the decrypted byte streams of the two records to the application layer; but the complete byte stream received by Alice would not be in the correct order due to reversal of the records!

You are encouraged to walk through similar scenarios for when Trudy removes segments or when Trudy replays segments.

The solution to this problem, as you probably guessed, is to use sequence numbers. TLS does this as follows. Bob maintains a sequence number counter, which begins at zero and is incremented for each TLS record he sends. Bob doesn't actually include a sequence number in the record itself, but when he calculates the HMAC, he includes the sequence number in the HMAC calculation. Thus, the HMAC is now a hash of the data plus the HMAC key  $M_B$  plus the current sequence number. Alice tracks Bob's sequence numbers, allowing her to verify the data integrity of a record by including the appropriate sequence number in the HMAC calculation. This use of TLS sequence numbers prevents Trudy from carrying out a woman-in-the-middle attack, such as reordering or replaying segments. (Why?)

### TLS Record

The TLS record (as well as the almost-TLS record) is shown in Figure 8.26. The record consists of a type field, version field, length field, data field, and HMAC field. Note that the first three fields are not encrypted. The type field indicates whether the record is a handshake message or a message that contains application data. It is also used to close the TLS connection, as discussed below. TLS at the receiving end uses the length field to extract the TLS records out of the incoming TCP byte stream. The version field is self-explanatory.



**Figure 8.26** ♦ Record format for TLS

## 8.6.2 A More Complete Picture

The previous subsection covered the almost-TLS protocol; it served to give us a basic understanding of the why and how of TLS. Now that we have a basic understanding, we can dig a little deeper and examine the essentials of the actual TLS protocol. In parallel to reading this description of the TLS protocol, you are encouraged to complete the Wireshark TLS lab, available at the textbook's Web site.

### TLS Handshake

SSL does not mandate that Alice and Bob use a specific symmetric key algorithm or a specific public-key algorithm. Instead, TLS allows Alice and Bob to agree on the cryptographic algorithms at the beginning of the TLS session, during the handshake phase. Additionally, during the handshake phase, Alice and Bob send nonces to each other, which are used in the creation of the session keys ( $E_B$ ,  $M_B$ ,  $E_A$ , and  $M_A$ ). The steps of the real TLS handshake are as follows:

1. The client sends a list of cryptographic algorithms it supports, along with a client nonce.
2. From the list, the server chooses a symmetric algorithm (for example, AES) and a public key algorithm (for example, RSA with a specific key length), and HMAC algorithm (MD5 or SHA-1) along with the HMAC keys. It sends back to the client its choices, as well as a certificate and a server nonce.
3. The client verifies the certificate, extracts the server's public key, generates a Pre-Master Secret (PMS), encrypts the PMS with the server's public key, and sends the encrypted PMS to the server.
4. Using the same key derivation function (as specified by the TLS standard), the client and server independently compute the Master Secret (MS) from the PMS and nonces. The MS is then sliced up to generate the two encryption and two HMAC keys. Furthermore, when the chosen symmetric cipher employs CBC (such as 3DES or AES), then two Initialization Vectors (IVs)—one for each side of the connection—are also obtained from the MS. Henceforth, all messages sent between client and server are encrypted and authenticated (with the HMAC).
5. The client sends the HMAC of all the handshake messages.
6. The server sends the HMAC of all the handshake messages.

The last two steps protect the handshake from tampering. To see this, observe that in step 1, the client typically offers a list of algorithms—some strong, some weak. This list of algorithms is sent in cleartext, since the encryption algorithms and keys have not yet been agreed upon. Trudy, as a woman-in-the-middle, could delete the stronger algorithms from the list, forcing the client to select a weak algorithm. To

prevent such a tampering attack, in step 5, the client sends the HMAC of the concatenation of all the handshake messages it sent and received. The server can compare this HMAC with the HMAC of the handshake messages it received and sent. If there is an inconsistency, the server can terminate the connection. Similarly, the server sends the HMAC of the handshake messages it has seen, allowing the client to check for inconsistencies.

You may be wondering why there are nonces in steps 1 and 2. Don't sequence numbers suffice for preventing the segment replay attack? The answer is yes, but they don't alone prevent the "connection replay attack." Consider the following connection replay attack. Suppose Trudy sniffs all messages between Alice and Bob. The next day, Trudy masquerades as Bob and sends to Alice exactly the same sequence of messages that Bob sent to Alice on the previous day. If Alice doesn't use nonces, she will respond with exactly the same sequence of messages she sent the previous day. Alice will not suspect any funny business, as each message she receives will pass the integrity check. If Alice is an e-commerce server, she will think that Bob is placing a second order (for exactly the same thing). On the other hand, by including a nonce in the protocol, Alice will send different nonces for each TCP session, causing the encryption keys to be different on the two days. Therefore, when Alice receives played-back TLS records from Trudy, the records will fail the integrity checks, and the bogus e-commerce transaction will not succeed. In summary, in TLS, nonces are used to defend against the "connection replay attack" and sequence numbers are used to defend against replaying individual packets during an ongoing session.

### Connection Closure

At some point, either Bob or Alice will want to end the TLS session. One approach would be to let Bob end the TLS session by simply terminating the underlying TCP connection—that is, by having Bob send a TCP FIN segment to Alice. But such a naive design sets the stage for the *truncation attack* whereby Trudy once again gets in the middle of an ongoing TLS session and ends the session early with a TCP FIN. If Trudy were to do this, Alice would think she received all of Bob's data when actually she only received a portion of it. The solution to this problem is to indicate in the type field whether the record serves to terminate the TLS session. (Although the TLS type is sent in the clear, it is authenticated at the receiver using the record's HMAC.) By including such a field, if Alice were to receive a TCP FIN before receiving a closure TLS record, she would know that something funny was going on.

This completes our introduction to TLS. We've seen that it uses many of the cryptography principles discussed in Sections 8.2 and 8.3. Readers who want to explore TLS on yet a deeper level can read Rescorla's highly readable book on SSL/TLS [Rescorla 2001].

## 8.7 Network-Layer Security: IPsec and Virtual Private Networks

The IP security protocol, more commonly known as **IPsec**, provides security at the network layer. IPsec secures IP datagrams between any two network-layer entities, including hosts and routers. As we will soon describe, many institutions (corporations, government branches, non-profit organizations, and so on) use IPsec to create **virtual private networks (VPNs)** that run over the public Internet.

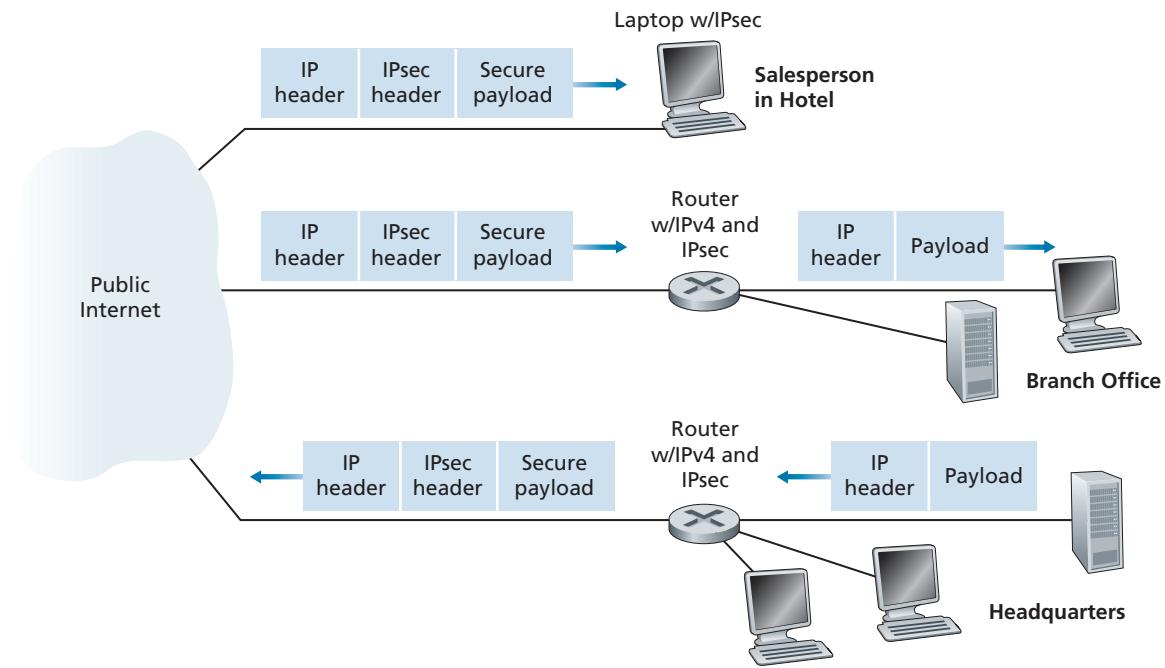
Before getting into the specifics of IPsec, let's step back and consider what it means to provide confidentiality at the network layer. With network-layer confidentiality between a pair of network entities (for example, between two routers, between two hosts, or between a router and a host), the sending entity encrypts the payloads of all the datagrams it sends to the receiving entity. The encrypted payload could be a TCP segment, a UDP segment, an ICMP message, and so on. If such a network-layer service were in place, all data sent from one entity to the other—including e-mail, Web pages, TCP handshake messages, and management messages (such as ICMP and SNMP)—would be hidden from any third party that might be sniffing the network. For this reason, network-layer security is said to provide “blanket coverage.”

In addition to confidentiality, a network-layer security protocol could potentially provide other security services. For example, it could provide source authentication, so that the receiving entity can verify the source of the secured datagram. A network-layer security protocol could provide data integrity, so that the receiving entity can check for any tampering of the datagram that may have occurred while the datagram was in transit. A network-layer security service could also provide replay-attack prevention, meaning that Bob could detect any duplicate datagrams that an attacker might insert. We will soon see that IPsec indeed provides mechanisms for all these security services, that is, for confidentiality, source authentication, data integrity, and replay-attack prevention.

### 8.7.1 IPsec and Virtual Private Networks (VPNs)

An institution that extends over multiple geographical regions often desires its own IP network, so that its hosts and servers can send data to each other in a secure and confidential manner. To achieve this goal, the institution could actually deploy a stand-alone physical network—including routers, links, and a DNS infrastructure—that is completely separate from the public Internet. Such a disjoint network, dedicated to a particular institution, is called a **private network**. Not surprisingly, a private network can be very costly, as the institution needs to purchase, install, and maintain its own physical network infrastructure.

Instead of deploying and maintaining a private network, many institutions today create VPNs over the existing public Internet. With a VPN, the institution's inter-office traffic is sent over the public Internet rather than over a physically



**Figure 8.27** ♦ Virtual private network (VPN)

independent network. But to provide confidentiality, the inter-office traffic is encrypted before it enters the public Internet. A simple example of a VPN is shown in Figure 8.27. Here the institution consists of a headquarters, a branch office, and traveling salespersons that typically access the Internet from their hotel rooms. (There is only one salesperson shown in the figure.) In this VPN, whenever two hosts within headquarters send IP datagrams to each other or whenever two hosts within the branch office want to communicate, they use good-old vanilla IPv4 (that is, without IPsec services). However, when two of the institution's hosts communicate over a path that traverses the public Internet, the traffic is encrypted before it enters the Internet.

To get a feel for how a VPN works, let's walk through a simple example in the context of Figure 8.27. When a host in headquarters sends an IP datagram to a salesperson in a hotel, the gateway router in headquarters converts the vanilla IPv4 datagram into an IPsec datagram and then forwards this IPsec datagram into the Internet. This IPsec datagram actually has a traditional IPv4 header, so that the routers in the public Internet process the datagram as if it were an ordinary IPv4 datagram—to them, the datagram is a perfectly ordinary datagram. But, as shown Figure 8.27, the payload of the IPsec datagram includes an IPsec header, which is used for IPsec processing; furthermore, the payload of the IPsec datagram is encrypted. When the

IPsec datagram arrives at the salesperson's laptop, the OS in the laptop decrypts the payload (and provides other security services, such as verifying data integrity) and passes the unencrypted payload to the upper-layer protocol (for example, to TCP or UDP).

We have just given a high-level overview of how an institution can employ IPsec to create a VPN. To see the forest through the trees, we have brushed aside many important details. Let's now take a closer look.

### 8.7.2 The AH and ESP Protocols

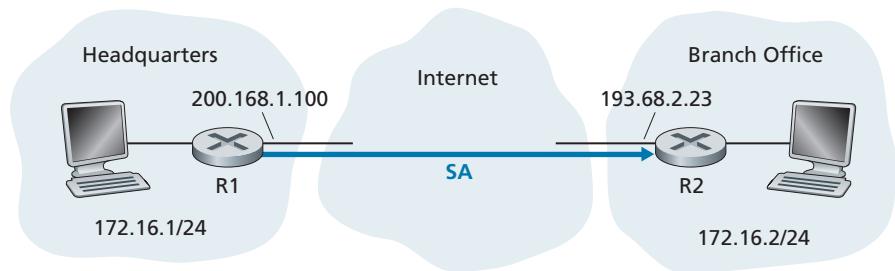
IPsec is a rather complex animal—it is defined in more than a dozen RFCs. Two important RFCs are RFC 4301, which describes the overall IP security architecture, and RFC 6071, which provides an overview of the IPsec protocol suite. Our goal in this textbook, as usual, is not simply to re-hash the dry and arcane RFCs, but instead take a more operational and pedagogic approach to describing the protocols.

In the IPsec protocol suite, there are two principal protocols: the **Authentication Header (AH)** protocol and the **Encapsulation Security Payload (ESP)** protocol. When a source IPsec entity (typically a host or a router) sends secure datagrams to a destination entity (also a host or a router), it does so with either the AH protocol or the ESP protocol. The AH protocol provides source authentication and data integrity but *does not* provide confidentiality. The ESP protocol provides source authentication, data integrity, *and* confidentiality. Because confidentiality is often critical for VPNs and other IPsec applications, the ESP protocol is much more widely used than the AH protocol. In order to de-mystify IPsec and avoid much of its complication, we will henceforth focus exclusively on the ESP protocol. Readers wanting to learn also about the AH protocol are encouraged to explore the RFCs and other online resources.

### 8.7.3 Security Associations

IPsec datagrams are sent between pairs of network entities, such as between two hosts, between two routers, or between a host and router. Before sending IPsec datagrams from source entity to destination entity, the source and destination entities create a network-layer logical connection. This logical connection is called a **security association (SA)**. An SA is a simplex logical connection; that is, it is unidirectional from source to destination. If both entities want to send secure datagrams to each other, then two SAs (that is, two logical connections) need to be established, one in each direction.

For example, consider once again the institutional VPN in Figure 8.27. This institution consists of a headquarters office, a branch office and, say,  $n$  traveling salespersons. For the sake of example, let's suppose that there is bi-directional IPsec traffic between headquarters and the branch office and bi-directional IPsec traffic between headquarters and the salespersons. In this VPN, how many SAs are there? To answer this question, note that there are two SAs between the headquarters gateway router and the branch-office gateway router (one in each direction); for each



**Figure 8.28** ♦ Security association (SA) from R1 to R2

salesperson’s laptop, there are two SAs between the headquarters gateway router and the laptop (again, one in each direction). So, in total, there are  $(2 + 2n)$  SAs. *Keep in mind, however, that not all traffic sent into the Internet by the gateway routers or by the laptops will be IPsec secured.* For example, a host in headquarters may want to access a Web server (such as Amazon or Google) in the public Internet. Thus, the gateway router (and the laptops) will emit into the Internet both vanilla IPv4 datagrams and secured IPsec datagrams.

Let’s now take a look “inside” an SA. To make the discussion tangible and concrete, let’s do this in the context of an SA from router R1 to router R2 in Figure 8.28. (You can think of Router R1 as the headquarters gateway router and Router R2 as the branch office gateway router from Figure 8.27.) Router R1 will maintain state information about this SA, which will include:

- A 32-bit identifier for the SA, called the **Security Parameter Index (SPI)**
- The origin interface of the SA (in this case 200.168.1.100) and the destination interface of the SA (in this case 193.68.2.23)
- The type of encryption to be used (for example, 3DES with CBC)
- The encryption key
- The type of integrity check (for example, HMAC with MD5)
- The authentication key

Whenever router R1 needs to construct an IPsec datagram for forwarding over this SA, it accesses this state information to determine how it should authenticate and encrypt the datagram. Similarly, router R2 will maintain the same state information for this SA and will use this information to authenticate and decrypt any IPsec datagram that arrives from the SA.

An IPsec entity (router or host) often maintains state information for many SAs. For example, in the VPN example in Figure 8.27 with  $n$  salespersons, the headquarters gateway router maintains state information for  $(2 + 2n)$  SAs. An IPsec entity stores the state information for all of its SAs in its **Security Association Database (SAD)**, which is a data structure in the entity’s OS kernel.

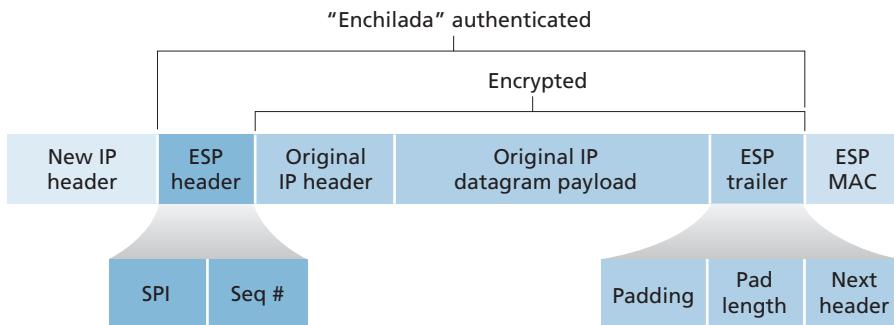
### 8.7.4 The IPsec Datagram

Having now described SAs, we can now describe the actual IPsec datagram. IPsec has two different packet forms, one for the so-called **tunnel mode** and the other for the so-called **transport mode**. The tunnel mode, being more appropriate for VPNs, is more widely deployed than the transport mode. In order to further de-mystify IPsec and avoid much of its complication, we henceforth focus exclusively on the tunnel mode. Once you have a solid grip on the tunnel mode, you should be able to easily learn about the transport mode on your own.

The packet format of the IPsec datagram is shown in Figure 8.29. You might think that packet formats are boring and insipid, but we will soon see that the IPsec datagram actually looks and tastes like a popular Tex-Mex delicacy! Let's examine the IPsec fields in the context of Figure 8.28. Suppose router R1 receives an ordinary IPv4 datagram from host 172.16.1.17 (in the headquarters network) which is destined to host 172.16.2.48 (in the branch-office network). Router R1 uses the following recipe to convert this “original IPv4 datagram” into an IPsec datagram:

- Appends to the back of the original IPv4 datagram (which includes the original header fields!) an “ESP trailer” field
- Encrypts the result using the algorithm and key specified by the SA
- Appends to the front of this encrypted quantity a field called “ESP header”; the resulting package is called the “enchilada”
- Creates an authentication MAC over the *whole enchilada* using the algorithm and key specified in the SA
- Appends the MAC to the back of the *enchilada* forming the *payload*
- Finally, creates a brand new IP header with all the classic IPv4 header fields (together normally 20 bytes long), which it appends before the payload

Note that the resulting IPsec datagram is a bona fide IPv4 datagram, with the traditional IPv4 header fields followed by a payload. But in this case, the payload



**Figure 8.29** ♦ IPsec datagram format

contains an ESP header, the original IP datagram, an ESP trailer, and an ESP authentication field (with the original datagram and ESP trailer encrypted). The original IP datagram has 172.16.1.17 for the source IP address and 172.16.2.48 for the destination IP address. Because the IPsec datagram includes the original IP datagram, these addresses are included (and encrypted) as part of the payload of the IPsec packet. But what about the source and destination IP addresses that are in the new IP header, that is, in the left-most header of the IPsec datagram? As you might expect, they are set to the source and destination router interfaces at the two ends of the tunnels, namely, 200.168.1.100 and 193.68.2.23. Also, the protocol number in this new IPv4 header field is not set to that of TCP, UDP, or SMTP, but instead to 50, designating that this is an IPsec datagram using the ESP protocol.

After R1 sends the IPsec datagram into the public Internet, it will pass through many routers before reaching R2. Each of these routers will process the datagram as if it were an ordinary datagram—they are completely oblivious to the fact that the datagram is carrying IPsec-encrypted data. For these public Internet routers, because the destination IP address in the outer header is R2, the ultimate destination of the datagram is R2.

Having walked through an example of how an IPsec datagram is constructed, let's now take a closer look at the ingredients in the enchilada. We see in Figure 8.29 that the ESP trailer consists of three fields: padding; pad length; and next header. Recall that block ciphers require the message to be encrypted to be an integer multiple of the block length. Padding (consisting of meaningless bytes) is used so that when added to the original datagram (along with the pad length and next header fields), the resulting “message” is an integer number of blocks. The pad-length field indicates to the receiving entity how much padding was inserted (and thus needs to be removed). The next header identifies the type (e.g., UDP) of data contained in the payload-data field. The payload data (typically the original IP datagram) and the ESP trailer are concatenated and then encrypted.

Appended to the front of this encrypted unit is the ESP header, which is sent in the clear and consists of two fields: the SPI and the sequence number field. The SPI indicates to the receiving entity the SA to which the datagram belongs; the receiving entity can then index its SAD with the SPI to determine the appropriate authentication/decryption algorithms and keys. The sequence number field is used to defend against replay attacks.

The sending entity also appends an authentication MAC. As stated earlier, the sending entity calculates a MAC over the whole enchilada (consisting of the ESP header, the original IP datagram, and the ESP trailer—with the datagram and trailer being encrypted). Recall that to calculate a MAC, the sender appends a secret MAC key to the enchilada and then calculates a fixed-length hash of the result.

When R2 receives the IPsec datagram, R2 observes that the destination IP address of the datagram is R2 itself. R2 therefore processes the datagram. Because the protocol field (in the left-most IP header) is 50, R2 sees that it should apply IPsec ESP processing to the datagram. First, peering into the enchilada, R2 uses the SPI to determine to which SA the datagram belongs. Second, it calculates the MAC of the enchilada and verifies that the MAC is consistent with the value in the ESP

MAC field. If it is, it knows that the enchilada comes from R1 and has not been tampered with. Third, it checks the sequence-number field to verify that the datagram is fresh (and not a replayed datagram). Fourth, it decrypts the encrypted unit using the decryption algorithm and key associated with the SA. Fifth, it removes padding and extracts the original, vanilla IP datagram. And finally, sixth, it forwards the original datagram into the branch office network toward its ultimate destination. Whew, what a complicated recipe, huh? Well no one ever said that preparing and unraveling an enchilada was easy!

There is actually another important subtlety that needs to be addressed. It centers on the following question: When R1 receives an (unsecured) datagram from a host in the headquarters network, and that datagram is destined to some destination IP address outside of headquarters, how does R1 know whether it should be converted to an IPsec datagram? And if it is to be processed by IPsec, how does R1 know which SA (of many SAs in its SAD) should be used to construct the IPsec datagram? The problem is solved as follows. Along with a SAD, the IPsec entity also maintains another data structure called the **Security Policy Database (SPD)**. The SPD indicates what types of datagrams (as a function of source IP address, destination IP address, and protocol type) are to be IPsec processed; and for those that are to be IPsec processed, which SA should be used. In a sense, the information in a SPD indicates “what” to do with an arriving datagram; the information in the SAD indicates “how” to do it.

### Summary of IPsec Services

So what services does IPsec provide, exactly? Let us examine these services from the perspective of an attacker, say Trudy, who is a woman-in-the-middle, sitting somewhere on the path between R1 and R2 in Figure 8.28. Assume throughout this discussion that Trudy does not know the authentication and encryption keys used by the SA. What can and cannot Trudy do? First, Trudy cannot see the original datagram. If fact, not only is the data in the original datagram hidden from Trudy, but so is the protocol number, the source IP address, and the destination IP address. For datagrams sent over the SA, Trudy only knows that the datagram originated from 200.168.1.100 and is destined to 193.68.2.23. She does not know if it is carrying TCP, UDP, or ICMP data; she does not know if it is carrying HTTP, SMTP, or some other type of application data. This confidentiality thus goes a lot farther than SSL. Second, suppose Trudy tries to tamper with a datagram in the SA by flipping some of its bits. When this tampered datagram arrives at R2, it will fail the integrity check (using the MAC), thwarting Trudy’s vicious attempts once again. Third, suppose Trudy tries to masquerade as R1, creating a IPsec datagram with source 200.168.1.100 and destination 193.68.2.23. Trudy’s attack will be futile, as this datagram will again fail the integrity check at R2. Finally, because IPsec includes sequence numbers, Trudy will not be able create a successful replay attack. In summary, as claimed at the beginning of this section, IPsec provides—between any pair of devices that process packets through the network layer—confidentiality, source authentication, data integrity, and replay-attack prevention.

### 8.7.5 IKE: Key Management in IPsec

When a VPN has a small number of end points (for example, just two routers as in Figure 8.28), the network administrator can manually enter the SA information (encryption/authentication algorithms and keys, and the SPIs) into the SADs of the endpoints. Such “manual keying” is clearly impractical for a large VPN, which may consist of hundreds or even thousands of IPsec routers and hosts. Large, geographically distributed deployments require an automated mechanism for creating the SAs. IPsec does this with the Internet Key Exchange (IKE) protocol, specified in RFC 5996.

IKE has some similarities with the handshake in SSL (see Section 8.6). Each IPsec entity has a certificate, which includes the entity’s public key. As with SSL, the IKE protocol has the two entities exchange certificates, negotiate authentication and encryption algorithms, and securely exchange key material for creating session keys in the IPsec SAs. Unlike SSL, IKE employs two phases to carry out these tasks.

Let’s investigate these two phases in the context of two routers, R1 and R2, in Figure 8.28. The first phase consists of two exchanges of message pairs between R1 and R2:

- During the first exchange of messages, the two sides use Diffie-Hellman (see Homework Problems) to create a bi-directional **IKE SA** between the routers. To keep us all confused, this bi-directional IKE SA is entirely different from the IPsec SAs discussed in Sections 8.6.3 and 8.6.4. The IKE SA provides an authenticated and encrypted channel between the two routers. During this first message-pair exchange, keys are established for encryption and authentication for the IKE SA. Also established is a master secret that will be used to compute IPsec SA keys later in phase 2. Observe that during this first step, RSA public and private keys are not used. In particular, neither R1 nor R2 reveals its identity by signing a message with its private key.
- During the second exchange of messages, both sides reveal their identity to each other by signing their messages. However, the identities are not revealed to a passive sniffer, since the messages are sent over the secured IKE SA channel. Also during this phase, the two sides negotiate the IPsec encryption and authentication algorithms to be employed by the IPsec SAs.

In phase 2 of IKE, the two sides create an SA in each direction. At the end of phase 2, the encryption and authentication session keys are established on both sides for the two SAs. The two sides can then use the SAs to send secured datagrams, as described in Sections 8.7.3 and 8.7.4. The primary motivation for having two phases in IKE is computational cost—since the second phase doesn’t involve any public-key cryptography, IKE can generate a large number of SAs between the two IPsec entities with relatively little computational cost.

## 8.8 Securing Wireless LANs and 4G/5G Cellular Networks

Security is a particularly important concern in wireless networks, where the attacker can sniff frames by simply positioning a receiving device anywhere within the transmission range of the sender. This is true in both 802.11 wireless LANs, as well as in 4G/5G cellular networks. In both settings, we'll see extensive use of the fundamental security techniques that we studied earlier in this chapter, including the use of nonces for authentication, cryptographic hashing for message integrity, derivation of shared symmetric keys for encrypting user-session data, and the extensive use of the AES encryption standard. We will also see, as is also the case in wired Internet settings, that wireless security protocols have undergone constant evolution, as researchers and hackers discover weaknesses and flaws in existing security protocols.

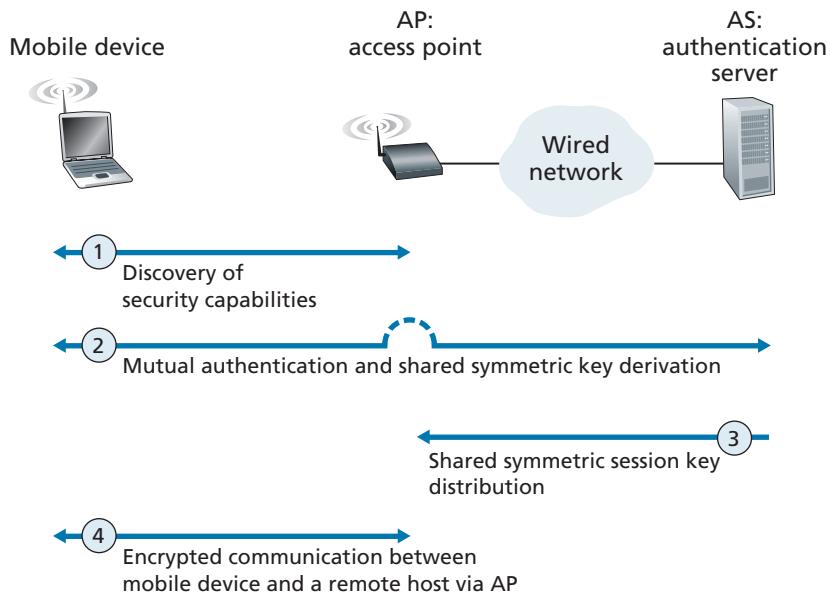
In this section, we present a brief introduction to wireless security in both 802.11(WiFi) and 4G/5G settings. For a more in-depth treatment, see the highly readable 802.11 security books [Edney 2003; Wright 2015], the excellent coverage of 3G/4G/5G security in [Sauter 2014], and recent surveys [Zou 2016; Kohlios 2018].

### 8.8.1 Authentication and Key Agreement in 802.11 Wireless LANs

Let's start our discussion of 802.11 security by identifying two (of many [Zou 2016]) critical security concerns that we'll want an 802.11 network to handle:

- *Mutual authentication.* Before a mobile device is allowed to fully attach to an access point and send datagrams to remote hosts, the network will typically want to first authenticate the device—to verify the identity of the mobile device attaching to the network, and to check that device's access privileges. Similarly, the mobile device will want to authenticate the network to which it is attaching—to make sure that the network it is joining is truly the network to which it wants to attach. This two-way authentication is known as **mutual authentication**.
- *Encryption.* Since 802.11 frames will be exchanged over a wireless channel that can be sniffed and manipulated by potential ne'er do-wells, it will be important to encrypt link-level frames carrying user-level data exchanged between the mobile device and the access point (AP). Symmetric key encryption is used in practice, since encryption and decryption must be performed at high speeds. The mobile device and AP will need to derive the symmetric encryption and decryption keys to be used.

Figure 8.30 illustrates the scenario of a mobile device wishing to attach to an 802.11 network. We see the two usual network components that we encountered



**Figure 8.30** ♦ Mutual authentication and encryption-key derivation in WPA

in our earlier study of 802.11 networks in Section 7.3—the mobile device and the AP. We also see a new architectural component, the **authentication server** (AS) that will be responsible for authenticating the mobile device. The authentication server might be co-located in the AP, but more typically and as shown in Figure 8.30, it is implemented as a separate server that provides authentication services. For authentication, the AP serves as a pass-through device, relaying authentication and key derivation messages between the mobile device and the authentication server. Such an authentication server would typically provide authentication services for all APs within its network.

We can identify four distinct phases to the process of mutual authentication and encryption-key derivation and use in Figure 8.30:

1. *Discovery*. In the discovery phase, the AP advertises its presence and the forms of authentication and encryption that can be provided to the mobile device. The mobile device then requests the specific forms of authentication and encryption that it desires. Although the device and AP are already exchanging messages, the device has not yet been authenticated nor does it have an encryption key for frame transmission over the wireless link, and so several more steps will be required before the device can communicate securely through the AP.
2. *Mutual authentication and shared symmetric key derivation*. This is the most critical step in “securing” the 802.11 channel. As we will see, this step is

greatly facilitated by assuming (which is true in practice in both 802.11 and 4G/5G networks) that the authentication server and the mobile device already have a **shared common secret** before starting mutual authentication. In this step, the device and the authentication server will use this shared secret along with nonces (to prevent relay attacks) and cryptographic hashing (to ensure message integrity) in authenticating each other. They will also derive the shared session key to be used by the mobile device and the AP to encrypt frames transmitted over the 802.11 wireless link.

3. *Shared symmetric session key distribution.* Since the symmetric encryption key is derived at the mobile device and the authentication server, a protocol will be needed for the authentication server to inform the AP of the shared symmetric session key. While this is rather straightforward, it still is a necessary step.
4. *Encrypted communication between mobile device and a remote host via the AP.* This communication happens as we saw earlier in Section 7.3.2, with the link-layer frames sent between the mobile device and the AP being encrypted using the shared session key created and distributed by Steps 2 and 3. AES symmetric key cryptography, which we covered earlier in Section 8.2.1, is typically used in practice for encrypting/decrypting 802.11 frame data.

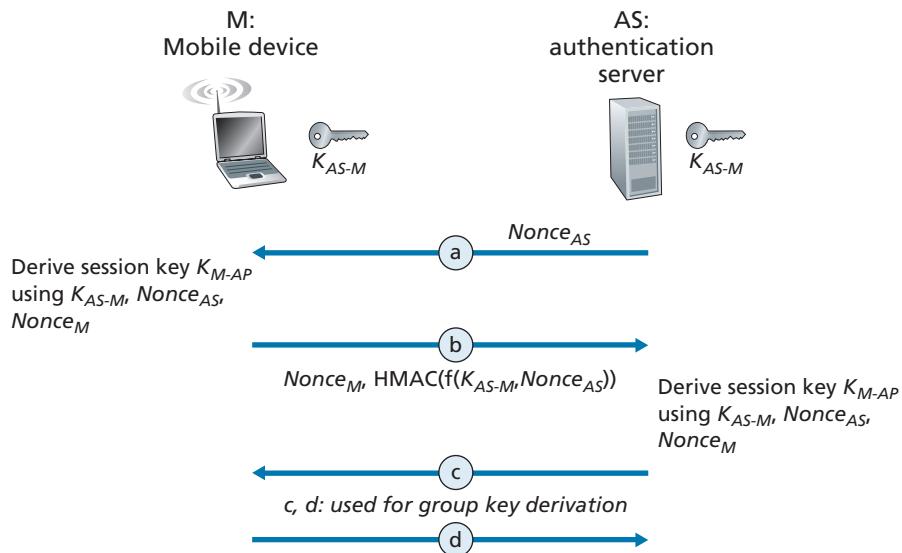
### Mutual Authentication and Shared Symmetric Session Key Derivation

The topics of mutual authentication and shared symmetric session key derivation are the central components of 802.11 security. Since it is here that security flaws in various earlier versions of 802.11 security have been discovered, let's tackle these challenges first.

The issue of 802.11 security has attracted considerable attention in both technical circles and in the media. While there has been considerable discussion, there has been little debate—there is universal agreement that the original 802.11 security specification known collectively as Wired Equivalent Privacy (WEP) contained a number of serious security flaws [Fluhrer 2001; Stubblefield 2002]. Once these flaws were discovered, public domain software was soon available exploiting these holes, making users of WEP-secured 802.11 WLANs as open to security attacks as users who used no security features at all. Readers interested in learning about WEP can consult the references, as well as earlier editions of this textbook, which covered WEP. As always, retired material from this book is available on the Companion Website.

WiFi Protected Access (WPA1) was developed in 2003 by the WiFi Alliance [WiFi 2020] to overcome WEP's security flaws. The initial version of WPA1 improved on WEP by introducing message integrity checks, and avoiding attacks that allowed a user to infer encryption keys after observing the stream of encrypted messages for a period of time. WPA1 soon gave way to WPA2, which mandated the use of AES symmetric key encryption.

At the heart of WPA is a four-way handshake protocol that performs both mutual authentication and shared symmetric session-key derivation. The handshake protocol is shown in Figure 8.31 in simplified form. Note that both the mobile device (M) and the authentication server (AS) begin knowing a shared secret key  $K_{AS-M}$ .



**Figure 8.31** ♦ The WPA2 four-way handshake

(e.g., a password). One of their tasks will be to derive a shared symmetric session-key,  $K_{M-AP}$ , which will be used to encrypt/decrypt frames that are later transmitted between the mobile device (M) and the AP.

Mutual authentication and shared symmetric session-key derivation are accomplished in the first two steps, a and b, of the four-way handshake shown in Figure 8.31. Steps c and d are used to derive a second key used for group communication; see [Kohlios 2018; Zou 2016] for details.

- In this first step, the authentication server (AS) generates a nonce,  $Nonce_{AS}$ , and sends it to the mobile device. Recall from Section 8.4 that nonces are used to avoid playback attacks and prove the “liveness” of the other side being authenticated.
- The mobile device, M, receives the nonce,  $Nonce_{AS}$ , from the AS and generates its own nonce,  $Nonce_M$ . The mobile device then generates the symmetric shared session key,  $K_{M-AP}$ , using  $Nonce_{AS}$ ,  $Nonce_M$ , the initial shared secret key  $K_{AS-M}$ , its MAC address, and the MAC address of the AS. It then sends its nonce,  $Nonce_M$ , and an HMAC-signed (see Figure 8.9) value that encodes  $Nonce_{AS}$  and the original shared secret.

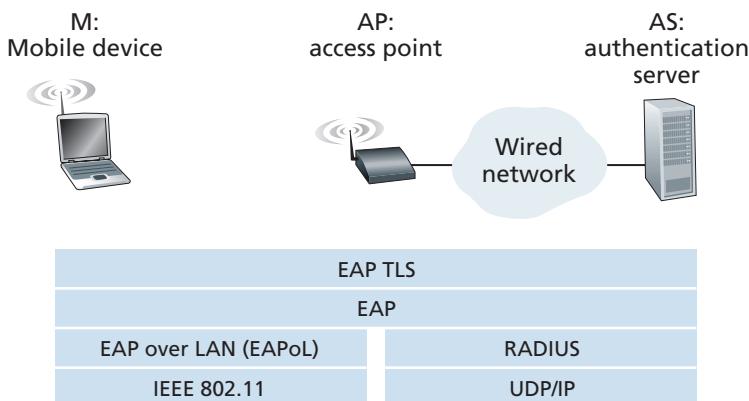
The AS receives this message from M. By looking at the HMAC-signed version of the nonce it had just recently sent,  $Nonce_{AS}$ , the authentication server knows the mobile device is live; because the mobile device was able to encrypt using the shared secret key,  $K_{AS-M}$ , the AS also knows that the mobile device

is indeed who it claims to be (i.e., a device that knows the shared initial secret). The AS has thus authenticated the mobile device! The AS can also now perform the exact same computation as the mobile device to derive the shared symmetric session-key,  $K_{M-AP}$ , using the  $Nonce_M$  it received,  $Nonce_{AS}$ , the initial shared secret key  $K_{AS-M}$ , its MAC address and the MAC address of the mobile device. At this point both the mobile device and the authentication server have computed the same shared symmetric key,  $K_{M-AP}$ , which will be used to encrypt/decrypt frames transmitted between the mobile device and the AP. The AS informs the AP of this key value in Step 3 in Figure 8.30.

WPA3 was released in June 2018 as an update to WPA2. The update addresses an attack on the four-way handshake protocol that could induce the reuse of previously used nonces [Vannoof 2017] but still permits the use of the four-way handshake as a legacy protocol and includes longer key lengths, among other changes [WiFi 2019].

### 802.11 Security Messaging Protocols

Figure 8.32 shows the protocols used to implement the 802.11 security framework discussed above. The Extensible Authentication Protocol (EAP) [RFC 3748] defines the end-to-end message formats used in a simple request/response mode of interaction between the mobile device and authentication server, and are certified under WPA2. As shown in Figure 8.32, EAP messages are encapsulated using EAPoL (EAP over LAN) and sent over the 802.11 wireless link. These EAP messages are then decapsulated at the access point, and then re-encapsulated using the RADIUS protocol for



**Figure 8.32** ♦ EAP is an end-to-end protocol. EAP messages are encapsulated using EAPoL over the wireless link between the mobile device and the access point, and using RADIUS over UDP/IP between the access point and the authentication server

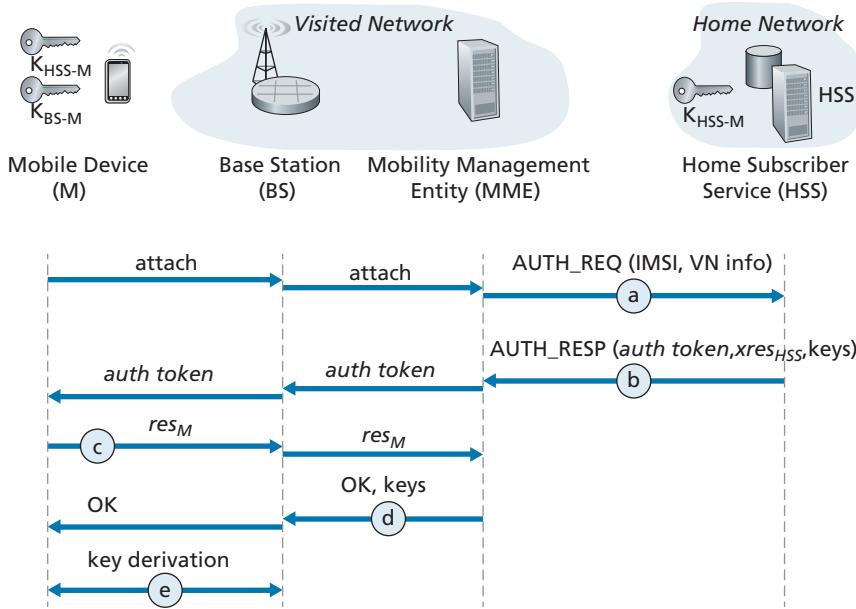
transmission over UDP/IP to the authentication server. While the RADIUS server and protocol [RFC 2865] are not required, they are *de facto* standard components. The recently standardized DIAMETER protocol [RFC 3588] is projected to eventually replace RADIUS in the future.

### 8.8.2 Authentication and Key Agreement in 4G/5G Cellular Networks

In this section, we describe mutual authentication and key-generation mechanisms in 4G/5G networks. Many of the approaches we'll encounter here parallel those that we just studied in 802.11 networks, with the notable exception that in 4G/5G networks, mobile devices may be attached to their home network (i.e., the cellular carrier network to which they are subscribed), or may be roaming on a visited network. In this latter case, the visited and home networks will need to interact when authenticating a mobile device and generating encryption keys. Before continuing, you may want to re-familiarize yourself with 4G/5G network architecture by re-reading Sections 7.4 and 7.7.1.

The goals of mutual authentication and key generation are the same in the 4G/5G setting as in the 802.11 setting. In order to encrypt the contents of frames being transmitted over the wireless channel, the mobile device and base station will need to derive a shared symmetric encryption key. In addition, the network to which the mobile device is attaching will need to authenticate the device's identity and check its access privileges. Similarly, the mobile device will also want to authenticate the network to which it is attaching. While the network's need to authenticate a mobile device may be obvious, the need for authentication in the reverse direction may not be so clear. However, there are documented cases of ne'er-do-wells operating rogue cellular base stations that entice unsuspecting mobile devices to attach to the rogue network, exposing a device to a number of attacks [Li 2017]. So, as in the case of 802.11 WLANs, a mobile device should exercise abundant caution when attaching to a cellular network!

Figure 8.33 illustrates the scenario of mobile device attaching to a 4G cellular network. At the top of Figure 8.33, we see many of the 4G components that we encountered earlier in Section 7.4—the mobile device (M), the base station (BS), the mobility management entity (MME) in the network to which the mobile device wants to attach, and the home subscriber service (HSS) in the mobile device's home network. A comparison of Figures 8.30 and 8.33 shows the similarities and differences between the 802.11 and 4G security settings. We again see a mobile device and a base station; the user session-key derived during network attachment,  $K_{BS-M}$ , will be used to encrypt/decrypt frames transmitted over their wireless link. The 4G MME and HSS together will play a role similar to that of the authentication server in the 802.11 setting. Note that the HSS and the mobile device also share a common secret,  $K_{HSS-M}$ , known to both entities before authentication begins. This key is stored in the mobile device's SIM card, and in the HSS database in the mobile device's home network.



**Figure 8.33** ♦ Mutual authentication and key agreement in a 4G LTE cellular network

The 4G Authentication and Key Agreement (AKA) protocol consists of the following steps:

- Authentication request to HSS.** When the mobile device first requests, via a base station, to attach to the network, it sends an attach message containing its international mobile subscriber identity (IMSI) that is relayed to the Mobility Management Entity (MME). The MME will then send the IMSI and information about the visited network (shown as “VN info” in Figure 8.33) to the Home Subscriber Service (HSS) in the device’s home network. In Section 7.4, we described how the MME is able to communicate with the HSS through the all-IP global network of interconnected cellular networks.
- Authentication response from HSS.** The HSS performs cryptographic operations using the shared-in-advance secret key,  $K_{HSS-M}$ , to derive an authentication token,  $auth\_token$ , and an expected authentication response token,  $xres_{HSS}$ .  $auth\_token$  contains information encrypted by the HSS using  $K_{HSS-M}$  that will allow the mobile device to know that whoever computed  $auth\_token$  knows the secret key. For example, suppose the HSS computes  $K_{HSS-M}(\text{IMSI})$ , that is, encrypts the device’s IMSI using  $K_{HSS-M}$  and sends that value as  $auth\_token$ . When the mobile device receives that encrypted value and uses its secret key to decrypt this value, that is, to compute

$K_{HSS-M}(K_{HSS-M}(\text{IMSI})) = \text{IMSI}$ , it knows that the HSS that generated  $\text{auth\_token}$  knows its secret key. The mobile device can thus authenticate the HSS.

The expected authentication response token,  $xres_{HSS}$ , contains a value that the mobile device will need to be able to compute (using  $K_{HSS-M}$ ) and return to the MME to prove that *it* (the mobile device) knows the secret key, thus authenticating the mobile device to the MME.

Note that the MME only plays a middleman role here, receiving the authentication response message, keeping  $xres_{HSS}$  for later use, extracting the authentication token and forwarding it to the mobile device. In particular it need not know, and will not learn, the secret key,  $K_{HSS-M}$ .

- c. *Authentication response from mobile device.* The mobile device receives  $\text{auth\_token}$  and computes  $K_{HSS-M}(K_{HSS-M}(\text{IMSI})) = \text{IMSI}$ , thus authenticating the HSS. The mobile device then computes a value  $res_M$ —using its secret key to make the exact same cryptographic calculation that the HSS had made to compute  $xres_{HSS}$ —and sends this value to the MME.
- d. *Mobile device authentication.* The MMS compares the mobile-computed value of  $res_M$  with the HSS-computed value of  $xres_{HSS}$ . If they match, the mobile device is authenticated, since the mobile has proven to the MME that it and the HSS both know the common secret key. The MMS informs the base station and mobile device that mutual authentication is complete, and sends the base station keys that will be used in step e.
- e. *Data plane and control plane key derivation.* The mobile device and the base station will each determine the keys used for encrypting/decrypting their frame transmissions over the wireless channel. Separate keys will be derived for data plane and control plane frame transmissions. The AES encryption algorithm that we saw in use in 802.11 networks is also used in 4G/5G networks.

Our discussion above has focused on authentication and key agreement in 4G networks. Although much of the 4G security is being carried forward into 5G, there are some important changes:

- First, note that in our discussion above that it is the MME in the visited network that makes the authentication decision. A significant change underway in 5G network security is to allow authentication services to be provided by the home network, with the visited network playing an even smaller middleman role. While the visited network may still reject an authentication from a mobile device, it is up to the home network to accept the authentication request in this new 5G scenario.
- 5G networks will support the Authentication and Key Agreement (AKA) protocol described above, as well as two new additional protocols for authentication and key agreement. One of these, known as AKA', is closely related to the 4G AKA protocol. It also uses the shared-in-advance secret key,  $K_{HSS-M}$ . However, since it uses the EAP protocol that we encountered earlier in Figure 8.33 in the context of 802.11 authentication, 5G AKA' has different message flows than that

of 4G AKA. The second new 5G protocol is meant for an IoT environment, and does not require a shared-in-advance secret key.

- An additional change in 5G is to use public key cryptography techniques to encrypt a device’s permanent identity (i.e., its IMSI) so that it is never transmitted in cleartext.

In this section, we have only briefly overviewed mutual authentication and key agreement in 4G /5G networks. As we have seen, they make extensive use of the security techniques that we studied earlier in this chapter. More details on 4G/5G security can be found in [3GPP SAE 2019; Cable Labs 2019; Cichonski 2017].

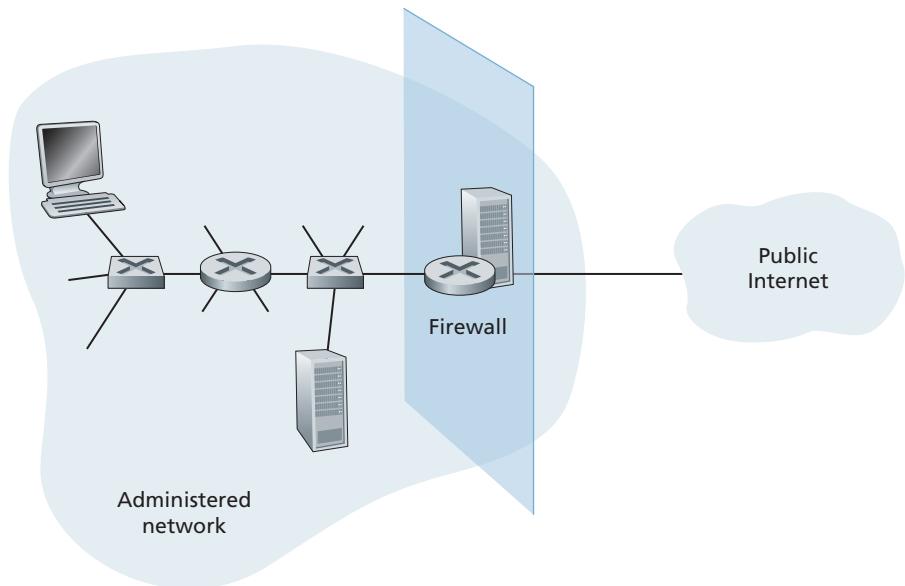
## 8.9 Operational Security: Firewalls and Intrusion Detection Systems

We’ve seen throughout this chapter that the Internet is not a very safe place—bad guys are out there, wreaking all sorts of havoc. Given the hostile nature of the Internet, let’s now consider an organization’s network and the network administrator who administers it. From a network administrator’s point of view, the world divides quite neatly into two camps—the good guys (who belong to the organization’s network, and who should be able to access resources inside the organization’s network in a relatively unconstrained manner) and the bad guys (everyone else, whose access to network resources must be carefully scrutinized). In many organizations, ranging from medieval castles to modern corporate office buildings, there is a single point of entry/exit where both good guys and bad guys entering and leaving the organization are security-checked. In a castle, this was done at a gate at one end of the drawbridge; in a corporate building, this is done at the security desk. In a computer network, when traffic entering/leaving a network is security-checked, logged, dropped, or forwarded, it is done by operational devices known as firewalls, intrusion detection systems (IDSs), and intrusion prevention systems (IPSs).

### 8.9.1 Firewalls

A **firewall** is a combination of hardware and software that isolates an organization’s internal network from the Internet at large, allowing some packets to pass and blocking others. A firewall allows a network administrator to control access between the outside world and resources within the administered network by managing the traffic flow to and from these resources. A firewall has three goals:

- *All traffic from outside to inside, and vice versa, passes through the firewall.* Figure 8.34 shows a firewall, sitting squarely at the boundary between the administered network and the rest of the Internet. While large organizations may use



**Figure 8.34** ♦ Firewall placement between the administered network and the outside world

multiple levels of firewalls or distributed firewalls [Skoudis 2006], locating a firewall at a single access point to the network, as shown in Figure 8.34, makes it easier to manage and enforce a security-access policy.

- *Only authorized traffic, as defined by the local security policy, will be allowed to pass.* With all traffic entering and leaving the institutional network passing through the firewall, the firewall can restrict access to authorized traffic.
- *The firewall itself is immune to penetration.* The firewall itself is a device connected to the network. If not designed or installed properly, it can be compromised, in which case it provides only a false sense of security (which is worse than no firewall at all!).

Cisco and Check Point are two of the leading firewall vendors today. You can also easily create a firewall (packet filter) from a Linux box using iptables (public-domain software that is normally shipped with Linux). Furthermore, as discussed in Chapters 4 and 5, firewalls are now frequently implemented in routers and controlled remotely using SDNs.

Firewalls can be classified in three categories: **traditional packet filters**, **stateful filters**, and **application gateways**. We'll cover each of these in turn in the following subsections.

### Traditional Packet Filters

As shown in Figure 8.34, an organization typically has a gateway router connecting its internal network to its ISP (and hence to the larger public Internet). All traffic leaving and entering the internal network passes through this router, and it is at this router where **packet filtering** occurs. A packet filter examines each datagram in isolation, determining whether the datagram should be allowed to pass or should be dropped based on administrator-specific rules. Filtering decisions are typically based on:

- IP source or destination address
- Protocol type in IP datagram field: TCP, UDP, ICMP, OSPF, and so on
- TCP or UDP source and destination port
- TCP flag bits: SYN, ACK, and so on
- ICMP message type
- Different rules for datagrams leaving and entering the network
- Different rules for the different router interfaces

A network administrator configures the firewall based on the policy of the organization. The policy may take user productivity and bandwidth usage into account as well as the security concerns of an organization. Table 8.5 lists a number of possible policies an organization may have, and how they would be addressed with a packet filter. For example, if the organization doesn't want any incoming TCP connections except those for its public Web server, it can block all incoming TCP SYN segments except TCP SYN segments with destination port 80 and the destination IP address corresponding to the Web server. If the organization doesn't want its users to monopolize access bandwidth with Internet radio applications, it can block all not-critical

| Policy                                                                               | Firewall Setting                                                                |
|--------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| No outside Web access.                                                               | Drop all outgoing packets to any IP address, port 80.                           |
| No incoming TCP connections, except those for organization's public Web server only. | Drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80.    |
| Prevent Web radios from eating up the available bandwidth.                           | Drop all incoming UDP packets—except DNS packets.                               |
| Prevent your network from being used for a smurf DoS attack.                         | Drop all ICMP ping packets going to a "broadcast" address (eg 130.207.255.255). |
| Prevent your network from being tracerouted.                                         | Drop all outgoing ICMP TTL expired traffic.                                     |

**Table 8.5** ♦ Policies and corresponding filtering rules for an organization's network 130.207/16 with Web server at 130.207.244.203

UDP traffic (since Internet radio is often sent over UDP). If the organization doesn't want its internal network to be mapped (tracerouted) by an outsider, it can block all ICMP TTL expired messages leaving the organization's network.

A filtering policy can be based on a combination of addresses and port numbers. For example, a filtering router could forward all Telnet datagrams (those with a port number of 23) except those going to and coming from a list of specific IP addresses. This policy permits Telnet connections to and from hosts on the allowed list. Unfortunately, basing the policy on external addresses provides no protection against datagrams that have had their source addresses spoofed.

Filtering can also be based on whether or not the TCP ACK bit is set. This trick is quite useful if an organization wants to let its internal clients connect to external servers but wants to prevent external clients from connecting to internal servers. Recall from Section 3.5 that the first segment in every TCP connection has the ACK bit set to 0, whereas all the other segments in the connection have the ACK bit set to 1. Thus, if an organization wants to prevent external clients from initiating connections to internal servers, it simply filters all incoming segments with the ACK bit set to 0. This policy kills all TCP connections originating from the outside, but permits connections originating internally.

Firewall rules are implemented in routers with access control lists, with each router interface having its own list. An example of an access control list for an organization 222.22/16 is shown in Table 8.6. This access control list is for an interface that connects the router to the organization's external ISPs. Rules are applied to each datagram that passes through the interface from top to bottom. The first two rules together allow internal users to surf the Web: The first rule allows any TCP packet with destination port 80 to leave the organization's network; the second rule allows any TCP packet with source port 80 and the ACK bit set to enter the organization's network. Note that if an external source attempts to establish a TCP connection with

| action | source address          | dest address            | protocol | source port | dest port | flag bit |
|--------|-------------------------|-------------------------|----------|-------------|-----------|----------|
| allow  | 222.22/16               | outside of<br>222.22/16 | TCP      | > 1023      | 80        | any      |
| allow  | outside of<br>222.22/16 | 222.22/16               | TCP      | 80          | > 1023    | ACK      |
| allow  | 222.22/16               | outside of<br>222.22/16 | UDP      | > 1023      | 53        | —        |
| allow  | outside of<br>222.22/16 | 222.22/16               | UDP      | 53          | > 1023    | —        |
| deny   | all                     | all                     | all      | all         | all       | all      |

**Table 8.6** ♦ An access control list for a router interface

an internal host, the connection will be blocked, even if the source or destination port is 80. The second two rules together allow DNS packets to enter and leave the organization's network. In summary, this rather restrictive access control list blocks all traffic except Web traffic initiated from within the organization and DNS traffic. [CERT Filtering 2012] provides a list of recommended port/protocol packet filterings to avoid a number of well-known security holes in existing network applications.

Readers with sharp memories may recall we encountered access control lists similar to Table 8.6 when we studied generalized forwarding in Section 4.4.3 of Chapter 4. Indeed, we provided an example there of how generalized forwarding rules can be used to build a packet-filtering firewall.

### Stateful Packet Filters

In a traditional packet filter, filtering decisions are made on each packet in isolation. Stateful filters actually track TCP connections, and use this knowledge to make filtering decisions.

To understand stateful filters, let's reexamine the access control list in Table 8.6. Although rather restrictive, the access control list in Table 8.6 nevertheless allows any packet arriving from the outside with ACK = 1 and source port 80 to get through the filter. Such packets could be used by attackers in attempts to crash internal systems with malformed packets, carry out denial-of-service attacks, or map the internal network. The naive solution is to block TCP ACK packets as well, but such an approach would prevent the organization's internal users from surfing the Web.

Stateful filters solve this problem by tracking all ongoing TCP connections in a connection table. This is possible because the firewall can observe the beginning of a new connection by observing a three-way handshake (SYN, SYNACK, and ACK); and it can observe the end of a connection when it sees a FIN packet for the connection. The firewall can also (conservatively) assume that the connection is over when it hasn't seen any activity over the connection for, say, 60 seconds. An example connection table for a firewall is shown in Table 8.7. This connection table indicates that there are currently three ongoing TCP connections, all of which have been initiated from within the organization. Additionally, the stateful filter includes a new column, "check connection," in its access control list, as

| source address | dest address  | source port | dest port |
|----------------|---------------|-------------|-----------|
| 222.22.1.7     | 37.96.87.123  | 12699       | 80        |
| 222.22.93.2    | 199.1.205.23  | 37654       | 80        |
| 222.22.65.143  | 203.77.240.43 | 48712       | 80        |

**Table 8.7** ♦ Connection table for stateful filter

| action | source address          | dest address            | protocol | source port | dest port | flag bit | check connxion |
|--------|-------------------------|-------------------------|----------|-------------|-----------|----------|----------------|
| allow  | 222.22/16               | outside of<br>222.22/16 | TCP      | > 1023      | 80        | any      |                |
| allow  | outside of<br>222.22/16 | 222.22/16               | TCP      | 80          | > 1023    | ACK      | X              |
| allow  | 222.22/16               | outside of<br>222.22/16 | UDP      | > 1023      | 53        | —        |                |
| allow  | outside of<br>222.22/16 | 222.22/16               | UDP      | 53          | > 1023    | —        | X              |
| deny   | all                     | all                     | all      | all         | all       | all      |                |

**Table 8.8** ♦ Access control list for stateful filter

shown in Table 8.8. Note that Table 8.8 is identical to the access control list in Table 8.6, except now it indicates that the connection should be checked for two of the rules.

Let's walk through some examples to see how the connection table and the extended access control list work hand-in-hand. Suppose an attacker attempts to send a malformed packet into the organization's network by sending a datagram with TCP source port 80 and with the ACK flag set. Further suppose that this packet has source port number 12543 and source IP address 150.23.23.155. When this packet reaches the firewall, the firewall checks the access control list in Table 8.7, which indicates that the connection table must also be checked before permitting this packet to enter the organization's network. The firewall duly checks the connection table, sees that this packet is not part of an ongoing TCP connection, and rejects the packet. As a second example, suppose that an internal user wants to surf an external Web site. Because this user first sends a TCP SYN segment, the user's TCP connection gets recorded in the connection table. When the Web server sends back packets (with the ACK bit necessarily set), the firewall checks the table and sees that a corresponding connection is in progress. The firewall will thus let these packets pass, thereby not interfering with the internal user's Web surfing activity.

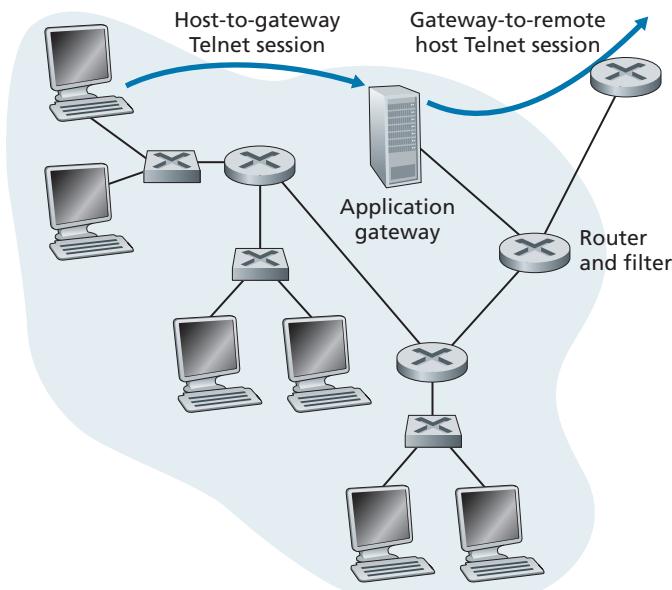
### Application Gateway

In the examples above, we have seen that packet-level filtering allows an organization to perform coarse-grain filtering on the basis of the contents of IP and TCP/UDP headers, including IP addresses, port numbers, and acknowledgment bits. But what if an organization wants to provide a Telnet service to a restricted set of internal users

(as opposed to IP addresses)? And what if the organization wants such privileged users to authenticate themselves first before being allowed to create Telnet sessions to the outside world? Such tasks are beyond the capabilities of traditional and stateful filters. Indeed, information about the identity of the internal users is application-layer data and is not included in the IP/TCP/UDP headers.

To have finer-level security, firewalls must combine packet filters with application gateways. Application gateways look beyond the IP/TCP/UDP headers and make policy decisions based on application data. An **application gateway** is an application-specific server through which all application data (inbound and outbound) must pass. Multiple application gateways can run on the same host, but each gateway is a separate server with its own processes.

To get some insight into application gateways, let's design a firewall that allows only a restricted set of internal users to Telnet outside and prevents all external clients from Telneting inside. Such a policy can be accomplished by implementing a combination of a packet filter (in a router) and a Telnet application gateway, as shown in Figure 8.35. The router's filter is configured to block all Telnet connections except those that originate from the IP address of the application gateway. Such a filter configuration forces all outbound Telnet connections to pass through the application gateway. Consider now an internal user who wants to Telnet to the outside world. The user must first set up a Telnet session with the application gateway. An application running in the gateway, which listens for incoming Telnet sessions, prompts the



**Figure 8.35** ♦ Firewall consisting of an application gateway and a filter

user for a user ID and password. When the user supplies this information, the application gateway checks to see if the user has permission to Telnet to the outside world. If not, the Telnet connection from the internal user to the gateway is terminated by the gateway. If the user has permission, then the gateway (1) prompts the user for the host name of the external host to which the user wants to connect, (2) sets up a Telnet session between the gateway and the external host, and (3) relays to the external host all data arriving from the user, and relays to the user all data arriving from the external host. Thus, the Telnet application gateway not only performs user authorization but also acts as a Telnet server and a Telnet client, relaying information between the user and the remote Telnet server. Note that the filter will permit step 2 because the gateway initiates the Telnet connection to the outside world.

## CASE HISTORY

### ANONYMITY AND PRIVACY

Suppose you want to visit a controversial Web site (for example, a political activist site) and you (1) don't want to reveal your IP address to the Web site, (2) don't want your local ISP (which may be your home or office ISP) to know that you are visiting the site, and (3) don't want your local ISP to see the data you are exchanging with the site. If you use the traditional approach of connecting directly to the Web site without any encryption, you fail on all three counts. Even if you use SSL, you fail on the first two counts: Your source IP address is presented to the Web site in every datagram you send; and the destination address of every packet you send can easily be sniffed by your local ISP.

To obtain privacy and anonymity, you can instead use a combination of a trusted proxy server and SSL, as shown in Figure 8.36. With this approach, you first make an SSL connection to the trusted proxy. You then send, into this SSL connection,

The diagram shows a user named Alice connected to a computer. A blue line labeled "SSL" connects her computer to a vertical rectangular box labeled "Anonymizing Proxy". From the proxy, a blue line labeled "Cleartext" connects to a final server. The path between Alice's computer and the proxy passes through two circular nodes with "X" symbols. The path from the proxy to the final server passes through three such nodes. The final segment of the path, from the proxy to the server, is explicitly labeled "Cleartext".

**Figure 8.36** ♦ Providing anonymity and privacy with a proxy

an HTTP request for a page at the desired site. When the proxy receives the SSL-encrypted HTTP request, it decrypts the request and forwards the cleartext HTTP request to the Web site. The Web site then responds to the proxy, which in turn forwards the response to you over SSL. Because the Web site only sees the IP address of the proxy, and not of your client's address, you are indeed obtaining anonymous access to the Web site. And because all traffic between you and the proxy is encrypted, your local ISP cannot invade your privacy by logging the site you visited or recording the data you are exchanging. Many companies today (such as proxify.com) make available such proxy services.

Of course, in this solution, your proxy knows everything: It knows your IP address and the IP address of the site you're surfing; and it can see all the traffic in cleartext exchanged between you and the Web site. Such a solution, therefore, is only as good as the trustworthiness of the proxy. A more robust approach, taken by the TOR anonymizing and privacy service, is to route your traffic through a series of non-colluding proxy servers [TOR 2020]. In particular, TOR allows independent individuals to contribute proxies to its proxy pool. When a user connects to a server using TOR, TOR randomly chooses (from its proxy pool) a chain of three proxies and routes all traffic between client and server over the chain. In this manner, assuming the proxies do not collude, no one knows that communication took place between your IP address and the target Web site. Furthermore, although cleartext is sent between the last proxy and the server, the last proxy doesn't know what IP address is sending and receiving the cleartext.

Internal networks often have multiple application gateways, for example, gateways for Telnet, HTTP, FTP, and e-mail. In fact, an organization's mail server (see Section 2.3) and Web cache are application gateways.

Application gateways do not come without their disadvantages. First, a different application gateway is needed for each application. Second, there is a performance penalty to be paid, since all data will be relayed via the gateway. This becomes a concern particularly when multiple users or applications are using the same gateway machine. Finally, the client software must know how to contact the gateway when the user makes a request, and must know how to tell the application gateway what external server to connect to.

### 8.9.2 Intrusion Detection Systems

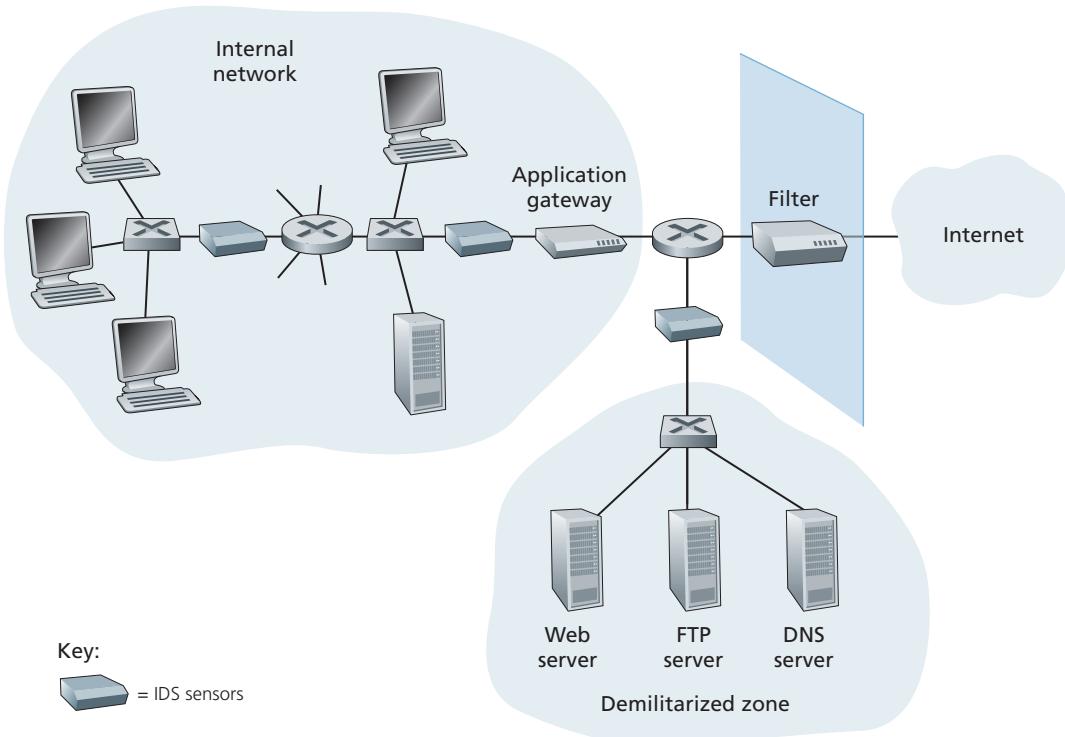
We've just seen that a packet filter (traditional and stateful) inspects IP, TCP, UDP, and ICMP header fields when deciding which packets to let pass through the firewall. However, to detect many attack types, we need to perform **deep packet inspection**, that is, look beyond the header fields and into the actual application data that the packets carry. As we saw in Section 8.9.1, application gateways often do deep packet inspection. But an application gateway only does this for a specific application.

Clearly, there is a niche for yet another device—a device that not only examines the headers of all packets passing through it (like a packet filter), but also performs deep packet inspection (unlike a packet filter). When such a device observes a suspicious packet, or a suspicious series of packets, it could prevent those packets from entering the organizational network. Or, because the activity is only deemed as suspicious, the device could let the packets pass, but send alerts to a network administrator, who can then take a closer look at the traffic and take appropriate actions. A device that generates alerts when it observes potentially malicious traffic is called an **intrusion detection system (IDS)**. A device that filters out suspicious traffic is called an **intrusion prevention system (IPS)**. In this section we study both systems—IDS and IPS—together, since the most interesting technical aspect of these systems is how they detect suspicious traffic (and not whether they send alerts or drop packets). We will henceforth collectively refer to IDS systems and IPS systems as IDS systems.

An IDS can be used to detect a wide range of attacks, including network mapping (emanating, for example, from nmap), port scans, TCP stack scans, DoS bandwidth-flooding attacks, worms and viruses, OS vulnerability attacks, and application vulnerability attacks. (See Section 1.6 for a survey of network attacks.) Today, thousands of organizations employ IDS systems. Many of these deployed systems are proprietary, marketed by Cisco, Check Point, and other security equipment vendors. But many of the deployed IDS systems are public-domain systems, such as the immensely popular Snort IDS system (which we'll discuss shortly).

An organization may deploy one or more IDS sensors in its organizational network. Figure 8.37 shows an organization that has three IDS sensors. When multiple sensors are deployed, they typically work in concert, sending information about suspicious traffic activity to a central IDS processor, which collects and integrates the information and sends alarms to network administrators when deemed appropriate. In Figure 8.37, the organization has partitioned its network into two regions: a high-security region, protected by a packet filter and an application gateway and monitored by IDS sensors; and a lower-security region—referred to as the **demilitarized zone (DMZ)**—which is protected only by the packet filter, but also monitored by IDS sensors. Note that the DMZ includes the organization's servers that need to communicate with the outside world, such as its public Web server and its authoritative DNS server.

You may be wondering at this stage, why multiple IDS sensors? Why not just place one IDS sensor just behind the packet filter (or even integrated with the packet filter) in Figure 8.37? We will soon see that an IDS not only needs to do deep packet inspection, but must also compare each passing packet with tens of thousands of “signatures”; this can be a significant amount of processing, particularly if the organization receives gigabits/sec of traffic from the Internet. By placing the IDS sensors further downstream, each sensor sees only a fraction of the organization's traffic, and can more easily keep up. Nevertheless, high-performance IDS and IPS systems are available today, and many organizations can actually get by with just one sensor located near its access router.



**Figure 8.37** ♦ An organization deploying a filter, an application gateway, and IDS sensors

IDS systems are broadly classified as either **signature-based systems** or **anomaly-based systems**. A signature-based IDS maintains an extensive database of attack signatures. Each signature is a set of rules pertaining to an intrusion activity. A signature may simply be a list of characteristics about a single packet (e.g., source and destination port numbers, protocol type, and a specific string of bits in the packet payload), or may relate to a series of packets. The signatures are normally created by skilled network security engineers who research known attacks. An organization's network administrator can customize the signatures or add its own to the database.

Operationally, a signature-based IDS sniffs every packet passing by it, comparing each sniffed packet with the signatures in its database. If a packet (or series of packets) matches a signature in the database, the IDS generates an alert. The alert could be sent to the network administrator in an e-mail message, could be sent to the network management system, or could simply be logged for future inspection.

Signature-based IDS systems, although widely deployed, have a number of limitations. Most importantly, they require previous knowledge of the attack to generate an accurate signature. In other words, a signature-based IDS is completely blind to new attacks that have yet to be recorded. Another disadvantage is that even if a signature is matched, it may not be the result of an attack, so that a false alarm is generated. Finally, because every packet must be compared with an extensive collection of signatures, the IDS can become overwhelmed with processing and actually fail to detect many malicious packets.

An anomaly-based IDS creates a traffic profile as it observes traffic in normal operation. It then looks for packet streams that are statistically unusual, for example, an inordinate percentage of ICMP packets or a sudden exponential growth in port scans and ping sweeps. The great thing about anomaly-based IDS systems is that they don't rely on previous knowledge about existing attacks—that is, they can potentially detect new, undocumented attacks. On the other hand, it is an extremely challenging problem to distinguish between normal traffic and statistically unusual traffic. To date, most IDS deployments are primarily signature-based, although some include some anomaly-based features.

### Snort

Snort is a public-domain, open source IDS with hundreds of thousands of existing deployments [Snort 2012; Koziol 2003]. It can run on Linux, UNIX, and Windows platforms. It uses the generic sniffing interface libpcap, which is also used by Wireshark and many other packet sniffers. It can easily handle 100 Mbps of traffic; for installations with gigabit/sec traffic rates, multiple Snort sensors may be needed.

To gain some insight into Snort, let's take a look at an example of a Snort signature:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any  
(msg:"ICMP PING NMAP"; dsiz: 0; itype: 8;)
```

This signature is matched by any ICMP packet that enters the organization's network (\$HOME\_NET) from the outside (\$EXTERNAL\_NET), is of type 8 (ICMP ping), and has an empty payload (dsiz = 0). Since nmap (see Section 1.6) generates ping packets with these specific characteristics, this signature is designed to detect nmap ping sweeps. When a packet matches this signature, Snort generates an alert that includes the message "ICMP PING NMAP".

Perhaps what is most impressive about Snort is the vast community of users and security experts that maintain its signature database. Typically within a few hours of a new attack, the Snort community writes and releases an attack signature, which is then downloaded by the hundreds of thousands of Snort deployments distributed around the world. Moreover, using the Snort signature syntax, network administrators can tailor the signatures to their own organization's needs by either modifying existing signatures or creating entirely new ones.

## 8.10 Summary

In this chapter, we've examined the various mechanisms that our secret lovers, Bob and Alice, can use to communicate securely. We've seen that Bob and Alice are interested in confidentiality (so they alone are able to understand the contents of a transmitted message), end-point authentication (so they are sure that they are talking with each other), and message integrity (so they are sure that their messages are not altered in transit). Of course, the need for secure communication is not confined to secret lovers. Indeed, we saw in Sections 8.5 through 8.8 that security can be used in various layers in a network architecture to protect against bad guys who have a large arsenal of possible attacks at hand.

The first part of this chapter presented various principles underlying secure communication. In Section 8.2, we covered cryptographic techniques for encrypting and decrypting data, including symmetric key cryptography and public key cryptography. DES and RSA were examined as specific case studies of these two major classes of cryptographic techniques in use in today's networks.

In Section 8.3, we examined two approaches for providing message integrity: message authentication codes (MACs) and digital signatures. The two approaches have a number of parallels. Both use cryptographic hash functions and both techniques enable us to verify the source of the message as well as the integrity of the message itself. One important difference is that MACs do not rely on encryption whereas digital signatures require a public key infrastructure. Both techniques are extensively used in practice, as we saw in Sections 8.5 through 8.8. Furthermore, digital signatures are used to create digital certificates, which are important for verifying the validity of public keys. In Section 8.4, we examined endpoint authentication and introduced nonces to defend against the replay attack.

In Sections 8.5 through 8.8 we examined several security networking protocols that enjoy extensive use in practice. We saw that symmetric key cryptography is at the core of PGP, SSL, IPsec, and wireless security. We saw that public key cryptography is crucial for both PGP and TLS. We saw that PGP uses digital signatures for message integrity, whereas TLS and IPsec use MACs. We also explored security in wireless networks, including WiFi networks and 4G/5G cellular networks. Having now an understanding of the basic principles of cryptography, and having studied how these principles are actually used, you are now in position to design your own secure network protocols!

Armed with the techniques covered in Sections 8.2 through 8.8, Bob and Alice can communicate securely. But confidentiality is only a small part of the network security picture. As we learned in Section 8.9, increasingly, the focus in network security has been on securing the network infrastructure against a potential onslaught by the bad guys. In the latter part of this chapter, we thus covered firewalls and IDS systems which inspect packets entering and leaving an organization's network.

## Homework Problems and Questions

---

### Chapter 8 Review Problems

#### SECTION 8.1

- R1. What are the differences between message confidentiality and message integrity? Can you have confidentiality without integrity? Can you have integrity without confidentiality? Justify your answer.
- R2. Internet entities (routers, switches, DNS servers, Web servers, user end systems, and so on) often need to communicate securely. Give three specific example pairs of Internet entities that may want secure communication.

#### SECTION 8.2

- R3. From a service perspective, what is an important difference between a symmetric-key system and a public-key system?
- R4. Suppose that an intruder has an encrypted message as well as the decrypted version of that message. Can the intruder mount a ciphertext-only attack, a known-plaintext attack, or a chosen-plaintext attack?
- R5. Consider an 8-bit block cipher. How many possible input blocks does this cipher have? How many possible mappings are there? If we view each mapping as a key, then how many possible keys does this cipher have?
- R6. Suppose  $N$  people want to communicate with each of  $N - 1$  other people using symmetric key encryption. All communication between any two people,  $i$  and  $j$ , is visible to all other people in this group of  $N$ , and no other person in this group should be able to decode their communication. How many keys are required in the system as a whole? Now suppose that public key encryption is used. How many keys are required in this case?
- R7. Suppose  $n = 10,000$ ,  $a = 10,023$ , and  $b = 10,004$ . Use an identity of modular arithmetic to calculate in your head  $(a \cdot b) \bmod n$ .
- R8. Suppose you want to encrypt the message 10101111 by encrypting the decimal number that corresponds to the message. What is the decimal number?

#### SECTIONS 8.3–8.4

- R9. In what way does a hash provide a better message integrity check than a checksum (such as the Internet checksum)?
- R10. Can you “decrypt” a hash of a message to get the original message? Explain your answer.

- R11. Consider a variation of the MAC algorithm (Figure 8.9) where the sender sends  $(m, H(m) + s)$ , where  $H(m) + s$  is the concatenation of  $H(m)$  and  $s$ . Is this variation flawed? Why or why not?
- R12. What does it mean for a signed document to be verifiable and nonforgeable?
- R13. In what way does the public-key encrypted message hash provide a better digital signature than the public-key encrypted message?
- R14. Suppose certifier.com creates a certificate for foo.com. Typically, the entire certificate would be encrypted with certifier.com's public key. True or false?
- R15. Suppose Alice has a message that she is ready to send to anyone who asks. Thousands of people want to obtain Alice's message, but each wants to be sure of the integrity of the message. In this context, do you think a MAC-based or a digital-signature-based integrity scheme is more suitable? Why?
- R16. What is the purpose of a nonce in an end-point authentication protocol?
- R17. What does it mean to say that a nonce is a once-in-a-lifetime value? In whose lifetime?
- R18. Is the message integrity scheme based on HMAC susceptible to playback attacks? If so, how can a nonce be incorporated into the scheme to remove this susceptibility?

#### SECTIONS 8.5–8.8

- R19. Suppose that Bob receives a PGP message from Alice. How does Bob know for sure that Alice created the message (rather than, say, Trudy)? Does PGP use a MAC for message integrity?
- R20. In the TLS record, there is a field for TLS sequence numbers. True or false?
- R21. What is the purpose of the random nonces in the TLS handshake?
- R22. Suppose an TLS session employs a block cipher with CBC. True or false: The server sends to the client the IV in the clear.
- R23. Suppose Bob initiates a TCP connection to Trudy who is pretending to be Alice. During the handshake, Trudy sends Bob Alice's certificate. In what step of the TLS handshake algorithm will Bob discover that he is not communicating with Alice?
- R24. Consider sending a stream of packets from Host A to Host B using IPsec. Typically, a new SA will be established for each packet sent in the stream. True or false?
- R25. Suppose that TCP is being run over IPsec between headquarters and the branch office in Figure 8.28. If TCP retransmits the same packet, then the two corresponding packets sent by R1 packets will have the same sequence number in the ESP header. True or false?

- R26. An IKE SA and an IPsec SA are the same thing. True or false?
- R27. Consider WEP for 802.11. Suppose that the data is 10101100 and the key-stream is 1111000. What is the resulting ciphertext?

#### SECTION 8.9

- R28. Stateful packet filters maintain two data structures. Name them and briefly describe what they do.
- R29. Consider a traditional (stateless) packet filter. This packet filter may filter packets based on TCP flag bits as well as other header fields. True or false?
- R30. In a traditional packet filter, each interface can have its own access control list. True or false?
- R31. Why must an application gateway work in conjunction with a router filter to be effective?
- R32. Signature-based IDSs and IPSs inspect into the payloads of TCP and UDP segments. True or false?

### Problems

---

- P1. Using the monoalphabetic cipher in Figure 8.3, encode the message “This is an easy problem.” Decode the message “rmijj'u uamu xyj.”
- P2. Show that Trudy’s known-plaintext attack, in which she knows the (ciphertext, plaintext) translation pairs for seven letters, reduces the number of possible substitutions to be checked in the example in Section 8.2.1 by approximately  $10^9$ .
- P3. Consider the polyalphabetic system shown in Figure 8.4. Will a chosen-plaintext attack that is able to get the plaintext encoding of the message “The quick brown fox jumps over the lazy dog.” be sufficient to decode all messages? Why or why not?
- P4. Consider the block cipher in Figure 8.5. Suppose that each block cipher  $T_i$  simply reverses the order of the eight input bits (so that, for example, 11110000 becomes 00001111). Further suppose that the 64-bit scrambler does not modify any bits (so that the output value of the  $m$ th bit is equal to the input value of the  $m$ th bit). (a) With  $n = 3$  and the original 64-bit input equal to 10100000 repeated eight times, what is the value of the output? (b) Repeat part (a) but now change the last bit of the original 64-bit input from a 0 to a 1. (c) Repeat parts (a) and (b) but now suppose that the 64-bit scrambler inverses the order of the 64 bits.
- P5. Consider the block cipher in Figure 8.5. For a given “key” Alice and Bob would need to keep eight tables, each 8 bits by 8 bits. For Alice (or Bob) to store all eight tables, how many bits of storage are necessary? How does this number compare with the number of bits required for a full-table 64-bit block cipher?

- P6. Consider the 3-bit block cipher in Table 8.1. Suppose the plaintext is 100100100. (a) Initially assume that CBC is not used. What is the resulting ciphertext? (b) Suppose Trudy sniffs the ciphertext. Assuming she knows that a 3-bit block cipher without CBC is being employed (but doesn't know the specific cipher), what can she surmise? (c) Now suppose that CBC is used with IV = 111. What is the resulting ciphertext?
- P7. (a) Using RSA, choose  $p = 3$  and  $q = 11$ , and encode the word “dog” by encrypting each letter separately. Apply the decryption algorithm to the encrypted version to recover the original plaintext message. (b) Repeat part (a) but now encrypt “dog” as one message  $m$ .
- P8. Consider RSA with  $p = 5$  and  $q = 11$ .
- What are  $n$  and  $z$ ?
  - Let  $e$  be 3. Why is this an acceptable choice for  $e$ ?
  - Find  $d$  such that  $de = 1 \pmod{z}$  and  $d < 160$ .
  - Encrypt the message  $m = 8$  using the key  $(n, e)$ . Let  $c$  denote the corresponding ciphertext. Show all work. *Hint:* To simplify the calculations, use the fact:

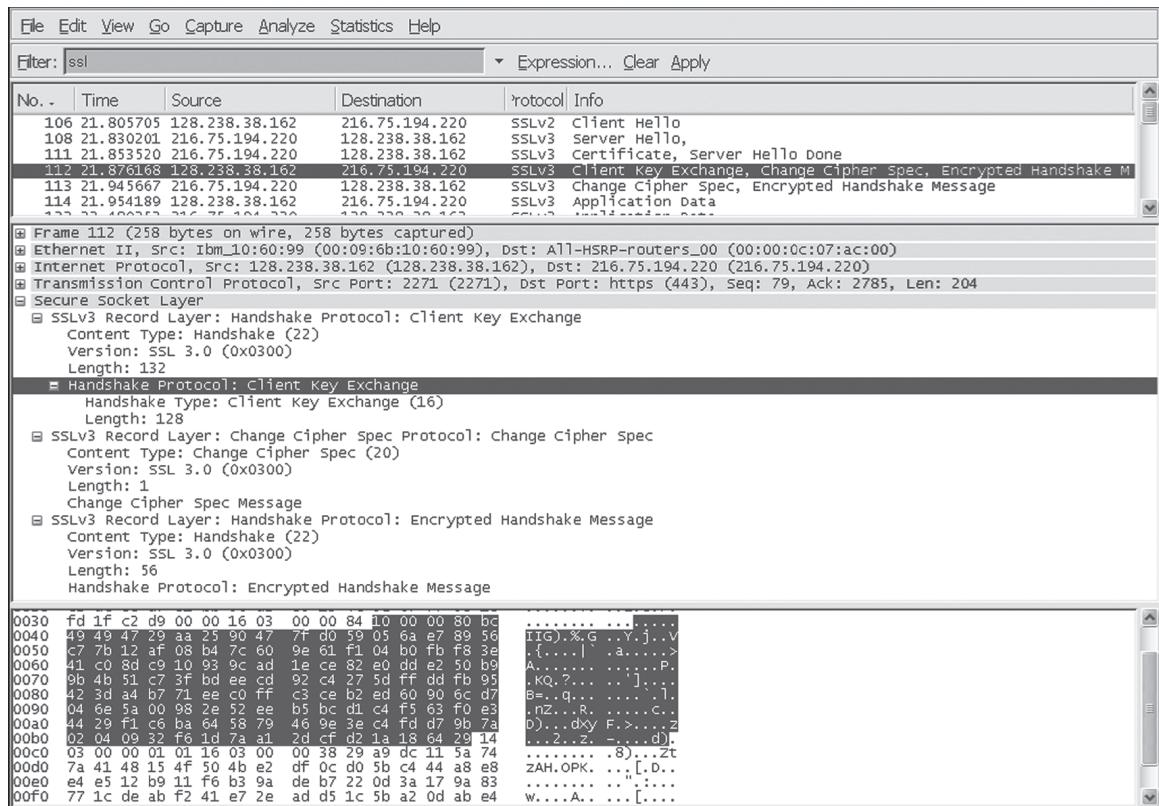
$$[(a \pmod{n}) \cdot (b \pmod{n})] \pmod{n} = (a \cdot b) \pmod{n}$$

- P9. In this problem, we explore the Diffie-Hellman (DH) public-key encryption algorithm, which allows two entities to agree on a shared key. The DH algorithm makes use of a large prime number  $p$  and another large number  $g$  less than  $p$ . Both  $p$  and  $g$  are made public (so that an attacker would know them). In DH, Alice and Bob each independently choose secret keys,  $S_A$  and  $S_B$ , respectively. Alice then computes her public key,  $T_A$ , by raising  $g$  to  $S_A$  and then taking mod  $p$ . Bob similarly computes his own public key  $T_B$  by raising  $g$  to  $S_B$  and then taking mod  $p$ . Alice and Bob then exchange their public keys over the Internet. Alice then calculates the shared secret key  $S$  by raising  $T_B$  to  $S_A$  and then taking mod  $p$ . Similarly, Bob calculates the shared key  $S'$  by raising  $T_A$  to  $S_B$  and then taking mod  $p$ .
- Prove that, in general, Alice and Bob obtain the same symmetric key, that is, prove  $S = S'$ .
  - With  $p = 11$  and  $g = 2$ , suppose Alice and Bob choose private keys  $S_A = 5$  and  $S_B = 12$ , respectively. Calculate Alice's and Bob's public keys,  $T_A$  and  $T_B$ . Show all work.
  - Following up on part (b), now calculate  $S$  as the shared symmetric key. Show all work.
  - Provide a timing diagram that shows how Diffie-Hellman can be attacked by a man-in-the-middle. The timing diagram should have three vertical lines, one for Alice, one for Bob, and one for the attacker Trudy.

- P10. Suppose Alice wants to communicate with Bob using symmetric key cryptography using a session key  $K_S$ . In Section 8.2, we learned how public-key cryptography can be used to distribute the session key from Alice to Bob. In this problem, we explore how the session key can be distributed—without public key cryptography—using a key distribution center (KDC). The KDC is a server that shares a unique secret symmetric key with each registered user. For Alice and Bob, denote these keys by  $K_{A-KDC}$  and  $K_{B-KDC}$ . Design a scheme that uses the KDC to distribute  $K_S$  to Alice and Bob. Your scheme should use three messages to distribute the session key: a message from Alice to the KDC; a message from the KDC to Alice; and finally a message from Alice to Bob. The first message is  $K_{A-KDC}(A, B)$ . Using the notation,  $K_{A-KDC}$ ,  $K_{B-KDC}$ ,  $S$ ,  $A$ , and  $B$  answer the following questions.
- What is the second message?
  - What is the third message?
- P11. Compute a third message, different from the two messages in Figure 8.8, that has the same checksum as the messages in Figure 8.8.
- P12. Suppose Alice and Bob share two secret keys: an authentication key  $S_1$  and a symmetric encryption key  $S_2$ . Augment Figure 8.9 so that both integrity and confidentiality are provided.
- P13. In the BitTorrent P2P file distribution protocol (see Chapter 2), the seed breaks the file into blocks, and the peers redistribute the blocks to each other. Without any protection, an attacker can easily wreak havoc in a torrent by masquerading as a benevolent peer and sending bogus blocks to a small subset of peers in the torrent. These unsuspecting peers then redistribute the bogus blocks to other peers, which in turn redistribute the bogus blocks to even more peers. Thus, it is critical for BitTorrent to have a mechanism that allows a peer to verify the integrity of a block, so that it doesn't redistribute bogus blocks. Assume that when a peer joins a torrent, it initially gets a `.torrent` file from a *fully* trusted source. Describe a simple scheme that allows peers to verify the integrity of blocks.
- P14. The OSPF routing protocol uses a MAC rather than digital signatures to provide message integrity. Why do you think a MAC was chosen over digital signatures?
- P15. Consider our authentication protocol in Figure 8.18 in which Alice authenticates herself to Bob, which we saw works well (i.e., we found no flaws in it). Now suppose that while Alice is authenticating herself to Bob, Bob must authenticate himself to Alice. Give a scenario by which Trudy, pretending to be Alice, can now authenticate herself to Bob as Alice. (*Hint:* Consider that the sequence of operations of the protocol, one with Trudy initiating and one with Bob initiating, can be arbitrarily interleaved. Pay particular attention to the fact that both Bob and Alice will use a nonce, and that if care is not taken, the same nonce can be used maliciously.)

- P16. A natural question is whether we can use a nonce and public key cryptography to solve the end-point authentication problem in Section 8.4. Consider the following natural protocol: (1) Alice sends the message “I am Alice” to Bob. (2) Bob chooses a nonce,  $R$ , and sends it to Alice. (3) Alice uses her *private* key to encrypt the nonce and sends the resulting value to Bob. (4) Bob applies Alice’s public key to the received message. Thus, Bob computes  $R$  and authenticates Alice.
- Diagram this protocol, using the notation for public and private keys employed in the textbook.
  - Suppose that certificates are not used. Describe how Trudy can become a “woman-in-the-middle” by intercepting Alice’s messages and then pretending to be Alice to Bob.
- P17. Figure 8.21 shows the operations that Alice must perform with PGP to provide confidentiality, authentication, and integrity. Diagram the corresponding operations that Bob must perform on the package received from Alice.
- P18. Suppose Alice wants to send an e-mail to Bob. Bob has a public-private key pair  $(K_B^+, K_B^-)$ , and Alice has Bob’s certificate. But Alice does not have a public, private key pair. Alice and Bob (and the entire world) share the same hash function  $H(\cdot)$ .
- In this situation, is it possible to design a scheme so that Bob can verify that Alice created the message? If so, show how with a block diagram for Alice and Bob.
  - Is it possible to design a scheme that provides confidentiality for sending the message from Alice to Bob? If so, show how with a block diagram for Alice and Bob.
- P19. Consider the Wireshark output below for a portion of an SSL session.
- Is Wireshark packet 112 sent by the client or server?
  - What is the server’s IP address and port number?
  - Assuming no loss and no retransmissions, what will be the sequence number of the next TCP segment sent by the client?
  - How many SSL records does Wireshark packet 112 contain?
  - Does packet 112 contain a Master Secret or an Encrypted Master Secret or neither?
  - Assuming that the handshake type field is 1 byte and each length field is 3 bytes, what are the values of the first and last bytes of the Master Secret (or Encrypted Master Secret)?
  - The client encrypted handshake message takes into account how many SSL records?
  - The server encrypted handshake message takes into account how many SSL records?

- P20. In Section 8.6.1, it is shown that without sequence numbers, Trudy (a woman-in-the middle) can wreak havoc in a TLS session by interchanging TCP segments. Can Trudy do something similar by deleting a TCP segment? What does she need to do to succeed at the deletion attack? What effect will it have?



(Wireshark screenshot reprinted by permission of the Wireshark Foundation.)

- P21. Suppose Alice and Bob are communicating over a TLS session. Suppose an attacker, who does not have any of the shared keys, inserts a bogus TCP segment into a packet stream with correct TCP checksum and sequence numbers (and correct IP addresses and port numbers). Will TLS at the receiving side accept the bogus packet and pass the payload to the receiving application? Why or why not?
- P22. The following true/false questions pertain to Figure 8.28.
- When a host in 172.16.1/24 sends a datagram to an Amazon.com server, the router R1 will encrypt the datagram using IPsec.

- b. When a host in 172.16.1/24 sends a datagram to a host in 172.16.2/24, the router R1 will change the source and destination address of the IP datagram.
  - c. Suppose a host in 172.16.1/24 initiates a TCP connection to a Web server in 172.16.2/24. As part of this connection, all datagrams sent by R1 will have protocol number 50 in the left-most IPv4 header field.
  - d. Consider sending a TCP segment from a host in 172.16.1/24 to a host in 172.16.2/24. Suppose the acknowledgment for this segment gets lost, so that TCP resends the segment. Because IPsec uses sequence numbers, R1 will not resend the TCP segment.
- P23. Consider the example in Figure 8.28. Suppose Trudy is a woman-in-the-middle, who can insert datagrams into the stream of datagrams going from R1 and R2. As part of a replay attack, Trudy sends a duplicate copy of one of the datagrams sent from R1 to R2. Will R2 decrypt the duplicate datagram and forward it into the branch-office network? If not, describe in detail how R2 detects the duplicate datagram.
- P24. Provide a filter table and a connection table for a stateful firewall that is as restrictive as possible but accomplishes the following:
- a. Allows all internal users to establish Telnet sessions with external hosts.
  - b. Allows external users to surf the company Web site at 222.22.0.12.
  - c. But otherwise blocks all inbound and outbound traffic.
- The internal network is 222.22/16. In your solution, suppose that the connection table is currently caching three connections, all from inside to outside. You'll need to invent appropriate IP addresses and port numbers.
- P25. Suppose Alice wants to visit the Web site activist.com using a TOR-like service. This service uses two non-colluding proxy servers, Proxy1 and Proxy2. Alice first obtains the certificates (each containing a public key) for Proxy1 and Proxy2 from some central server. Denote  $K_1^+(\ )$ ,  $K_2^+(\ )$ ,  $K_1^-(\ )$ , and  $K_2^-(\ )$  for the encryption/decryption with public and private RSA keys.
- a. Using a timing diagram, provide a protocol (as simple as possible) that enables Alice to establish a shared session key  $S_1$  with Proxy1. Denote  $S_1(m)$  for encryption/decryption of data  $m$  with the shared key  $S_1$ .
  - b. Using a timing diagram, provide a protocol (as simple as possible) that allows Alice to establish a shared session key  $S_2$  with Proxy2 *without revealing her IP address to Proxy2*.
  - c. Assume now that shared keys  $S_1$  and  $S_2$  are now established. Using a timing diagram, provide a protocol (as simple as possible and *not using public-key cryptography*) that allows Alice to request an html page from activist.com *without revealing her IP address to Proxy2 and without revealing to Proxy1 which site she is visiting*. Your diagram should end with an HTTP request arriving at activist.com.

## Wireshark Lab: SSL

---

In this lab (available from the book Web site), we investigate the Secure Sockets Layer (SSL) protocol. Recall from Section 8.6 that SSL is used for securing a TCP connection, and that it is extensively used in practice for secure Internet transactions. In this lab, we will focus on the SSL records sent over the TCP connection. We will attempt to delineate and classify each of the records, with a goal of understanding the why and how for each record. We investigate the various SSL record types as well as the fields in the SSL messages. We do so by analyzing a trace of the SSL records sent between your host and an e-commerce server.

## IPsec Lab

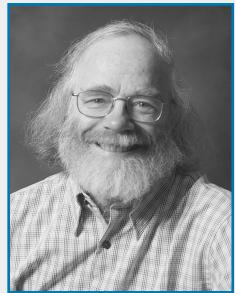
---

In this lab (available from the book Web site), we will explore how to create IPsec SAs between linux boxes. You can do the first part of the lab with two ordinary linux boxes, each with one Ethernet adapter. But for the second part of the lab, you will need four linux boxes, two of which having two Ethernet adapters. In the second half of the lab, you will create IPsec SAs using the ESP protocol in the tunnel mode. You will do this by first manually creating the SAs, and then by having IKE create the SAs.

## AN INTERVIEW WITH...

### Steven M. Bellovin

Steven M. Bellovin joined the faculty at Columbia University after many years at the Network Services Research Lab at AT&T Labs Research in Florham Park, New Jersey. His focus is on networks, security, and why the two are incompatible. In 1995, he was awarded the Usenix Lifetime Achievement Award for his work in the creation of Usenet, the first newsgroup exchange network that linked two or more computers and allowed users to share information and join in discussions. Steve is also an elected member of the National Academy of Engineering. He received his BA from Columbia University and his PhD from the University of North Carolina at Chapel Hill.



Courtesy of Steven Bellovin

#### What led you to specialize in the networking security area?

This is going to sound odd, but the answer is simple: It was fun. My background was in systems programming and systems administration, which leads fairly naturally to security. And I've always been interested in communications, ranging back to part-time systems programming jobs when I was in college.

My work on security continues to be motivated by two things—a desire to keep computers useful, which means that their function can't be corrupted by attackers, and a desire to protect privacy.

#### What was your vision for Usenet at the time that you were developing it? And now?

We originally viewed it as a way to talk about computer science and computer programming around the country, with a lot of local use for administrative matters, for-sale ads, and so on. In fact, my original prediction was one to two messages per day, from 50 to 100 sites at the most—ever. However, the real growth was in people-related topics, including—but not limited to—human interactions with computers. My favorite newsgroups, over the years, have been things like *rec.woodworking*, as well as *sci.crypt*.

To some extent, netnews has been displaced by the Web. Were I to start designing it today, it would look very different. But it still excels as a way to reach a very broad audience that is interested in the topic, without having to rely on particular Web sites.

#### Has anyone inspired you professionally? In what ways?

Professor Fred Brooks—the founder and original chair of the computer science department at the University of North Carolina at Chapel Hill, the manager of the team that developed the IBM S/360 and OS/360, and the author of *The Mythical Man-Month*—was a tremendous

influence on my career. More than anything else, he taught outlook and trade-offs—how to look at problems in the context of the real world (and how much messier the real world is than a theorist would like), and how to balance competing interests in designing a solution. Most computer work is engineering—the art of making the right trade-offs to satisfy many contradictory objectives.

**What is your vision for the future of networking and security?**

Thus far, much of the security we have has come from isolation. A firewall, for example, works by cutting off access to certain machines and services. But we're in an era of increasing connectivity—it's gotten harder to isolate things. Worse yet, our production systems require far more separate pieces, interconnected by networks. Securing all that is one of our biggest challenges.

**What would you say have been the greatest advances in security? How much further do we have to go?**

At least scientifically, we know how to do cryptography. That's been a big help. But most security problems are due to buggy code, and that's a much harder problem. In fact, it's the oldest unsolved problem in computer science, and I think it will remain that way. The challenge is figuring out how to secure systems when we have to build them out of insecure components. We can already do that for reliability in the face of hardware failures; can we do the same for security?

**Do you have any advice for students about the Internet and networking security?**

Learning the mechanisms is the easy part. Learning how to “think paranoid” is harder. You have to remember that probability distributions don't apply—the attackers can and will find improbable conditions. And the details matter—a lot.

## Reti multimediali

In ogni parte del mondo Internet viene usata per guardare film e televisione on demand. Le aziende di distribuzione di film e televisione su Internet, come Netflix e Hulu nel Nord America e Youku e Kankan in Cina, sono diventate molto popolari. La gente non solo guarda video su Internet, ma usa anche siti come YouTube per caricare e distribuire i contenuti che genera, diventando sia produttrice sia consumatrice di contenuti video su Internet. Inoltre, le applicazioni di rete come Skype, Google Talk e WeChat (molto popolare in Cina) permettono non solo di telefonarsi su Internet, ma anche di arricchire le chiamate con video e conferenze a più persone. Possiamo tranquillamente immaginare che alla fine di questo decennio tutta la distribuzione video e le telefonate avverranno su Internet, spesso tramite dispositivi wireless connessi tramite reti di accesso 4G e WiFi. La telefonia e la televisione tradizionali diventeranno presto obsolete.

Iniziamo questo capitolo con una tassonomia delle applicazioni multimediali (Paragrafo 9.1), che possono essere classificate come *audio/video streaming registrati*, *audio/video streaming in diretta* o *audio/video over IP in tempo reale interattivo*. Vedremo inoltre che ciascuna di queste classi di applicazioni ha un differente insieme di requisiti di servizio in rete che differisce in modo significativo da quello delle tradizionali applicazioni elastiche quali e-mail, navigazione sul Web e login remoto. Nel Paragrafo 9.2 esamineremo nel dettaglio lo streaming<sup>1</sup> video di contenuti pre-registrati. Esploreremo molti dei principi che stanno alla base dello streaming video, tra i quali la gestione del buffer lato client, il prefetching e l'adattamento della qualità del video alla banda disponibile.

<sup>1</sup> Cioè la fruizione di contenuti spesso multimediali, vincolati dal punto di vista temporale e che vengono consumati senza prima essere memorizzati dal fruitore in un file (*N.d.R.*).

Tratteremo anche le reti di distribuzione di contenuti (CDN, *content distribution network*), usate in modo estensivo dai principali sistemi di streaming video. Nel Paragrafo 9.3 tratteremo le telefonate voce e video che, al contrario delle applicazioni elastiche, sono molto sensibili al ritardo end-to-end, ma possono tollerare perdite occasionali di dati. Vedremo come tecniche quali la riproduzione adattativa, la correzione degli errori di inoltro e l'occultamento degli errori possano mitigare la perdita dei pacchetti e i ritardi indotti dalla rete. Come caso di studio esamineremo Skype. Nel Paragrafo 9.4 studieremo RTP e SIP, due protocolli molto diffusi per le applicazioni in tempo reale di audio e video conferenza. Nel Paragrafo 9.5 studieremo i meccanismi che possono essere usati nella rete per distinguere una classe di traffico, per esempio quella delle applicazioni sensibili al ritardo come la telefonia voce, da un'altra, per esempio come quella delle applicazioni elastiche quali la navigazione su pagine web, e per fornire servizi differenziati a più classi di traffico.

## 9.1 Applicazioni multimediali di rete

Definiamo applicazione di rete multimediale qualsiasi applicazione che impieghi audio o video. In questo paragrafo forniremo una tassonomia delle applicazioni multimediali. Vedremo che ogni classe di applicazioni in tale tassonomia ha requisiti di servizio e problematiche di progettazione proprie. Tuttavia, prima di entrare nel dettaglio della discussione sulle applicazioni multimediali in Internet, è utile vedere le caratteristiche intrinseche dei contenuti audio e video.

### 9.1.1 Proprietà del video

Forse la caratteristica più saliente del video è l'elevato tasso con cui è necessario inviare i bit sulla rete (**bit rate**).<sup>2</sup> I video distribuiti in Internet variano da 100 kbps per le conferenze video a bassa qualità a oltre 3 Mbps per i film in streaming ad alta definizione. Per confrontare le richieste di banda dei contenuti video con quelle di altre applicazioni in Internet, consideriamo tre utenti che stiano usando tre diverse applicazioni per Internet. Il primo, Frank, sta velocemente guardando le foto pubblicate dai suoi amici sulle loro pagine Facebook. Assumiamo che Frank guardi una nuova foto ogni 10 secondi e che le foto siano mediamente grandi 200 Kbyte. Come al solito faremo l'assunzione semplificativa che 1 Kbyte sia uguale a 8000 bit. Il secondo utente, Martha, sta ascoltando musica in streaming su Internet, usando un servizio quale Spotify. Assumiamo che Martha ascolti molte canzoni MP3, una dopo l'altra, ognuna codificata a 128 kbps. Il terzo utente, Victor, sta guardando un video codificato a 2 Mbps. Infine, supponiamo che tutti abbiano una sessione di uguale durata e pari a

---

<sup>2</sup> Bit rate (tasso dei bit) a volte viene indicato anche come l'ampiezza di banda necessaria per trasmettere il contenuto (*N.d.R.*).

**Tabella 9.1** Confronto dei requisiti di banda di tre applicazioni Internet.

|                | Bit rate | Byte trasferiti in 67 min |
|----------------|----------|---------------------------|
| Facebook Frank | 160 kbps | 80 Mb                     |
| Martha Music   | 128 kbps | 64 Mb                     |
| Victor Video   | 2 Mbps   | 1 Gb                      |

4000 secondi, circa 67 minuti. Un confronto tra i bit rate e il numero totale di byte trasferiti per i tre utenti è riportato nella Tabella 9.1. Vediamo che lo streaming video è quello che consuma molta più banda, avendo un bit rate di oltre 10 volte superiore a quello delle applicazioni Facebook e di streaming musicale. Quindi, nella progettazione di applicazioni video, la prima cosa da tenere presente sono i requisiti di bit rate elevati. Data la diffusione dei video e l'alto bit rate richiesto, forse non risulta sorprendente la stima di Cisco [Cisco 2015] che sostiene che i video sia in streaming sia registrati (cioè trasferiti come file per essere fruiti localmente a destinazione) rappresenteranno, nel 2019, circa il 80% del traffico totale di Internet.

Un'altra caratteristica del video è la sua possibilità di essere compresso, raggiungendo un compromesso tra qualità del video e bit rate. Un video è una sequenza di immagini, visualizzate tipicamente a tasso costante di, per esempio, 24 o 30 immagini al secondo. Un'immagine non compressa e codificata digitalmente consiste di un array di pixel ognuno dei quali codificato con un numero di bit per rappresentare luminanza e crominanza. Nella **compressione video** vengono sfruttati due tipi di ridondanza. La *ridondanza spaziale* è quella all'interno di una data immagine. Intuitivamente, un'immagine formata per lo più dallo stesso colore omogeneo ha un alto grado di ridondanza e può essere compressa in modo efficace senza sacrificarne in modo significativo la qualità. La *ridondanza temporale* è data dalla ripetizione della stessa immagine in due tempi successivi. Se, per esempio, un'immagine e quella subito dopo sono esattamente uguali, non c'è alcuna ragione per ricodificare la seconda, mentre è molto più efficiente indicare semplicemente durante la codifica che l'immagine successiva sarà uguale a quella corrente. Gli algoritmi di compressione oggi disponibili sono in grado di comprimere un video a qualsiasi bit rate si desideri. Ovviamente, più alto è il bit rate, migliore è la qualità dell'immagine e l'esperienza visiva globale dell'utente.

La compressione può essere usata anche per creare **versioni multiple** dello stesso video, a livelli di qualità diversi. Per esempio, si può usare la compressione per creare tre versioni dello stesso video, a 300 kbps, 1 Mbps e 3 Mbps. Gli utenti possono decidere quale versione vogliono guardare in funzione della larghezza di banda disponibile. Gli utenti con connessioni Internet ad alta velocità potrebbero scegliere la versione a 3 Mbps; gli utenti che guardano i video tramite 3G su uno smartphone potrebbero scegliere quella a 300 kbps. In modo simile, il video in un'applicazione di videoconferenza può venire compresso al volo per fornire la miglior qualità video data la banda end-to-end disponibile tra i partecipanti alla conferenza.

### 9.1.2 Proprietà dell'audio

L'audio digitale, come la voce e la musica, benché abbia esigenze di larghezza di banda meno stringenti di quelle del video, ha tuttavia proprietà peculiari di cui bisogna tener conto quando si progettano applicazioni di rete multimediali. Per capirle, vediamo come il segnale audio analogico, che varia con continuità e che deriva da un discorso o da della musica, venga normalmente convertito in un segnale digitale secondo il seguente schema.

- Per prima cosa si procede al campionamento del segnale analogico a una frequenza fissata, per esempio di 8000 campioni al secondo. Il valore di ciascun campione è un numero reale arbitrario.
- Si procede poi con l'operazione di **quantizzazione**, durante la quale i campioni sono arrotondati a numeri interi in un intervallo finito di valori (chiamati abitualmente *valori di quantizzazione* e che sono di solito una potenza di due, per esempio 256).
- Tutti i valori di quantizzazione sono rappresentati dallo stesso numero di bit. Per esempio, se ci sono 256 valori di quantizzazione, tutti i valori (e quindi tutti i campioni audio) sono rappresentati da 1 byte. Le rappresentazioni in bit di tutti i campioni vengono poi concatenate a formare la rappresentazione digitale del segnale. Per esempio, se un segnale audio analogico è campionato 8000 volte al secondo e ciascun campione è quantizzato e rappresentato con 8 bit, allora il segnale digitale che ne risulta avrà un tasso di 64.000 bps. Questo segnale digitale può essere riconvertito (cioè decodificato) in un segnale analogico per la riproduzione che però, di solito, è diverso da quello originale e la qualità può degradare molto, per esempio a causa della perdita delle componenti ad alta frequenza. Incrementando il tasso di campionamento e il numero di valori di quantizzazione, il segnale decodificato può approssimarsi meglio al segnale analogico originale. Quindi, vi è un chiaro compromesso fra la qualità del segnale decodificato e i requisiti di memoria e larghezza di banda del segnale digitale.

La tecnica che abbiamo appena descritto è detta **modulazione a codifica di impulso** (PCM, *pulse code modulation*) spesso utilizzata per la codifica della voce al tasso di 8000 campioni/secondo e 8 bit per campione, per ottenere un bit rate di 64 kbps. Anche i CD audio utilizzano PCM, ma con un tasso di 44.100 campioni/secondo e 16 bit per campione; questo produce un bit rate di 705,6 kbps per l'audio mono e di 1,411 Mbps per quello stereo.

Tuttavia, le codifiche PCM per fonia vocale e musica sono raramente utilizzate su Internet. Dove invece, come nel caso del video, vengono impiegate tecniche di compressione per ridurre il bit rate del flusso di dati (che prende anche il nome di stream). La voce umana può essere compressa a meno di 10 kbps ed essere ancora intelligibile. Una nota tecnica di compressione per musica stereo di qualità prossima a quella dei CD è **MPEG 1 layer 3**, più generalmente conosciuta come **MP3**. MP3 può produrre molti differenti bit rate; quello più comune per la musica a 128 kbps permette anche di ottenere una ridotta degradazione del suono. Uno standard collegato a MP3 è l'*ad-*

*vanced audio coding* (AAC) reso popolare da Apple. Come per il video, possono essere create più versioni di uno stream audio, a differenti bit rate.

Anche se i bit rate usati per l'audio sono generalmente molto più bassi di quelli per il video, gli utenti sono generalmente molto più sensibili a disturbi nel suono piuttosto che nel video. Si consideri, per esempio, una videoconferenza su Internet; se, ogni tanto, il segnale video viene perso per alcuni secondi si può procedere senza troppa frustrazione da parte degli utenti. Se, invece, il segnale audio viene perso di frequente, l'utente potrebbe voler interrompere la sessione.

### 9.1.3 Tipi di applicazioni multimediali

Internet supporta una gran varietà di interessanti applicazioni multimediali. In questo paragrafo considereremo tre classi: *streaming audio/video di contenuti registrati*, *conversazione voce/video su IP* e *streaming audio/video in tempo reale*. Tutte queste applicazioni hanno propri requisiti e problematiche di progettazione.

#### Streaming audio/video di contenuti registrati

Per mantenere la discussione su un piano concreto ci focalizzeremo sullo streaming video, che di solito combina componenti sia video che audio. Lo streaming del solo audio (come la musica) è molto simile, a parte il fatto che usa bit rate molto più bassi.

In questa classe di applicazioni i contenuti sono video, quali film, trasmissioni televisive, eventi sportivi o video generati dagli utenti, come quelli su YouTube, memorizzati su server a disposizione degli utenti su richiesta (*on demand*). Migliaia di siti forniscono oggi streaming di audio e video registrati, tra cui YouTube (Google), Netflix, Amazon e Hulu. Tre sono le caratteristiche fondamentali che contraddistinguono questa classe.

- *Streaming*. Nelle applicazioni per lo streaming di audio/video registrati, il client tipicamente inizia la riproduzione audio/video pochi secondi dopo aver iniziato a ricevere il file dal server. Questo significa che il client inizia la riproduzione di una parte del file audio/video prima di averlo interamente scaricato dal server. Tale tecnica, detta **streaming**, evita lo scaricamento dell'intero file e quindi di incorrere in potenziali, lunghi ritardi.
- *Interattività*. Il contenuto multimediale è registrato e archiviato sul server. Quindi, gli utenti possono usare le funzioni di pausa, riavvolgimento e avanzamento rapido. Per un utilizzo accettabile, il tempo trascorso dal momento in cui il client invia la richiesta a quello in cui viene visualizzato il contenuto dovrebbe essere dell'ordine di pochi secondi.
- *Riproduzione continua*. Quando la riproduzione inizia, dovrebbe procedere secondo i tempi di registrazione originali. Ciò impone che i dati debbano essere ricevuti dal client in tempo utile per la loro riproduzione, altrimenti l'utente potrebbe assistere a un blocco dell'inquadratura (frame freezing) quando aspetta dati in ritardo o il salto di alcuni fotogrammi (frame skipping).

La misura in assoluto più importante per lo streaming video è il throughput medio, il cui valore deve essere almeno pari a quello del bit rate del video per avere una riproduzione continua. Come vedremo nel Paragrafo 9.2, purché il throughput mediato su 5-10 secondi rimanga al di sopra del bit rate del video, è possibile ottenere riproduzione continua anche in presenza di fluttuazioni del throughput [Wang 2008].

Spesso il video è memorizzato e mandato in streaming da una CDN piuttosto che da un singolo data center. Esistono anche molte applicazioni di streaming P2P in cui il video è contenuto negli host degli utenti (peer) e può quindi arrivare da più peer sparsi per il globo. Vista la sua importanza, discuteremo in dettaglio lo streaming video nel Paragrafo 9.2, con particolare attenzione alla gestione del buffer, prefetching, adattamento della qualità alla banda e la distribuzione via CDN.

### **Conversazione audio e video su IP**

Questa classe di applicazioni viene comunemente definita **telefonia Internet** o **Voice-over-IP (VoIP)** in quanto, per l’utente, è simile al tradizionale servizio telefonico a commutazione di circuito. La conversazione video è simile, tranne per il fatto che include anche le immagini degli interlocutori oltre che le loro voci. Si possono fare conferenze audio/video con tre o più persone. Centinaia di milioni di persone utilizzano giornalmente i servizi di Skype, QQ e Google Talk per conferenze audio/video.

Nella nostra discussione riguardo i requisiti dei servizi applicativi nel Capitolo 2 (Figura 2.4) abbiamo identificato degli assi lungo i quali questi requisiti potevano essere classificati. Due di questi assi sono particolarmente importanti per le applicazioni di conversazione audio e video: la temporizzazione e la tolleranza alla perdite di dati. L’aspetto della temporizzazione è importante, in quanto queste applicazioni sono altamente **sensibili ai ritardi** (*delay-sensitive*). In una conversazione a cui partecipano due o più utenti il ritardo dal momento in cui un utente parla o si muove a quando l’azione di manifesta agli altri partecipanti dovrebbe essere inferiore a poche centinaia di millisecondi. Per la voce, ritardi inferiori a 150 millisecondi non sono percepiti dall’ascoltatore, fino a 400 millisecondi possono essere accettabili, ma se li superano possono risultare fastidiosi, o rendere completamente incomprensibile la conversazione.

D’altro canto tali applicazioni sono **tolleranti alle perdite** (*loss-tolerant*): perdite occasionali causano solo marginali interferenze nella riproduzione audio/video, che per di più possono essere parzialmente o completamente mascherate. Queste caratteristiche sono chiaramente diverse da quelle di applicazioni elastiche come Web, e-mail, social network e sessioni di lavoro remoto, per le quali i lunghi ritardi possono risultare fastidiosi, ma non particolarmente dannosi, mentre completezza e integrità dei dati trasferiti rivestono fondamentale importanza. Entreremo nei dettagli di questo tipo di applicazioni nel Paragrafo 9.3, con particolare attenzione riguardo alla riproduzione adattativa, la correzione degli errori di inoltro e la cancellazione degli errori per mitigare il ritardo e le perdite di pacchetti causati dalla rete.

### **Streaming audio/video in tempo reale (live)**

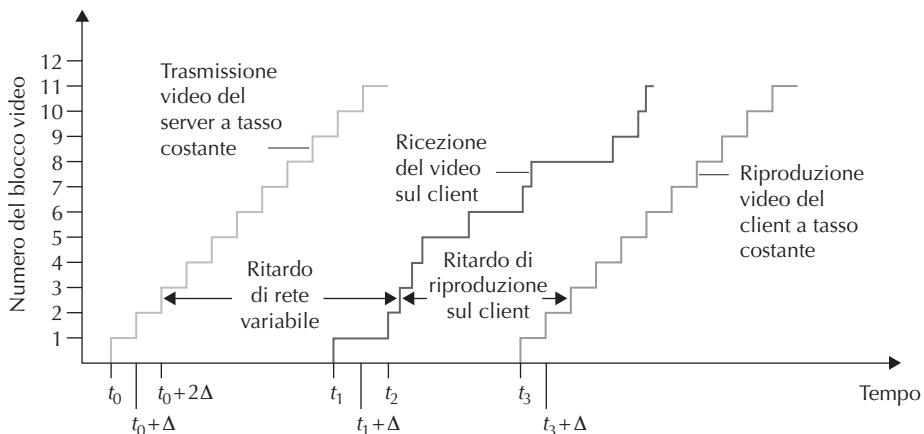
Questa terza classe di applicazioni è simile alle tradizionali trasmissioni radiotelevisive, a parte il fatto che avviene su Internet. Consente agli utenti di ricevere in diretta trasmissioni radio o televisive (come eventi sportivi o giornalistici) provenienti da ogni parte del mondo. Oggi ci sono migliaia di stazioni radio e televisive che distribuiscono contenuti attraverso Internet.

Le trasmissioni di programmi in diretta sono spesso ricevute contemporaneamente da molti client; la loro distribuzione attualmente avviene tramite CDN (Paragrafo 2.6). Tuttavia, l'odierna distribuzione di audio/video in diretta è molto spesso realizzata attraverso il multicast a livello applicativo (usando CDN o reti P2P) o tramite flussi unicast multipli e separati. Come per lo streaming di contenuti registrati, anche in questo caso la rete deve fornire a ogni flusso un throughput medio maggiore del suo bit rate per garantire la continuità della riproduzione. Siccome l'evento è in diretta, anche il ritardo potrebbe essere un problema, sebbene i limiti sulle tempistiche siano meno restrittivi rispetto alle applicazioni di conversazione. Nelle trasmissioni in diretta possono infatti essere tollerati ritardi fino a una decina di secondi da quando l'utente richiede l'invio a quando inizia la riproduzione. Non ci occuperemo di streaming in tempo reale in questo libro, in quanto molte delle sue tecniche di streaming (ritardo iniziale, uso adattativo della banda, e distribuzione via CDN) sono simili a quelle usate per i contenuti registrati.

## **9.2 Streaming di video registrato**

Per le applicazioni di streaming video, i contenuti video registrati sono memorizzati su server a cui gli utenti inviano richieste per vedere i video. Gli utenti possono guardare i video dall'inizio alla fine senza interruzioni, possono fermare la visione prima della fine o interrompere il video mettendolo in pausa e ricominciando da una scena passata o successiva. I sistemi di video streaming sono classificabili in tre categorie: **streaming UDP**, **streaming HTTP** e **streaming HTTP** adattativi (Paragrafo 2.6). Nonostante tutti e tre siano di fatto utilizzati, la maggior parte dei sistemi moderni impiega gli ultimi due.

Una caratteristica comune a tutte e tre le forme di video streaming è l'uso esteso dei buffer dell'applicazione sul lato client per mitigare gli effetti del ritardo end-to-end variabile e la variabilità della banda disponibile tra server e client. Nello streaming video, sia memorizzato sia in tempo reale, gli utenti in genere possono tollerare un ritardo iniziale di alcuni secondi tra quando il client richiede un video a quando la riproduzione inizia. Di conseguenza il client, quando gli comincia ad arrivare il video, non deve immediatamente iniziare la riproduzione, ma può costruirsi una riserva in un buffer dell'applicazione. Una volta che il client ha costruito tale riserva di alcuni secondi di video memorizzati, ma non ancora riprodotti, può iniziare la riproduzione. Vi sono due importanti vantaggi derivati da questa procedura di **buffering lato client**. In primo luogo può assorbire le variazioni del tempo di ritardo tra server e client: se una particolare parte dei dati video è in ritardo, purché arrivi prima che il buffer sia



**Figura 9.1** Ritardo di riproduzione del client in uno streaming video.

esaurito, non verrà notato dall’utente. In secondo luogo, se la banda tra server e client improvvisamente va al di sotto del tasso di consumo dei dati, l’utente non se ne accorge finché il buffer non sarà vuoto.

Il buffering lato client è illustrato nella Figura 9.1. In questo semplice esempio supponete che il video sia codificato a un bit rate fisso e che quindi ogni blocco video contenga fotogrammi (detti anche frame) che vengono riprodotti nella stessa quantità di tempo  $\Delta$ . Il server trasmette il primo blocco al tempo  $t_0$ , il secondo al tempo  $t_0 + \Delta$ , il terzo al tempo  $t_0 + 2\Delta$  e così via. Una volta che il client inizia la riproduzione, ogni blocco dovrebbe essere riprodotto per  $\Delta$  unità di tempo dopo il blocco precedente per riprodurre la temporizzazione del video originale. A causa dei ritardi di rete end-to-end variabili, differenti blocchi video subiscono ritardi diversi. Il primo blocco arriva al client al tempo  $t_1$ , il secondo al tempo  $t_2$ . Il ritardo di rete dell’i-esimo blocco è la differenza tra il tempo in cui è stato ricevuto dal client e il tempo in cui il blocco è stato trasmesso dal server; notate che il ritardo di rete varia da un blocco video all’altro. In questo esempio, se il client iniziasse la riproduzione non appena fosse arrivato il primo blocco al tempo  $t_1$ , il secondo blocco non arriverebbe in tempo per essere riprodotto all’istante  $t_1 + \Delta$ . In questo caso, la riproduzione del video dovrebbe fermarsi, aspettando l’arrivo del blocco 1, o saltare il blocco 1; entrambi i casi portano a uno spiacevole danneggiamento della riproduzione. Al contrario, se il client ritardasse l’inizio della riproduzione al tempo  $t_3$ , quando fossero arrivati i blocchi da 1 a 6, la riproduzione periodica procederebbe avendo ricevuto tutti i blocchi prima del loro tempo di riproduzione.

### 9.2.1 Screaming UDP

Discuteremo lo streaming UDP solo brevemente in questo paragrafo. Nello streaming UDP il server trasmette il video allo stesso bit rate a cui il client lo consuma, inviando i blocchi video in pacchetti UDP a un tasso costante. Per esempio, se il tasso di consumo del video fosse di 2 Mbps e ogni pacchetto UDP trasportasse 8000 bit di video, il server immetterebbe un pacchetto UDP nella sua socket ogni  $(8000 \text{ bit})/2 \text{ Mbps} = 4 \text{ ms}$ . Come visto nel Capitolo 3, poiché UDP non ha un meccanismo di controllo di congestione, il server può immettere nella rete i pacchetti al tasso di consumo del video senza le restrizioni di TCP. Lo streaming UDP usa generalmente un buffer lato client molto piccolo, in grado di contenere meno di un secondo di video.

Il server, prima di passare i blocchi video a UDP, li incapsula in pacchetti di trasporto appositamente progettati per audio e video, usando il protocollo di trasporto in tempo reale (RTP, *real-time transport protocol*) [RFC 3550] o altri schemi simili, magari proprietari. Tratteremo RTP nel contesto dei sistemi per conversazioni voce e video nel Paragrafo 9.3.

Un'altra proprietà che contraddistingue lo streaming UDP è il fatto che client e server mantengono oltre al flusso video anche, in parallelo, una connessione di controllo separata sulla quale il client invia i comandi riguardanti i cambiamenti di stato della sessione, quali la pausa, la ripresa della riproduzione, il riposizionamento e così via. Il *real-time streaming protocol* (RTSP) [RFC 2326], spiegato in parte nel sito web del testo, è un protocollo aperto e usato diffusamente per questo tipo di connessioni di controllo.

Sebbene venga impiegato in molti sistemi open-source e prodotti proprietari, soffre di tre svantaggi significativi. Il primo è il fatto che lo streaming UDP a tasso costante può non riuscire a fornire riproduzione continua, data la quantità di banda disponibile tra server e client non solo variabile, ma anche impredicibile. Considerate, per esempio, uno scenario in cui il tasso di consumo del video sia 1 Mbps e la banda disponibile tra server e client sia normalmente più di 1 Mbps, ma che ogni pochi minuti la banda disponibile scenda al di sotto di 1 Mbps per alcuni secondi. Il sistema di streaming UDP che trasmette il video a un tasso costante di 1 Mbps su RTP/UDP fornirebbe all'utente un pessimo servizio, con frame saltati o fermi immagine subito dopo il crollo della banda. Il secondo svantaggio dello streaming UDP è il fatto che richiede un server di controllo, come un server RTSP, per elaborare le richieste interattive da client a server e tracciare lo stato del client (per esempio, il punto di riproduzione del video, se il video è in pausa o in riproduzione e così via) per ogni sessione client attiva. Tutto ciò aumenta la complessità e i costi di installazione di un sistema di video-on-demand su larga scala. Il terzo svantaggio deriva dal fatto che molti utenti non possono ricevere video UDP, in quanto i loro firewall sono configurati per bloccare tale traffico.

## 9.2.2 Screaming HTTP

Nello streaming HTTP il video viene semplicemente memorizzato in un server HTTP come un file ordinario con un URL specifico. Quando un utente vuole vedere un video, il client stabilisce una connessione TCP con il server e invia una richiesta GET HTTP per il suo URL. Il server invia il file video, all'interno di un messaggio di risposta HTTP, più velocemente possibile, vale a dire tanto più velocemente quanto il controllo di flusso e di congestione TCP lo permettono. Sul lato client i byte vengono memorizzati in un buffer dell'applicazione client. Quando il numero di byte nel buffer supera una soglia fissata, l'applicazione client inizia la riproduzione: periodicamente prende i frame video dal buffer, li decomprime e li visualizza all'utente.

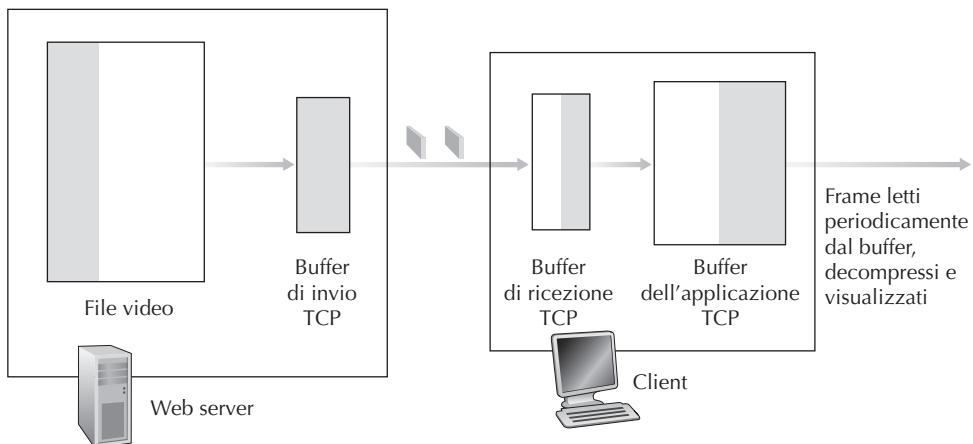
Come visto nel Capitolo 3 il tasso di trasmissione da server a client durante il trasferimento di un file su TCP può variare in modo significativo a causa dei meccanismi di controllo di congestione di TCP. Il tasso di trasmissione varia spesso con un comportamento a dente di sega tipico del controllo di congestione TCP. Inoltre, i pacchetti possono subire un ritardo significativo a causa della ritrasmissione di TCP. A seguito di tutte queste caratteristiche di TCP, nel 1990 l'impressione generale era che lo streaming video su TCP non avrebbe mai funzionato bene. Tuttavia, col tempo, i progettisti dei sistemi di streaming video videro che, usando buffer e prefetching (che discuteremo nel prossimo paragrafo) si poteva ottenere una riproduzione continua anche con in presenza dei meccanismi di trasferimento dati affidabile e di controllo di congestione di TCP.

Inoltre l'uso di HTTP e TCP permette ai video di attraversare più facilmente firewall e i NAT che sono spesso configurati per bloccare la maggior parte del traffico UDP, ma lasciano passare la maggior parte del traffico HTTP. Inoltre, lo streaming HTTP elimina la necessità di avere un server di controllo, come un server RTSP, riducendo i costi di un'installazione su larga scala su Internet. Per tutti questi vantaggi la maggior parte delle applicazioni di video streaming, compresi YouTube e Netflix, usano oggiorno lo streaming HTTP su TCP come protocollo di streaming.

### Prefetching del video

Come abbiamo appena visto, il buffer lato client può essere usato per mitigare gli effetti della variabilità del ritardo end-to-end e della banda disponibile. Nell'esempio della Figura 9.1, il server trasmette il video al tasso al quale il video viene riprodotto. Tuttavia, nello streaming di video registrato, il client può tentare di scaricare il video a un tasso più alto di quello di consumo, facendo il **prefetching** (letteralmente, “andando a prendere anzitempo”) dei dati che verranno consumati più tardi. Naturalmente il video ottenuto tramite prefetch viene memorizzato nel buffer dell'applicazione client. Il prefetching avviene in modo naturale con lo streaming TCP, in quanto il meccanismo di controllo della congestione di TCP tenterà di usare tutta la banda disponibile tra il server e il client.

Consideriamo un semplice esempio: supponete che il tasso di consumo del video sia 1 Mbps, ma che la rete sia in grado di consegnare il video al client a un tasso costante di 1,5 Mbps. Il client non solo sarà in grado di riprodurre il video con un ritardo



**Figura 9.2** Streaming video pre-registrato su HTTP/TCP.

molto piccolo, ma potrà anche aumentare la quantità di video memorizzata nel buffer di 500 Kbit ogni secondo. In questo modo il client, se in un tempo successivo ricevesse i dati a un tasso minore di 1 Mbps per un breve periodo, sarà in grado di continuare la riproduzione attingendo alla riserva nel buffer. [Wang 2008] mostra che quando il throughput medio di TCP è circa il doppio del bit rate del contenuto, lo streaming su TCP garantisce la presenza dei dati da riprodurre nella stragrande maggioranza dei casi e produce piccoli ritardi dovuti al buffer.

### Buffer dell'applicazione client e buffer TCP

La Figura 9.2 mostra l'interazione tra client e server nello streaming HTTP. Lato server, la porzione del file video in bianco è quella già inviata sulla socket, quella grigia è quanto rimane da trasmettere. I byte, dopo essere passati attraverso la socket, vengono posti nel buffer di trasmissione di TCP prima di essere inviati in Internet, come descritto nel Capitolo 3. Nella Figura 9.2, poiché il buffer di invio TCP è pieno, il server non può momentaneamente inviare altri byte nella socket. L'applicazione client (il media player) legge i byte dal buffer di ricezione TCP, attraverso la sua socket, e li pone nel buffer dell'applicazione client. Parallelamente, l'applicazione client prende periodicamente i frame video dal buffer, li decomprime e li visualizza. Si noti che se il buffer è più grande del file del video, l'intero processo di spostamento dei byte dalla memoria del server al buffer dell'applicazione client è equivalente a scaricare un file ordinario su HTTP; semplicemente il client prende il video dal server tanto velocemente quanto TCP lo permette.

Consideriamo ora che cosa succede quando l'utente mette in pausa il video durante il processo di streaming. Durante il periodo di pausa, i bit continuano ad arrivare al buffer dal server, ma non ne vengono rimossi. Se il buffer ha dimensione finita, alla fine potrebbe diventare pieno. Una volta che il buffer dell'applicazione client è pieno, i byte non possono venir rimossi dal buffer di ricezione TCP che quindi diventa

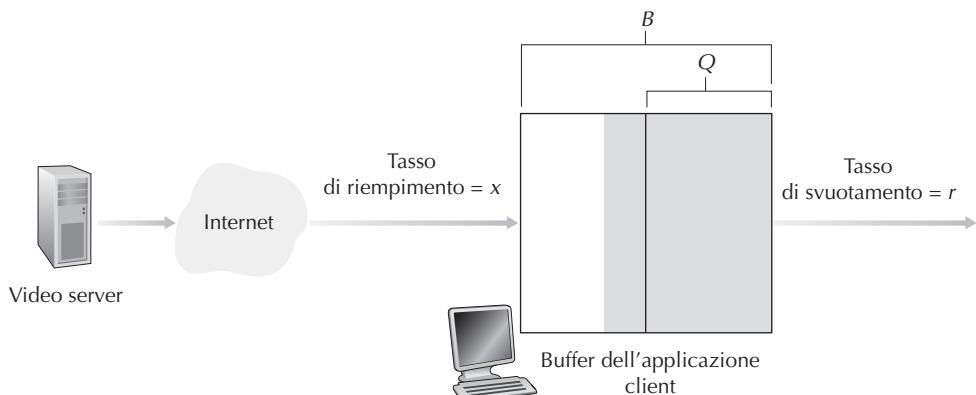
anch'esso pieno. Una volta che il buffer di ricezione TCP del client diventa pieno, i byte non possono più essere rimossi dal buffer di invio TCP del client, che diventa anch'esso pieno. A questo punto il server non può più inviare byte nella socket. Quindi quando l'utente mette in pausa il video, il server è costretto a fermare la trasmissione finché il video non viene fatto ripartire.

Anche durante la riproduzione regolare senza pause, se il buffer dell'applicazione client diventa pieno, anche i buffer TCP diventano pieni e il server viene forzato a ridurre il suo tasso di invio. Per determinare il tasso risultante, notate che quando l'applicazione client rimuove  $f$  bit, crea nel buffer dell'applicazione client spazio per  $f$  bit, permettendo al server di inviare ulteriori  $f$  bit; quindi, il tasso di invio del server non può essere maggiore del tasso di consumo del video del client. Perciò un buffer dell'applicazione client pieno impone indirettamente un limite al tasso con cui il video viene trasmesso dal server al client quando si usa lo streaming su HTTP.

### Analisi dello streaming video

Facciamo ora un semplice modello per comprendere meglio il ritardo iniziale di riproduzione e il blocco della riproduzione dovuto allo svuotamento del buffer dell'applicazione. Come mostrato nella Figura 9.3, sia  $B$  la dimensione in bit del buffer dell'applicazione client e sia  $Q$  il numero di bit che deve essere memorizzato nel buffer prima che l'applicazione client inizi la riproduzione. Ovviamente deve valere  $Q < B$ . Sia  $r$  il tasso di consumo del video, cioè il tasso al quale il client prende i bit dal buffer dell'applicazione durante la riproduzione. Quindi, per esempio, se il video prevede 30 frame al secondo e ogni frame compresso è di 100.000 bit, allora  $r = 3$  Mbps. Per rimanere a una visione ad alto livello, trascuriamo i buffer di invio e ricezione TCP.

Assumiamo che il server invii bit a un tasso costante  $x$  quando il buffer del client non è pieno. Questa è una grossa semplificazione, perché il tasso di invio di TCP varia a causa del controllo di congestione; esamineremo in uno dei problemi elencati alla fine del capitolo il caso più realistico di tasso  $x(t)$  dipendente dal tempo. Supponente



**Figura 9.3** Analisi delle operazioni di buffering sul lato client per lo streaming video.

che al tempo  $t = 0$ , il buffer dell'applicazione sia vuoto e il video inizi ad arrivare al buffer dell'applicazione client. A quale tempo  $t = t_p$  inizia la riproduzione? E a quale tempo  $t = t_f$  il buffer dell'applicazione client diventa pieno?

Determiniamo innanzitutto  $t_p$ , tempo al quale sono entrati nel buffer dell'applicazione  $Q$  bit e la riproduzione inizia. Si ricordi che i bit arrivano al buffer dell'applicazione client con tasso  $x$  e nessun bit viene rimosso dal buffer prima dell'inizio della riproduzione. Quindi, la quantità di tempo richiesta per avere  $Q$  bit (il ritardo iniziale di buffering) è  $t_p = Q/x$ .

Determiniamo ora  $t_f$ , tempo al quale il buffer dell'applicazione client diventa pieno. Osserviamo innanzitutto che, se  $x < r$  (il tasso di invio del server è minore del tasso di consumo del video) il buffer del client non diventerà mai pieno. Quindi, a partire dal tempo  $t_p$ , il buffer verrà svuotato con tasso  $r$  e verrà riempito al tasso  $x < r$ . Infine il buffer del client verrà interamente svuotato, il video si bloccherà sullo schermo, mentre il buffer del client aspetterà altri  $t_p$  secondi per costruire  $Q$  bit di video. *Quindi, quando il tasso disponibile nella rete è minore di quello del video, la riproduzione subisce periodi alternati di riproduzione continua e di fermi immagine.* In uno dei problemi a fine capitolo vi si chiederà di determinare la lunghezza di tali periodi in funzione di  $Q$ ,  $r$  e  $x$ . Determiniamo ora  $t_f$  nel caso  $x > r$ . In questo caso, a partire dal tempo  $t_p$ , il buffer aumenta da  $Q$  a  $B$  con tasso  $x - r$  perché, come mostrato nella Figura 9.3, i bit vengono svuotati con tasso  $r$ , ma arrivano con tasso  $x$ . Con questi suggerimenti potrete risolvere il problema in cui vi si chiede di determinare il tempo  $t_f$  a cui il buffer del client diventa pieno. Si noti che, usando TCP, *quando il tasso disponibile della rete è maggiore di quello del video, l'utente, dopo un ritardo di buffering iniziale, potrà ottenere una riproduzione continua fino alla fine del video.*

### Interruzione anticipata e riposizionamento del video

I sistemi di streaming HTTP spesso usano l'**intestazione HTTP byte-range** nel messaggio di richiesta GET di HTTP, che specifica l'intervallo di byte del video desiderato che il client vuole ricevere. Questa procedura è particolarmente utile quando l'utente vuole saltare a un punto successivo del video; il client invia una nuova richiesta HTTP, indicando da quale byte del file il server dovrebbe inviare i dati. Il server, quando riceve la nuova richiesta HTTP, si dimentica di quelle precedenti e invia i byte cominciando da quello indicato nella nuova richiesta.

Facciamo brevemente notare che quando un utente si riposiziona su un punto successivo del video o effettua una interruzione anticipata, alcuni dati ottenuti tramite prefetch, ma non ancora trasmessi dal server, non verranno visti dall'utente: uno spreco di banda e di risorse del server. Supponiamo, per esempio, che il buffer del client sia pieno con  $B$  bit del video al tempo  $t_0$  e che a tale tempo l'utente effettui un riposizionamento all'istante  $t > t_0 + B/r$  del video che quindi vedrà fino alla fine. In questo caso, tutti i  $B$  bit del buffer non verranno visti e le risorse del server e la banda usate per trasmetterli verranno sprecate.

Un terzo tipo di streaming, oltre ai due appena visti, è lo streaming dinamico adattativo su HTTP (DASH, *dynamic adaptive streaming over HTTP*), trattato nel Para-

grafo 2.6.2, in cui i video vengono codificati in diverse versioni, ognuna avente un bit rate differente e quindi un differente livello di qualità. La distribuzione di video tramite CDN è stata trattata nel Paragrafo 2.6.3.

## 9.3 Voice-over-IP

Il servizio di conversazione in tempo reale su Internet viene comunemente chiamato **telefonia Internet** o **Voice-over-IP** (VoIP) in quanto, per l’utente, è simile al tradizionale servizio telefonico a commutazione di circuito per comunicare in tempo reale. In questo paragrafo descriveremo i principi e i protocolli di VoIP. La conversazione video è simile, per molti aspetti, al VoIP, se non per il fatto che include il video dei partecipanti oltre che le loro voci. Per meglio mantenere il discorso su di un piano concreto ci focalizziamo sulla sola voce piuttosto che sulla combinazione di voce e video.

### 9.3.1 Limiti del servizio best-effort di IP

Come detto in precedenza, il protocollo a livello di rete di Internet, IP, fornisce un servizio best-effort, ossia fa del suo meglio per recapitare i datagrammi il più velocemente possibile, senza però fornire alcuna assicurazione in merito al ritardo, alla perdita o all’entità del jitter dei pacchetti. Essendo la telefonia Internet e la videoconferenza in tempo reale particolarmente sensibili a questi aspetti, l’assenza di garanzie crea notevoli problemi alla progettazione di tali applicazioni.

In questo paragrafo tratteremo molti metodi con i quali si possono migliorare le prestazioni del VoIP rispetto al servizio best-effort. Ci concentreremo sulle tecniche a livello applicativo, cioè sugli approcci che non richiedono alcun cambiamento al nucleo della rete o al livello di trasporto dei sistemi periferici. Per concretezza, tratteremo il caso di un’applicazione di telefonia Internet. L’utente genera un segnale audio costituito da un flusso di 8000 byte al secondo: ogni 20 ms li riunisce in blocchi da 160 byte ai quali è collegata una speciale intestazione (il cui contenuto sarà descritto in seguito). Quindi, il numero di byte in un blocco è  $(20 \text{ ms}) \times (8000 \text{ byte/s}) = 160 \text{ byte}$  e un segmento UDP viene inviato ogni 20 ms.

Se ogni pacchetto arriva a destinazione con un ritardo end-to-end costante, i pacchetti giungono con intervalli di 20 ms al ricevente che, in condizioni ideali, deve solamente riprodurre ciascun blocco al momento del suo arrivo. Ma, sfortunatamente, alcuni pacchetti possono andare persi e la maggior parte di essi non avrà lo stesso ritardo, anche in condizioni di congestione lieve. Per questo motivo il ricevente deve prestare molta attenzione nel determinare il momento in cui riprodurre un blocco e nel decidere che cosa fare dei blocchi mancanti.

#### Perdita di pacchetti

Consideriamo uno dei segmenti UDP generati dall’applicazione VoIP. Il segmento è incapsulato in un datagramma IP che, mentre viaggia nella rete, passa attraverso i buffer dei router, cioè le loro code, per poter accedere al collegamento di uscita.

È possibile che uno o più di questi buffer siano pieni e non possano ricevere il datagramma che, in questo caso, viene scartato e non arriverà mai a destinazione.

Le perdite possono essere eliminate inviando i pacchetti con il protocollo TCP (che fornisce un trasferimento affidabile) invece che con UDP. Infatti, TCP ritrasmette i pacchetti che non arrivano a destinazione. Tuttavia, la ritrasmissione è spesso inaccettabile per le applicazioni audio interattive in tempo reale come VoIP, in quanto aumenta il ritardo complessivo [Bolot 1996]. Inoltre, a causa del controllo della congestione di TCP, dopo la perdita di un pacchetto il tasso originale di trasmissione può scendere al di sotto di quello con cui il ricevente svuota il proprio buffer, che potrebbe non essere alimentato a sufficienza, compromettendo anche pesantemente l'intelligenza della voce in ricezione. Per queste ragioni, quasi tutte le applicazioni di VoIP fanno uso per default di UDP, senza curarsi della ritrasmissione dei pacchetti persi. [Baset 2006] riporta che UDP viene usato da Skype a meno che l'utente sia dietro un NAT o un firewall che blocca i segmenti UDP, nel qual caso usa TCP.

La perdita dei pacchetti non è infatti così grave come si potrebbe pensare: tassi compresi fra 1 e 20% possono essere tollerati, a seconda di come la voce è codificata e trasmessa, e di come la perdita è mascherata in ricezione, per esempio utilizzando la correzione anticipata degli errori (FEC, *forward error correction*) che trasmette informazioni ridondanti insieme a quelle originali, in modo che alcuni dati persi possono essere recuperati. Nonostante ciò, se uno o più collegamenti tra il trasmittente e il ricevente sono gravemente congestionati e la perdita dei pacchetti supera il 10-20% (per esempio su collegamenti wireless), allora risulta impossibile raggiungere una qualità accettabile. È chiaro che il servizio best-effort non è privo di limitazioni.

## Ritardo end-to-end

La locuzione *end-to-end delay* indica la somma dei ritardi di trasmissione, di elaborazione e di accodamento nei router, più quelli di propagazione lungo i collegamenti e di elaborazione sui terminali. Per un'applicazione audio in tempo reale come VoIP, ritardi end-to-end complessivi inferiori a 150 ms non sono percepiti dall'orecchio umano, fra 150 e 400 ms possono essere accettabili ma non ideali, se invece superano i 400 ms possono limitare seriamente l'interattività nella conversazione. Il lato ricevente di un'applicazione VoIP trascura solitamente i pacchetti con ritardi superiori a una certa soglia, generalmente di 400 ms. Quindi, questi pacchetti sono effettivamente persi.

## Jitter di un pacchetto

Un aspetto cruciale del ritardo end-to-end è costituito dalla variabilità nelle code dei router, che può generare sostanziali differenze nel tempo impiegato da ciascun pacchetto tra origine e destinazione, da quando viene creato a quando viene ricevuto, come mostrato nella Figura 9.1. Questo fenomeno è detto **jitter**. Come esempio, consideriamo due pacchetti consecutivi in un'applicazione VoIP. Il trasmittente invia il secondo pacchetto 20 ms dopo aver spedito il primo. Tuttavia l'intervallo con cui i due pacchetti giungono al ricevente può essere superiore a quello di trasmissione. Per rendercene conto, supponiamo che il primo pacchetto giunga al router quando questi

ha una coda quasi vuota e che essa, pochi istanti prima che sopraggiunga il secondo, riceva un gran numero di pacchetti provenienti da altre sorgenti. Poiché il primo pacchetto è soggetto a un ritardo piccolo, mentre il secondo a uno grande presso quel router, i due pacchetti arriveranno separati da un ritardo che supera i 20 ms. Ma non sempre è così: ci sono casi in cui l'intervallo può anche essere inferiore a 20 ms. Supponiamo che il primo pacchetto debba accodarsi a un gran numero di pacchetti e che dopo non ne giungano altri. Ora i due pacchetti si trovano uno dietro l'altro nella coda. Se il tempo richiesto per la ritrasmissione sul collegamento in uscita dal router è inferiore a 20 ms, allora il primo e il secondo pacchetto saranno distanziati da meno di 20 ms.

La situazione è analoga a quella della guida di un automobile. Supponete che voi e un vostro amico stiate percorrendo l'Autostrada del sole, da Milano a Roma, che abbiate lo stesso stile di guida e che viaggiate a 100 km/h, trafficando. È chiaro che, se il vostro amico parte un'ora prima di voi, indipendentemente dal traffico, voi dovreste arrivare a Roma più o meno un'ora dopo di lui.

Se il ricevente ignora il jitter, e riproduce i blocchi man mano che sopraggiungono, la qualità audio può risultare non intelligibile. Fortunatamente, spesso il jitter può essere rimosso tramite **numeri di sequenza** (*sequence number*), **marcature temporali** (*timestamp*) e **ritardo di riproduzione** (*playout delay*), come discusso qui di seguito.

### 9.3.2 Rimozione del jitter al ricevente

Per un'applicazione vocale come VoIP, nella quale i pacchetti vengono generati periodicamente, il ricevente deve cercare di fornire la riproduzione sincrona dei blocchi vocali, anche in presenza di jitter provocato dalla rete. Ciò è solitamente possibile combinando i seguenti due meccanismi.

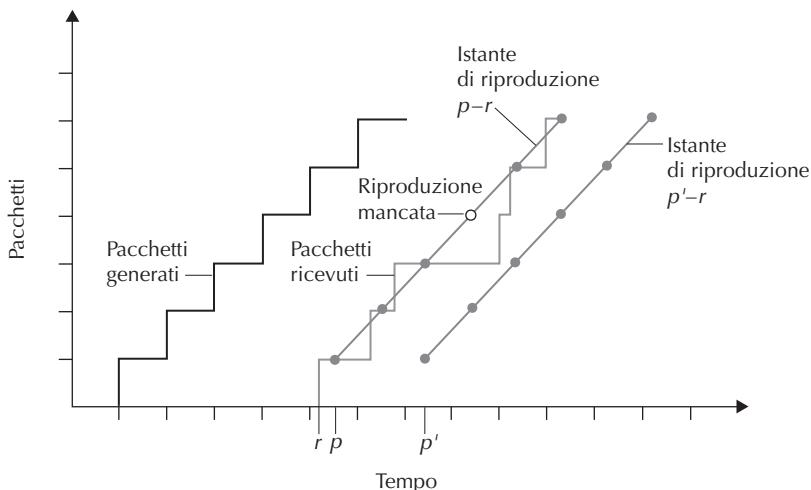
- *Facendo precedere i blocchi da una marcatura temporale.* Il trasmittente contrassegna ciascun blocco con l'indicazione dell'istante in cui è stato generato.
- *Inserendo un ritardo nella riproduzione del blocco al suo ricevimento, che deve essere abbastanza lungo da consentire la ricezione della maggior parte dei pacchetti prima di iniziare la loro riproduzione.* Questo ritardo può essere fissato per tutta la durata della conferenza, o variato durante il suo svolgimento.

Discuteremo ora come la combinazione di questi tre meccanismi possa ridurre o anche eliminare del tutto gli effetti del jitter. Esamineremo due principali strategie di riproduzione: con ritardo di riproduzione fisso oppure adattativo.

#### Ritardo di riproduzione fisso

Con questa strategia il ricevente tenta di riprodurre ciascun blocco esattamente  $q$  milisecondi dopo che è stato generato. Così, se un blocco è contrassegnato da un tempo di generazione  $t$ , il ricevente lo riproduce nell'istante  $t + q$  (se il blocco è arrivato in tempo utile, altrimenti lo scarta e lo considera perso).

Qual è la miglior scelta per  $q$ ? VoIP può supportare ritardi fino a 400 ms, sebbene la qualità sia migliore con valori minori di  $q$ . D'altra parte, se  $q$  è molto inferiore a



**Figura 9.4** Perdita di pacchetti per la diversità dei ritardi di riproduzione.

400 ms, molti pacchetti arriverebbero troppo tardi rispetto all’istante in cui sono programmati per la riproduzione a causa del jitter. In linea di massima, in presenza di ampie variazioni nel ritardo end-to-end, è preferibile utilizzare un elevato valore di  $q$ ; se invece i ritardi (e le variazioni) sono minimi, è meglio utilizzare un valore di  $q$  inferiore a 150 ms.

Il legame fra ritardo di riproduzione e perdita dei pacchetti è illustrato nella Figura 9.4 che mostra gli istanti in cui i pacchetti sono generati e riprodotti, per un singolo flusso di parole. Sono considerati due ritardi iniziali di riproduzione. Come indica la linea a gradini più a sinistra, il trasmittente genera pacchetti a intervalli regolari, supponiamo ogni 20 ms. Il primo pacchetto è ricevuto al tempo  $r$ , mentre quelli successivi non sono equidistanziati a causa del jitter.

Nel primo caso, il ritardo di riproduzione iniziale fisso è posto a  $p - r$ . Con questo valore, il quarto pacchetto non arriverà entro il tempo programmato per la riproduzione e il ricevente lo considererà perso. Nel secondo caso, invece, il ritardo iniziale di riproduzione è posto a  $p' - r$ . Ora tutti i pacchetti arriveranno entro i tempi programmati per la loro riproduzione e non ci saranno perdite.

### Ritardo di riproduzione adattativo

Il nostro esempio evidenzia l’importante compromesso fra ritardo e perdita dei pacchetti che occorre stabilire quando si progetta una strategia di riproduzione con ritardi fissi. Scegliendo un ampio ritardo di riproduzione iniziale, molti pacchetti arriveranno in tempo utile e le perdite saranno trascurabili. Per un servizio interattivo, come VoIP, lunghi ritardi possono però diventare irritanti, se non intollerabili. Idealmente, vorremmo un ritardo di riproduzione minimo con il vincolo che la perdita sia al di sotto dei pochi punti percentuale.

Il modo naturale di trattare questo compromesso è stimare il ritardo della rete e le sue variazioni, regolando conseguentemente il ritardo di riproduzione con l'inizio di ciascun periodo di attività vocale. Questa regolazione adattativa del ritardo di riproduzione all'inizio dei periodi di attività vocale farà in modo che le pause dei trasmittenti siano compresse o prolungate, secondo la necessità. Questa soluzione, se contenuta in limiti ragionevoli, non è rilevabile dall'ascoltatore.

Seguendo [Ramjee 1994], descriviamo ora un algoritmo generico che il ricevente può utilizzare per la regolazione adattativa dei suoi ritardi di riproduzione. Poniamo:

$t_i$  = marcatura temporale dell' $i$ -esimo pacchetto = istante in cui il pacchetto è generato dal mittente;

$r_i$  = istante in cui il pacchetto  $i$  è ricevuto;

$p_i$  = istante in cui il pacchetto  $i$  è riprodotto.

Il ritardo di rete end-to-end dell' $i$ -esimo pacchetto è  $r_i - t_i$ . A causa del jitter questo ritardo varierà da pacchetto a pacchetto. Indichiamo con  $d_i$  una stima del valore medio del ritardo alla ricezione dell' $i$ -esimo pacchetto. Questa stima è ricavata dalle marcature temporali come segue:

$$d_i = (1 - u) d_{i-1} + u (r_i - t_i)$$

dove  $u$  è una costante fissa (per esempio,  $u = 0,01$ ). Quindi  $d_i$  è una media livellata dei ritardi di rete osservati  $r_1 - t_1, \dots, r_i - t_i$ . La stima assegna maggior peso ai ritardi più recenti rispetto a quelli più lontani nel tempo. Questa forma di stima non dovrebbe essere del tutto sconosciuta; infatti è simile a quella dei tempi di round-trip in TCP (Capitolo 3). Indichiamo con  $v_i$  una stima della deviazione media dal ritardo medio stimato. Anche questo valore è ricavato dalle marcature temporali:

$$v_i = (1 - u) v_{i-1} + u |r_i - t_i - d_i|$$

Le stime di  $d_i$  e  $v_i$  sono calcolate per ogni pacchetto ricevuto, sebbene possano essere utilizzate solo per determinare il punto di riproduzione del primo pacchetto in qualsiasi periodo di attività.

Una volta calcolate queste stime, il ricevente impiega il seguente algoritmo per la riproduzione dei pacchetti. Se  $i$  è il primo pacchetto di un periodo di attività, il suo istante di riproduzione è dato da:

$$p_i = t_i + d_i + Kv_i$$

dove  $K$  è una costante positiva (per esempio,  $K = 4$ ). Lo scopo del termine  $Kv_i$  è di impostare l'istante di inizio della riproduzione con sufficiente ritardo da consentire che solo una piccola frazione dei pacchetti durante il periodo di attività vada persa a causa del ritardo. In tale periodo, il punto di riproduzione per ogni pacchetto successivo è calcolato come lo spostamento dal momento in cui è riprodotto il primo pacchetto. In particolare, poniamo che

$$q_i = p_i - t_i$$

sia il tempo che trascorre da quando il primo pacchetto nel periodo di attività è generato al momento della sua riproduzione. Se anche il pacchetto  $j$  appartiene a questo periodo di attività, esso viene riprodotto all’istante

$$p_j = t_j + q_i$$

Questo algoritmo funziona perfettamente quando il ricevente sa se il pacchetto è il primo di un periodo di attività vocale; e questo può essere dedotto attraverso l’esame dell’energia del segnale in ogni pacchetto.

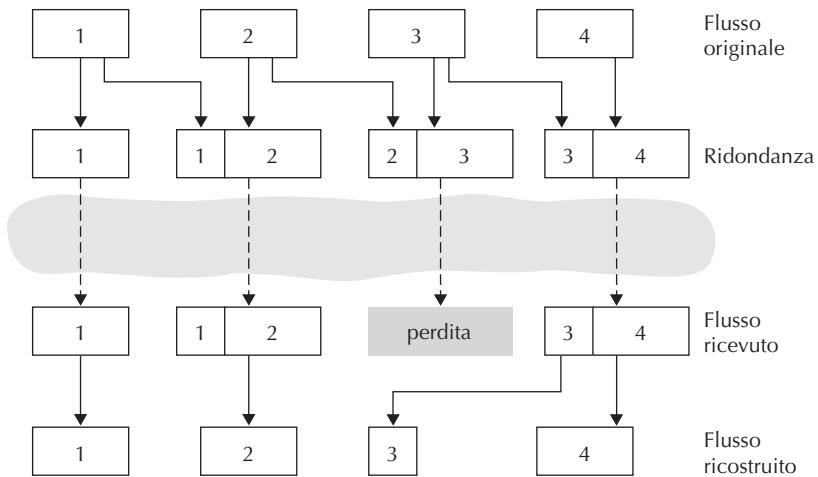
### 9.3.3 Recupero dei pacchetti persi

Abbiamo visto, con un certo dettaglio, come le applicazioni VoIP possano trattare il jitter dei pacchetti. Ora ci occuperemo degli **schemi di recupero delle perdite** (*loss recovery scheme*), che consentono di mantenere una qualità audio accettabile anche in presenza di perdita dei pacchetti. In questo caso consideriamo la perdita di un pacchetto nel suo senso più ampio: ossia quando non giunge mai al ricevente o arriva dopo il tempo programmato per la sua riproduzione. Il nostro esempio di VoIP servirà nuovamente come contesto per descrivere questi schemi.

Come detto all’inizio del paragrafo, la ritrasmissione dei pacchetti persi non si adice alle applicazioni interattive in tempo reale come il VoIP in quanto, se questi giungono oltre il tempo di riproduzione previsto, non hanno alcuna utilità. Inoltre, la ritrasmissione di un pacchetto che ha sovraccaricato la coda in un router non può avvenire abbastanza in fretta. Fatte queste considerazioni, le applicazioni VoIP utilizzano spesso alcuni schemi di anticipazione delle perdite. Due tipi di questi schemi sono la **correzione anticipata degli errori** (FEC, *forward error correction*) e l’**interfogliazione** (*interleaving*).

#### Correzione anticipata degli errori (FEC)

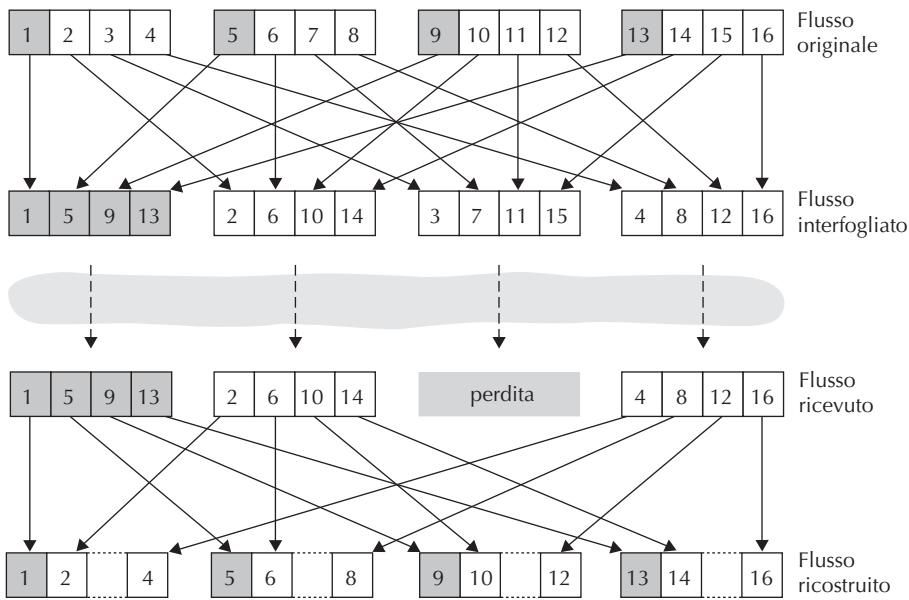
L’idea base di questo schema è quella di aggiungere informazioni ridondanti al flusso originale dei pacchetti. In cambio di un aumento marginale nel tasso trasmissivo dell’audio, la ridondanza può essere utilizzata per fornire un’approssimazione dell’esatta versione di alcuni pacchetti persi. Seguendo [Bolot 1996] e [Perkins 1998] tratteremo due meccanismi FEC. Il primo meccanismo invia, dopo ogni  $n$  blocchi, un blocco ridondante ottenuto da un’operazione di OR esclusivo degli  $n$  blocchi originali [Shacham 1990]. In questo modo, se qualche pacchetto del gruppo degli  $n+1$  pacchetti va perso, il ricevente lo può ricostruire integralmente. Ma se in un gruppo si perdono due o più pacchetti, il ricevente non può ricostruirli. Limitando la dimensione del gruppo, gran parte dei pacchetti smarriti possono essere recuperati se le perdite non sono eccessive. Tuttavia, più piccole sono le dimensioni del gruppo, maggiore è l’incremento richiesto al tasso trasmissivo. In particolare, la frequenza trasmissiva deve aumentare di un fattore  $1/n$ . Così, se  $n=3$ , allora si ha un incremento del 33%. Inoltre, questo semplice schema accresce il ritardo di riproduzione, in quanto il ricevente deve attendere di aver ricevuto l’intero gruppo di pacchetti prima di poterne iniziare la riproduzione. Per dettagli pratici sul funzionamento di FEC nel trasporto multimediale potete consultare [RFC 5109].



**Figura 9.5** Informazioni ridondanti di bassa qualità con piggyback.

Il secondo meccanismo FEC consiste nell’inviare uno stream audio a bassa risoluzione come informazione ridondante. Per esempio, il trasmittente può creare un flusso audio nominale e un corrispondente flusso a bassa risoluzione con bit rate più basso; per esempio, lo stream nominale potrebbe avere una codifica PCM a 64 kbps e quello a bassa qualità una codifica GSM a 13 kbps. A quest’ultimo ci si riferisce come al flusso ridondante. Come mostrato nella Figura 9.5, il trasmittente crea l’ $n$ -esimo pacchetto aggiungendo il blocco  $(n - 1)$ -esimo dello stream ridondante all’ $n$ -esimo blocco dello stream nominale. In tal modo, ogni volta che si perdono pacchetti non consecutivi, il ricevente può mascherare la perdita riproducendo il blocco codificato a bassa velocità che arriva con il pacchetto successivo. Certamente, questo blocco ha una qualità inferiore rispetto a quello nominale. Tuttavia, occasionali blocchi di bassa qualità in un flusso con elevate caratteristiche e in presenza di tutti i blocchi, non compromettono il buon livello dell’audio generale. Notiamo che in questo schema al ricevente sono sufficienti due pacchetti prima della riproduzione e questo, quindi, fa in modo che l’aumento del ritardo di riproduzione sia piccolo. Inoltre, se la codifica a bassa velocità è molto inferiore alla codifica nominale, allora anche l’aumento marginale della frequenza trasmissiva sarà contenuto.

Per poter rimediare alle perdite consecutive si può impiegare una semplice variante. Invece di aggiungere solo il blocco  $(n - 1)$ -esimo a bassa velocità all’ $n$ -esimo blocco nominale, il trasmittente può aggiungere anche il precedente blocco (o più di uno) a bassa qualità. In tal modo, la qualità audio al ricevente diventa accettabile per un’ampia varietà di ambienti best-effort. D’altra parte, l’aggiunta di blocchi aumenta la larghezza di banda richiesta per la trasmissione e il ritardo di riproduzione.



**Figura 9.6** Invio di audio interfogliato.

## Interfogliazione

In alternativa alla trasmissione ridondante, le applicazioni VoIP possono inviare audio interfogliato. Come mostra la Figura 9.6, il trasmittente pone in sequenza le unità dati audio, in modo che quelle adiacenti siano separate da una data distanza nel flusso trasmesso. L'interfogliazione (*interleaving*) può ridurre gli effetti della perdita di pacchetti. Se, per esempio, le unità hanno durata di 5 ms e i blocchi di 20 ms (cioè, 4 unità per blocco), allora il primo blocco può contenere le unità 1, 5, 9 e 13; il secondo le unità 2, 6, 10 e 14, e così via. La Figura 9.6 mostra che la perdita di un singolo pacchetto da un flusso interfogliato genera numerose piccole lacune nel flusso ricostruito, invece di una più vasta che si sarebbe verificata in uno flusso sequenziale.

L'interfogliazione può migliorare significativamente la qualità con cui si percepisce un flusso audio [Perkins 1998] e presenta anche bassa ridondanza. Il suo più grande vantaggio è che non richiede l'aumento di larghezza di banda del flusso. Lo svantaggio è che incrementa la latenza, limitando così il suo utilizzo in applicazioni interattive come la telefonia, sebbene possa dare buone prestazioni nello streaming di audio registrato.

## Nascondere gli errori

Gli schemi per nascondere gli errori cercano di sostituire un pacchetto perso con uno simile all'originale. Come discusso in [Perkins 1998], ciò è possibile perché il segnale audio, in particolare quello vocale, presenta forti caratteristiche di auto-somiglianza (*self similarity*) su brevi periodi. Queste tecniche funzionano bene per perdite contenute (inferiori al 15%) e per pacchetti piccoli (tali per cui se ne manda uno ogni 4-40 ms).

Quando la dimensione della perdita si avvicina a quella di un fonema (5-100 ms) queste tecniche falliscono, in quanto l'intero fonema potrebbe essere perduto dall'ascoltatore.

Forse, la forma più semplice di recupero da parte del ricevente è la ripetizione dei pacchetti, secondo la quale si possono sostituire i pacchetti persi con quelli pervenuti immediatamente prima della perdita. Questa tecnica richiede una bassa complessità computazionale e le prestazioni sono ragionevolmente buone. Un'altra di queste forme di recupero è l'interpolazione, che utilizza l'audio precedente e successivo alla perdita per generare un pacchetto adatto a coprirla. L'interpolazione funziona leggermente meglio della ripetizione dei pacchetti, ma richiede calcoli più complicati [Perkins 1998].

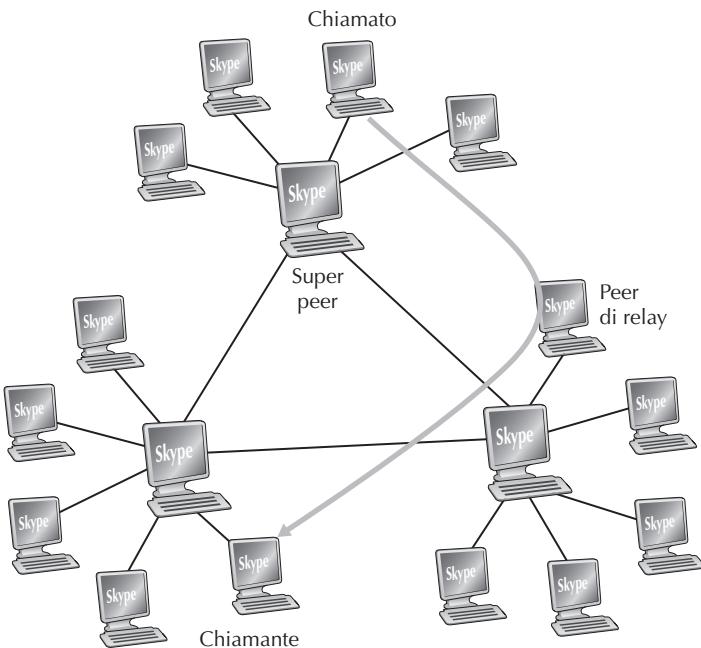
### 9.3.4 Un caso di studio: VoIP con Skype

Skype è un'applicazione VoIP molto popolare, con più di 50 milioni di utenti attivi su base giornaliera. Oltre a fornire il servizio VoIP da host a host, Skype offre anche servizi da host a telefono, da telefono a host e video conferenze con più partecipanti (per host si intende un dispositivo IP connesso a Internet, tra cui PC, tablet e smartphone). Skype è stato acquistato nel 2011 da Microsoft.

Poiché il protocollo Skype è proprietario e tutti i pacchetti sono cifrati, è difficile capire con precisione come funziona. Nonostante ciò, si è capito come funziona a livello generale utilizzando il sito web di Skype e alcuni studi che hanno effettuato delle misurazioni [Baset 2006; Guha 2006; Chen 2006; Suh 2006; Ren 2006; Zhang X 2012]. I client Skype hanno a disposizione, sia per la voce sia per il video, molti codificatori differenti, che possono codificare il contenuto con un'ampia gamma di bit rate e livelli di qualità. Per esempio, è stato misurato [Zhang X 2012] che le sessioni Skype vanno da 30 kbps per quelle a bassa qualità fino a 1 Mbps per quelle ad alta qualità. Tipicamente, la qualità dell'audio di Skype è più alta di quella di una linea telefonica tradizionale; infatti Skype campiona la voce almeno 16.000 volte al secondo, mentre la telefonia tradizionale ne usa solo 8000. Di default, Skype invia i pacchetti audio e video su UDP, mentre invia i pacchetti di controllo su TCP; tuttavia in presenza di firewall che bloccano UDP anche i pacchetti con contenuto multimediale sono inviati su TCP. Skype usa FEC per il ripristino delle perdite per i flussi sia voce sia video inviati su UDP. Inoltre, un client Skype adatta i flussi audio e video alle condizioni della rete, variando la qualità del video e dei dati aggiuntivi per il FEC [Zhang X 2012].

Skype usa le tecniche P2P in molti modi innovativi, mostrando come P2P possa essere usato in applicazioni che vanno oltre la distribuzione dei contenuti e la condivisione dei file. Come nella messaggistica istantanea, la telefonia Internet host-to-host è intrinsecamente P2P, in quanto coppie di utenti, i peer, comunicano tra di loro in tempo reale. Skype impiega le tecniche P2P per altre due importanti funzionalità: la localizzazione dell'utente e l'attraversamento dei NAT.

Come mostrato nella Figura 9.7, i peer (host) in Skype sono organizzati in una rete di overlay gerarchica in cui i peer sono classificati o come super peer o come peer ordinari. Skype mantiene un indice distribuito tra i super peer con l'associazione tra i nomi utente Skype e gli indirizzi IP correnti (e i numeri di porta). Quando Alice vuole chiamare Bob, il suo client Skype ricerca l'indice distribuito per determinarne



**Figura 9.7** Peer Skype.

l’indirizzo IP corrente di Bob. Essendo il protocollo Skype proprietario non possiamo sapere come gli indici vengano organizzati tra i super peer; è probabile che usi qualche forma di DHT.

Le tecniche P2P sono usate anche nei **relay** Skype, che svolgono la funzione di trarre vantaggio nello stabilire chiamate tra host nelle reti domestiche, che spesso accedono a Internet tramite NAT, come discusso nel Capitolo 4. Ricordiamo che un NAT impedisce a un host al di fuori della rete domestica di instaurare una connessione con un host all’interno. Quindi, se entrambi i partecipanti a una chiamata Skype hanno un router NAT si presenta un problema: nessuno dei due può accettare una chiamata attivata dall’altro. Un uso intelligente dei super peer e dei relay risolve brillantemente questo problema. Supponiamo che Alice, quando si autentica, inizi una sessione con un super peer che non è connesso tramite NAT. Poiché è Alice a iniziare la sessione, il suo NAT glielo consente. Alice e il suo super peer possono scambiarsi messaggi di controllo e anche Bob potrà farlo, quando si autenticherà. Alice, quando vuole chiamare Bob, informa il suo super peer, che informa il super peer di Bob, che a sua volta informa Bob che sta arrivando una chiamata da Alice. Se Bob accetta la chiamata, i due super peer selezionano un terzo super peer senza NAT, il relay, il cui compito è quello di trasportare i dati tra Alice e Bob. I due super peer istruiscono quindi Alice e Bob in modo che inizino una sessione con il relay. Come mostrato nella Figura 9.7, Alice invia i pacchetti voce al relay sulla connessione da Alice al relay, che è stata instaurata da Alice e il relay inoltra a sua volta tali pacchetti sulla connessione con Bob, che è stata instaurata da Bob; i pacchetti da Bob ad Alice fanno percorsi inversi.

Et voilà! Bob e Alice hanno una connessione end-to-end anche se nessuno originalmente poteva accettare una sessione proveniente dall'esterno della rete.

Fino a questo momento la nostra discussione su Skype si è concentrata su chiamate che coinvolgono due persone; esaminiamo ora le conferenze audio tra più persone. Se ognuno degli utenti inviasse una copia del suo flusso audio a ognuno degli altri  $N - 1$  utenti, sarebbe necessario immettere in rete  $N(N - 1)$  flussi audio. Per ridurre l'utilizzo di banda Skype usa una tecnica intelligente di distribuzione: ogni utente invia il suo flusso audio all'organizzatore della conferenza, che è colui che l'ha iniziata. Costui combina tutti i flussi audio in un unico flusso e ne invia una copia a ognuno dei  $N - 1$  partecipanti. In questo modo, il numero di flussi si riduce a  $2(N - 1)$ . Per le normali conversazioni video tra due persone, Skype utilizza una chiamata da peer a peer, a meno che sia necessario l'attraversamento di un NAT, caso in cui la chiamata passa attraverso un peer non sottoposto a NAT, come descritto precedentemente. Per una conferenza video con più di due partecipanti, Skype, data la natura del video, non può fare l'unione in un unico stream in un peer e quindi ridistribuirlo, come nel caso della voce. Ogni flusso video viene invece instradato a un cluster di server (in Estonia, nel 2011), che inoltra a ogni partecipante gli  $N - 1$  flussi degli altri [Zhang X 2012]. Ci si potrebbe chiedere perché non inviare tutte le copie a un server invece che a ogni partecipante, in quanto in entrambi gli approcci collettivamente vengono ricevuti  $N(N - 1)$  stream. La ragione è che la banda in upstream è, in generale, significativamente più bassa di quella in downstream nella maggior parte dei collegamenti di accesso e quindi i collegamenti in upstream potrebbero non essere in grado di supportare gli  $N - 1$  stream dell'approccio P2P.

I sistemi VoIP come Skype, QQ e Google Talk introducono nuove problematiche riguardanti la privacy. Alice, quando comunica con Bob su VoIP, può spiare l'indirizzo IP di Bob e usare servizi di geolocalizzazione [MaxMind 2016; Quova 2016] per determinare la sua posizione e il suo ISP. Infatti, con Skype, Alice può bloccare la trasmissione di alcuni pacchetti durante l'attivazione della chiamata per ottenere l'indirizzo IP di Bob, per esempio ogni ora, senza che Bob si accorga di essere tracciato e senza comparire nella lista dei contatti di Bob. Inoltre, l'indirizzo IP scoperto con Skype può essere correlato con quello trovato in BitTorrent, in modo da determinare quali file Bob stia scaricando [LeBlond 2011]. Infine, è possibile decodificare parzialmente una chiamata Skype effettuando un'analisi di traffico sulla grandezza dei pacchetti del flusso [White 2011].

## 9.4 Protocolli per applicazioni in tempo reale

Le applicazioni per conversazioni interattive in tempo reale (VoIP e video conferenza) sono molto diffuse. Non bisogna quindi stupirsi se organismi di standardizzazione, quali IETF e ITU, sono da molti anni impegnati nella stesura di standard per questa classe di applicazioni al fine di consentire alle aziende di sviluppare prodotti nuovi ed efficaci, che possono cooperare tra loro. In questo paragrafo esamineremo RTP e SIP, adottati da un'ampia gamma di prodotti.

### 9.4.1 RTP

Nel paragrafo precedente abbiamo appreso che il lato trasmittente di un'applicazione multimediale aggiunge campi di intestazione ai blocchi audio prima di passarli al livello di trasporto. Questi campi, che comprendono numeri di sequenza e marcature temporali, sono utilizzati da molte applicazioni di rete multimediali. Conviene quindi disporre di una struttura di pacchetto standardizzata per i campi dati audio/video, i numeri di sequenza e le marcature temporali, come pure per altri campi potenzialmente utili. RTP, definito nell'RFC 3550, può essere utilizzato per trasportare formati comuni come PCM, ACC e MP3 per l'audio e MPEG e H.263 per il video, ma anche per formati proprietari. Attualmente, RTP è implementato in centinaia di prodotti e prototipi di ricerca ed è anche utilizzato in abbinamento con altri importanti protocolli, come SIP.

In questo paragrafo presenteremo brevemente RTP. Il lettore interessato all'argomento può visitare il sito RTP di Henning Schulzrinne [Schulzrinne-RTP 2012] che fornisce interessanti informazioni, e quello di RAT [RAT 2012] che descrive un'applicazione VoIP che utilizza RTP.

#### Introduzione a RTP

Normalmente RTP utilizza UDP: il lato trasmittente incapsula un blocco di dati audio o video in un pacchetto RTP, incapsula poi quest'ultimo in un segmento UDP e lo affida a IP. Il lato ricevente estraе il pacchetto RTP dal segmento UDP, recupera il contenuto multimediale dal pacchetto e lo passa al media player per la decodifica e la riproduzione.

Consideriamo, come esempio, l'utilizzo di RTP per il trasporto della voce. Supponiamo che la sorgente vocale sia codificata con PCM (cioè campionata, quantizzata e digitalizzata) a 64 kbps e che l'applicazione raccolga i dati in blocchi di 20 ms, ovvero 160 byte in un blocco. Il lato trasmittente fa precedere ciascun blocco da una **intestazione RTP**, che comprende il tipo di codifica, un numero di sequenza e una marcatura temporale, e che occupa normalmente 12 byte. Il blocco e l'intestazione formano il **pacchetto RTP** che viene inviato, tramite una socket UDP, al lato ricevente. L'applicazione estraе il blocco audio dal pacchetto RTP e utilizza l'intestazione per decodificarlo e riprodurlo correttamente.

Le applicazioni che incorporano RTP (invece di uno schema proprietario per fornire tipo di payload, numero di sequenza o marcatura temporale) possono interagire con altri software di rete multimediali, consentendo agli utenti di comunicare tra loro anche se utilizzano prodotti sviluppati da aziende diverse. Nel Paragrafo 9.4.2 vedremo che RTP è spesso impiegato unitamente a SIP, un importante standard per la telefonia Internet.

Occorre sottolineare che RTP non fornisce alcun meccanismo per assicurare la spedizione tempestiva dei dati o altre forme di qualità del servizio e non garantisce nemmeno la consegna dei pacchetti né il loro ordine. Infatti, l'incapsulamento di RTP è visto solo dal punto terminale. I router non distinguono tra datagrammi IP che trasportano pacchetti RTP e altri datagrammi IP.

| Tipo di payload | Numero di sequenza | Marcatura temporale | Identificatore sorgente di sincronizzazione | Varie |
|-----------------|--------------------|---------------------|---------------------------------------------|-------|
|-----------------|--------------------|---------------------|---------------------------------------------|-------|

**Figura 9.8** Campi di intestazione RTP.

RTP consente di assegnare a ciascuna sorgente (una videocamera o un microfono) il proprio flusso indipendente di pacchetti. Per esempio, nel caso di una videoconferenza con due partecipanti, possono essere aperti quattro flussi RTP: due per l'audio (uno in ciascuna direzione) e due per il video. Tuttavia, molte tecniche di codifica diffuse (tra cui MPEG 1 e MPEG 2) uniscono audio e video durante la codifica, generando un solo flusso RTP per ciascuna direzione.

I pacchetti RTP non sono limitati alle applicazioni unicast, ma possono anche essere inviati su alberi multicast; nelle sessioni multicast molti a molti, tutti i trasmittenti inviano flussi RTP sullo stesso gruppo multicast. Questi flussi, come quelli audio e video emessi da più trasmittenti in una videoconferenza, costituiscono una **sessione RTP**.

### Intestazione dei pacchetti RTP

Come mostrato nella Figura 9.8 i quattro campi principali dell'intestazione del pacchetto RTP sono il tipo di payload (*payload type*), il numero di sequenza (*sequence number*), la marcatura temporale (*timestamp*) e l'identificatore della sorgente (*source identifier*).

Il campo tipo di payload è di 7 bit e nel caso di flussi audio indica la codifica impiegata (per esempio, PCM, modulazione delta adattativa o codifica lineare predittiva). Se il trasmittente decide di variare la codifica durante una sessione, per aumentare la qualità audio o per diminuire il tasso trasmissivo del flusso RTP, può informare il ricevente del cambiamento attraverso questo campo.

La Tabella 9.2 elenca alcuni tipi di payload audio previsti da RTP.

Per i flussi video, questo campo indica la codifica video (per esempio, JPEG dinamici, MPEG 1, MPEG 2 o H.261). Anche in questo caso, il trasmittente può variare la

**Tabella 9.2** Tipologia di payload audio supportato da RTP.

| Codice | Formato            | Frequenza | Banda      |
|--------|--------------------|-----------|------------|
| 0      | PCM "legge $\mu$ " | 8 kHz     | 64 kbps    |
| 1      | 1016               | 8 kHz     | 4,8 kbps   |
| 3      | GSM                | 8 kHz     | 13 kbps    |
| 7      | LPC                | 8 kHz     | 2,4 kbps   |
| 9      | G.722              | 16 kHz    | 48-64 kbps |
| 14     | Audio MPEG         | 90 kHz    | —          |
| 15     | G.728              | 8 kHz     | 16 kbps    |

**Tabella 9.3** Tipi di payload video supportato da RTP.

| Codice | Formato      |
|--------|--------------|
| 26     | JPEG         |
| 31     | H.261        |
| 32     | Video MPEG 1 |
| 33     | Video MPEG 2 |

codifica in corso di trasmissione nell’ambito di una sessione. La Tabella 9.3 elenca alcuni tipi di payload video supportati da RTP. Altri campi importanti sono i seguenti.

- *Numero di sequenza (16 bit)*. Il numero di sequenza è incrementato di un’unità per ogni pacchetto RTP inviato e può essere utilizzato dal ricevente per rilevare le perdite e ricostruire la sequenza dei pacchetti. Per esempio, se il flusso di pacchetti presenta al ricevente una lacuna fra 86 e 89, allora il ricevente sa che i pacchetti 87 e 88 sono mancanti e può tentare di recuperare i dati persi.
- *Marcatura temporale (32 bit)*. Riporta l’istante del campionamento del primo byte nel pacchetto dati RTP. La marcatura temporale, come abbiamo visto nel paragrafo precedente, deriva da un orologio di campionamento del trasmittente e il ricevente la può utilizzare per rimuovere il jitter dei pacchetti introdotto dalla rete e per fornire una riproduzione sincronizzata. Nel caso dell’audio, l’orologio è incrementato di un’unità a ogni campionamento (per esempio, ogni 125 µs per un campionamento a 8 kHz). Se l’applicazione audio genera blocchi costituiti da 160 campioni codificati, allora la marca temporale cresce di 160 per pacchetto RTP quando la sorgente è attiva. La marcatura temporale è incrementata con un tasso costante anche se la sorgente è inattiva.
- *Identificatore della sorgente di sincronizzazione (32 bit)*. Identifica la sorgente del flusso RTP. Di solito ogni flusso di una sessione RTP ha il proprio SSRC (*synchronization source identifier*). Questo indicatore non è l’indirizzo IP del trasmittente, ma un numero che la sorgente assegna arbitrariamente quando inizializza un nuovo flusso. La probabilità che a due flussi venga assegnato lo stesso SSRC è molto bassa. Qualora dovesse accadere, le due sorgenti sceglierrebbero un nuovo valore.

## 9.4.2 SIP

Il protocollo SIP (*session initiation protocol*), definito in [RFC 3261; RFC 5411] è un protocollo “leggero” e aperto che offre i seguenti servizi.

- Fornisce i meccanismi che consentono al chiamante di connettersi al chiamato su una rete IP e notificargli che vuole iniziare una chiamata; permette ai partecipanti di accordarsi sulle codifiche dei contenuti multimediali e di terminare le chiamate.

- Fornisce al chiamante i meccanismi necessari per determinare l'attuale indirizzo IP del chiamato. Gli utenti non hanno un unico indirizzo IP fisso, in quanto questo può essere assegnato dinamicamente (utilizzando DHCP) e gli utenti possono avere più dispositivi IP, ciascuno con un diverso indirizzo.
- Fornisce le procedure per la gestione della chiamata, durante la quale è possibile aggiungere nuovi flussi multimediali, cambiare la codifica, invitare nuovi partecipanti, trasferirla o metterla in attesa.

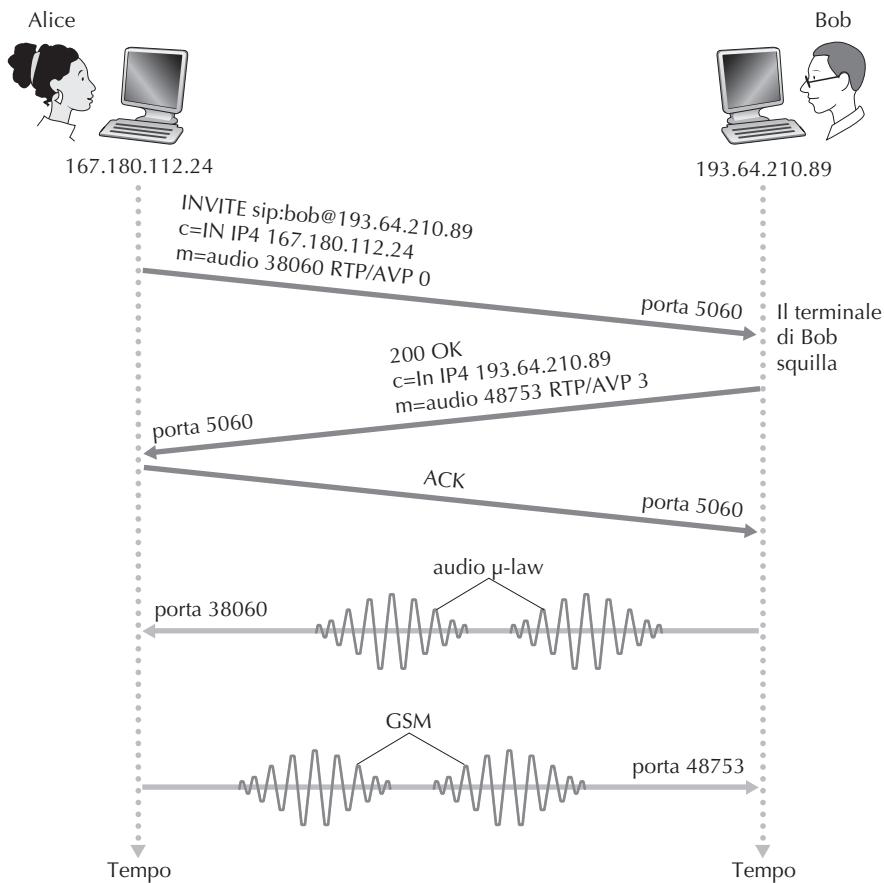
### Inizializzazione di una chiamata verso un indirizzo IP

Per capire l'essenza di SIP è opportuno ricorrere a un esempio. Supponiamo che Alice voglia chiamare Bob, che entrambi stiano lavorando col proprio PC e che dispongano del software SIP per fare e ricevere chiamate telefoniche. Inizialmente, assumeremo che Alice conosca l'indirizzo IP del PC di Bob.

Nella Figura 9.9 vediamo che la sessione SIP inizia quando Alice invia a Bob un messaggio INVITE, che assomiglia a un messaggio di richiesta HTTP, su UDP alla porta 5060 (i messaggi SIP possono anche essere inviati su TCP). Il messaggio INVITE comprende un identificativo di Bob (sip:bob@193.64.210.89) e le indicazioni concernenti l'attuale indirizzo IP di Alice, il fatto che vuole ricevere l'audio codificato nel formato AVP 0 (PCM codificato con sistema  $\mu$ -law) e incapsulato in RTP, e che i pacchetti RTP le devono pervenire sulla porta 38060. Dopo aver ricevuto il messaggio INVITE di Alice, Bob invia sulla porta 5060 un messaggio di risposta SIP, simile a quello HTTP, che comprende oltre alla stringa 200 OK le indicazioni relative al suo indirizzo IP, alle preferenze sulla codifica e sulla pacchettizzazione in ricezione, al numero di porta cui i pacchetti audio dovranno pervenire. Notiamo che nell'esempio Alice e Bob utilizzano differenti meccanismi di codifica audio: alla prima è richiesto di codificare l'audio con GSM, mentre il secondo deve utilizzare PCM con  $\mu$ -law. Dopo aver ricevuto la risposta di Bob, Alice gli invia un segnale di riscontro SIP. Dopo questa transazione, i due possono parlare. Per comodità grafica, nella Figura 9.11 Alice inizia a parlare dopo Bob, ma nella realtà la conversazione si svolge simultaneamente. A questo punto Bob codifica e impacchetta l'audio come richiesto e manda i pacchetti audio alla porta 38060 dell'indirizzo IP 167.180.112.24. Anche Alice codifica e impacchetta l'audio come richiesto e spedisce i pacchetti audio alla porta 48753 dell'indirizzo IP 193.64.210.89.

Da questo semplice esempio abbiamo appreso molte caratteristiche chiave di SIP. Primo, che è un protocollo fuori-banda (*out-of-band*): i messaggi SIP sono inviati e ricevuti su socket diverse da quelle utilizzate per inviare e ricevere i dati audio (o video). Secondo, i messaggi SIP sono in ASCII leggibile e assomigliano ai messaggi HTTP. Terzo, SIP richiede che tutti i messaggi abbiano un acknowledgement, quindi può funzionare sia con UDP sia con TCP.

Consideriamo ora come andrebbero le cose se Bob non disponesse di un codificatore PCM  $\mu$ -law. In questo caso, invece che con 200 OK, probabilmente avrebbe risposto con 600 Not Acceptable e avrebbe elencato nel messaggio tutte le codifiche che poteva utilizzare. Alice avrebbe quindi scelto una delle codifiche elencate e in-



**Figura 9.9** Istituzione di una chiamata SIP quando Alice conosce l'indirizzo IP di Bob.

viato un altro messaggio INVITE, con l'indicazione di quello prescelto. Oppure, Bob avrebbe potuto semplicemente non accettare la chiamata inviando un codice di risposta di rifiuto come "occupato", "fuori servizio", "servizio a pagamento" o "vietato".

### Indirizzi SIP

Nell'esempio precedente l'indirizzo SIP di Bob è `sip:bob@193.64.210.89`. Tuttavia, vorremmo che molti (se non la maggior parte) degli indirizzi SIP assomigliassero a quelli di posta elettronica come: `sip:bob@domain.com`. Quando il dispositivo SIP di Alice invia un messaggio INVITE, che include un indirizzo simile a quello della posta elettronica, l'infrastruttura SIP lo instrada al dispositivo IP attualmente utilizzato da Bob (come vedremo successivamente). Altre possibili forme di indirizzi SIP potrebbero essere il vecchio numero telefonico di Bob o semplicemente i dati anagrafici (assumendo che non ve ne siano altri uguali).

Una caratteristica interessante degli indirizzi SIP è che possono essere inclusi nelle pagine web, proprio come quelli di posta elettronica. Per esempio, supponiamo che

Bob disponga di una propria pagina personale, e che voglia fornire ai visitatori un mezzo per contattarlo. Potrebbe allora includere l'URL `sip:bob@domain.com` con cui viene lanciata l'applicazione SIP che consente di far pervenire a Bob il messaggio INVITE.

## Messaggi SIP

In questa breve introduzione non tratteremo tutti i tipi di messaggi e di intestazioni SIP, ma ci concentreremo sul messaggio INVITE, includendo alcune linee di intestazione. Supponiamo di nuovo che Alice voglia effettuare una chiamata VoIP a Bob, ma che questa volta conosca solo l'indirizzo SIP di Bob, `sip:bob@domain.com`, e non quello IP del dispositivo che Bob sta usando attualmente. In questo caso il messaggio sarà:

```
INVITE sip:bob@domain.com SIP/2.0
Via: SIP/2.0/UDP 167.180.112.24
From: sip:alice@hereway.com
To: sip:bob@domain.com
Call-ID: a2e3a@pigeon.here way.com
Content-Type: application/sdp
Content-Length: 885
c=IN IP4 167.180.112.24
m=audio 38060 RTP/AVP 0
```

La linea INVITE comprende la versione SIP, come nei messaggi di richiesta HTTP. Quando il messaggio SIP passa attraverso un dispositivo SIP (compreso quello che genera il messaggio), quest'ultimo inserisce un'intestazione via, che indica l'indirizzo IP del dispositivo. Più avanti vedremo che il tipico messaggio INVITE attraversa molti dispositivi SIP prima di raggiungere l'applicazione del ricevente. Come i messaggi di posta elettronica, anche quello SIP comprende due linee di intestazione: From e To. Il messaggio comprende, inoltre, un Call-ID, che identifica univocamente la chiamata (analogo al message-ID nella posta elettronica) e due linee d'intestazione, Content-Type e Content-Length, rispettivamente indicanti il formato e la lunghezza in byte del contenuto del messaggio SIP che viene quindi inserito dopo una linea vuota. Nel nostro specifico caso questo fornisce informazioni sull'indirizzo IP di Alice e sulla tipologia dell'audio.

## Traduzione dei nomi e localizzazione degli utenti

Nell'esempio della Figura 9.9 abbiamo assunto che il dispositivo SIP di Alice conoscesse l'indirizzo IP per contattare Bob. Questa ipotesi è abbastanza irrealistica, non soltanto perché gli indirizzi IP sono spesso assegnati dinamicamente con DHCP, ma anche perché Bob può avere vari dispositivi IP (per esempio, a casa, al lavoro e in auto). Supponiamo ora che Alice conosca solo l'indirizzo di posta elettronica di Bob, `bob@domain.com`, utilizzato anche per le chiamate SIP semplicemente apponendovi

il prefisso “sip:”. In questo caso, le occorre l’indirizzo IP del dispositivo che l’utente bob@domain.com sta utilizzando in quel momento. Per scoprirlo, crea un messaggio che inizia con INVITE sip:bob@domain.com SIP/2.0 e lo invia a un SIP proxy. Questo fornirà una risposta che potrebbe includere l’indirizzo IP del dispositivo o, in alternativa, l’indirizzo IP della sua casella vocale oltre all’URL di una pagina web in cui è scritto, per esempio, “Bob sta dormendo. Non disturbate!”. Il tipo di risposta potrebbe però variare in base al chiamante: così, se a voler contattare Bob fosse sua moglie, il proxy potrebbe accettare la chiamata e rispondere indicando l’indirizzo IP. Se a cercarlo fosse invece la suocera allora potrebbe rispondere con l’URL che punta alla pagina web.

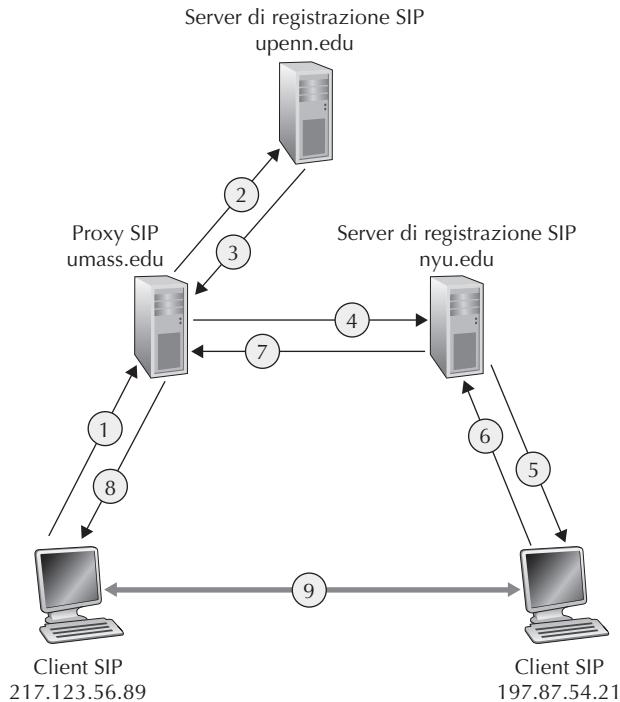
A questo punto vi starete probabilmente chiedendo in che modo il server proxy riesce a individuare l’attuale indirizzo IP di sip:bob@domain.com. Occorre sapere che a ciascun utente è associato un **server di registrazione SIP** (*SIP registrar*) al quale l’applicazione SIP su un dispositivo, quando viene lanciata, invia un messaggio di registrazione contenente l’attuale indirizzo IP presso cui l’utente può essere contattato. Per esempio, quando Bob lancia l’applicazione SIP sul suo palmare, questa invierà un messaggio simile al seguente:

```
REGISTER sip:domain.com SIP/2.0
Via: SIP/2.0/UDP 193.64.210.89
From: sip:bob@domain.com
To: sip:bob@domain.com
Expires: 3600
```

Il server di registrazione di Bob memorizza il suo attuale indirizzo IP. Quando Bob passa a un nuovo dispositivo SIP, questo invierà un nuovo messaggio di registrazione che indica un nuovo indirizzo IP. Se questo rimane invariato, il dispositivo SIP invia messaggi di aggiornamento (*refresh*) della registrazione, che indicano che l’indirizzo IP inviato più recentemente è ancora valido. Nell’esempio precedente i messaggi di refresh devono essere inviati ogni 3600 secondi per mantenere l’indirizzo nel server di registrazione. Va notato che il server di registrazione è simile a un DNS autoritativo: l’uno traspone gli identificativi fissi in linguaggio corrente (per esempio, sip:bob@domain.com) in indirizzi IP dinamici, l’altro traduce i nomi fissi degli host in indirizzi IP fissi. Spesso, server di registrazione SIP e proxy SIP sono eseguiti sulla stessa macchina.

Esaminiamo ora com’è possibile ottenere l’indirizzo di un utente. Dal precedente esempio notiamo che il server proxy di Alice deve solo inoltrare il messaggio INVITE a quello di Bob che lo inoltrerà al dispositivo SIP che Bob sta usando in questo momento; a questo punto, Bob potrà inviare una risposta SIP ad Alice.

Consideriamo la Figura 9.10 che illustra la procedura seguita da jim@umass.edu per avviare una sessione vocale IP con keith@upenn.edu. Assumiamo che il primo sia su 217.123.56.89 e il secondo su 197.87.54.21 e vediamo i passi che occorre eseguire. (1) Jim invia un messaggio INVITE al proxy SIP di umass; (2) questo fa una



**Figura 9.10** Creazione della sessione con coinvolgimento server proxy e di registrazione SIP.

ricerca DNS (non mostrata nella figura) dell’indirizzo dei server di registrazione SIP di upenn.edu e quindi inoltra il messaggio al server di registrazione. (3) Dato che keith@upenn.edu non è più presente nel server di upenn, quest’ultimo invia una risposta di redirezione, indicando che occorre cercare keith@nyu.edu. (4) Il proxy umass trasmette un INVITE al server di registrazione SIP di nyu (5) che conosce l’indirizzo IP di keith@nyu.edu e inoltra l’INVITE al terminale 197.87.54.21, su cui gira il client SIP di Keith. (6-8) Tramite i server di registrazione/proxy viene inviata una risposta SIP verso il client SIP sulla macchina 217.123.56.89. (9) I contenuti multimediali vengono scambiati direttamente tra i due client (il messaggio di riscontro SIP non è mostrato).

La nostra discussione è stata focalizzata sull’attivazione delle chiamate vocali, ma SIP è un protocollo di segnalazione che può essere impiegato anche per altri usi, quali le videoconferenze o le sessioni per l’elaborazione di testi, ed è diventato una componente essenziale in molte applicazioni di messaggistica istantanea. I lettori interessati ad approfondire la conoscenza di questo argomento possono visitare il sito web di Henning Schulzrinne [Schulzrinne-SIP 2016], sul quale troveranno anche software open source per client e server SIP [SIP Software 2016].

## 9.5 Supporto di Internet alle applicazioni multimediali

Nei Paragrafi 9.2-9.4 abbiamo visto come meccanismi a livello di applicazione possano essere usati dalle applicazioni multimediali per aumentare le loro prestazioni; abbiamo anche visto come si possono usare le CDN e le reti P2P per fornire un approccio a livello di sistema alla consegna dei contenuti multimediali. Tutte queste tecniche e approcci sono stati progettati per essere usati nell'attuale Internet che fa uso di best-effort; in realtà vengono usati proprio perché attualmente Internet fornisce solo una classe di servizio: quella best-effort. Ma, come progettisti di reti di calcolatori, non possiamo fare a meno di chiederci se una rete, piuttosto che le applicazioni o l'infrastruttura a livello di applicazione, possa fornire meccanismi per supportare la consegna dei contenuti multimediali. Come vedremo fra poco, la risposta, ovviamente, è sì. Ma vedremo anche che molti di questi meccanismi a livello di rete non sono ancora stati installati, sia per la loro complessità sia perché le tecniche a livello di applicazione, insieme al servizio best-effort e a un dimensionamento appropriato delle risorse di rete, quali la banda, possono effettivamente fornire un servizio di consegna multimediale end-to-end sufficientemente buono, anche se non sempre perfetto.

La Tabella 9.4 riassume tre grandi approcci per il supporto a livello rete delle applicazioni multimediali.

- Utilizzare al meglio il servizio best-effort.* I meccanismi a livello di applicazione e le infrastrutture appena viste sono adeguate quando la rete è ben dimensionata. Quando devono soddisfare una crescente domanda, gli ISP procedono di pari passo al potenziamento delle loro reti incrementando, in particolare, banda e capacità

**Tabella 9.4** Tre diversi approcci per supportare applicazioni multimediali a livello rete.

| Approccio                                    | Granularità                                         | Garanzie                                                     | Meccanismo                                                                                                      | Complessità | Adozione |
|----------------------------------------------|-----------------------------------------------------|--------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|-------------|----------|
| Utilizzare al meglio il servizio best-effort | Tutto il traffico ovunque trattato allo stesso modo | Nessuna o lasche                                             | Supporto al livello di applicazione CDN, overlay, erogazione, delle risorse a livello di rete                   | Minimale    | Ovunque  |
| Servizi differenziati                        | Classi di traffico differenti trattate diversamente | Nessuna o lasche                                             | Marcatura dei pacchetti, controllo del profilo di traffico, scheduling                                          | Media       | Moderata |
| QoS garantita per singola connessione        | Flussi individuali minimi trattati diversamente     | Lasche e stringenti, una volta che il flusso è stato ammesso | Marcatura dei pacchetti, controllo del profilo di traffico, scheduling, call admission e segnalazione di eventi | Bassa       | Minima   |

di commutazione in modo da limitare i ritardi e la perdita di dati [Huang 2005]. Discuteremo il **dimensionamento di una rete** nel Paragrafo 9.5.1.

- *Servizi differenziati.* Sin dall'inizio di Internet si pensava di assegnare a tipi di traffico diversi (come quelli indicati nel campo Tipo di Servizio dell'intestazione IPv4) differenti classi di servizio, invece della sola best-effort. Con i **servizi differenziati** un tipo di traffico, quale quello di un'applicazione di conversazione in tempo reale, potrebbe avere priorità rispetto a un altro quando entrambe sono in coda in un router. Tratteremo nei Paragrafi 9.5.2 e 9.5.3 i meccanismi per implementare i servizi differenziati, quali la marcatura dei pacchetti per indicare la classe di servizio, lo scheduling e altri ancora.
- *Garanzia di Qualità del Servizio (QoS, Quality of Service) per connessione.* Si intende che ogni istanza di un'applicazione prenota esplicitamente banda end-to-end e ha fissate garanzie di prestazione. Una **garanzia stringente (hard guarantee)** significa che un'applicazione riceverà sicuramente la qualità di servizio che ha richiesto, mentre una **garanzia lasca (soft guarantee)** significa che riceverà la qualità di servizio che ha richiesto con elevata probabilità. Se un utente volesse fare, per esempio, una chiamata telefonica tra l'host A e l'host B, allora l'applicazione telefonica Internet dell'utente dovrebbe essere in grado di riservare la banda esplicitamente in ciascun collegamento lungo tutto il percorso tra i due host. Tuttavia, permettere alle applicazioni di fare prenotazioni e richiedere alla rete di onorarle comporta grandi cambiamenti. Innanzitutto, è necessario un protocollo che riservi banda sul collegamento lungo il percorso tra mittente e ricevente per conto delle applicazioni. In secondo luogo, occorrerebbe modificare la politica di gestione delle code nei router, per tener conto delle prenotazioni. Terzo, per essere in grado di soddisfare le prenotazioni, le applicazioni devono fornire alla rete informazioni sulla quantità di dati che intendono inviare. La rete dovrebbe quindi disporre di un servizio di controllo per monitorare il traffico e verificare che corrisponda a quanto dichiarato. Infine, la rete deve poter determinare se dispone di sufficiente larghezza di banda per accettare nuove richieste di prenotazione. La combinazione di questi meccanismi richiede l'implementazione di nuovi e complessi programmi nei router. Poiché tale servizio è poco diffuso lo tratteremo solo brevemente nel Paragrafo 9.5.4.

### 9.5.1 Dimensionamento delle reti best-effort

Fondamentalmente la difficoltà nel supportare le applicazioni multimediali nasce dai loro stringenti requisiti sulle prestazioni (ritardo end-to-end dei pacchetti, jitter e perdite devono essere bassi) e dal fatto che questi problemi si verificano ogni volta che la rete diventa congestionata. Un approccio definitivo per migliorare la qualità delle applicazioni multimediali, che può essere spesso usato per risolvere qualsiasi problema dove le risorse sono limitate, è l'investimento di denaro per evitare contesa sulle risorse. Nel caso della multimedialità in rete, questo significa fornire sufficiente ca-

pacità sui collegamenti in tutta la rete, in modo che la congestione della rete e le sue conseguenze di ritardo e perdita dei pacchetti non capitino mai, o almeno molto raramente. Con sufficiente capacità dei collegamenti, i pacchetti potrebbero attraversare velocemente l'odierna Internet, senza ritardo di accodamento e perdite. Da molti punti di vista, questa è una situazione ideale: le applicazioni multimediali funzionerebbero perfettamente, gli utenti sarebbero contenti e tutto ciò potrebbe essere raggiunto senza cambiamenti all'architettura best-effort di Internet.

La domanda, certamente, è quanta capacità sia “sufficiente” per raggiungere questo paradiso e se i costi per fornire sufficiente banda siano ragionevoli dal punto di vista degli affari degli ISP. La domanda relativa a quanta capacità fornire ai collegamenti di rete in una certa topologia per raggiungere un determinato livello di prestazioni end-to-end è spesso nota come **fornitura di larghezza di banda** (*bandwidth provisioning*). L'ancora più complesso problema, relativo al progetto della topologia di rete (dove mettere i router, come interconnetterli e quanta capacità assegnare ai collegamenti) per raggiungere un certo livello di prestazioni end-to-end, è un problema di progettazione di rete, chiamato **dimensionamento della rete** (*network dimensioning*). La fornitura di larghezza di banda e il dimensionamento della rete sono argomenti complessi, ben oltre l'ambito di questo testo. Facciamo notare, tuttavia, che le seguenti questioni devono essere considerate al fine di prevedere le prestazioni a livello applicativo tra due punti terminali in rete e quindi fornire sufficiente capacità per soddisfare i requisiti prestazionali dell'applicazione.

- *Modelli di richiesta di traffico tra punti terminali della rete.* Può essere necessario specificare modelli sia a livello di chiamata (per esempio utenti che “entrano” nella rete e fanno partire applicazioni tra punti terminali) sia a livello di pacchetto (per esempio, i pacchetti che vengono generati dalle applicazioni in esecuzione). Notate che il carico di rete può cambiare nel tempo.
- *Requisiti di prestazioni ben definiti.* Per esempio, un requisito per supportare traffico sensibile al ritardo, come le applicazioni audio/video interattive, potrebbe essere che la probabilità che il ritardo end-to-end di un pacchetto sia maggiore di un ritardo massimo tollerabile sia minore di un (piccolo) valore prefissato [Fraleigh 2003].
- *Modelli per prevedere le prestazioni end-to-end per un dato modello di carico di lavoro e tecniche per trovare un'allocazione di banda a costo minimo, che si traduca nel soddisfacimento di tutti i requisiti dell'utente.* In quest'ambito i ricercatori sono impegnati nello sviluppo di modelli di accodamento, che possono quantificare le prestazioni per un dato carico di traffico e tecniche di ottimizzazione per trovare l'allocazione di banda a costo minimo che soddisfa i requisiti prestazionali.

Dato che l'odierna Internet best-effort potrebbe, da un punto di vista tecnologico, supportare traffico multimediale con prestazioni adeguate se avesse dimensioni tali per farlo, la domanda naturale è perché non farlo. Le risposte sono principalmente

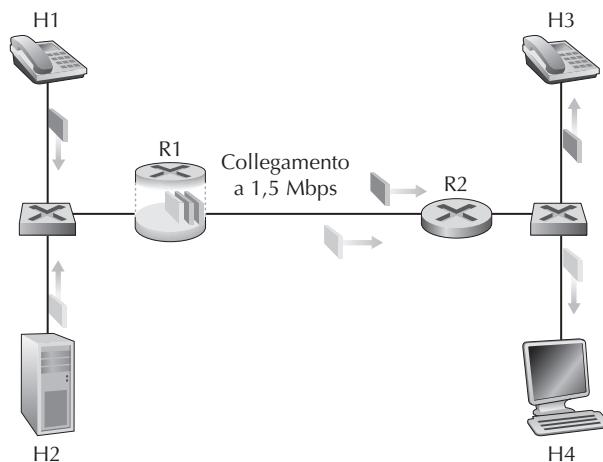
economiche e organizzative. Da un punto di vista economico, gli utenti sarebbero disposti a pagare i propri ISP, in modo tale che questi ultimi installino banda sufficiente per supportare applicazioni multimediali in Internet con un servizio best-effort? Le questioni amministrative sono ancora più scoraggianti. Dato che il percorso tra due punti terminali che fanno uso di multimedia passerà attraverso le reti di più ISP, da un punto di vista organizzativo, questi ISP dovrebbero essere disposti a cooperare (forse con una condivisione dei guadagni) per assicurare che il percorso tra i due nodi sia dimensionato per supportare le applicazioni multimediali? [Davies 2005] fornisce una prospettiva su queste questioni economiche e organizzative, mentre [Fraleigh 2003] offre una panoramica del dimensionamento di reti dorsali di primo livello per supportare traffico sensibile ai ritardi.

### 9.5.2 Fornitura di più classi di servizio

Un semplice modello di servizio migliore del best-effort consiste nel dividere il traffico in classi e fornire loro diversi livelli di servizio, a seconda della classe. Per esempio, un ISP potrebbe voler fornire, a ragion veduta, una classe di servizio più alta al traffico sensibile al ritardo come VoIP e di teleconferenza (e quindi far pagare di più questo servizio), rispetto al traffico elastico come e-mail o HTTP. In alternativa, un ISP potrebbe semplicemente essere interessato a una qualità di servizio migliore per i clienti che sono disposti a pagare di più. Un certo numero di ISP che forniscono accesso residenziale cablato o cellulare ha adottato questo livello di servizio stratificato.

Nella nostra vita di tutti i giorni siamo abituati alle diverse classi di servizio: i passeggeri in prima classe su una linea aerea hanno diritto a un servizio migliore di quelli che volano in classe economica, i VIP hanno la possibilità di accedere immediatamente a quegli eventi per cui tutti devono fare la coda, gli anziani sono riveriti in alcuni paesi e a essi sono riservati i posti d'onore e il cibo migliore a tavola. È importante notare come questo tipo di servizi differenziati sia fornito a un aggregato di traffico, cioè a classi di traffico, e non a singole connessioni. Per esempio, tutti i passeggeri di prima classe sono trattati allo stesso modo e nessuno riceve un trattamento migliore degli altri, esattamente come tutti i pacchetti VoIP riceveranno lo stesso trattamento all'interno della rete, indipendentemente dalla particolare connessione alla quale appartengono. Come vedremo, trattando con un numero ridotto di aggregati di traffico, piuttosto che con un gran numero di connessioni singole, il nuovo meccanismo di rete, richiesto per fornire un servizio migliore di quello best-effort, può essere mantenuto relativamente semplice.

I primi progettisti di Internet avevano chiara in mente questa nozione di classi di servizio: ricordiamo il campo Tipo di Servizio (TOS) nell'intestazione IPv4 discusso nel Capitolo 4. IEN123 [ISI 1979] spiega che il campo TOS era presente anche nell'antenato del datagramma IPv4: “Il campo tipo di servizio fornisce un’indicazione dei parametri astratti della qualità di servizio desiderata. Questi parametri sono da usare per guidare la selezione dei parametri di servizio effettivi, quando si trasmette un datagramma attraverso una particolare rete. Molte reti offrono il servizio di precedenza, che in un qualche modo tratta il traffico con precedenza più alta come più



**Figura 9.11** Concorrenza tra applicazioni audio e HTTP.

importante rispetto al resto del traffico.” Anche trent’anni fa la visione della fornitura di diversi livelli di servizio a differenti livelli di traffico era chiara. Tuttavia c’è voluto un periodo altrettanto lungo per realizzare questa visione.

### Scenari di riferimento

Cominciamo la nostra discussione sui meccanismi di rete per fornire più classi di servizio introducendo qualche scenario di riferimento.

La Figura 9.11 mostra un semplice scenario: supponiamo che due flussi di pacchetti applicativi abbiano origine dagli host H1 e H2 e siano destinati agli host H3 e H4, collocati su due LAN diverse, e che i rispettivi router dispongano di un collegamento a 1,5 Mbps. Assumiamo che le capacità trasmissive delle LAN siano significativamente più alte e concentriamo la nostra attenzione sulla coda in uscita dal router R1; è qui che si verificheranno ritardi e perdite di pacchetti se il tasso aggregato di invio di H1 e H2 supererà 1,5 Mbps. Supponiamo inoltre che un’applicazione audio a 1 Mbps (per esempio, con qualità CD) condivida il collegamento a 1,5 Mbps fra R1 e R2 con un’applicazione web HTTP che sta trasferendo una pagina web da H2 a H4.

Nella filosofia best-effort Internet mescola i pacchetti audio e HTTP nella coda in uscita da R1 e (generalmente) li trasmette nell’ordine di arrivo (FIFO). In questo scenario, una raffica di pacchetti dalla sorgente HTTP può potenzialmente riempire la coda, provocando un grande ritardo o la perdita di alcuni pacchetti audio a causa della saturazione del buffer in R1. Come possiamo risolvere questo problema? Dato che l’applicazione HTTP non ha vincoli temporali, potremmo pensare di dare la precedenza ai pacchetti audio in R1. Con una modalità di scheduling governata unicamente dalla priorità, i pacchetti audio nel buffer di uscita da R1 dovrebbero sempre essere trasmessi prima di quelli HTTP. L’intera banda del collegamento da R1 a R2 sarebbe dedicata al traffico audio e, solo una volta che si sia esaurita la coda nel buffer, HTTP potrebbe iniziare a trasmettere. Affinché R1 possa distinguere fra traffico audio e

HTTP, ciascun pacchetto deve essere contrassegnato come appartenente a una delle due classi. Questo era l’obiettivo originale del campo ToS (*type-of-service*) di IPv4. Per quanto ovvio possa sembrare, questo è il primo principio richiesto per fornire classi di traffico diverse:

**Principio 1. La marcatura dei pacchetti** (*packet marking*) consente ai router di distinguerli in base alla loro classe di traffico.

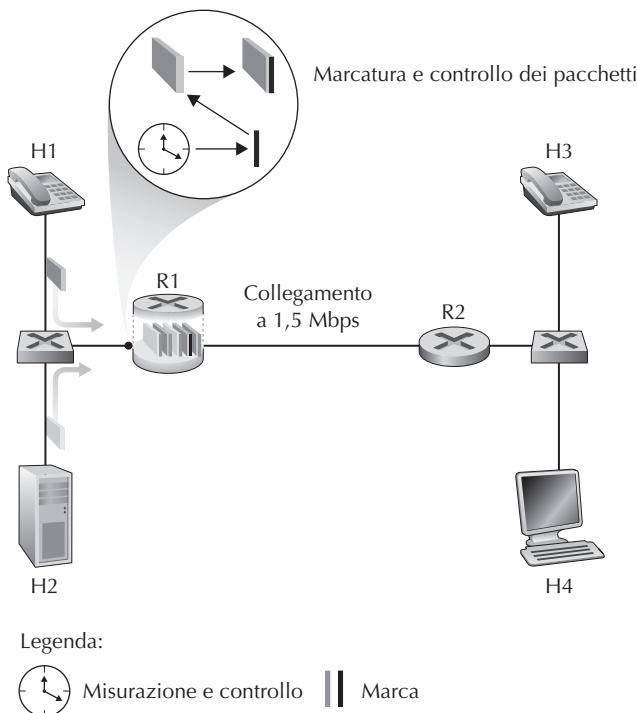
Si noti che, nonostante questo esempio consideri flussi multimediali e non in competizione, lo stesso principio può essere applicato a tipi di servizio differenziati dalla tariffazione: un sistema di marcatura dei pacchetti è ancora necessario per indicare a quale classe un pacchetto appartiene.

Ipotizziamo ora che, grazie a meccanismi che studieremo nei paragrafi successivi, il router debba dare priorità ai pacchetti dell’applicazione audio a 1 Mbps. Essendo la capacità trasmissiva del collegamento in uscita di 1,5 Mbps, anche se ai pacchetti HTTP fosse attribuita una bassa priorità, usufruirebbero comunque, in media, di 0,5 Mbps. Che cosa accadrebbe, allora, se l’applicazione audio, volutamente o per errore, iniziasse a inviare pacchetti a un tasso di 1,5 Mbps o più? In questo caso, i pacchetti HTTP non riceveranno alcun servizio sul collegamento da R1 a R2. Problemi simili si verificherebbero se varie applicazioni con la stessa classe di servizio dell’applicazione audio dovessero spartirsi la banda del collegamento; potrebbero congiuntamente bloccare la sessione HTTP. Da un punto di vista ideale, sarebbe quindi auspicabile una forma di isolamento tra le classi di traffico. Tale protezione può essere implementata in varie parti della rete: a ogni router, quando entra nella rete o ai confini del dominio. Queste considerazioni portano al secondo principio.

**Principio 2.** È auspicabile che sia fornito un **grado di isolamento tra le classi di traffico**, in modo che una classe non subisca gli effetti negativi derivanti dal comportamento non conforme di un’altra.

Esamineremo più avanti numerose tecniche di isolamento specifiche delle classi di traffico; per ora ci limitiamo a osservare che si possono seguire due differenti approcci. Il primo è quello di stabilire dei controlli sul profilo del traffico (traffic policing) (Figura 9.12). Se una classe di traffico o un flusso deve rispettare certi criteri (per esempio, il flusso audio non superi la soglia di 1 Mbps) allora possiamo mettere in piedi un sistema di controllo per assicurarcici il rispetto di quanto stabilito e che, qualora un’applicazione si comporti in modo non conforme, intervenga per ripristinare la situazione ottimale, per esempio ritardando o scartando i pacchetti che stanno violando i criteri. Il meccanismo di leaky bucket (letteralmente, secchio bucato) che esamineremo tra poco è forse il sistema di controllo del profilo di traffico più diffuso. Nella Figura 9.12 classificazione e marcatura dei pacchetti (Principio 1) e di policing (Principio 2) sono implementate entrambe al bordo della rete: in un sistema periferico o un router di bordo.

Un approccio alternativo per fornire isolamento tra classi di traffico prevede che lo scheduling dei pacchetti a livello di collegamento assegni esplicitamente a ciascuna classe una porzione fissa di larghezza di banda. Per esempio, in R1, alla classe della

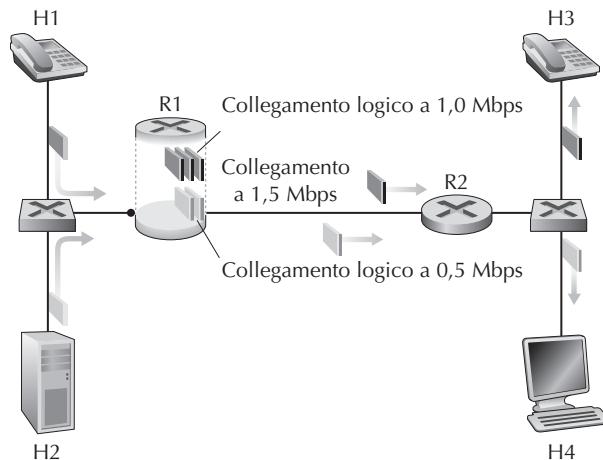


**Figura 9.12** Controllo del profilo (e marcatura) dei flussi di traffico audio e HTTP.

trasmissione audio potrebbe essere dedicato 1 Mbps e alla classe HTTP la restante banda di 0,5 Mbps. In questo caso, i flussi vedono due collegamenti logici, con capacità di 1 e di 0,5 Mbps, rispettivamente (Figura 9.13). In presenza di un controllo stringente della larghezza di banda a livello di collegamento, una classe può usare solo la larghezza di banda assegnata, anche se non vi sono altre applicazioni attive. Così, nei momenti in cui viene temporaneamente sospeso l'invio di pacchetti audio (per esempio, quando chi parla rimane in silenzio), il flusso HTTP non potrà comunque usare più di 0,5 Mbps sul collegamento da R1 a R2, anche se la banda di 1 Mbps allocata per il flusso audio non viene usata in quel momento. Siccome la banda è una risorsa che non può essere messa da parte, non c'è ragione di impedire a HTTP di usare la banda lasciata libera dal traffico audio. Vorremmo quindi usare la banda nella maniera più efficiente possibile, senza sprecarla quando è disponibile. Questo ci porta al terzo principio:

**Principio 3.** È auspicabile che l'utilizzo delle risorse (per esempio, buffer e larghezza di banda) sia quanto più efficiente possibile anche in presenza di isolamento delle classi.

Nei Paragrafi 1.3 e 4.3 abbiamo visto che i pacchetti appartenenti a diversi flussi sono riuniti e accodati nei buffer di uscita associati a ogni collegamento in attesa di essere



**Figura 9.13** Isolamento logico delle classi di traffico audio e HTTP.

trasmessi. La procedura con cui i pacchetti sono selezionati per la trasmissione sul collegamento è nota come **disciplina di scheduling del collegamento** (*link-scheduling discipline*) ed è stata trattata nel Paragrafo 4.2. In particolare sono stati trattati i modelli FIFO, a coda di priorità e l'accodamento equo ponderato che, come vedremo a breve, gioca un ruolo importante nell'isolare le classi di traffico.

### Policing con leaky bucket

Uno dei precedenti principi era che il monitoraggio (*policing*), vale a dire il controllo del tasso con cui una classe o un flusso (nella nostra trattazione seguente assumiamo che l'unità per il monitoraggio sia il flusso) immette i pacchetti nella rete sia un importante meccanismo per garantire la QoS. A questo fine possiamo identificare tre criteri basilari nelle procedure di policing, stabiliti in base alla scala temporale con la quale il flusso viene monitorato.

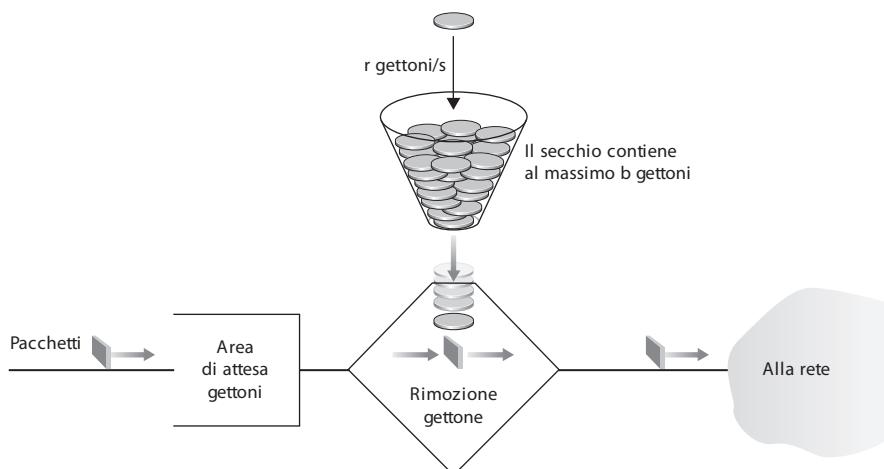
- **Tasso medio.** La rete può voler limitare la frequenza trasmittiva media, ossia il numero di pacchetti in un dato intervallo di tempo a lungo termine, alla quale il flusso può essere inviato. Un aspetto cruciale è qui rappresentato dall'intervallo di tempo rispetto al quale calcolare il tasso medio. Un flusso con una frequenza trasmittiva media vincolata a 100 pacchetti al secondo subisce maggiori restrizioni rispetto a una sorgente il cui limite è di 6000 pacchetti al minuto, anche se entrambi presentano la stessa media. Nel secondo caso, infatti, il flusso può essere inviato, nei momenti di picco, anche a un tasso di 1000 pacchetti al secondo, purché nell'arco di un minuto non siano complessivamente superati i 6000 pacchetti, comportamento non consentito dai limiti imposti al primo caso.
- **Tasso di picco.** Mentre il vincolo sul tasso medio indica la quantità massima di traffico calcolata su un periodo relativamente lungo, quello di picco pone un limite al massimo numero di pacchetti che può essere inviato in un breve lasso di tempo.

Riferendoci all'esempio precedente, potremmo dire che la rete può imporre al flusso un tasso medio di 6000 pacchetti al minuto con il vincolo di non superare, in alcun caso, i 1500 pacchetti al secondo.

- *Dimensione di una raffica (burst size).* La rete potrebbe anche voler limitare il numero massimo di pacchetti che possono essere inviati in un lasso di tempo ancora più breve. Al limite, quando la lunghezza dell'intervallo tende a zero, la dimensione della raffica vincola il numero di pacchetti che possono essere trasmessi istantaneamente. Anche se è impossibile immettere istantaneamente in rete più di un pacchetto, in quanto non si può superare il tasso trasmissivo fisico, indicare una massima dimensione della raffica può ugualmente risultare utile.

Per monitorare il rispetto dei limiti imposto a un flusso di pacchetti si può ricorrere a un meccanismo di policing detto **leaky bucket** (“secchio bucato”), un’astrazione costituita da un recipiente in grado di contenere fino a  $b$  gettoni (*token*) (Figura 9.14); questi sono generati al tasso di  $r$  gettoni al secondo e quindi immessi nel contenitore. Se il secchio contiene un numero di gettoni inferiore a  $b$  allora il nuovo gettone può essere aggiunto agli altri, altrimenti viene scartato e il contenitore rimane con  $b$  gettoni.

Facciamo ora un esempio che ci può aiutare a comprenderne il funzionamento. Supponiamo che, per essere immesso nella rete, il pacchetto debba prendere un gettone dal contenitore. Se il contenitore dei gettoni è vuoto si pongono due alternative: il pacchetto aspetta un nuovo gettone oppure viene scartato, eventualità, quest’ultima, che per semplicità espositiva non prendiamo in considerazione. Dato che il recipiente può contenere fino a  $b$  gettoni, la massima dimensione di una raffica è di  $b$  pacchetti. Inoltre, essendo  $r$  il tasso di generazione dei gettoni, il massimo numero di pacchetti che possono essere trasmessi in qualsiasi intervallo di tempo di lunghezza  $t$  è uguale a  $rt + b$ . Quindi, il tasso con cui vengono generati i gettoni serve a limitare il tasso medio a lungo termine con cui i pacchetti possono essere immessi nella rete. Metten-



**Figura 9.14** Controllo del profilo di traffico tramite leaky bucket.

do in serie due leaky bucket è anche possibile utilizzare questo meccanismo per regolare il tasso di picco. Questo argomento verrà ripreso nei problemi elencati al termine del capitolo.

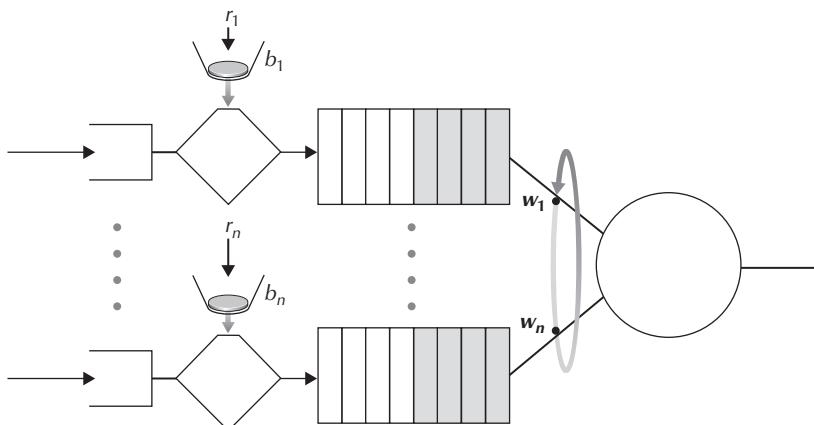
### Leaky bucket + WFQ = ritardo massimo dimostrabile in una coda

Concludiamo questo paragrafo prendendo in considerazione il collegamento di uscita da un router che combina  $n$  flussi, ciascuno controllato tramite un leaky bucket con parametri  $b_i$  e  $r_i$  (con  $i = 1, \dots, n$ ) e con lo scheduling WFQ (Capitolo 4). In questo caso, useremo il termine flusso in modo improprio per indicare l'insieme di pacchetti che non possono essere distinti dallo scheduler: in pratica, il flusso può essere costituito sia dal traffico di una sola connessione end-to-end, sia da quello di più connessioni (Figura 9.15).

Ricordiamo che a ciascun flusso  $i$  viene garantita una porzione della larghezza di banda uguale o maggiore di  $R \cdot w_i / (\sum w_j)$ , dove  $R$  è il tasso trasmissivo del collegamento in pacchetti al secondo. Qual è allora il massimo ritardo di cui risentirà un pacchetto mentre attende il servizio in WFQ, cioè dopo essere passato attraverso il controllo del leaky bucket? Supponiamo che il contenitore del flusso 1 sia inizialmente pieno e che arrivi una raffica di  $b_1$  pacchetti. Questi rimuovono tutti i gettoni dal secchio (senza alcuna attesa) e poi si uniscono nell'area di attesa della WFQ riservata al flusso 1. Nonostante i  $b_1$  pacchetti siano serviti a un tasso almeno pari a  $R \cdot w_1 / (\sum w_j)$  pacchetti al secondo, l'ultimo accumulerà un ritardo complessivo massimo,  $d_{\max}$ , prima che la sua trasmissione sia terminata,

$$d_{\max} = \frac{b_1}{R \cdot w_1 / \sum w_j}$$

Dalla formula si evince che in presenza di  $b_1$  pacchetti serviti (o rimossi dalla coda) a un tasso almeno pari a  $R \cdot w_1 / (\sum w_j)$  pacchetti al secondo, il tempo necessario per trasmettere fino all'ultimo bit dell'ultimo pacchetto non possa superare  $b_1 / (R \cdot w_1 / (\sum w_j))$ .



**Figura 9.15**  $n$  flussi leaky bucket in multiplexing con uno scheduling WFQ.

In uno dei problemi elencati a fine capitolo vi si chiederà di provare che se  $r_1 < R \cdot w_1 / (\sum w_j)$ , allora  $d_{\max}$  è effettivamente il massimo ritardo che ciascun pacchetto del flusso 1 può accumulare nella coda WFQ.

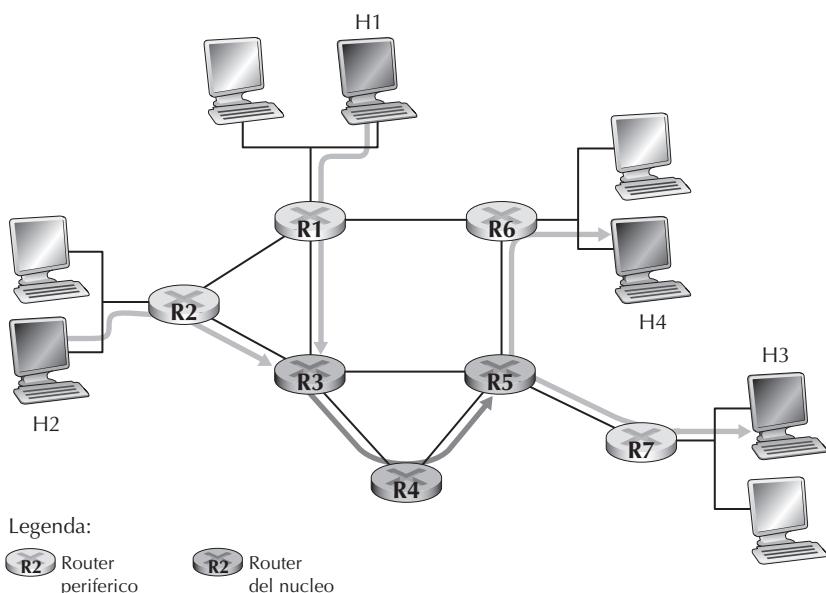
### 9.5.3 Diffserv

Avendo visto motivazioni, principi e meccanismi specifici per fornire diverse classi di servizio, tiriamo le fila del discorso con un esempio.

L'architettura Internet Diffserv (*differentiated service*, servizi differenziati) [RFC 2475; Kilkki 1999] fornisce una differenziazione dei servizi, vale a dire, la possibilità di gestire in maniera scalabile differenti classi di traffico in maniere distinte su Internet. La necessità di un servizio scalabile nasce dal fatto che ai router delle dorsali possono giungere milioni di flussi di traffico simultanei. In seguito vedremo come questa esigenza sia soddisfatta collocando alcune semplici funzionalità nel nucleo della rete e implementando le operazioni di controllo più complesse in periferia.

Iniziamo la nostra trattazione dalla semplice rete mostrata nella Figura 9.16. Nel corso di questo paragrafo descriveremo uno degli eventuali impieghi di Diffserv; sono possibili, infatti, molte variazioni, come descritto nell'RFC 2475. L'architettura Diffserv è costituita da due gruppi di elementi funzionali.

- Funzioni periferiche: *classificazione dei pacchetti e condizionamento del traffico*. All'ingresso della rete (cioè, o nell'host Diffserv-compatibile che genera traffico o nel primo router incontrato sul percorso tra sorgente e destinazione), i pacchetti sono contrassegnati con un dato valore nel campo DS dell'intestazione IPv4 o IPv6 [RFC 3260]. La definizione del campo DS intende sostituire le precedenti



**Figura 9.16** Esempio di rete Diffserv.

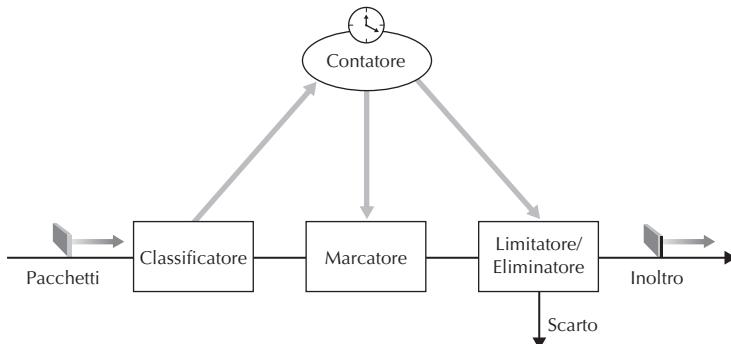
definizioni del campo ToS in IPv4 e del campo relativo alle classi di traffico di IPv6, discusse nel Capitolo 4. Per esempio (Figura 9.16), i pacchetti spediti da H1 a H3 potrebbero essere contrassegnati con R1, e quelli da H2 a H4 con R2. Diverse classi di traffico riceveranno servizi diversi nel nucleo della rete.

- Funzioni interne: *inoltro*. Quando un pacchetto, con marcatura DS, giunge a un router DiffServ-compatibile viene inoltrato in base al cosiddetto comportamento ad ogni hop (*PHB, per-hop behavior*) associato alla classe del pacchetto. Questa funzione determina come buffer e larghezza di banda sono condivisi dalle classi di traffico. Un principio basilare dell’architettura DiffServ è che il PHB di un router è esclusivamente basato sulla marcatura del pacchetto, cioè sulla classe di traffico cui appartiene. Quindi, se i pacchetti spediti da H1 a H3 ricevono la stessa marcatura di quelli da H2 a H4, allora i router li trattano come un gruppo, senza distinguere tra i pacchetti originati da H1 o da H2 (Figura 9.16). Per esempio, R3 potrebbe non distinguere fra quelli provenienti da H1 o da H2 nel momento in cui li inoltra verso R4. Quindi, l’architettura del servizio differenziato non prevede la gestione dello stato dei router per coppie individuali sorgente-destinazione. Ciò è importante per i requisiti di scalabilità affrontati all’inizio di questo paragrafo.

Per meglio chiarire questo concetto faremo ricorso a un’analogia. Consideriamo un evento di vasta risonanza (come un congresso, un concerto, una finale di coppa), in cui gli spettatori ricevono un pass suddiviso per categorie: Vip, giovani, stampa e semplice ingresso per tutti gli altri. I lasciapassare sono generalmente distribuiti alla biglietteria, vale a dire in una zona periferica rispetto all’area in cui si svolge la manifestazione, e indicano i posti e i privilegi assegnati. È qui, ai bordi della rete, dove vengono eseguite le operazioni più pesanti computazionalmente, come il pagamento per l’ingresso, la verifica del tipo corretto di invito e la corrispondenza dell’invito a un identificativo. Inoltre, potrebbe essere stabilito un limite a seconda delle categorie, per il numero di persone ammesse all’evento: alcuni dovranno quindi attendere prima che venga consentito loro l’accesso. Una volta all’interno, poi, i partecipanti riceveranno un servizio differenziato in base alla tipologia del loro pass: poltrone numerate e aree esclusive riservate ai Vip, un trattamento più spartano con vincoli di spostamento circoscritti a settori ubicati in posizioni meno comode per gli altri. In entrambi i casi, il servizio ricevuto dipende dal tipo di lasciapassare e tutti i membri di una classe sono trattati allo stesso modo.

Osserviamo ora la Figura 9.17 che illustra la struttura logica della classificazione e della funzione di assegnamento della marcatura nei router di bordo. I pacchetti che pervengono ai router periferici vengono innanzitutto selezionati in base al valore di uno o più campi di intestazione (per esempio, l’indirizzo sorgente, l’indirizzo di destinazione, la porta sorgente, la porta di destinazione o il protocollo encapsulato) e indirizzati alla marcatura.

In alcuni casi l’utente può aver accettato un limite al tasso trasmissivo dei pacchetti per essere conforme al **profilo di traffico** (*traffic profile*) dichiarato. Questo potrebbe contenere un valore massimo stabilito per il tasso di picco, o le caratteristi-



**Figura 9.17** Esempio di rete Diffserv.

che delle raffiche dei pacchetti, come abbiamo visto precedentemente con il meccanismo del leaky bucket. Finché l’utente invia pacchetti nella rete in modo conforme al profilo di traffico negoziato, i pacchetti ottengono la marcatura di priorità e sono inoltrati sul loro percorso verso la destinazione, mentre quelli che non rispettano il profilo potrebbero ricevere un diverso trattamento: per esempio, essere ritardati se non addirittura scartati. La **funzione di conteggio** (*metering function*) nella Figura 9.17 serve a confrontare il flusso dei pacchetti in arrivo con il profilo di traffico negoziato per verificarne la concordanza. L’effettiva decisione se marcare, instradare, ritardare o scartare un pacchetto è una politica di gestione determinata dall’amministratore di rete e non è specificata nell’architettura Diffserv.

Il secondo componente chiave dell’architettura Diffserv riguarda il PHB fornito dai router Diffserv-compatibili. In modo piuttosto criptico, ma accurato, PHB è definito come “una descrizione dell’instradamento osservabile dall’esterno di un nodo Diffserv, applicato a un particolare aggregato di comportamento Diffserv” [RFC 2475]. Esaminando questa definizione possiamo trarne alcune importanti considerazioni.

- Un PHB può fornire diverse prestazioni (cioè, distinti comportamenti di instradamento osservabili dall’esterno) a differenti classi di traffico.
- Un PHB definisce diverse prestazioni (comportamenti) per le classi, ma non impone alcuna particolare procedura per raggiungere questi comportamenti. Purché sia soddisfatto il criterio dell’osservabilità esterna, può essere utilizzata qualunque tecnica e qualsiasi politica di allocazione di buffer e larghezza di banda. Per esempio, un PHB non richiede che per raggiungere un particolare comportamento venga utilizzato un determinato criterio di accodamento dei pacchetti, per esempio con priorità, WFQ o FIFO. Il PHB è il “fine”, mentre l’allocazione delle risorse e i dispositivi di implementazione sono il “mezzo”.
- Le differenze nelle prestazioni devono essere osservabili e quindi misurabili.

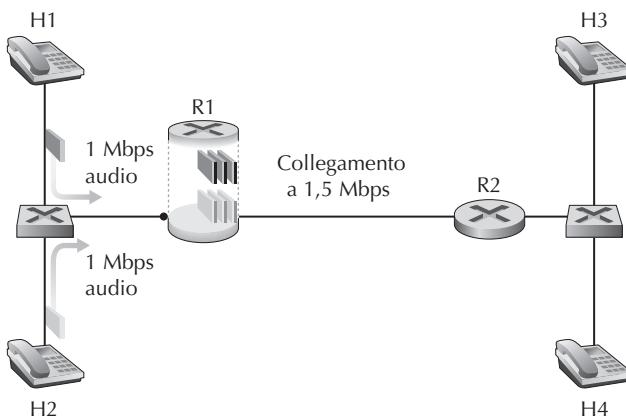
Attualmente sono state definite due tipologie di PHB: expedited forwarding (EF, inoltro rapido) [RFC 3246] e assured forwarding (AF, inoltro assicurato) [RFC 2597]. **Expedited forwarding** specifica che il tasso trasmisivo di una classe di traffico dal router deve essere uguale o superiore a un valore prestabilito, mentre **assured forwarding** suddivide il traffico in quattro classi, dove a ciascuna classe AF è garantita la fornitura di un quantitativo minimo di banda e di memorizzazione nei buffer.

Concludiamo la nostra discussione su Diffserv con alcune considerazioni sul suo modello di servizio. Nella nostra precedente trattazione abbiamo assunto implicitamente che l’architettura Diffserv sia installata nell’ambito di un singolo dominio amministrativo, anche se tipicamente un servizio end-to-end deve attraversare più di un ISP. Per poter erogare servizi Diffserv end-to-end, i vari ISP ubicati tra i sistemi periferici non solo devono fornire servizi Diffserv, ma dovranno anche cooperare e accordarsi per assicurare agli utenti finali un effettivo servizio differenziato end-to-end. Senza questo tipo di collaborazione si troverebbero continuamente a ripetere: “Sappiamo che pagate di più, ma non abbiamo stabilito un accordo sui servizi con l’ISP che ha bloccato il vostro traffico. Siamo spiacenti per le numerose lacune presenti nella vostra chiamata VoIP.” In secondo luogo, se la rete usasse Diffserv e avesse un carico moderato per la maggior parte del tempo, non si riscontrerebbe una differenza sensibile tra il servizio best-effort e quello Diffserv. Effettivamente, oggi, il ritardo end-to-end è originato più dai tassi trasmisivi delle reti di accesso e dal numero di hop, piuttosto che dai ritardi nelle code dei router. Immaginiamo il disappunto di un cliente Diffserv che, avendo pagato per un servizio extra, scopra che il più economico servizio best-effort ha quasi sempre prestazioni identiche al suo.

#### **9.5.4 Fornire garanzie di qualità del servizio (QoS) per ogni connessione: prenotazione delle risorse e ammissione delle chiamate**

Nel paragrafo precedente abbiamo visto che la marcatura dei pacchetti e il controllo del profilo di traffico, l’isolamento del traffico e lo scheduling a livello di collegamento possono fornire a una classe di servizio migliori prestazioni di un’altra. Con certe politiche di scheduling, come quelle a priorità, le classi di traffico a bassa priorità sono fondamentalmente invisibili a quelle ad alta priorità. Con il corretto dimensionamento della rete, le classi ad alta priorità possono effettivamente raggiungere valori di ritardo e perdita dei pacchetti estremamente bassi, con prestazioni in fondo analoghe alla commutazione di circuito. Tuttavia, la rete non può garantire che un flusso in corso in una classe di traffico ad alta priorità continuerà a ricevere quel tipo di servizio per tutta la durata del flusso usando solo il meccanismo che abbiamo descritto poco fa. In questo paragrafo vedremo perché sono necessari altri meccanismi di rete aggiuntivi per fornire qualità del servizio garantita alle singole connessioni.

Torniamo ai nostri scenari del Paragrafo 9.5.2 e consideriamo due applicazioni audio da 1 Mbps su un collegamento a 1,5 Mbps (Figura 9.18). I due flussi di dati combinati (2 Mbps) superano la capacità del collegamento; anche facendo ricorso a



**Figura 9.18** Due applicazioni audio concorrenti che sovraccaricano il collegamento da R1 a R2...

quanto precedentemente previsto (classificazione, marcatura, isolamento dei flussi e condivisione della banda non utilizzata), chiaramente qui c'è ben poco da fare. La larghezza di banda non è sufficiente per soddisfare contemporaneamente le necessità delle due applicazioni che, se anche ottenessero un'equa suddivisione, perderebbero il 25% dei pacchetti trasmessi. La qualità del servizio risulterebbe così talmente bassa da renderle inutilizzabili.

Appurato che le due applicazioni non possono essere soddisfatte contemporaneamente, che cosa dovrebbe fare la rete? Consentendo a entrambe di procedere con una QoS inutilizzabile, sprecherebbe risorse di rete in flussi applicativi che alla fine non fornirebbero alcuna utilità all'utente finale. La risposta è chiara: uno dei due flussi applicativi va bloccato, cioè gli deve essere negato l'accesso alla rete, mentre all'altro dovrebbe essere consentito di continuare, usando tutta la banda di 1 Mbps necessaria per l'applicazione. La rete telefonica ne è un esempio: quando non è possibile allocare la risorsa richiesta (un circuito), viene negato alla chiamata l'ingresso nella rete e l'utente riceve un segnale di occupato. Dal precedente esempio risulta chiaro che consentire a un flusso di accedere alla rete senza garantirgli la necessaria QoS non offre alcun vantaggio agli utenti e costituisce un ingiustificato spreco di risorse.

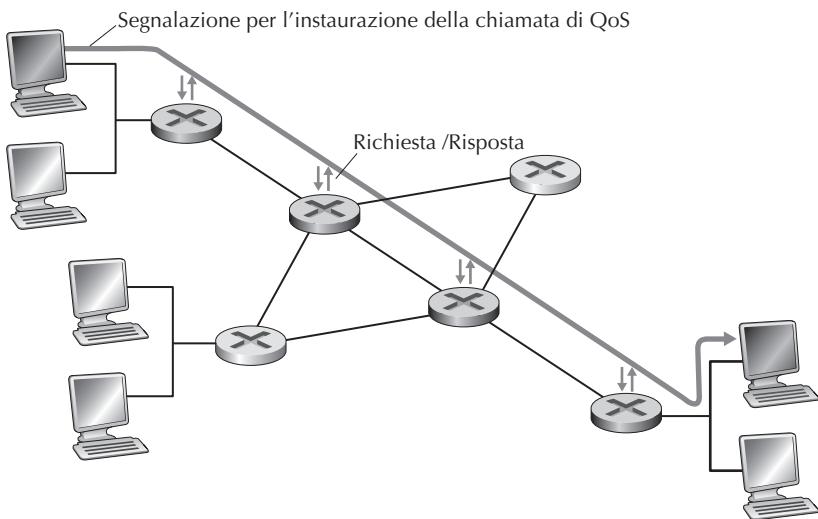
Ammettendo o bloccando esplicitamente un flusso, in base ai requisiti sulle risorse e ai requisiti delle sorgenti dei flussi già ammessi, la rete può garantire a un flusso ammesso che sarà in grado di ricevere la QoS richiesta. Occorre quindi che il flusso indichi quali sono i requisiti di servizio che gli sono necessari e in base ai quali verrà consentito o negato il suo accesso alla rete. Questo processo, detto **call admission** (*ammissione di chiamata*), ispira il quarto principio (in aggiunta ai tre visti nel Paragrafo 9.5.2).

**Principio 4.** È necessario un processo di ammissione di chiamata durante il quale vengono confrontati i requisiti di servizio dei flussi (QoS richiesta) con le risorse disponibili in quel dato momento. Se la richiesta può essere soddisfatta il flusso potrà accedere alla rete, altrimenti il suo ingresso sarà negato.

Il nostro esempio rappresentato nella Figura 9.18 sottolinea la necessità di nuovi meccanismi e protocolli se, come abbiamo visto, a una chiamata (un flusso end-to-end) deve essere garantita una certa qualità del servizio, una volta che è iniziata.

- *Prenotazione di risorse*: l'unico modo per garantire che una chiamata avrà le risorse necessarie (spazio nel buffer e banda sui collegamenti) per soddisfare la QoS desiderata è di allocare esplicitamente quelle risorse per la chiamata, un processo noto nel gergo delle reti come **resource reservation**. Una volta che le risorse sono state allocate, la chiamata ha accesso a richiesta a queste risorse per tutta la sua durata, indipendentemente dalle richieste di altre chiamate. Se una chiamata riserva e riceve una garanzia per  $x$  Mbps di banda su un collegamento e non trasmette a un tasso maggiore di  $x$ , non sarà soggetta a perdite e ritardi.
- *Call admission*: se le risorse sono riservate, allora la rete deve avere un meccanismo tramite il quale le chiamate richiedono e riservano le risorse, un processo noto come call admission. Poiché le risorse non sono infinite, a una chiamata sarà negata la sua richiesta di ammissione, cioè sarà bloccata, se le risorse richieste non sono disponibili. Questo tipo di call admission è eseguito dalla rete telefonica: noi richiediamo le risorse quando digitiamo il numero. Se i circuiti (gli slot TDMA) necessari per completare la chiamata sono disponibili, questi vengono allocati e la chiamata viene completata. Se invece non lo sono, allora la chiamata viene bloccata e riceviamo il segnale di occupato. Una chiamata bloccata può tentare di nuovo di accedere alla rete, ma non le sarà concesso mandare traffico in rete fino a che non avrà completato con successo il processo di ammissione. Certamente, un router, che alloca la banda su un collegamento, non dovrebbe allocarne di più di quella disponibile. Tipicamente una chiamata può riservare solo una frazione della banda del collegamento e quindi un router può allocare risorse a più di una chiamata. Tuttavia, la somma della banda allocata alle chiamate dovrebbe essere minore della capacità del collegamento.
- *Segnalazione per l'instaurazione della chiamata*: il processo di call admission descritto precedentemente richiede che la chiamata possa riservare le risorse sufficienti per assicurarsi che i requisiti di QoS end-to-end di cui necessita possano essere soddisfatti dai router collocati sul percorso tra sorgente e destinazione. Questo processo di instaurazione della chiamata (*call setup*) richiede che i router determinino le risorse locali richieste dalla sessione, considerino quelle già impegnate e stabiliscano se dispongono di risorse sufficienti per soddisfare i requisiti di QoS, ovviamente senza sottrarle a sessioni già in corso. Per coordinare queste attività è necessario un protocollo di segnalazione. Questo è il compito del **protocollo di instaurazione della chiamata** (*call setup protocol*), illustrato nella Figura 9.19. Il **protocollo RSVP** [Zhang 1993, RFC 2210] fu proposto per fornire garanzie di qualità del servizio su Internet. Nelle reti ATM, il protocollo Q2931b [Black 1995] trasporta queste informazioni tra i commutatori e i punti terminali della rete.

Nonostante gli sforzi di ricerca e sviluppo nonché prodotti commerciali che forniscono garanzie di qualità di servizio alle singole connessioni, per molte ragioni questi



**Figura 9.19** Procedura di instaurazione di una chiamata.

servizi non sono diffusi. Innanzitutto, perché probabilmente i semplici meccanismi studiati nei Paragrafi da 9.2 a 9.4, insieme a un appropriato dimensionamento della rete, forniscono una rete best-effort le cui prestazioni sono sufficienti per le applicazioni multimediali. Inoltre, gli ISP potrebbero pensare che i costi di installazione e gestione di una rete per dare qualità di servizio alle singole connessioni siano semplicemente troppo alti se confrontati con i potenziali guadagni.

## 9.6 Riepilogo

Le reti multimediali sono oggi uno degli sviluppi più interessanti di Internet. Persone in tutto il mondo trascorrono meno tempo ascoltando radio e vedendo televisione, mentre tendono a passare a Internet per ricevere trasmissioni audio e video, in diretta o registrate. Tale tendenza si accentuerà certamente mano a mano che le reti di accesso wireless si affermeranno. Inoltre, grazie a siti quali YouTube gli utenti sono diventati non solo fruitori, ma anche produttori di contenuti multimediali. Oltre alla distribuzione video, Internet viene anche usata per la telefonia. Infatti, nei prossimi dieci anni, Internet coadiuvata dall'accesso wireless potrebbe rendere obsoleto il tradizionale sistema telefonico a commutazione di circuito. Il VoIP non solo fornisce un servizio di telefonia economico, ma anche molti servizi a valore aggiunto quali la videoconferenza, la consultazione di cataloghi on-line, la messaggistica vocale e l'integrazione con social network quali Facebook e WeChat.

Nel Paragrafo 9.1, dopo aver descritto le caratteristiche peculiari di audio e video, abbiamo classificato le applicazioni multimediali in tre categorie: (1) streaming di audio e video registrato, (2) conversazioni audio e video e (3) streaming di audio e video in tempo reale.

Nel Paragrafo 9.2 abbiamo studiato in dettaglio lo streaming di contenuti video registrati. Per queste applicazioni i video sono memorizzati in server ai quali gli utenti inviano richieste per accedervi on demand. Abbiamo visto che i sistemi di video streaming possono essere classificati in due categorie: streaming UDP e streaming HTTP. Abbiamo visto come uno degli indici di prestazione più importanti per lo streaming video sia rappresentato dal throughput medio.

Nel Paragrafo 9.3 abbiamo esaminato come le applicazioni di conversazione multimediali, come VoIP, possano essere progettate per funzionare su una rete best-effort. Per tali applicazioni la temporizzazione è importante, perché sono molto sensibili al ritardo mentre, d'altra parte, sono tolleranti alle perdite, che causano solo interruzioni occasionali nelle riproduzioni audio e video e possono essere ripristinate parzialmente o completamente. Abbiamo visto come una combinazione di buffer, numeri di sequenza sui pacchetti e marcature temporali possa molto alleviare gli effetti del jitter indotto dalla rete. Abbiamo anche dato un'occhiata alla tecnologia di Skype, una delle aziende più importanti di video e voce su IP. Nel Paragrafo 9.4 abbiamo esaminato i due principali protocolli standard per VoIP: RTP e SIP.

Nel Paragrafo 9.5 abbiamo visto come molti meccanismi di rete (politiche di scheduling a livello di collegamento e tecniche di policing del traffico) possano essere usate per fornire un servizio differenziato tra classi di traffico.

## **Domande e problemi**

---

### **Domande di revisione**

#### **PARAGRAFO 9.1**

- R1.** Ricostruite la Tabella 9.1 per il caso in cui Victor Video guarda un video di 4 Mbps, Facebook Frank una nuova immagine da 100 Kbyte ogni 20 secondi e Martha Music sta ascoltando audio in streaming a 200 kbps.
- R2.** Nei video ci sono due tipi di ridondanza: descriveteli e spiegate come possono essere sfruttati per ottenere una compressione efficiente.
- R3.** Supponete che un segnale audio analogico sia campionato 16.000 volte al secondo e che ogni campione sia quantizzato in 1024 livelli. Qual è il bit rate risultante del segnale audio digitale PCM?
- R4.** Le applicazioni multimediali possono essere classificate in tre categorie: elencate e descrivetevole.

#### **PARAGRAFO 9.2**

- R5.** I sistemi di video streaming possono essere classificati in tre categorie: elencate e descrivetevole brevemente.
- R6.** Elencate tre svantaggi dello streaming UDP.

- R7.** Nello streaming HTTP il buffer di ricezione TCP e quello dell'applicazione client sono la stessa cosa? In caso negativo, come interagiscono?
- R8.** Considerate il semplice modello di streaming HTTP. Supponete che il server invii i bit a un tasso costante pari a 2 Mbps e la riproduzione inizi quando sono stati ricevuti 8 milioni di bit. Qual è il ritardo iniziale di buffer  $t_p$ ?

### **PARAGRAFO 9.3**

- R9.** Qual è la differenza fra ritardo end-to-end e jitter di un pacchetto? Quali sono le cause del jitter?
- R10.** Perché un pacchetto ricevuto dopo il tempo previsto per la riproduzione è considerato perso?
- R11.** Riassumete i due schemi FEC descritti nel Paragrafo 9.3. Si noti che entrambi aumentano il tasso trasmissivo del flusso aggiungendo ridondanza. Anche l'interleaving incrementa il tasso trasmissivo?

### **PARAGRAFO 9.4**

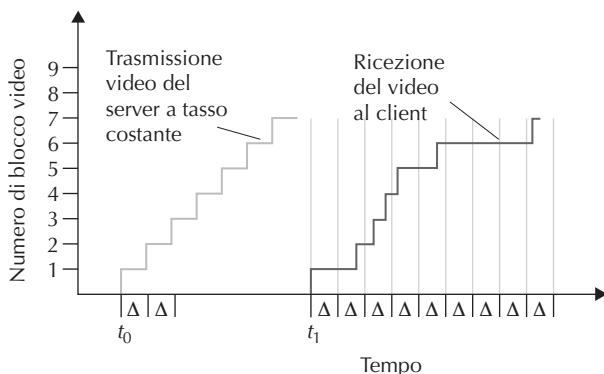
- R12.** Come può un ricevente identificare differenti flussi RTP in diverse sessioni? Come possono essere identificati quelli della stessa sessione?
- R13.** Qual è il ruolo di un server di registrazione SIP e che cosa lo differenzia da quello di un agente domestico in IP Mobile?

## **Problemi**

**P1.** Considerate la figura riportata di seguito. Richiamando la discussione riguardante la Figura 9.1, supponete che il video sia codificato a un bit rate fisso e che quindi ogni blocco del video contenga frame che vengono riprodotti nella stessa quantità di tempo,  $\Delta$ . Il server trasmette il primo blocco video al tempo  $t_0$ , il secondo blocco al tempo  $t_0 + \Delta$ , il terzo a  $t_0 + 2\Delta$  e così via. Quando il client inizia la riproduzione, ogni blocco dovrebbe essere riprodotto  $\Delta$  unità di tempo dopo quello precedente.

- Supponete che il client inizi la riproduzione non appena arriva il primo blocco al tempo  $t_1$ . Nella figura sottostante, quanti blocchi video, compreso il primo, arriveranno al client in tempo per essere riprodotti? Spiegate il ragionamento che avete fatto.
- Supponete ora che il client inizi la riproduzione al tempo  $t_1 + \Delta$ . Quanti blocchi video, compreso il primo, arriveranno al client in tempo per essere riprodotti? Spiegate il ragionamento che avete fatto.
- Nello stesso scenario del punto (b), qual è il massimo numero di blocchi che verranno memorizzati nel buffer del client aspettando di essere riprodotti? Spiegate il ragionamento che avete fatto.

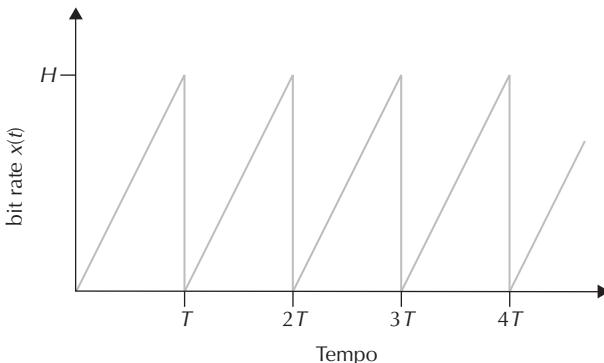
- (d) Qual è il più piccolo ritardo di riproduzione al client, tale che ogni blocco arrivi in tempo per essere riprodotto? Spiegate il ragionamento che avete fatto.



**P2.** Riprendete il semplice modello di streaming HTTP mostrato nella Figura 9.3, nel quale  $B$  denota la grandezza del buffer dell'applicazione client e  $Q$  denota il numero di bit che deve essere memorizzato nel buffer prima che l'applicazione client inizi la riproduzione. Sia  $r$  inoltre il tasso di consumo del video. Assumete che il server invii bit a un tasso costante  $x$ , finché il buffer del client non sia pieno.

- Supponete  $x < r$ . Come discusso nel testo, in questo caso la riproduzione avrà periodi alternati di continuità e di blocco. Determinate la lunghezza di questi due tipi di periodi, in funzione di  $Q$ ,  $r$  e  $x$ .
- Supponete ora  $x > r$ . A quale tempo  $t = t_f$  il buffer dell'applicazione client si riempie?

**P3.** Riprendete il semplice modello di streaming HTTP mostrato nella Figura 9.3. Supponete che la grandezza del buffer sia infinita, ma che il tasso,  $x(t)$ , a cui il server invia bit sia variabile; in particolare, supponete che abbia il comportamento a dente di sega seguente: al tempo iniziale  $t = 0$ , il tasso è zero e quindi ha una crescita lineare fino a  $H$  al tempo  $t = T$ . Tale comportamento si ripete nel tempo, come mostrato nella figura sottostante.



- (a) Qual è il tasso medio di trasmissione del server?
- (b) Supponete  $Q = 0$ , in modo che il client inizi la riproduzione non appena riceve un frame video. Che cosa succede?
- (c) Supponete ora  $Q > 0$ . Determinate il tempo in cui inizia per la prima volta la riproduzione, in funzione di  $Q$ ,  $H$  e  $T$ .
- (d) Supponete  $H > 2r$  e  $Q = HT/2$ . Dimostrate che, dopo il ritardo iniziale di riproduzione, non vi sarà alcun blocco dell'immagine.
- (e) Supponete  $H > 2r$ . Trovate il minimo valore di  $Q$  tale che non vi sia alcun blocco dopo il ritardo iniziale di riproduzione.
- (f) Supponete ora che la dimensione del buffer  $B$  sia finita e che  $H > 2r$ . Determinate il tempo  $t = t_f$  nel quale il buffer dell'applicazione client diventa pieno, in funzione di  $Q$ ,  $B$ ,  $T$  e  $H$ .

**P4.** Riprendete il semplice modello di streaming HTTP mostrato nella Figura 9.3. Supponete che la grandezza del buffer sia infinita, ma che il tasso,  $x(t)$ , a cui il server invia bit sia costante e che il tasso di consumo sia  $r$  con  $r < x$ ; supponete inoltre che la riproduzione inizi immediatamente e che l'utente la termini a  $t = E$ , tempo in cui il server quindi smette di inviare bit.

- (a) Supponendo che il video sia infinitamente lungo, quanti bit vengono sprecati perché trasmessi, ma non visti?
- (b) Supponendo che il video sia lungo  $T$  secondi, con  $T > E$ , quanti bit vengono sprecati perché trasmessi, ma non visti?

**P5.** Considerate un sistema DASH con  $N$  versioni video, a  $N$  bit rate e qualità diversi, e  $N$  versioni audio, a  $N$  bit rate e qualità diversi. Supponete di voler dare la possibilità all'utente di scegliere in ogni istante quale delle  $N$  versioni video e audio voglia.

- (a) Se i file che creiamo mescolano audio e video, in modo che il server invii solo uno stream in un dato istante, quanti file deve memorizzare il server (ognuno a un URL diverso)?
- (b) Se invece il server invia separatamente gli stream audio e video e il client li sincronizza, quanti file deve memorizzare il server?

**P6.** Nell'esempio di VoIP riportato nel Paragrafo 9.3 abbiamo indicato con  $h$  il numero totale di byte di intestazione aggiunti a ogni blocco, comprese le intestazioni UDP e IP.

- (a) Assumendo che venga emesso un datagramma IP ogni 20 ms, trovate il tasso trasmissivo in bit al secondo per i datagrammi generati da un lato di questa applicazione.
- (b) Qual è un tipico valore di  $h$  quando si utilizza RTP?

**P7.** Considerate la procedura descritta nel Paragrafo 9.3 per stimare il ritardo medio  $d$ . Supponete  $u = 0,1$ , che  $r_1 - t_1$  sia il ritardo campionario più di recente,  $r_2 - t_2$  sia il ritardo campionario successivamente e così via.

- Per una data applicazione audio assumiamo che quattro pacchetti siano arrivati al ricevente con ritardi campionati di  $r_4 - t_4$ ,  $r_3 - t_3$ ,  $r_2 - t_2$  e  $r_1 - t_1$ . Esprimete il ritardo stimato  $d$  per i quattro campioni.
- Generalizzate la formula ottenuta per  $n$  ritardi campionati.
- Nella parte (b) fate tendere  $n$  all’infinito e fornite la formula risultante. Commentate i motivi per cui questa procedura è detta media mobile esponenziale.

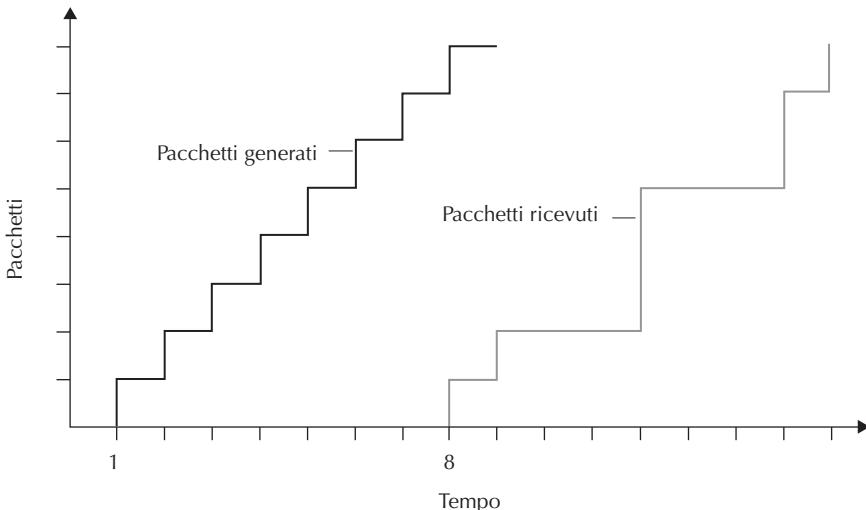
**P8.** Ripetete le parti (a) e (b) del problema precedente per la stima della deviazione dal ritardo medio.

**P9.** Nell’esempio di VoIP nel Paragrafo 9.3 abbiamo introdotto una procedura (media mobile esponenziale) per stimare il ritardo. In questo problema esamineremo una procedura alternativa. Indichiamo con  $t_i$  la marcatura temporale del pacchetto  $i$ -esimo pervenuto e con  $r_i$  l’istante della sua ricezione. Sia  $d_n$  la nostra stima del ritardo medio dopo la ricezione del pacchetto  $n$ -esimo. Dopo la ricezione del primo pacchetto, poniamo la stima del ritardo uguale a  $d_1 = r_1 - t_1$ .

- Supponiamo di voler trovare  $d_n = (r_1 - t_1 + r_2 - t_2 + \dots + r_n - t_n) / n$  per ogni  $n$  intero e positivo. Indicate la formula ricorsiva per  $d_n$  in termini di  $d_{n-1}$ ,  $r_n$  e  $t_n$ .
- Spiegate perché nella telefonia su Internet la stima del ritardo descritta nel Paragrafo 9.3 è più appropriata di quella proposta nella parte (a).

**P10.** Confrontate la procedura descritta nel Paragrafo 9.3 per la stima del ritardo medio con quella del Paragrafo 3.5 per la stima del tempo di andata e ritorno. Che cosa hanno in comune? In che cosa differiscono?

**P11.** Considerate la figura seguente, simile alla Figura 9.3. Un mittente inizia a mandare audio pacchettizzato periodicamente a  $t = 1$ . Il primo pacchetto perviene al ricevente a  $t = 8$ .



- (a) Quali sono i ritardi (dal mittente al ricevente, ignorando qualsiasi ritardo di riproduzione) dei pacchetti da 2 a 8? Si noti che i tratti verticali e orizzontali nella figura hanno una lunghezza di 1, 2 o 3 unità di tempo.
- (b) Se la riproduzione audio inizia appena il primo pacchetto arriva al ricevente a  $t = 8$ , quale dei primi 8 pacchetti inviati non arriverà in tempo per la riproduzione?
- (c) Se la riproduzione audio inizia a  $t = 9$ , quale dei primi 8 pacchetti inviati non arriverà in tempo per la riproduzione?
- (d) Qual è il minimo ritardo di riproduzione al ricevente, che risulta in tutti i primi 8 pacchetti che arrivano in tempo per essere riprodotti?

**P12.** Considerate di nuovo la figura del problema precedente, che mostra i tempi di trasmissione e ricezione dei pacchetti audio.

- (a) Calcolate la stima del ritardo per i pacchetti da 2 a 8, usando la formula per  $d_i$  del Paragrafo 9.3.2. Usate il valore  $u = 0,1$ .
- (b) Calcolate la stima della deviazione standard del ritardo, dalla stima della media per i pacchetti da 2 a 8, usando la formula del Paragrafo 9.3.2. Usate il valore  $u = 0,1$ .

**P13.** Facciamo riferimento agli schemi FEC per VoIP descritti nel Paragrafo 9.3 e supponiamo che il primo generi un blocco ridondante ogni quattro e che il secondo impieghi una codifica la cui frequenza trasmissiva corrisponde al 25% di quella nominale del flusso.

- (a) Quanta larghezza di banda aggiuntiva richiedono gli schemi? Quanto ritardo di riproduzione aggiungono?
- (b) Come si comportano i due schemi se il primo pacchetto di ciascun gruppo di cinque va perso? Quale schema avrebbe la migliore qualità audio?
- (c) Come si comportano i due schemi se il primo pacchetto in ciascun gruppo di due va perso? Quale schema avrebbe la migliore qualità audio?

**P14.** (a) Considerate una audio conferenza Skype con  $N > 2$  partecipanti, ognuno dei quali genera uno stream costante al tasso di  $r$  bps. Chi inizia la conferenza quanti bit al secondo deve trasmettere? Quanti ognuno degli altri partecipanti? Qual è il bit rate totale di invio, aggregato su tutti i partecipanti?

- (b) Ripetete l'esercizio (a) considerando una video conferenza Skype che usa un server centralizzato.
- (c) Ripetete l'esercizio (b), nel caso in cui ogni partecipante invii una copia del suo stream video a ciascuno degli altri.

**P15.** (a) Supponiamo di spedire in Internet due datagrammi IP, contenenti due diversi segmenti UDP. Il primo datagramma ha indirizzo IP sorgente A1 e destinazione B, porta sorgente P1 e destinazione T. Il secondo datagramma ha indirizzo IP

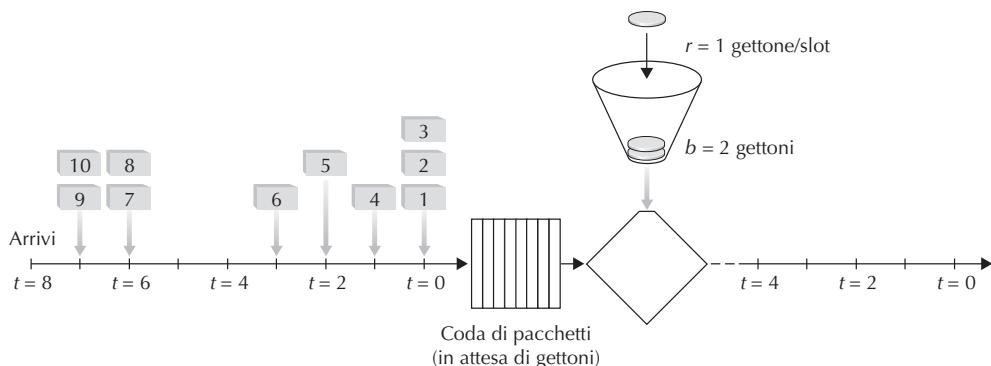
sorgente A2, diverso da A1, e destinazione B, porta sorgente P2, diversa da P1, e destinazione T. Se i datagrammi raggiungono la destinazione finale, saranno ricevuti dalla stessa socket? Perché?

- (b) Assumete che Alice, Bob e Claire vogliano realizzare una conferenza audio utilizzando SIP e RTP. Affinché Alice possa scambiare pacchetti RTP con Bob e Claire, è sufficiente una sola socket UDP (oltre a quella necessaria per i messaggi SIP)? Se sì, come fa il client SIP di Alice a distinguere i pacchetti RTP ricevuti da Bob da quelli di Claire?

**P16.** Vero o falso?

- (a) Se un video registrato è inviato direttamente in un flusso da un web server a un media player, l'applicazione utilizza TCP come protocollo di trasporto sottostante.
- (b) Quando si utilizza RTP è possibile che un mittente cambi la codifica durante una sessione.
- (c) Tutte le applicazioni che utilizzano RTP devono usare la porta 87.
- (d) Le sessioni RTP con flussi audio e video separati per ogni trasmittente usano lo stesso SSRC per i flussi.
- (e) Nei servizi differenziati il comportamento per-hop definisce le differenze di prestazioni tra le classi, ma non impone particolari meccanismi per ottenerle.
- (f) Supponete che Alice voglia stabilire una sessione SIP con Bob. Nel suo messaggio INVITE inserisce la linea:  $m = \text{audio } 48753 \text{ RTP/AVP } 3$  (dove AVP 3 indica l'audio GSM). Alice ha quindi indicato in questo messaggio che vuole inviare audio GSM.
- (g) Con riferimento all'affermazione precedente, Alice ha indicato nel suo messaggio INVITE che invierà l'audio alla porta 48753.
- (h) I messaggi SIP sono generalmente inviati tra entità che utilizzano un numero di porta SIP di default.
- (i) Per conservare la registrazione, i client SIP devono mandare periodicamente dei messaggi REGISTER.
- (j) SIP impone che tutti i client SIP supportino la codifica audio G.711.

**P17.** Considerate la figura seguente, che mostra un controllo con leaky bucket che viene alimentato da un flusso di pacchetti. Il buffer dei gettoni può contenere al massimo due gettoni e al tempo  $t = 0$  è inizialmente pieno. I nuovi gettoni arrivano a un tasso di un gettone per slot. La velocità del collegamento in uscita è tale che se due pacchetti ottengono i gettoni all'inizio dello slot temporale, possono entrambi andare sul collegamento di uscita nello stesso slot. I dettagli di temporizzazione del sistema sono i seguenti.



- I pacchetti (se vi sono) arrivano all'inizio dello slot. Quindi, nella figura, i pacchetti 1, 2 e 3 arrivano nello slot 0. Se ci sono già pacchetti in coda, allora i pacchetti che arrivano si accodano dietro ai precedenti. I pacchetti attraversano la coda in modalità FIFO.
- Dopo che i pacchetti arrivati sono stati aggiunti alla coda, se ci sono pacchetti accodati, 1 o 2 di quei pacchetti (a seconda del numero di gettoni disponibili) rimuoveranno ciascuno un gettone dal buffer dei gettoni e usciranno sul collegamento durante quello slot. Quindi i pacchetti 1 e 2 rimuoveranno un gettone ciascuno dal buffer, che inizialmente conteneva 2 gettoni, e usciranno sul collegamento nello slot 0.
- Un nuovo gettone viene aggiunto al buffer dei gettoni se non è pieno, dato che la frequenza di generazione dei gettoni è  $r = 1$  gettone/slot.
- Il tempo poi prosegue allo slot successivo e questi passi si ripetono.

Rispondete alle seguenti domande:

- Per ogni slot temporale identificate i pacchetti che sono in coda e il numero di gettoni nel secchio, immediatamente dopo che gli arrivi sono stati elaborati (passo 1 precedente), ma prima che qualsiasi pacchetto sia passato attraverso la coda e abbia rimosso un gettone. Quindi per lo slot temporale 0 nell'esempio precedente, i pacchetti 1, 2 e 3 sono in coda e 2 gettoni sono nel buffer.
- Per ogni slot temporale indicate quali pacchetti compaiono all'uscita dopo che i gettoni sono stati rimossi dalla coda. Quindi per lo slot temporale 0 nell'esempio precedente, i pacchetti 1, 2 appaiono al collegamento di uscita dal buffer del secchio durante lo slot 0.

- P18.** Ripetete il problema precedente, ma assumete che  $r = 2$  e che, di nuovo, il bucket sia inizialmente pieno.
- P19.** Considerate adesso il Problema 18 e supponete ora che  $r = 3$  e che  $b = 2$  come prima. La vostra risposta alla domanda precedente cambia?

- P20. Considerate la strategia leaky bucket che controlla il tasso medio e le dimensioni dei picchi di un flusso di pacchetti. Ora vogliamo prendere in esame anche il tasso di picco,  $p$ . Mostrate come possa essere impiegato un secondo bucket posto in serie, in modo da controllare il tasso medio, quello di picco e le dimensioni della raffica. Assicuratevi di stabilire le dimensioni dei bucket e la frequenza di generazione dei gettoni.
- P21. Un flusso di pacchetti è detto conforme alle specifiche di un leaky bucket con dimensione del picco  $b$  e tasso medio  $r$  se il numero di pacchetti che gli arrivano è inferiore a  $rt + b$  per ogni intervallo di tempo di lunghezza  $t$ . Può un flusso di pacchetti che si adatta alle specifiche  $(r, b)$  essere posto in attesa da un leaky bucket con parametri  $r$  e  $b$ ? Argomentate la vostra risposta.
- P22. Dimostrate che se  $r_1 < R w_1 / (\sum w_j)$ , allora  $d_{\max}$  è il ritardo massimo osservato da ogni pacchetto del flusso 1 nella coda WFQ.

## Esercizi di programmazione

---

L'obiettivo è quello di realizzare un client e un server per lo streaming di video, dove il primo utilizza il protocollo di streaming in tempo reale, RTSP, per controllare le azioni del server, e il secondo usa il protocollo RTP, per preparare i pacchetti video per il trasporto su UDP.

Supponendo di disporre del codice Python che implementa parzialmente RTSP e RTP, il vostro compito sarà completare il codice del client e del server. Quando avrete finito, avrete creato un'applicazione client-server che realizza quanto segue.

- Il client invia comandi RTSP SETUP, PLAY, PAUSE e TEARDOWN, e il server risponde ai comandi.
- Quando il server è nello stato *playing*, periodicamente prende un frame JPEG memorizzato, lo pacchettizza con RTP e lo invia tramite una socket UDP.
- Il client riceve i pacchetti RTP, estrae i frame JPEG, li decomprime e li riproduce.

Il codice che vi sarà fornito implementa RTSP nel server e la depacchettizzazione RTP nel client, e si occupa anche di mostrare il video trasmesso. Voi dovete implementare RTSP nel client e RTP nel server.

Questo esercizio di programmazione è fortemente raccomandato, in quanto accresce in modo significativo la comprensione di RTP, di RTSP e dello streaming video. È possibile anche integrarlo con l'implementazione del comando RTSP DESCRIBE nel client e nel server.

Sul sito web del libro [www.pearsonhighered.com/cs-resources](http://www.pearsonhighered.com/cs-resources) troverete dettagli completi relativi all'esercizio e una panoramica del protocollo RTSP.

# Intervista a...



## Henning Schulzrinne

*Co-autore di RTP, RTSP, SIP e GIST (protocolli chiave per la comunicazione audio/video su Internet) Henning Schulzrinne è professore e direttore del dipartimento di informatica della Columbia University nonché direttore del laboratorio “Internet Real-Time” della stessa università. Henning ha conseguito la laurea in Ingegneria elettronica e industriale presso l’Università di Darmstadt in Germania, il master in Ingegneria e informatica alla University of Cincinnati e il dottorato in Ingegneria elettronica alla University of Massachusetts ad Amherst.*

### **Perché ha deciso di specializzarsi nelle reti multimediali?**

Accadde per caso. Da studente di dottorato sono stato coinvolto nei lavori su DARTnet, una rete sperimentale con collegamenti T1 che copriva gli Stati Uniti e utilizzata come terreno di collaudo per il multicast e gli strumenti Internet real-time. In questa occasione ho realizzato la mia prima applicazione audio, NeVoT. In seguito sono entrato in contatto con IETF, nell'allora nascente gruppo di lavoro sul trasporto audio e video che, più tardi, produsse lo standard RTP.

### **Quale fu il suo primo lavoro nell'industria dei calcolatori? Che cosa le ha lasciato?**

Quando ero studente al liceo di Livermore, in California, ho lavorato all'assemblaggio fisico di un computer Altair. Tornato in Germania, ho fondato una piccola società di consulenza con la quale realizzammo un programma di gestione degli indirizzi per un'agenzia di viaggi. Memorizzavamo i dati per il nostro TRS-80 su cassette e usavamo una macchina da scrivere Selectric IBM con un'interfaccia hardware "fatta in casa" per la stampa.

Il mio primo vero lavoro fu presso i laboratori AT&T della Bell, dove sviluppai un emulatore di reti.

### **Quali sono gli obiettivi del laboratorio Internet Real-Time?**

Innanzitutto quello di fornire componenti per l'infrastruttura di Internet come unica futura piattaforma di comunicazione. Ciò comprende lo sviluppo di protocolli, come GIST (per la segnalazione a livello di rete) e LOST (per trovare risorse in base alla loro posizione), o arricchendo protocolli sui quali abbiamo già lavorato prima, come SIP, lavorando su un servizio di presenza più ricco, sui sistemi P2P, sulle chiamate di emergenza di nuova generazione e sugli strumenti per la creazione dei servizi. Recentemente abbiamo considerato estensivamente i sistemi wireless per VoIP, come le reti 802.11b e 802.11n ed è probabile che le reti Wi-Max diventino importanti tecnologie per l'ultimo miglio della telefonia. Stiamo inoltre tentando di aumentare la capacità degli utenti di rilevare guasti nel garbuglio di provider e apparati usando un sistema di diagnostica peer-to-peer chiamato DYSWYS (Do You See What I See).

Cerchiamo di fare un lavoro che sia rilevante anche dal punto di vista pratico, realizzando prototipi e sistemi open source, misurando le prestazioni dei sistemi reali e contribuendo agli standard IETF.

## Che cosa prevede per il futuro delle reti multimediali?

Direi che attualmente siamo in una fase di transizione; entro pochi anni IP diventerà la piattaforma universale per i servizi multimediali, da IPTV a VoIP. Oggi tutti si aspettano che la radio, il telefono e la televisione funzionino anche durante una tempesta di neve o un terremoto e da Internet pre-tenderanno lo stesso livello di affidabilità.

Dovremo imparare a progettare tecnologie di rete per un ecosistema di operatori in concorrenza, fornitori di servizi e contenuti, che servano molti utenti privi di conoscenze tecniche e che li difendano da piccoli e distruttivi gruppi di utenti malintenzionati. Cambiare i protocolli sta diventando sempre più difficile e questi ultimi stanno diventando sempre più complessi, in quanto devono tenere presenti interessi commerciali concorrenti, sicurezza, privacy e la mancanza di trasparenza delle reti causata dai firewall e dai NAT.

Poiché le reti multimediali stanno diventando il fondamento di quasi tutti i servizi di intrattenimento per i consumatori, ci sarà un enfasi su una gestione a basso costo di reti molto grandi. Gli utenti si aspettano la facilità di utilizzo, così come di trovare lo stesso contenuto su tutti i loro dispositivi.

## Perché SIP ha un futuro promettente?

Visto che le attuali reti wireless si stanno adeguando alle reti 3G, c'è la speranza di avere un'unica tecnologia di segnalazione per tutti i tipi di reti, da quelle collegate con i modem alle reti telefoniche, fino alle reti wireless pubbliche. Assieme alle software radio, in futuro sarà possibile utilizzare un singolo dispositivo (come i telefoni senza fili BlueTooth) nelle reti domestiche, in quelle aziendali con 802.11 e nelle reti estese con 3G. Ma ancor prima di poter impiegare dispositivi wireless universali, già i meccanismi di mobilità individuale rendono possibile questo approccio mascherando le differenze tra le reti. Un identificatore può rappresentare il mezzo ideale per rintracciare una persona, piuttosto che dover ricordare una decina di numeri o essere smistati attraverso innumerevoli centralini telefonici.

SIP separerà la fornitura del trasporto della voce in bit dai servizi vocali. È diventato tecnicamente possibile smontare il monopolio della telefonia locale, per cui una sola azienda fornisce il trasporto neutrale dei bit, mentre altre forniscono telefonia su IP e servizi telefonici classici, come l'inoltro di chiamata e l'identificazione del chiamante.

Oltre alla trasmissione multimediale, SIP offre un nuovo servizio di cui Internet aveva bisogno: la notifica di eventi. Ci siamo avvicinati a questo tipo di servizio con HTTP ed e-mail, ma senza raggiungere risultati veramente soddisfacenti. Poiché gli eventi sono astrazioni comuni per i sistemi distribuiti, la costruzione di nuovi servizi potrebbe essere semplificata.

## Darebbe qualche consiglio agli studenti che si affacciano allo studio delle reti?

Le reti possono essere considerate come una classica materia interdisciplinare che deriva dall'ingegneria elettronica, dall'informatica, dalla ricerca operativa, dalla statistica, dall'economia e da altri campi di studio. Pertanto, chi fa ricerca nel campo delle reti deve avere familiarità anche con i principali argomenti di altri settori.

Siccome le reti stanno conquistandosi una posizione dominante nella vita di tutti i giorni, gli studenti che vogliono dare un loro contributo dovrebbero riflettere sui nuovi vincoli a cui sono soggette le risorse di rete: tempo e costo umano piuttosto che solo banda e capacità di immagazzinamento.

Lavorare nelle reti può essere molto soddisfacente, in quanto consente alle persone di comunicare e di scambiarsi idee: uno degli aspetti fondamentali dell'animo umano. Internet è diventata la terza grande infrastruttura mondiale, dopo quella dei trasporti e dell'energia. Non esiste alcuna parte dell'economia che possa funzionare senza reti ad alte prestazioni, per cui dovrebbe esserci un sacco di opportunità in futuro.



# Bibliografia

**Nota sugli URL:** in questa bibliografia sono indicati gli URL relativi a pagine web, a documenti disponibili solo via web e ad altro materiale non pubblicato su riviste o atti di conferenze (quando gli autori hanno potuto rintracciare gli indirizzi web relativi). A differenza di quanto fatto nelle edizioni precedenti, non sono forniti gli URL relativi a pubblicazioni e atti di conferenze, in quanto si tratta di documenti che possono facilmente essere reperiti tramite un motore di ricerca o attraverso il sito web della conferenza (ad esempio gli atti dei convegni ACM Sigcomm sono disponibili all'indirizzo <http://www.acm.org/sigcomm>), o ricevuti da chi è abbonato a una biblioteca digitale. In ogni caso, tutti gli indirizzi forniti erano sicuramente validi nel gennaio 2016, ma sappiamo che talvolta diventano obsoleti in fretta. Per avere sempre l'indirizzo corretto, potete consultare la versione on-line del libro all'indirizzo [www.pearsonhighered.com/cs-resources](http://www.pearsonhighered.com/cs-resources), dove la bibliografia viene costantemente aggiornata.

**Nota sui Request for Comments (RFC):** copie degli RFC sono disponibili su diversi siti. I curatori degli RFC dell'Internet Society (l'ente che sovrintende a tutte le RFC) tengono costantemente aggiornato il sito <http://www.rfc-editor.org>. Questo sito consente una specifica ricerca per titolo, numero o autore e mostra tutti gli aggiornamenti per ciascuna RFC. Le RFC Internet, infatti, possono esser aggiornate o rese obsolete da RFC più recenti. Questo è il nostro sito di riferimento per questo tipo di documenti.

- [**3GPP 2020**] 3GPP, 3GPP Specification Set, <https://www.3gpp.org/dynareport/SpecList.htm>
- [**3GPP GTPv1-U 2019**] 3GPP, “Tunnelling Protocol User Plane (GTPv1-U),” 3GPP Technical Specification 29.281 version 15.3.0, 2018.
- [**3GPP PDCP 2019**] 3GPP, “Packet Data Convergence Protocol (PDCP) Specification,” 3GPP Technical Specification 36.323 version 15.4.0, 2019.
- [**3GPP RLCP 2018**] 3GPP, “Radio Link Control (RLC) protocol specification,” 3GPP Technical Specification 25.322 version 15.0.0, 2018.
- [**3GPP SAE 2019**] 3GPP, “System Architecture Evolution (SAE); Security architecture,” Technical Specification 33.401, version 15.9.0, October 2019.”
- [**Abramson 1970**] N. Abramson, “The Aloha System—Another Alternative for Computer Communications,” *Proc. 1970 Fall Joint Computer Conference, AFIPS Conference*, p. 37, 1970.
- [**Abramson 1985**] N. Abramson, “Development of the Alohanet,” *IEEE Transactions on Information Theory*, Vol. IT-31, No. 3 (Mar. 1985), pp. 119–123.
- [**Abramson 2009**] N. Abramson, “The Alohanet—Surfing for Wireless Data,” *IEEE Communications Magazine*, Vol. 47, No. 12, pp. 21–25.

- [Adhikari 2011a]** V. K. Adhikari, S. Jain, Y. Chen, Z. L. Zhang, “Vivisecting YouTube: An Active Measurement Study,” Technical Report, University of Minnesota, 2011.
- [Adhikari 2012]** V. K. Adhikari, Y. Gao, F. Hao, M. Varvello, V. Hilt, M. Steiner, Z. L. Zhang, “Unreeling Netflix: Understanding and Improving Multi-CDN Movie Delivery,” Technical Report, University of Minnesota, 2012.
- [Afanasyev 2010]** A. Afanasyev, N. Tilley, P. Reiher, L. Kleinrock, “Host-to-Host Congestion Control for TCP,” *IEEE Communications Surveys & Tutorials*, Vol. 12, No. 3, pp. 304–342.
- [Agarwal 2009]** S. Agarwal, J. Lorch, “Matchmaking for Online Games and Other Latency-sensitive P2P Systems,” *Proc. 2009 ACM SIGCOMM*.
- [Ager 2012]** B. Ager, N. Chatzis, A. Feldmann, N. Sarrar, S. Uhlig, W. Willinger, “Anatomy of a Large European ISP,” *Proc. 2012 ACM SIGCOMM*.
- [Akamai 2020]** Akamai homepage, <http://www.akamai.com>
- [Akella 2003]** A. Akella, S. Seshan, A. Shaikh, “An Empirical Evaluation of Wide-Area Internet Bottlenecks,” *Proc. 2003 ACM Internet Measurement Conference* (Miami, FL, Nov. 2003).
- [Akhshabi 2011]** S. Akhshabi, A. C. Begen, C. Dovrolis, “An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP,” *Proc. 2011 ACM Multimedia Systems Conf.*
- [Akhshabi 2011]** S. Akhshabi, C. Dovrolis, “The evolution of layered protocol stacks leads to an hourglass-shaped architecture,” *Proceedings 2011 ACM SIGCOMM*, pp. 206–217.
- [Akyildiz 2010]** I. Akyildiz, D. Gutierrez-Estevez, E. Reyes, “The Evolution to 4G Cellular Systems, LTE Advanced,” *Physical Communication*, Elsevier, 3 (2010), pp. 217–244.
- [Albitz 1993]** P. Albitz and C. Liu, *DNS and BIND*, O’Reilly & Associates, Petaluma, CA, 1993.
- [Alexandris 2016]** K. Alexandris, N. Nikaein, R. Knopp and C. Bonnet, “Analyzing X2 handover in LTE/LTE-A,” *2016 14th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, Tempe, AZ, pp. 1–7.
- [Alizadeh 2010]** M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, “Data center TCP (DCTCP),” *Proc. 2010 ACM SIGCOMM Conference*, ACM, New York, NY, USA, pp. 63–74.
- [Alizadeh 2013]** M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, S. Shenker, “pFabric: Minimal Near-Optimal Datacenter Transport,” *Proc. 2013 ACM SIGCOMM Conference*.
- [Alizadeh 2014]** M. Alizadeh, T. Edsall, S. Dharmapurikar, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, G. Varghese, “CONGA: Distributed Congestion-Aware Load Balancing for Datacenters,” *Proc. 2014 ACM SIGCOMM Conference*.
- [Allman 2011]** E. Allman, “The Robustness Principle Reconsidered: Seeking a Middle Ground,” *Communications of the ACM*, Vol. 54, No. 8 (Aug. 2011), pp. 40–45.
- [Almers 2007]** P. Almers, et al., “Survey of Channel and Radio Propagation Models for Wireless MIMO Systems,” *Journal on Wireless Communications and Networking*, 2007.
- [Amazon 2014]** J. Hamilton, “AWS: Innovation at Scale,” YouTube video, [https://www.youtube.com/watch?v=JIQETrFC\\_SQ](https://www.youtube.com/watch?v=JIQETrFC_SQ)
- [Anderson 1995]** J. B. Andersen, T. S. Rappaport, S. Yoshida, “Propagation Measurements and Models for Wireless Communications Channels,” *IEEE Communications Magazine*, (Jan. 1995), pp. 42–49.
- [Appenzeller 2004]** G. Appenzeller, I. Keslassy, N. McKeown, “Sizing Router Buffers,” *Proc. 2004 ACM SIGCOMM Conference* (Portland, OR, Aug. 2004).

- [Arkko 2012]** J. Arkko, “Analysing IP Mobility Protocol Deployment Difficulties,” 83rd IETF meeting, March, 2012.
- [ASO-ICANN 2020]** The Address Supporting Organization homepage, <http://www.aso.icann.org>
- [AT&T 2019]** A. Fuetsch, “From Next-Gen to Now: SDN, White Box and Open Source Go Mainstream,” [https://about.att.com/innovationblog/2019/09/sdn\\_white\\_box\\_and\\_open\\_source\\_go\\_mainstream.html](https://about.att.com/innovationblog/2019/09/sdn_white_box_and_open_source_go_mainstream.html)
- [Atheros 2020]** Atheros Communications Inc., “Atheros AR5006 WLAN Chipset Product Bulletins,” <http://www.atheros.com/pt/AR5006Bulletins.htm>
- [Ayanoglu 1995]** E. Ayanoglu, S. Paul, T. F. La Porta, K. K. Sabnani, R. D. Gitlin, “AIRMAIL: A Link-Layer Protocol for Wireless Networks,” *ACM ACM/Baltzer Wireless Networks Journal*, 1: 47–60, Feb. 1995.
- [Bakre 1995]** A. Bakre, B. R. Badrinath, “I-TCP: Indirect TCP for Mobile Hosts,” *Proc. 1995 Int. Conf. on Distributed Computing Systems (ICDCS)* (May 1995), pp. 136–143.
- [Baldauf 2007]** M. Baldauf, S. Dustdar, F. Rosenberg, “A Survey on Context-Aware Systems,” *Int. J. Ad Hoc and Ubiquitous Computing*, Vol. 2, No. 4 (2007), pp. 263–277.
- [Baran 1964]** P. Baran, “On Distributed Communication Networks,” *IEEE Transactions on Communication Systems*, Mar. 1964. Rand Corporation Technical report with the same title (Memorandum RM-3420-PR, 1964). <http://www.rand.org/publications/RM/RM3420/>
- [Bardwell 2004]** J. Bardwell, “You Believe You Understand What You Think I Said . . . The Truth About 802.11 Signal and Noise Metrics: A Discussion Clarifying Often-Misused 802.11 WLAN Terminologies,” [http://www.connect802.com/download/techpubs/2004/you\\_believe\\_D100201.pdf](http://www.connect802.com/download/techpubs/2004/you_believe_D100201.pdf)
- [Barford 2009]** P. Barford, N. Duffield, A. Ron, J. Sommers, “Network Performance Anomaly Detection and Localization,” *Proc. 2009 IEEE INFOCOM* (Apr. 2009).
- [Beck 2019]** M. Beck, “On the hourglass model,” *Commun. ACM*, Vol. 62, No. 7 (June 2019), pp. 48–57.
- [Beheshti 2008]** N. Beheshti, Y. Ganjali, M. Ghobadi, N. McKeown, G. Salmon, “Experimental Study of Router Buffer Sizing,” *Proc. ACM Internet Measurement Conference* (Oct. 2008, Vouliagmeni, Greece).
- [Bender 2000]** P. Bender, P. Black, M. Grob, R. Padovani, N. Sindhushayana, A. Viterbi, “CDMA/HDR: A Bandwidth-Efficient High-Speed Wireless Data Service for Nomadic Users,” *IEEE Commun. Mag.*, Vol. 38, No. 7 (July 2000), pp. 70–77.
- [Berners-Lee 1989]** T. Berners-Lee, CERN, “Information Management: A Proposal,” Mar. 1989, May 1990. <http://www.w3.org/History/1989/proposal.html>
- [Berners-Lee 1994]** T. Berners-Lee, R. Cailliau, A. Luotonen, H. Frystyk Nielsen, A. Secret, “The World-Wide Web,” *Communications of the ACM*, Vol. 37, No. 8 (Aug. 1994), pp. 76–82.
- [Bertsekas 1991]** D. Bertsekas, R. Gallagher, *Data Networks*, 2nd Ed., Prentice Hall, Englewood Cliffs, NJ, 1991.
- [Biersack 1992]** E. W. Biersack, “Performance Evaluation of Forward Error Correction in ATM Networks,” *Proc. 1999 ACM SIGCOMM Conference* (Baltimore, MD, Aug. 1992), pp. 248–257.
- [BIND 2020]** Internet Software Consortium page on BIND, <http://www.isc.org/bind.html>
- [Bisdikian 2001]** C. Bisdikian, “An Overview of the Bluetooth Wireless Technology,” *IEEE Communications Magazine*, No. 12 (Dec. 2001), pp. 86–94.
- [Bishop 2003]** M. Bishop, *Computer Security: Art and Science*, Boston: Addison Wesley, Boston MA, 2003.
- [Bishop 2004]** M. Bishop, *Introduction to Computer Security*, Addison-Wesley, 2004.
- [Björnson 2017]** E. Björnson, J. Hoydis, L. Sanguinetti, *Massive MIMO Networks: Spectral, Energy, and Hardware Efficiency*, Now Publishers, 2017.

- [Black 1995]** U. Black, *ATM Volume I: Foundation for Broadband Networks*, Prentice Hall, 1995.
- [Bluetooth 2020]** *The Bluetooth Special Interest Group*, <http://www.bluetooth.com/>
- [Blumenthal 2001]** M. Blumenthal, D. Clark, “Rethinking the Design of the Internet: The End-to-end Arguments vs. the Brave New World,” *ACM Transactions on Internet Technology*, Vol. 1, No. 1 (Aug. 2001), pp. 70–109.
- [Bochman 1984]** G. V. Bochmann, C. A. Sunshine, “Formal Methods in Communication Protocol Design,” *IEEE Transactions on Communications*, Vol. 28, No. 4 (Apr. 1980) pp. 624–631.
- [Bosshart 2013]** P. Bosshart, G. Gibb, H. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, M. Horowitz, “Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN,” *Proc. 2013 SIGCOMM Conference*, pp. 99–110.
- [Bosshart 2014]** P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker, “P4: Programming Protocol-Independent Packet Processors,” *Proc. 2014 ACM SIGCOMM Conference*, pp. 87–95.
- [Böttger 2018]** T. Böttger, F. Cuadrado, G. Tyson, I. Castro, S. Uhlig, Open connect everywhere: A glimpse at the internet ecosystem through the lens of the Netflix CDN, *Proc. 2018 ACM SIGCOMM Conference*.
- [Brakmo 1995]** L. Brakmo, L. Peterson, “TCP Vegas: End to End Congestion Avoidance on a Global Internet,” *IEEE Journal of Selected Areas in Communications*, Vol. 13, No. 8 (Oct. 1995), pp. 1465–1480.
- [Bryant 1988]** B. Bryant, “Designing an Authentication System: A Dialogue in Four Scenes,” <http://web.mit.edu/kerberos/www/dialogue.html>
- [Bush 1945]** V. Bush, “As We May Think,” *The Atlantic Monthly*, July 1945. <http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm>
- [Byers 1998]** J. Byers, M. Luby, M. Mitzenmacher, A. Rege, “A Digital Fountain Approach to Reliable Distribution of Bulk Data,” *Proc. 1998 ACM SIGCOMM Conference* (Vancouver, Canada, Aug. 1998), pp. 56–67.
- [Cable Labs 2019]** Cable Labs, “A Comparative Introduction to 4G and 5G Authentication,” <https://www.cablelabs.com/insights/a-comparative-introduction-to-4g-and-5g-authentication>
- [Caesar 2005b]** M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, J. van der Merwe, “Design and implementation of a Routing Control Platform,” *Proc. Networked Systems Design and Implementation* (May 2005).
- [Caesar 2005b]** M. Caesar, J. Rexford, “BGP Routing Policies in ISP Networks,” *IEEE Network Magazine*, Vol. 19, No. 6 (Nov. 2005).
- [CAIDA 2020]** Center for Applied Internet Data Analysis, [www.caida.org](http://www.caida.org)
- [Caldwell 2020]** C. Caldwell, “The Prime Pages,” <http://www.utm.edu/research/primes/prove>
- [Cardwell 2017]** N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, V. Jacobson. “BBR: congestion-based congestion control,” *Commun. ACM*, Vol. 60, No. 2 (Jan. 2017), pp. 58–66.
- [Casado 2007]** M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker, “Ethane: Taking Control of the Enterprise,” *Proc. 2007 ACM SIGCOMM Conference*, New York, pp. 1–12. See also *IEEE/ACM Trans. Networking*, Vol. 17, No. 4 (Aug. 2007), pp. 1270–1283.
- [Casado 2009]** M. Casado, M. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, S. Shenker, “Rethinking Enterprise Network Control,” *IEEE/ACM Transactions on Networking (ToN)*, Vol. 17, No. 4 (Aug. 2009), pp. 1270–1283.
- [Casado 2014]** M. Casado, N. Foster, A. Guha, “Abstractions for Software- Defined Networks,” *Communications of the ACM*, Vol. 57, No. 10, (Oct. 2014), pp. 86–95.
- [Cerf 1974]** V. Cerf, R. Kahn, “A Protocol for Packet Network Interconnection,” *IEEE Transactions on Communications Technology*, Vol. COM-22, No. 5, pp. 627–641.

- [CERT 2001–09]** CERT, “Advisory 2001–09: Statistical Weaknesses in TCP/IP Initial Sequence Numbers,” <http://www.cert.org/advisories/CA-2001-09.html>
- [CERT 2003–04]** CERT, “CERT Advisory CA-2003-04 MS-SQL Server Worm,” <http://www.cert.org/advisories/CA-2003-04.html>
- [CERT 2020]** The CERT division of the Software Engineering Institute, <https://www.sei.cmu.edu/about/divisions/cert>, 2020
- [CERT Filtering 2012]** CERT, “Packet Filtering for Firewall Systems,” [http://www.cert.org/tech\\_tips/packet\\_filtering.html](http://www.cert.org/tech_tips/packet_filtering.html)
- [Cert SYN 1996]** CERT, “Advisory CA-96.21: TCP SYN Flooding and IP Spoofing Attacks,” <http://www.cert.org/advisories/CA-1998-01.html>
- [Chandra 2007]** T. Chandra, R. Greisemer, J. Redstone, “Paxos Made Live: an Engineering Perspective,” *Proc. of 2007 ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 398–407.
- [Chao 2011]** C. Zhang, P. Dunghel, D. Wu, K. W. Ross, “Unraveling the BitTorrent Ecosystem,” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 22, No. 7 (July 2011).
- [Chen 2011]** Y. Chen, S. Jain, V. K. Adhikari, Z. Zhang, “Characterizing Roles of Front-End Servers in End-to-End Performance of Dynamic Content Distribution,” *Proc. 2011 ACM Internet Measurement Conference* (Berlin, Germany, Nov. 2011).
- [Chiu 1989]** D. Chiu, R. Jain, “Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks,” *Computer Networks and ISDN Systems*, Vol. 17, No. 1, pp. 1–14. [http://www.cs.wustl.edu/~jain/papers/cong\\_av.htm](http://www.cs.wustl.edu/~jain/papers/cong_av.htm)
- [Christiansen 2001]** M. Christiansen, K. Jeffay, D. Ott, F. D. Smith, “Tuning Red for Web Traffic,” *IEEE/ACM Transactions on Networking*, Vol. 9, No. 3 (June 2001), pp. 249–264.
- [Cichonski 2017]** J. Cichonski, J. Franklin, M. Bartock, Guide to LTE Security, NIST Special Publication 800–187, Dec. 2017.
- [Cisco 2012]** Cisco 2012, Data Centers, <http://www.cisco.com/go/dce>
- [Cisco 2020]** Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper.
- [Cisco 6500 2020]** Cisco Systems, “Cisco Catalyst 6500 Architecture White Paper,” [http://www.cisco.com/c/en/us/products/collateral/switches\\_catalyst-6500-series-switches/prod\\_white\\_paper0900aecd80673385.html](http://www.cisco.com/c/en/us/products/collateral/switches_catalyst-6500-series-switches/prod_white_paper0900aecd80673385.html)
- [Cisco 7600 2020]** Cisco Systems, “Cisco 7600 Series Solution and Design Guide,” [http://www.cisco.com/en/US/products/hw/routers/ps368/prod\\_technical\\_reference09186a0080092246.html](http://www.cisco.com/en/US/products/hw/routers/ps368/prod_technical_reference09186a0080092246.html)
- [Cisco 8500 2020]** Cisco Systems Inc., “Catalyst 8500 Campus Switch Router Architecture,” [http://www.cisco.com/univercd/cc/td/doc/product/l3sw/8540/rel\\_12\\_0/w5\\_6f/softcnfg/1cfg8500.pdf](http://www.cisco.com/univercd/cc/td/doc/product/l3sw/8540/rel_12_0/w5_6f/softcnfg/1cfg8500.pdf)
- [Cisco 12000 2020]** Cisco Systems Inc., “Cisco XR 12000 Series and Cisco 12000 Series Routers,” <http://www.cisco.com/en/US/products/ps6342/index.html>
- [Cisco Queue 2016]** Cisco Systems Inc., “Congestion Management Overview,” [http://www.cisco.com/en/US/docs/ios/12\\_2/qos/configuration/guide/qcfconmg.html](http://www.cisco.com/en/US/docs/ios/12_2/qos/configuration/guide/qcfconmg.html)
- [Cisco SYN 2016]** Cisco Systems Inc., “Defining Strategies to Protect Against TCP SYN Denial of Service Attacks,” [http://www.cisco.com/en/US/tech/tk828/technologies\\_tech\\_note09186a00800f67d5.shtml](http://www.cisco.com/en/US/tech/tk828/technologies_tech_note09186a00800f67d5.shtml)
- [Cisco TCAM 2014]** Cisco Systems Inc., “CAT 6500 and 7600 Series Routers and Switches TCAM Allocation Adjustment Procedures,” <http://www.cisco.com/c/en/us/support/docs/switches/catalyst-6500-series-switches/117712-problemsolution-cat6500-00.html>
- [Cisco VNI 2020]** Cisco Systems Inc., “Visual Networking Index,” <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html>

- [Claise 2019]** B. Calise, J. Clarke, J. Lindblad, *Network Programmability with YANG*, Pearson, 2019.
- [Clark 1988]** D. Clark, “The Design Philosophy of the DARPA Internet Protocols,” *Proc. 1988 ACM SIGCOMM Conference* (Stanford, CA, Aug. 1988).
- [Clark 1997]** D. Clark, “Interoperation, open interfaces and protocol architecture,” in *The Unpredictable Certainty*, The National Academies Press, 1997, pp. 133–144.
- [Clark 2005]** D. Clark, J. Wroclawski, K. R. Sollins, R. Braden, “Tussle in cyberspace: defining tomorrow’s internet,” *IEEE/ACM Trans. Networking*, Vol. 13, No. 3 (June 2005), pp. 462–475.
- [Clos 1953]** C. Clos, “A study of non-blocking switching networks,” *Bell System Technical Journal*, Vol. 32, No. 2 (Mar. 1953), pp. 406–424.
- [Cohen 2003]** B. Cohen, “Incentives to Build Robustness in BitTorrent,” First Workshop on the Economics of Peer-to-Peer Systems, Berkeley, CA, June 2003.
- [Colbach 2017]** G. Colbach, *Wireless Technologies: An introduction to Bluetooth and WiFi*, 2017.
- [Condoluci 2018]** M. Condoluci, T. Mahmoodi, “Softwareization and virtualization in 5G mobile networks: Benefits, trends and challenges,” *Computer Networks*, Vol. 146 (2018), pp. 65–84.
- [Cormen 2001]** T. H. Cormen, *Introduction to Algorithms*, 2nd Ed., MIT Press, Cambridge, MA, 2001.
- [Crow 1997]** B. Crow, I. Widjaja, J. Kim, P. Sakai, “IEEE 802.11 Wireless Local Area Networks,” *IEEE Communications Magazine* (Sept. 1997), pp. 116–126.
- [Cusumano 1998]** M. A. Cusumano, D. B. Yoffie, *Competing on Internet Time: Lessons from Netscape and Its Battle with Microsoft*, Free Press, New York, NY, 1998.
- [Czyz 2014]** J. Czyz, M. Allman, J. Zhang, S. Iekel-Johnson, E. Osterweil, M. Bailey, “Measuring IPv6 Adoption,” *Proc. ACM SIGCOMM 2014 Conference*, ACM, New York, NY, USA, pp. 87–98.
- [Dahlman 2018]** E. Dahlman, S. Parkvall, J. Skold, *5G NR: The Next Generation Wireless Access Technology*, Academic Press, 2018.
- [DAM 2020]** Digital Attack Map, <http://www.digitalattackmap.com>
- [Davie 2000]** B. Davie and Y. Rekhter, *MPLS: Technology and Applications*, Morgan Kaufmann Series in Networking, 2000.
- [DEC 1990]** Digital Equipment Corporation, “In Memoriam: J. C. R. Licklider 1915–1990,” SRC Research Report 61, Aug. 1990. <http://www.memex.org/licklider.pdf>
- [DeClercq 2002]** J. DeClercq, O. Paridaens, “Scalability Implications of Virtual Private Networks,” *IEEE Communications Magazine*, Vol. 40, No. 5 (May 2002), pp. 151–157.
- [Demers 1990]** A. Demers, S. Keshav, S. Shenker, “Analysis and Simulation of a Fair Queuing Algorithm,” *Internetworking: Research and Experience*, Vol. 1, No. 1 (1990), pp. 3–26.
- [dhc 2020]** IETF Dynamic Host Configuration working group homepage, <https://datatracker.ietf.org/wg/dhc/about/>
- [Diffie 1976]** W. Diffie, M. E. Hellman, “New Directions in Cryptography,” *IEEE Transactions on Information Theory*, Vol. IT-22 (1976), pp. 644–654.
- [Diggavi 2004]** S. N. Diggavi, N. Al-Dhahir, A. Stamoulis, R. Calderbank, “Great Expectations: The Value of Spatial Diversity in Wireless Networks,” *Proceedings of the IEEE*, Vol. 92, No. 2 (Feb. 2004).
- [Dilley 2002]** J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, B. Weihl, “Globally Distributed Content Delivery,” *IEEE Internet Computing* (Sept.–Oct. 2002).
- [Dimitropoulos 2007]** X. Dimitropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, K. C. Claffy, G. Riley, “AS Relationships: Inference and Validation,” *ACM Computer Communication Review*, Vol. 37, No. 1 (Jan. 2007).

- [DOCSIS3.1 2014]** *Data-Over-Cable Service Interface Specification, MAC and Upper Layer Protocols Interface Specification DOCSIS 3.1* (CM-SP-MULPIv3.1-104-141218), and *Data-Over-Cable Service Interface Specification, Physical Layer Specification DOCSIS 3.1* (CM-SP-PHYv3.1-104-141218), Dec. 2014.
- [Donahoo 2001]** M. Donahoo, K. Calvert, *TCP/IP Sockets in C: Practical Guide for Programmers*, Morgan Kaufman, 2001.
- [Droms 2002]** R. Droms, T. Lemon, *The DHCP Handbook*, 2nd edition, SAMS Publishing, 2002.
- [Eckel 2017]** C. Eckel, Using OpenDaylight, [https://www.youtube.com/watch?v=rAm48gVv8\\_A](https://www.youtube.com/watch?v=rAm48gVv8_A)
- [Economides 2017]** N. Economides, “A Case for Net Neutrality,” *IEEE Spectrum*, Dec. 2017, <https://spectrum.ieee.org/tech-talk/telecom/internet/a-case-for-net-neutrality>
- [Edney 2003]** J. Edney and W. A. Arbaugh, *Real 802.11 Security: Wi-Fi Protected Access and 802.11i*, Addison-Wesley Professional, 2003.
- [Eduroam 2020]** Eduroam, <https://www.eduroam.org/>
- [Eisenbud 2016]** D. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, Cilingiroglu, and B. Cheyney, W. Shang, J.D. Hosein, “Maglev: A Fast and Reliable Software Network Load Balancer,” *NSDI 2016*.
- [Ellis 1987]** H. Ellis, “The Story of Non-Secret Encryption,” <http://jya.com/ellisdoc.htm>
- [Erickson 2013]** D. Erickson, “The Beacon Openflow Controller,” 2nd *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking* (HotSDN ’13). ACM, New York, NY, USA, pp. 13–18.
- [Facebook 2014]** A. Andreyev, “Introducing Data Center Fabric, the Next-Generation Facebook Data Center Network,” <https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network>
- [Faloutsos 1999]** C. Faloutsos, M. Faloutsos, P. Faloutsos, “What Does the Internet Look Like? Empirical Laws of the Internet Topology,” *Proc. 1999 ACM SIGCOMM Conference* (Boston, MA, Aug. 1999).
- [Farrington 2010]** N. Farrington, G. Porter, S. Radhakrishnan, H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, A. Vahdat, “Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers,” *Proc. 2010 ACM SIGCOMM Conference*.
- [Faulhaber 2012]** G. Faulhaber, “The Economics of Network Neutrality: Are ‘Prophylactic’ Remedies to Nonproblems Needed?,” *Regulation*, Vol. 34, No. 4, p. 18, Winter 2011–2012.
- [FB 2014]** Facebook, “Introducing data center fabric, the next-generation Facebook data center network.” <https://engineering.fb.com/production-engineering/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/>
- [FB 2019]** Facebook, “Reinventing Facebook’s Data Center Network,” <https://engineering.fb.com/data-center-engineering/f16-minipack/>
- [FCC 2008]** US Federal Communications Commission, *Memorandum Opinion and Order: Formal Complaint of Free Press and Public Knowledge Against Comcast Corporation for Secretly Degrading Peer-to-Peer Applications*, FCC 08-083.
- [FCC 2015]** US Federal Communications Commission, Protecting and Promoting the Open Internet, Report and Order on Remand, Declaratory Ruling, and Order, GN Docket No. 14-28. (March 12, 2015), [https://apps.fcc.gov/edocs\\_public/attachmatch/FCC-15-24A1.pdf](https://apps.fcc.gov/edocs_public/attachmatch/FCC-15-24A1.pdf)
- [FCC 2017]** *Restoring Internet Freedom*, Declaratory Ruling, Report and Order and Order, WC Docket No. 17-108, December 14, 2017. [https://transition.fcc.gov/Daily\\_Releases/Daily\\_Business/2018/db0105/FCC-17-166A1.pdf](https://transition.fcc.gov/Daily_Releases/Daily_Business/2018/db0105/FCC-17-166A1.pdf)

- [Feamster 2004]** N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, K. van der Merwe, “The Case for Separating Routing from Routers,” *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, Sept. 2004.
- [Feamster 2004]** N. Feamster, J. Winick, J. Rexford, “A Model for BGP Routing for Network Engineering,” *Proc. 2004 ACM SIGMETRICS Conference* (New York, NY, June 2004).
- [Feamster 2005]** N. Feamster, H. Balakrishnan, “Detecting BGP Configuration Faults with Static Analysis,” *NSDI* (May 2005).
- [Feamster 2013]** N. Feamster, J. Rexford, E. Zegura, “The Road to SDN,” *ACM Queue*, Volume 11, Issue 12, (Dec. 2013).
- [Feamster 2018]** N. Feamster, J. Rexford, “Why (and How) Networks Should Run Themselves,” *Proc. 2018 ACM Applied Networking Research Workshop (ANRW ’18)*.
- [Feldmeier 1995]** D. Feldmeier, “Fast Software Implementation of Error Detection Codes,” *IEEE/ACM Transactions on Networking*, Vol. 3, No. 6 (Dec. 1995), pp. 640–652.
- [Fiber Broadband 2020]** Fiber Broadband Association <https://www.fiberbroadband.org/>
- [Fielding 2000]** R. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” 2000. PhD Thesis, UC Irvine, 2000.
- [FIPS 1995]** Federal Information Processing Standard, “Secure Hash Standard,” FIPS Publication 180-1. <http://www.itl.nist.gov/fipspubs/fip180-1.htm>
- [Floyd 1999]** S. Floyd, K. Fall, “Promoting the Use of End-to-End Congestion Control in the Internet,” *IEEE/ACM Transactions on Networking*, Vol. 6, No. 5 (Oct. 1998), pp. 458–472.
- [Floyd 2000]** S. Floyd, M. Handley, J. Padhye, J. Widmer, “Equation-Based Congestion Control for Unicast Applications,” *Proc. 2000 ACM SIGCOMM Conference* (Stockholm, Sweden, Aug. 2000).
- [Floyd 2016]** S. Floyd, “References on RED (Random Early Detection) Queue Management,” <http://www.icir.org/floyd/red.html>
- [Floyd Synchronization 1994]** S. Floyd, V. Jacobson, “Synchronization of Periodic Routing Messages,” *IEEE/ACM Transactions on Networking*, Vol. 2, No. 2 (Apr. 1997) pp. 122–136.
- [Floyd TCP 1994]** S. Floyd, “TCP and Explicit Congestion Notification,” *ACM SIGCOMM Computer Communications Review*, Vol. 24, No. 5 (Oct. 1994), pp. 10–23.
- [Fluhrer 2001]** S. Fluhrer, I. Mantin, A. Shamir, “Weaknesses in the Key Scheduling Algorithm of RC4,” *Eighth Annual Workshop on Selected Areas in Cryptography* (Toronto, Canada, Aug. 2002).
- [Ford 2005]** Bryan Ford, Pyda Srisuresh, and Dan Kegel. 2005. Peer-to-peer communication across network address translators. *In Proceedings of the annual conference on USENIX Annual Technical Conference (ATEC ’05)*.
- [Fortz 2000]** B. Fortz, M. Thorup, “Internet Traffic Engineering by Optimizing OSPF Weights,” *Proc. 2000 IEEE INFOCOM* (Tel Aviv, Israel, Apr. 2000).
- [Fortz 2002]** B. Fortz, J. Rexford, M. Thorup, “Traffic Engineering with Traditional IP Routing Protocols,” *IEEE Communication Magazine*, Vol. 40, No. 10 (Oct. 2002).
- [Frost 1994]** J. Frost, “BSD Sockets: A Quick and Dirty Primer,” <http://world.std.com/~jimf/papers/sockets/sockets.html>
- [Gao 2001]** L. Gao, J. Rexford, “Stable Internet Routing Without Global Coordination,” *IEEE/ACM Transactions on Networking*, Vol. 9, No. 6 (Dec. 2001), pp. 681–692.
- [Garfinkel 2003]** S. Garfinkel, “The End of End-to-End?,” MIT Technology Review, July 2003.

- [Gauthier 1999]** L. Gauthier, C. Diot, and J. Kurose, “End-to-End Transmission Control Mechanisms for Multiparty Interactive Applications on the Internet,” *Proc. 1999 IEEE INFOCOM* (New York, NY, Apr. 1999).
- [Gieben 2004]** M. Gieben, “DNSSEC,” *The Internet Protocol Journal*, 7 [2] (June 2004), <http://ipj.dreamhosters.com/internet-protocol-journal/issues/back-issues/>
- [Giust 2015]** F. Giust, L. Cominardi and C. J. Bernardos, “Distributed mobility management for future 5G networks: overview and analysis of existing approaches,” in *IEEE Communications Magazine*, Vol. 53, No. 1, pp. 142–149, January 2015.
- [Goldsmith 2005]** A. Goldsmith, *Wireless Communications*, Cambridge University Press, 2005.
- [Goodman 1997]** David J. Goodman, *Wireless Personal Communications Systems*, Prentice-Hall, 1997.
- [Google CDN 2020]** Google Data Center Locations <https://cloud.google.com/cdn/docs/locations>
- [Google IPv6 2020]** Google Inc. “IPv6 Statistics,” <https://www.google.com/intl/en/ipv6/statistics.html>
- [Google Locations 2020]** Google data centers. <http://www.google.com/corporate/datacenter/locations.html>
- [Goralski 1999]** W. Goralski, *Frame Relay for High-Speed Networks*, John Wiley, New York, 1999.
- [Greenberg 2009a]** A. Greenberg, J. Hamilton, D. Maltz, P. Patel, “The Cost of a Cloud: Research Problems in Data Center Networks,” *ACM Computer Communications Review* (Jan. 2009).
- [Greenberg 2009b]** A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, S. Sengupta, “VL2: A Scalable and Flexible Data Center Network,” *Proc. 2009 ACM SIGCOMM Conference*.
- [Greenberg 2015]** A. Greenberg, “SDN for the Cloud,” 2015 ACM SIGCOMM Conference 2015 Keynote Address, <http://conferences.sigcomm.org/sigcomm/2015/pdf/papers/keynote.pdf>
- [GSMA 2018a]** GSM Association, “Guidelines for IPX Provider networks,” Document IR.34, Version 14.0, August 2018.
- [GSMA 2018b]** GSM Association, “Migration from Physical to Virtual Network Functions: Best Practices and Lessons Learned,” July 2019.
- [GSMA 2019a]** GSM Association, “LTE and EPC Roaming Guidelines,” Document IR.88, June 2019.
- [GSMA 2019b]** GSM Association, “IMS Roaming, Interconnection and Interworking Guidelines,” Document IR.65, April 2019.
- [GSMA 2019c]** GSM Association, “5G Implementation Guidelines,” July 2019.
- [Gude 2008]** N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, “NOX: Towards an Operating System for Networks,” *ACM SIGCOMM Computer Communication Review*, July 2008.
- [Guo 2009]** C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, S. Lu, “BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers,” *Proc. 2009 ACM SIGCOMM Conference*.
- [Guo 2016]** C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, M. Lipshteyn, “RDMA over Commodity Ethernet at Scale,” *Proc. 2016 ACM SIGCOMM Conference*.
- [Gupta 2001]** P. Gupta, N. McKeown, “Algorithms for Packet Classification,” *IEEE Network Magazine*, Vol. 15, No. 2 (Mar./Apr. 2001), pp. 24–32.
- [Gupta 2014]** A. Gupta, L. Vanbever, M. Shahbaz, S. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, E. Katz-Bassett, “SDX: A Software Defined Internet Exchange,” *Proc. 2014 ACM SIGCOMM Conference* (Aug. 2014), pp. 551–562.
- [Ha 2008]** S. Ha, I. Rhee, L. Xu, “CUBIC: A New TCP-Friendly High-Speed TCP Variant,” *ACM SIGOPS Operating System Review*, 2008.

- [Halabi 2000]** S. Halabi, *Internet Routing Architectures*, 2nd Ed., Cisco Press, 2000.
- [Hamzeh 2015]** B. Hamzeh, M. Toy, Y. Fu and J. Martin, “DOCSIS 3.1: scaling broadband cable to Gigabit speeds,” *IEEE Communications Magazine*, Vol. 53, No. 3, pp. 108–113, March 2015.
- [Hanabali 2005]** A. A. Hanabali, E. Altman, P. Nain, “A Survey of TCP over Ad Hoc Networks,” *IEEE Commun. Surveys and Tutorials*, Vol. 7, No. 3 (2005), pp. 22–36.
- [He 2015]** K. He , E. Rozner , K. Agarwal , W. Felter , J. Carter , A. Akella, “Presto: Edge-based Load Balancing for Fast Datacenter Networks,” *Proc. 2015 ACM SIGCOMM Conference*.
- [Heidemann 1997]** J. Heidemann, K. Obraczka, J. Touch, “Modeling the Performance of HTTP over Several Transport Protocols,” *IEEE/ACM Transactions on Networking*, Vol. 5, No. 5 (Oct. 1997), pp. 616–630.
- [Held 2001]** G. Held, *Data Over Wireless Networks: Bluetooth, WAP, and Wireless LANs*, McGraw-Hill, 2001.
- [Holland 2001]** G. Holland, N. Vaidya, V. Bahl, “A Rate-Adaptive MAC Protocol for Multi-Hop Wireless Networks,” *Proc. 2001 ACM Int. Conference of Mobile Computing and Networking* (Rome, Italy, July 2001).
- [Hollot 2002]** C.V. Hollot, V. Misra, D. Towsley, W. Gong, “Analysis and Design of Controllers for AQM Routers Supporting TCP Flows,” *IEEE Transactions on Automatic Control*, Vol. 47, No. 6 (June 2002), pp. 945–959.
- [Hong 2012]** C.Y. Hong, M. Caesar, P. B. Godfrey, “Finishing Flows Quickly with Preemptive Scheduling,” *Proc. 2012 ACM SIGCOMM Conference*.
- [Hong 2013]** C. Hong, S. Kandula, R. Mahajan, M.Zhang, V. Gill, M. Nanduri, R. Wattenhofer, “Achieving High Utilization with Software-driven WAN,” *Proc. ACM SIGCOMM Conference* (Aug. 2013), pp.15–26.
- [Hong 2018]** C. Hong et al., “B4 and after: managing hierarchy, partitioning, and asymmetry for availability and scale in Google’s software-defined WAN,” *Proc. 2018 ACM SIGCOMM Conference*, pp. 74–87.
- [HTTP/3 2020]** M. Bishop. Ed, “Hypertext Transfer Protocol Version 3 (HTTP/3),” Internet Draft draft-ietf-quic-http-23, expires March 15, 2020.
- [Huang 2002]** C. Haung, V. Sharma, K. Owens, V. Makam, “Building Reliable MPLS Networks Using a Path Protection Mechanism,” *IEEE Communications Magazine*, Vol. 40, No. 3 (Mar. 2002), pp. 156–162.
- [Huang 2008]** C. Huang, J. Li, A. Wang, K. W. Ross, “Understanding Hybrid CDN-P2P: Why Limelight Needs Its Own Red Swoosh,” *Proc. 2008 NOSSDAV*, Braunschweig, Germany.
- [Huang 2013]** J. Huang, F. Qian, Y. Guo, Yu. Zhou, Q. Xu, Z. Mao, S. Sen, O. Spatscheck, “An in-depth study of LTE: effect of network protocol and application behavior on performance,” *Proc. 2013 ACM SIGCOMM Conference*.
- [Huitema 1998]** C. Huitema, *IPv6: The New Internet Protocol*, 2nd Ed., Prentice Hall, Englewood Cliffs, NJ, 1998.
- [Huston 1999a]** G. Huston, “Interconnection, Peering, and Settlements—Part I,” *The Internet Protocol Journal*, Vol. 2, No. 1 (Mar. 1999).
- [Huston 2004]** G. Huston, “NAT Anatomy: A Look Inside Network Address Translators,” *The Internet Protocol Journal*, Vol. 7, No. 3 (Sept. 2004).
- [Huston 2008b]** G. Huston, G. Michaelson, “IPv6 Deployment: Just where are we?” <http://www.potaroo.net/ispcol/2008-04/ipv6.html>
- [Huston 2011a]** G. Huston, “A Rough Guide to Address Exhaustion,” *The Internet Protocol Journal*, Vol. 14, No. 1 (Mar. 2011).
- [Huston 2011b]** G. Huston, “Transitioning Protocols,” *The Internet Protocol Journal*, Vol. 14, No. 1 (Mar. 2011).
- [Huston 2012]** G. Huston, “A Quick Primer on Internet Peering and Settlements,” April 2012, <http://www.potaroo.net/ispcol/2012-04/interconnection-primer.html>

- [Huston 2017]** G. Huston, “BBR, the new kid on the TCP block,” <https://blog.apnic.net/2017/05/09/bbr-new-kid-tcp-block/>
- [Huston 2017]** G. Huston, “An Opinion in Defence of NAT,” <https://www.potaroo.net/ispcol/2017-09/natdefence.html>
- [Huston 2019]** G. Huston, “Addressing 2018,” <https://www.potaroo.net/ispcol/2019-01/addr2018.html>
- [Huston 2019a]** G. Huston, “Happy Birthday BGP,” June 2019, <http://www.potaroo.net/ispcol/2019-06/bgp30.html>
- [Huston 2019b]** G. Huston, “BGP in 2018, Part 1 - The BGP Table,” <https://www.potaroo.net/ispcol/2019-01/bgp2018-part1.html>
- [Hwang 2009]** T. Hwang, C. Yang, G. Wu, S. Li and G. Ye Li, “OFDM and Its Wireless Applications: A Survey,” *IEEE Transactions on Vehicular Technology*, Vol. 58, No. 4, pp. 1673–1694, May 2009.
- [IAB 2020]** Internet Architecture Board homepage, <http://www.iab.org/>
- [IANA 2020]** Internet Assigned Names Authority, <https://www.iana.org/>
- [IANA Protocol Numbers 2016]** Internet Assigned Numbers Authority, Protocol Numbers, <http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>
- [ICANN 2020]** The Internet Corporation for Assigned Names and Numbers homepage, <http://www.icann.org>
- [IEEE 802 2020]** IEEE 802 LAN/MAN Standards Committee homepage, <http://www.ieee802.org/>
- [IEEE 802.11 1999]** IEEE 802.11, “1999 Edition (ISO/IEC 8802-11: 1999) IEEE Standards for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Network— Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification,” <http://standards.ieee.org/getieee802/download/802.11-1999.pdf>
- [IEEE 802.1q 2005]** IEEE, “IEEE Standard for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks,” <http://standards.ieee.org/getieee802/download/802.1Q-2005.pdf>
- [IEEE 802.3 2020]** IEEE, “IEEE 802.3 CSMA/CD (Ethernet),” <http://grouper.ieee.org/groups/802/3/>
- [IEEE 802.5 2012]** IEEE, IEEE 802.5 homepage, <http://www.ieee802.org/5/www8025org/>
- [IEEE 802.11 2020]** IEEE 802.11 Wireless Local Area Networks, the Working Group for WLAN Standards, <http://www.ieee802.org/11/>
- [IETF 2020]** Internet Engineering Task Force homepage, <http://www.ietf.org>
- [IETF QUIC2020]** Internet Engineering Task Force, QUIC Working Group, <https://datatracker.ietf.org/wg/quic/about/>
- [Intel 2020]** Intel Corp., “Intel 710 Ethernet Adapter,” <http://www.intel.com/content/www/us/en/ethernet-products/converged-network-adapters/ethernet-xl710.html>
- [ISC 2020]** Internet Systems Consortium homepage, <http://www.isc.org>
- [ITU 2005a]** International Telecommunication Union, “ITU-T X.509, The Directory: Public-key and attribute certificate frameworks” (Aug. 2005).
- [ITU 2014]** ITU, “G.fast broadband standard approved and on the market,” [http://www.itu.int/net/pressoffice/press\\_releases/2014/70.aspx](http://www.itu.int/net/pressoffice/press_releases/2014/70.aspx)
- [ITU 2020]** The ITU homepage, <http://www.itu.int>
- [Iyer 2008]** S. Iyer, R. R. Kompella, N. McKeown, “Designing Packet Buffers for Router Line Cards,” *IEEE/ACM Transactions on Networking*, Vol. 16, No. 3 (June 2008), pp. 705–717.
- [Jacobson 1988]** V. Jacobson, “Congestion Avoidance and Control,” *Proc. 1988 ACM SIGCOMM Conference* (Stanford, CA, Aug. 1988), pp. 314–329.

- [Jain 1986]** R. Jain, “A Timeout-Based Congestion Control Scheme for Window Flow-Controlled Networks,” *IEEE Journal on Selected Areas in Communications SAC-4*, 7 (Oct. 1986).
- [Jain 1989]** R. Jain, “A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks,” *ACM SIGCOMM Computer Communications Review*, Vol. 19, No. 5 (1989), pp. 56–71.
- [Jain 1994]** R. Jain, *FDDI Handbook: High-Speed Networking Using Fiber and Other Media*, Addison-Wesley, Reading, MA, 1994.
- [Jain 1996]** R. Jain, S. Kalyanaraman, S. Fahmy, R. Goyal, S. Kim, “Tutorial Paper on ABR Source Behavior,” *ATM Forum/96-1270*, Oct. 1996. <http://www.cse.wustl.edu/~jain/atmfv/ftp/atm96-1270.pdf>
- [Jain 2013]** S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hözle, S. Stuart, A. Vahdat, “B4: Experience with a Globally Deployed Software Defined Wan,” *Proc. 2013 ACM SIGCOMM Conference*, pp. 3–14.
- [Jimenez 1997]** D. Jimenez, “Outside Hackers Infiltrate MIT Network, Compromise Security,” *The Tech*, Vol. 117, No. 49 (Oct. 1997), p. 1, <http://www-tech.mit.edu/V117/N49/hackers.49n.html>
- [Juniper MX2020 2020]** Juniper Networks, “MX2020 and MX2010 3D Universal Edge Routers,” <https://www.juniper.net/us/en/products-services/routing/mx-series/mx2020/>
- [Kaaranen 2001]** H. Kaaranen, S. Naghian, L. Laitinen, A. Ahtiainen, V. Niemi, *Networks: Architecture, Mobility and Services*, New York: John Wiley & Sons, 2001.
- [Kahn 1967]** D. Kahn, *The Codebreakers: The Story of Secret Writing*, The Macmillan Company, 1967.
- [Kahn 1978]** R. E. Kahn, S. Gronemeyer, J. Burchfiel, R. Kunzelman, “Advances in Packet Radio Technology,” *Proc. IEEE*, Vol. 66, No. 11 (Nov. 1978), pp. 1468–1496.
- [Kberman 1997]** A. Kberman, L. Monteban, “WaveLAN-II: A High-Performance Wireless LAN for the Unlicensed Band,” *Bell Labs Technical Journal* (Summer 1997), pp. 118–133.
- [Kar 2000]** K. Kar, M. Kodialam, T. V. Lakshman, “Minimum Interference Routing of Bandwidth Guaranteed Tunnels with MPLS Traffic Engineering Applications,” *IEEE J. Selected Areas in Communications* (Dec. 2000).
- [Karn 1987]** P. Karn, C. Partridge, “Improving Round-Trip Time Estimates in Reliable Transport Protocols,” *Proc. 1987 ACM SIGCOMM Conference*.
- [Karol 1987]** M. Karol, M. Hluchyj, A. Morgan, “Input Versus Output Queuing on a Space-Division Packet Switch,” *IEEE Transactions on Communications*, Vol. 35, No. 12 (Dec. 1987), pp. 1347–1356.
- [Kaufman 2002]** C. Kaufman, R. Perlman, M. Speciner, *Network Security: Private Communication in a Public World*, 2nd edition, Prentice Hall, 2002.
- [Kelly 1998]** F. P. Kelly, A. Maulloo, D. Tan, “Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability,” *J. Operations Res. Soc.*, Vol. 49, No. 3 (Mar. 1998), pp. 237–252.
- [Kim 2008]** C. Kim, M. Caesar, J. Rexford, “Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises,” *Proc. 2008 ACM SIGCOMM Conference* (Seattle, WA, Aug. 2008).
- [Kleinrock 1961]** L. Kleinrock, “Information Flow in Large Communication Networks,” *RLE Quarterly Progress Report*, July 1961.
- [Kleinrock 1964]** L. Kleinrock, *1964 Communication Nets: Stochastic Message Flow and Delay*, McGraw-Hill, New York, NY, 1964.
- [Kleinrock 1975]** L. Kleinrock, *Queueing Systems*, Vol. 1, John Wiley, New York, 1975.
- [Kleinrock 1975b]** L. Kleinrock, F. A. Tobagi, “Packet Switching in Radio Channels: Part I—Carrier Sense Multiple-Access Modes and Their Throughput-Delay Characteristics,” *IEEE Transactions on Communications*, Vol. 23, No. 12 (Dec. 1975), pp. 1400–1416.

- [Kleinrock 1976]** L. Kleinrock, *Queueing Systems, Vol. 2*, John Wiley, New York, 1976.
- [Kleinrock 2004]** L. Kleinrock, “The Birth of the Internet,” <http://www.lk.cs.ucla.edu/LK/Inet/birth.html>
- [Kleinrock 2018]** L. Kleinrock, “Internet congestion control using the power metric: Keep the pipe just full, but no fuller,” *Ad Hoc Networks*, Vol. 80, 2018, pp. 142–157.
- [Kohler 2006]** E. Kohler, M. Handley, S. Floyd, “DDCP: Designing DCCP: Congestion Control Without Reliability,” *Proc. 2006 ACM SIGCOMM Conference* (Pisa, Italy, Sept. 2006).
- [Kohlios 2018]** C. Kohlios, T. Hayajneh, “A Comprehensive Attack Flow Model and Security Analysis for Wi-Fi and WPA3,” *Electronics*, Vol. 7, No. 11, 2018.
- [Kolding 2003]** T. Kolding, K. Pedersen, J. Wigard, F. Frederiksen, P. Mogensen, “High Speed Downlink Packet Access: WCDMA Evolution,” *IEEE Vehicular Technology Society News* (Feb. 2003), pp. 4–10.
- [Koponen 2010]** T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, S. Shenker, “Onix: A Distributed Control Platform for Large-Scale Production Networks,” *9th USENIX conference on Operating systems design and implementation (OSDI'10)*, pp. 1–6.
- [Koponen 2011]** T. Koponen, S. Shenker, H. Balakrishnan, N. Feamster, I. Ganichev, A. Ghodsi, P. B. Godfrey, N. McKeown, G. Parulkar, B. Raghavan, J. Rexford, S. Arianfar, D. Kuptsov, “Architecting for Innovation,” *ACM Computer Communications Review*, 2011.
- [Korhonen 2003]** J. Korhonen, *Introduction to 3G Mobile Communications*, 2nd edition, Artech House, 2003.
- [Koziol 2003]** J. Koziol, *Intrusion Detection with Snort*, Sams Publishing, 2003.
- [Kreutz 2015]** D. Kreutz, F.M.V. Ramos, P. Esteves Verissimo, C. Rothenberg, S. Azodolmolky, S. Uhlig, “Software-Defined Networking: A Comprehensive Survey,” *Proceedings of the IEEE*, Vol. 103, No. 1 (Jan. 2015), pp. 14–76. This paper is also being updated at <https://github.com/SDN-Survey/latex/wiki>
- [Krishnamurthy 2001]** B. Krishnamurthy, J. Rexford, *Web Protocols and Practice: HTTP/1.1, Networking Protocols, and Traffic Measurement*, Addison-Wesley, Boston, MA, 2001.
- [Kühlewind 2013]** M. Kühlewind, S. Neuner, B. Trammell, “On the state of ECN and TCP options on the internet,” *Proc. 14th International Conference on Passive and Active Measurement (PAM'13)*, pp. 135–144.
- [Kulkarni 2005]** S. Kulkarni, C. Rosenberg, “Opportunistic Scheduling: Generalizations to Include Multiple Constraints, Multiple Interfaces, and Short Term Fairness,” *Wireless Networks*, 11 (2005), pp. 557–569.
- [Kumar 2006]** R. Kumar, K.W. Ross, “Optimal Peer-Assisted File Distribution: Single and Multi-Class Problems,” *IEEE Workshop on Hot Topics in Web Systems and Technologies* (Boston, MA, 2006).
- [Labovitz 1997]** C. Labovitz, G. R. Malan, F. Jahanian, “Internet Routing Instability,” *Proc. 1997 ACM SIGCOMM Conference* (Cannes, France, Sept. 1997), pp. 115–126.
- [Labovitz 2010]** C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, F. Jahanian, “Internet Inter-Domain Traffic,” *Proc. 2010 ACM SIGCOMM Conference*.
- [Labrador 1999]** M. Labrador, S. Banerjee, “Packet Dropping Policies for ATM and IP Networks,” *IEEE Communications Surveys*, Vol. 2, No. 3 (Third Quarter 1999), pp. 2–14.
- [Lacage 2004]** M. Lacage, M.H. Manshaei, T. Turletti, “IEEE 802.11 Rate Adaptation: A Practical Approach,” *ACM Int. Symposium on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWiM)* (Venice, Italy, Oct. 2004).
- [Lakhina 2005]** A. Lakhina, M. Crovella, C. Diot, “Mining Anomalies Using Traffic Feature Distributions,” *Proc. 2005 ACM SIGCOMM Conference*.

- [Lakshman 1997]** T. V. Lakshman, U. Madhow, “The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss,” *IEEE/ACM Transactions on Networking*, Vol. 5, No. 3 (1997), pp. 336–350.
- [Lakshman 2004]** T. V. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, T. Woo, “The SoftRouter Architecture,” *Proc. 3rd ACM Workshop on Hot Topics in Networks (Hotnets-III)*, Nov. 2004.
- [Lam 1980]** S. Lam, “A Carrier Sense Multiple Access Protocol for Local Networks,” *Computer Networks*, Vol. 4 (1980), pp. 21–32.
- [Lamport 1989]** L. Lamport, “The Part-Time Parliament,” Technical Report 49, Systems Research Center, Digital Equipment Corp., Palo Alto, Sept. 1989.
- [Lampson 1983]** Lampson, Butler W. “Hints for computer system design,” *ACM SIGOPS Operating Systems Review*, Vol. 17, No. 5, 1983.
- [Lampson 1996]** B. Lampson, “How to Build a Highly Available System Using Consensus,” *Proc. 10th International Workshop on Distributed Algorithms (WDAG '96)*, Özalp Babaoglu and Keith Marzullo (Eds.), Springer-Verlag, pp. 1–17.
- [Langley 2017]** A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W. Chang, Z. Shi, “The QUIC Transport Protocol: Design and Internet-Scale Deployment,” *Proc. 2017 ACM SIGCOMM Conference*.
- [Lawton 2001]** G. Lawton, “Is IPv6 Finally Gaining Ground?” *IEEE Computer Magazine* (Aug. 2001), pp. 11–15.
- [Leighton 2009]** T. Leighton, “Improving Performance on the Internet,” *Communications of the ACM*, Vol. 52, No. 2 (Feb. 2009), pp. 44–51.
- [Leiner 1998]** B. Leiner, V. Cerf, D. Clark, R. Kahn, L. Kleinrock, D. Lynch, J. Postel, L. Roberts, S. Woolf, “A Brief History of the Internet,” <http://www.isoc.org/internet/history/brief.html>
- [Leung 2006]** K. Leung, V. O. K. Li, “TCP in Wireless Networks: Issues, Approaches, and Challenges,” *IEEE Commun. Surveys and Tutorials*, Vol. 8, No. 4 (2006), pp. 64–79.
- [Levin 2012]** D. Levin, A. Wundsam, B. Heller, N. Handigol, A. Feldmann, “Logically Centralized?: State Distribution Trade-offs in Software Defined Networks,” *Proc. First Workshop on Hot Topics in Software Defined Networks* (Aug. 2012), pp. 1–6.
- [Li 2004]** L. Li, D. Alderson, W. Willinger, J. Doyle, “A First-Principles Approach to Understanding the Internet’s Router-Level Topology,” *Proc. 2004 ACM SIGCOMM Conference* (Portland, OR, Aug. 2004).
- [Li 2007]** J. Li, M. Guidero, Z. Wu, E. Purpus, T. Ehrenkranz, “BGP Routing Dynamics Revisited.” *ACM Computer Communication Review* (Apr. 2007).
- [Li 2015]** S. Q. Li, “Building Softcom Ecosystem Foundation,” Open Networking Summit, 2015.
- [Li 2017]** Z. Li, W. Wang, C. Wilson, J. Chen, C. Qian, T. Jung, L. Zhang, K. Liu, X. Li, Y. Liu, “FBS-Radar: Uncovering Fake Base Stations at Scale in the Wild,” *ISOC Symposium on Network and Distributed System Security (NDSS)*, February 2017.
- [Li 2018]** Z. Li, D. Levin, N. Spring, B. Bhattacharjee, “Internet anycast: performance, problems, & potential,” *Proc. 2018 ACM SIGCOMM Conference*, pp. 59–73.
- [Lin 2001]** Y. Lin, I. Chlamtac, *Wireless and Mobile Network Architectures*, John Wiley and Sons, New York, NY, 2001.
- [Liogkas 2006]** N. Liogkas, R. Nelson, E. Kohler, L. Zhang, “Exploiting BitTorrent for Fun (but Not Profit),” *6th International Workshop on Peer-to-Peer Systems (IPTPS 2006)*.
- [Liu 2003]** J. Liu, I. Matta, M. Crovella, “End-to-End Inference of Loss Nature in a Hybrid Wired/Wireless Environment,” *Proc. WiOpt’03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*.

- [Locher 2006]** T. Locher, P. Moor, S. Schmid, R. Wattenhofer, “Free Riding in BitTorrent is Cheap,” *Proc. ACM HotNets 2006* (Irvine CA, Nov. 2006).
- [Madhyastha 2017]** H. Madhyastha, “A Case Against Net Neutrality,” *IEEE Spectrum*, Dec. 2017, <https://spectrum.ieee.org/tech-talk/telecom/internet/a-case-against-net-neutrality>
- [Mahdavi 1997]** J. Mahdavi, S. Floyd, “TCP-Friendly Unicast Rate-Based Flow Control,” unpublished note (Jan. 1997).
- [Mao 2002]** Z. Mao, C. Cranor, F. Douglis, M. Rabinovich, O. Spatscheck, J. Wang, “A Precise and Efficient Evaluation of the Proximity Between Web Clients and Their Local DNS Servers,” *2002 USENIX Annual Technical Conference*, pp. 229–242.
- [Mathis 1997]** M. Mathis, J. Semke, J. Mahdavi, T. Ott, T. 1997, “The macroscopic behavior of the TCP congestion avoidance algorithm,” *ACM SIGCOMM Computer Communication Review*, 27(3): pp. 67–82.
- [MaxMind 2020]** <http://www.maxmind.com/app/ip-location>
- [McKeown 1997a]** N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, M. Horowitz, “The Tiny Tera: A Packet Switch Core,” *IEEE Micro Magazine* (Jan.–Feb. 1997).
- [McKeown 1997b]** N. McKeown, “A Fast Switched Backplane for a Gigabit Switched Router,” *Business Communications Review*, Vol. 27, No. 12. [http://tiny-tera.stanford.edu/~nickm/papers/cisco\\_fasts\\_wp.pdf](http://tiny-tera.stanford.edu/~nickm/papers/cisco_fasts_wp.pdf)
- [McKeown 2008]** N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (Mar. 2008), pp. 69–74.
- [McQuillan 1980]** J. McQuillan, I. Richer, E. Rosen, “The New Routing Algorithm for the Arpanet,” *IEEE Transactions on Communications*, Vol. 28, No. 5 (May 1980), pp. 711–719.
- [Metcalfe 1976]** R. M. Metcalfe, D. R. Boggs. “Ethernet: Distributed Packet Switching for Local Computer Networks,” *Communications of the Association for Computing Machinery*, Vol. 19, No. 7 (July 1976), pp. 395–404.
- [Meyers 2004]** A. Myers, T. Ng, H. Zhang, “Rethinking the Service Model: Scaling Ethernet to a Million Nodes,” *ACM Hotnets Conference*, 2004.
- [Mijumbi 2016]** R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck and R. Boutaba, “Network Function Virtualization: State-of-the-Art and Research Challenges,” *IEEE Communications Surveys & Tutorials*, Vol. 18, No. 1, pp. 236–262, 2016.
- [MIT TR 2019]** MIT Technology Review, “How a quantum computer could break 2048-bit RSA encryption in 8 hours,” May 2019, <https://www.technologyreview.com/s/613596/how-a-quantum-computer-could-break-2048-bit-rsa-encryption-in-8-hours/>
- [Mittal 2015]** R. Mittal, V. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, D. Zats, “TIMELY: RTT-based Congestion Control for the Datacenter,” *Proc. 2015 ACM SIGCOMM Conference*, pp. 537–550.
- [Mockapetris 1988]** P. V. Mockapetris, K. J. Dunlap, “Development of the Domain Name System,” *Proc. 1988 ACM SIGCOMM Conference* (Stanford, CA, Aug. 1988).
- [Mockapetris 2005]** P. Mockapetris, Sigcomm Award Lecture, video available at <http://www.postel.org/sigcomm>
- [Molinero-Fernandez 2002]** P. Molinaro-Fernandez, N. McKeown, H. Zhang, “Is IP Going to Take Over the World (of Communications)?” *Proc. 2002 ACM Hotnets*.
- [Molle 1987]** M. L. Molle, K. Sohraby, A. N. Venetsanopoulos, “Space-Time Models of Asynchronous CSMA Protocols for Local Area Networks,” *IEEE Journal on Selected Areas in Communications*, Vol. 5, No. 6 (1987), pp. 956–968.
- [Moshref 2016]** M. Moshref, M. Yu, R. Govindan, A. Vahdat, “Trumpet: Timely and Precise Triggers in Data Centers,” *Proc. 2016 ACM SIGCOMM Conference*.

- [Motorola 2007]** Motorola, “Long Term Evolution (LTE): A Technical Overview,” [http://www.motorola.com/staticfiles/Business/Solutions/Industry%20Solutions/Service%20Providers/Wireless%20Operators/LTE/\\_Document/Static%20Files/6834\\_MotDoc\\_New.pdf](http://www.motorola.com/staticfiles/Business/Solutions/Industry%20Solutions/Service%20Providers/Wireless%20Operators/LTE/_Document/Static%20Files/6834_MotDoc_New.pdf)
- [Mouly 1992]** M. Mouly, M. Pautet, *The GSM System for Mobile Communications*, Cell and Sys, Palaiseau, France, 1992.
- [Moy 1998]** J. Moy, *OSPF: Anatomy of An Internet Routing Protocol*, Addison-Wesley, Reading, MA, 1998.
- [Mysore 2009]** R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, A. Vahdat, “PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric,” *Proc. 2009 ACM SIGCOMM Conference*.
- [Nahum 2002]** E. Nahum, T. Barzilai, D. Kandlur, “Performance Issues in WWW Servers,” *IEEE/ACM Transactions on Networking*, Vol 10, No. 1 (Feb. 2002).
- [Narayan 2018]** A. Narayan, F. Cangialosi, D. Raghavan, P. Goyal, S. Narayana, R. Mittal, M. Alizadeh, H. Balakrishnan, “Restructuring endpoint congestion control,” *Proc. ACM SIGCOMM 2018 Conference*, pp. 30–43.
- [Netflix Open Connect 2020]** Netflix Open Connect CDN, 2016, <https://openconnect.netflix.com/>
- [Netflix Video 1]** Designing Netflix’s Content Delivery System, D. Fullager, 2014, <https://www.youtube.com/watch?v=LkLLpYdDINA>
- [Netflix Video 2]** Scaling the Netflix Global CDN, D. Temkin, 2015, [https://www.youtube.com/watch?v=tbqcsHg-Q\\_o](https://www.youtube.com/watch?v=tbqcsHg-Q_o)
- [Neumann 1997]** R. Neumann, “Internet Routing Black Hole,” *The Risks Digest: Forum on Risks to the Public in Computers and Related Systems*, Vol. 19, No. 12 (May 1997). <http://catless.ncl.ac.uk/Risks/19.12.html#subj1.1>
- [Neville-Neil 2009]** G. Neville-Neil, “Whither Sockets?” *Communications of the ACM*, Vol. 52, No. 6 (June 2009), pp. 51–55.
- [Nguyen 2016]** T. Nguyen, C. Bonnet and J. Harri, “SDN-based distributed mobility management for 5G networks,” *2016 IEEE Wireless Communications and Networking Conference*, Doha, 2016, pp. 1–7.
- [Nichols 2012]** K. Nichols, V. Jacobson. Controlling Queue Delay. *ACM Queue*, Vol. 10, No. 5, May 2012.
- [Nicholson 2006]** A Nicholson, Y. Chawathe, M. Chen, B. Noble, D. Wetherall, “Improved Access Point Selection,” *Proc. 2006 ACM Mobicom Conference* (Uppsala Sweden, 2006).
- [Nielsen 1997]** H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud’hommeaux, H. W. Lie, C. Lilley, “Network Performance Effects of HTTP/1.1, CSS1, and PNG,” *W3C Document*, 1997 (also appears in *Proc. 1997 ACM SIGCOM Conference* (Cannes, France, Sept 1997), pp. 155–166.
- [NIST 2001]** National Institute of Standards and Technology, “Advanced Encryption Standard (AES),” Federal Information Processing Standards 197, Nov. 2001, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
- [NIST IPv6 2020]** US National Institute of Standards and Technology, “Estimating IPv6 & DNSSEC Deployment SnapShots,” <http://fedv6-deployment.antd.nist.gov/snap-all.html>
- [Nmap 2020]** Nmap homepage, <https://nmap.org>
- [Nonnenmacher 1998]** J. Nonnenmacher, E. Biersak, D. Towsley, “Parity-Based Loss Recovery for Reliable Multicast Transmission,” *IEEE/ACM Transactions on Networking*, Vol. 6, No. 4 (Aug. 1998), pp. 349–361.
- [Noormohammadpour 2018]** M. Noormohammadpour, C. Raghavendra, Cauligi, “Datacenter Traffic Control: Understanding Techniques and Trade-offs,” *IEEE Communications Surveys & Tutorials*, Vol. 20 (2018), pp. 1492–1525.

- [Nygren 2010]** Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun, “The Akamai Network: A Platform for High-performance Internet Applications,” *SIGOPS Oper. Syst. Rev.* 44, 3 (Aug. 2010), pp. 2–19.
- [ONF 2020]** Open Networking Foundation, Specification, <https://www.opennetworking.org/software-defined-standards/specifications/>
- [ONOS 2020]** ONOS, <https://onosproject.org/collateral/>
- [OpenDaylight 2020]** OpenDaylight, <https://www.opendaylight.org/>
- [OpenDaylight 2020]** OpenDaylight, <https://www.opendaylight.org/what-we-do/current-release/sodium>
- [OpenSignal 2019]** Opensignal, <https://www.opensignal.com/>
- [Ordonez-Lucena 2017]** J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca and J. Folgueira, “Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges,” *IEEE Communications Magazine*, Vol. 55, No. 5, pp. 80–87, May 2017.
- [Osterweil 2012]** E. Osterweil, D. McPherson, S. DiBenedetto, C. Papadopoulos, D. Massey, “Behavior of DNS Top Talkers,” *Passive and Active Measurement Conference*, 2012.
- [P4 2020]** P4 Language Consortium, <https://p4.org/>
- [Padhye 2000]** J. Padhye, V. Firoiu, D. Towsley, J. Kurose, “Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation,” *IEEE/ACM Transactions on Networking*, Vol. 8, No. 2 (Apr. 2000), pp. 133–145.
- [Padhye 2001]** J. Padhye, S. Floyd, “On Inferring TCP Behavior,” *Proc. 2001 ACM SIGCOMM Conference* (San Diego, CA, Aug. 2001).
- [Palat 2009]** S. Palat, P. Godin, “The LTE Network Architecture: A Comprehensive Tutorial,” in *LTE—The UMTS Long Term Evolution: From Theory to Practice*. Also available as a standalone Alcatel white paper.
- [Panda 2013]** A. Panda, C. Scott, A. Ghodsi, T. Koponen, S. Shenker, “CAP for Networks,” *Proc. 2013 ACM HotSDN Conference*, pp. 91–96.
- [Parekh 1993]** A. Parekh, R. Gallagher, “A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case,” *IEEE/ACM Transactions on Networking*, Vol. 1, No. 3 (June 1993), pp. 344–357.
- [Partridge 1998]** C. Partridge, et al. “A Fifty Gigabit per second IP Router,” *IEEE/ACM Transactions on Networking*, Vol. 6, No. 3 (Jun. 1998), pp. 237–248.
- [Patel 2013]** P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu, C. Kim, N. Karri, “Ananta: Cloud Scale Load Balancing,” *Proc. 2013 ACM SIGCOMM Conference*.
- [Pathak 2010]** A. Pathak, Y. A. Wang, C. Huang, A. Greenberg, Y. C. Hu, J. Li, K. W. Ross, “Measuring and Evaluating TCP Splitting for Cloud Services,” *Passive and Active Measurement (PAM) Conference* (Zurich, 2010).
- [Peering DB 2020]** “The Interconnection Database,” <https://www.peeringdb.com/>
- [Peha 2006]** J. Peha, “The Benefits and Risks of Mandating Network Neutrality, and the Quest for a Balanced Policy,” *Proc. 2006 Telecommunication Policy Research Conference (TPRC)*, <https://ssrn.com/abstract=2103831>
- [Perkins 1994]** A. Perkins, “Networking with Bob Metcalfe,” *The Red Herring Magazine* (Nov. 1994).
- [Perkins 1998b]** C. Perkins, *Mobile IP: Design Principles and Practice*, Addison-Wesley, Reading, MA, 1998.
- [Perkins 2000]** C. Perkins, *Ad Hoc Networking*, Addison-Wesley, Reading, MA, 2000.
- [Perlman 1999]** R. Perlman, *Interconnections: Bridges, Routers, Switches, and Internetworking Protocols*, 2nd edition, Addison-Wesley Professional Computing Series, Reading, MA, 1999.
- [PGP 2020]** Symantec PGP, <https://www.symantec.com/products/encryption>, 2020

- [Phifer 2000]** L. Phifer, “The Trouble with NAT,” *The Internet Protocol Journal*, Vol. 3, No. 4 (Dec. 2000), [http://www.cisco.com/warp/public/759/ipj\\_3-4/ipj\\_3-4\\_nat.html](http://www.cisco.com/warp/public/759/ipj_3-4/ipj_3-4_nat.html)
- [Piatek 2008]** M. Piatek, T. Isdal, A. Krishnamurthy, T. Anderson, “One Hop Reputations for Peer-to-peer File Sharing Workloads,” *Proc. NSDI* (2008).
- [Pickholtz 1982]** R. Pickholtz, D. Schilling, L. Milstein, “Theory of Spread Spectrum Communication—a Tutorial,” *IEEE Transactions on Communications*, Vol. 30, No. 5 (May 1982), pp. 855–884.
- [PingPlotter 2020]** PingPlotter homepage, <http://www.pingplotter.com>
- [Pomeranz 2010]** H. Pomeranz, “Practical, Visual, Three-Dimensional Pedagogy for Internet Protocol Packet Header Control Fields,” <https://righteousit.wordpress.com/2010/06/27/practical-visual-three-dimensional-pedagogy-for-internet-protocol-packet-header-control-fields/>, June 2010.
- [Quagga 2012]** Quagga, “Quagga Routing Suite,” <http://www.quagga.net/>
- [Qualcomm 2019]** Qualcomm, “Everything you want to know about 5G,” <https://www.qualcomm.com/invention/5g/what-is-5g>
- [Qazi 2013]** Z. Qazi, C. Tu, L. Chiang, R. Miao, V. Sekar, M. Yu, “SIMPLE-fying Middlebox Policy Enforcement Using SDN,” *Proc. ACM SIGCOMM Conference* (Aug. 2013), pp. 27–38.
- [Quic 2020]** <https://quicwg.org/>
- [QUIC-recovery 2020]** J. Iyengar, Ed., I. Swett, Ed., “QUIC Loss Detection and Congestion Control,” Internet Draft *draft-ietf-quic-recovery-latest*, April 20, 2020.
- [Quittner 1998]** J. Quittner, M. Slatalla, *Speeding the Net: The Inside Story of Netscape and How It Challenged Microsoft*, Atlantic Monthly Press, 1998.
- [Quova 2020]** [www.quova.com](http://www.quova.com)
- [Raiciu 2010]** C. Raiciu, C. Pluntke, S. Barre, A. Greenhalgh, D. Wischik, M. Handley, “Data Center Networking with Multipath TCP,” *Proc. 2010 ACM SIGCOMM Conference*.
- [Ramakrishnan 1990]** K. K. Ramakrishnan, R. Jain, “A Binary Feedback Scheme for Congestion Avoidance in Computer Networks,” *ACM Transactions on Computer Systems*, Vol. 8, No. 2 (May 1990), pp. 158–181.
- [Raman 2007]** B. Raman, K. Chebrolu, “Experiences in Using WiFi for Rural Internet in India,” *IEEE Communications Magazine*, Special Issue on New Directions in Networking Technologies in Emerging Economies (Jan. 2007).
- [Ramjee 1994]** R. Ramjee, J. Kurose, D. Towsley, H. Schulzrinne, “Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks,” *Proc. 1994 IEEE INFOCOM*.
- [Rescorla 2001]** E. Rescorla, *SSL and TLS: Designing and Building Secure Systems*, Addison-Wesley, Boston, 2001.
- [RFC 001]** S. Crocker, “Host Software,” RFC 001 (the *very first* RFC!).
- [RFC 768]** J. Postel, “User Datagram Protocol,” RFC 768, Aug. 1980.
- [RFC 791]** J. Postel, “Internet Protocol: DARPA Internet Program Protocol Specification,” RFC 791, Sept. 1981.
- [RFC 792]** J. Postel, “Internet Control Message Protocol,” RFC 792, Sept. 1981.
- [RFC 793]** J. Postel, “Transmission Control Protocol,” RFC 793, Sept. 1981.
- [RFC 801]** J. Postel, “NCP/TCP Transition Plan,” RFC 801, Nov. 1981.
- [RFC 826]** D. C. Plummer, “An Ethernet Address Resolution Protocol—or—Converting Network Protocol Addresses to 48-bit Ethernet Address for Transmission on Ethernet Hardware,” RFC 826, Nov. 1982.
- [RFC 829]** V. Cerf, “Packet Satellite Technology Reference Sources,” RFC 829, Nov. 1982.
- [RFC 854]** J. Postel, J. Reynolds, “TELNET Protocol Specification,” RFC 854, May 1993.
- [RFC 950]** J. Mogul, J. Postel, “Internet Standard Subnetting Procedure,” RFC 950, Aug. 1985.

- [RFC 959] J. Postel and J. Reynolds, “File Transfer Protocol (FTP),” RFC 959, Oct. 1985.
- [RFC 1034] P. V. Mockapetris, “Domain Names—Concepts and Facilities,” RFC 1034, Nov. 1987.
- [RFC 1035] P. Mockapetris, “Domain Names—Implementation and Specification,” RFC 1035, Nov. 1987.
- [RFC 1071] R. Braden, D. Borman, and C. Partridge, “Computing the Internet Checksum,” RFC 1071, Sept. 1988.
- [RFC 1122] R. Braden, “Requirements for Internet Hosts—Communication Layers,” RFC 1122, Oct. 1989.
- [RFC 1191] J. Mogul, S. Deering, “Path MTU Discovery,” RFC 1191, Nov. 1990.
- [RFC 1320] R. Rivest, “The MD4 Message-Digest Algorithm,” RFC 1320, Apr. 1992.
- [RFC 1321] R. Rivest, “The MD5 Message-Digest Algorithm,” RFC 1321, Apr. 1992.
- [RFC 1422] S. Kent, “Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management,” RFC 1422.
- [RFC 1546] C. Partridge, T. Mendez, W. Milliken, “Host Anycasting Service,” RFC 1546, 1993.
- [RFC 1584] J. Moy, “Multicast Extensions to OSPF,” RFC 1584, Mar. 1994.
- [RFC 1633] R. Braden, D. Clark, S. Shenker, “Integrated Services in the Internet Architecture: an Overview,” RFC 1633, June 1994.
- [RFC 1752] S. Bradner, A. Mankin, “The Recommendations for the IP Next Generation Protocol,” RFC 1752, Jan. 1995.
- [RFC 1918] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear, “Address Allocation for Private Internets,” RFC 1918, Feb. 1996.
- [RFC 1930] J. Hawkinson, T. Bates, “Guidelines for Creation, Selection, and Registration of an Autonomous System (AS),” RFC 1930, Mar. 1996.
- [RFC 1945] T. Berners-Lee, R. Fielding, H. Frystyk, “Hypertext Transfer Protocol—HTTP/1.0,” RFC 1945, May 1996.
- [RFC 1958] B. Carpenter, “Architectural Principles of the Internet,” RFC 1958, June 1996.
- [RFC 2003] C. Perkins, “IP Encapsulation Within IP,” RFC 2003, Oct. 1996.
- [RFC 2004] C. Perkins, “Minimal Encapsulation Within IP,” RFC 2004, Oct. 1996.
- [RFC 2018] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, “TCP Selective Acknowledgment Options,” RFC 2018, Oct. 1996.
- [RFC 2104] H. Krawczyk, M. Bellare, R. Canetti, “HMAC: Keyed-Hashing for Message Authentication,” RFC 2104, Feb. 1997.
- [RFC 2131] R. Droms, “Dynamic Host Configuration Protocol,” RFC 2131, Mar. 1997.
- [RFC 2136] P. Vixie, S. Thomson, Y. Rekhter, J. Bound, “Dynamic Updates in the Domain Name System,” RFC 2136, Apr. 1997.
- [RFC 2328] J. Moy, “OSPF Version 2,” RFC 2328, Apr. 1998.
- [RFC 2420] H. Kummert, “The PPP Triple-DES Encryption Protocol (3DESE),” RFC 2420, Sept. 1998.
- [RFC 2460] S. Deering, R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification,” RFC 2460, Dec. 1998.
- [RFC 2578] K. McCloghrie, D. Perkins, J. Schoenwaelder, “Structure of Management Information Version 2 (SMIV2),” RFC 2578, Apr. 1999.
- [RFC 2579] K. McCloghrie, D. Perkins, J. Schoenwaelder, “Textual Conventions for SMIV2,” RFC 2579, Apr. 1999.
- [RFC 2580] K. McCloghrie, D. Perkins, J. Schoenwaelder, “Conformance Statements for SMIV2,” RFC 2580, Apr. 1999.
- [RFC 2581] M. Allman, V. Paxson, W. Stevens, “TCP Congestion Control,” RFC 2581, Apr. 1999.
- [RFC 2663] P. Srisuresh, M. Holdrege, “IP Network Address Translator (NAT) Terminology and Considerations,” RFC 2663.

- [RFC 2702] D. Awdanche, J. Malcolm, J. Agogbua, M. O'Dell, J. McManus, "Requirements for Traffic Engineering Over MPLS," RFC 2702, Sept. 1999.
- [RFC 2827] P. Ferguson, D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which Employ IP Source Address Spoofing," RFC 2827, May 2000.
- [RFC 2865] C. Rigney, S. Willens, A. Rubens, W. Simpson, "Remote Authentication Dial In User Service (RADIUS)," RFC 2865, June 2000.
- [RFC 2992] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," RFC 2992, Nov 2000.
- [RFC 3007] B. Wellington, "Secure Domain Name System (DNS) Dynamic Update," RFC 3007, Nov. 2000.
- [RFC 3022] P. Srisuresh, K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022, Jan. 2001.
- [RFC 3031] E. Rosen, A. Viswanathan, R. Callon, "Multiprotocol Label Switching Architecture," RFC 3031, Jan. 2001.
- [RFC 3032] E. Rosen, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, A. Conta, "MPLS Label Stack Encoding," RFC 3032, Jan. 2001.
- [RFC 3168] K. Ramakrishnan, S. Floyd, D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, Sept. 2001.
- [RFC 3209] D. Awdanche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels," RFC 3209, Dec. 2001.
- [RFC 3232] J. Reynolds, "Assigned Numbers: RFC 1700 Is Replaced by an On-line Database," RFC 3232, Jan. 2002.
- [RFC 3234] B. Carpenter, S. Brim, "Middleboxes: Taxonomy and Issues," RFC 3234, Feb. 2002.
- [RFC 3261] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, "SIP: Session Initiation Protocol," RFC 3261, July 2002.
- [RFC 3272] J. Boyle, V. Gill, A. Hannan, D. Cooper, D. Awdanche, B. Christian, W. S. Lai, "Overview and Principles of Internet Traffic Engineering," RFC 3272, May 2002.
- [RFC 3286] L. Ong, J. Yoakum, "An Introduction to the Stream Control Transmission Protocol (SCTP)," RFC 3286, May 2002.
- [RFC 3346] J. Boyle, V. Gill, A. Hannan, D. Cooper, D. Awdanche, B. Christian, W. S. Lai, "Applicability Statement for Traffic Engineering with MPLS," RFC 3346, Aug. 2002.
- [RFC 3390] M. Allman, S. Floyd, C. Partridge, "Increasing TCP's Initial Window," RFC 3390, Oct. 2002.
- [RFC 3410] J. Case, R. Mundy, D. Partain, "Introduction and Applicability Statements for Internet Standard Management Framework," RFC 3410, Dec. 2002.
- [RFC 3439] R. Bush, D. Meyer, "Some Internet Architectural Guidelines and Philosophy," RFC 3439, Dec. 2003.
- [RFC 3447] J. Jonsson, B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1," RFC 3447, Feb. 2003.
- [RFC 3468] L. Andersson, G. Swallow, "The Multiprotocol Label Switching (MPLS) Working Group Decision on MPLS Signaling Protocols," RFC 3468, Feb. 2003.
- [RFC 3469] V. Sharma, Ed., F. Hellstrand, Ed, "Framework for Multi-Protocol Label Switching (MPLS)-based Recovery," RFC 3469, Feb. 2003. <ftp://ftp.rfc-editor.org/in-notes/rfc3469.txt>
- [RFC 3535] J. Schönwälder, "Overview of the 2002 IAB Network Management Workshop," RFC 3535, May 2003.
- [RFC 3550] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 3550, July 2003.
- [RFC 3588] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, J. Arkko, "Diameter Base Protocol," RFC 3588, Sept. 2003.

- [RFC 3746] L. Yang, R. Dantu, T. Anderson, R. Gopal, “Forwarding and Control Element Separation (ForCES) Framework,” Internet, RFC 3746, Apr. 2004.
- [RFC 3748] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, H. Levkowetz, Ed., “Extensible Authentication Protocol (EAP),” RFC 3748, June 2004.
- [RFC 4022] R. Raghunarayanan, Ed., “Management Information Base for the Transmission Control Protocol (TCP),” RFC 4022, March 2005.
- [RFC 4033] R. Arends, R. Austein, M. Larson, D. Massey, S. Rose, “DNS Security Introduction and Requirements, RFC 4033, March 2005.
- [RFC 4113] B. Fenner, J. Flick, “Management Information Base for the User Datagram Protocol (UDP),” RFC 4113, June 2005.
- [RFC 4213] E. Nordmark, R. Gilligan, “Basic Transition Mechanisms for IPv6 Hosts and Routers,” RFC 4213, Oct. 2005.
- [RFC 4271] Y. Rekhter, T. Li, S. Hares, Ed., “A Border Gateway Protocol 4 (BGP-4),” RFC 4271, Jan. 2006.
- [RFC 4291] R. Hinden, S. Deering, “IP Version 6 Addressing Architecture,” RFC 4291, Feb. 2006.
- [RFC 4293] S. Routhier, Ed., “Management Information Base for the Internet Protocol (IP),” RFC 4293, April 2006.
- [RFC 4340] E. Kohler, M. Handley, S. Floyd, “Datagram Congestion Control Protocol (DCCP),” RFC 4340, Mar. 2006.
- [RFC 4346] T. Dierks, E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.1,” RFC 4346, Apr. 2006.
- [RFC 4514] K. Zeilenga, Ed., “Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names,” RFC 4514, June 2006.
- [RFC 4632] V. Fuller, T. Li, “Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan,” RFC 4632, Aug. 2006.
- [RFC 4960] R. Stewart, ed., “Stream Control Transmission Protocol,” RFC 4960, Sept. 2007.
- [RFC 4987] W. Eddy, “TCP SYN Flooding Attacks and Common Mitigations,” RFC 4987, Aug. 2007.
- [RFC 5128] P. Srisuresh, B. Ford, D. Kegel, “State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs),” March 2008, RFC 5128.
- [RFC 5246] T. Dierks, E. Rescorla, “The Transport Layer Security (TLS) Protocol, Version 1.2,” RFC 5246, Aug. 2008.
- [RFC 5277] S. Chisholm H. Trevino, “NETCONF Event Notifications,” RFC 5277, July 2008.
- [RFC 5321] J. Klensin, “Simple Mail Transfer Protocol,” RFC 5321, Oct. 2008.
- [RFC 5389] J. Rosenberg, R. Mahy, P. Matthews, D. Wing, “Session Traversal Utilities for NAT (STUN),” RFC 5389, Oct. 2008.
- [RFC 5681] M. Allman, V. Paxson, E. Blanton, “TCP Congestion Control,” RFC 5681, Sept. 2009.
- [RFC 5944] C. Perkins, Ed., “IP Mobility Support for IPv4, Revised,” RFC 5944, Nov. 2010.
- [RFC 6020] M. Bjorklund, “YANG—A Data Modeling Language for the Network Configuration Protocol (NETCONF),” RFC 6020, Oct. 2010.
- [RFC 6241] R. Enns, M. Bjorklund, J. Schönwälder, A. Bierman, “Network Configuration Protocol (NETCONF),” RFC 6241, June 2011.
- [RFC 6265] A Barth, “HTTP State Management Mechanism,” RFC 6265, Apr. 2011.
- [RFC 6298] V. Paxson, M. Allman, J. Chu, M. Sargent, “Computing TCP’s Retransmission Timer,” RFC 6298, June 2011.
- [RFC 6582] T. Henderson, S. Floyd, A. Gurkov, Y. Nishida, “The NewReno Modification to TCP’s Fast Recovery Algorithm,” RFC 6582, April 2012.

- [RFC 6733] V. Fajardo, J. Arkko, J. Loughney, G. Zorn, “Diameter Base Protocol,” RFC 6733, Oct. 2012.
- [RFC 7020] R. Housley, J. Curran, G. Huston, D. Conrad, “The Internet Numbers Registry System,” RFC 7020, Aug. 2013.
- [RFC 7094] D. McPherson, D. Oran, D. Thaler, E. Osterweil, “Architectural Considerations of IP Anycast,” RFC 7094, Jan. 2014.
- [RFC 7230] R. Fielding, Ed., J. Reschke, “Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing,” RFC 7230, June 2014.
- [RFC 7232] R. Fielding, Ed., J. Reschke, Ed., “Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests,” RFC 7232, June 2014.
- [RFC 7234] R. Fielding, Ed., M. Nottingham, Ed., J. Reschke, Ed., “Hypertext Transfer Protocol (HTTP/1.1): Caching,” RFC 7234, June 2014.
- [RFC 7323] D. Borman, S. Braden, V. Jacobson, R. Scheffenegger, “TCP Extensions for High Performance,” RFC 7323, Sept. 2014.
- [RFC 7540] M. Belshe, R. Peon, M. Thomson (Eds), “Hypertext Transfer Protocol Version 2 (HTTP/2),” RFC 7540, May 2015.
- [RFC 8033] R. Pan, P. Natarajan, F. Baker, G. White, “Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem,” RFC 8033, Feb. 2017.
- [RFC 8034] G. White, R. Pan, “Active Queue Management (AQM) Based on Proportional Integral Controller Enhanced (PIE) for Data-Over-Cable Service Interface Specifications (DOCSIS) Cable Modems,” RFC 8034, Feb. 2017.
- [RFC 8257] S. Bensley, D. Thaler, P. Balasubramanian, L. Eggert, G. Judd, “Data Center TCP (DCTCP): TCP Congestion Control for Data Centers,” RFC 8257, October 2017.
- [RFC 8312] L. Xu, S. Ha, A. Zimmermann, L. Eggert, R. Scheffenegger, “CUBIC for Fast Long-Distance Networks,” RFC 8312, Feb. 2018.
- [Richter 2015] P. Richter, M. Allman, R. Bush, V. Paxson, “A Primer on IPv4 Scarcity,” *ACM SIGCOMM Computer Communication Review*, Vol. 45, No. 2 (Apr. 2015), pp. 21–32.
- [Roberts 1967] L. Roberts, T. Merrill, “Toward a Cooperative Network of Time-Shared Computers,” *AFIPS Fall Conference* (Oct. 1966).
- [Rom 1990] R. Rom, M. Sidi, *Multiple Access Protocols: Performance and Analysis*, Springer-Verlag, New York, 1990.
- [Rommer 2019] S. Rommer, P. Hedman, M. Olsson, L. Frid, S. Sultana, C. Mulligan, *5G Core Networks: Powering Digitalization*, Academic Press, 2019.
- [Root Servers 2020] Root Servers home page, <http://www.root-servers.org/>
- [Roy 2015] A. Roy, H.i Zeng, J. Bagga, G. Porter, A. Snoeren, “Inside the Social Network’s (Datacenter) Network,” *Proc. 2015 ACM SIGCOMM Conference*, pp. 123–137.
- [RSA 1978] R. Rivest, A. Shamir, L. Adelman, “A Method for Obtaining Digital Signatures and Public-key Cryptosystems,” *Communications of the ACM*, Vol. 21, No. 2 (Feb. 1978), pp. 120–126.
- [Rubenstein 1998] D. Rubenstein, J. Kurose, D. Towsley, “Real-Time Reliable Multicast Using Proactive Forward Error Correction,” *Proceedings of NOSSDAV ’98* (Cambridge, UK, July 1998).
- [Ruiz-Sánchez 2001] M. Ruiz-Sánchez, E. Biersack, W. Dabbous, “Survey and Taxonomy of IP Address Lookup Algorithms,” *IEEE Network Magazine*, Vol. 15, No. 2 (Mar./Apr. 2001), pp. 8–23.
- [Saltzer 1984] J. Saltzer, D. Reed, D. Clark, “End-to-End Arguments in System Design,” *ACM Transactions on Computer Systems (TOCS)*, Vol. 2, No. 4 (Nov. 1984).
- [Saroiu 2002] S. Saroiu, P. K. Gummadi, S. D. Gribble, “A Measurement Study of Peer-to-Peer File Sharing Systems,” *Proc. of Multimedia Computing and Networking (MMCN)* (2002).

- [Sauter 2014]** M. Sauter, *From GSM to LTE-Advanced*, John Wiley and Sons, 2014.
- [Savage 2015]** D. Savage, J. Ng, S. Moore, D. Slice, P. Paluch, R. White, “Enhanced Interior Gateway Routing Protocol,” Internet Draft, draft-eigrp-04.txt, Aug. 2015.
- [Saydam 1996]** T. Saydam, T. Magedanz, “From Networks and Network Management into Service and Service Management,” *Journal of Networks and System Management*, Vol. 4, No. 4 (Dec. 1996), pp. 345–348.
- [Schiller 2003]** J. Schiller, *Mobile Communications*, 2nd edition, Addison Wesley, 2003.
- [Schneier 2015]** B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, Wiley, 2015.
- [Schönwälder 2010]** J. Schönwälder, M. Björklund, P. Shafer, “Network configuration management using NETCONF and YANG,” *IEEE Communications Magazine*, 2010, Vol. 48, No. 9, pp. 166–173.
- [Schwartz 1977]** M. Schwartz, *Computer-Communication Network Design and Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1997.
- [Schwartz 1980]** M. Schwartz, *Information, Transmission, Modulation, and Noise*, McGraw Hill, New York, NY 1980.
- [Schwartz 1982]** M. Schwartz, “Performance Analysis of the SNA Virtual Route Pacing Control,” *IEEE Transactions on Communications*, Vol. 30, No. 1 (Jan. 1982), pp. 172–184.
- [Scourias 2012]** J. Scourias, “Overview of the Global System for Mobile Communications: GSM.” <http://www.privateline.com/PCS/GSM0.html>
- [Segaller 1998]** S. Segaller, *Nerds 2.0.1, A Brief History of the Internet*, TV Books, New York, 1998.
- [Serpanos 2011]** D. Serpanos, T. Wolf, *Architecture of Network Systems*, Morgan Kaufmann Publishers, 2011.
- [Shacham 1990]** N. Shacham, P. McKenney, “Packet Recovery in High-Speed Networks Using Coding and Buffer Management,” *Proc. 1990 IEEE Infocom* (San Francisco, CA, Apr. 1990), pp. 124–131.
- [Shaikh 2001]** A. Shaikh, R. Tewari, M. Agrawal, “On the Effectiveness of DNS-based Server Selection,” *Proc. 2001 IEEE INFOCOM*.
- [Sherry 2012]** J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, V. Sekar, “Making middleboxes someone else’s problem: network processing as a cloud service,” *Proc. 2012 ACM SIGCOMM Conference*.
- [Singh 1999]** S. Singh, *The Code Book: The Evolution of Secrecy from Mary, Queen of Scots to Quantum Cryptography*, Doubleday Press, 1999.
- [Singh 2015]** A. Singh et al., “Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google’s Datacenter Network,” *Proc. 2015 ACM SIGCOMM Conference*, pp. 183–197.
- [Smith 2009]** J. Smith, “Fighting Physics: A Tough Battle,” *Communications of the ACM*, Vol. 52, No. 7 (July 2009), pp. 60–65.
- [Smithsonian 2017]** Smithsonian Magazine, “How Other Countries Deal with Net Neutrality,” <https://www.smithsonianmag.com/innovation/how-other-countries-deal-net-neutrality-180967558/>
- [Snort 2012]** Sourcefire Inc., Snort homepage, <http://www.snort.org/>
- [Solensky 1996]** F. Solensky, “IPv4 Address Lifetime Expectations,” in *IPng: Internet Protocol Next Generation* (S. Bradner, A. Mankin, ed.), Addison-Wesley, Reading, MA, 1996.
- [Speedtest 2020]** <https://www.speedtest.net/>
- [Spragins 1991]** J. D. Spragins, *Telecommunications Protocols and Design*, Addison-Wesley, Reading, MA, 1991.
- [Srikant 2012]** R. Srikant, *The mathematics of Internet congestion control*, Springer Science & Business Media, 2012.

- [Statista 2019]** “Mobile internet usage worldwide - Statistics & Facts,” <https://www.statista.com/topics/779/mobile-internet/>
- [Steinder 2002]** M. Steinder, A. Sethi, “Increasing Robustness of Fault Localization Through Analysis of Lost, Spurious, and Positive Symptoms,” *Proc. 2002 IEEE INFOCOM*.
- [Stevens 1990]** W. R. Stevens, *Unix Network Programming*, Prentice-Hall, Englewood Cliffs, NJ.
- [Stevens 1994]** W. R. Stevens, *TCP/IP Illustrated, Vol. 1: The Protocols*, Addison-Wesley, Reading, MA, 1994.
- [Stevens 1997]** W. R. Stevens, *Unix Network Programming, Volume 1: Networking APIs-Sockets and XTI*, 2nd edition, Prentice-Hall, Englewood Cliffs, NJ, 1997.
- [Stewart 1999]** J. Stewart, *BGP4: Interdomain Routing in the Internet*, Addison-Wesley, 1999.
- [Stone 1998]** J. Stone, M. Greenwald, C. Partridge, J. Hughes, “Performance of Checksums and CRC’s Over Real Data,” *IEEE/ACM Transactions on Networking*, Vol. 6, No. 5 (Oct. 1998), pp. 529–543.
- [Stone 2000]** J. Stone, C. Partridge, “When Reality and the Checksum Disagree,” *Proc. 2000 ACM SIGCOMM Conference* (Stockholm, Sweden, Aug. 2000).
- [Strayer 1992]** W. T. Strayer, B. Dempsey, A. Weaver, *XTP: The Xpress Transfer Protocol*, Addison-Wesley, Reading, MA, 1992.
- [Stubblefield 2002]** A. Stubblefield, J. Ioannidis, A. Rubin, “Using the Fluhrer, Mantin, and Shamir Attack to Break WEP,” *Proceedings of 2002 Network and Distributed Systems Security Symposium* (2002), pp. 17–22.
- [Subramanian 2000]** M. Subramanian, *Network Management: Principles and Practice*, Addison-Wesley, Reading, MA, 2000.
- [Subramanian 2002]** L. Subramanian, S. Agarwal, J. Rexford, R. Katz, “Characterizing the Internet Hierarchy from Multiple Vantage Points,” *Proc. 2002 IEEE INFOCOM*.
- [Sundaresan 2006]** K. Sundaresan, K. Papagiannaki, “The Need for Cross-layer Information in Access Point Selection,” *Proc. 2006 ACM Internet Measurement Conference* (Rio De Janeiro, Oct. 2006).
- [Sunshine 1978]** C. Sunshine, Y. Dalal, “Connection Management in Transport Protocols,” *Computer Networks*, North-Holland, Amsterdam, 1978.
- [Tan 2006]** K. Tan, J. Song, Q. Zhang and M. Sridharan, “A Compound TCP Approach for High-Speed and Long Distance Networks,” *Proc. 2006 IEEE INFOCOM*.
- [Tariq 2008]** M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, M. Ammar, “Answering What-If Deployment and Configuration Questions with WISE,” *Proc. 2008 ACM SIGCOMM Conference* (Aug. 2008).
- [Teixeira 2006]** R. Teixeira, J. Rexford, “Managing Routing Disruptions in Internet Service Provider Networks,” *IEEE Communications Magazine* Vol. 44, No. 3 (Mar. 2006) pp. 160–165.
- [Think 2012]** Technical History of Network Protocols, “Cyclades,” <http://www.cs.utexas.edu/users/chris/think/Cyclades/index.shtml>
- [Tian 2012]** Y. Tian, R. Dey, Y. Liu, K. W. Ross, “China’s Internet: Topology Mapping and Geolocating,” *IEEE INFOCOM Mini-Conference 2012* (Orlando, FL, 2012).
- [TLD list 2020]** TLD list maintained by Wikipedia, [https://en.wikipedia.org/wiki/List\\_of\\_Internet\\_top-level\\_domains](https://en.wikipedia.org/wiki/List_of_Internet_top-level_domains)
- [Tobagi 1990]** F. Tobagi, “Fast Packet Switch Architectures for Broadband Integrated Networks,” *Proc. IEEE*, Vol. 78, No. 1 (Jan. 1990), pp. 133–167.
- [TOR 2020]** Tor: Anonymity Online, <http://www.torproject.org>
- [Torres 2011]** R. Torres, A. Finamore, J. R. Kim, M. M. Munafó, S. Rao, “Dissecting Video Server Selection Strategies in the YouTube CDN,” *Proc. 2011 Int. Conf. on Distributed Computing Systems*.

- [Tourrilhes 2014]** J. Tourrilhes, P. Sharma, S. Banerjee, J. Petit, "SDN and Openflow Evolution: A Standards Perspective," *IEEE Computer Magazine*, Nov. 2014, Vol. 47, No. 11, pp. 22–29.
- [Turner 1988]** J. S. Turner, "Design of a Broadcast packet switching network," *IEEE Transactions on Communications*, Vol. 36, No. 6 (June 1988), pp. 734–743.
- [Turner 2012]** B. Turner, "2G, 3G, 4G Wireless Tutorial," <http://blogs.nmscommunications.com/communications/2008/10/2g-3g-4g-wireless-tutorial.html>
- [van der Berg 2008]** R. van der Berg, "How the 'Net Works: An Introduction to Peering and Transit," <http://arstechnica.com/guides/other/peering-and-transit.ars>
- [van der Merwe 1998]** J. van der Merwe, S. Rooney, I. Leslie, S. Crosby, "The Tempest: A Practical Framework for Network Programmability," *IEEE Network*, Vol. 12, No. 3 (May 1998), pp. 20–28.
- [Vanhoeft 2017]** M. Vanhoef, F. Piessens, "Key Reinstallation Attacks: ForcingNonce Reuse in WPA2," *2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*, pp. 1313–1328.
- [Varghese 1997]** G. Varghese, A. Lauck, "Hashed and Hierarchical Timing Wheels: Efficient Data Structures for Implementing a Timer Facility," *IEEE/ACM Transactions on Networking*, Vol. 5, No. 6 (Dec. 1997), pp. 824–834.
- [Vasudevan 2005]** S. Vasudevan, C. Diot, J. Kurose, D. Towsley, "Facilitating Access Point Selection in IEEE 802.11 Wireless Networks," *Proc. 2005 ACM Internet Measurement Conference*, (San Francisco CA, Oct. 2005).
- [Venkataramani 2014]** A. Venkataramani, J. Kurose, D. Raychaudhuri, K. Nagaraja, M. Mao, S. Banerjee, "MobilityFirst: A Mobility-Centric and Trustworthy Internet Architecture," *ACM Computer Communication Review*, July 2014.
- [Villamizar 1994]** C. Villamizar, C. Song, "High Performance TCP in ANSNET," *ACM SIGCOMM Computer Communications Review*, Vol. 24, No. 5 (1994), pp. 45–60.
- [Viterbi 1995]** A. Viterbi, *CDMA: Principles of Spread Spectrum Communication*, Addison-Wesley, Reading, MA, 1995.
- [Vixie 2009]** P. Vixie, "What DNS Is Not," *Communications of the ACM*, Vol. 52, No. 12 (Dec. 2009), pp. 43–47.
- [Wakeman 1992]** I. Wakeman, J. Crowcroft, Z. Wang, D. Sirovica, "Is Layering Harmful (remote procedure call)," *IEEE Network*, Vol. 6, No. 1 (Jan. 1992), pp. 20–24.
- [Waldrop 2007]** M. Waldrop, "Data Center in a Box," *Scientific American* (July 2007).
- [Walfish 2004]** M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, S. Shenker, "Middleboxes No Longer Considered Harmful," *USENIX OSDI 2004* San Francisco, CA, December 2004.
- [Wang 2011]** Z. Wang, Z. Qian, Q. Xu, Z. Mao, M. Zhang, "An untold story of middleboxes in cellular networks," *Proc. 2011 ACM SIGCOMM Conference*.
- [Wei 2006]** D. X. Wei, C. Jin, S. H. Low and S. Hegde, "FAST TCP: Motivation, Architecture, Algorithms, Performance," *IEEE/ACM Transactions on Networking*, Vol. 14, No. 6, pp. 1246–1259, Dec. 2006.
- [Wei 2006]** W. Wei, C. Zhang, H. Zang, J. Kurose, D. Towsley, "Inference and Evaluation of Split-Connection Approaches in Cellular Data Networks," *Proc. Active and Passive Measurement Workshop* (Adelaide, Australia, Mar. 2006).
- [Weiser 1991]** M. Weiser, "The Computer for the Twenty-First Century," *Scientific American* (Sept. 1991): 94–10. <http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html>
- [Wifi 2019]** The WiFi Alliance, "WPA3™ Security Considerations Overview," April 2019.
- [WiFi 2020]** The WiFi Alliance, <https://www.wi-fi.org/>

- [Williams 1993]** R. Williams, “A Painless Guide to CRC Error Detection Algorithms,” <http://www.ross.net/crc/crcpaper.html>
- [Wireshark 2020]** Wireshark homepage, <http://www.wireshark.org>
- [Wischik 2005]** D. Wischik, N. McKeown, “Part I: Buffer Sizes for Core Routers,” *ACM SIGCOMM Computer Communications Review*, Vol. 35, No. 3 (July 2005).
- [Woo 1994]** T. Woo, R. Bindignavle, S. Su, S. Lam, “SNP: an interface for secure network programming,” *Proc. 1994 Summer USENIX* (Boston, MA, June 1994), pp. 45–58.
- [Wright 2015]** J. Wright, J., *Hacking Exposed Wireless*, McGraw-Hill Education, 2015.
- [Wu 2005]** J. Wu, Z. M. Mao, J. Rexford, J. Wang, “Finding a Needle in a Haystack: Pinpointing Significant BGP Routing Changes in an IP Network,” *Proc. USENIX NSDI* (2005).
- [W3Techs]** World Wide Web Technology Surveys, 2020. <https://w3techs.com/technologies/details/ce-http2/all/all>.
- [Xanadu 2012]** Xanadu Project homepage, <http://www.xanadu.com/>
- [Xiao 2000]** X. Xiao, A. Hannan, B. Bailey, L. Ni, “Traffic Engineering with MPLS in the Internet,” *IEEE Network* (Mar./Apr. 2000).
- [Xu 2004]** L. Xu, K. Harfoush, I. Rhee, “Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks,” *IEEE INFOCOM 2004*, pp. 2514–2524.
- [Yang 2014]** P. Yang, J. Shao, W. Luo, L. Xu, J. Deogun, Y. Lu, “TCP congestion avoidance algorithm identification,” *IEEE/ACM Trans. Netw.* Vol. 22, No. 4 (Aug. 2014), pp. 1311–1324.
- [Yavatkar 1994]** R. Yavatkar, N. Bhagwat, “Improving End-to-End Performance of TCP over Mobile Internetworks,” *Proc. Mobile 94 Workshop on Mobile Computing Systems and Applications* (Dec. 1994).
- [YouTube 2009]** YouTube 2009, Google container data center tour, 2009.
- [Yu 2004]** Yu, Fang, H. Katz, Tirunellai V. Lakshman. “Gigabit Rate Packet Pattern-Matching Using TCAM,” *Proc. 2004 Int. Conf. Network Protocols*, pp. 174–183.
- [Yu 2011]** M. Yu, J. Rexford, X. Sun, S. Rao, N. Feamster, “A Survey of VLAN Usage in Campus Networks,” *IEEE Communications Magazine*, July 2011.
- [Zegura 1997]** E. Zegura, K. Calvert, M. Donahoo, “A Quantitative Comparison of Graph-based Models for Internet Topology,” *IEEE/ACM Transactions on Networking*, Vol. 5, No. 6, (Dec. 1997). See also <http://www.cc.gatech.edu/projects/gtitm> for a software package that generates networks with a transit-stub structure.
- [Zhang 2007]** L. Zhang, “A Retrospective View of NAT,” *The IETF Journal*, Vol. 3, No. 2 (Oct. 2007).
- [Zheng 2008]** N. Zheng and J. Wigard, “On the Performance of Integrator Handover Algorithm in LTE Networks,” *2008 IEEE 68th Vehicular Technology Conference*, Calgary, BC, 2008, pp. 1–5.
- [Zhu 2015]** Y. Zhu, H. Eran, D. Firestone, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel Mohamad, H. Yahia, M. Zhang, J. Padhye, “Congestion Control for Large-Scale RDMA Deployments,” *Proc. 2015 ACM SIGCOMM Conference*.
- [Zilberman 2019]** N. Zilberman, G. Bracha, G. Schzukin. “Stardust: Divide and conquer in the data center network,” *2019 USENIX Symposium on Networked Systems Design and Implementation*.
- [Zink 2009]** M. Zink, K. Suh, Y. Gu, J. Kurose, “Characteristics of YouTube Network Traffic at a Campus Network—Measurements, Models, and Implications,” *Computer Networks*, Vol. 53, No. 4, pp. 501–514, 2009.
- [Zou 2016]** Y. Zou, J. Zhu, X. Wang, L. Hanzo, “A Survey on Wireless Security: Technical Challenges, Recent Advances, and Future Trends,” *Proceedings of the IEEE*, Vol. 104, No. 9, 2016.

# Indice analitico

## #

3G e LTE 4G e 5G, 16

5G Fixed Wireless, 15

## A

Access network, 10

Accesso a Internet via cavo, 12

Accesso al collegamento, 367

Accesso multiplo a divisione di codice, 379

Accodamento in ingresso, 266

Accodamento in uscita, 267

ACK duplicati, 203

Acknowledgment, 84

Acknowledgment cumulativo, 180, 193

Acknowledgment selettivo, 206

Action, 298

Active optical networks, 14

Active Queue Management (AQM), 268

Adattatore di rete, 368

Address Resolution Protocol, 390

Address Supporting Organization, 284

Agenti utente, 99

Algoritmi di instradamento, 253, 312

Algoritmi di routing, 253

Algoritmi insensibili al carico, 314

Algoritmi link-state, 313

Algoritmo di Bellman-Ford, 319, 320

Algoritmo di controllo di congestione di TCP, 224

Algoritmo di Dijkstra, 314

Algoritmo di instradamento centralizzato, 313

Algoritmo di instradamento decentralizzato, 313

Algoritmo di selezione delle rotte, 333

Algoritmo sensibile al carico, 314

Alias, 106

ALOHA, 381

Ampiezza della finestra, 179

Ampiezza di banda, 25

Analisi del traffico, 51

--- packet sniffer, 52

API (Application Programming Interface), 72

Applicazioni distribuite, 5

Applicazioni elastiche, 75

Applicazioni sensibili alla banda, 75

Architettura client-server, 69

--- client, 69

--- server, 69

Architettura dell'applicazione, 69

Architettura P2P, 71

ARP, 393

AS-PATH, 331

Attesa binaria esponenziale, 385, 386

Auto-temporizzato, TCP, 223

**B**

- Bandwidth, 25
- Best-effort, 151
- Best-Effort Delivery Service, 151
- BGP, Border Gateway Protocol, 329
- Bilanciamento del carico, 415
- Binary Exponential Backoff, 385
- Bit di parità, 371
- Bit rate, 124
- BitTorrent, 121
  - chunk, 121
  - neighboring peer, 122
  - optimistically unchoked, 123
  - rarest first, 122
  - tabelle hash distribuite (DHT), 123
  - Torrent, 121
  - tracker, 121
  - unchoked, 123
- Blocco in testa alla coda, 266
- Border router, 415
- Bottleneck link, 41
- Browser web, 81
- Buffer di invio, 189
- Bufferbloat, 270

**C**

- Cable head end, 13
- Cable modem, 13
- Cable Modem Termination System, 13, 388
- Campo del numero di porta di destinazione, 154
- Campo del numero di porta di origine, 154
- Canali radio satellitari, 19
  - commutatori di pacchetto, 20
  - messaggi, 19
  - pacchetti, 20
- Canali radio terrestri, 18
- Carico offerto alla rete, 217
- Casella di posta, 99
- Cavo coassiale, 18
- CDN (Content Distribution Networks), 126

- CDN di terze parti, 126
- CDN private, 126
- DNS server locale (LDNS), 128
- strategia di selezione del cluster, 128
- Centrale locale, 11
- Channel partitioning protocol, 377
- Checksum, 368
  - tecniche di, 371
- Checksum di Internet, 373
- Chokepacket, 220
- Classful addressing, 281, 282
- Classless Interdomain Routing (CIDR), 281
- Client, 9
- Cliente, 29
- Coda di messaggi, 99
- Code con priorità, 271
  - accodamento a priorità senza prelazione, 272
  - non-preemptive priority queuing, 272
  - priority queuing, 271
- Code Division Multiple Access, 379
- Codici di controllo a ridondanza ciclica, 373
- Codici polinomiali, 373
- Collegamenti (link), 366
- Collegamento broadcast, 375
- Collegamento punto a punto, 375
- Collisione, 377
- Collo di bottiglia, 41
- Communication link, 2
- Commutatore di pacchetto, 2, 257
- Commutatori a livello di collegamento, 4, 257
- Commutazione attraverso rete di interconnessione, 264
- Commutazione in memoria, 263
- Commutazione tramite bus, 264
- Comunicazione logica, 148
- Congestion avoidance, 225
- Connessioni non persistenti, 82
- Connessioni persistenti, 82
- Consegna affidabile, 368
- Contatore, 173
- Content Distribution Network (CDN), 95
- Content provider, 31
- Control Agent, 310

Controllo a ridondanza ciclica, 371  
 Controllo di congestione, 152, 206  
   --- assistito dalla rete, 220  
   --- ATM ABR, 220  
   --- basato sul ritardo, 233  
   --- end-to-end, 220  
 Controllo di parità, 371  
 Controllo logicamente centralizzato (CA), 310  
 Controllo per router, 310  
 Cookie, 89  
 Correzione degli errori in avanti, 372  
 Countdown timer, 173  
 Count-to-infinity problem, 324  
 CSMA, 382  
   --- protocolli CSMA, 383  
   --- protocolli CSMA/CD, 383  
   --- rilevamento della collisione, 383  
   --- rilevamento della portante, 382  
   --- ritardo di propagazione, 383  
 CSMA/CD, 383  
   --- efficienza, 387  
 Customer, 29  
 Cyclic redundancy check, 371, 373

**D**

DASH, 125  
   --- manifest file, 125  
 Data center, 10, 70  
   --- controllo e gestione SDN centralizzati, 419  
   --- modularità e personalizzazione hardware, 420  
   --- riduzione dei costi, 418  
   --- vincoli fisici, 420  
   --- virtualizzazione, 419  
 Data Center Network, 414  
 Data Center Network Design, 415  
 Data-over-cable service interface specification, 388  
 Demultiplexing, 153  
 Demultiplexing a livello di trasporto, 151  
 Digital Subscriber Line, 11

Digital Subscriber Line Access Multiplex, 11  
 Dimensione massima di segmento, 190  
 Distance-vector, 318  
 Distributed applications, 5  
 Distribution time, 119  
 Distribuzione del carico di rete (load distribution), 107  
 Dynamic Host Configuration Protocol (DHCP), 284  
   --- DHCP ACK, 287  
   --- gateway, 284  
   --- indirizzo IP temporaneo, 284  
   --- individuazione del server DHCP, 285  
   --- offerta del server DHCP, 286  
   --- plug-and-play, 284  
   --- richiesta DHCP, 287  
   --- router di default, 284  
   --- zero-conf, 284  
 DNS Caching, 112  
 DNS server autoritativi, 110  
 DNS server locale, 110  
 Domain name system (DNS), 105  
 Doppino di rame intrecciato, 17  
   --- doppino intrecciato non schermato, 17  
 Drop-tail, eliminazione in coda, 267  
 DSL, 11  
 Duplicate data packets, 173  
 Dynamic Adaptive Streaming Over HTTP (DASH), 125

**E**

Edge router, 10  
 End system, 2, 9  
 Error-detection and -correction, 370  
 Errori non rilevati, 371  
 Ethernet, tecnologie, 400  
   --- autoapprendimento, 403  
   --- base station, 15  
   --- dispositivi plug-and-play, 404  
   --- e WiFi, 15  
   --- filtraggio, 402

--- inoltro, 402  
--- local area network, 15  
--- repeater, 400  
--- rete locale, 15  
--- tabella di commutazione, 402  
Etichetta VLAN, 409

## F

Fast recovery, 227  
FDM, 25  
Fiber To The Home (FTTH), 14  
Fibra ottica, 18  
File HTML principale, 81  
Finestra di congestione, 222  
Finestra di ricezione, 206  
First-Come-First-Served, 270  
First-In-First-Out, 270, 271  
Flooding, 326  
Formula di Bellman-Ford, 318  
Fornitore, 29  
Forward error correction, 372  
Forwarding table, 254  
Frame del livello di collegamento, 366  
Frame Ethernet, struttura, 397  
Framing, 367  
Framing HTTP/2, 97  
Frequency Division Multiplexing, 25, 377

## G

Generatore, 373  
Gerarchia di router e switch, 416  
Gestione attiva della coda, 268  
GET condizionale, 95  
Grafo, 312

## H

Handshake a tre vie, 84, 189, 209  
Head of Line (HOL) blocking, 96, 266  
High-level data link control, 375  
Host, 2, 9  
Host aliasing, 106  
Hostname, 105  
Hostname canonico, 106  
HTTP con connessioni non persistenti, 82  
HTTP con connessioni persistenti, 84  
HTTP/3, 98  
Hub, 397  
HyperText Transfer Protocol (HTTP), 79, 80

## I

ICMP (Internet Control Message Protocol), 348  
IETF, 5  
Incremento additivo, decremento  
    moltiplicativo, 229  
Indirizzamento, 73  
    --- indirizzi IP, 73  
    --- numero di porta di destinazione, 73  
Indirizzi IP, 105  
Indirizzi MAC, 391  
    --- indirizzo fisico, 391  
    --- indirizzo LAN, 391  
    --- indirizzo MAC, 391  
    --- indirizzo MAC broadcast, 392  
Infrastruttura di gestione di rete, 350  
    --- agente di gestione, 351  
    --- CLI, 352  
    --- dati di configurazione, 351  
    --- dati operativi, 351  
    --- dispositivo di rete gestito, 350  
    --- NETCONF/YANG, 352  
    --- oggetti MIB, 352  
    --- protocollo di gestione di rete, 351  
    --- server di gestione, 350  
    --- SNMP/MIB, 352  
    --- statistiche del dispositivo, 351

- Ingegneria del traffico, 413  
 Instradamento “a patata bollente”, 332  
 Instradamento ciclico, 323  
 Instradamento distance-vector, 322, 324  
 Instradamento hot potato, 332  
 Intensità di traffico, 36  
 Inter-Autonomous System Routing Protocol, 329  
 Interfaccia socket, 5  
 Interleaving, 98  
 Internet Assigned Numbers Authority (IANA), 109  
 Internet Corporation for Assigned Names and Numbers, 116, 284  
 Internet Engineering Task Force, 5  
 Internet Exchange Point, 30  
 Internet Mail Access Protocol (IMAP), 104  
 Internet Protocol (IP), 4, 275  
 Internet Service Provider, 4  
 Internetting, 55  
 Interprocess communication, 71  
 Intervalli di tempo, 377  
 Intervallo degli indirizzi di destinazione, 261  
 Intra-AS routing protocol, 326  
 Inversione avvelenata, 324  
 IPv4, datagrammi, 275
  - checksum dell'intestazione, 277
  - dati, 278
  - identificatore, flag, offset di frammentazione, 276
  - indirizzi IP sorgente e destinazione, 277
  - lunghezza del datagramma, 276
  - lunghezza dell'intestazione, 276
  - numero di versione, 275
  - opzioni, 277
  - protocollo, 277
  - tempo di vita, 277
  - tipo di servizio, 276
 IPv4, indirizzamento, 278
  - interfaccia, 278
  - maschera di sottorete, 279
  - notazione decimale puntata, 278
  - sottorete, 279
 IPv6, 289  
 IPv6, datagrammi, 291
  - etichettatura dei flussi, 291
  - flusso, 291
  - indirizzamento esteso, 291
  - intestazione ottimizzata di 40 byte, 291
 ISP, 4
  - di accesso multi-homed, 335
  - di primo livello, 29
  - regionale, 29
 IXP, 30
- L**
- Label-Switched Router, 412  
 Least-Cost Path, 313  
 Line card, 368  
 Link, 366  
 Link Layer Controller, 369  
 Link-Layer Frame, 366  
 Link-Layer Switch, 4, 257  
 Link-State Broadcast, 314  
 Livello di applicazione, 46  
 Livello di collegamento, 47
  - frame, 47
 Livello fisico, 48
  - datagramma a livello di rete, 49
  - frame a livello di collegamento, 49
  - incapsulamento, 49
  - malware, 50
  - messaggio a livello di applicazione, 49
  - payload, 49
  - segmento a livello di trasporto, 49
 Livello di rete, 47
  - botnet, 50
  - datagrammi, 47
  - denial-of-service (DoS), 51
  - malware auto-replicante, 50
  - negazione del servizio, 51
 Livello di trasporto, 46  
 Load balancer, 415  
 Logical communication, 148

**M**

- Macchina a stati finiti, 166
- Mail server, 99
- Mail server aliasing, 106
- Mailbox, 99
- Mascheramento, 53
  - end-point authentication, 53
- IP spoofing, 53
- Match, 296
- Media mobile esponenziale ponderata, 197
- Medium Access Control, 367
- Mesh, 29
- Messaggi DNS, 114
- Messaggio trap, 353
- MIB (Management Information Base), 355
- Middlebox, dispositivi, 300
- Modello di servizio del livello di rete, 256
  - banda minima garantita, 256
  - consegna garantita, 256
  - consegna garantita con ritardo limitato, 256
  - consegna ordinata, 256
  - servizi di sicurezza, 256
  - servizio best-effort, 256
- Multi-homing o multi-home, 30
- Multiplexing, 151, 153
  - a divisione di frequenza, 25
  - a divisione di tempo, 25
  - e demultiplexing non orientati alla connessione, 154
  - e demultiplexing orientati alla connessione, 156
- Multi-Protocol Label Switching, 411

**N**

- NAT (Network Address Translation), 287
  - attraversamento del, 289
  - reame con indirizzi privati, 287
  - reti private, 287
  - tabella di traduzione NAT, 288
- NCP (Network-Control Protocol), 54
- NETCONF, 352, 356

**N**

- Network adapter, 368
- Network edge, 8
- Network Interface Controller, 369
- Network of networks, 28
- NEXT-HOP, 332
- Nodo, 366, 377
- Notifica esplicita di congestione, 232
- Notifiche negative, 167
- Notifiche positive, 167
- Numeri di porta di origine e di destinazione, 191
- Numeri di porta noti, 154
- Numeri di sequenza e numeri di acknowledgment, 193
- Numero di porta della socket, 131
- Numero di sequenza, 170
- Numero di sequenza per un segmento, 193

**O**

- On demand, 124
- Optical Line Terminator, 14
- Optical Network Terminator, 14
- Orientato alla connessione, TCP, 188
- OSPF (Open Shortest Path First), 325
  - autonomia amministrativa, 326
  - scalabilità, 326
- Ospite, 2

**P**

- Pacchetti di dati duplicati, 173
- Pacchetti duplicati, 170
- Pacchetto ARP, 394
- Pacchetto, 3
- Packet, 3
- Packet scheduler, 268
- Packet switch, 2, 257
- Packet-dropping policy, 271

- Pagina web, 81  
 Parità bidimensionale, 372  
 Parity check, 371  
 Passive Optical Networks, 14  
 Path, 4  
 Peer, 71  
 Peering, 30, 336  
 Peer-to-peer, 71  
 Percorso (path), 4  
 Percorso a costo minimo, 313  
 Percorso in un grafo, 312  
 Percorso più breve, 313  
 Perdita di pacchetti, 37, 265
  - buffer overflow, 37
  - end-to-end delay, 37
 Periodi di silenzio, 26  
 Piggybacked acknowledgement, 195  
 Pipelining, 85, 178  
 Point of presence, 30  
 Point-To-Point, 189  
 Point-To-Point Protocol, 375  
 Poisoned reverse, 324  
 Politiche di instradamento, 335  
 PoP, 30  
 Prefisso di rete, 281  
 Princípio end-to-end, 164  
 Problema dell'accesso multiplo, 376  
 Problema di conteggio all'infinito, 324  
 Processi client e server
  - client, 71, 72
  - server, 71, 72
 Processi comunicanti, 71  
 Processo, 71  
 Programma nslookup, 115  
 Programmazione basata su eventi, 182  
 Protocol, 6  
 Protocolli a ripetizione selettiva, 183  
 Protocolli a rotazione, 377, 387  
 Protocolli a suddivisione del canale, 377  
 Protocolli ad accesso casuale, 377  
 Protocolli ARQ, 167  
 Protocolli di accesso multiplo, 376  
 Protocolli di rete, 8  
 Protocolli stop-and-wait, 168  
 Protocollo, 6, 8
  - a finestra scorrevole, 179
  - a livello di applicazione, 79
  - ad alternanza di bit, 174
  - di instradamento inter-AS, 329
  - di instradamento interno al sistema autonomo, 326
  - di risoluzione degli indirizzi, 393
  - di trasferimento dati affidabile, 164
  - Go-Back-N (GBN), 178
  - OpenFlow, 342
  - polling, 387
  - senza memoria di stato, 82
  - SMTP, 99
  - token-passing, 388
 Provider, 29  
 Proxy, 88  
 Proxy server, 91  
 Punto a punto, 189
- Q**  
 Query, 108  
 Query iterativa, 111  
 Query ricorsive, 111  
 QUIC, 237  
 Quick UDP Internet Connection, 162
- R**  
 Random Access Protocol, 377  
 Random delay, 379  
 Random Early Detection (RED), 268  
 rdt1.0, 166  
 rdt2.0, 167  
 rdt3.0, 173  
 Record di risorsa, 113

Registrar, 115  
Regola di corrispondenza a prefisso più lungo, 262  
Request For Comment (RFC), 5  
Rete di accesso di un ISP, 335  
Rete di collegamenti, 2  
Rete di reti, 28, 55  
Reti di accesso, 10  
Reti private virtuali, 413  
Reti stub, 335  
RFC, 5  
Rilevamento e correzione degli errori sui bit, 370  
Rilevazione e correzione degli errori, 368  
Ritardo di accodamento, 32, 33  
Ritardo di elaborazione, 32, 33  
Ritardo di pacchettizzazione, 40  
Ritardo di propagazione, 32, 33  
Ritardo di trasmissione, 32, 33  
Ritardo totale di nodo, 32  
Root server, 108, 109  
Rotta (route), 331  
Round Robin e Weighted Fair Queuing (WFQ), 274  
--- accodamento equo ponderato, 274  
--- modalità conservativa, 274  
--- Weighted Fair Queuing (WFQ), 274  
Round-Trip Time (RTT), 83  
Route, 4  
Router, 4, 257, 258  
--- piano di controllo, 259  
--- porte di ingresso, 258  
--- porte di uscita, 259  
--- processore di instradamento, 259  
--- struttura di commutazione, 259  
Router a commutazione di etichetta, 412  
Router di confine, 415  
Routing algorithm, 312  
RTT, Round-Trip Time, 196, 197, 221, 222

**S**  
Scalabilità, 71  
Scalabilità dell'architettura P2P, 118  
Scheda di rete, 368  
Schedulatore di pacchetti, 268  
SDN, architettura, 338  
--- funzioni di controllo di rete: esterne agli switch del piano dei dati, 339  
--- inoltro basato sui flussi, 338  
--- separazione del piano dei dati e del piano di controllo, 339  
--- una rete programmabile, 339  
Segmenti, 148  
Segmenti TCP, 190  
Sequence number, 170  
Server, 9  
Server autoritativi, 108  
Server di posta, 99  
Servizio di controllo di flusso, 206  
Servizio di trasporto dati affidabile, 199  
Servizio full-duplex, 189  
Servizio non affidabile, 151  
Sessione BGP, 330  
--- BGP esterna, 330  
--- BGP interna, 330  
Sessione iBGP, 330  
Shortest path, 313  
Simple Mail Transfer Protocol, 99  
Sistemi autonomi, 326  
--- protocollo di instradamento interno al sistema autonomo, 326  
Sistemi periferici, 2, 9  
Slot riuscito, 380  
Slot temporali, 378  
Slotted ALOHA, 379  
Slow start, fase iniziale, 224  
SNMPv3, 353  
Socket, 72, 152  
Socket interface, 5  
Software-Defined Networking (SDN), 256  
Standard di Internet, 5  
Stati TCP, 210  
Streaming dinamico adattativo su HTTP, 125

- Stratificazione dei protocolli, 45
  - layer, 45
  - livelli, 45
  - pila di protocolli, 46
  - protocol stack, 46
  - strati, 45
- Struttura dei segmenti TCP, 191
- Switch, 397
  - Switch vs router, 405
- Switched Ethernet, 397
- Switched local area network, 389
- Switching, 263
- Switching fabric, 263
  
- T**
- Tabella ARP, 394
- Tabella dei flussi, 296
- Tabella di forwarding, 254
- Tabella di inoltro, 23, 254
- Tabelle di instradamento, 322
- Tag Control Information, 410
- Tag Protocol Identifier, 410
- Tag VLAN, 409
- Taking-Turn Protocol, 377
- TCP (Transmission Control Protocol), 4, 15
- TCP CUBIC, 229
- TCP Reno, 227
- TCP, servizi, 76
  - connessione TCP, 76
  - handshaking, 76
  - servizio di trasferimento affidabile, 77
  - servizio orientato alla connessione (connection-oriented service), 76
- TCP Tahoe, 227
- TCP/IP, 4
- TCPCClient.py, 137
- TCPServer.py, 138
- TDM, 25
- Telnet, 194
- Tempo di backoff, 385
- Tempo di distribuzione, 119
- Temporizzazione, 75
- Trasferimento dati affidabile, 74
  - applicazioni che tollerano le perdite, 74
- Trasmissione store-and-forward, 20
  - buffer di output, 22
  - coda di output, 22
  - commutazione di circuito, 24
  - commutazione di pacchetto, 24
  - connessione end-to-end, 24
  - connessione punto a punto, 24
  - forwarding table, 23
  - packet loss, 22
  - perdita di pacchetto, 22
  - protocolli di instradamento, 24
  - ritardi di accodamento, 22
  - routing protocol, 24
- Throughput, 40, 74, 270
- Throughput medio, 40
- Tiers, 416
- Time division multiplexing, 25, 377
- Time frame, 378
- Time slot, 378
- Timing, 75
- Token, 388
- Top of rack (TOR) switch, 414
- Top-level domain (TLD) server, 108, 109
- Traceroute, 38
- Traffic engineering, 413
- Transmission Control Protocol, 4
- Transmission rate, 2, 262
- Transport-layer segment, 148
- Trasferimento dati affidabile, 152
- Trasferimento dati bidirezionale, 166
- Trasferimento dati unidirezionale, 166
- TTL, 394
- Tunnel, 293
- Tunneling, 293

## **U**

UDP (User Datagram Protocol), 151  
UDP, servizi, 77  
UDPClient.py, 132  
UDPServer.py, 134  
Unità trasmissiva massima, 190  
Unshielded Twisted Pair (UTP), 17  
User agent, 99

## **V**

Velocità (o tasso) di trasmissione, 2  
Virtual LAN, 407  
Virtual Local Area Network, 407  
Virtual Private Network, 413  
Virtualizzazione delle funzioni di rete, 345

VLAN, 407  
VLAN trunking, 409  
Voice-over-IP, 40

## **W**

Web cache, 91  
Web server, 81  
Web server e TCP, 157

## **Y**

YANG, 352, 356, 359