

## CAPITOLO

# 1

# 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, bilance 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, prederemo 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 cuore 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.

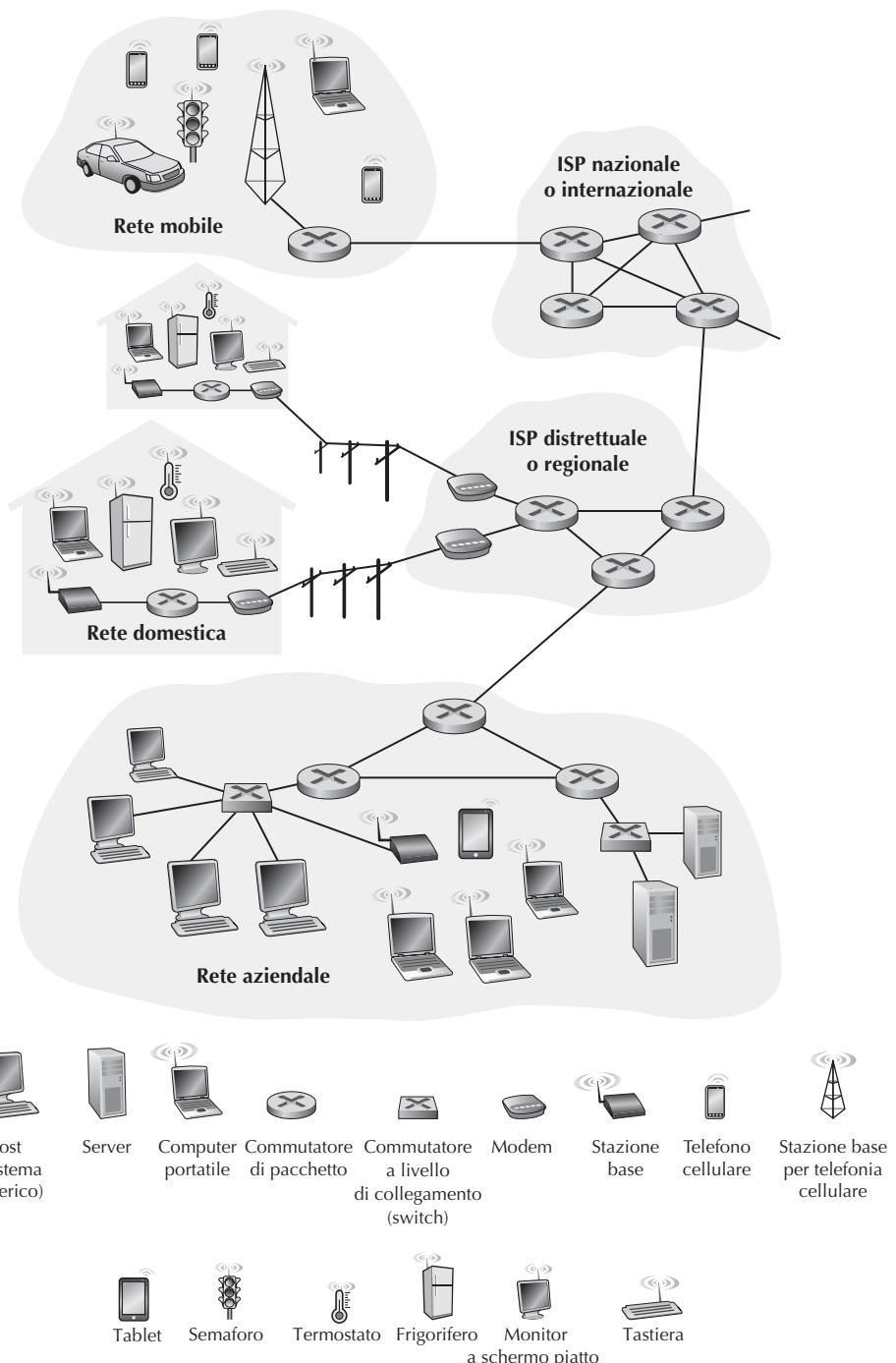
## 1.1 Che cos’è Internet?

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. Al giorno d’oggi invece vengono sempre più spesso connesse a Internet “cose”, quali computer portatili, tablet, smartphone, TV, console di gioco, termostati, sistemi di sorveglianza, elettrodomestici, orologi, occhiali, automobili, sistemi di controllo del traffico e molte altre. 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 (ospiti)** o **sistemi periferici (end system)**. Si stima che nel 2015 i sistemi periferici connessi a Internet fossero circa 5 miliardi che diventeranno 25 miliardi nel 2020 [Gartner 2014]. Nel complesso si stima che gli utenti Internet nel 2015 fossero 3,2 miliardi, pari a circa il 40% della popolazione mondiale [ITU 2015].

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.



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

Collegamenti diversi possono trasmettere dati a velocità differenti, e tale **velocità 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'intestazione a ciascuna di esse: l'insieme delle informazioni risultanti, nel gergo delle reti, viene chiamato **pacchetto**. 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 o cammino** (*route o path*) attraverso la rete. Cisco prevede che il traffico IP globale per anno supererà la soglia dello zettabyte ( $10^{21}$  byte) alla fine del 2016 e raggiungerà il valore annuale di 2 zettabyte nel 2019 [Cisco VNI 2015].

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 pacchetti agli incroci e i sistemi periferici agli edifici. Come un camion segue un percorso lungo la rete autostradale, così un pacchetto procede lungo un cammino 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. 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, quello senza fili (wireless) e in mobilità. Inoltre gli ISP rendono disponibile l'accesso a Internet ai fornitori di contenuti, connettendone direttamente i siti web e i video server a Internet. Per consentire la comunicazione, i provider di livello gerarchico più basso sono interconnessi a quelli nazionali e internazionali di livello più alto, quali Level 3 Communication, AT&T, Sprint e NTT. Un ISP di livello superiore è costituito da router ad alta velocità interconnessi tipicamente 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. Ecco dove entrano in gioco gli standard. Gli **standard di Internet** vengono sviluppati dall'Internet Engineering Task Force (IETF) [IETF 2016]. 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 settemila RFC. Anche altri enti specificano standard per i componenti di rete, in particolare per i collegamenti di rete. L'IEEE 802 LAN/MAN Standards Committee [IEEE 802 2016], 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 network, sistemi di videoconferenza e sistemi di raccomandazioni basati sulla posizione. Queste applicazioni sono dette **applicazioni distribuite**, 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 un'applicazione Internet reale? 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 appli-

cazione 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. Descriviamo 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 lancerla 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.

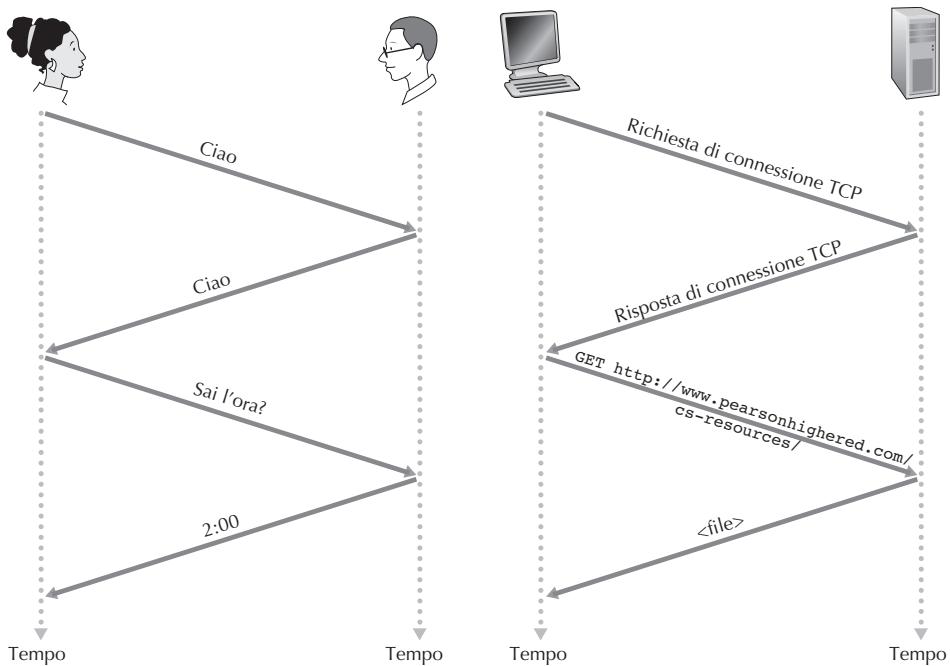
Abbiamo appena fornito due descrizioni di Internet, una nei termini dei suoi componenti hardware e software, l'altra nei termini dei servizi forniti alle applicazioni distribuite. Forse però il lettore è ancora confuso rispetto a che cosa sia Internet. Che cosa sono i commutatori di pacchetto e TCP/IP? 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**. 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



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

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 specifiche 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 implicito

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.

**BOX 1.1****TEORIA E PRATICA****Internet of Things (Internet delle cose)**

Riuscite a immaginare un mondo in cui quasi tutto è connesso a Internet? Un mondo in cui la maggior parte delle persone, automobili, biciclette, occhiali, orologi, giocattoli, attrezzature ospedaliere, sensori domestici, aule, sistemi di videosorveglianza, sensori atmosferici, prodotti in vendita e animali domestici sia connessa? Questo mondo della Internet of Things (IoT) potrebbe in realtà essere proprio dietro l'angolo.

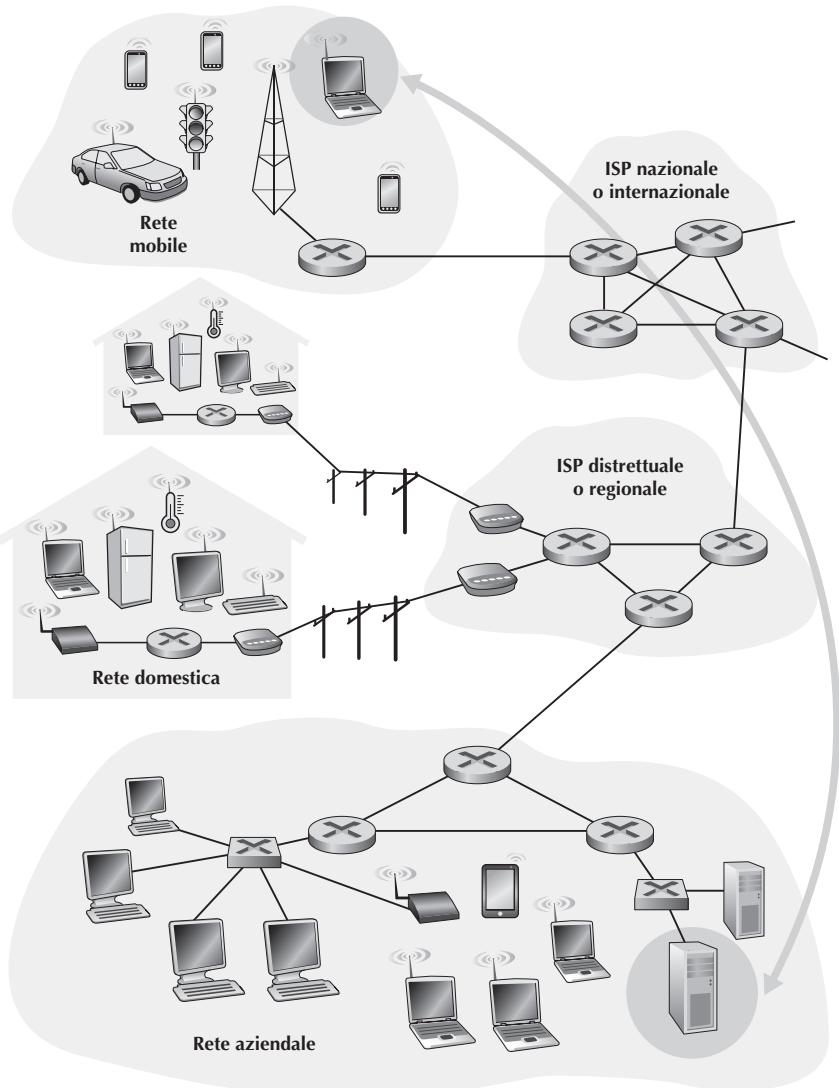
Secondo alcune stime, già nel 2015 vi erano 5 miliardi di cose connesse a Internet, e tale numero potrebbe raggiungere i 25 miliardi entro il 2020 [Gartner 2014]. Queste cose sono i nostri smartphone, che già ci seguono in casa, ufficio e automobile, inviando la nostra posizione geografica e i nostri dati di utilizzo al nostro ISP e alle applicazioni Internet che usiamo. Ma, oltre agli smartphone, una vasta gamma di cose non tradizionali è già disponibile, per esempio i dispositivi indossabili collegati a Internet, tra cui orologi (da Apple e molti altri) e occhiali. Questi ultimi possono, per esempio, caricare tutto ciò che vediamo nel cloud, permettendoci così di condividere le nostre esperienze visive con persone di tutto il mondo in tempo reale. Ci sono già disponibili cose connesse a Internet per la casa intelligente, come termostati che possono essere controllati a distanza dai nostri smartphone e bilance che ci permettono di visualizzare graficamente sullo smartphone il progresso della nostra dieta. Esistono giocattoli collegati a Internet, tra cui bambole che riconoscono e interpretano il discorso di un bambino e rispondono in modo appropriato.

La IoT potenzialmente offre benefici rivoluzionari agli utenti. Ma allo stesso tempo presenta enormi rischi per la sicurezza e la privacy, in quanto i dispositivi IoT e i rispettivi server possono subire un attacco via Internet. Ad esempio, un utente malintenzionato potrebbe prendere il controllo di una bambola collegata a Internet e parlare direttamente con un bambino; o un utente malintenzionato potrebbe entrare in un database che memorizza le informazioni mediche e personali raccolte dai dispositivi indossabili. Queste preoccupazioni sulla sicurezza e sulla privacy potrebbero minare la fiducia dei consumatori necessaria perché queste tecnologie possano esprimere il loro pieno potenziale, riducendone la diffusione [FTC 2015].

## 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 dall'esterno della rete 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 verso il nucleo per esaminare la commutazione e l'indirizzamento 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” (si veda il Box 1.1 “Teoria e pratica”).



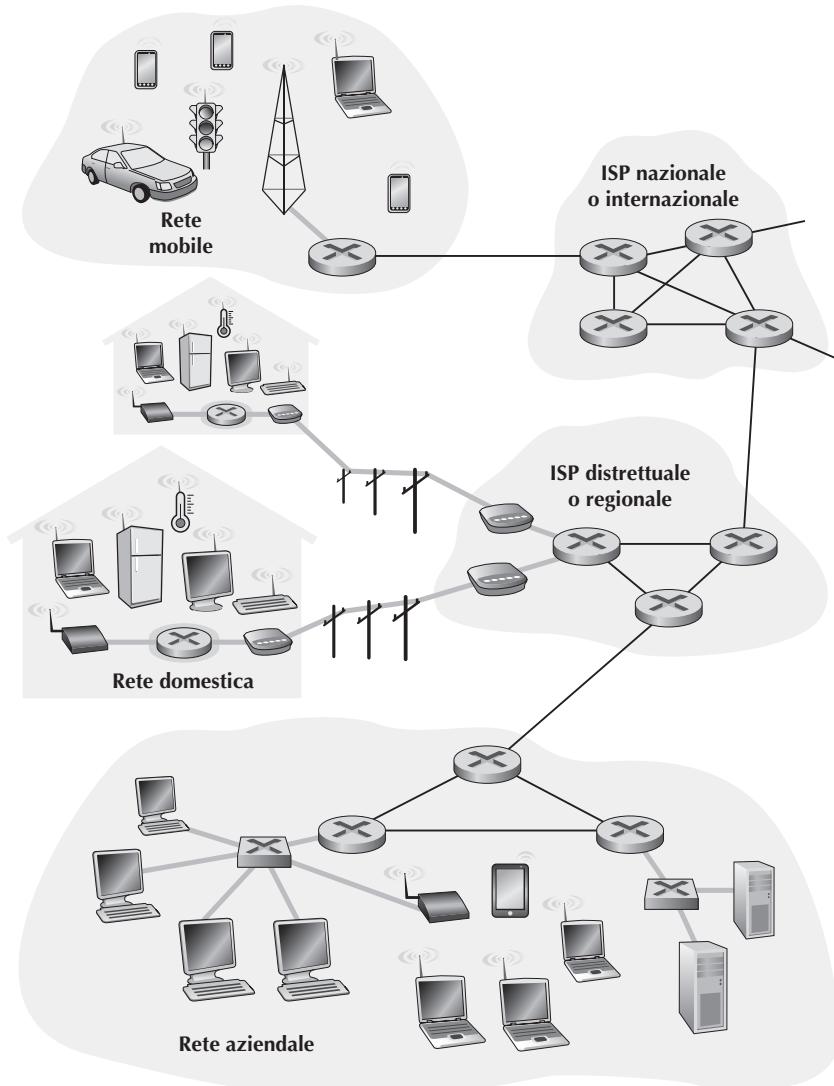
**Figura 1.3** Interazione tra sistemi periferici.

I sistemi periferici vengono anche detti **host** in quanto ospitano (ed eseguono) programmi applicativi quali browser e 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 host che richiedono dei servizi e tendono a essere PC, 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 pa-

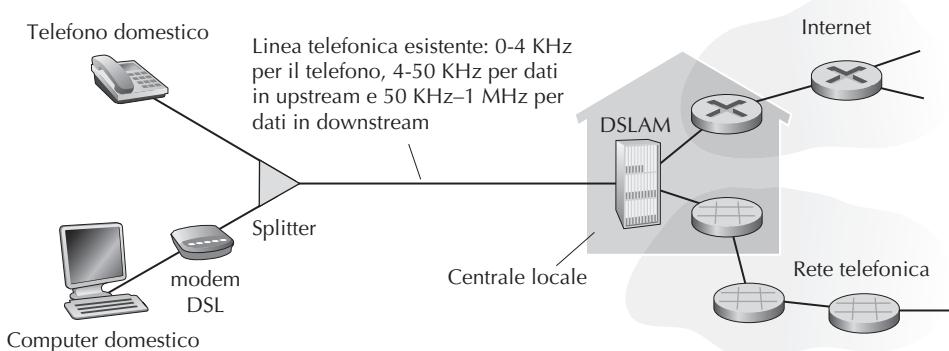
gine Web e i video è collocata in grandi **data center**. Google, per esempio, dispone di 50-100 data center, di cui 15 molto grandi, aventi ciascuno più di 100.000 server.

### 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 acces-



**Figura 1.4** Reti di accesso.



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

so, evidenziate da linee spesse e ombreggiate e gli ambienti (casa, azienda e aree geografiche con accesso in mobilità wireless) in cui vengono usate.

### Accesso residenziale: DSL, via cavo, FTTH, dial-up e satellite

Già dal 2014 più del 78% delle abitazioni nei paesi sviluppati ha accesso a Internet; tale percentuale supera l’80% in Corea, Paesi Bassi, Finlandia e Svezia, quasi sempre a larga banda [ITU 2015]. Dato il vivo interesse per gli accessi residenziali, iniziamo da questi la nostra trattazione delle connessioni a Internet.

Oggigiorno 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 (doppino telefonico intrecciato in rame, trattato nel Paragrafo 1.2.2) 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 segnali analogici in arrivo dalle abitazioni vengono riconvertiti in formato digitale nel DSLAM.

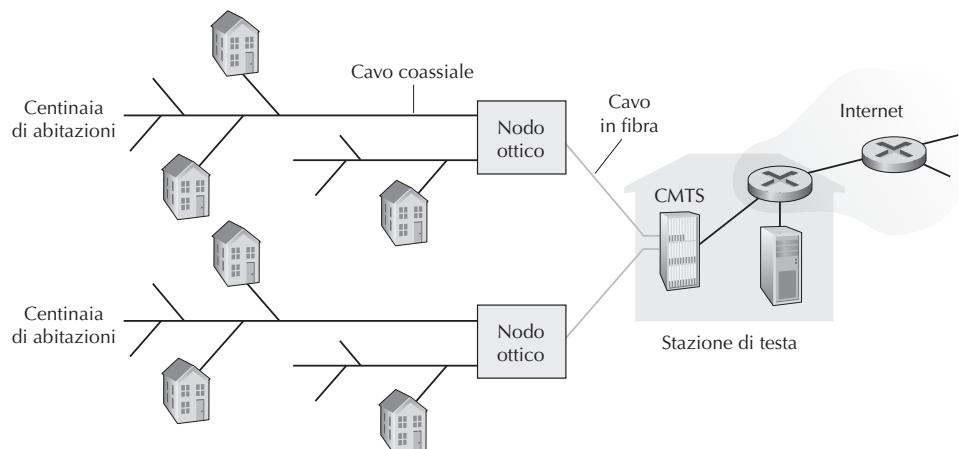
Le linee telefoniche residenziali trasportano contemporaneamente dati e segnali telefonici tradizionali codificandoli in tre bande di frequenza non sovrapposte:

- 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 possano 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 [Dirschinger 2007].

Gli standard DSL definiscono il tasso di trasmissione in downstream a 12 Mbps e quello in upstream a 1,8 Mbps [ITU 1999], o anche 55 Mbps in downstream e 15 Mbps in upstream [ITU 2006]. 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 televisione 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 general-



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

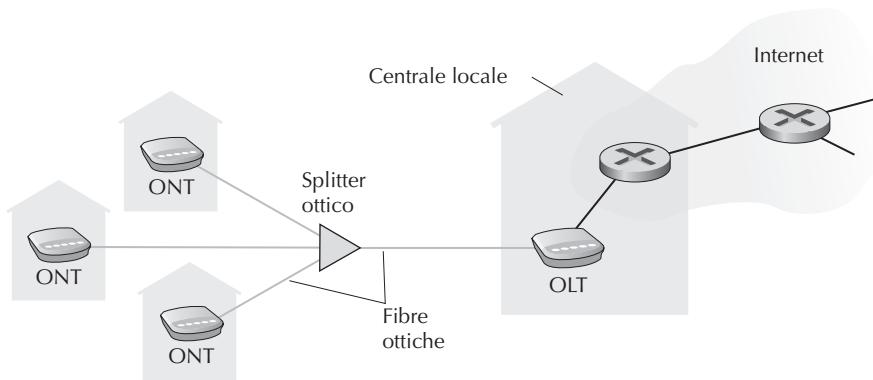
mente da 500 a 5000 abitazioni. Tale sistema viene spesso chiamato hybrid fiber coax (HFC), in quanto impiega sia la fibra ottica sia il cavo coassiale.

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 (CMTS, *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 definisce tassi di trasmissione in downstream fino a 42,8 Mbps e in upstream fino a 30,7 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 più dell' 85% degli accessi residenziali a banda larga negli Stati Uniti, una tecnologia promettente che vanta velocità ancora maggiori è detta **fiber to the home (FTTH)** [FTTH Council 2016]. Come suggerito dal nome, il concetto di FTTH è semplice: fornire fibra ottica dalla centrale locale direttamente alle abitazioni. Attualmente in molti stati quali Emirati Arabi Uniti, Corea del Sud, Hong Kong, Giappone, Singapore, Taiwan, Lituania e Svezia FTTH è impiegata in più del 30% dei casi [FTTH Council 2016].

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



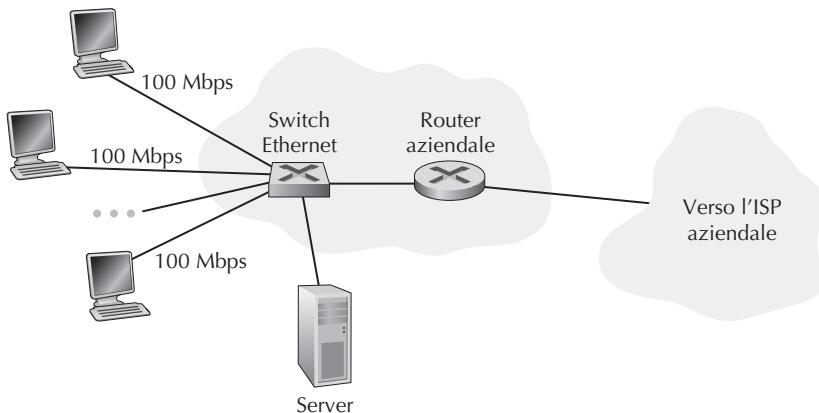
**Figura 1.7** Accesso a Internet tramite FTTH.

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

La tecnologia FTTH può potenzialmente fornire velocità di accesso a Internet di gigabit al secondo. Tuttavia, la maggior parte degli ISP FTTH offre velocità diverse, con le più alte, naturalmente, a prezzi maggiori. La velocità media in downstream di un utente FTTH negli Stati Uniti nel 2011 era circa di 20 Mbps (da confrontare con quelle via cavo di 13 Mbps e quelle DSL inferiori ai 5 Mbps) [FTTH Council 2011b].

Sono inoltre disponibili altre due tecnologie di accesso residenziale a Internet. Nelle località in cui non sono disponibili le reti DSL, via cavo e FTTH (per esempio in alcune località rurali), è utilizzabile un collegamento satellitare per connettere un'abitazione a Internet a velocità maggiori di 1 Mbps; due di questi provider satellitari sono StarBand e HughesNet. L'accesso in dial-up su linee telefoniche tradizionali è basato sullo stesso modello della DSL: un modem in casa dell'utente si connette a un modem dell'ISP su una linea telefonica. L'accesso dial-up, se paragonato a reti di accesso a larga banda come la DSL, è terribilmente lento con i suoi 56 kbps.



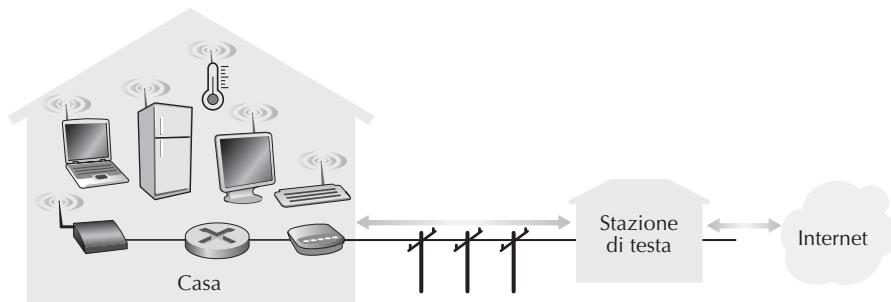
**Figura 1.8** Accesso a Internet tramite Ethernet.

### 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. L'accesso tramite Ethernet ha generalmente velocità di 100 Mbps, che possono arrivare fino a 1 o anche 10 Gbps.

Sempre più utenti accedono a Internet via wireless da computer portatili, smartphone, tablet e altre “cose” (si veda ancora il Box 1.1). 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. In molte città è possibile trovarsi a un incrocio ed essere situato all'interno del raggio di dieci o venti access point (per una mappa globale e navigabile degli access point 802.11, scoperti e riportati su un sito web da persone che si divertono a fare cose di questo tipo, si veda [wigle.net 2016]). La tecnologia IEEE 802.11 più comunemente installata, che discuteremo in dettaglio nel Capitolo 7, fornisce una velocità trasmissiva condivisa intorno ai 100 Mbps.

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 [Edwards 2011]. La Figura 1.9 presenta lo schema di una tipica rete domestica, composta da un computer portatile e da un PC fisso, un access point che comunica con il



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

computer portatile in modalità wireless, un cable modem che fornisce accesso a banda larga a Internet e un router che interconnette l'access point e i PC fissi con il cable modem. 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

I dispositivi iPhone e Android vengono sempre più frequentemente impiegati per inviare messaggi, condividere fotografie, 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 terza generazione (3G), che consentono accesso wireless a Internet con commutazione a pacchetto e su scala geografica a velocità che superano 1 Mbps. Tuttavia sono già disponibili tecnologie di accesso su scala geografica a velocità ancora più alta: la quarta generazione (4G). LTE (“Long-Term Evolution”, candidato al premio come peggior acronimo dell'anno) fonda le sue basi nella tecnologia 3G e può potenzialmente raggiungere velocità superiori a 10 Mbps. In ambiti industriali sono state riscontrate velocità LTE in downstream di molte decine di Mbps. Tratteremo i principi fondamentali delle reti wireless e mobili, così come le tecnologie WiFi, 3G, LTE (e altre ancora) nel Capitolo 7.

## 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 coaxiali, 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 doppini 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 odierne 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 decine 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. [Mukherjee 2006, Ramaswami 2010] approfondiscono vari aspetti di questo tipo di reti.

### 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 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.

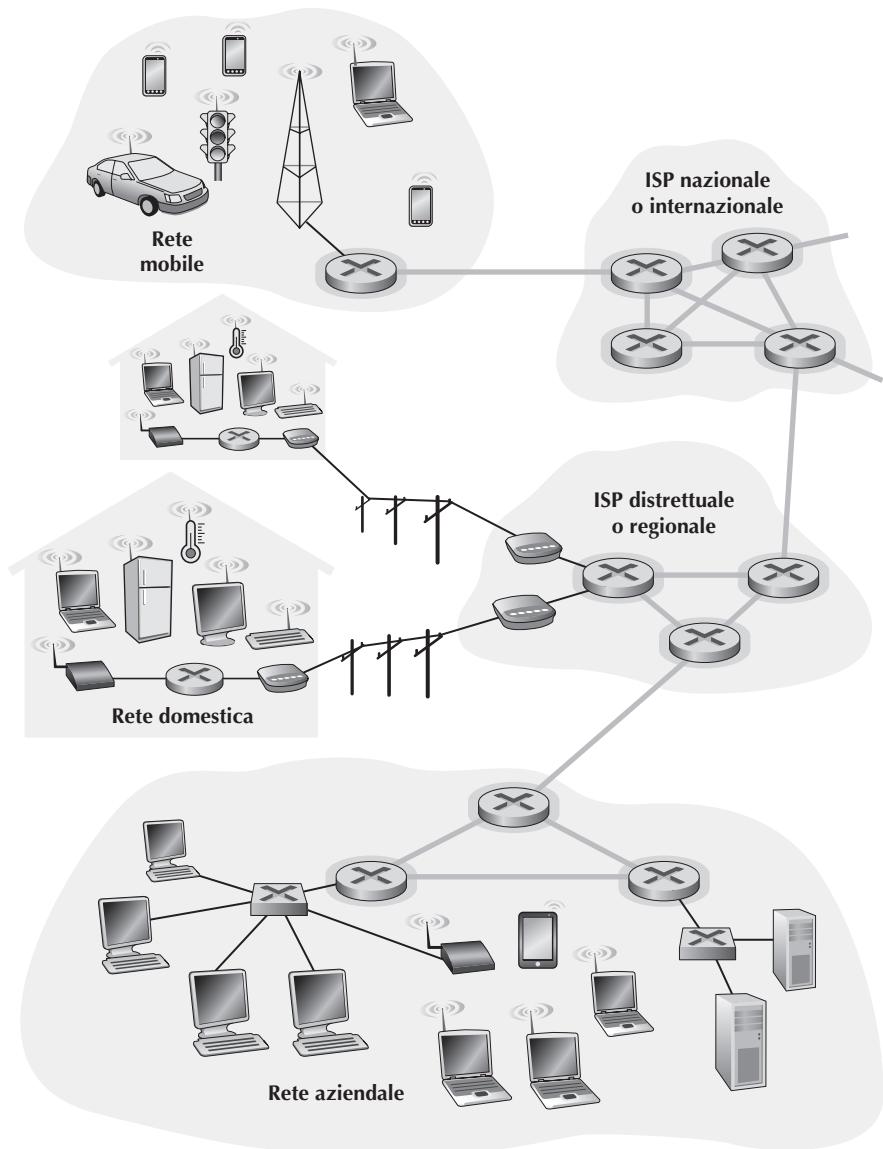
A grandi linee, i canali radio terrestri possono essere classificati in tre gruppi: quelli che operano su distanze molto piccole (uno o due metri), quelli che operano in aree locali in un raggio di qualche centinaio di metri, e quelli che operano in aree più vaste, che si estendono per decine di chilometri. 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.

### 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*) [Wiki Satellite 2016].

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.



**Figura 1.10** Il nucleo della rete.

### 1.3 Il nucleo della rete

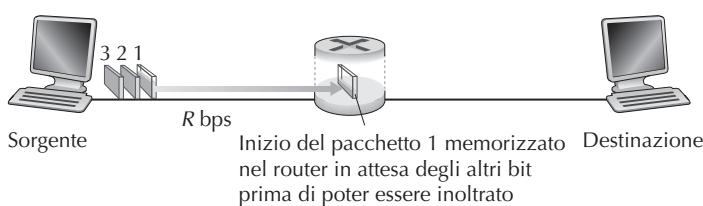
Esaminati i confini di Internet, approfondiamo ora in dettaglio il nucleo della rete: una maglia di commutatori di pacchetti e collegamenti che interconnettono i sistemi periferici di Internet. Nella Figura 1.10 il nucleo della rete è evidenziato con linee più spesse e ombreggiate.

### 1.3.1 Commutazione 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 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 inviano 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 poterne cominciare a trasmettere sul collegamento in uscita il primo bit. Per capire meglio la trasmissione store-and-forward si consideri la semplice rete 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



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

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:

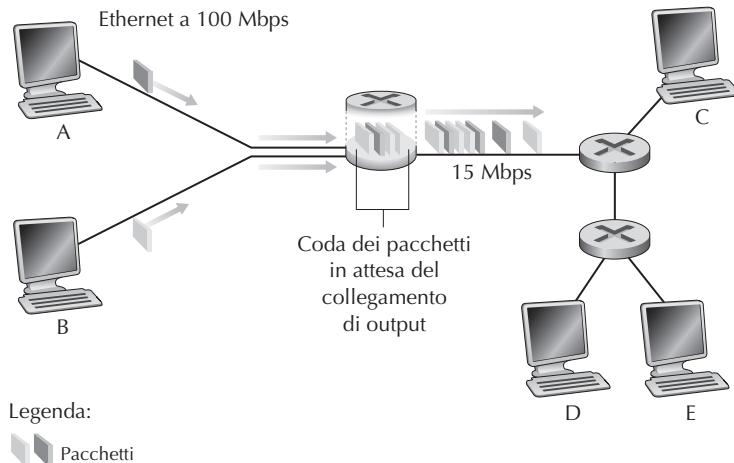
$$d_{\text{end-to-end}} = N \frac{L}{R} \quad (1.1)$$

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** nei buffer di output. 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



**Figura 1.12** Commutazione di pacchetto.

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 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 descriviamo 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 **tavola 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.

Non sarebbe bello poter seguire il percorso che i pacchetti compiono da un punto all'altro di Internet? Si può fare interagendo con il programma Traceroute; visitate il sito [www.traceroute.org](http://www.traceroute.org): scegliete una sorgente e tracciate il percorso da essa al vostro computer (si consulti a proposito il Paragrafo 1.4).

### 1.3.2 Commutazione di circuito

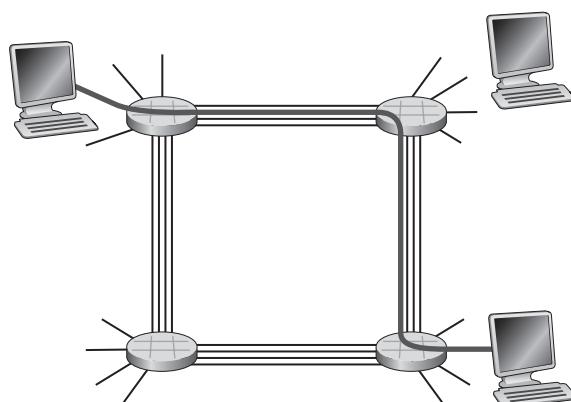
Per spostare i dati in una rete di collegamenti e commutatori esistono due approcci fondamentali: la **commutazione di circuito** e la **commutazione di pacchetto**. Di quest'ultima abbiamo parlato nel paragrafo precedente, esaminiamo ora le reti a commutazione di circuito.

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 trasmittiva 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 cia-



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

scuno 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 trasmissiva totale del collegamento per la durata 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.

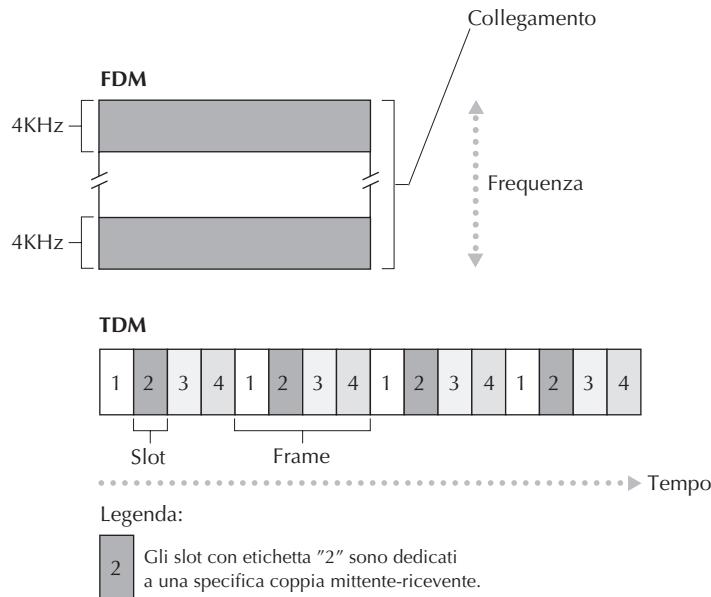
Si consideri invece che cosa accade quando un host invia un pacchetto a un altro host su una rete a commutazione di pacchetto, quale è Internet. Come nella commutazione di circuito, il pacchetto viene trasmesso su di una sequenza di collegamenti ma, a differenza della commutazione di circuito, il pacchetto 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.

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.



**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).

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 predispone 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 sprecate, 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 commuta-

zione 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 delle 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 o di televisione via cavo, 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*: 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 invero 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.

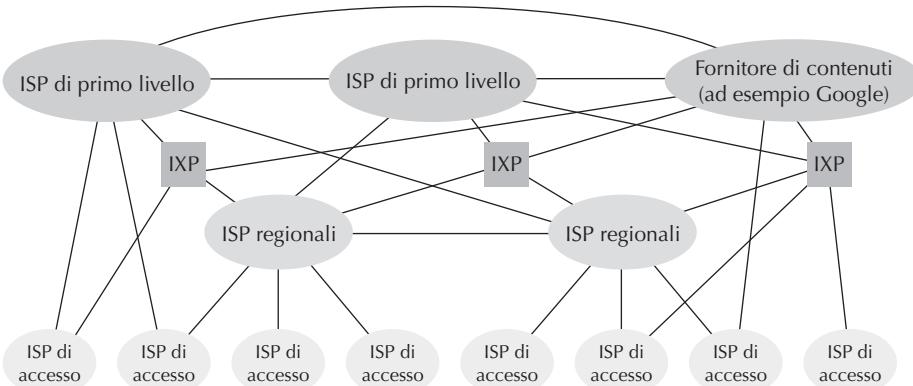
Tornando alla nostra rete di reti, non solo ci sono più ISP di primo livello in competizione, ma anche nelle regioni ci sono più ISP regionali in competizione tra di loro. In questa gerarchia ogni ISP di accesso paga l'ISP regionale a cui si connette, che a sua volta paga il suo ISP di primo livello. 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 [Ager 2012]. Nell'Internet attuale ci sono circa 400 IXP [IXP List 2016]. Ci riferiremo a questo ecosiste-

---

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



**Figura 1.15** Interconnessioni tra ISP.

ma, 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 odierna. La *Struttura di rete 5*, mostrata nella Figura 1.15, è costruita sulla *Struttura di rete 4* aggiungendo le reti che si occupano di distribuire contenuti (*content provider networks*). Google è attualmente uno degli esempi di punta di tali reti. Al momento si stima che Google disponga da 50 a 100 data center distribuiti tra Nord America, Europa, Asia, Sud America e Australia. Alcuni di questi data center ospitano più di centomila server, mentre altri più piccoli ne ospitano solo centinaia. 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 maggior 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

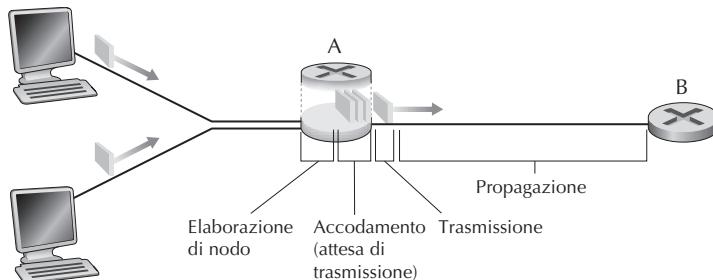
Nel Paragrafo 1.1 abbiamo affermato che Internet può essere vista come un’infrastruttura che fornisce servizi alle applicazioni distribuite in esecuzione sui sistemi periferici. Idealmente, vorremmo che i servizi Internet fossero in grado di spostare una quantità di dati qualsiasi tra due sistemi periferici, 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 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.



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

### 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.

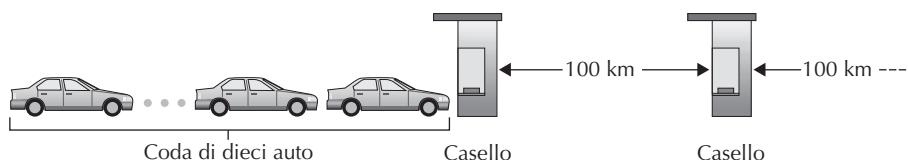
### Ritardo di propagazione

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, doppiino in rame e così via) ed è compresa nell'intervallo che va dai  $2 \times 10^8$  m/s ai  $3 \times 10^8$  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,  $s$  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



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

$\text{km} / (100 \text{ km/h}) = 1$  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, 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: il sito web del testo fornisce un'applet Java 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

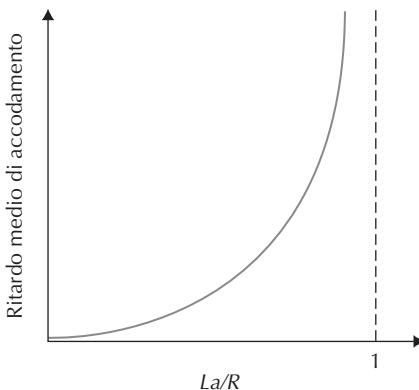
La componente più complessa e interessante del ritardo totale di nodo è il ritardo di accodamento,  $d_{\text{acc}}$ , tanto da aver ispirato migliaia di articoli e numerosi libri: [Bertsekas 1991, Daigle 1991, Kleinrock 1975 e 1976, Ross 1995]. In questa sede ne forniamo solo una trattazione ad alto livello e intuitiva, ma il lettore appassionato può andarsene 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 coda, 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 semplificità, 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 influisce 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



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

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.

Per capire realmente da che cosa sia determinato il ritardo di accodamento, siete incoraggiati, ancora una volta, a visitare il sito web del volume che fornisce un'applet Java 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'applet Java 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 pac-

chetto 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

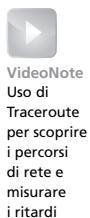
$$d_{\text{end-to-end}} = N(d_{\text{elab}} + d_{\text{trasm}} + d_{\text{prop}}) \quad (1.2)$$

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  $3N$  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 l'University of Massachusetts) alla destinazione cis.poly.edu (presso la Polytechnic University of Brooklyn). 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 cs-gw (128.119.240.254) 1.009 ms 0.899 ms 0.993 ms
2 128.119.3.154 (128.119.3.154) 0.931 ms 0.441 ms 0.651 ms
3 -border4-rt-gi-1-3.gw.umass.edu (128.119.2.194) 1.032 ms 0.484 ms 0.451 ms
4 -acr1-ge-2-1-0.Boston.cw.net (208.172.51.129) 10.006 ms 8.150 ms 8.460 ms
5 -agr4-loopback.NewYork.cw.net (206.24.194.104) 12.272 ms 14.344 ms 13.267 ms
6 -acr2-loopback.NewYork.cw.net (206.24.194.62) 13.225 ms 12.292 ms 12.148 ms
7 -pos10-2.core2.NewYork1.Level3.net (209.244.160.133) 12.218 ms 11.823 ms 11.793 ms
8 -gige9-1-52.hsipaccess1.NewYork1.Level3.net (64.159.17.39) 13.081 ms 11.556 ms 13.297 ms
9 -p0-0.polyu.bbnplanet.net (4.25.109.122) 12.716 ms 13.052 ms 12.786 ms
10 cis.poly.edu (128.238.32.126) 14.080 ms 13.035 ms 12.802 ms

```

Nel percorso seguito abbiamo nove 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 3 è border4-rt-gi-1-3.gw.umass.edu e il suo indirizzo è 128.119.2.194. 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 1,03 ms. I ritardi di andata e ritorno per le successive due prove sono stati di 0,48 e 0,45 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 6 appaiono superiori ai ritardi nel Router 7.

Per provare Traceroute visitate il sito <http://www.traceroute.org/>, che fornisce un'interfaccia web e una lunga lista di sorgenti. Basta scegliere una sorgente e immettere il nome della destinazione, e il programma Traceroute svolgerà il resto del lavoro. Ci sono parecchi programmi software gratuiti che forniscono un'interfaccia grafica a Traceroute; uno dei nostri preferiti è PingPlotter [PingPlotter 2016].

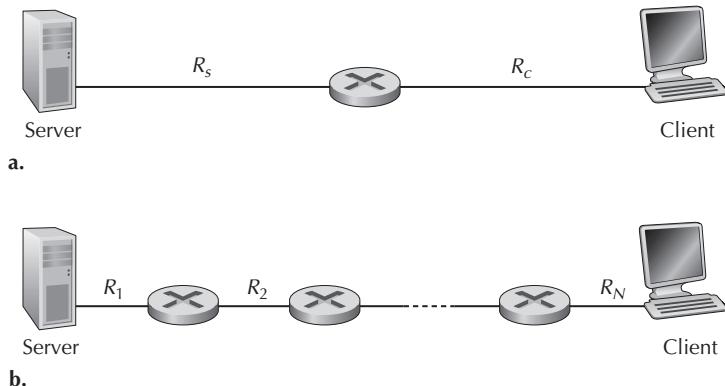
## Sistemi periferici, applicazioni e altri ritardi

Oltre ai ritardi di elaborazione, trasmissione e propagazione, si potrebbero manifestare ulteriori, significativi, ritardi nei sistemi periferici. Per esempio, un sistema 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 peer a un altro in un sistema di condivisione di file P2P. Il throughput istantaneo in ogni istante di tempo è la velocità (in bps) alla quale B sta ricevendo il file: molte applicazioni, compresi molti sistemi di condivisione file P2P, mostrano il throughput istantaneo durante il download nell’interfaccia utente. 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

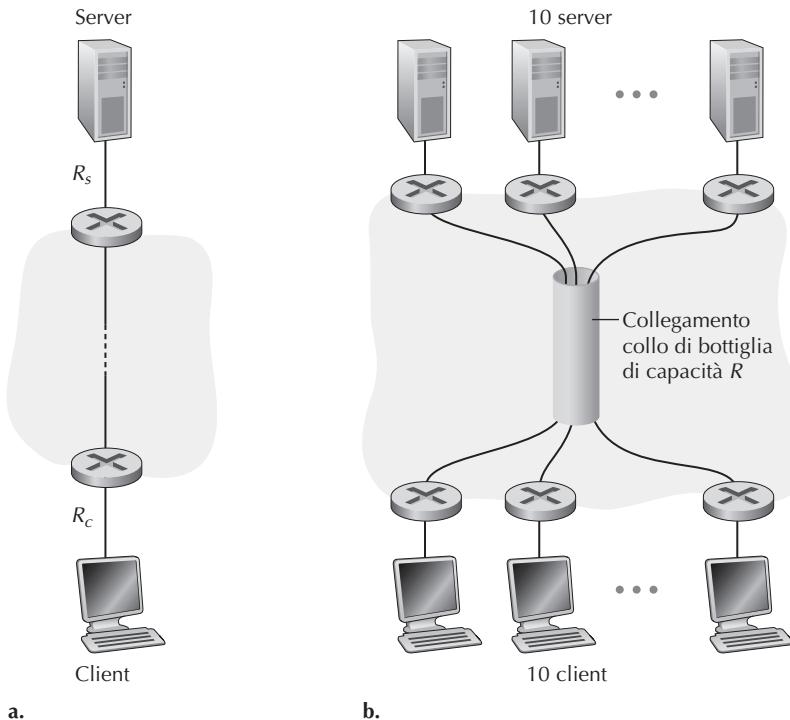


**Figura 1.19** Throughput per il trasferimento di un file dal server al client.

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 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 e 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



**Figura 1.20** Throughput end-to-end: (a) il client scarica un file dal server; (b) 10 client scaricano da 10 server.

dal server al client. Dato che, in questo esempio, il nucleo della rete è come un condotto largo, la velocità alla quale i bit fuiscono 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. 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  $R_s = 2$  Mbps,  $R_c = 1$  Mbps e  $R = 5$  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.

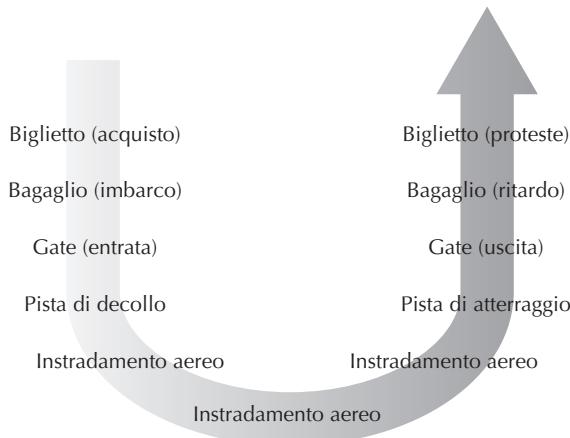
## 1.5 Livelli dei protocolli e loro modelli di servizio

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.

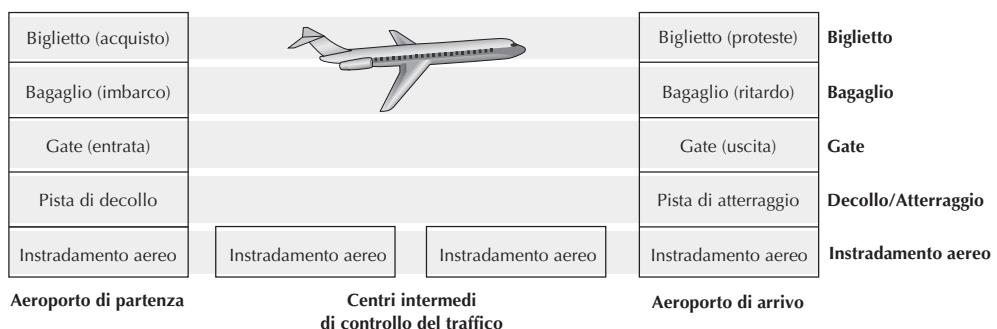
Possiamo già notare alcune analogie con le reti di calcolatori: la linea aerea vi trasporta da una località di partenza a una di destinazione, così come in Internet un pacchetto viene trasportato da un host sorgente a uno di destinazione. Ma questo non è



**Figura 1.21** Viaggio aereo: azioni.

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 imbarcati. Per i passeggeri che hanno già superato il gate, esistono le funzioni di decollo e atterraggio, e mentre si vola è in funzione l'intradamento 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 for-



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

nisce 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

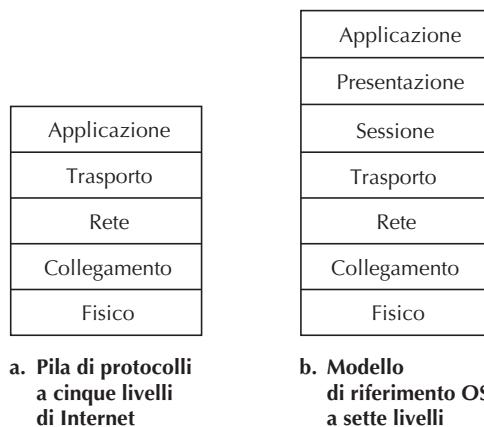
Dopo aver parlato di linee aeree, rivolgiamo ora la nostra attenzione ai protocolli di rete. Per dare struttura alla loro progettazione, i progettisti organizzano i protocolli e l'hardware e software che li implementano in **livelli o strati (layer)**. Ciascun protocollo appartiene a uno dei livelli, 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 duplichia 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 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 consiste di cinque livelli: fisico, collegamento, rete, trasporto e applicazione (Figura 1.23(a)). 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.



**Figura 1.23** La pila di protocolli Internet (a) e il modello di riferimento OSI (b).

### 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 (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 trasmittiva 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

<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.*).

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 colante 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.

### 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 trasmissivo (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.

### Modello OSI

Dopo aver trattato in dettaglio la pila di protocolli Internet dobbiamo menzionare il fatto che questa non è l'unica pila di protocolli. In particolare, nei lontani anni '70, l'International Organization for Standardization (ISO) propose che le reti di calcolatori fossero organizzate in sette livelli, chiamati modello Open Systems Interconnection (OSI) [ISO 2016]. Il modello OSI prese forma quando i protocolli, che sarebbero

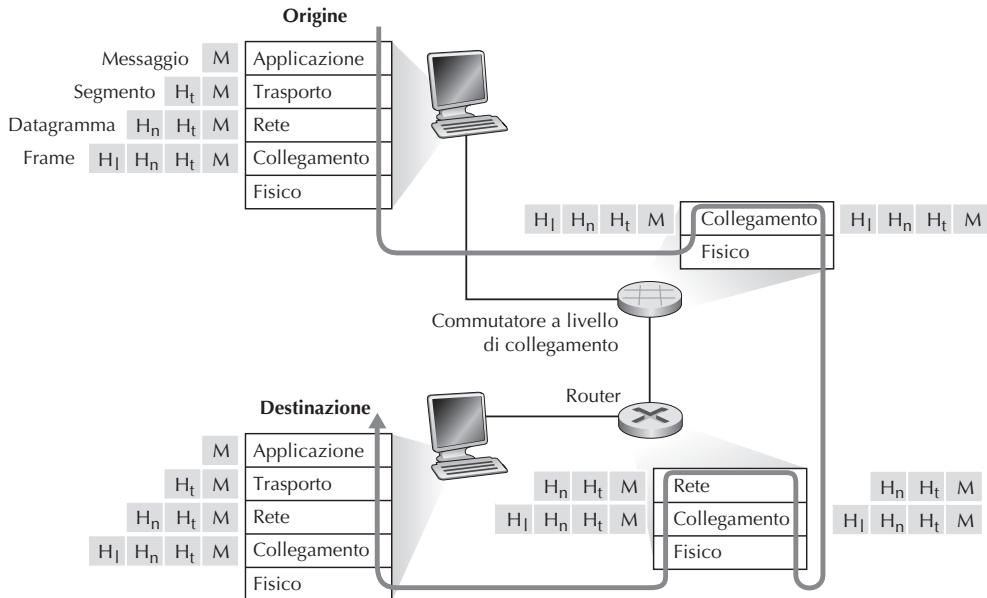
diventati i protocolli di Internet, erano ai loro esordi ed erano solo uno dei tanti insiemi di protocolli in via di sviluppo. Infatti, gli inventori del modello OSI originale non avevano probabilmente in mente Internet, quando lo crearono. Ciò nonostante, a partire dagli anni '70, furono organizzati molti corsi aziendali e universitari intorno al modello OSI a sette livelli. A causa del suo impatto iniziale sulla didattica nel campo delle reti, il modello a sette livelli continua a rimanere in alcuni libri di testo e in alcuni corsi professionali.

I sette livelli del modello di riferimento OSI, illustrati nella Figura 1.23 (b), sono: applicazione, presentazione, sessione, trasporto, rete, collegamento e fisico. Le funzionalità di cinque di questi livelli sono più o meno le stesse degli omonimi della controparte Internet, quindi consideriamo i due livelli aggiuntivi presenti nel modello di riferimento OSI: il livello di presentazione e quello di sessione. Il ruolo del livello di presentazione è fornire servizi che consentono ad applicazioni che vogliono comunicare di interpretare il significato dei dati scambiati. Questi servizi comprendono la compressione e la cifratura dei dati (che sono auto esplicative) come pure la descrizione dei dati che libera le applicazioni dalle preoccupazioni riguardo al formato interno nel quale sono rappresentati/memorizzati, che potrebbe essere diverso da un computer a un altro. Il livello di sessione fornisce la delimitazione e la sincronizzazione dello scambio di dati, compresi i mezzi per costruire uno schema di controllo e di recupero degli stessi.

Il fatto che a Internet manchino due livelli che invece si trovano nel modello di riferimento OSI pone un paio di domande interessanti: i servizi forniti da questi due livelli sono meno importanti? Che cosa accade se un'applicazione necessita di questi servizi? La risposta di Internet a entrambe queste domande è la stessa: dipende da chi sviluppa l'applicazione decidere se un servizio è importante e, se lo è, è compito suo sviluppare queste funzionalità nell'applicazione.

## 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.



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

La Figura 1.24 illustra anche l'importante **concepto di encapsulamento**. 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. Supponente 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 pagine web e messaggi di posta elettronica, MP3, 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 i malware auto-replicanti possono diffondersi a velocità esponenziale. I malware possono diffondersi sotto forma di virus o worm. I **virus** sono malware che richiedono una qualche forma di interazione con l’utente per infettarne il dispositivo. L’esempio classico è l’allegato e-mail contenente codice eseguibile dannoso. Se un utente riceve e apre questo tipo di allegato, inavvertitamente esegue il malware sul dispositivo. In genere questi tipi di virus delle e-mail sono auto replicanti: una volta eseguiti i virus potrebbero inviare un messaggio identico con un allegato dannoso identico a, per esempio, tutti i contatti nella rubrica dell’utente. I cosiddetti **worm** sono malware che possono entrare in un dispositivo senza alcuna interazione esplicita con l’utente. Per esempio, l’utente potrebbe eseguire un’applicazione di rete vulnerabile, contro la quale un attaccante può mandare un malware. In alcuni casi, senza alcun intervento dell’utente, l’applicazione può accettare il malware da Internet ed eseguirlo, creando un worm. Questo, una volta infettato il dispositivo, esplora Internet alla ricerca di altri host che hanno in esecuzione la stessa applicazione di rete vulnerabile. Quando trova altri host vulnerabili, manda loro una copia di se stesso. Oggi i malware sono molto diffusi ed è costoso difendersi da loro. Man mano che procederete in questo testo, vi incoraggeremo a riflettere sul seguente quesito: come possono i progettisti di reti difendere i dispositivi collegati a Internet dagli attacchi basati su malware?

## Attacchi ai server e all'infrastruttura di rete

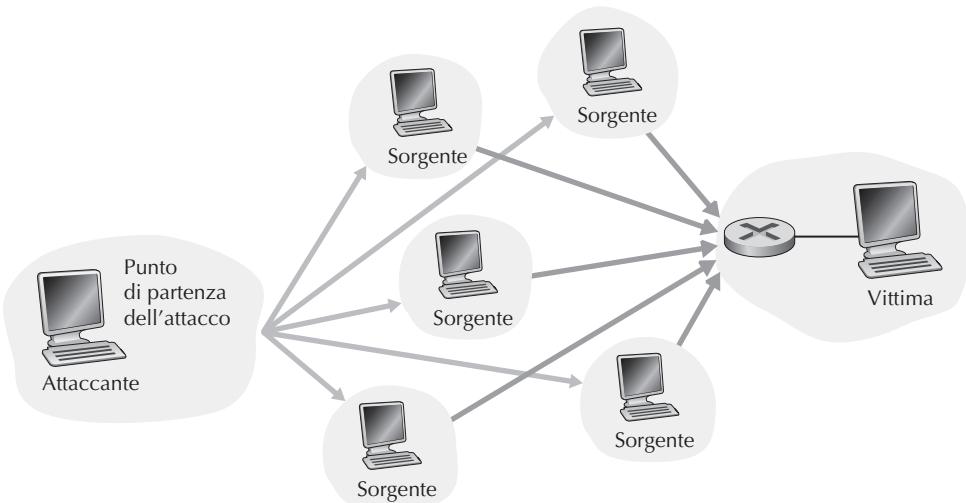
Un'ampia classe di minacce alla sicurezza può essere classificata come attacchi di **negazione del servizio** (DoS, *denial-of-service*). Come suggerisce il nome, un attacco DoS rende inutilizzabile dagli utenti legittimi una rete, un host o un'altra parte di infrastruttura. Web server di posta elettronica, 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 [Moore 2001]. Il sito Digital Attack Map [DAM 2016] 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 2016]. 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



**Figura 1.25** Un attacco DoS distribuito.

(Capitolo 7). Se da un lato l’accesso a Internet da qualsiasi luogo è estremamente comodo e rende possibile 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 estrarlarne 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 2016] proposte in questo testo (si veda il laboratorio con Wireshark introduttivo 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 on-line esamineremo come questa venga applicata alla sicurezza di rete.

## Mascheramento

È sorprendentemente facile (avrete le conoscenze su come fare appena procederete 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 on-line.

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) 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, dovreste 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,



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

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.

## 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 crea-

rono 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, e-Bay, Google e Amazon.

### 1.7.5 Il nuovo millennio

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, 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 multutente (Skype, Facetime e Google Hangouts).
- La sempre maggiore presenza di reti pubbliche Wi-Fi ad alta velocità (54 Mbps o superiori) e l'accesso a Internet a media velocità (decine di Mbps), per le reti di telefonia cellulare 4G, non solo sta rendendo possibile la connessione permanente, ma sta anche consentendo nuovi e interessanti servizi basati sulla posizione quali Yelp, Tinder, Yik Yak 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.
- I 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 e 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 di commercio su Internet stanno ora eseguendo le loro applicazioni nel “cloud”, come EC2 di Amazon, l’Application Engine di Google o Azure di Microsoft. 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 implicito 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 promes-

sa, 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 quest’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
8. Sicurezza nelle reti di calcolatori (on-line)
9. Reti multimediali.

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 da 7 a 9) analizza in dettaglio tre aspetti di fondamentale importanza (e in qualche modo indipendenti) nelle odierne reti di calcolatori. Nel Capitolo 7 esaminiamo le reti wireless mobili, incluse le LAN wireless (compresi Wi-Fi e Bluetooth), le reti di telefonia cellulare (compreso il GSM, 3G e 4G) e la mobilità (sia nelle reti IP che GSM). Nel corso del Capitolo 8 (on-line) (sicurezza nelle reti di calcolatori) 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. Nell’ultimo capitolo on-line (reti multimediali) prenderemo in considerazione le applicazioni audio e video quali la telefonia su Internet, la videoconferenza e lo streaming di dati multimediali. Considereremo, inoltre, come le reti a commutazione di pacchetto possano essere progettate per fornire un servizio di qualità adeguato per le applicazioni audio e video.

## CAPITOLO

# 2

## 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 in 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 per 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 la navigazione sul Web, la ricerca di informazioni e il commercio elettronico (e-commerce). Altre due applicazioni per Internet sono la messaggistica istantanea e la condivisione di file tramite P2P: le due killer application introdotte a fine millennio. Dall'anno 2000 abbiamo assistito a un'esplosione di popolari applicazioni voce e video, 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, Twitter e WeChat, che hanno creato una rete di persone sopra la rete Internet composta da router e collegamenti fisici.

<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.).

Più recentemente, con la diffusione degli smartphone, sono apparse molte nuove applicazioni basate sulla posizione quali Yelp, Tinder, Yik Yak e Waz. E 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!

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 (il Capitolo 9 on-line tratterà le applicazioni multimediali, compresi lo streaming video e il VoIP). 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.

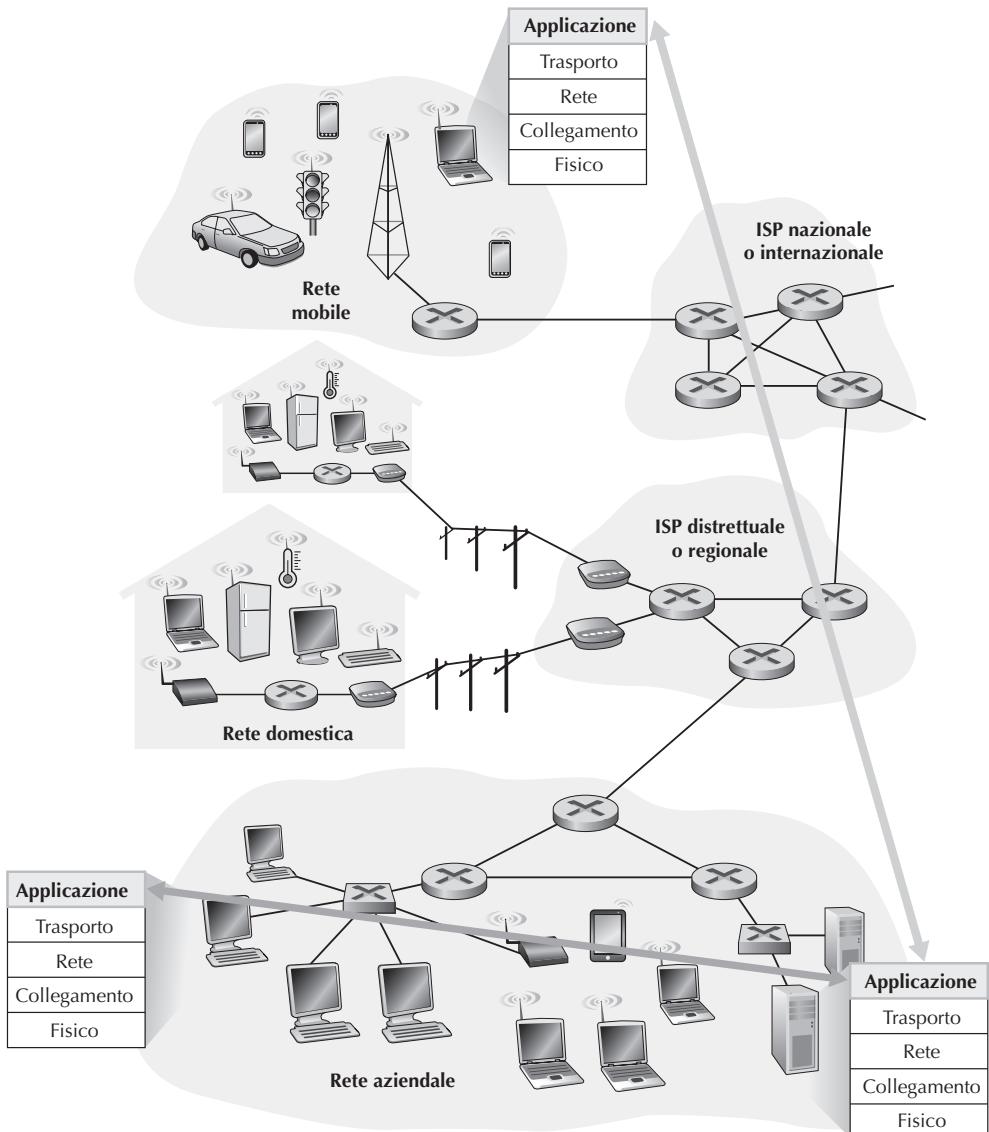
Il livello di applicazione è un ambiente particolarmente favorevole per iniziare lo studio dei protocolli perché ci è familiare. Ci fornirà una buona base per comprendere di che cosa si occupano i protocolli e introdurrà molti argomenti che rivedremo durante lo studio dei protocolli a livello di trasporto,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

## 2.1 Princìpi delle applicazioni di rete

Supponete 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 compilazione 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, PDA, telefono cellulare e così via) e il web server che si trova nell'host che viene di solito chiamato anch'esso web server. Considerando un altro esempio, in un sistema di condivisione di file tramite P2P troviamo un programma su ogni host che partecipa alla comunità di condivisione. In questo caso, i programmi in esecuzione sugli host possono essere simili o identici.

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 li-



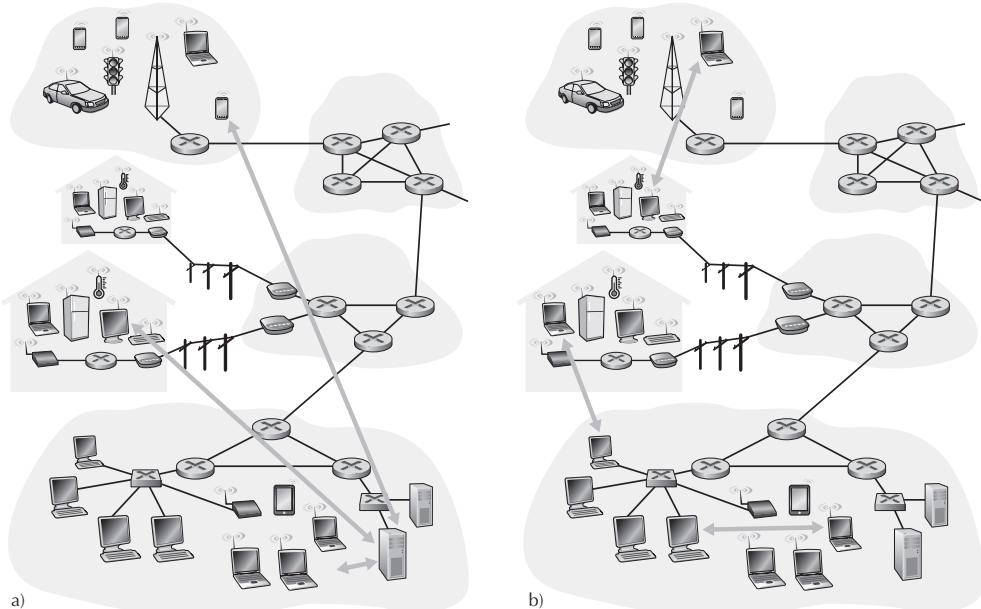
**Figura 2.1** La comunicazione in un'applicazione di rete ha luogo tra sistemi periferici a livello di applicazione.

vello 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 di rete è fissata e fornisce alle applicazioni uno specifico insieme di servizi; il suo compito è progettare l’**architettura dell’applicazione** 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 P2P.

Nell’architettura client-server 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 un web server, sempre attivo, risponde alle richieste dei browser in funzione sui client. Quando riceve una richiesta di un oggetto da parte di un client, il server 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).



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

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), il social networking (per esempio, Facebook, Instagram, Twitter e WeChat) utilizzano uno o più data center. Come discusso nel Paragrafo 1.3.3, Google dispone dai 50 ai 100 data center sparzi 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 arbitrarie di host, chiamati *peer* (ossia pari), collegati in modo intermittente. I peer non appartengono a un fornitore di servizi, ma sono computer fissi e portatili, controllati dagli utenti, che per la maggior parte 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 (per esempio, BitTorrent), sistemi per aumentare le prestazioni nel trasferimento di file (per esempio, Xunlei), la telefonia e la videoconferenza su Internet (per esempio, Skype). L'architettura P2P è illustrata nella Figura 2.2(b). Abbiamo accennato che alcune applicazioni presentano un'architettura ibrida, combinando sia elementi client-server sia P2P. Per esempio, per molte applicazioni di messaggistica istantanea, i server sono usati per tenere traccia degli indirizzi IP degli utenti, ma i messaggi tra utenti sono inviati direttamente tra gli host degli utenti, senza passare attraverso server intermedi.

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 sistemi terminali. 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 sistemi terminali comunicano scambiandosi **messaggi** attraverso la rete: il processo mittente crea e invia messaggi nella 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>

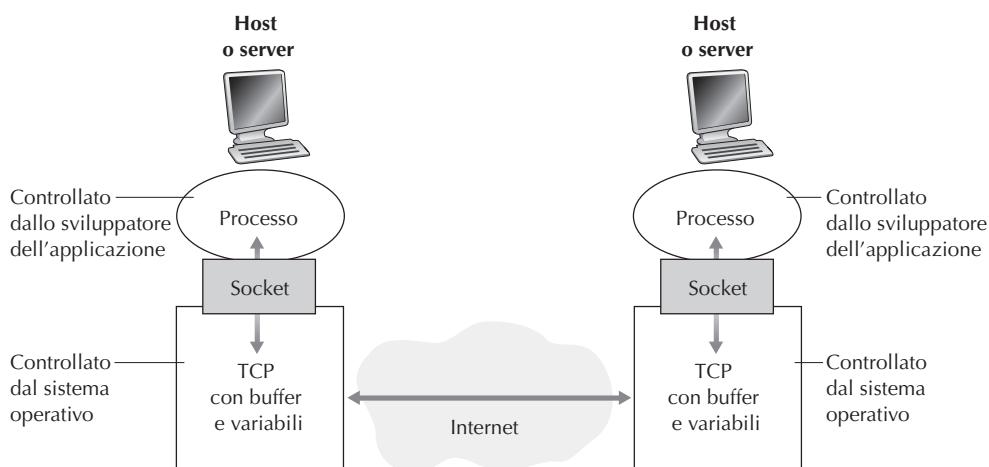
<sup>2</sup> Questa definizione si riferisce a una famiglia specifica di servizi: quelli fruiti in modalità pull; infatti, il client tira verso di sé i dati. Il lettore cerchi di non confondersi con i servizi fruiti in modalità push, dove il server spinge i dati verso il client, come nel caso dei servizi 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.).

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 che la socket rappresenta l’interfaccia di programmazione con cui le applicazioni di rete vengono costruite. Il



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

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, dovendo 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 **trasferimento 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 tollerano le perdite** (*loss-tolerant*): in particolare le applicazioni multimediali 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 deterioramento 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 percorso in rete, è il tasso 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 a quello stesso tasso. Se il protocollo a livello di trasporto non può fornire questo throughput, l'applicazione dovrà 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 ser-

vire poco o nulla. Le applicazioni che hanno requisiti di throughput vengono dette **applicazioni sensibili alla banda (bandwidth-sensitive)**. Molte delle odierni 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** 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. A questo riguardo si veda il Capitolo 9 on-line [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 destinazione, 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 on-line.

## 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

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

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

loro prime decisioni 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.** 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.
- **Servizio di trasferimento affidabile.** 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 conseguire 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.

**BOX 2.1** **TEORIA E PRATICA****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 **secure sockets layer (SSL)**. TCP, arricchito da SSL, non solo fa tutto ciò di cui è capace il TCP tradizionale, ma fornisce anche servizi critici di sicurezza tra processi, compresa la cifratura, il controllo dell'integrità dei dati e l'autenticazione end-to-end. Sottolineiamo che SSL 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 SSL, ha necessità di includere del codice SSL (library e classi esistenti molto ottimizzate) sia sul lato client sia sul lato server dell'applicazione. SSL 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 SSL, 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 SSL, che li decifra. Infine, SSL passa i dati in chiaro attraverso la sua socket SSL al processo ricevente. Tratteremo SSL in maggior dettaglio nel Capitolo 8 on-line.

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 questa 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

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 [RFC 2616]	TCP
Trasferimento file	FTP [RFC 959]	TCP
Multimedia streaming	HTTP (per esempio: YouTube)	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.

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 SSL 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. Approfondiremo questi “trucchi di progettazione” nel corso del Capitolo 9 on-line. 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 fornisce 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.

## 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 nell'RFC 2616. 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), browser (per esempio: Firefox e Microsoft Internet Explorer), 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 consideriamo l'applicazione di posta elettronica su Internet. Anch'essa presenta molte componenti tra cui: server di posta che ospitano le caselle degli utenti, client di posta (come Microsoft Outlook) che consentono agli utenti di leggere o creare messaggi, protocolli a livello di applicazione che definiscono le modalità di scambio dei messaggi tra server diversi, tra server e client e come interpretare le intestazioni dei messaggi. Di conseguenza, **SMTP** (*simple mail transfer protocol*) [RFC 5321], il principale protocollo a livello di applicazione per la posta elettronica, rappresenta solo una parte (sebbene consistente) dell'applicazione di posta elettronica.

## 2.1.6 Applicazioni di rete trattate in questo libro

Ogni giorno vengono sviluppate nuove applicazioni per Internet, sia di pubblico dominio, sia proprietarie. Così, piuttosto che trattarne molte in modo enciclopedico, 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. Prenderemo in esame le applicazioni multimediali, tra cui lo streaming video e la telefonia su IP, nel Capitolo 9 on-line.

## 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 estremamen-

<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.*).

te 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, le applet Java e molti altri elementi ci consentono di interagire con le pagine e con i siti web. Inoltre il Web fornisce una piattaforma a molte killer application, 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] e in [RFC 2616], 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 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.

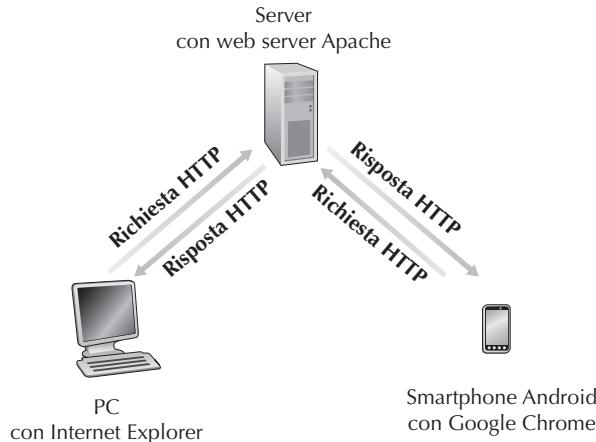
Una pagina web (*web page*), detta anche documento, è costituita da oggetti. Un oggetto è semplicemente un file (quale un file HTML, un’immagine JPEG, un’applet Java, una clip video e così via) indirizzabile tramite un URL. La maggioranza delle pagine web consiste di un **file HTML principale** e diversi 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 riferenzia 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 Firefox) 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 messaggi di risposta attraverso la pro-



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

pria 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.

## 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, esamiamo 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.

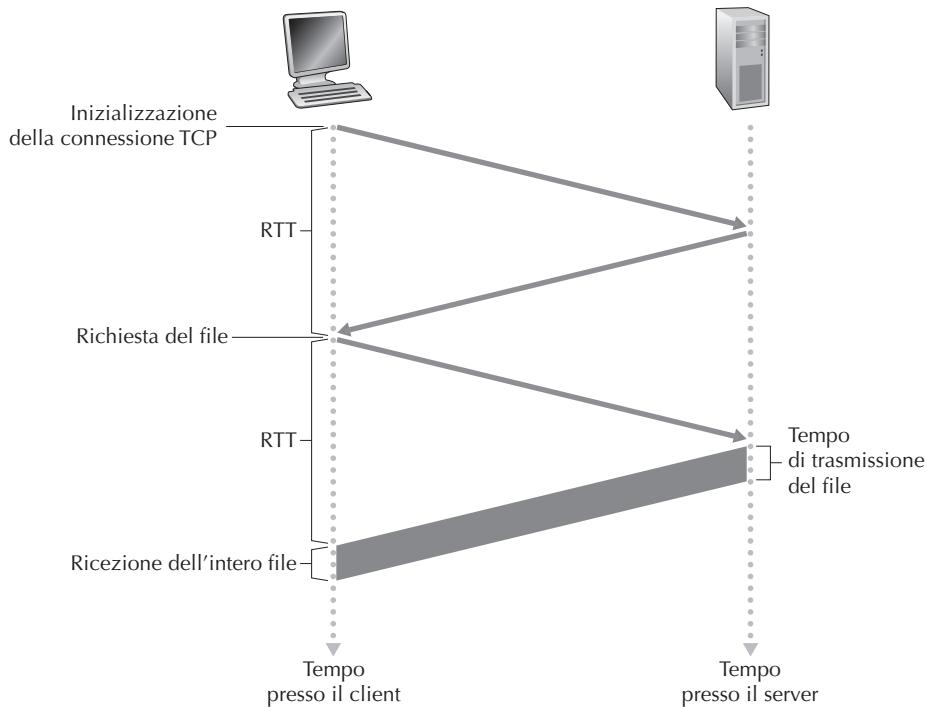
Quando il browser riceve la pagina web, la visualizza all'utente. Due browser diversi possono interpretare (e quindi mostrare all'utente) una stessa pagina web secondo modalità leggermente diverse. HTTP non ha niente a che vedere con l'interpretazione della pagina web da parte di un client. Le specifiche HTTP ([RFC 1945] e [RFC 2616]) 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.

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, e ciascuna di queste 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 piccolo 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



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

web server. Ciò comporta un **handshake** (letteralmente, “stretta di mano”) a tre vie (*three-way handshake*): il client invia un piccolo segmento TCP al server, 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, o 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. Più recentemente sulla base di HTTP 1.1 è stato sviluppato HTTP/2 [RFC 7540] che ha introdotto la possibilità di avere richieste e risposte multiple intercalate sulla stessa connessione e un meccanismo di priorità. 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 2616; 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*) e 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 auto esplicativa: nell'esempio, il browser, che implementa la versione HTTP/1.1, sta richiedendo l'oggetto `/somedir/page.html`.

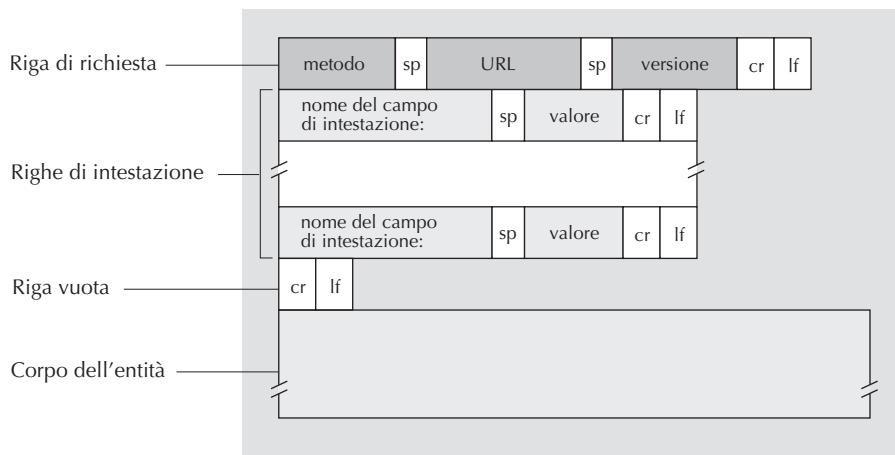
Consideriamo ora le righe di intestazione dell'esempio.

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.<sup>4</sup> 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` specifica il tipo di browser che sta effettuando la richiesta al server, in questo caso 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). Notiamo che questo segue da vicino l'esempio precedente. Potreste aver osservato, tuttavia, che dopo le linee di intestazione (e i caratteri di ritorno a capo e

---

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



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

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 usano necessariamente il metodo POST. Anzi, i form HTML 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 avrà 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.

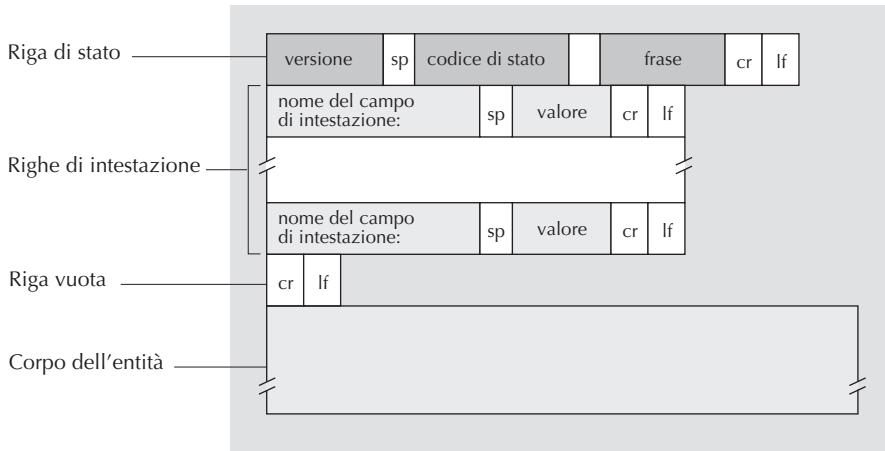
```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html
(data data data data data ...)
```

Analizzando in dettaglio questo messaggio di risposta, osserviamo tre sezioni: una **riga di stato** iniziale, sei **righe di intestazione** e il **corpo**. 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 (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 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.

---

<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.

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 (per esempio, un browser HTTP/1.0 non genera righe HTTP/1.1), 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, e tutto ciò influenza le righe dell'intestazione dei messaggi di risposta.

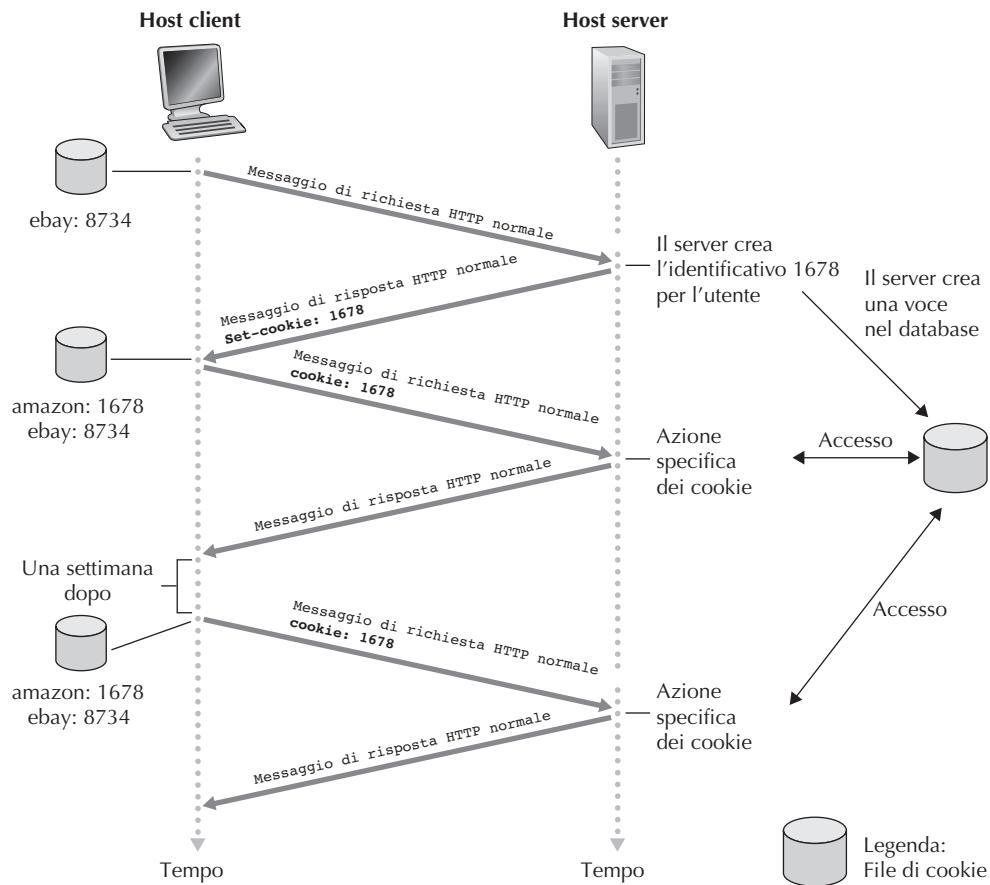
## 2.2.4 Interazione utente-server: i cookie

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`: 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.



**Figura 2.10** Memorizzazione dello stato dell'utente con i cookie.

stazione **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.

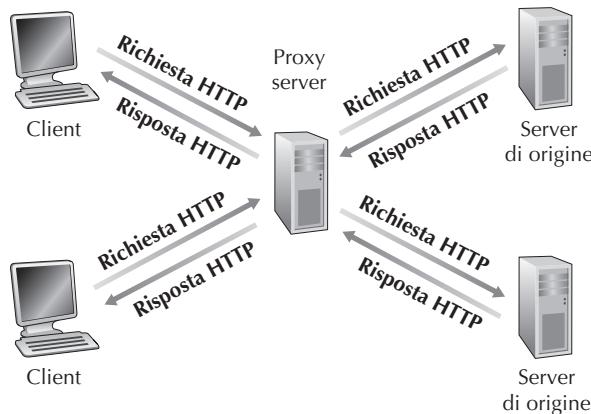
Da quanto precedentemente detto si evince che i cookie possono essere usati per identificare gli utenti. La prima volta che visita un sito, un utente può fornire un’identificazione (magari il suo nome). Successivamente il browser passa un’intestazione di cookie al server durante tutte le successive visite 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, come Hotmail, 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. Cookie Central [Cookie Central 2016] include molte informazioni sulle controversie relative ai cookie.

## 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. 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.



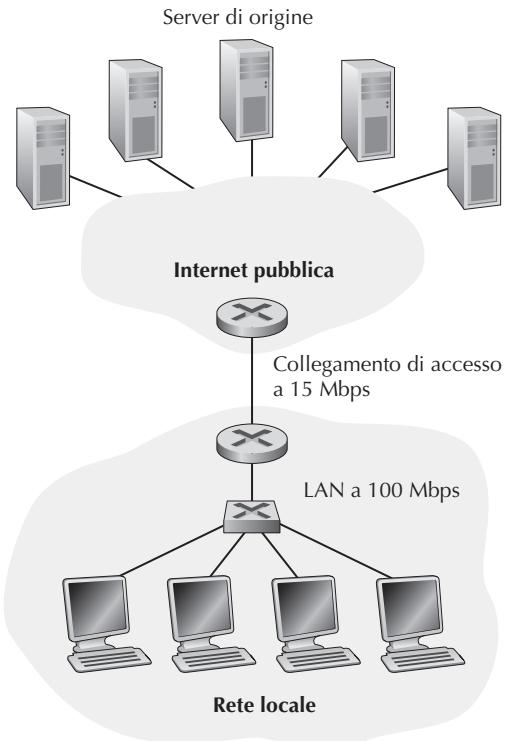
**Figura 2.11** Client che richiedono oggetti attraverso un proxy server.

3. Se, invece, la cache non dispone dell’oggetto, apre una connessione TCP verso il server di origine; ossia, nel nostro esempio, `www.someschool.edu`. Poi, 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 preconfigurare 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.



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

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 server 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 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

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

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.

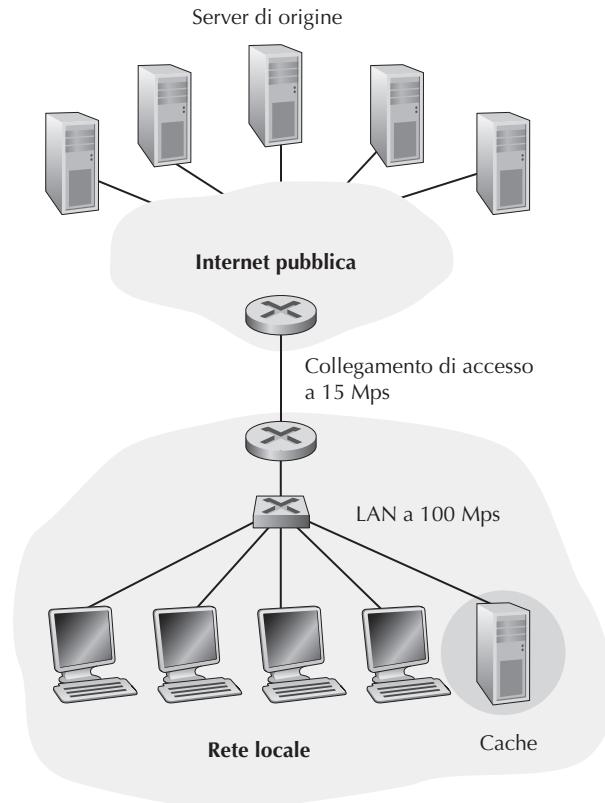
Una possibilità consiste nell'incremento della banda sul collegamento di accesso a Internet, per esempio da 15 Mbps a 100 Mbps. Ciò abbasserà l'intensità di traffico sul collegamento di accesso fino a 0,15, il che si traduce in ritardi trascurabili tra i due router. In questo caso, il tempo totale di 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, variano in pratica tra 0,2 e 0,7. 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 con-



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

divise (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)**. 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 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.

Si conclude qui la discussione del primo protocollo Internet che tratteremo, HTTP, di cui abbiamo visto il formato dei messaggi, le azioni che server e client intraprendono in risposta ai messaggi e alcuni elementi dell’infrastruttura dell’applicazione Web, quali cache e cookie, connessi al protocollo HTTP.

## 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.

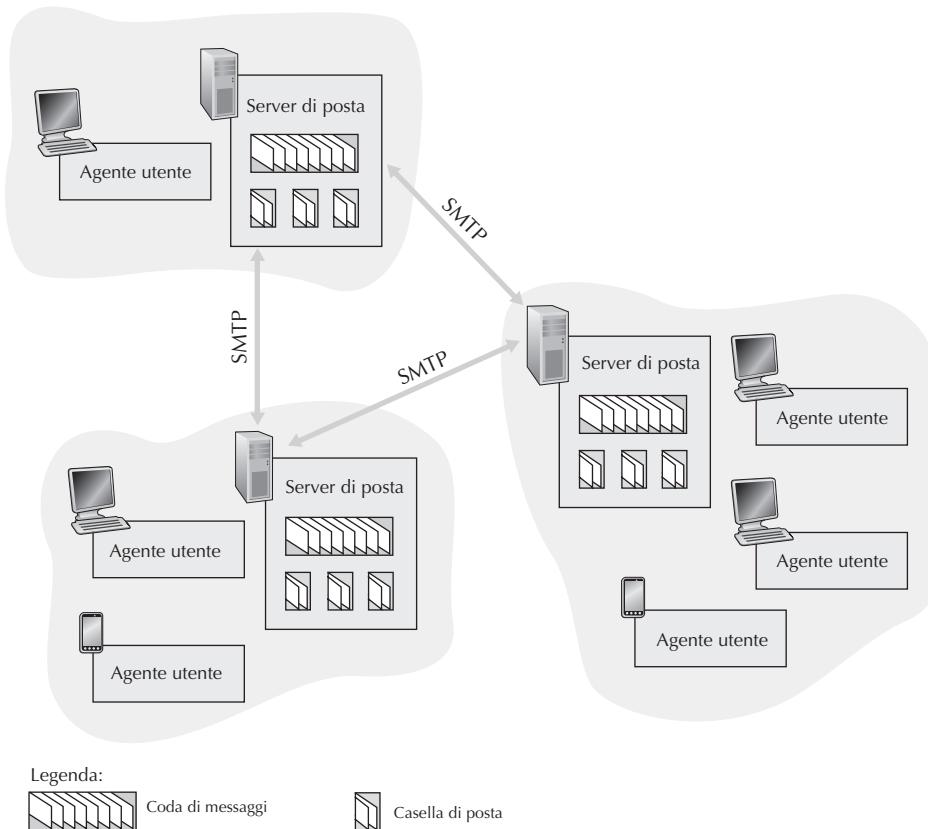
Come il servizio postale ordinario, l'e-mail rappresenta un mezzo di comunicazione asincrono: le persone inviano e leggono i messaggi nel momento per loro più opportuno, senza doversi coordinare con altri utenti. A differenza del servizio postale ordinario, però, la posta elettronica è veloce, facile 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 **server di posta** (o *mail server*) 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 e Apple Mail) 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 **casella di posta** (*mailbox*) 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. Per accedere ai messaggi della propria casella Bob 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** e cerca di trasferirla in un secondo momento. In genere i tentativi vengono effettuati ogni 30 minuti e, se dopo alcuni giorni non si ottiene successo, il server rimuove il messaggio e notifica la mancata consegna al mittente (Alice) con un messaggio di posta elettronica.

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



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

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

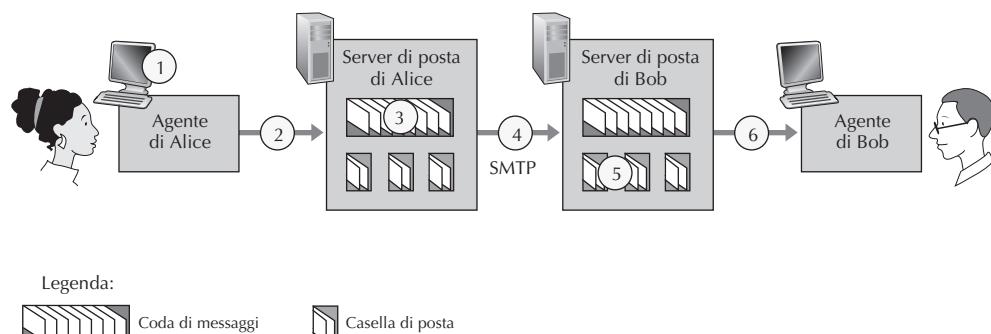
trasmissiva era scarsa e nessuno inviava per posta elettronica grandi allegati quali immagini, audio o video, ma oggi, in piena era multimediale, la restrizione all'ASCII a 7 bit è piuttosto penalizzante, in quanto richiede che i dati multimediali binari vengano codificati in ASCII prima di essere inviati 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



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

di Bob è spento, il messaggio rimane nel mail server di Alice e attende un nuovo tentativo. Il messaggio non viene posizionato 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 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 coman-

di: 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 Confronto con HTTP

Confrontiamo brevemente SMTP e HTTP. I due protocolli vengono utilizzati per trasferire file da un host a un altro. **HTTP trasferisce file** (spesso chiamati oggetti) da un **web server a un web client** (solitamente un browser). **SMTP trasferisce file** (ossia messaggi di posta elettronica) da un **mail server a un altro**. Durante il trasferimento, sia **HTTP persistente** sia **SMTP** utilizzano connessioni persistenti e quindi presentano **caratteristiche comuni**. Esistono però sostanziali differenze. Innanzitutto, **HTTP è principalmente un protocollo pull**: qualcuno carica informazioni su un web server e gli utenti usano HTTP per *attirarle* a sé (pull) dal server. In particolare, la connessione TCP viene iniziata dalla macchina che vuole ricevere il file. Al contrario, **SMTP è sostanzialmente un protocollo push**: il mail server di invio *spinge* (push) i file al mail server in ricezione. In particolare, la connessione TCP viene iniziata dall'host che vuole spedire il file.

Una seconda differenza, cui abbiamo già accennato, è che SMTP deve comporre l'intero messaggio (compreso il corpo) in ASCII a 7 bit. Anche se il messaggio contiene caratteri che non appartengono ad ASCII a 7 bit (per esempio, caratteri con accenti) o dati binari (come un'immagine), il messaggio deve essere comunque codificato in ASCII a 7 bit. HTTP non impone tale vincolo.

Un’altra importante differenza riguarda la gestione di un documento che contiene testo e immagini (insieme ad altri possibili tipi di media). Come abbiamo imparato nel Paragrafo 2.2, HTTP incapsula ogni oggetto nel proprio messaggio di risposta HTTP. La posta elettronica colloca tutti gli oggetti in un unico messaggio.

### 2.3.3 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.

### 2.3.4 Protocolli di accesso alla posta

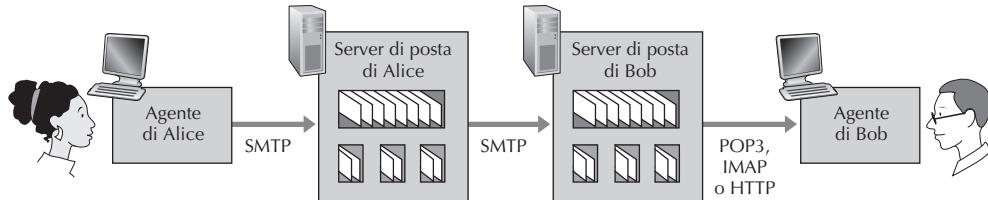
Quando SMTP consegna il messaggio di Alice al mail server destinatario, questo lo colloca nella casella di posta di Bob. Per tutta la trattazione abbiamo tacitamente ipotizzato che Bob legga la propria posta collegandosi all’host che svolge la funzione di server ed eseguendo un programma di lettura. Fino ai primi anni ’90 questo era il metodo standard. Attualmente, l’accesso alla posta elettronica utilizza un’architettura client-server: l’utente legge le e-mail con un client in esecuzione sul proprio sistema periferico (il PC dell’ufficio, un laptop o uno smartphone). Eseguendo un client di posta su un PC locale, gli utenti beneficiano di molteplici possibilità, tra cui la capacità di visualizzare messaggi multimediali e allegati.

Nell’ipotesi che Bob (il destinatario) esegua il proprio user agent sul suo PC locale, verrebbe naturale pensare alla collocazione di un mail server anch’esso sul PC locale. Con questo approccio, il mail server di Alice dialogherebbe direttamente con il PC di Bob. In ogni caso, esiste un problema collegato 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 PC 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 ilabbi in esecuzione uno user agent sul PC locale, ma acceda alla propria casella memorizzata su un mail server condiviso con altri utenti e sempre attivo. Questo server è generalmente gestito dall’ISP dell’utente (per esempio, un’università o un’azienda).

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 e tutto questo potrebbe essere conseguito da SMTP. 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. E nel caso in cui il proprio server di posta sia inattivo, Alice può lamentarsi con il proprio amministratore di sistema! L’RFC relativo a SMTP definisce i comandi SMTP per consegnare un messaggio attraverso più server SMTP.

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*. Il puzzle viene completato introducendo uno speciale protocollo di accesso alla posta, che trasferisce i messaggi dal mail server di Bob al suo PC locale. Attualmente esistono svariati protocolli del genere, tra cui *post office protocol – versione 3 (POP3)*, *Internet mail access protocol (IMAP)* e HTTP.

La Figura 2.16 fornisce un riassunto dei protocolli di posta su Internet: SMTP è usato per trasferire posta dal server del mittente a quello del destinatario ed è anche utilizzato per trasferire la posta dallo user agent al mail server del mittente. Per trasferire messaggi dal mail server allo user agent del destinatario viene impiegato un protocollo di accesso alla posta, quale POP3.



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

### POP3

POP3 è un protocollo di accesso alla posta estremamente semplice, definito in [RFC 1939], che è breve e di agevole lettura. Dato che il protocollo è tanto semplice, le sue funzionalità sono piuttosto limitate. POP3 entra in azione quando lo user agent (il client) apre una connessione TCP verso il mail server (il server) sulla porta 110. Quando la connessione TCP è stabilita, POP3 procede in tre fasi: autorizzazione, transazione e aggiornamento. Durante la prima fase (autorizzazione) lo user agent invia nome utente e password (in chiaro) per autenticare l'utente. Durante la seconda fase (transazione) lo user agent recupera i messaggi; inoltre, durante questa fase, può marcicare i messaggi per la cancellazione, rimuovere i marcatori di cancellazione e ottenere statistiche sulla posta. La fase di aggiornamento ha luogo dopo che il client ha inviato il comando `quit`, che conclude la sessione POP3; in questo istante, il server di posta rimuove i messaggi che sono stati marcati per la cancellazione.

In una transazione POP3 lo user agent invia comandi e il server reagisce a ogni comando con una tra due possibili risposte: `+OK` (talvolta seguito da dati dal server al client), usato dal server per indicare che il precedente comando andava bene; e `-ERR`, utilizzato dal server per indicare che qualcosa non ha funzionato nel precedente comando.

La fase di autorizzazione ha due principali comandi: `user <username>` e `pass <password>`. Per comprenderli meglio, vi suggeriamo di adoperare telnet direttamente con un server POP3 usando la porta 110 e inviare i comandi. Supponiamo che il vostro mail server si chiami `mailServer`. Vedremo qualcosa che assomiglia a:

```

telnet mailServer 110
+OK POP3 server ready
user bob
+OK
pass hungry
+OK user successfully logged on
  
```

Se sbagliate a scrivere un comando, il server POP3 risponde con un messaggio `-ERR`. Prendiamo ora in considerazione la fase di transazione. Uno user agent che usa POP3 può spesso essere configurato (dall'utente) per “scaricare e cancellare” o per “scaricare e mantenere”. La sequenza di comandi inviati da uno user agent POP3 dipende dal modo operativo scelto. Nella modalità “scarica e cancella” lo user agent manderà

i comandi `list`, `retr` e `dele`. Per esempio, supponiamo che l’utente abbia due messaggi nella propria casella di posta. Nel dialogo che segue, C: (client) è lo user agent e S: è il mail server.

La transazione sarà:

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: (bla bla ...
S: .....
S: .....bla)
S: .
C: dele 1
C: retr 2
S: (bla bla ...
S: .....
S: .....bla)
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

Lo user agent chiede innanzitutto al server di elencare la dimensione dei messaggi memorizzati, quindi recupera e cancella ogni messaggio dal server. Si noti che dopo la fase di autorizzazione, lo user agent ha utilizzato solo quattro comandi: `list`, `retr`, `dele` e `quit`. La sintassi è definita nell’RFC 1939. Una volta elaborato il comando `quit`, il server POP3 entra nella fase di aggiornamento e rimuove i messaggi 1 e 2 dalla casella di posta.

Un problema legato alla modalità “scarica e cancella” è che il destinatario, Bob, potrebbe voler accedere ai propri messaggi di posta da più macchine, per esempio dal PC del suo ufficio, dal computer di casa o dal portatile. La modalità “scarica e cancella” ripartisce i messaggi di posta di Bob sulle tre macchine; in particolare, se Bob legge dapprima un messaggio sul PC del suo ufficio, non sarà in grado di rileggerlo dal portatile. In modalità “scarica e mantieni”, lo user agent lascia i messaggi sul server di posta dopo averli scaricati. In questo caso, Bob può rileggere i messaggi da diverse macchine, in differenti momenti.

Durante una sessione tra uno user agent e il mail server, il server POP3 mantiene alcune informazioni di stato; in particolare, tiene traccia dei messaggi dell’utente marcati come cancellati. Tuttavia, il server POP3 non trasporta informazione di stato tra sessioni POP3. Questa mancanza di informazione di stato persistente tra sessioni semplifica di molto l’implementazione.

## IMAP

Con l'accesso tramite POP3, dopo aver scaricato i messaggi sulla macchina locale, Bob può creare cartelle nelle quali includere i messaggi scaricati. Può poi spostarli tra cartelle, effettuare ricerche per nome del mittente o per oggetto, oppure decidere di cancellare i messaggi. Questo paradigma di cartelle e messaggi sulla macchina locale pone tuttavia problemi agli utenti mobili che preferirebbero mantenere una gerarchia di cartelle su un server remoto cui accedere da differenti calcolatori. Ciò non è possibile con POP3, dato che questo protocollo non fornisce all'utente alcuna procedura per creare cartelle remote e assegnare loro i messaggi.

Per risolvere questi e altri problemi è stato messo a punto il protocollo IMAP, definito in [RFC 3501]. Anche IMAP è un protocollo di accesso alla posta, ma presenta maggiori potenzialità rispetto a POP3 ed è quindi assai più complesso. Di conseguenza risultano molto più complesse anche le implementazioni del lato client e server.

Un server IMAP associa ogni messaggio arrivato al server a una cartella. I messaggi in arrivo sono associati alla cartella INBOX del destinatario. Quest'ultimo può poi spostare il messaggio in una nuova cartella creata dall'utente, leggerlo, cancellarlo e così via. Il protocollo IMAP fornisce comandi per consentire agli utenti di creare cartelle e spostare i messaggi da una cartella a un'altra. Fornisce anche comandi che consentono agli utenti di effettuare ricerche nelle cartelle remote sulla base di criteri specifici. Si noti che, a differenza di POP3, i server IMAP conservano informazioni di stato sull'utente da una sessione all'altra: per esempio, i nomi delle cartelle e l'associazione tra i messaggi e le cartelle.

Un'altra caratteristica importante di IMAP è la presenza di comandi che permettono agli user agent di ottenere singole parti dei messaggi. Per esempio, uno user agent può tenere solo l'intestazione o una parte di un messaggio composto da più elementi. Questa caratteristica è utile quando si dispone di una connessione con limitata ampiezza di banda tra lo user agent e il proprio mail server. È il caso, per esempio, di un collegamento con modem a bassa velocità. In caso di connessione lenta, l'utente potrebbe non voler scaricare tutti i messaggi presenti nella sua casella, evitando in particolare i messaggi lunghi che potrebbero contenere audio o videoclip.

## Posta basata sul Web

È oggi in costante crescita il numero di utenti che inviano posta e accedono alle proprie e-mail tramite un browser web. Hotmail ha introdotto l'accesso basato sul Web a metà degli anni '90; ora la posta elettronica via Web viene fornita anche da Google e Yahoo! così come da ogni grande università e azienda. Grazie a tale servizio, lo user agent è un semplice browser web e l'utente comunica con la propria casella remota via HTTP. Quando un destinatario, quale Bob, vuole accedere alla propria casella, il messaggio e-mail viene spedito dal server di posta al browser di Bob usando il protocollo HTTP anziché POP3 o IMAP. Quando un mittente, quale Alice, vuole inviare un messaggio di posta elettronica, quest'ultimo viene spedito dal suo browser al suo server di posta su HTTP anziché su SMTP. Il server di posta di Alice, in ogni caso, manda ancora i messaggi e li riceve utilizzando SMTP.

## 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. Potreste immaginare un dialogo del genere: "Ciao. Il mio nome è 132-67-9875. Ti presento mio marito, 178-87-1146."

Proprio come le persone, anche gli host Internet possono essere identificati in vari modi. I nomi degli host (*hostname*), quali `www.facebook.com`, `www.google.com` e `gaia.cs.umass.edu`, risultano abbastanza appropriati per l'uomo, ma forniscono 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 i nomi degli host 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 2016]. 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

---

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

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>

- **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 [Mockapetris 2005].

#### 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.

<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.).

## 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.

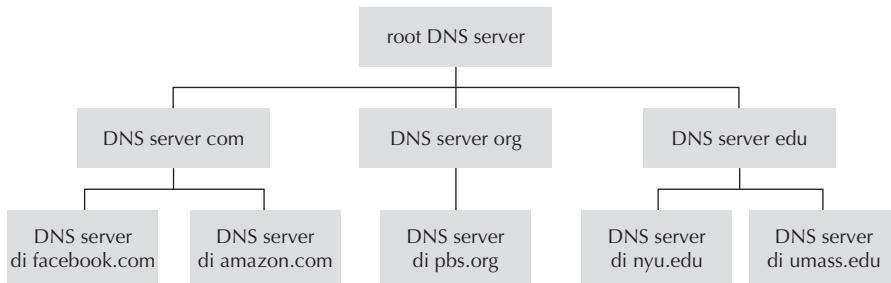
Supponiamo che una certa applicazione (quale per esempio un browser web o un programma per la lettura della posta) in esecuzione sull'host di un utente abbia necessità di tradurre un hostname in un indirizzo IP. L'applicazione invocherà il lato client del DNS, specificando l'hostname da tradurre. Su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



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

### 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 dapprima contatta uno dei root server, che gli restituisce uno o più indirizzi IP relativi al server TLD 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 server TLD, 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 400 root server, dislocati in tutto il mondo. La dislocazione dei root server è indicata nella Figura 2.18; il loro elenco insieme all’indirizzo IP e all’ente che gestisce ognuno dei root server è disponibile tramite [Root-servers 2016]. Questi root server sono gestiti da 13 diverse organizzazioni. I root server forniscono gli indirizzi IP dei server TLD.
- **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 relativi ai vari paesi, come `.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) e Educause amministra quelli per il dominio `.edu`. Per una lista dei server TLD si veda [TLD list 2016]. I server TLD 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

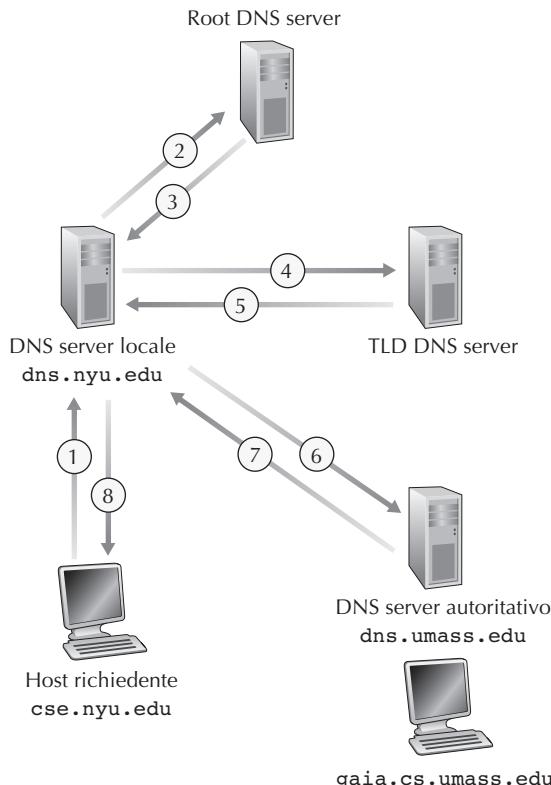


**Figura 2.18** Root DNS server nel 2016.

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 maggior dettaglio tra breve.

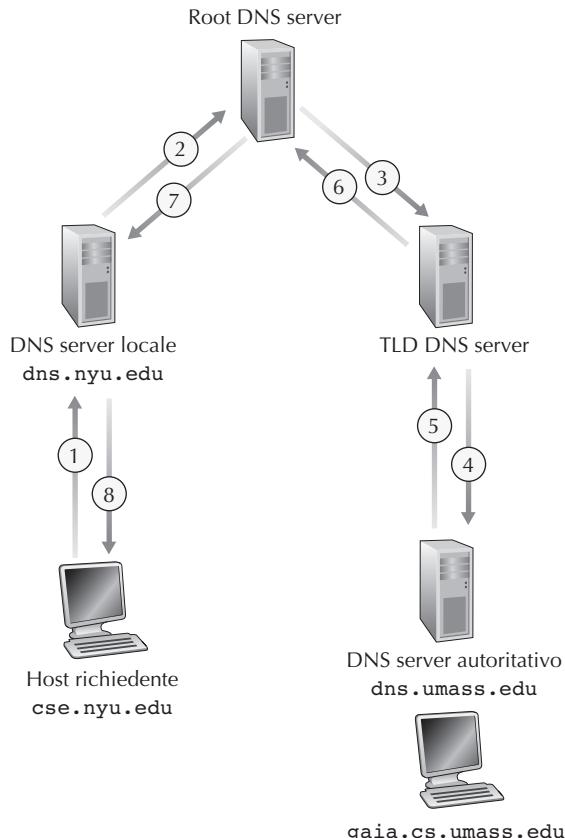
Consideriamo un semplice esempio. Supponiamo che l’host `cse.nyu.edu` voglia l’indirizzo IP di `gaia.cs.umass.edu`. Supponiamo, inoltre, che il DNS server locale per `cse.nyu.edu` sia `dns.nyu.edu`, mentre un server autoritativo per `gaia.cs.umass.edu` sia `dns.umass.edu`. Come mostrato nella Figura 2.19, l’host `cse.nyu.edu` dapprima invia un messaggio di richiesta DNS al proprio server locale



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

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.

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



**Figura 2.20** Query DNS ricorsive.

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.

## 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’indirizzo 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>

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).

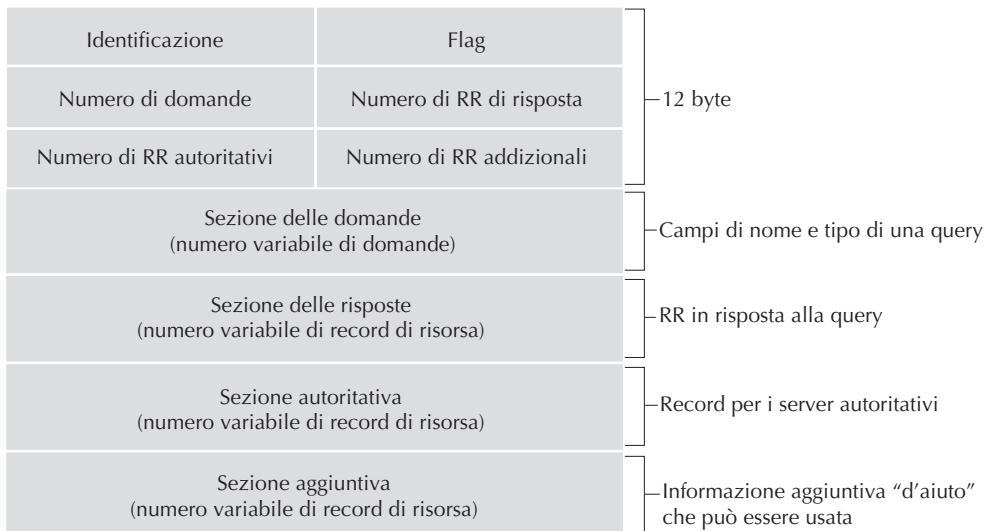
## Messaggi DNS

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*, che a sua volta contiene un certo numero di campi. Il primo è un numero di 16 bit che identifica la richie-

<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.).



**Figura 2.21** Formato dei messaggi DNS.

sta. 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 di **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 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, aprite 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'unica società, la Network Solutions, aveva il monopolio sulla registrazione dei nomi di dominio terminanti in **.com**, **.net** e **.org**. Ora però esistono molti registrar concorrenti, 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 **networkutopia.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.networkutopia.com**, **dns2.networkutopia.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 **networkutopia.com** il registrar inserirebbe nel sistema DNS i due seguenti record di risorsa:

(**networkutopia.com**, **dns1.networkutopia.com**, NS)  
(**dns1.networkutopia.com**, **212.212.212.1**, A)

Dovrete inoltre accertarvi che il record di risorsa di tipo A per il vostro web server [www.networkutopia.com](http://www.networkutopia.com) e che il record di risorsa di tipo MX per il vostro server di posta [mail.networkutopia.com](mailto:mail.networkutopia.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.networkutopia.com](http://www.networkutopia.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.networkutopia.com](http://www.networkutopia.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!

## 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 controllati dagli utenti.

In questo paragrafo esamineremo un'applicazione particolarmente adatte 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, un file musicale MP3 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

**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, portando con sé molte altre applicazioni Internet?

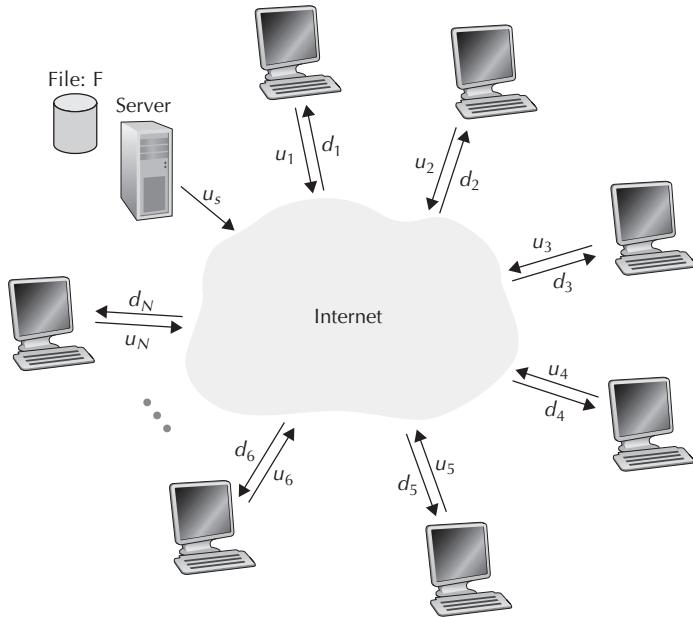
Il primo attacco che viene in mente è quello DDoS con flooding di banda (si veda il Paragrafo 1.6) contro i DNS server. Per esempio, un attaccante può tentare di inondare di pacchetti ciascun root server, in modo che molte delle richieste DNS legittime rimangano senza risposta. Un tipo di attacco DDoS su così larga scala a server DNS 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 server DNS 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.

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. Questi attacchi, tuttavia, sono difficili da realizzare, in quanto richiedono l’intercettazione dei pacchetti o l’inibizione dei server [Skoudis 2006].

In sintesi DNS si è dimostrato sorprendentemente robusto contro gli attacchi; fino a oggi non c’è stato un attacco che abbia avuto successo nell’ostacolare il servizio. Ci sono stati, invece, attacchi andati a buon fine con il metodo della riflessione, ma questi sono affrontabili tramite un’appropriata configurazione dei DNS server.

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 2016, 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



**Figura 2.22** Un problema illustrativo di distribuzione file.

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 fissato 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 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** 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’analoga 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 re-distribuire 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 dipende 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:

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

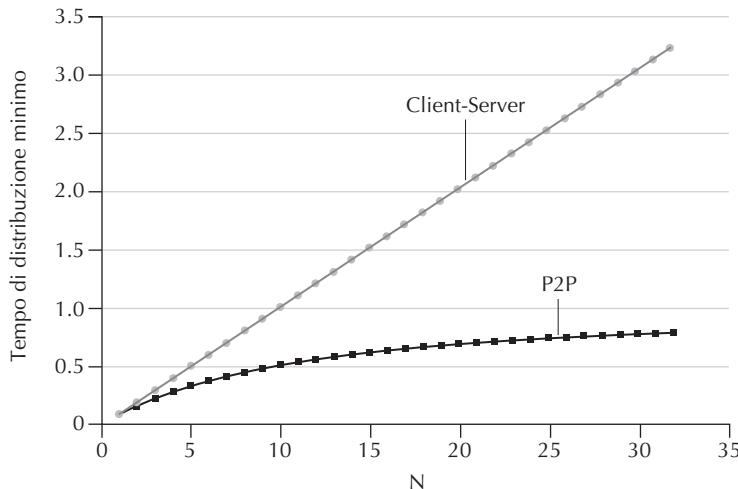
Mettendo assieme queste tre osservazioni, otteniamo il tempo di distribuzione minimo per l'architettura P2P, indicato con  $D_{\text{P2P}}$ :

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

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 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,



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

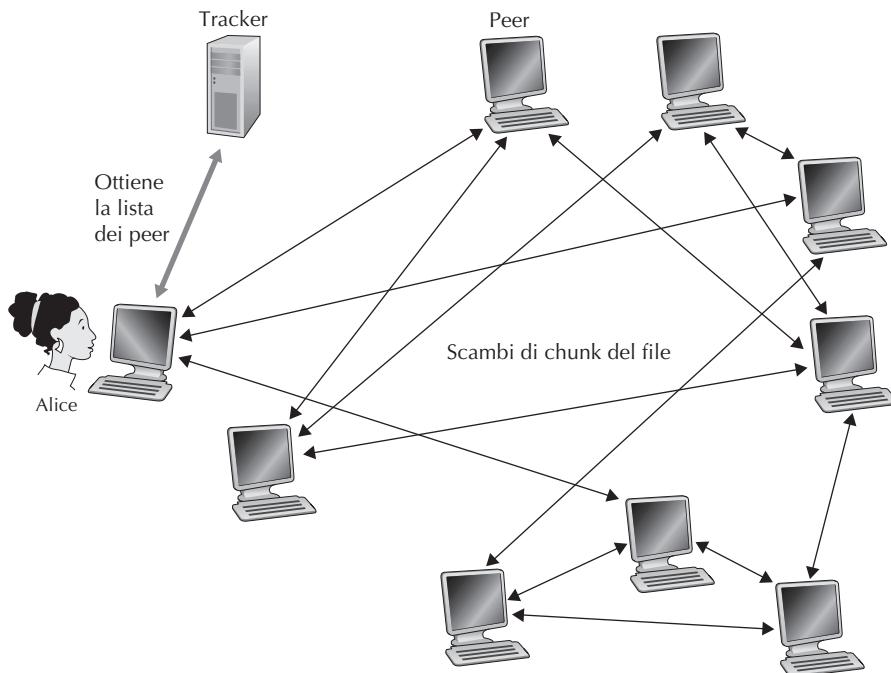
ma è anche minore di un'ora per qualsiasi numero di peer  $N$ . Quindi, le applicazioni con architettura P2P possono essere scalabili e la scalabilità è una diretta conseguenza del fatto che i peer re-distribuiscono i bit oltre che a scaricarli.

### 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. Quando un peer 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 sottinsieme 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**. 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 sottinsieme di peer (diciamo 50) dall'insieme dei peer che stanno partecipando a quel torrent, e invia l'indirizzo



**Figura 2.24** Distribuzione di file con BitTorrent.

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.

In un certo 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, cer-

cando, 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 velocità 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”). 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 è 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 2007], 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 sarebbero stati dei “freeriders”, avrebbero 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

Lo streaming di video registrati rappresenta attualmente la maggior parte del traffico di rete negli ISP residenziali del Nord America. In particolare, i servizi di Netflix e YouTube hanno consumato da soli un enorme 37% e 16%, rispettivamente, del traffico ISP residenziale nel 2015 [Sandvine 2015]. In questo paragrafo vi forniremo una panoramica di come i servizi di streaming video siano implementati nella Internet di oggi. Vedremo che sono implementati utilizzando protocolli a livello di applicazione e server che agiscono in un certo senso come una cache. Nel Capitolo 9 on-line, dedicato alle reti multimediali, esamineremo ulteriormente i video e gli altri servizi multimediali di Internet.

### 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 YouTube (Google), Netflix, Amazon e Youku.

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 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. 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.

Forse la caratteristica più saliente del video è l'elevato tasso 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 3 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 la stessa versione codificata del video, nonostante abbiano disponibile 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 (DASH, dynamic adaptive streaming over HTTP)**. 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. Comcast, per esempio, ha installato un sistema adattativo di streaming in cui ogni file sorgente di video è codificato in 8-10 formati MPEG-4 diversi.

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, o 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 il client 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, sceglierà una versione ad alto bit rate per il blocco successivo; al contrario, se ha pochi dati nel buffer e ban-

da bassa, sceglierà 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

Oggigiorno 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 è 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 del tasso 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 verrà 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 interrompe, l'azienda non sarà 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 ad esempio, 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** (letteralmente, “entrare in profondità”). Una filosofia, proposta da Akamai, è quella di entrare profondamente 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 circa 1700 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** (letteralmente, “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 ogni cluster in un luogo vicino ai PoP (Paragrafo 1.3) di molti ISP di livello 1. 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 in 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.

### 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).
3. Il server locale DNS (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](http://video.netcinema.com). Il DNS server autoritativo per NetCinema per passare l’interrogazione DNS a KingCDN, invece di restituire un indirizzo IP, restituisce

**BOX 2.4** **TEORIA E PRATICA****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.

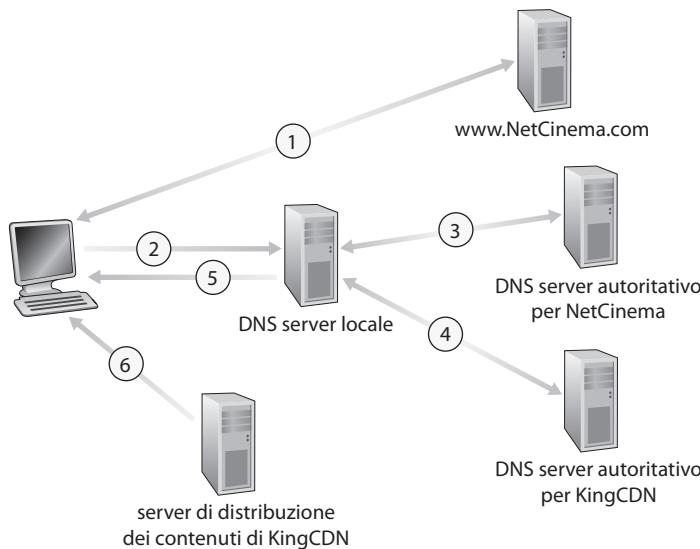
- Quattordici “mega data center” di cui otto negli Stati Uniti, quattro in Europa e due in Asia [Google Locations 2016], ognuno dei quali contiene circa 100.000 server. Sono deputati a offrire contenuti dinamici e spesso personalizzati; tra questi, i risultati delle ricerche e i messaggi di e-mail.
- Circa 50 cluster dislocati in IXP di tutto il mondo, ognuno dei quali con 100-500 server [Adhikari 2011a]. Questi sono deputati a offrire contenuti statici, tra cui i video YouTube [Adhikari 2011a].
- 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, inoltra 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.

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 LDSN 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. Lo 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 quel indirizzo IP e gli invia una richiesta GET per il video. Nel caso venga impiegato DASH, il server innanzitutto invierà al client il file manifesto con una lista di URL, uno per ogni versione del video, e il client selezionerà in modo dinamico i blocchi da versioni differenti.



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

### Strategie di selezione dei cluster

Il cuore dell’installazione di una CDN è la **strategia di selezione del cluster**: 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 naturali, 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 2016] e Max-Mind [MaxMind 2016]), 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** delle prestazioni di ritardo e perdita tra i loro cluster e i loro client; per esempio, facendo sondare pe-

riodicamente 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 un certo numero di 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. Quando questi due programmi 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, allora i due programmi saranno in grado di interagire. Infatti, la maggior parte delle odierne applicazioni di rete implica la comunicazione tra programmi client e server creati da sviluppatori indipendenti. Si consideri, per esempio, un browser Firefox 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 client-server. 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. Quando un programma client o server implementa un proto-

collo 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. Mostriremo 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 il sito web 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; Kurose 1996].

### 2.7.1 Programmazione delle socket con UDP

In questo paragrafo scriveremo dei semplici programmi client-server che usano UDP, nel successivo useremo 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, chia-

mato **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 riga per riga della copia di programmi client-server. 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.

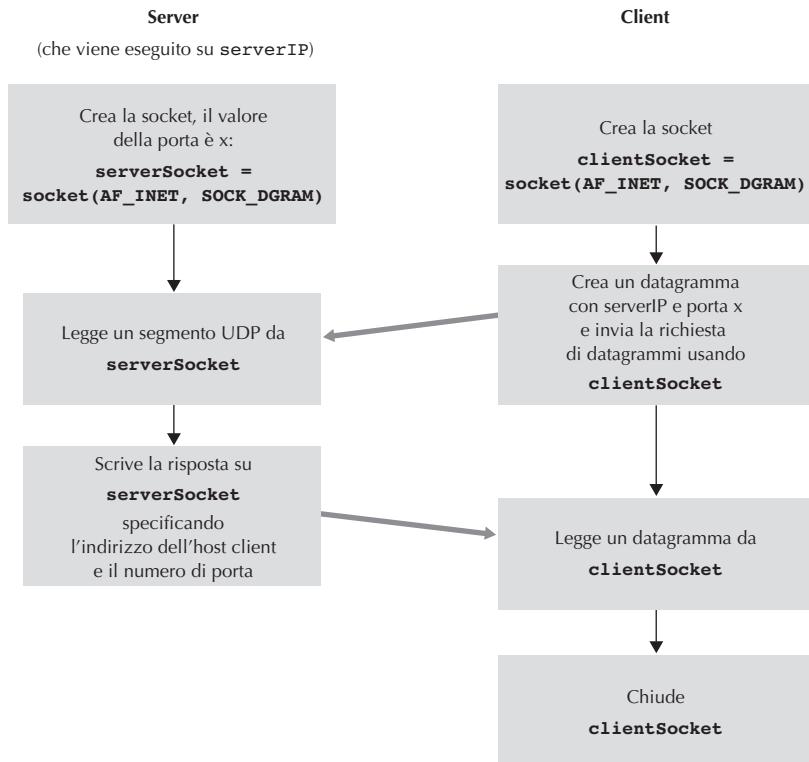
### **UDPClient.py**

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 = raw_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 *
```



**Figura 2.26** Applicazione client-server con UDP.

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 = raw_input('Frase in minuscolo:')
```

In Python `raw_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

---

<sup>9</sup> In realtà la cosa funziona anche se i due programmi vengono eseguiti sullo stesso host in finestre diverse (*N.d.R.*).

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.

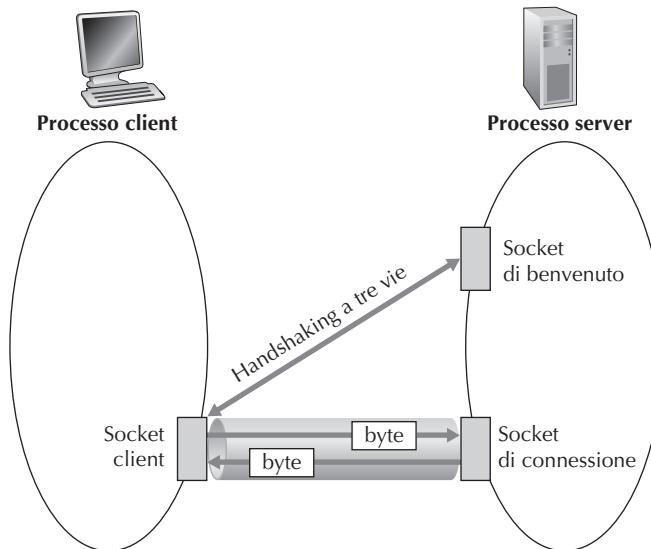
## 2.7.2 Programmazione delle socket con TCP

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 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 `Server-Socket`. La socket appena creata dedicata al client che fa la connessione sarà chiamata `connectionSocket`. Spesso chi affronta per la prima volta le socket TCP



**Figura 2.27** Un processo server TCP ha due socket.

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.

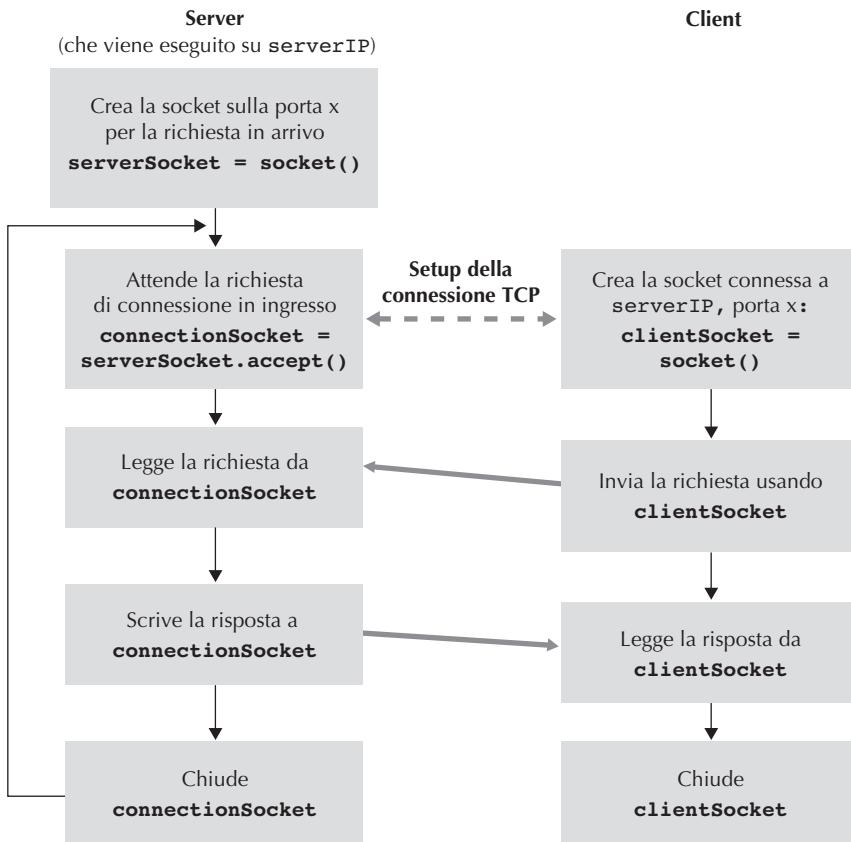
Dal punto di vista dell'applicazione, la connessione TCP è un condotto virtuale diretto tra la socket del client e la socket di connessione del server. Come mostrato nella Figura 2.27, il processo client può inviare quanti byte vuole nella sua socket e TCP garantisce che il processo server (attraverso 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))
```



**Figura 2.28** Applicazione client-server con TCP.

```

sentence = raw_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 = raw_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'
whileTrue:
    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 arrivarcì 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 analisi sulla 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, 4 e 9. 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, POP3 e DNS. Abbiamo studiato questi importanti protocolli a livello di applicazione e le loro corrispettive applicazioni (Web, trasferimento file, e-mail e DNS). Abbiamo, inoltre, studiato l’architettura P2P e come questa sia usata in molte applicazioni. 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, POP3 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). In ogni caso, 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?

## CAPITOLO

# 3

## 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 trasmissiva 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** tra processi applicativi di host differenti.

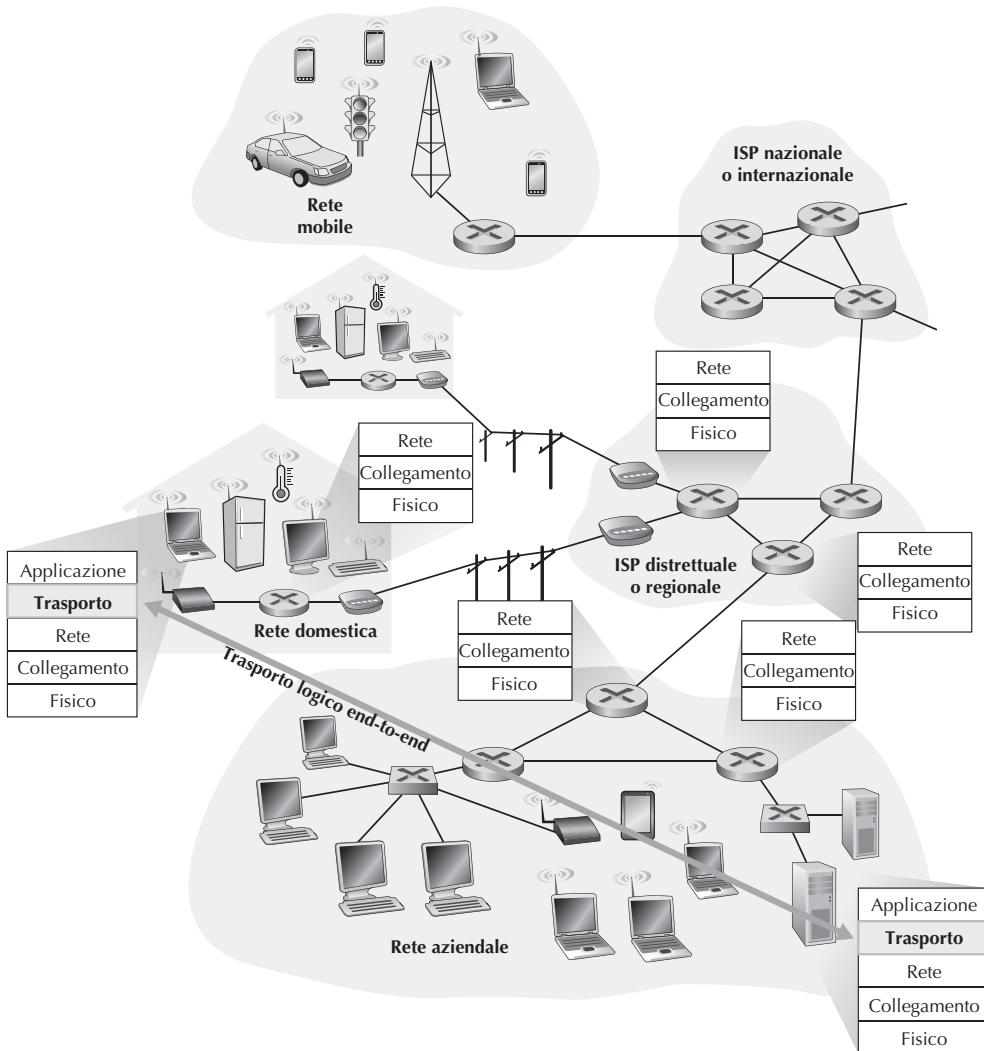
Per **comunicazione logica** si intende, dal punto di vista dell'applicazione, che tutto proceda come se gli host che eseguono i *processi* fossero direttamente connessi; in realtà, gli host si possono trovare agli antipodi del pianeta, connessi da numerosi router e da svariati tipi di collegamenti. 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 distri-



**Figura 3.1** Il livello di trasporto fornisce comunicazione logica anziché fisica tra processi applicativi.

buire la posta. Tutte le settimane Anna raccoglie la posta dai suoi fratelli e sorelle e provvede a imbucarla. Anna 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, allora 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 on-line, quando tratteremo la sicurezza di rete), un protocollo di trasporto può usare la crittografia 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, si usa 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** (letteralmente, “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**. 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-hosta consegna process-to-process, viene detto **multiplexing e demultiplexing a livello di trasporto** 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).

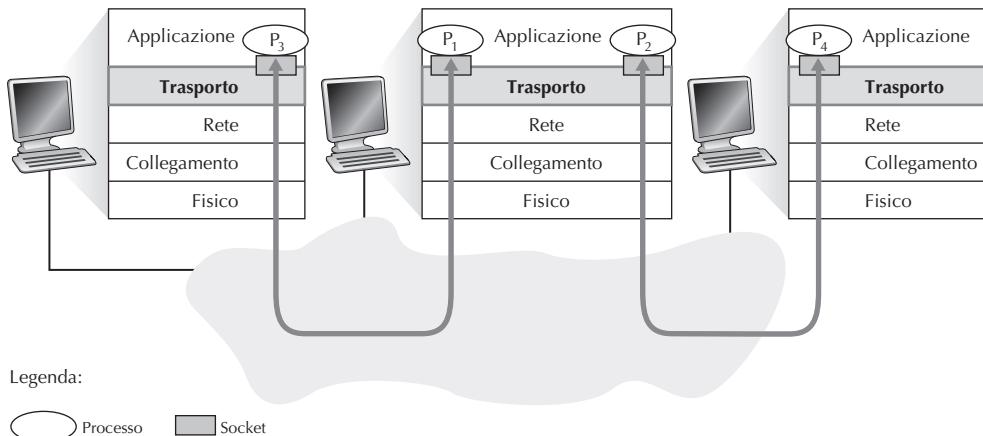
TCP, d'altra parte, offre alle applicazioni diversi servizi aggiuntivi. Innanzitutto, fornisce un **trasferimento dati affidabile**. Grazie al controllo di flusso, ai numeri di sequenza, agli acknowledgment e ai timer, tecniche che esploreremo in dettaglio in questo capitolo, assicura che i dati vengano 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**, 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 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 3.8). L'approccio adottato da 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 mette 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

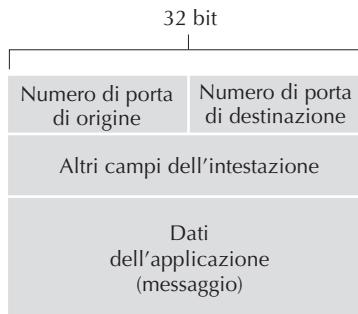


**Figura 3.2** Multiplexing e demultiplexing a livello di trasporto.

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 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  $P_1$  sia per  $P_2$ ; 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.



**Figura 3.3** I campi del numero di porta di origine e di destinazione nei segmenti a livello di trasporto.

Per mostrare il compito del multiplexing e del demultiplexing richiamiamo l'analogia del paragrafo precedente. Anna effettua un'operazione di multiplexing quando raccolgono 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.

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** e il **campo del numero di porta di destinazione**. 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()**:

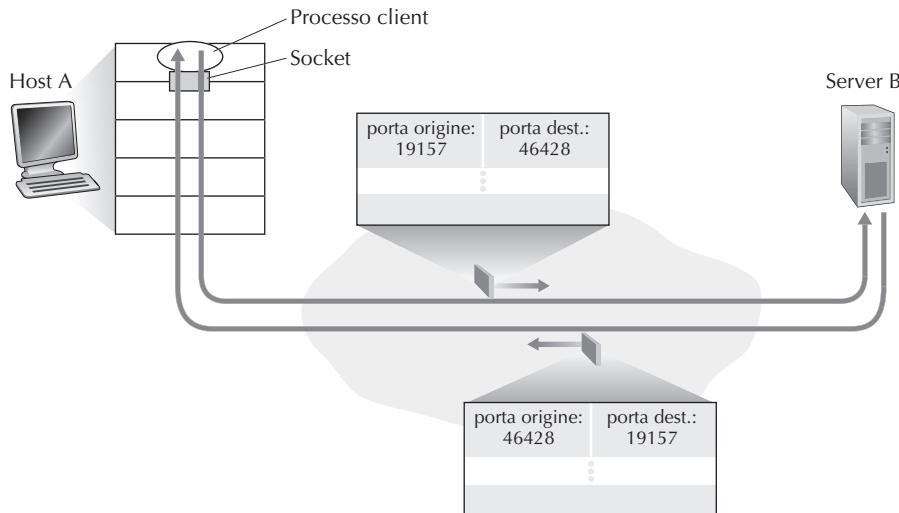
```
clientSocket.bind(('', 19157))
```

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 automatica (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, se due segmenti UDP presentano diversi indirizzi IP e/o diversi numeri di porta di origine, ma hanno lo stesso indirizzo IP e lo stesso numero di porta di destinazione, saranno 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 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.



**Figura 3.4** Inversione dei numeri di porta di origine e di destinazione.

### 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 2.29) 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.

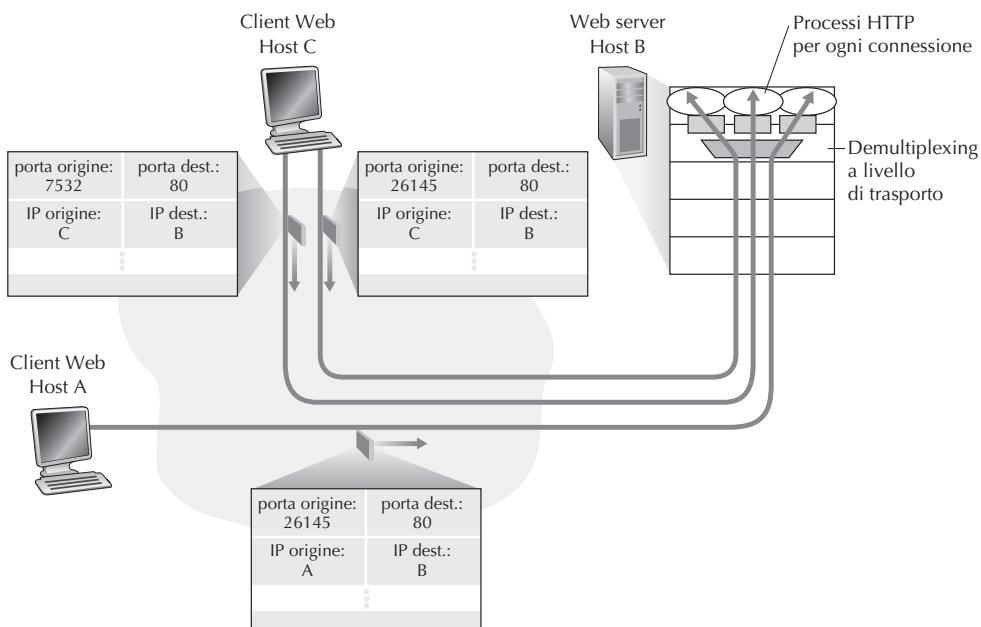
- Il sistema operativo dell'host che esegue il processo server, quando riceve il segmento con la richiesta di connessione con porta di destinazione 12000, localizza il processo server in attesa di accettare connessioni sulla porta 12000. Il processo server crea quindi una nuova connessione:

```
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.

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



**Figura 3.5** Due client che usano lo stesso numero di porta di destinazione (80) per comunicare con la stessa applicazione sul web server.

**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 SQL server 2000 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.

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 segmenti 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.

Se client e server usano HTTP persistente, allora 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 creata 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*.

DNS è un tipico esempio di protocollo a livello applicativo che utilizza UDP. Quando l'applicazione DNS in un host vuole effettuare una query, 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'indirizzamento 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 fine 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 trasmissione 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'affi-

fidabilità 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, [Iyengar 2015]), utilizzato nel browser Chrome di Google, utilizza UDP come protocollo di trasporto e implementa l'affidabilità in un protocollo a livello di applicazione.

- *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 della congestione e parametri sul numero di sequenza e di acknowledgment. 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.

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, il Web e il trasferimento di file utilizzano TCP in quanto richiedono un servizio di trasferimento dati affidabile. Ciò nondimeno, molte applicazioni importanti scelgono UDP; per esempio UDP viene inoltre 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.

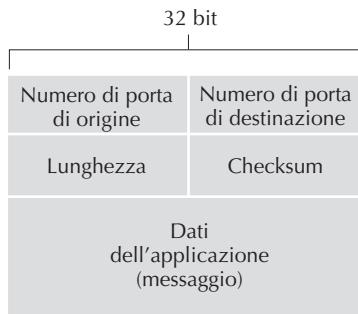
Come illustrato nella Figura 3.6, sia UDP che TCP sono oggi utilizzati per le applicazioni multimediali, quali la telefonia su Internet, la videoconferenza in tempo reale e lo streaming audio e video; tratteremo queste applicazioni nel Capitolo 9 on-line. Facciamo cenno, per ora, solo al fatto che queste 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. Tuttavia, TCP è sempre più utilizzato per il trasporto di dati multimediali in streaming. Quando il tasso di perdita dei pacchetti è basso e, dato che alcune istituzioni bloccano il traffico UDP per ragioni di sicurezza (si veda il Capitolo 8 on-line), TCP diventa sempre più allettante come protocollo per il trasporto dello streaming multimediale.

Applicazione	Protocollo a livello di applicazione	Protocollo di trasporto sottostante
Posta elettronica	SMTP	TCP
Accesso a terminali remoti	Telnet	TCP
Web	HTTP	TCP
Trasferimento file	FTP	TCP
Server di file remoti	NFS	generalmente UDP
Dati multimediali in streaming	generalmente proprietario	UDP o 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.

Sebbene costituisca oggigiorno una prassi comune, l’uso di UDP per le applicazioni multimediali è un argomento controverso. 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 destinazione. 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 trasmissiva 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: un problema decisamente serio [Floyd 1999]. 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, [Iyengar 2015]), utilizzato nel browser Chrome di Google, 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 trasmissiva imposti dai meccanismi di controllo della congestione di TCP.



**Figura 3.7** Struttura dei segmenti UDP.

### 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 perché 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
-----
1000111100001100
  
```

La somma delle prime due è:

$$\begin{array}{r} 0110011001100000 \\ 0101010101010101 \\ \hline 1011101110110101 \end{array}$$

Sommiamo la terza parola al risultato precedente otteniamo:

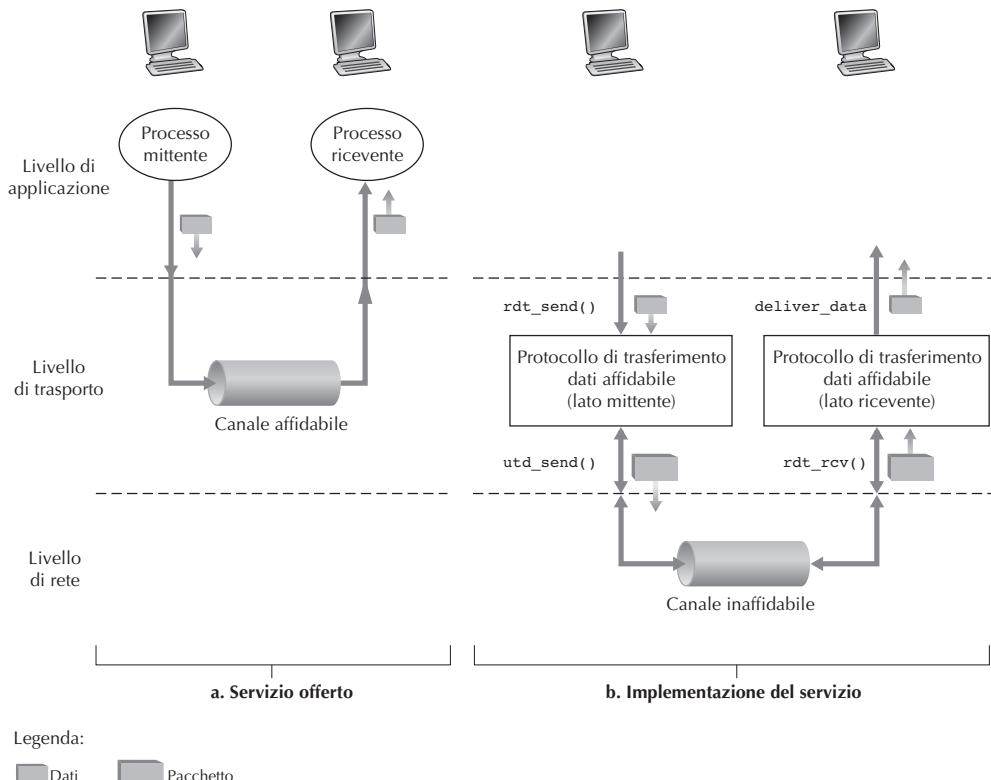
$$\begin{array}{r} 1011101110110101 \\ 1000111100001100 \\ \hline 0100101011000010 \end{array}$$

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.



**Figura 3.8** Trasferimento dati affidabile: modello di servizio e implementazione.

## 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 offerto 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** è l'implementazione di questa astrazione del servizio. Ciò è reso difficile 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 i nostri scopi, tuttavia, possiamo vedere questo livello inferiore semplicemente come un canale punto a punto inaffidabile.

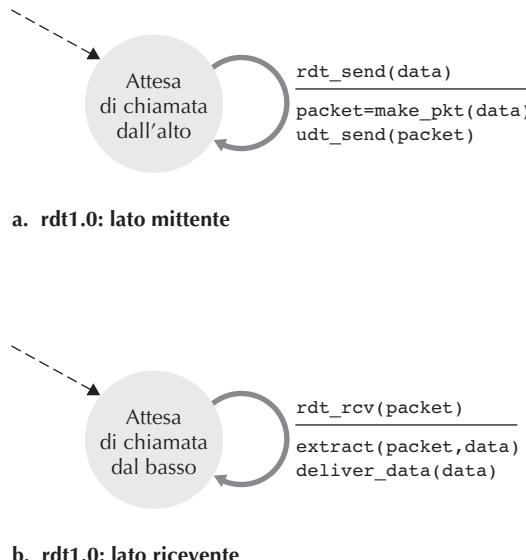
In questo paragrafo svilupperemo in modo complementare i lati mittente e ricevente di un protocollo di trasferimento dati affidabile, considerando modelli via via più complessi del sottostante canale. Per esempio, considereremo quali meccanismi di protocollo siano necessari quando il canale sottostante 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 alcuni possono 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, verrà chiamata `rdt_rcv()`. Nel momento in cui il protocollo `rdt` voglia consegnare i dati al livello superiore, lo farà chiamando `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 livello di trasporto in Internet, il termine generico pacchetto è forse più appropriato.

In questo paragrafo consideriamo solo il caso di **trasferimento dati unidirezionale**. Il caso del **trasferimento dati bidirezionale** (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



**Figura 3.9** rdt1.0: protocollo per un canale completamente affidabile.

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 etichetta, 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  $\Lambda$  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.

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 fluiscono dal mittente al destinatario; con un canale per-

fettamente 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.

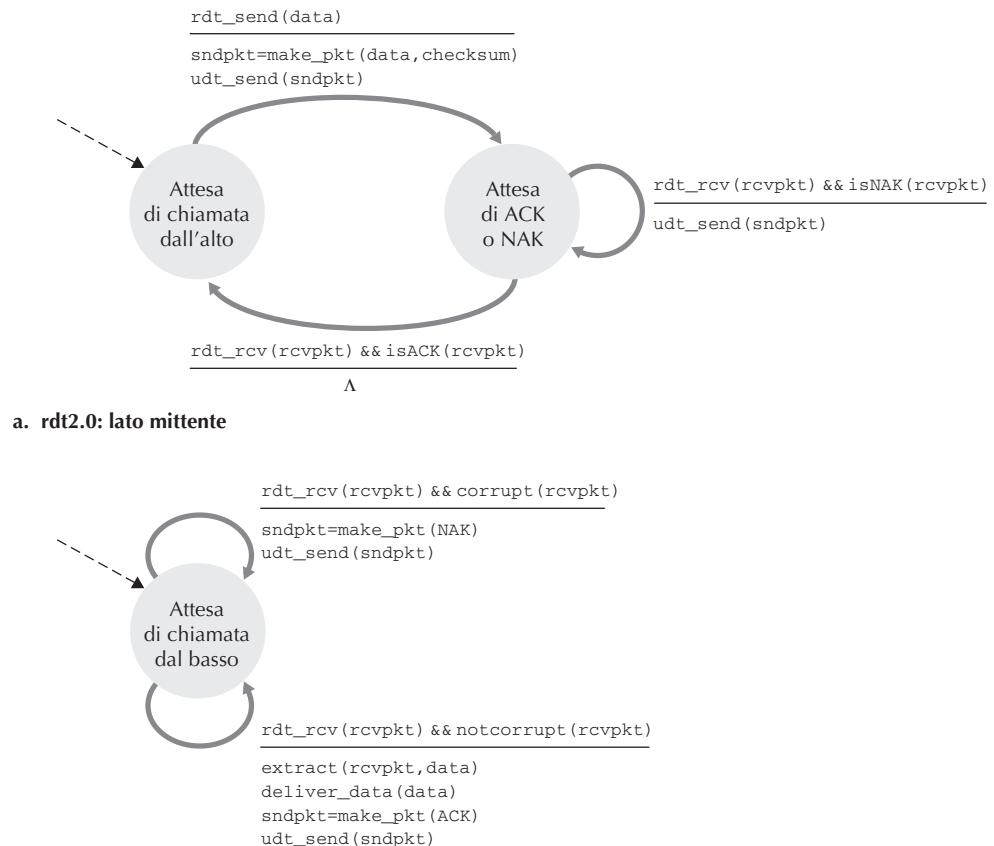
### **Trasferimento dati affidabile su un canale con errori sui bit: rdt2.0**

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 (acknowledgment) positive** (“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.* 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.* 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.* Un pacchetto ricevuto con errori sarà ritrasmesso dal mittente.



**Figura 3.10** rdt2.0: protocollo per un canale con errori sui bit.

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_rcv(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**.

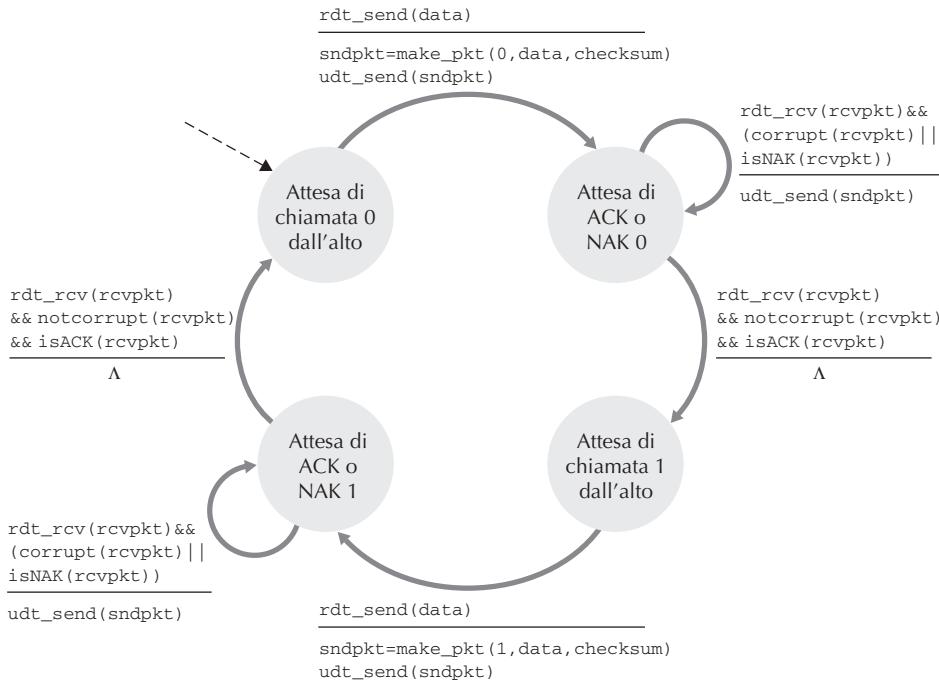
La FSM lato ricevente di `rdt2.0` ha ancora un solo stato. All'arrivo del pacchetto, il destinatario risponde o con un ACK o con un NAK, a seconda che il pacchetto sia corrotto o meno. Nella Figura 3.10 la notazione `rdt_rcv(rcvpkt) && corrupt(rcvpkt)` corrisponde al caso in cui si riceve un pacchetto 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, dobbiamo assolutamente pensare a come risolvere questo problema, quantomeno 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 probabilmente con “Che cos’hai detto *tu*?”. Ovviamamente, 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** 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.

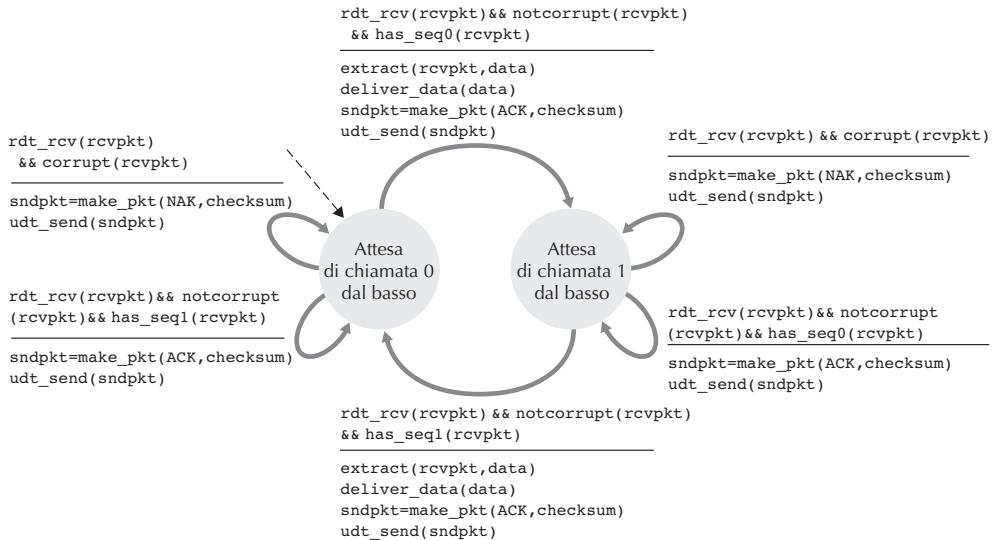
Una soluzione semplice a questo nuovo problema (adottata in quasi tutti i protocolli di trasferimento dati, tra cui TCP) consiste nell’aggiungere un campo al pacchetto



**Figura 3.11** Mittente rdt2.1.

dati, obbligando il mittente a numerare i propri pacchetti dati con un **numero di sequenza** nel nuovo campo. Al destinatario sarà sufficiente controllare questo numero per sapere se il pacchetto ricevuto rappresenti o 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.

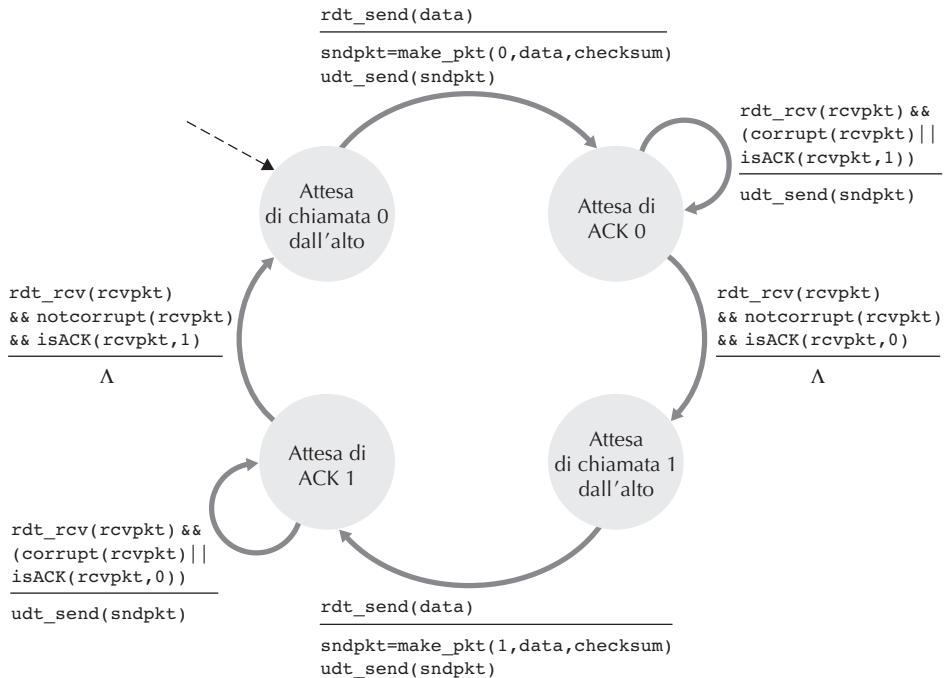
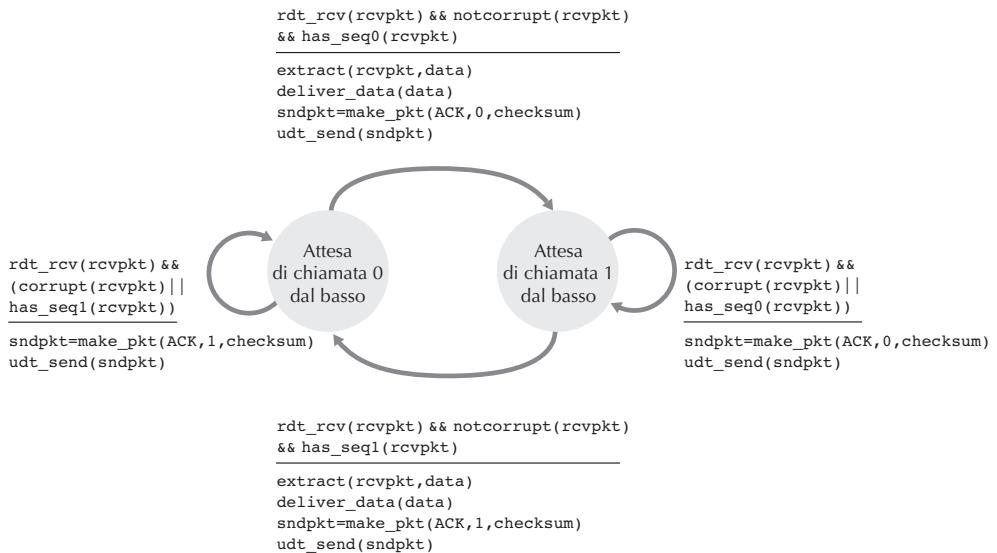


**Figura 3.12** Ricevente rdt2.1.

Il protocollo rdt2.1 usa acknowledgment 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 spedendo 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 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 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.

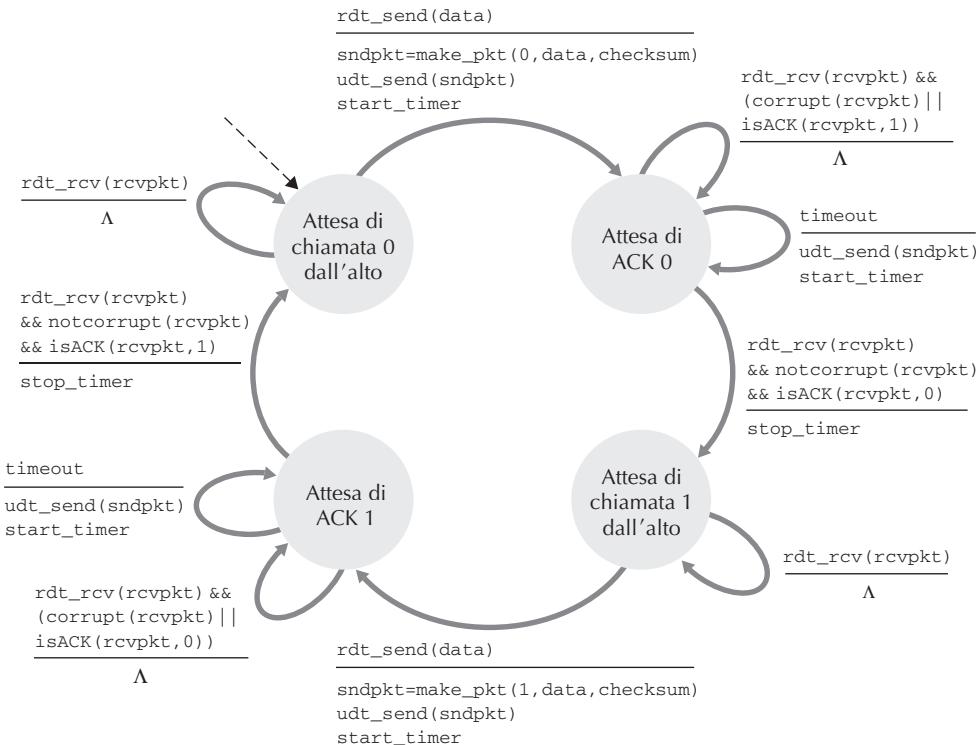
**Figura 3.13** Mittente rdt2.2.**Figura 3.14** Ricevente rdt2.2.

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 dati duplicati** 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** (*countdown 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 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



**Figura 3.15** Mittente rdt3.0.

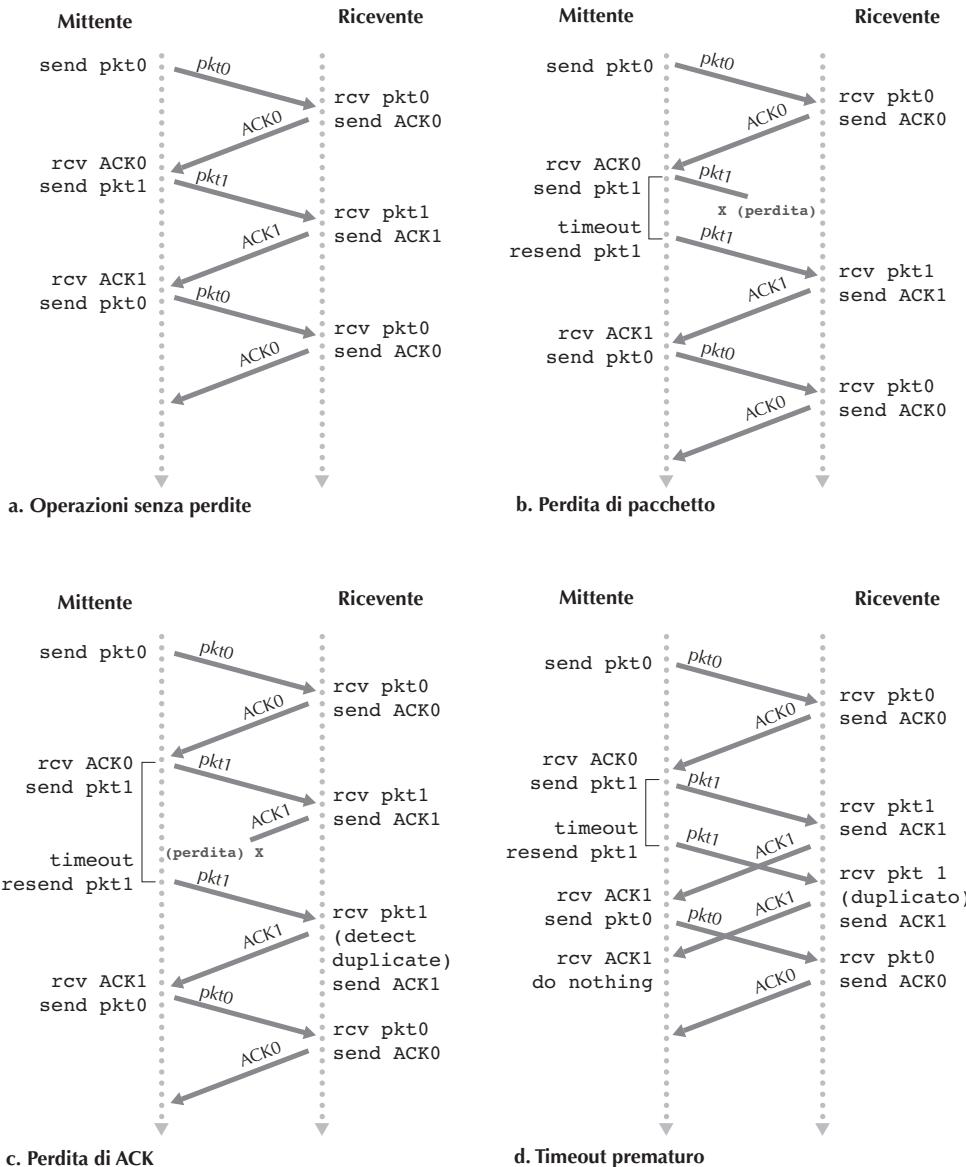
fine del capitolo. Dato che i numeri di sequenza dei pacchetti si alternano tra 0 e 1, il protocollo rdt3.0 viene talvolta detto **protocollo ad alternanza di bit**.

Ora abbiamo 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.

### 3.4.2 Protocolli per il trasferimento dati affidabile con pipeline

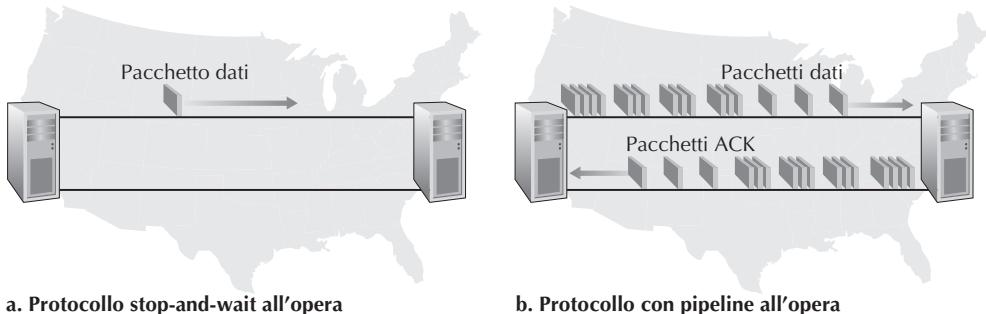
Il protocollo rdt3.0 è corretto dal punto di vista funzionale, ma difficilmente qualcuno apprezzerebbe le sue prestazioni, in particolare nelle odierne reti ad alta velocità. Il problema delle prestazioni risiede nel fatto che si tratta di un protocollo stop-and-wait.

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

**Figura 3.16** Operazioni di rdt3.0, il protocollo ad alternanza di bit.

sistemi siano connessi da un canale con tasso trasmisivo  $R$  di 1 Gbps (10<sup>9</sup> 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:

$$d_t = \frac{L}{R} = \frac{8000 \text{ bit per pacchetto}}{10^9 \text{ bit al secondo}} = 8 \text{ microsecondi}$$



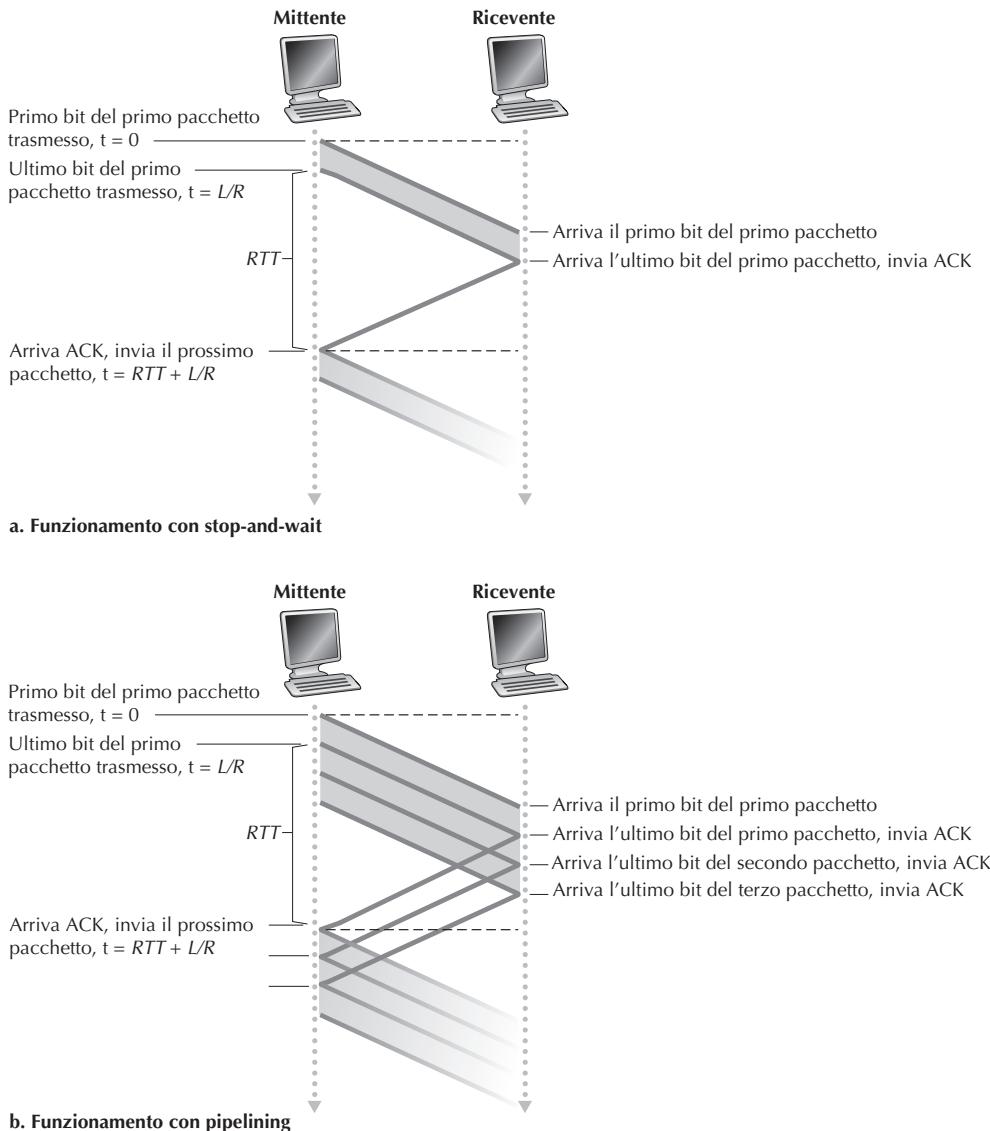
**Figura 3.17** Confronto tra protocolli stop-and-wait e con pipeline.

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  $t = RTT/2 + L/R = 15.008$  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  $t = RTT/2 + L/R + RTT/2 = 30,008$  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** 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 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,



**Figura 3.18** Invio per il protocollo stop-and-wait e con pipelining.

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*, letteralmente, “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 nu-

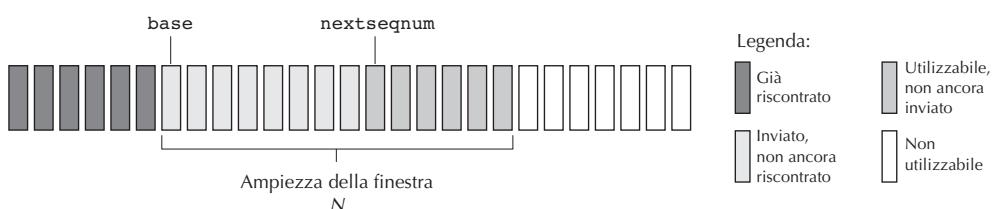
mero di sequenza univoco e che ci potrebbero essere più pacchetti in transito ancora in attesa di acknowledgment.

- 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, alternati 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 la applet GBN sul sito web del testo.

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  $[0, \text{base}-1]$  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 riscontro (nello specifico, il pacchetto con il numero di sequenza uguale a  $\text{base}$ ).



**Figura 3.19** Visione del mittente sui numeri di sequenza nel protocollo Go-Back-N.

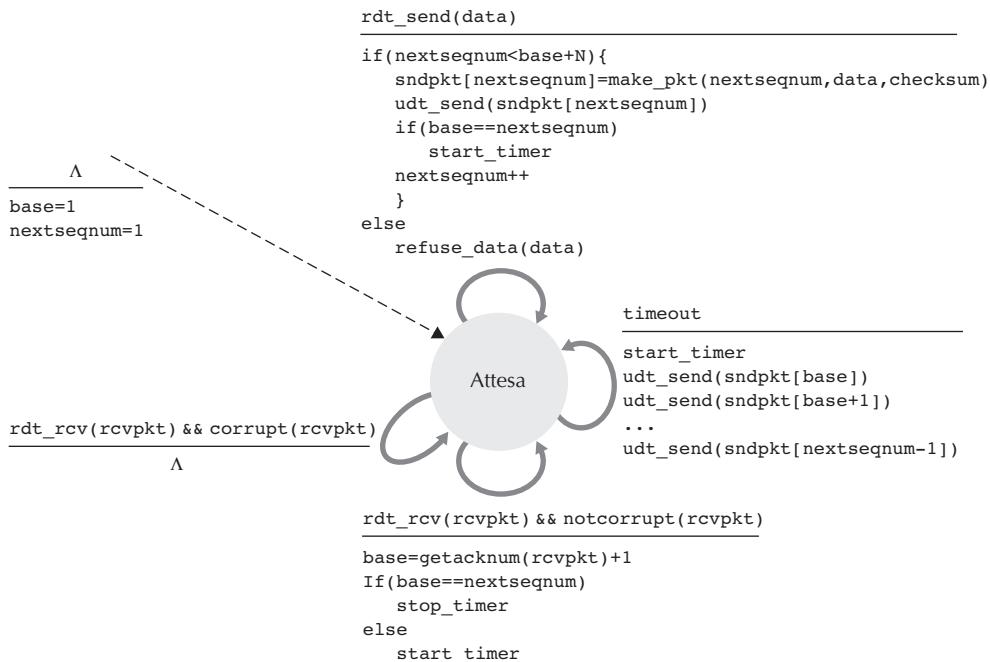
Come suggerito dalla Figura 3.19, l’intervallo di numeri di sequenza ammissibili per i pacchetti trasmessi, ma che non hanno ancora ricevuto alcun acknowledgment, può essere visto come una finestra di dimensione  $N$  sull’intervallo dei numeri di sequenza. Quando il protocollo è in funzione, questa 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.

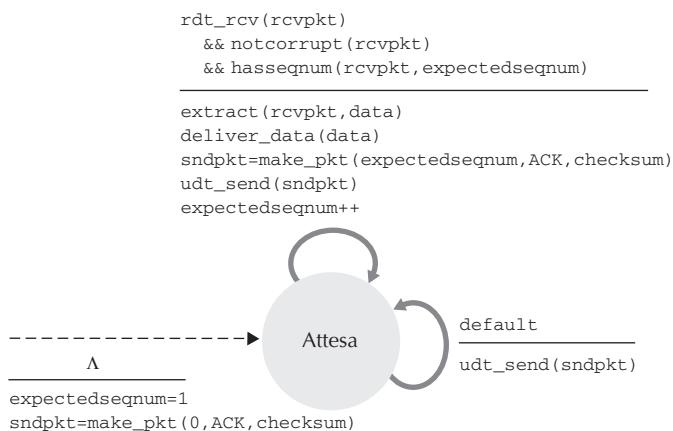
Il mittente GBN deve rispondere a tre tipi di evento.

- *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 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.



**Figura 3.20** Descrizione con automa esteso del mittente GBN.

- *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 dal destinatario. Torneremo sull’argomento tra breve, quando esamineremo GBN sul lato ricevente.



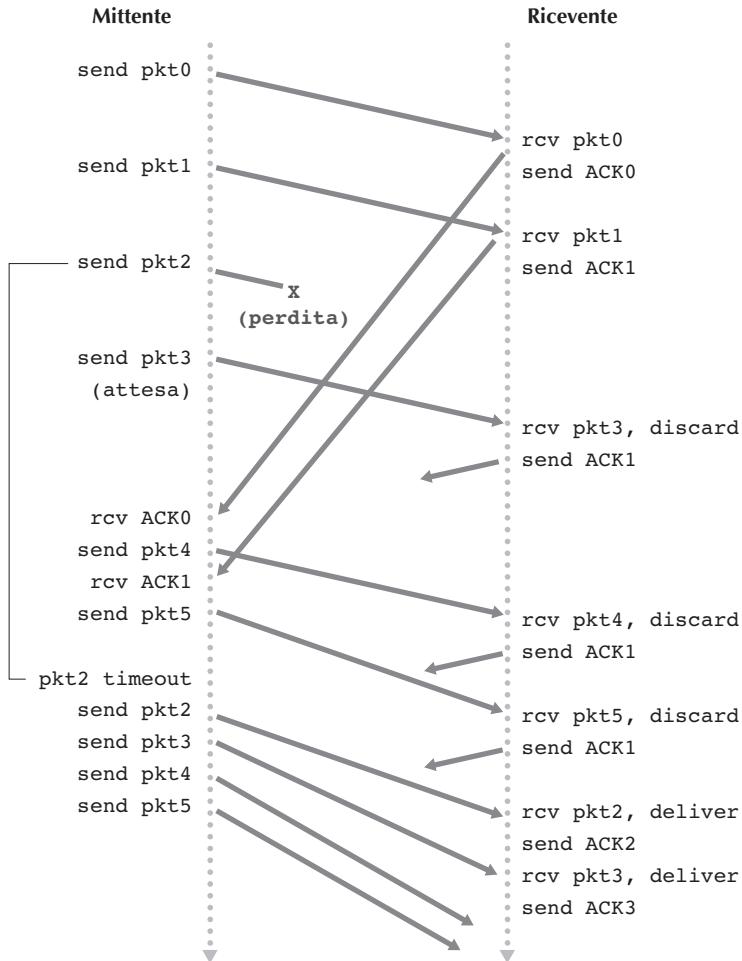
**Figura 3.21** Descrizione con automa esteso del ricevente GBN.

- *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, 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 di poter pro-



**Figura 3.22** Go-Back-N in funzione.

cedere. 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** le diverse procedure vengono invocate 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/rtrasmissione.

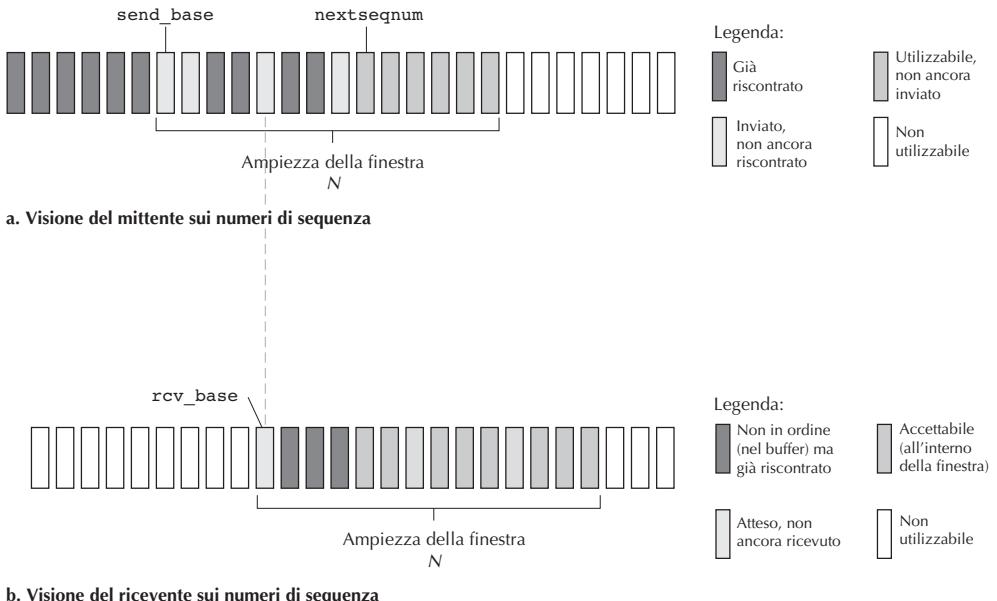
### 3.4.4 Ripetizione selettiva

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 rtrasmissioni, in molti casi inutili. Al crescere della probabilità di errore sul canale, la pipeline può saturarsi a causa di queste rtrasmissioni 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 rtrasmissioni non necessarie facendo rtrasmettere al mittente solo quei pacchetti su cui esistono sospetti di errore (ossia, smarrimento o alterazione). Questa forma di rtrasmissione 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 riscontro 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) i pacchetti già ricevuti con certi numeri di sequenza *al di sotto* dell’attuale base della finestra. Questa nuova notifica è



**Figura 3.23** Visione del mittente e del ricevente a ripetizione selettiva sullo spazio dei numeri di sequenza.

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 `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 pro-

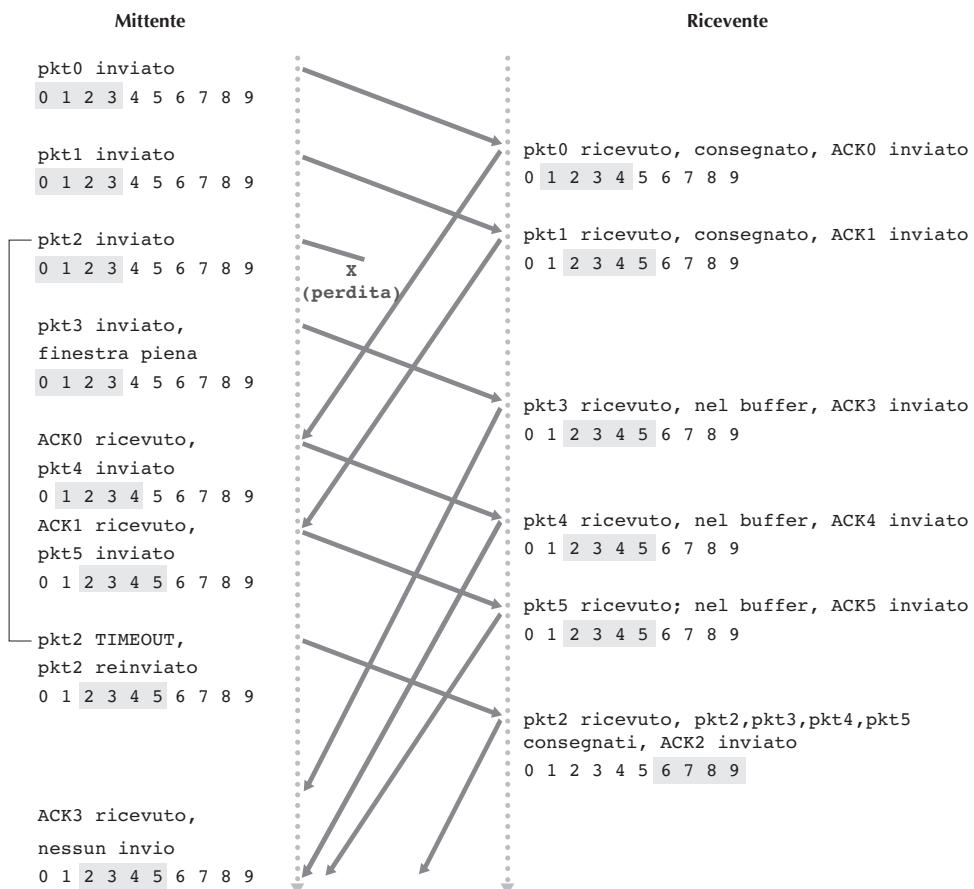
*1. Dati ricevuti dall'alto.* Quando si ricevono dati dall'alto, il mittente SR controlla il successivo numero di sequenza disponibile per il pacchetto. Se è all'interno della finestra del mittente, i dati vengono impacchettati e inviati; altrimenti sono salvati nei buffer o restituiti al livello superiore per una successiva ritrasmissione, come in GBN.

*2. Timeout.* 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].

*3. ACK ricevuto.* 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 `send_base`, 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.

**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.

toccoli 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.

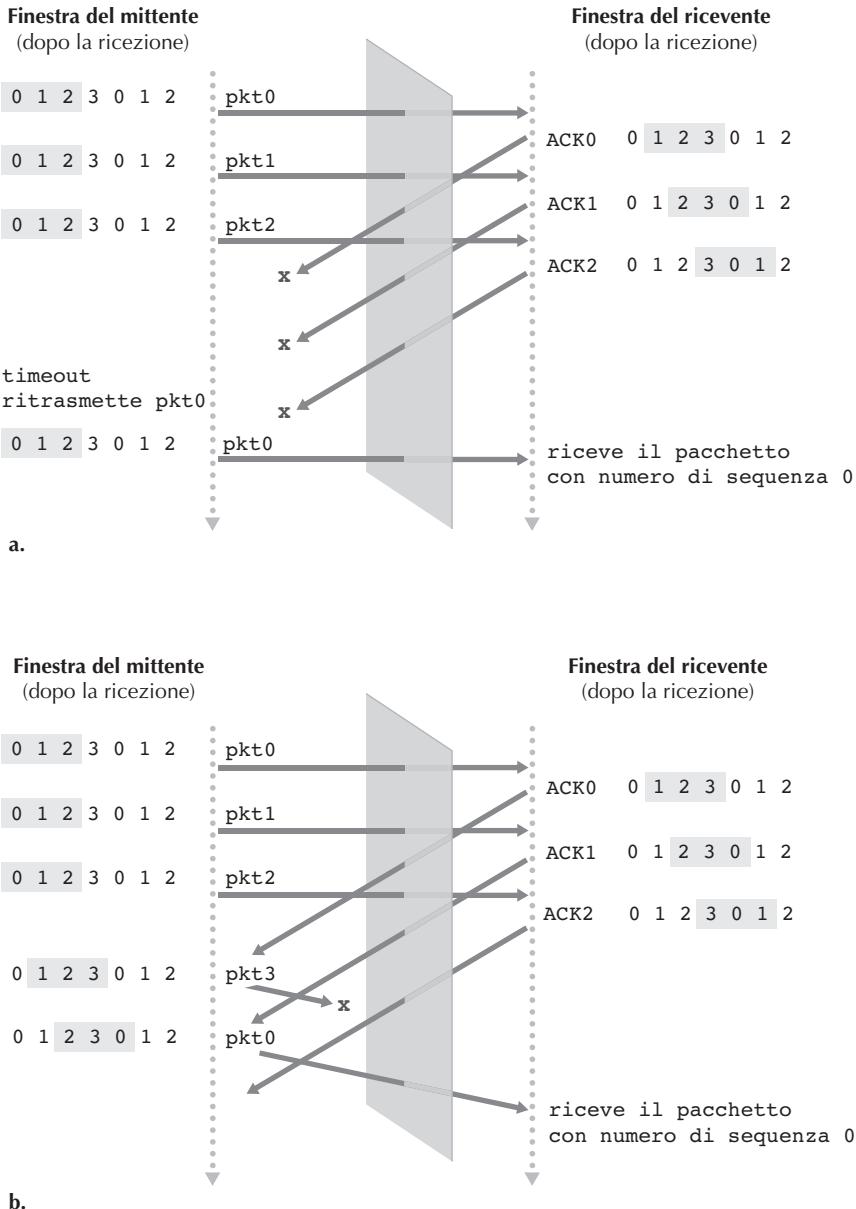
La mancanza di sincronizzazione tra le finestre del mittente e del destinatario ha conseguenze importanti quando abbiamo a che fare con un intervallo finito di numeri di sequenza. Consideriamo che cosa succederebbe, per esempio, con un intervallo di quattro numeri di sequenza per i pacchetti 0, 1, 2 e 3, e un'ampiezza di finestra pari a 3. 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 avanti 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 un problema 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.

Sul sito web del testo, trovate una applet che riproduce il funzionamento di un protocollo SR. Provate a rifare gli stessi esperimenti che avete eseguito con l'applet 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 rileggere 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



**Figura 3.27** Dilemma del ricevente SR con finestre troppo grandi: nuovo pacchetto o ritrasmissione?

pacchetti vengano mischiati. Una manifestazione di questo fenomeno è la possibile comparsa di vecchie copie di un pacchetto con numero di sequenza o di acknowledgement  $x$ , 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

**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.

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 1323] 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.

**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”.

## 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 i numeri di sequenza e gli acknowledgment. TCP viene definito negli RFC 793, 1122, 1323, 2018 e 2581.

### 3.5.1 Connessione TCP

TCP viene detto **orientato alla connessione** 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 (Capitolo 1), 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 connessione.

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**, ossia ha luogo tra un singolo mittente e un singolo destinatario. Il cosiddetto “multicast” (si veda il materiale supplementare online), ossia il trasferimento di dati da un mittente a molti destinatari in un'unica operazione, con TCP non è possibile.

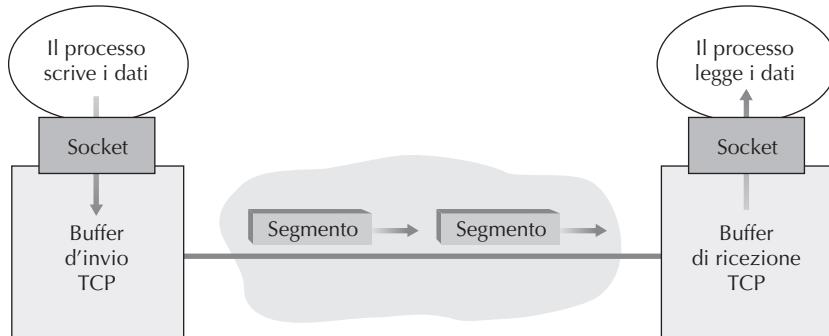
Diamo ora uno sguardo a come si instaurano le connessioni TCP. Supponiamo che il processo di un host voglia inizializzare (e quindi è il processo *client*) una connessione con il processo in un altro host (il processo *server*). Il processo applicativo client innanzitutto informa il livello di trasporto 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 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 trasmisiva massima** (MTU, *maximum transmission unit*) e poi scegliendo un MSS tale che il segmento TCP (una volta encapsulato in un datagramma IP) stia all'interno



**Figura 3.28** Buffer di invio e ricezione TCP.

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'applet di controllo di flusso in rete all'indirizzo <http://www.awl.com/kurose-ross>, 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.

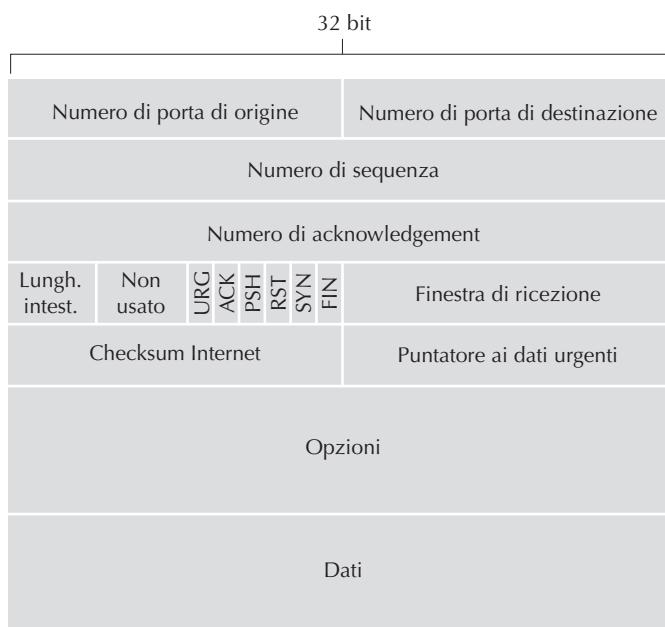
### 3.5.2 Struttura dei segmenti TCP

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 dimensioni inferiori). Le applicazioni interattive, invece, trasmettono spesso blocchi di dati più piccoli di MSS; per esempio, nelle applicazioni di login remoto quali Telnet, 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 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.

- Il **campo numero di sequenza** (*sequence number*) e il **campo numero di acknowledgement** (*acknowledgment number*), 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*), 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.



**Figura 3.29** Struttura dei segmenti TCP.

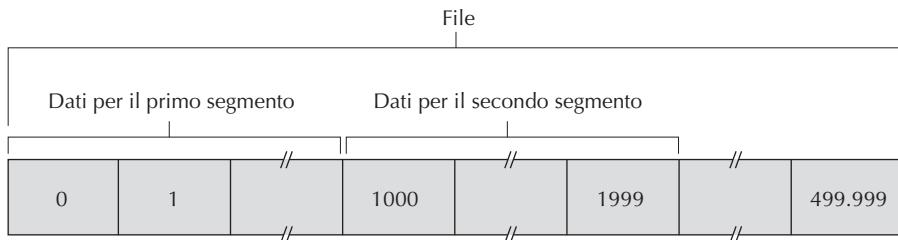
- Il campo **lunghezza dell'intestazione** (*header length*), di 4 bit, specifica la lunghezza dell'intestazione TCP in multipli di 32 bit. L'intestazione TCP ha lunghezza variabile a causa del campo delle opzioni TCP. Generalmente, il campo delle opzioni è vuoto e, pertanto, la lunghezza consueta è di 20 byte.
- 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 trasportato nel campo di acknowledgment è valido; ossia, il segmento contiene un acknowledgment 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**, 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 tuttavia 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** 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,



**Figura 3.30** Divisione di un file di dati in segmenti TCP.

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 sottile. 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 sottile: 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 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**

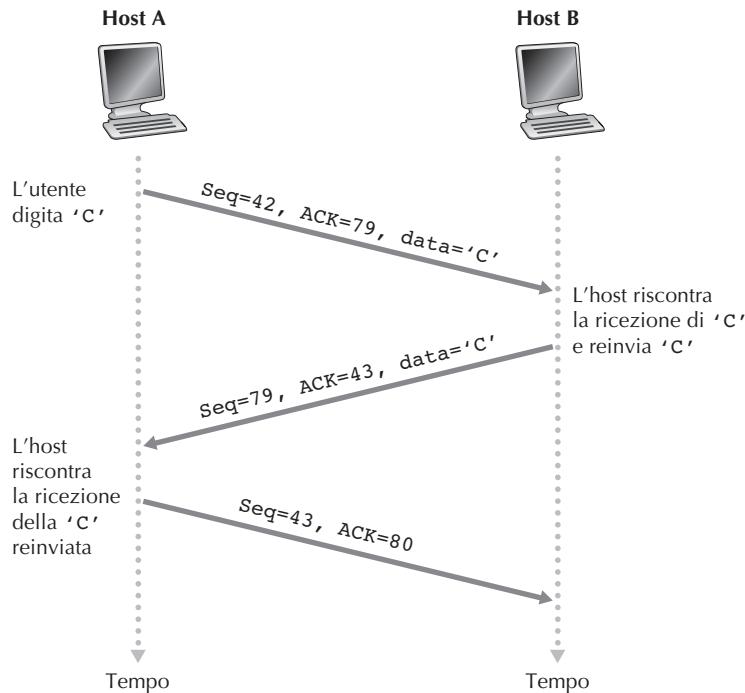
Telnet, definito nell’RFC 854, è un protocollo a livello di applicazione impiegato per il login remoto che utilizza TCP ed è progettato per funzionare tra qualsiasi coppia di host. A differenza delle applicazioni di trasferimento massiccio di dati descritte nel Capitolo 2, Telnet è un’applicazione interattiva. Presentiamo ora un esempio su Telnet, dato che illustra bene i numeri di sequenza e di acknowledgment di TCP. Abbiamo notato che oggigiorno 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*) che verranno trattati nel Paragrafo 8.7.

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. Questo “eco” viene utilizzato 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 digitì 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 segmento inviato dal client avrà numero di sequenza 42 e quello inviato dal server 79. Ricordiamo che il numero di acknowledgment 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 acknowledgment 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



**Figura 3.31** Numeri di sequenza e di acknowledgement per una semplice applicazione Telnet su TCP.

attendere i byte da 43 in poi. L'altra finalità di questo segmento è di mandare indietro un "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 riscontro 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 acknowledgment 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 acknowledgment 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

TCP, come pure il protocollo rdt descritto nel Paragrafo 3.4, utilizza un meccanismo di timeout 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*), 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

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 un problema a fine capitolo vi si chiederà il perché).

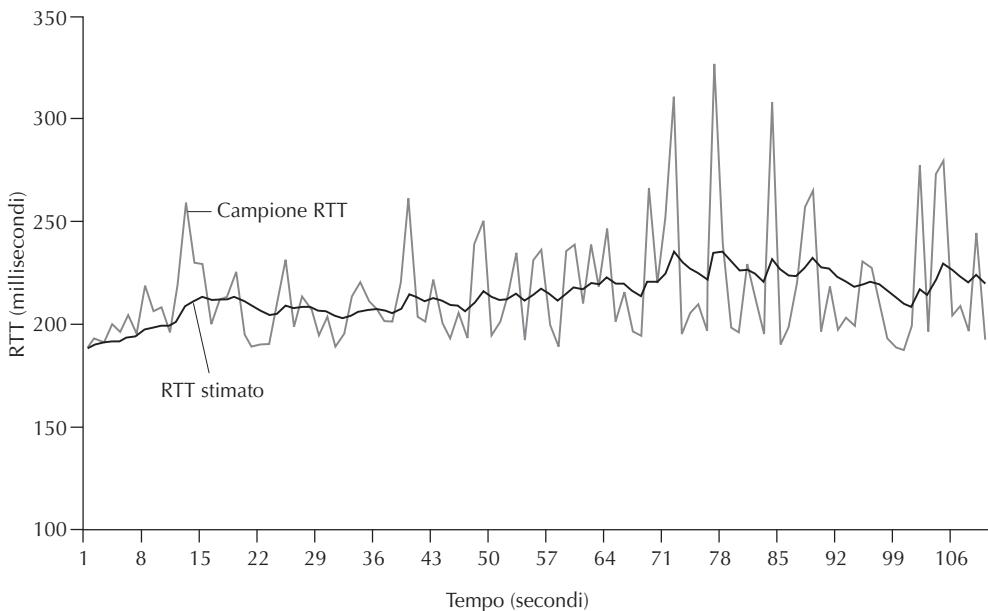
Ovviamente, i campioni variano da segmento a segmento in base alla congestione nei router e al diverso carico sui sistemi periferici. A causa di tale fluttuazione, ogni valore di *SampleRTT* può essere atipico. Per effettuare una stima risulta naturale calcolare una media dei valori di *SampleRTT*, che TCP chiama *EstimatedRTT*. Quando si ottiene un nuovo *SampleRTT*, TCP aggiorna *EstimatedRTT* secondo la formula:

$$\text{EstimatedRTT} = (1 - \alpha) \times \text{EstimatedRTT} + \alpha \times \text{SampleRTT}$$

La formula è scritta come in un linguaggio di programmazione: il nuovo valore di *EstimatedRTT* è una combinazione ponderata del suo precedente valore e 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:

$$\text{EstimatedRTT} = 0,875 \times \text{EstimatedRTT} + 0,125 \times \text{SampleRTT}$$

Si noti che *EstimatedRTT* è una media ponderata dei valori *SampleRTT*. Come discusso in un problema alla fine di questo capitolo, tale media attribuisce maggiore importanza ai campioni recenti rispetto a quelli vecchi. Questo 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, *exponen-*



**Figura 3.32** Campioni e stima di RTT.

*nential 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:

$$\text{DevRTT} = (1 - \beta) \times \text{DevRTT} + \beta \times | \text{SampleRTT} - \text{EstimatedRTT} |$$

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

**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 implicito di NAK: con un meccanismo di fast retransmit, la ricezione di tre ACK duplicati per un dato segmento equivale a un NAK implicito 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.

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:

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \times \text{DevRTT}$$

Nell'[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 ricalcolato 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 datagrammi, né l'integrità dei dati. Con il servizio IP, i datagrammi possono sovrapporsi 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** al di sopra del servizio inaffidabile.

dabile 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.

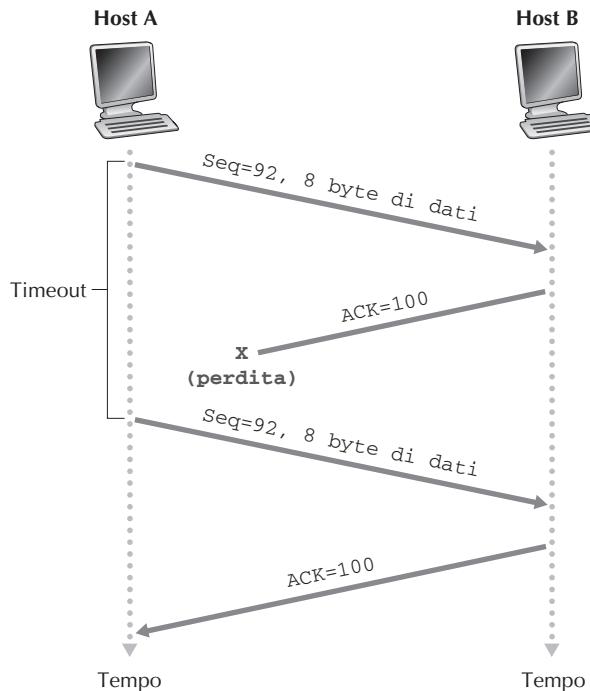
Il terzo evento, che deve essere gestito dal mittente TCP, è l'arrivo del segmento di acknowledgment con un valore valido nel campo ACK. Quando si verifica tale evento, TCP confronta il valore  $y$  di ACK con la propria variabile `SendBase`. La variabile di stato TCP `SendBase` è il numero di sequenza 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 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.

```
/* 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 */
```

**Figura 3.33** Mittente TCP semplificato.

### 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 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 ritrasmessione, 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.



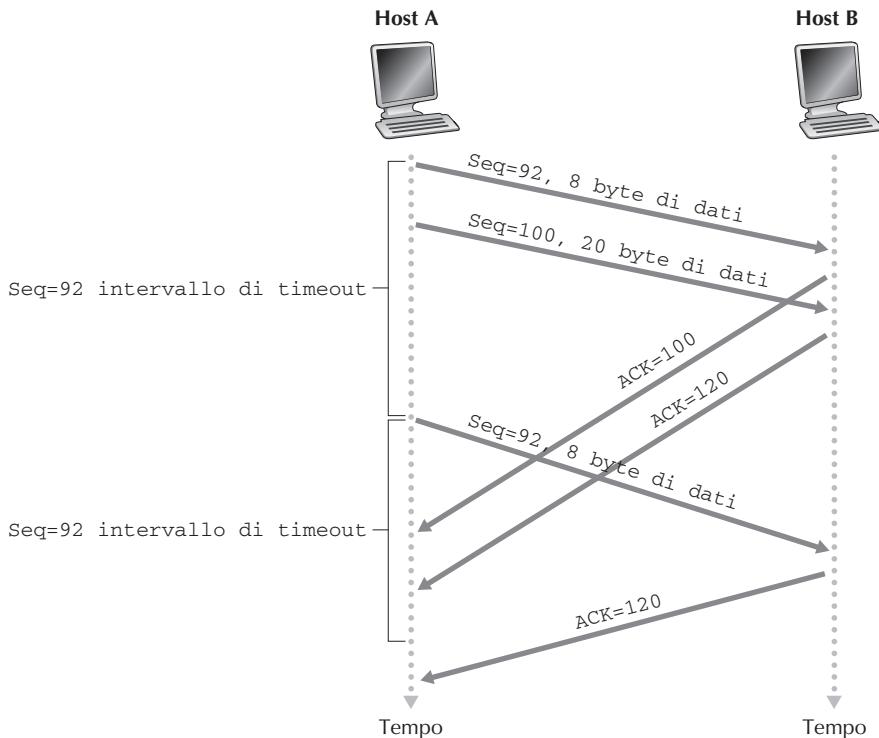
**Figura 3.34** Ritrasmissione dovuta alla perdita di un acknowledgement.

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 acknowledgement 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 acknowledgement 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.

### Raddoppio dell'intervallo di timeout

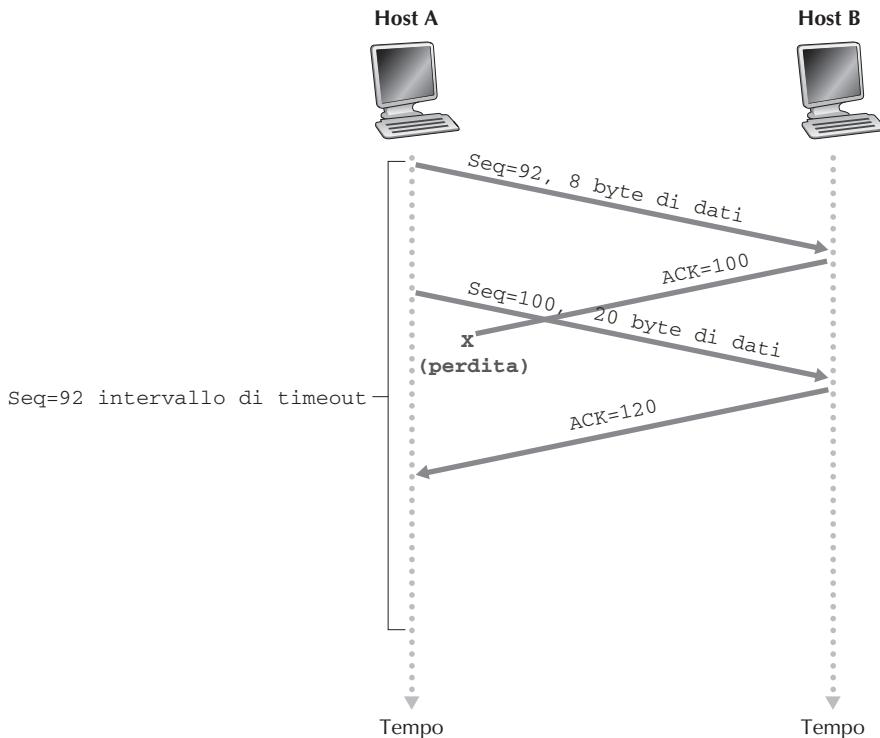
Vediamo ora alcune varianti utilizzate dalla maggior parte delle implementazioni TCP. La prima riguarda la lunghezza dell'intervallo di timeout dopo la scadenza di un timer. Con questa modifica, in tutti i casi in cui si verifica un timeout, TCP, come descritto precedentemente, ritrasmette il segmento con il più basso numero di sequenza che non abbia ancora ricevuto acknowledgement. Tuttavia, ogni volta che questo si verifica, TCP impone il successivo intervallo di timeout al doppio del valore prece-



**Figura 3.35** Segmento 100 non ritrasmesso.

dente, 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.



**Figura 3.36** Acknowledgement cumulativo che evita le ritrasmissioni del primo segmento.

### 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). Notiamo che la Tabella 3.2 contempla il caso in cui il destinatario non scarti i segmenti non ordinati.

**Tabella 3.2** Raccomandazioni sulla generazione degli acknowledgment [RFC5681].

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.

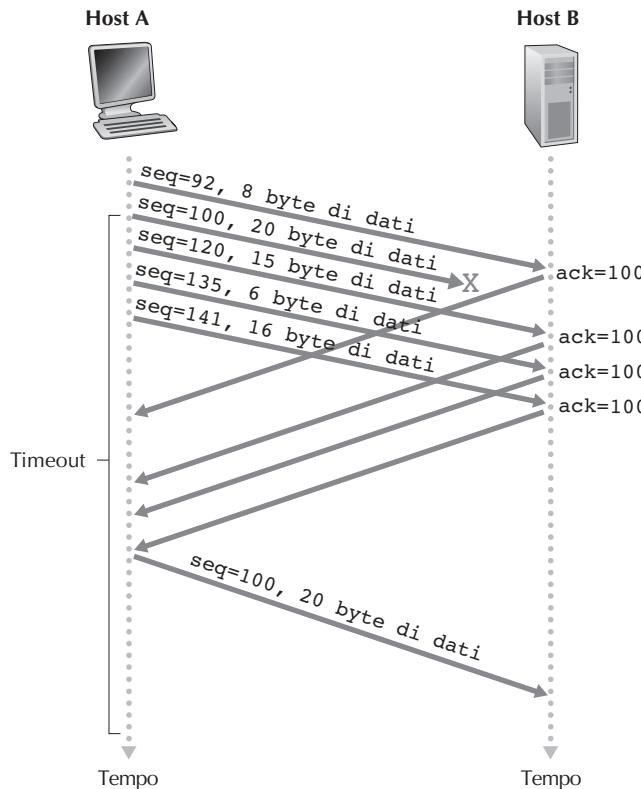
Dato che in molti casi il mittente invia un gran numero di segmenti, se uno di questi viene smarrito ci saranno probabilmente molti ACK duplicati. Se il mittente TCP riceve tre ACK duplicati per lo stesso dato, considera questo evento come indice che il segmento che lo segue è andato perduto (nei problemi 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 procedure qui de-



**Figura 3.37** Ritrasmissione veloce: il segmento perduto viene ritrasmesso prima che il suo timer scada.

scritte, evolute nel tempo e risultato di più di vent'anni di esperienza con i timer TCP, confermano questa osservazione.

### Go-Back-N o ripetizione selettiva?

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 ritrasmetterebbe 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 **riscontro 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 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 `RcvBuffer`. Di tanto in tanto, il processo applicativo nell'Host B legge dal buffer. Definiamo le seguenti variabili.

- `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:

$$\text{LastByteRcvd} - \text{LastByteRead} \leq \text{RcvBuffer}$$

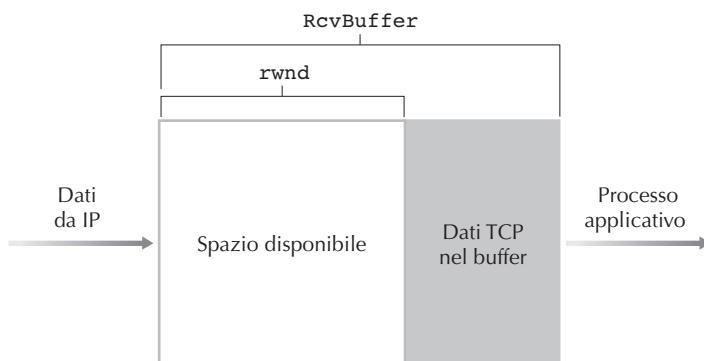
La finestra di ricezione, indicata con `rwnd`, viene impostata alla quantità di spazio disponibile nel buffer:

$$\text{rwnd} = \text{RcvBuffer} - [\text{LastByteRcvd} - \text{LastByteRead}]$$

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



**Figura 3.38** Finestra di ricezione (`rwnd`) e buffer di ricezione (`RcvBuffer`).

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

$$\text{LastByteSent} - \text{LastByteAcked} \leq rwnd$$

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  $rwnd$ .

Il sito <http://www.awl.com/kurose-ross> propone una applet Java 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, 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.

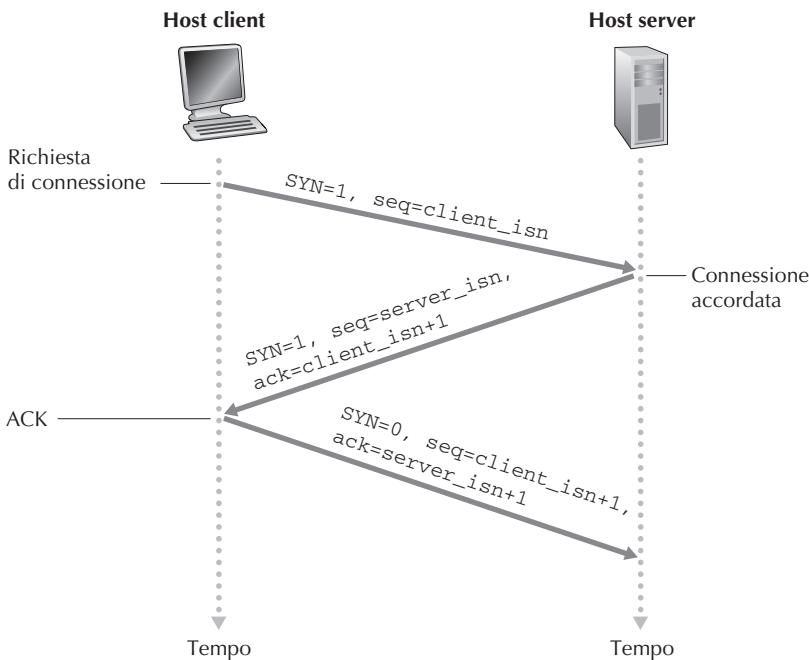
- *Passo 1.* TCP lato client invia uno speciale segmento al TCP lato server. Questo segmento speciale non contiene dati a livello applicativo, ma uno dei bit nell'in-

testazione 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].

- *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 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

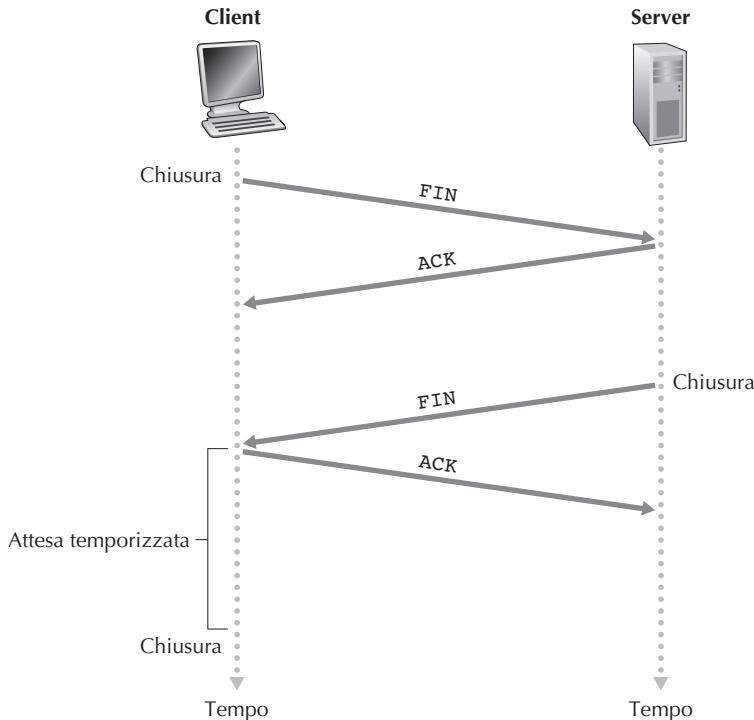


**Figura 3.39** Handshake a tre vie di TCP: scambio di segmenti.

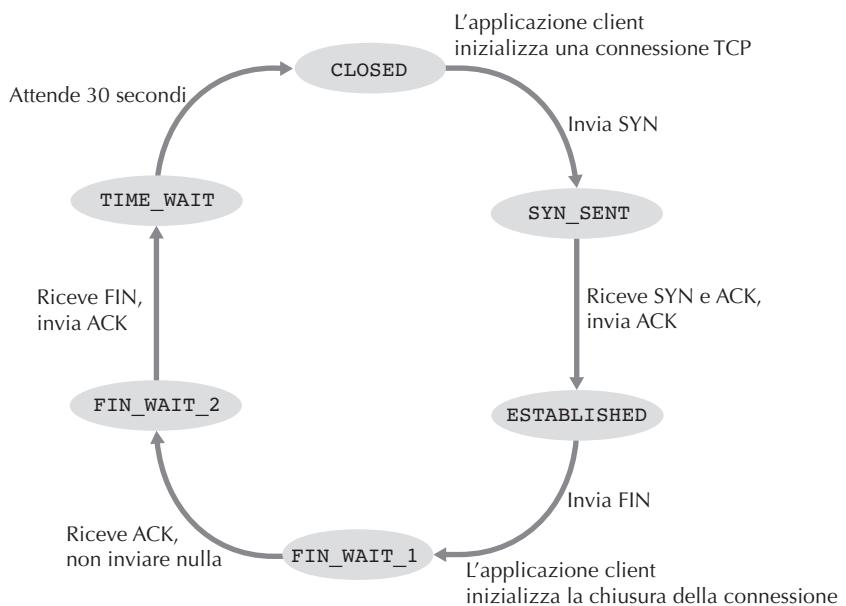
il bit FIN (Figura 3.29) con valore 1. Quando il server riceve questo segmento, risponde inviando un acknowledgment al client. Il server spedisce quindi il proprio segmento di shutdown, con il bit FIN uguale a 1. Infine, il client manda a sua volta un acknowledgment a quest'ultimo segmento del server. 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**. 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, 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



**Figura 3.40** Chiusura di una connessione TCP.

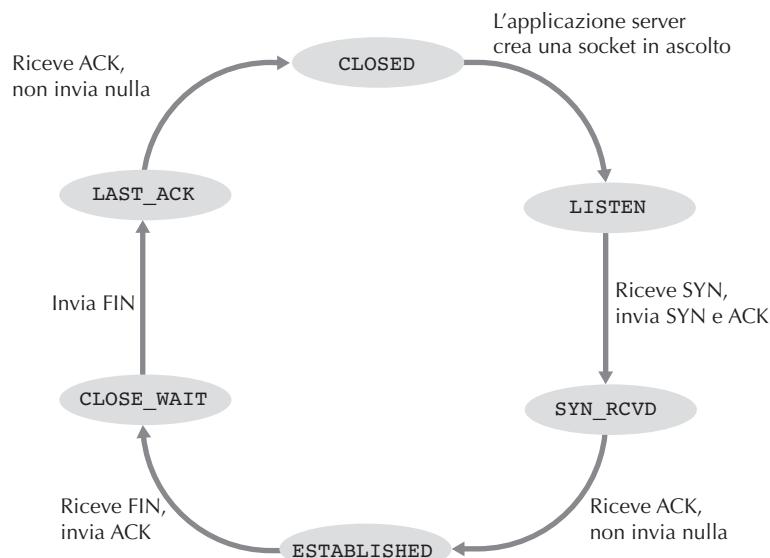


**Figura 3.41** Tipica sequenza di stati visitati da un client TCP.

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 autoesplicative. 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 chiudere 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 indirizzo 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 (Pa-



**Figura 3.42** Tipica sequenza di stati visitati da un server TCP.

ragraro 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 TCPSYNACK 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’attaccante 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 on-line).
- *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. Molto di ciò viene fatto manipolando i segmenti per la gestione delle connessioni TCP [Skoudis 2006]. Potete scaricare nmap per altri sistemi operativi 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 della congestione. Prima di farlo, tuttavia, facciamo un passo indietro ed esaminiamo i problemi relativi al controllo della 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.

**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 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.

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 op-

portuno dato che, esattamente come il trasferimento affidabile dei dati, si trova in cima alla nostra lista dei dieci problemi più importanti nel networking.

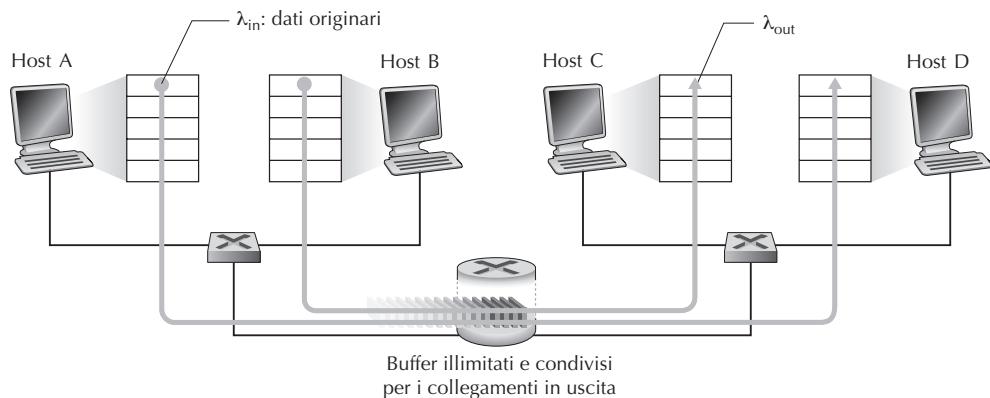
### 3.6.1 Cause e costi della congestione

Cominciamo la nostra trattazione generale del controllo della 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 trasmisivo e le reti diventano congestionate.

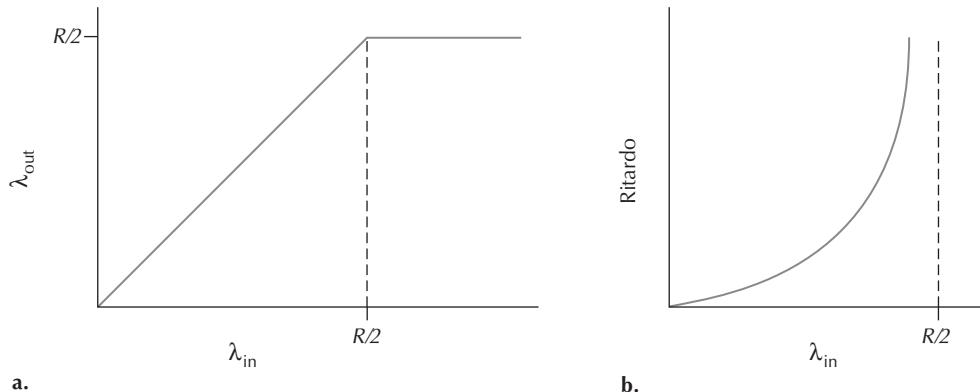
#### Scenario 1: due mittenti e un router con buffer illimitati

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 aggiuntivo 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



**Figura 3.43** Scenario di congestione 1: due connessioni che condividono un hop con buffer illimitato.

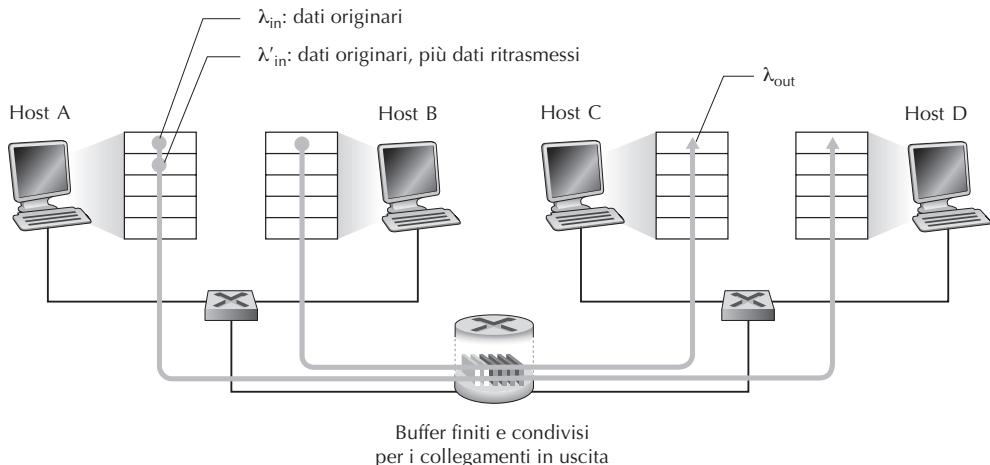


**Figura 3.44** Scenario di congestione 1: throughput e ritardi in funzione della frequenza trasmittiva dell'host.

consentono di memorizzare i pacchetti entranti quando la loro velocità di arrivo 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 alla velocità 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 collegamento, semplicemente, non è in grado di consegnare pacchetti al destinatario a un tasso superiore a  $R/2$ . Per quanto elevata sia la velocità 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 la velocità 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 questa velocità 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.



**Figura 3.45** Scenario 2: due host (con ritrasmissioni) e un router con buffer di dimensione finita.

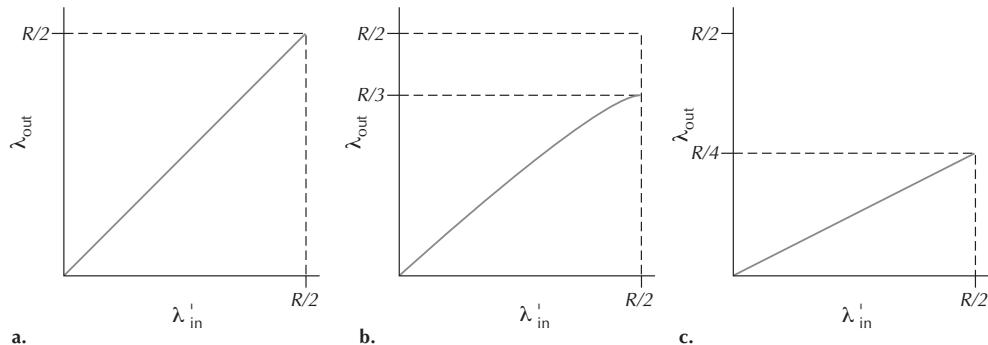
### Scenario 2: due mittenti e un router con buffer limitati

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".

In particolare, denotiamo ancora il tasso di trasmissione verso la socket con  $\lambda_{in}$  byte/s, e indichiamo con  $\lambda'_{in}$  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  $\lambda'_{in} = \lambda_{in}$  e il throughput della connessione sarebbe  $\lambda_{in}$ , 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 a pieno che



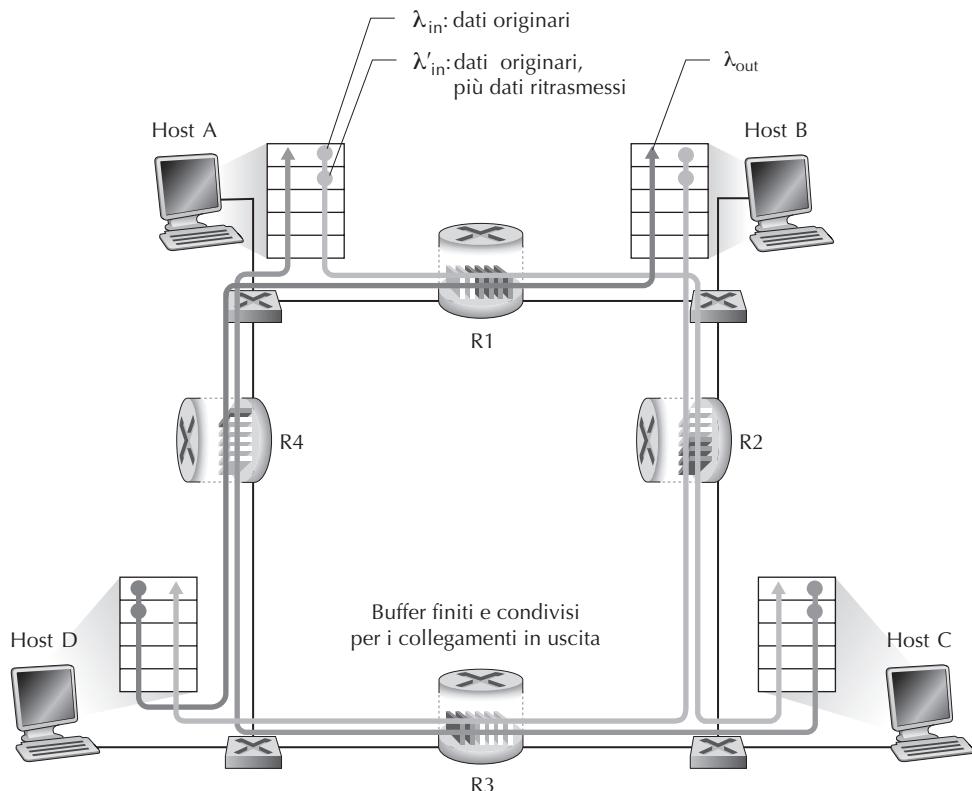
**Figura 3.46** Prestazioni dello Scenario 2 con buffer di dimensione finita.

cosa stia avvenendo, consideriamo il caso in cui il carico offerto,  $\lambda'_{\text{in}}$  (la velocità di trasmissione dei dati originari 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 destinatario 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

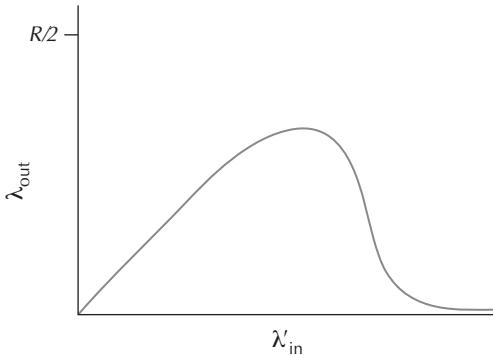
In questo caso supponiamo che i pacchetti siano trasmessi da quattro host, ciascuno su percorsi composti da due collegamenti sovrapposti tra loro (Figura 3.47); ciascun host, inoltre, utilizza un meccanismo di timeout e ritrasmissione per implementare il servizio affidabile di trasferimento dati, e tutti e quattro hanno lo stesso valore di  $\lambda_{\text{in}}$ . 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 inoltrato 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



**Figura 3.48** Prestazioni dello Scenario 3 (buffer di dimensione finita e percorsi multihop).

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 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 trasmissiva, utilizzata sui collegamenti per instradare il pacchetto fino al punto in cui è scartato, risulta sprecata.

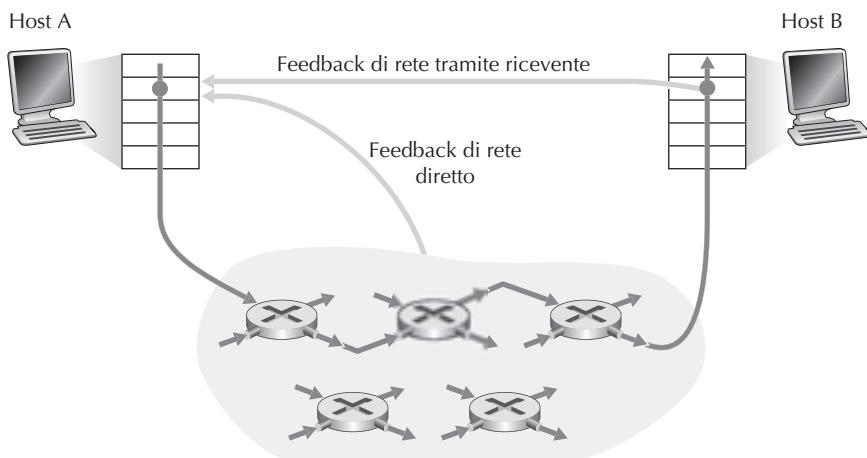
### 3.6.2 Approcci al controllo di congestione

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 bite rate)** consente a un router di informare il mittente in modo esplicito sulla frequenza trasmittiva 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



**Figura 3.49** Due modalità per ricevere feedback riguardo la congestione di rete.

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 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 che presenta, come altro componente chiave, il meccanismo di controllo della congestione. Inoltre, come indicato nel paragrafo precedente, deve utilizzare il controllo di congestione end-to-end anziché quello assistito dalla rete, dato che il livello IP non offre ai sistemi periferici un feedback esplicito sulla congestione della rete.

L’approccio scelto da TCP consiste nell’imporre a ciascun mittente un limite alla velocità 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 trasmissivo; se, invece, percepisce traffico lungo il percorso, lo riduce. Ma tale approccio solleva tre domande. Innanzitutto, come può il mittente TCP limitare la velocità 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 la velocità 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 tenere 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 la velocità trasmissiva 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, la velocità di invio del mittente è approssimativamente  $cwnd/RTT$  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 cammino 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 smarrimenti. 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 congestione (e, di conseguenza, la velocità trasmissiva). 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 di-

stribuito 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 la sua velocità trasmissiva 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 cammino dalla sorgente alla destinazione è privo di congestione ed eventi di perdita che indicano un percorso congestionato, la strategia di TCP per regolare la velocità di trasmissione è di incrementarla 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 la sua velocità di trasmissione per rilevare a che tasso trasmissivo comincia 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**, 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* è suggerita, ma non obbligatoria, per i mittenti TCP.

**BOX 3.5****TEORIA E PRATICA**

### TCP splitting: ottimizzazione delle prestazioni dei servizi di cloud computing

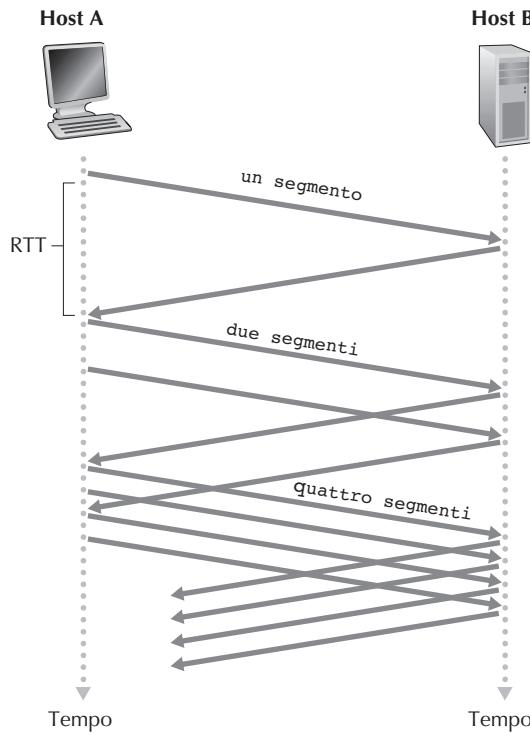
Per servizi di cloud computing quali motori di ricerca, e-mail e social network è preferibile fornire un alto livello in interazione all'utente, dandogli idealmente l'impressione che i servizi siano in esecuzione sul proprio sistema locale (smartphone inclusi). Questo può rappresentare un grave problema, 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 all'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 grossomodo 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].

### **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



**Figura 3.50** Slow start di TCP.

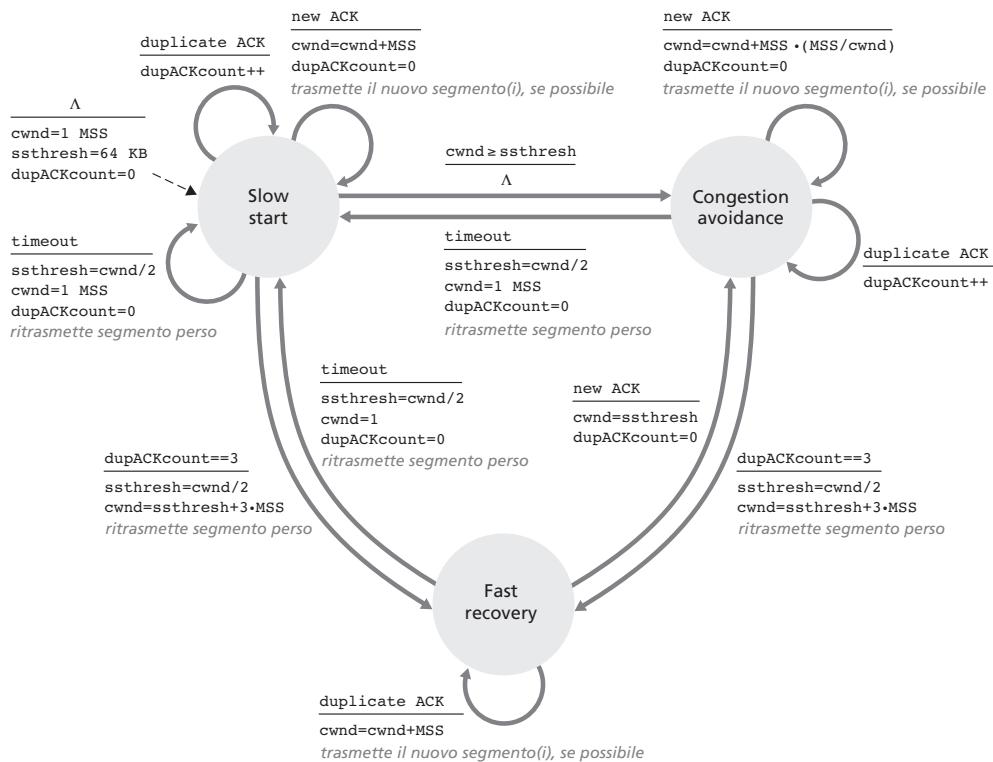
acknowledgment. Nello specifico (Figura 3.50) TCP invia il primo segmento nella rete e attende un riscontro. Se il segmento riceve un acknowledgment prima che si verifichi un evento di perdita, il mittente incrementa la finestra di congestione di 1 MSS e invia due segmenti di dimensione massima. Questi 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 raddoppio della velocità trasmissiva 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 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, 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  $MSS \times$



**Figura 3.51** Descrizione tramite automa a stati finiti del controllo di congestione TCP.

(MSS/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 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.



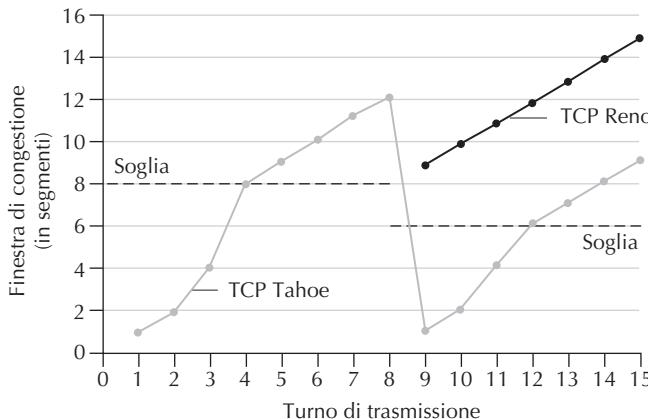
VideoNote  
Esame del  
comporta-  
mento  
di TCP

### Fast recovery

Durante la fase di fast recovery 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 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  $0,5 \times \text{cwnd} = 6$  MSS. Con TCP Reno, la finestra di congestione è posta a 6 MSS e poi cresce in modo lineare. Con TCP Tahoe, la finestra



**Figura 3.52** Evoluzione della finestra di congestione TCP (Tahoe e Reno).

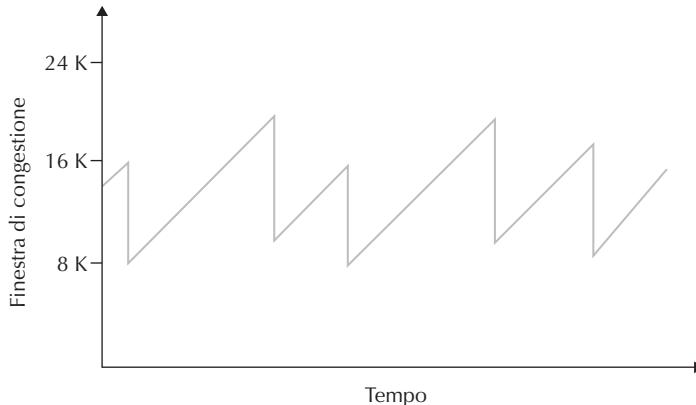
di congestione viene posta a 1 MSS e cresce in modo esponenziale fino a raggiungere la soglia  $ssthresh$ , punto dal quale cresce linearmente.

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 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 incremento 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 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 del tasso di trasmissione, finché si verifica un evento di triplice ACK duplicato. Quindi decremente 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.

Come notato precedentemente, la maggior parte delle attuali implementazioni di TCP utilizza l'algoritmo Reno [Padhye 2001]. Sono state proposte molte varianti



**Figura 3.53** Controllo di congestione a incremento additivo e decremento moltiplicativo.

dell'algoritmo Reno [RFC 3782; RFC 2018]. L'algoritmo Vegas proposto in [Brakmo 1995; Ahn 1995] tenta di evitare la congestione mantenendo allo stesso tempo un buon throughput. L'idea alla base di Vegas è (1) rilevare la congestione nei router tra origine e destinazione prima che si verifichi un evento di perdita e (2) abbassare la velocità in modo lineare quando si profila l'imminente perdita di un pacchetto. L'evento di perdita viene predetto osservando il valore *RTT*: più l'*RTT* dei pacchetti è grande, maggiore sarà la congestione nei router. Linux supporta tutti i citati algoritmi di controllo di congestione, oltre a SACK, TCP Vegas e BIC [Xu 2004]. Si veda [Afanasyev 2010] per un compendio delle molte varianti di TCP.

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 2004].

### Descrizione macroscopica del throughput di TCP

Assodato il comportamento a dente di sega di TCP, è naturale chiedersi quale potrebbe essere il throughput medio (ossia la frequenza media di trasmissioni) di una connessione TCP 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 frequenza trasmissiva 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 trasmissiva TCP varia tra  $W/(2 \times RTT)$  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:

$$\text{throughput medio di una connessione} = \frac{0,75 \times W}{RTT}$$

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 [Mahdavi 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].

### Futuro di TCP sulle connessioni a larga banda

È importante rendersi conto che il controllo di congestione TCP si è evoluto negli anni e continua a farlo. Una discussione delle attuali versioni di TCP e sulla sua evoluzione è reperibile in [RFC 5681; Floyd 2001; Afanasyev 2010]. Ciò che andava bene per Internet quando la maggior parte delle connessioni TCP trasportava traffico SMTP, FTP e Telnet non va necessariamente bene nell'odierna Internet dominata da HTTP e nemmeno nella futura Internet con servizi che ancora neppure immaginiamo.

Il bisogno di continue evoluzioni di TCP può essere illustrato considerando le connessioni ad alta velocità richieste per le applicazioni di elaborazione distribuita, come grid e cloud computing. Supponiamo per esempio di voler inviare dati a 10 Gbps attraverso una connessione TCP con segmenti da 1500 byte e  $RTT$  di 100 ms. Seguendo [RFC 3649] e usando la formula per il throughput di TCP precedentemente fornita, notiamo che per ottenere un throughput da 10 Gbps l'ampiezza media della finestra di congestione dovrebbe essere di 83.333 segmenti: sono tantissimi e la cosa ci dovrebbe preoccupare. Che cosa avverrebbe in caso di perdita? O, detto in altro modo, quale frazione dei segmenti trasmessi potrebbe venir persa consentendo ugualmente all'algoritmo di controllo di congestione specificato nella Figura 3.51 di raggiungere il tasso di 10 Gbps? Nelle domande alla fine del capitolo sarete portati a ricavare una formula che pone in relazione il throughput di una connessione TCP in funzione del tasso di perdita ( $L$ ),  $RTT$  e la dimensione massima di segmento (MSS):

$$\text{throughput medio di una connessione} = \frac{1,22 \times MSS}{RTT \sqrt{L}}$$

Usando questa formula, notiamo che per ottenere un throughput di 10 Gbps, gli odierni algoritmi di controllo di congestione possono tollerare una probabilità di perdita

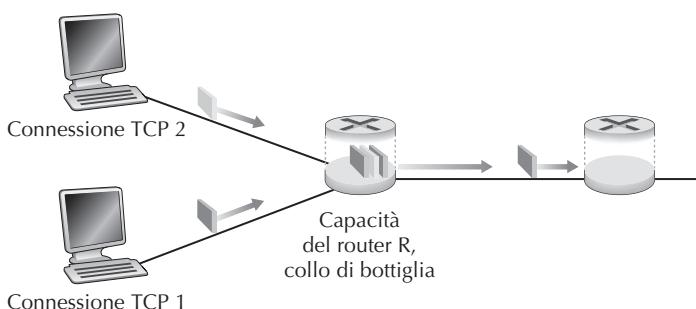
di segmenti di  $2 \times 10^{-10}$  (o equivalentemente, una perdita ogni 5 miliardi di segmenti): una probabilità molto bassa. Questa osservazione ha portato molti ricercatori a indagare su versioni di TCP più adatte ad ambienti ad alta velocità; per una trattazione di questi argomenti, si veda [Jin 2004, RFC 3649, Kelly 2003, Ha 2008, RFC 7323].

### 3.7.1 Fairness

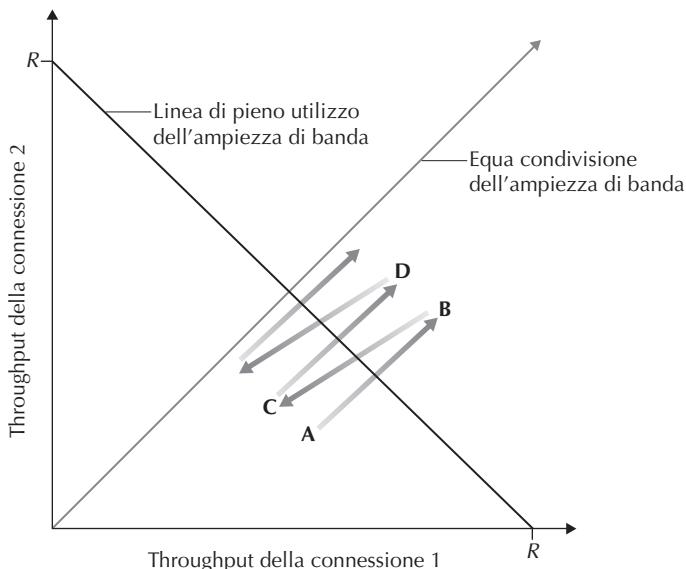
Consideriamo  $K$  connessioni TCP, ciascuna con un differente percorso end-to-end, ma che passano tutte attraverso un collegamento con capacità trasmissiva di  $R$  bps che costituisce il collo di bottiglia del sistema; con ciò intendiamo che tutti gli altri collegamenti non sono congestionati e dispongono 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 equo (*fair*) se la velocità trasmissiva 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 3.54). 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.



**Figura 3.54** Due connessioni TCP che condividono un singolo collegamento che fa da collo di bottiglia.



**Figura 3.55** Throughput delle connessioni TCP 1 e 2.

La Figura 3.55 traccia il throughput delle due connessioni. Se TCP sta suddividendo la larghezza di banda del collegamento in modo uguale tra le due connessioni, allora 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 3.55).

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 3.55. Dato che la porzione di banda del collegamento 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 congestione 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 decre-

menteranno di nuovo l'ampiezza delle loro finestre di un fattore 2, e così via. Dovrete 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, aproprio le proprie finestre di congestione 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 trasmisivo 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 trasmisivo venga ridotto, anche se la rete è molto congestionata. Piuttosto, queste applicazioni preferiscono utilizzare UDP, che non incorpora il controllo di congestione; in tal modo possono 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 trasmisivo 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 la loro velocità trasmisiva in modo appropriato. Dato che il controllo di congestione di TCP diminuisce il proprio tasso trasmisivo come risposta alla congestione crescente (e alle perdite), mentre UDP no, le sorgenti UDP possono soffocare il traffico TCP. Una tra le principali aree di ricerca è lo sviluppo di 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

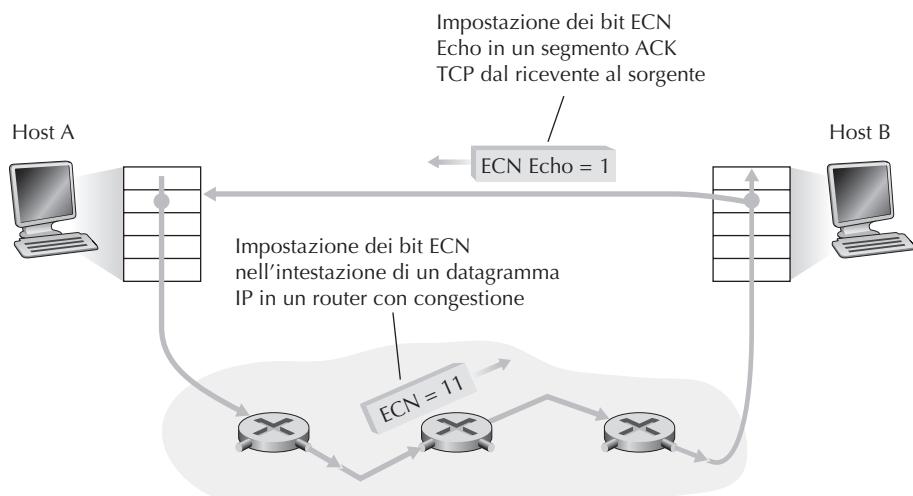
Anche se potessimo forzare il traffico UDP a comportarsi in modo equo, il problema della fairness non sarebbe tuttavia completamente risolto, perché nulla può impedire a un'applicazione basata su TCP di usare più connessioni in parallelo. Per esempio, spesso i browser web usano più connessioni TCP in 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 trasmittiva 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.

### 3.7.2 Notifica esplicita di congestione (ECN): controllo di congestione assistito dalla rete

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-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 pacchetti. Più di recente, sono state proposte, implementate e distribuite estensioni per IP e TCP [RFC 3168] che permettono alla rete di segnalare esplicitamente una congestione a mittente e ricevente TCP. Questa forma di controllo della congestione assistito dalla rete è noto come **notifica esplicita di congestione** (*explicit congestion notification*); come mostrato nella Figura 3.56 TCP e IP ne sono coinvolti.

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 3.56. 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.



**Figura 3.56** Notifica esplicita di congestione: controllo di congestione assistito dalla rete.

Come mostrato in Figura 3.56, quando il destinatario TCP riceve un’indicazione di congestione, 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 ECN: *datagram congestion control protocol* (DCCP) [RFC4340] fornisce un controllo di congestione ECN leggero, ma non affidabile, e DCTCP (*data center TCP*) [Alizadeh 2010], progettato specificatamente per infrastrutture di rete all’interno di data center.

## 3.8 Riepilogo

Abbiamo iniziato questo capitolo studiando i servizi offerti dai protocolli a livello di trasporto 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, garanzie su ritardi e su 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 garanzie su ritardi o banda sufficiente ai segmenti, il protocollo a livello di trasporto non può a sua volta fornire tali garanzie.

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 nel corso del capitolo abbiamo trattato il trasferimento affidabile dei dati, dovremmo ricordarci che questo 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 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 esaminato “dall’alto” il controllo di congestione e nel successivo abbiamo mostrato come TCP lo implementa. Abbiamo imparato che il controllo di congestione è assolutamente necessario per il benessere della rete. Senza di esso, la rete può facilmente rimanere bloccata e non trasportare dati. Abbiamo quindi appreso che TCP implementa un meccanismo di controllo di congestione end-to-end, che incrementa in modo additivo il tasso trasmisivo quando il percorso della connessione TCP è giudicato sgombro e lo 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 trasmisiva del collegamento. Abbiamo anche affrontato l’impatto della creazione di connessioni TCP e slow start sulla latenza, osservando come, in molti scenari importanti, questi due aspetti contribuiscono 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 prossimi anni.

La nostra trattazione sui protocolli di trasporto specifici di Internet, in questo capitolo, si è focalizzata su TCP e UDP, i due “cavalli da soma” del livello di trasporto di Internet. Tuttavia, due decenni di esperienza con questi due protocolli consentono di individuare circostanze nelle quali nulla va alla perfezione. I ricercatori hanno lavorato per sviluppare protocolli a livello di trasporto aggiuntivi, molti dei quali sono attualmente proposti per diventare standard di IETF.

Il *datagram congestion control protocol* (DCCP) [RFC 4340] fornisce un servizio con basso overhead, orientato ai messaggi e inaffidabile come UDP, ma con una forma di controllo di congestione selezionata dall’applicazione che è compatibile con TCP. Se per un’applicazione è necessario un trasferimento dati affidabile o quasi, allora questo dovrà essere previsto all’interno dell’applicazione stessa, magari utilizzando i meccanismi che abbiamo studiato nel Paragrafo 3.4. DCCP è stato pensato per essere usato nelle applicazioni di streaming multimediale (Capitolo 9 on-line) che possono sfruttare il compromesso tra la tempestività e l’affidabilità della consegna dei dati, ma che vogliono essere reattive alla congestione di rete.

Il protocollo QUIC (Quick UDP Internet Connection, [Iyengar 2016]), utilizzato nel browser Chrome di Google, impiega UDP come protocollo di trasporto e implementa l’affidabilità in un protocollo a livello di applicazione che utilizza meccanismi simili a quelli di TCP. Circa la metà delle richieste da browser Chrome ai server di Google sono effettuate utilizzando QUIC, come riportato da Google all’inizio del 2015. DCTCP (Data Center TCP) [Alizadeh 2010] è una versione di TCP progettata specificatamente per reti all’interno dei data center che utilizza ECN.

Lo *stream control transmission protocol* (SCTP) [RFC 4960, RFC 3286] è un protocollo affidabile, orientato ai messaggi, in cui molti “flussi” differenti di livello ap-

plicativo possono essere sottoposti a multiplexing in una singola connessione SCTP (un approccio noto come “multi-streaming”). Dal punto di vista dell’affidabilità, i differenti flussi all’interno della connessione vengono gestiti separatamente, in modo che la perdita di pacchetti in un flusso non influisca sulla consegna dei dati in un altro flusso. SCTP consente anche di trasferire i dati su due percorsi quando un host è collegato a due o più reti, la consegna opzionale dei dati fuori ordine e parecchie altre caratteristiche. Gli algoritmi di controllo di flusso e di congestione sono essenzialmente gli stessi di TCP.

Il protocollo *TCP friendly rate control* (TFRC) [RFC 5348] è un protocollo per il controllo di congestione piuttosto che un protocollo di trasporto. Specifica un meccanismo di controllo di congestione che potrebbe essere usato in un altro protocollo di trasporto come DCCP: effettivamente TFRC è uno dei due protocolli disponibili in DCCP selezionabili dalle applicazioni. L’obiettivo di TFRC è di appianare il comportamento “a dente di sega” nel controllo di congestione di TCP (Figura 3.53), pur mantenendo un tasso di invio ragionevolmente vicino a quello di TCP. Con una frequenza di invio più omogenea di TCP, TFRC è particolarmente adatto per le applicazioni multimediali come la telefonia su IP o lo streaming audio e video, dove un tasso di invio omogeneo è importante. TFRC è un protocollo “basato sulle equazioni”, che usa il tasso di perdita dei pacchetti misurato come input di una equazione [Padhye 2000] che stima quale sarebbe il throughput di TCP se la sessione TCP fosse soggetta a quel tasso di perdita. Questo tasso viene poi considerato come un obiettivo per la velocità di invio di TFRC.

Solo il futuro ci dirà se DCCP, SCTP o TFRC saranno installati su larga scala. Mentre questi protocolli chiaramente forniscono funzionalità avanzate rispetto a TCP e UDP, questi ultimi si sono dimostrati “sufficientemente adatti” nel corso degli anni. Se il “meglio” prevarrà sul “sufficientemente buono”, dipenderà da un complesso miscuglio di considerazioni tecniche, sociali ed economiche.

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 de-

## CAPITOLO

# 4

## Livello di rete: piano dei dati

Nel precedente capitolo abbiamo appreso che il livello di trasporto fornisce varie forme di comunicazione tra processi e fa 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 capitolo e nel prossimo imparerete che cosa sta dietro il servizio di comunicazione host-to-host e che cosa lo fa funzionare. Vedremo che, a differenza del livello di trasporto, una parte del livello di rete è presente in ognuno degli host e 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 Capitolo 4 tratteremo dapprima le funzioni del piano dei dati del livello di rete: le funzioni che ogni router implementa singolarmente (*per-router*) 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 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 in modo monolitico nei router. 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” remoto. 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 parecchi router sul percorso che li collega. Consideriamo il ruolo del livello di rete nell’ambito di una trasmissione da H1 a H2 e il compito dei router. 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. Nell’host H2, il livello di rete riceve i datagrammi dal proprio router vicino R2, estrae i segmenti e li consegna al livello di trasporto. In tale processo 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 trasferiti 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.

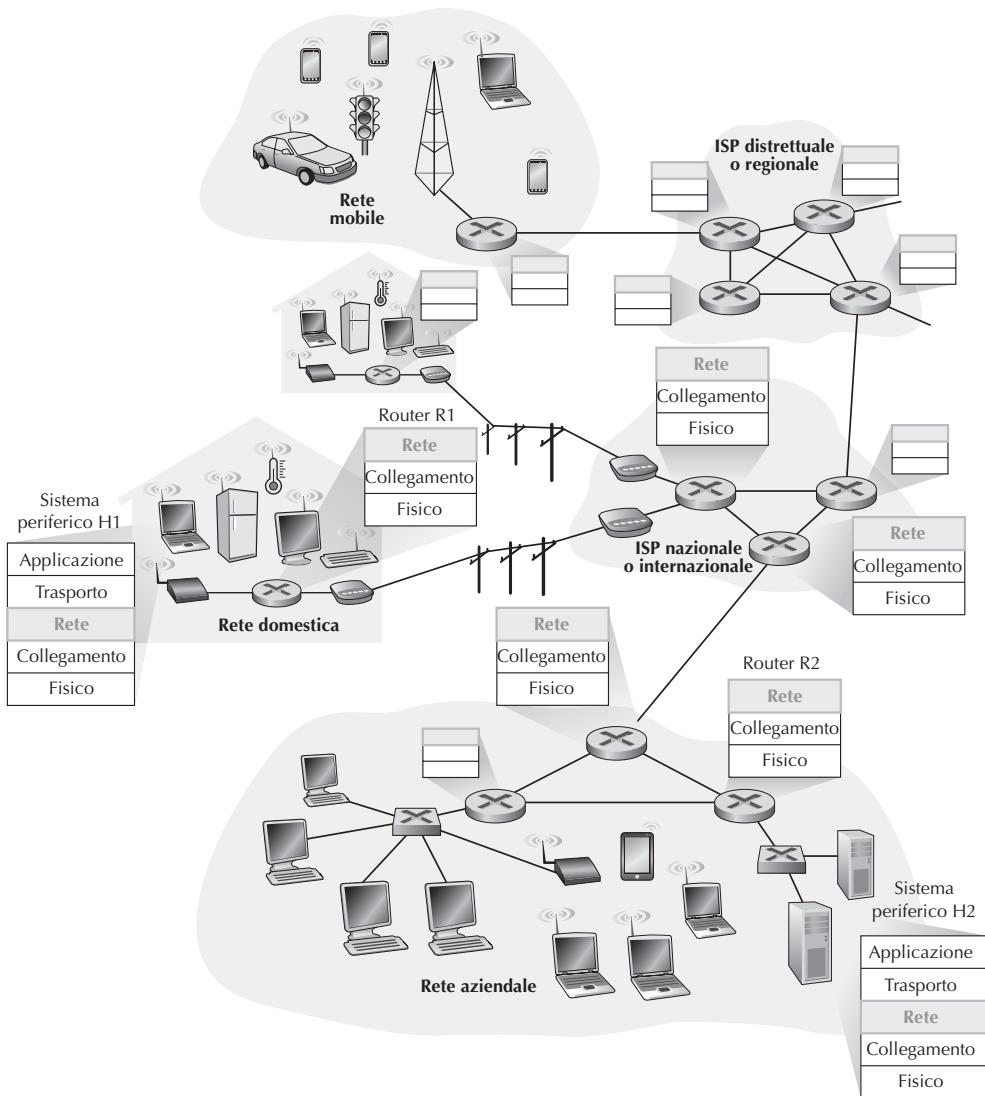


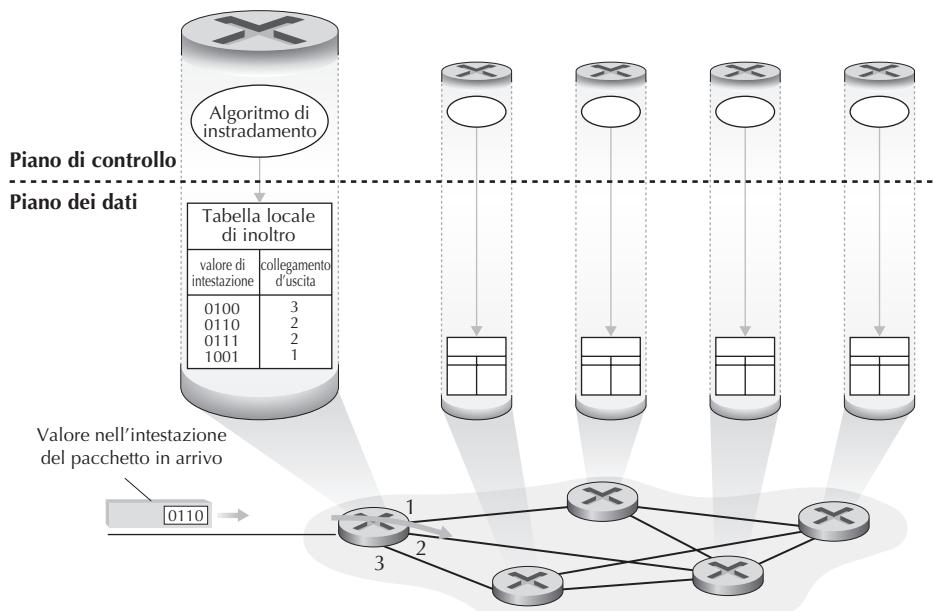
Figura 4.1 Livello di rete.

- **Instradamento (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 inherente il 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’instradamento 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’instradamento corrisponde al processo di pianificazione dell’intero viaggio cui si è provveduto consultando una cartina stradale e scegliendo uno tra i molteplici percorsi possibili. Lungo l’itinerario prestabilito, Giacomo percorre una serie di segmenti di strada e incontra numerosi svincoli il cui attraversamento può essere paragonato all’inoltro: l’auto entra nello svincolo, prende la direzione desiderata 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 i loro valori che utilizzano come indice nella **tabella di inoltro** (*forwarding table* 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 di 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.



**Figura 4.2** Gli algoritmi di instradamento determinano i valori nelle tabelle di inoltro.

### Piano di controllo: l'approccio tradizionale

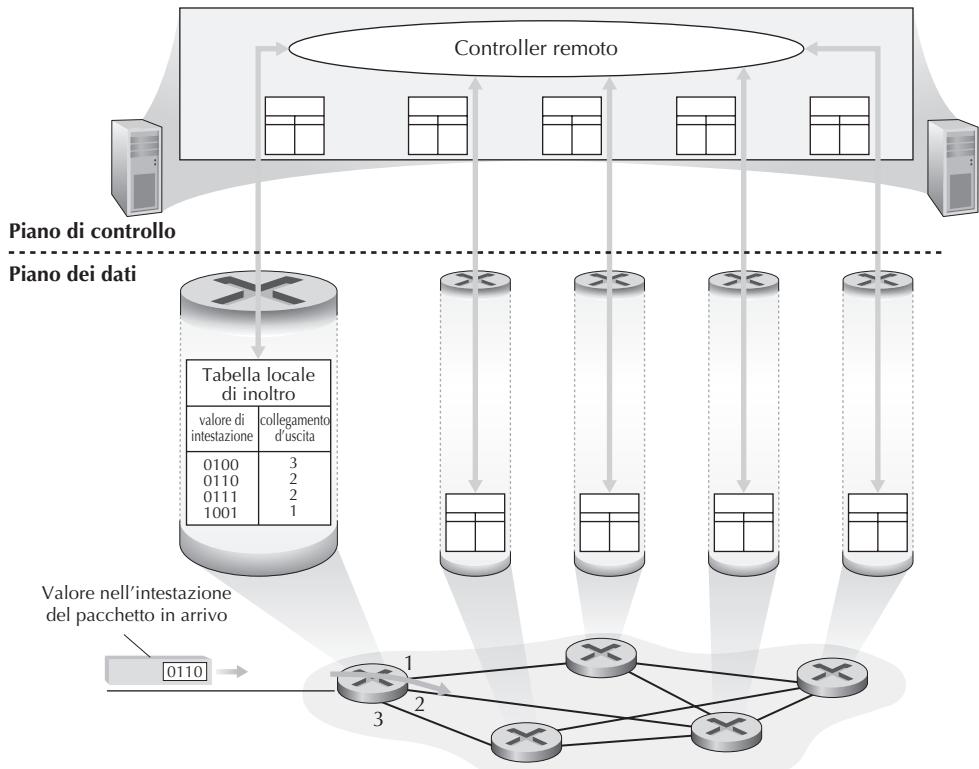
Vi starete indubbiamente chiedendo come vengano configurate le tabelle di inoltro nei router. Questo è un argomento cruciale che evidenzia l'importante interazione tra inoltro (nel piano dei dati) e instradamento (nel piano di controllo). Come mostrato nella Figura 4.2, l'algoritmo di instradamento determina 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 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, dove 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 tabelle 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” 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, per-



**Figura 4.3** Un controller remoto determina e distribuisce i valori delle tabelle di inoltro.

mettendo 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

Prima di addentrarci nel piano dei dati del livello di rete concludiamo la nostra introduzione a una visione più ampia e consideriamo 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 della 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 un tasso 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 avrà 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. Potrebbe sembrare che questa definizione sia in realtà un eufemismo per indicare “nessun servizio”. In effetti, una rete che non consegna 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 [MFA Forum 2012; 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 si è dimostrato abbastanza buono da supportare una stupefacente gamma di applicazioni quali i servizi di streaming video come Netflix, le applicazioni di voice-and-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 celebri 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 di 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).

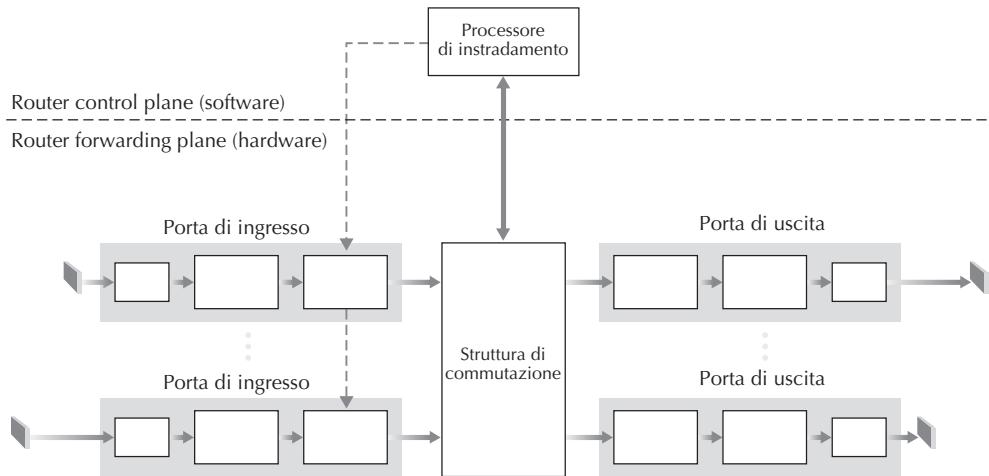
Citiamo qui di sfuggita che i termini di *forwarding* e *switching* sono spesso usati in modo intercambiabile dai ricercatori e professionisti delle reti, e così faremo in questo testo. Sempre a proposito di terminologia, vale la pena menzionare altri termini che vengono spesso impiegati in modo intercambiabile, 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 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 inter-operare con le analoghe funzioni all’altro capo del collegamento di ingresso. Svolgono inoltre la cruciale funzione di ricerca (riquadri



**Figura 4.4** Architettura di un router.

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 960 porte Ethernet a 10 Gbps, con una capacità totale del sistema di router di 80 Tbps [Juniper MX 2020 2016].

- **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 in-

stradamento 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, 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 10 Gbps e un datagramma IP di 64 byte, la porta di input ha solo 51,2 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. L’hardware 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 una CPU tradizionale).

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 potrebbe 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.

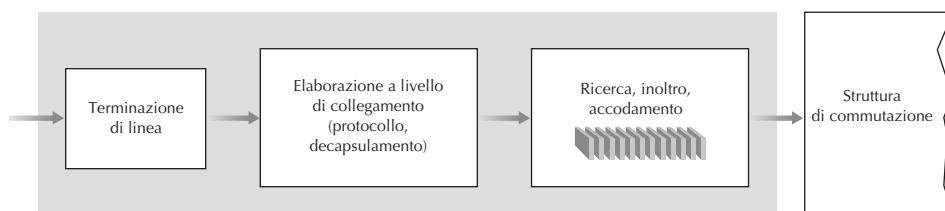
L’auto entra nella rotonda (che può avere altre auto entranti e uscenti) e infine esce sulla rampa di uscita prescritta, dove può incontrare altre auto.

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. [Iyer 2008; Chao 2001; Chuang 2005; Turner 1988; McKeown 1997a; Partridge 1998; Sopranos 2011] 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 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 por-



**Figura 4.5** Elaborazione alle porte di ingresso.

ta 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 è 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:

	Intervallo degli indirizzi di destinazione				Interfaccia
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:

	Corrispondenza di prefisso			Interfaccia
	11001000	00010111	00010	0
	11001000	00010111	00011000	1
	11001000	00010111	00011	2
	altrimenti			3

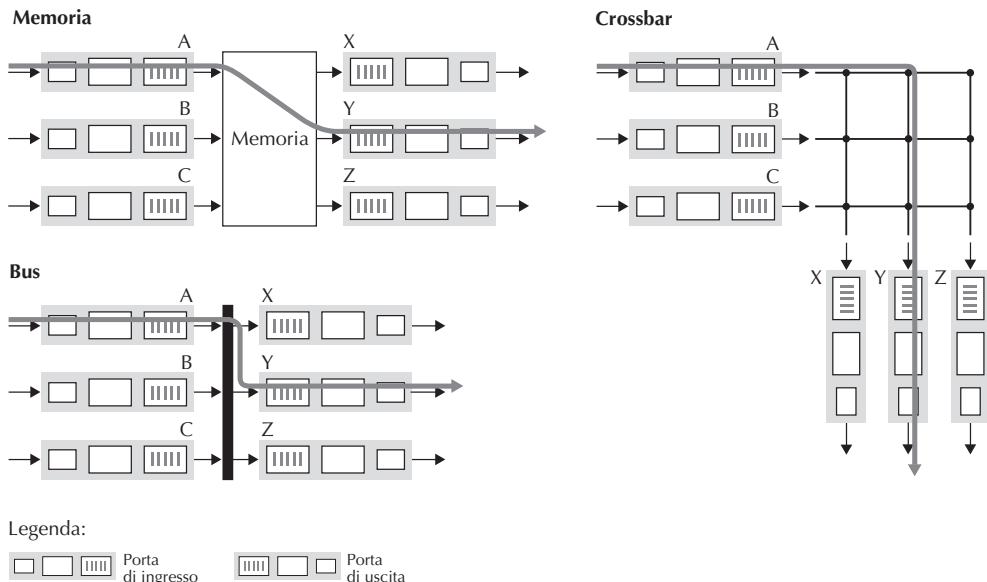
Con questa struttura il router confronta un prefisso dell’indirizzo di destinazione del pacchetto con una riga della tabella; se c’è corrispondenza, il router inoltra il pacchetto al collegamento associato. Per esempio, supponiamo che l’indirizzo di destinazione del pacchetto sia 11001000 00010111 00010110 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 del-

l'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, 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) e 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 on-line), dispositivi che filtrano i pacchetti in entrata, un pacchetto la cui intestazione corrisponde a un dato criterio (per esempio, una combina-



**Figura 4.6** Tre tecniche di commutazione.

zione 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

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 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

tro (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 2016] 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 2016] 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

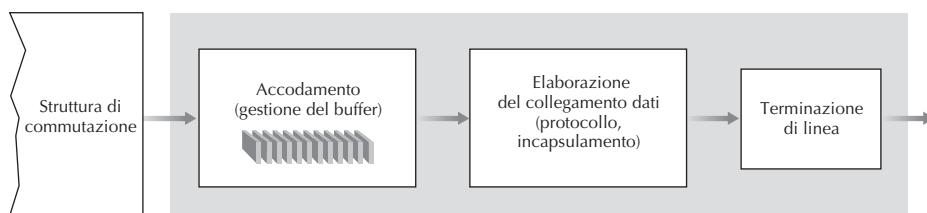
output diversi. Quindi, al contrario degli altri due approcci, la commutazione attraverso rete di interconnessione è in grado di inoltrare più pacchetti in parallelo. Una matrice di commutazione è **non-blocking**: un pacchetto in via di inoltro verso una porta di uscita non viene bloccato a meno che 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 2016] usa la commutazione attraverso rete di interconnessione; la serie Cisco 7600 [Cisco 7600 2016] 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 riassembra 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. Questo 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 aspettare 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 dalla linea.



**Figura 4.7** Elaborazione alle porte di uscita.

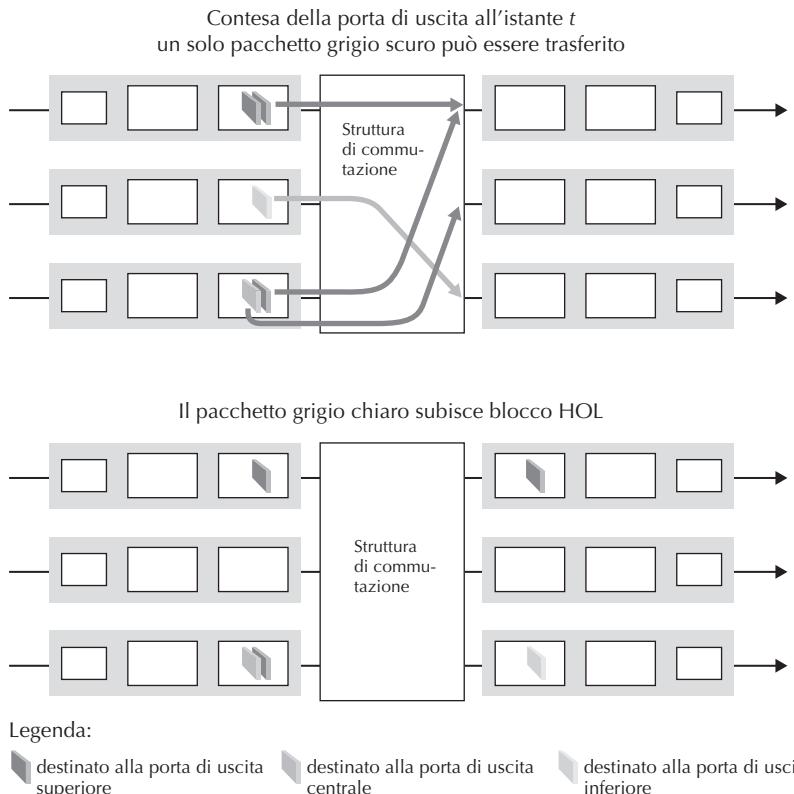
Quando queste code crescono, la memoria del router può esaurirsi e quindi può avvenire una perdita di pacchetti nel caso non vi sia memoria per immagazzinare quelli in arrivo. Nelle discussioni precedenti abbiamo detto che i pacchetti venivano “persi nella rete” o “scartati dai router”. È qui, a queste 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 sarà 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



**Figura 4.8** Blocco in testa (HOL) alla coda di input di uno switch.

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 crescerà indefinitamente (e quindi si verificheranno sostanziose perdite di pacchetti) quando il tasso di arrivo dei pacchetti sui collegamenti di ingresso raggiunge solo il 58% della loro capacità. Numerose soluzioni al problema sono trattate in [McKeown 1997].

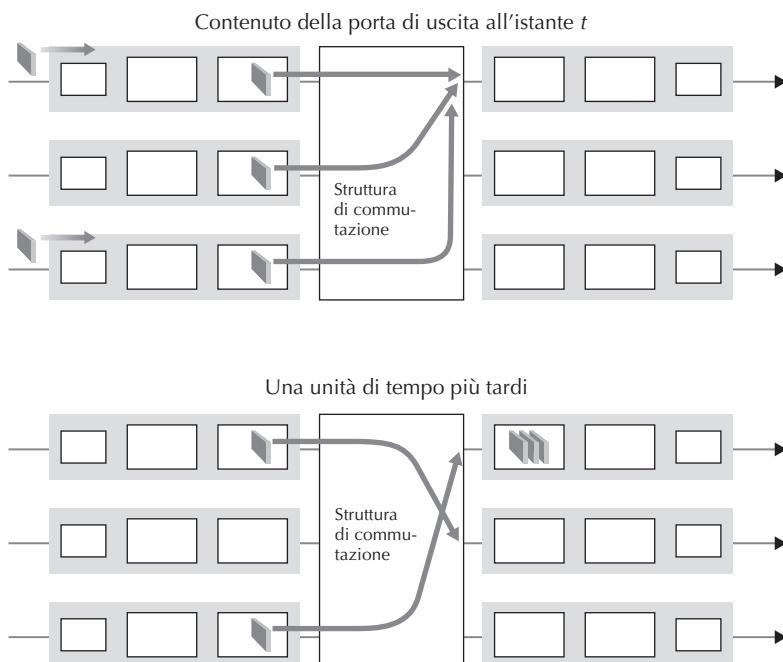
### Accodamento in uscita

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 arriveranno  $N$  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 neces-

sario 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, occorrerà 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 mittente un segnale di congestione. Sono state proposte e analizzate numerose politiche di eliminazione e marcatura dei pacchetti, tecniche che presentano il nome collettivo 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; Floyd 2016].

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 suc-



**Figura 4.9** Accodamento alle porte di uscita.

cessiva 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.

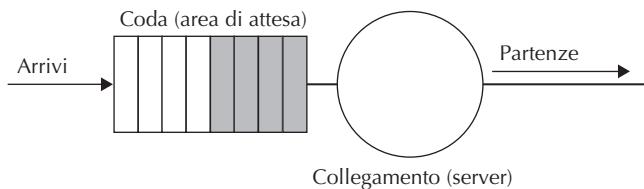
Dato che i buffer nei router sono necessari per assorbire le fluttuazioni del traffico, sorge la domanda su *quanto* debba essere la loro capacità. Per molti anni la regola empirica [RFC 3439] usata per dimensionare i buffer era che la quantità di memoria riservata al buffer ( $B$ ) dovesse essere uguale a un *RTT* medio (per esempio 250 ms) per la capacità del collegamento ( $C$ ). Questo risultato si basa sull'analisi delle dinamiche di accodamento di un numero relativamente piccolo di flussi TCP [Villamizar 1994]. Quindi, un collegamento a 10 Gbps con un *RTT* di 250 ms avrebbe bisogno di un buffer pari a  $B = RTT \times C = 2,5$  Gbit. Recenti studi teorici e sperimentali [Appenzeller 2004], tuttavia, suggeriscono che quando c'è un gran numero di flussi TCP ( $N$ ) che passano attraverso un collegamento, la quantità di buffer necessaria sia  $B = RTT \times C/\sqrt{N}$ . Con un gran numero di flussi, che tipicamente attraversano i grossi collegamenti dei router di dorsale (si veda per esempio [Fraleigh 2003]), 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.

#### 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 vengono servite a rotazione.

##### **Primo-arrivato-primo-servito (FIFO)**

La Figura 4.10 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



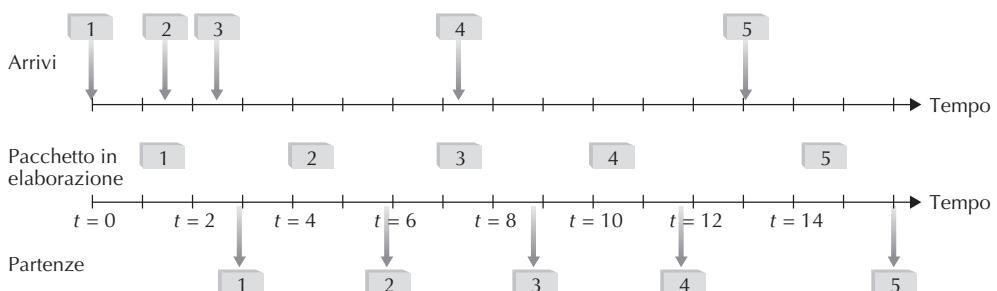
**Figura 4.10** Un modello di coda FIFO.

politiche di scarto e supporremo che il pacchetto venga rimosso solo dopo che è stato trasmesso.

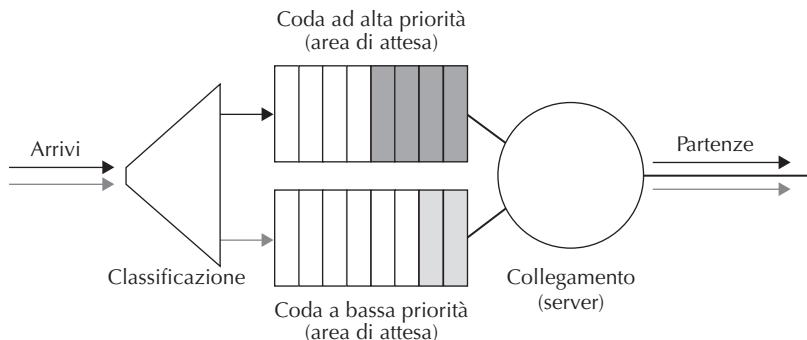
I pacchetti vengono quindi trasmessi nello stesso ordine con cui sono arrivati in coda. Un esempio familiare sono le file nei centri servizi, dove i clienti che arrivano si mettono in coda di attesa, rimangono nell'ordine di arrivo e poi vengono serviti quando viene il loro turno. La Figura 4.11 mostra 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 inviati. 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, possiamo notare che dopo la partenza del quarto pacchetto il collegamento resta inattivo fino all'arrivo del quinto.

### Code con priorità

Nel modello di accodamento con priorità (*priority queuing*) i pacchetti sono classificati in base a classi di priorità, come mostrato nella Figura 4.12. In pratica, un operatore di rete può configurare una coda in modo che i pacchetti con 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



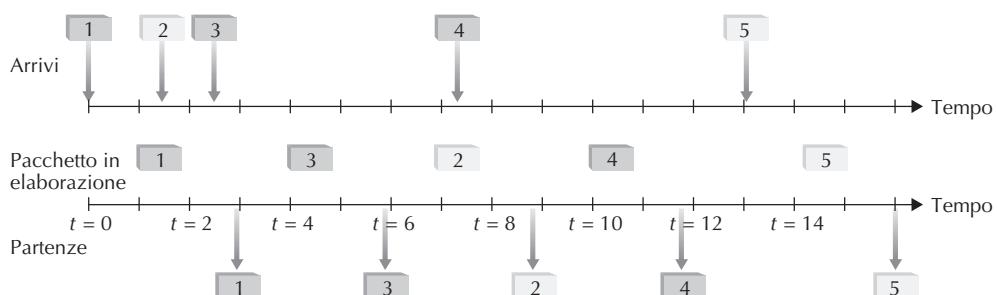
**Figura 4.11** Coda FIFO in azione.



**Figura 4.12** Coda con priorità.

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. Invece, la scelta fra i pacchetti di una classe è di solito effettuata seguendo la strategia FIFO.

La Figura 4.13 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 non prelazionabile (*nonpreemptive 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 4.13** Operazioni di una coda con priorità.

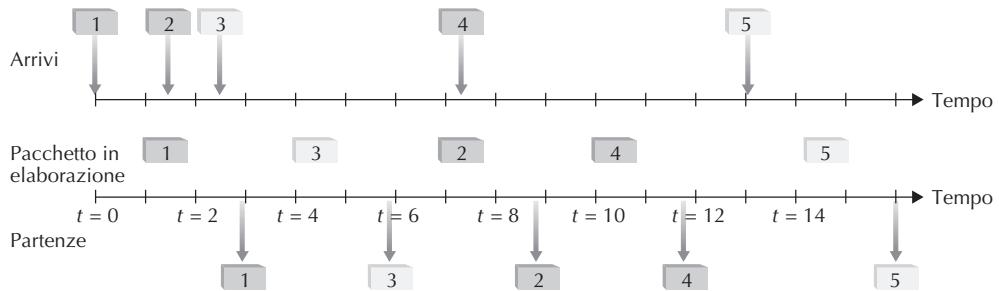


Figura 4.14 Operazioni di una coda round robin con due classi.

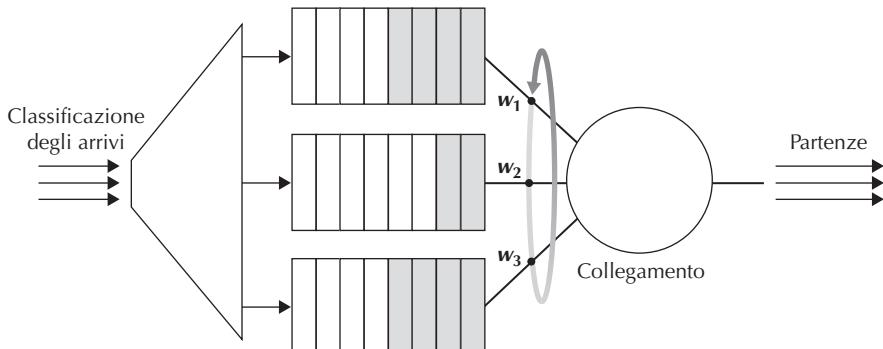
### Round robin e accodamento equo ponderato (WFQ)

Anche nella modalità di accodamento round robin (*round robin queuing*), i pacchetti sono suddivisi in classi, ma senza una rigida priorità di servizio. Infatti è prevista una sorta di sequenza ciclica tra le diverse 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 consultata quella successiva.

Un esempio di modalità round robin con due classi è mostrato nella Figura 4.14. 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 sopravvengono 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, 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; Cisco QoS 2016]. Nello schema WFQ, cui la Figura 4.15 fa riferimento, i pacchetti in arrivo sono classificati e accodati in base alla classe. Anche WFQ offre un servizio di tipo ciclico, per cui, nel caso di tre categorie, servirà 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 tutti i gruppi hanno pacchetti accodati,



**Figura 4.15** Accodamento equo ponderato (WFQ).

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].

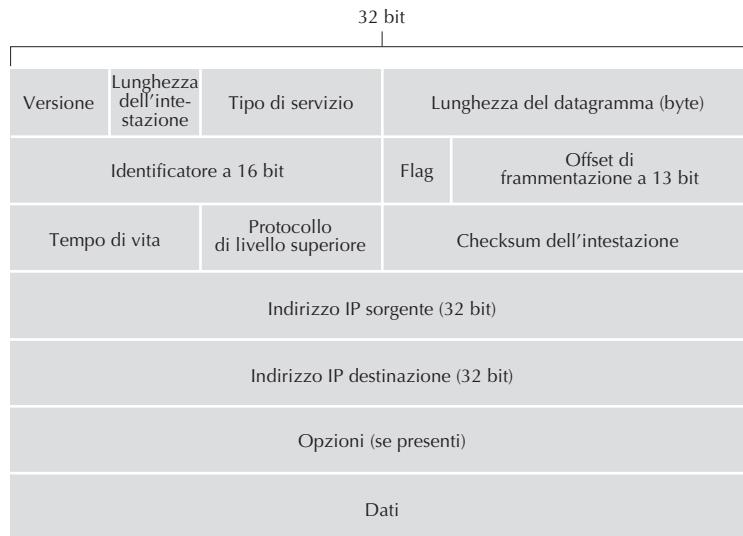
## 4.3 Il protocollo Internet (IP): IPv4, indirizzamento, IPv6 e altro ancora

Finora la nostra trattazione del livello di rete nel Capitolo 4 (il concetto di piano dei dati e di piano di controllo del livello di rete, la nostra distinzione tra inoltro e instradamento, l’identificazione di vari modelli di servizi di rete e la nostra analisi all’interno del router) spesso non ha fatto riferimento a un’architettura o 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.

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]. Alla fine nel Paragrafo 4.3.5 tratteremo la versione 6 di IP [RFC 2460; RFC 4291], indicata con IPv6 3 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



**Figura 4.16** Formato dei datagrammi IPv4.

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 4.16, i principali campi dei datagrammi IPv4 sono i seguenti:

- **Numero di versione.** Questi quattro bit, che specificano la versione del protocollo IP del datagramma, consentono al router la corretta interpretazione del datagramma; infatti, versioni diverse di IP hanno differenti formati per i datagrammi. La Figura 4.16 illustra il formato della versione corrente di IP, IPv4, mentre quello della nuova versione 6 (IPv6) verrà trattato nel Paragrafo 4.3.5.
- **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.** 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.** Questi tre campi servono per la cosiddetta frammentazione, che approfondiremo tra breve; IPv6 non consente frammentazione sui router.
- **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 Protocols 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.** 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.** Quando un host crea un datagramma, inserisce il proprio indirizzo IP nel campo indirizzo IP dell'origine e quello della destinazione nel campo indirizzo IP di destinazione. Spesso l'host sorgente determina l'indirizzo di destinazione attraverso una ricerca DNS (Capitolo 2). Tratteremo in dettaglio l'indirizzamento IP nel Paragrafo 4.3.3.
- **Opzioni.** 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 4.3.5).
- **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 ne hanno 40 byte complessivi di intestazione: 20 di intestazione IP più 20 di intestazione TCP, assieme al messaggio di livello applicativo.

### 4.3.2 Frammentazione dei datagrammi IPv4

Nel Capitolo 6 vedremo che non tutti i protocolli a livello di collegamento possono trasportare pacchetti della stessa dimensione a livello di rete. Per esempio, i frame Ethernet possono trasportare fino a 1500 byte di dati, mentre i frame di alcuni collegamenti su grandi distanze non possono trasportarne più di 576. La massima quantità di dati che un frame a livello di collegamento può trasportare è detta **unità massima di trasmissione (MTU, maximum transmission unit)**. Dato che, per il trasporto da un router a un altro, i datagrammi IP sono incapsulati in frame a livello di collegamento, la MTU di questo protocollo pone un limite rigido alla lunghezza dei datagrammi IP. Tale limite non costituisce un problema in sé, ma la difficoltà nasce dal fatto che le tratte del percorso tra mittente e destinatario possono utilizzare differenti protocolli a livello di collegamento e possono avere differenti MTU.

Per comprendere meglio questo aspetto, immaginate di essere un router che gestisce connessioni che presentano diversi protocolli a livello di collegamento e differenti MTU. Supponete di ricevere un datagramma IP da un collegamento: dalla vostra tabella di inoltro determinate il collegamento di uscita e scoprirete che questo ha MTU inferiore alla lunghezza del datagramma IP. Come dividete il datagramma nel campo dati del frame a livello di collegamento? La soluzione consiste nel frammentare i dati

del datagramma IP in due o più datagrammi IP più piccoli, detti **frammenti**, e quindi trasferirli sul collegamento di uscita.

I frammenti dovranno però essere riassemblati prima di raggiungere il livello di trasporto alla destinazione. Infatti, TCP e UDP si aspettano di ricevere segmenti completi da parte del livello di rete. I progettisti di IPv4 pensavano che riassemblare i datagrammi nei router avrebbe introdotto una complessità che ne avrebbe limitato le prestazioni. Tenendo fede al principio di mantenere semplice il nucleo della rete, i progettisti di IPv4 decisero di assegnare il compito del riassemblaggio dei datagrammi ai sistemi periferici anziché ai router interni.

Quando un host di destinazione riceve una serie di datagrammi dalla stessa origine, deve individuare i frammenti, determinare quando ha ricevuto l'ultimo e stabilire come debbano essere assemblati per formare il datagramma originario. Per consentire all'host di destinazione di svolgere questo compito, i progettisti hanno posto nell'intestazione del datagramma IP (versione 4) i campi identificazione, flag e offset di frammentazione. Quando crea un datagramma, l'host lo contrassegna con un numero identificativo e con gli indirizzi di sorgente e di destinazione. Generalmente, l'host di invio incrementa l'identificativo di ciascun datagramma che invia. Quando il router frammenta il datagramma, contrassegna i frammenti con gli indirizzi di sorgente e di destinazione e con l'identificatore numerico del datagramma originario. Quando la destinazione riceve una serie di datagrammi dallo stesso host mittente, può esaminare gli identificatori per individuare i frammenti di uno stesso datagramma. Siccome IP fornisce un servizio non affidabile, alcuni frammenti potrebbero non giungere mai a destinazione. Per questo motivo, affinché l'host di destinazione sia assolutamente certo di aver ricevuto tutti i frammenti e sia in grado di riassemblarli in modo corretto, l'ultimo ha il campo posto a 0, mentre in tutti gli altri è posto a 1. Si utilizza infine il campo di offset per specificare l'esatto ordine che i frammenti avevano originariamente all'interno del datagramma IP e per determinare se un frammento è stato perduto.

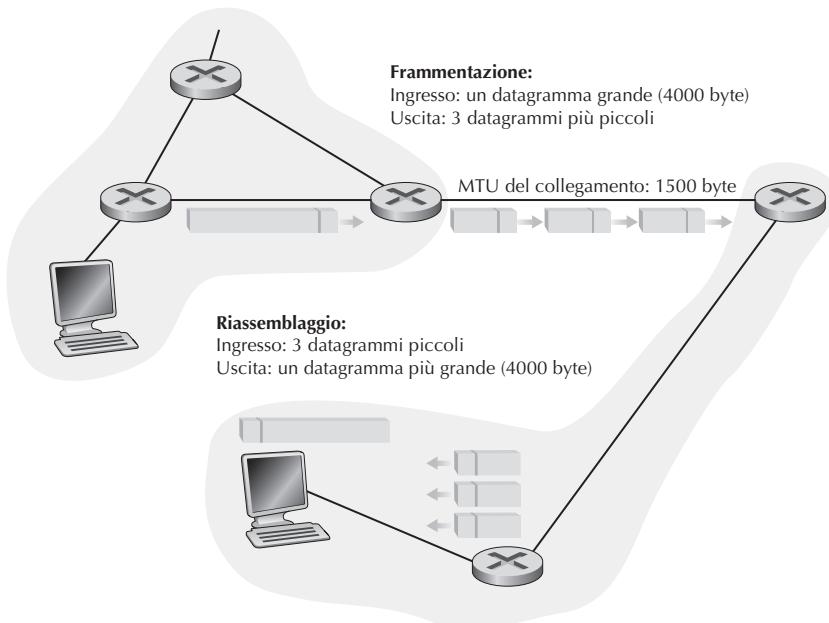
La Figura 4.17 mostra un esempio in cui un datagramma di 4000 byte (20 di intestazione IP più 3980 di dati), giunge a un router e deve essere inoltrato su un collegamento con MTU da 1500 byte. Ciò implica che i 3980 byte di dati del datagramma originario debbano essere suddivisi in tre frammenti separati, ciascuno dei quali rappresenta anch'esso un datagramma IP.

Nel sito web del libro potete trovare un'applet che genera frammenti. È sufficiente impostare dimensione, MTU e identificatore del datagramma in ingresso e l'applet provvederà a generare i frammenti.

### 4.3.3 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 collega-



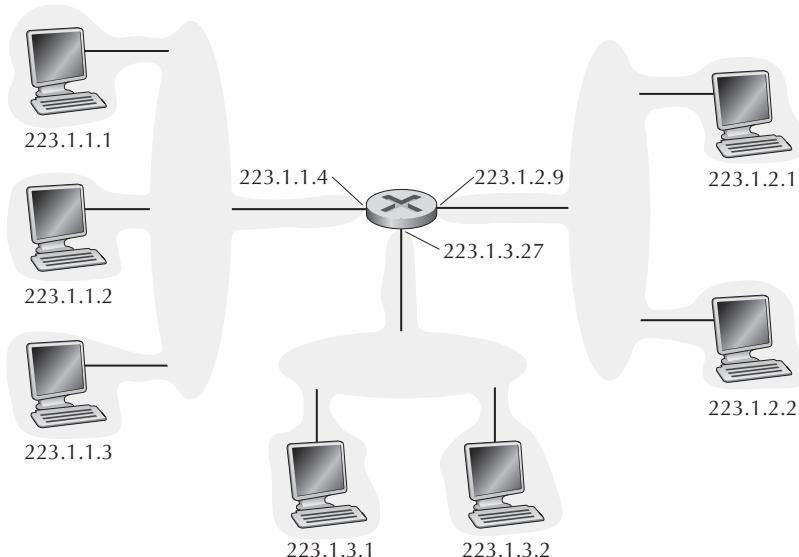
**Figura 4.17** Frammentazione e riassemblaggio IP.

mento con la rete; quando l'implementazione di IP dell'host vuole inviare un datagramma, lo fa su tale collegamento. Il confine tra host e collegamento fisico viene detto **interfaccia**. Invece, dato che il compito di un router è ricevere datagrammi da un collegamento e inoltrarli su un altro, questo 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 4.3.4). Tuttavia, tali indirizzi non possono essere scelti in modo arbitrario. Una parte dell'indirizzo di un'interfaccia è determinata dalla sottorete cui è collegata.

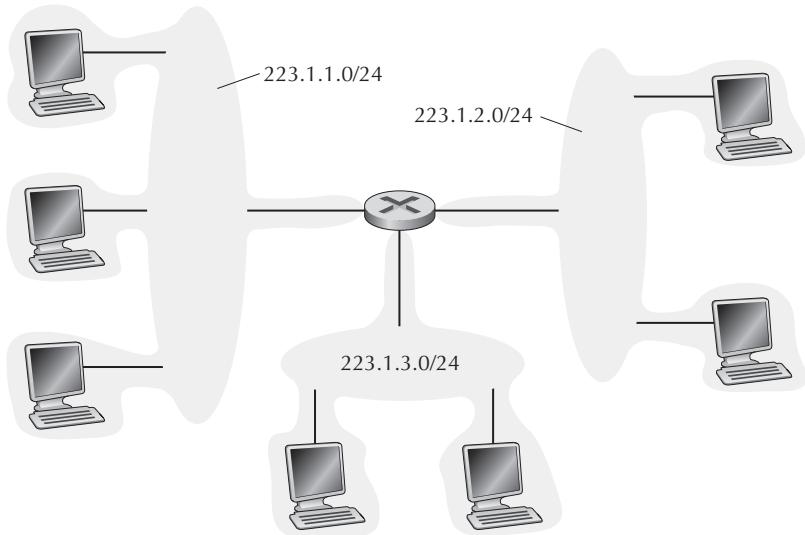


**Figura 4.18** Indirizzi delle interfacce e sottoreti.

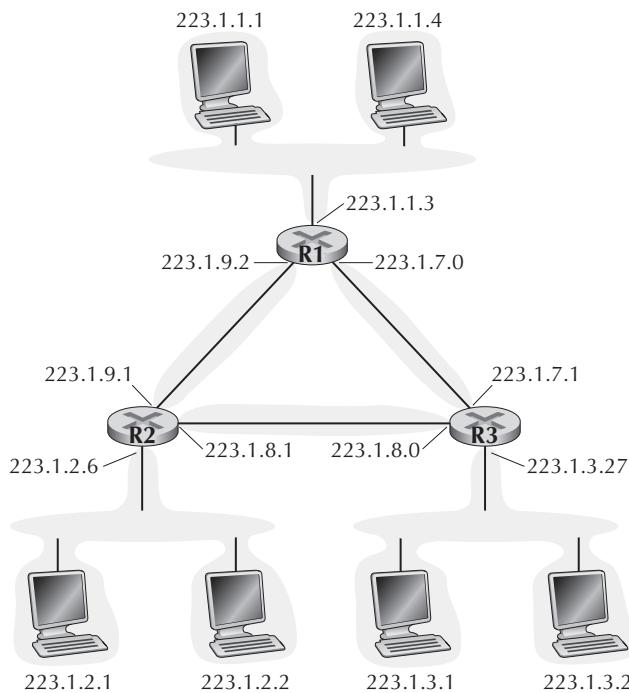
La Figura 4.18 mostra un router (con tre interfacce) che connette sette host. I tre a sinistra e l’interfaccia del router cui sono connessi hanno un indirizzo IP della forma 233.1.1.xxx: ossia, i 24 bit più a sinistra nell’indirizzo IP sono identici. Le quattro interfacce sono interconnesse da una rete 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 (Capitolo 7). 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. 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. 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 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



**Figura 4.19** Indirizzi di sottorete.



**Figura 4.20** Tre router che interconnettono sei sottoreti.

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.

*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 presenterà più sottoreti i cui dispositivi avranno 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) 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 address**.

**sing**, 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 crescita 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 65.534 host, risultavano sovradimensionate. 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 datagramma 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.

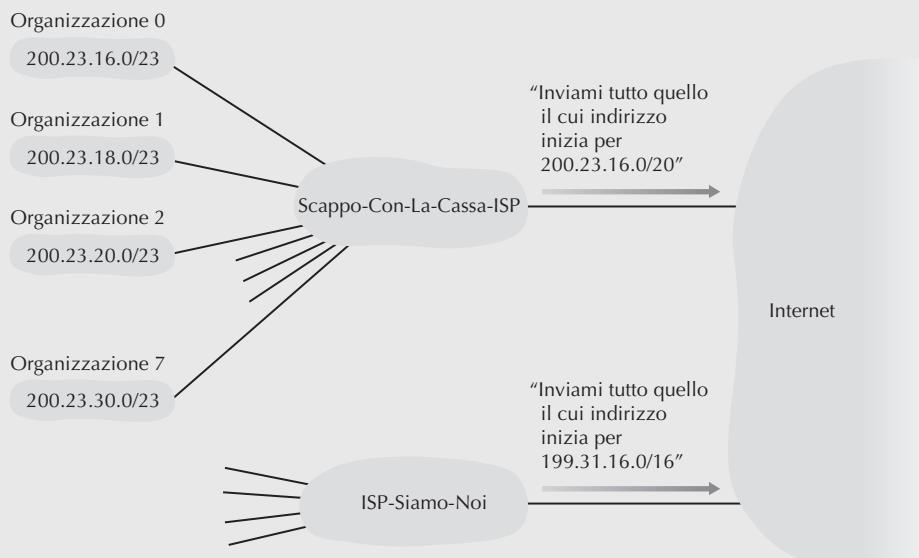
Blocco dell'ISP	200.23.16.0/20	<u>11001000</u> 00010111 00010000 00000000
Organizzazione 0	200.23.16.0/23	<u>11001000</u> 00010111 <u>00010000</u> 00000000
Organizzazione 1	200.23.18.0/23	<u>11001000</u> 00010111 00010010 00000000
Organizzazione 2	200.23.20.0/23	<u>11001000</u> 00010111 00010100 00000000
...	...	...
Organizzazione 7	200.23.30.0/23	<u>11001000</u> 00010111 00011110 00000000

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 2016], che opera sulla base delle linee guida

**BOX 4.1****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 4.18, 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*).

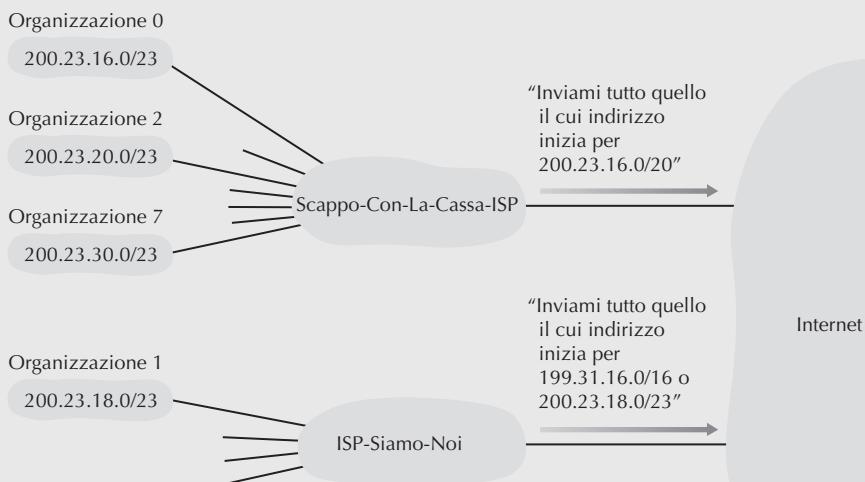
L'aggregazione di indirizzi funziona molto bene quando gli indirizzi sono allocati in blocchi agli ISP e da questi alle organizzazioni loro clienti. Ma che cosa avviene quando gli indirizzi non sono allocati in modo gerarchico? Che cosa succederebbe, per esempio, se Scappo-Con-La-Cassa-ISP acquisisse un provider chiamato ISP-Siamo-Noi e poi facesse connettere l'Organizzazione 1 a Internet attraverso la propria sussidiaria ISP-Siamo-Noi? Come mostrato nella Fi-



**Figura 4.21** Indirizzamento gerarchico e aggregazione di indirizzi.

tracciate in [RFC 2050]. Il ruolo di ICANN, un'organizzazione senza scopo di lucro [NTIA 1998], 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 assieme l'Address Supporting Organization

gura 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.

di ICANN [ASO-ICANN 2016]), i quali si occupano dell'allocazione e della gestione degli indirizzi all'interno delle rispettive regioni.<sup>1</sup>

<sup>1</sup> L'autorità per la zona europea è il RIPE, che si occupa anche del Medio Oriente e di alcune parti dell'Asia centrale (*N.d.R.*).

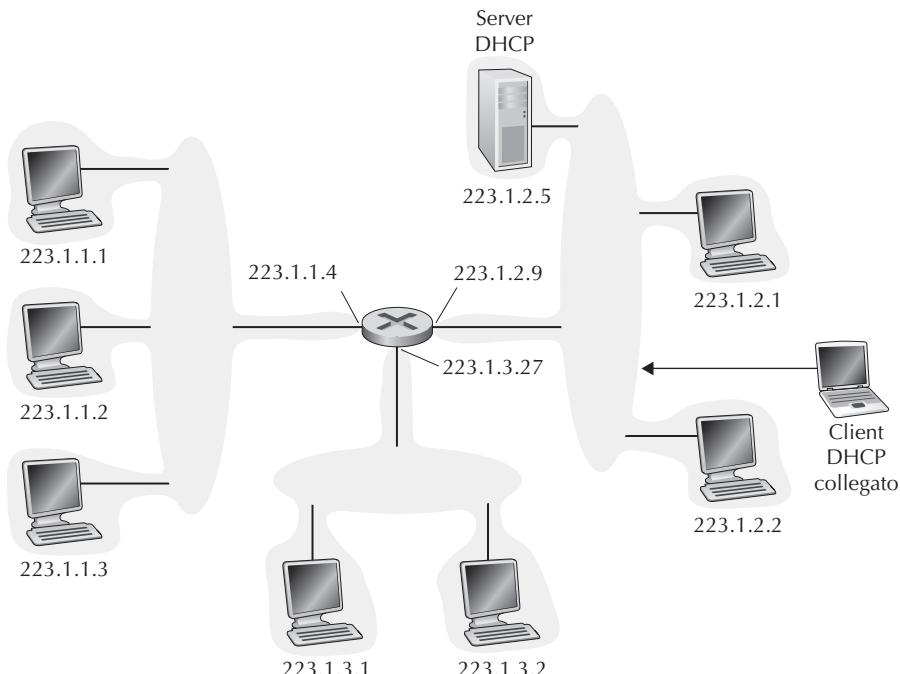
### Come ottenere l'indirizzo di un host: DHCP

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.

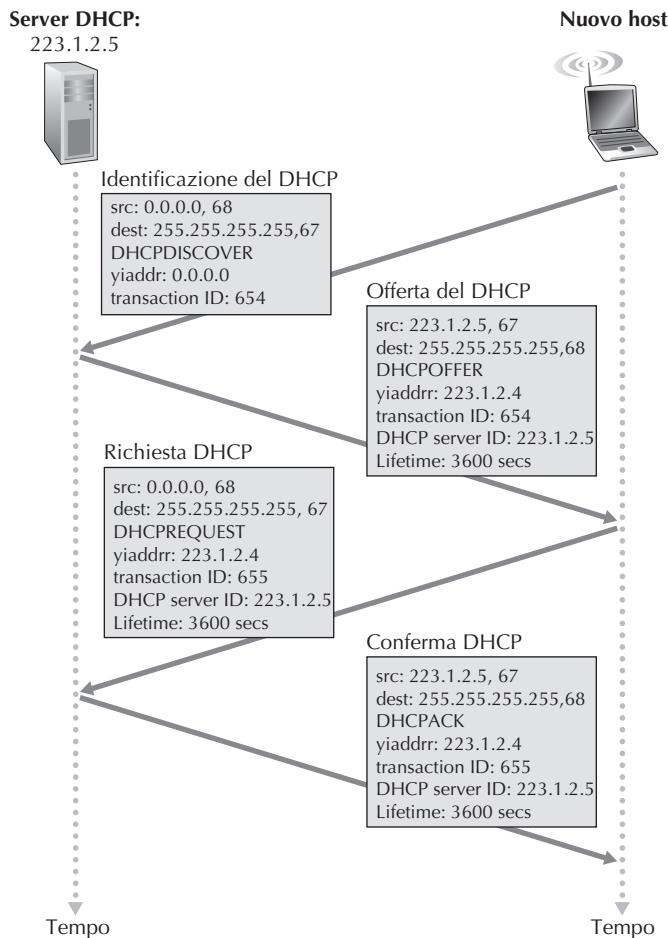
DHCP è un protocollo client-server. Un client è di solito un host appena connesso che desidera ottenere informazioni sulla configurazione della rete, non soltanto su uno specifico indirizzo IP. Nel caso più semplice, ogni sottorete (nel senso dell'indirizzamento nella Figura 4.20) dispone di un server 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:



**Figura 4.23** Scenario del protocollo DHCP client–server.

- **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 incapsulato 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 of-



**Figura 4.24** Interazione client–server DHCP

ferta 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 comunemente 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.

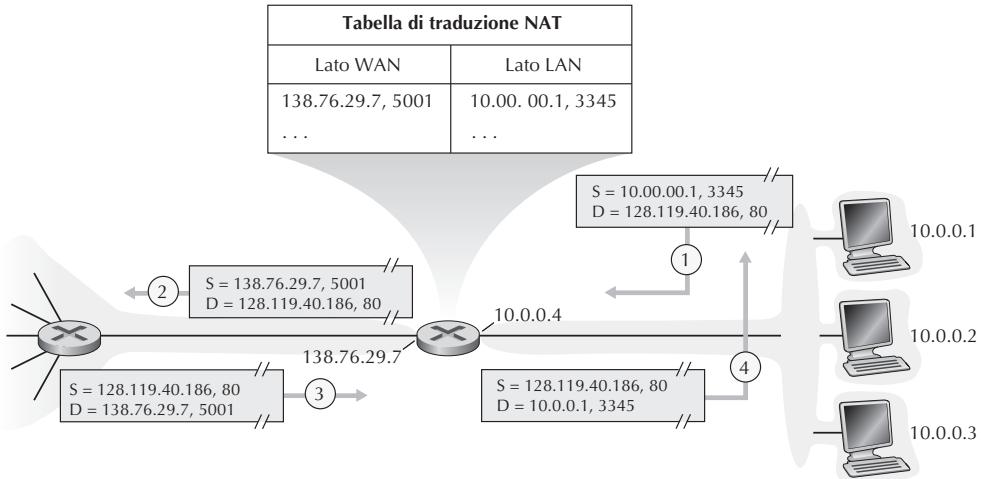
Tuttavia, dal punto di vista della mobilità, DHCP non è privo di difetti. Quando un nodo si connette a una nuova sottorete, DHCP gli rilascia un nuovo indirizzo IP; non si può quindi mantenere una connessione TCP a un'applicazione remota, spostandosi il nodo mobile da una sottorete a un'altra. Nel Capitolo 6 esamineremo gli indirizzi IP mobili: una recente estensione delle infrastrutture IP consente a un nodo mobile di conservare lo stesso indirizzo permanente anche spostandosi tra sottoreti diverse. Ulteriori dettagli su DHCP si possono trovare in [Droms 2002] e [dhc 2016]. Un'implementazione di riferimento di DHCP che mette a disposizione il codice sorgente è reperibile da ISC (Internet System Consortium) [ISC 2016].

#### 4.3.4 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. Di conseguenza, al crescere della LAN dovrebbe esserne allocato 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; Cisco NAT 2016].

La Figura 4.25 mostra l'attività di un router abilitato al NAT, con un'interfaccia 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 private, molte delle quali usano un identico spazio di indirizzamento, 10.0.0.0/24, per scambiare pacchetti fra i loro dispositivi. Ovviamente, quelli inviati 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



**Figura 4.25** NAT.

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 **tabella 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] e Universal Plug and Play (UPnP), un protocollo che consente a un host di individuare e configurare un NAT vicino [UPnP Forum 2016].

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 introduzione di nodi né modifica di indirizzi IP e di numeri di porta. Ma che piaccia o meno NAT è diventato un componente importante di Internet, così come altri dispositivi, chiamati **middlebox** [Sekar 2011], che operano a livello di rete, ma hanno funzionalità diverse dai router. I middlebox non eseguono il tradizionale inoltro dei pacchetti, ma effettuano altre funzioni, tra le quali NAT, bilanciamento di flussi e firewall. Il paradigma dell'inoltro generalizzato, che affronteremo brevemente nel Paragrafo 4.4, permette di eseguire tutte queste funzionalità, insieme all'inoltro tradizionale, in un unico modo integrato.

### 4.3.5 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

**BOX 4.2** **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 attaccanti, 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*).

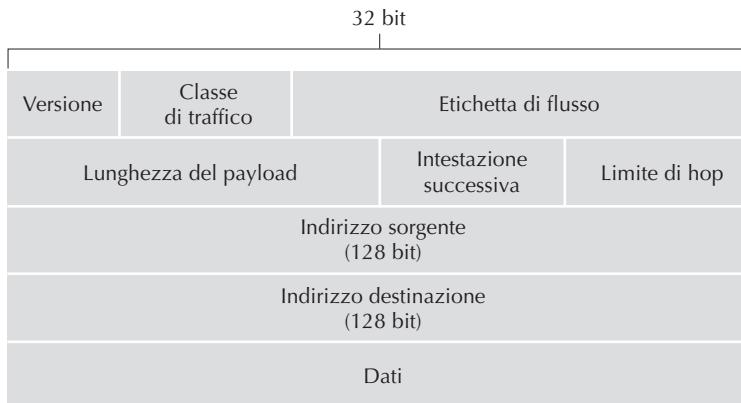
Come amministratore di rete, in primo luogo, potete provare a installare un firewall tra la vostra rete e Internet. Molti degli odierni router di accesso hanno le funzionalità dei firewall. I firewall controllano i datagrammi e i campi dell'intestazione dei segmenti in essi contenuti, impedendo l'accesso 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 attaccante 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 Capitolo 8 on-line 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 attaccanti 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.

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 blocchi [Huston 2011a]. Si consulti [Richter 2015] 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



**Figura 4.26** Formato dei datagrammi IPv6.

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 divenuto IPv5, ma questo venne successivamente scartato. Un'eccellente fonte d'informazione su IPv6 è [Huitema 1998].

### 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.** Una serie di campi IPv4 è stata eliminata o resa opzionale. 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 4.16 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 4.16) 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.

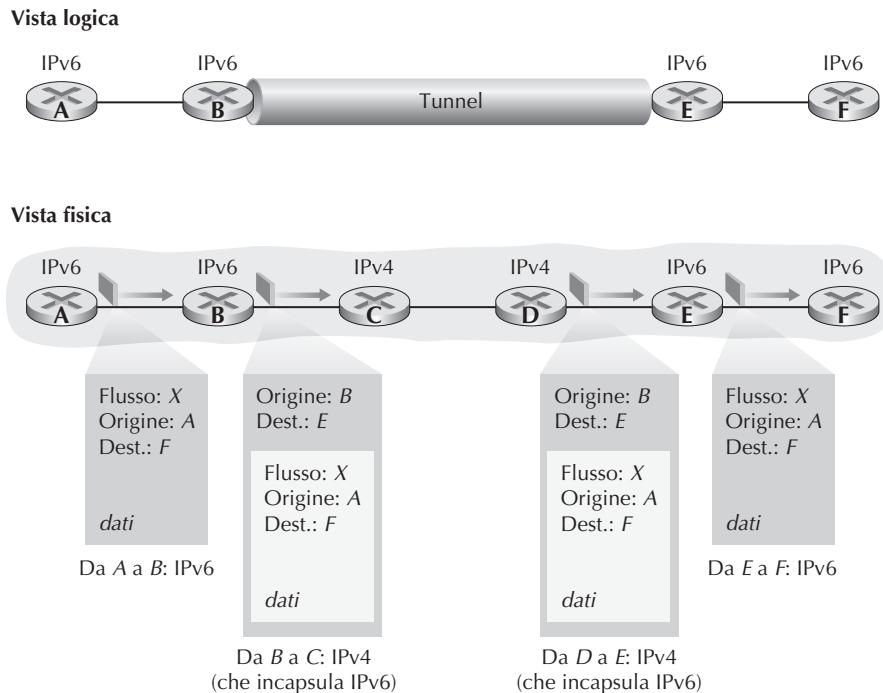
- **Checksum dell’intestazione.** Dal momento che i protocolli Internet a livello di trasporto (per esempio, TCP e UDP) e di collegamento (per esempio, Ethernet) calcolano un loro checksum, i progettisti di IP hanno probabilmente ritenuto questa funzionalità talmente ridondante nel livello di rete da 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, 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



**Figura 4.27** Tunneling.

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 instraderrà 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 2015]. D'altro canto Google riporta che circa solo l'8% dei client che si connettono ai suoi servizi utilizzano IPv6 [Google IPv6 2015]; 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 2016] 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 penetrazione molto modesta.

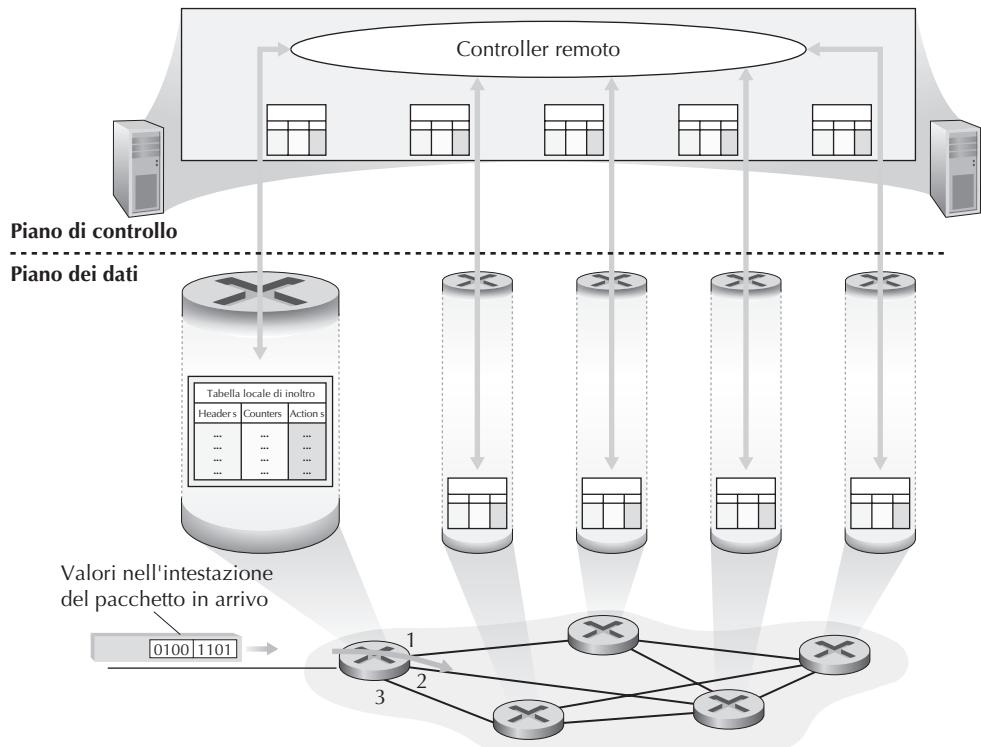
Tra questi protocolli includiamo IPv6, i protocolli multicast e quelli di prenotazione delle risorse (si veda il materiale supplementare del libro on-line per questi ultimi due). Infatti introdurre nuovi protocolli nel livello di rete è come sostituire le fondamenta di una casa; è difficile farlo senza 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

Nel Paragrafo 4.2.1 si è visto che la decisione di inoltro presso un router in Internet si basa tradizionalmente solo sull'indirizzo di destinazione del pacchetto. Tuttavia, nel precedente paragrafo, abbiamo anche visto che stiamo assistendo a una proliferazione di middlebox che effettuano molte funzioni del livello 3. I dispositivi NAT riscrivono nell'intestazione gli indirizzi IP e i numeri di porta. I firewall bloccano il traffico basandosi sui valori dei campi dell'intestazione o reindirizzano i pacchetti per elaborazioni supplementari, come nel caso del DPI (*deep packet inspection*). I load-balancer inoltrano i pacchetti che richiedono un dato servizio (per esempio, una richiesta HTTP) a uno dei server che forniscono tale servizio. Il documento [RFC 3234] enumera una serie di funzioni dei middlebox.

Questa proliferazione di middlebox, switch di livello 2 e router di livello 3 [Qazi 2013], ognuno con i suoi hardware, software e interfacce di gestione specializzate, ha sicuramente causato un bel mal di testa a molti operatori di rete. Tuttavia, recenti progressi nel software-defined networking hanno promesso e stanno ora implementando un approccio unificato alla fornitura di molte di queste funzioni del livello di rete e anche del livello di collegamento in modo moderno, elegante e integrato.

Ricordiamo dal Paragrafo 4.2.1 che l'inoltro basato sulla destinazione è caratterizzato da due passi: ricerca dell'indirizzo IP di destinazione (“match”), quindi 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 “match” può essere effettuato su più campi dell'intestazione associati a differenti protocolli corrispondenti a diversi livelli nello stack dei protocolli. L’“action” può includere l'inoltro del pacchetto a una o più porte di uscita (come l'inoltro basato sulla destinazione), il bilancio di carico dei pacchetti attraverso più interfacce di uscita che portano a un servizio (come nel load-balancing), la riscrittura dei valori dell'intestazione (come in NAT), bloccare/scartare un pacchet-



**Figura 4.28** Inoltro generalizzato: ogni packet switch ha una tabella match-action calcolata e distribuita da un controller remoto.

to (come in un firewall), inviare 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 per esempio si vede nella Figura 4.2), nella pratica sono implementate 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

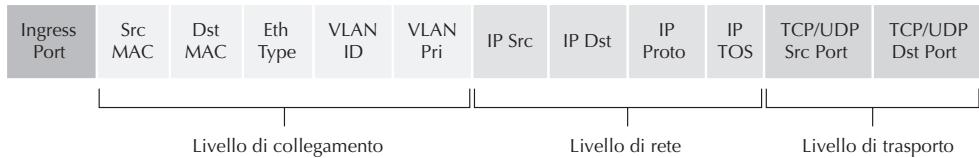
sulla destinazione mostrato nelle Figure 4.2 e 4.3 e l'inoltro generalizzato mostrato nella Figura 4.28?

La trattazione seguente sull'inoltro generalizzato sarà basata su OpenFlow [McKeown 2008; OpenFlow 2009; Casado 2014; Tourrilhes 2014], uno standard di successo, pioniere del paradigma match-action così come in generale della rivoluzione SDN [Feamster 2013]. Considereremo dapprima OpenFlow 1.0 che introduce le funzionalità 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 2016].

Ogni occorrenza (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 verrà 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 scartato 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 in un insieme di packet switch della rete [Casado 2014].



**Figura 4.29** Campi dell'intestazione del pacchetto per l'operazione di match in una tabella dei flussi OpenFlow 1.0.

#### 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 il principio di stratificazione visto 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 ricezione 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 [Bosschart 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 confrontati.

Le occorrenze della tabella dei flussi possono essere anche wildcard. 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 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 generalizzate; 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 [OpenFlow 2009; ONF 2016].

#### 4.4.2 Action

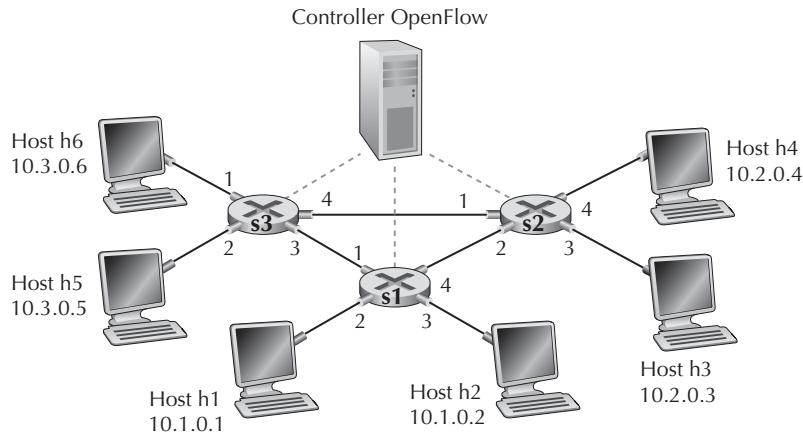
Come mostrato nella Figura 4.28, ogni occorrenza della tabella dei flussi ha una lista di azioni che determinano l'elaborazione da effettuare su un pacchetto che le corrisponde. In caso vi siano molteplici azioni, vengono effettuate nell'ordine specificato nella lista.

Di seguito sono elencate alcune delle più importanti azioni possibili.

- **Inoltro (forwarding).** Un pacchetto in entrata può essere inoltrato ad una particolare porta di uscita, inviato in broadcast a tutte le porte eccetto quella da cui è entrato o inviato in multicast ad un insieme selezionato di porte. Il pacchetto può essere incapsulato ed 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.



**Figura 4.30** Rete OpenFlow match-action con tre packet switch, 6 host e un controller OpenFlow.

### 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)
...	...

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 ritorneremo 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.

## 4.5 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 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 possano 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, sottile e interessante di quanto ci aspettassimo.

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?

## 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.

	<b>Bit rate</b>	<b>Byte trasferiti in 67 min</b>
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 remote, 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)

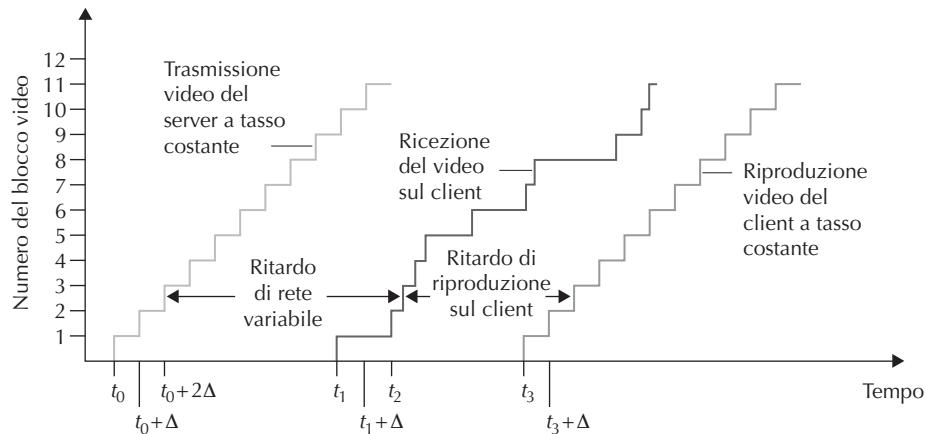
Questa terza classe di applicazioni è simile alle tradizionali trasmissioni radiotelevisive, a parte il fatto che avviene su Internet. Consente agli utenti di ricevere in diretta trasmissioni radio o televisive (come eventi sportivi o giornalistici) provenienti da ogni parte del mondo. Oggi ci sono migliaia 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 fissato 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 o 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 Streaming 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 Streaming 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.

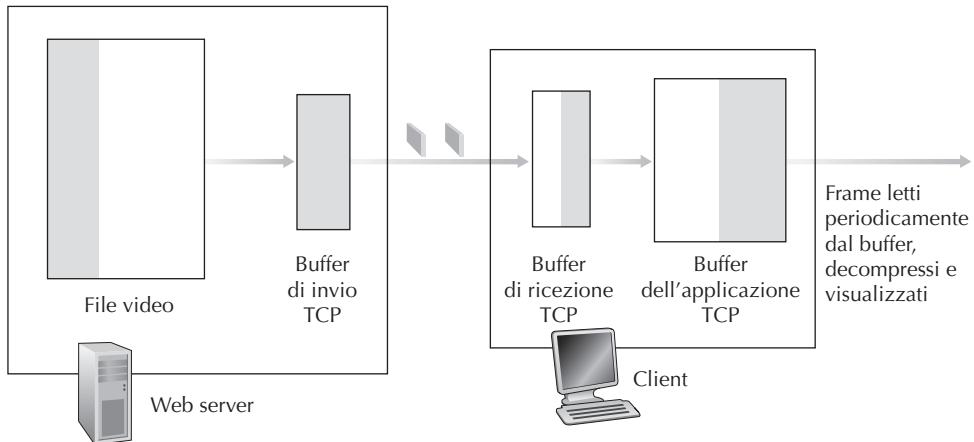
Come visto nel Capitolo 3 il tasso di trasmissione da server a client durante il trasferimento di un file su TCP può variare in modo significativo a causa dei meccanismi di controllo di congestione di TCP. Il tasso di trasmissione varia spesso con un comportamento a dente di sega tipico del controllo 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 oggi 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

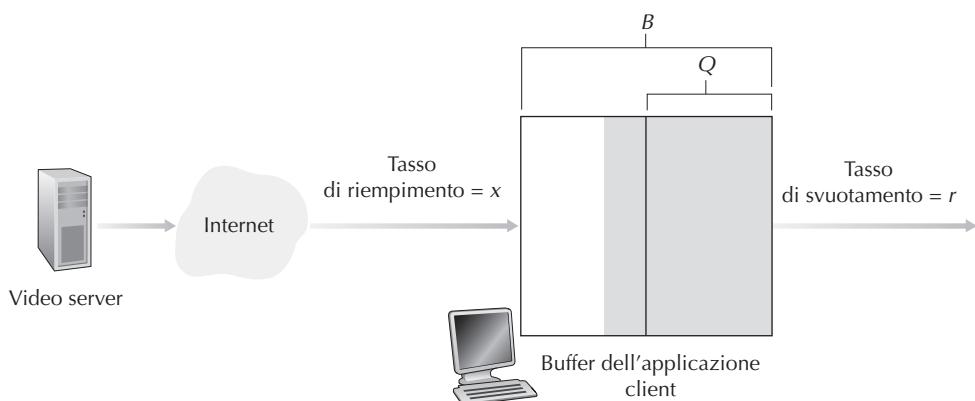
anch’esso pieno. Una volta che il buffer di ricezione TCP del client diventa pieno, i byte non possono più essere rimossi dal buffer di invio TCP del client, che diventa anch’esso pieno. A questo punto il server non può più inviare byte nella socket. Quindi quando l’utente mette in pausa il video, il 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. Supponete



**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  $t_f$  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, traffico permettendo. 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.

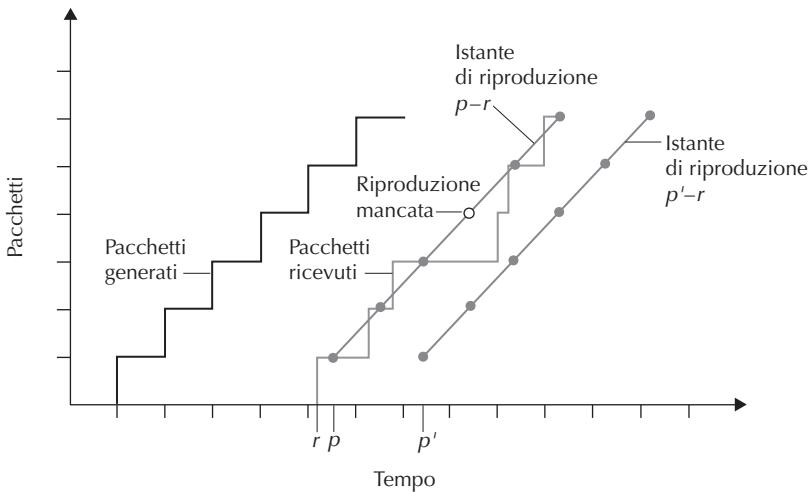
- *Facendo precedere i blocchi da una marcatura temporale.* Il trasmittente contrassegna ciascun blocco con l'indicazione dell'istante in cui è stato generato.
- *Inserendo un ritardo nella riproduzione del blocco al suo ricevimento, che deve essere abbastanza lungo da consentire la ricezione della maggior 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 trasmettente 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:

$$p_i = t_i + d_i + K v_i$$

dove  $K$  è una costante positiva (per esempio,  $K = 4$ ). Lo scopo del termine  $K v_i$  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.

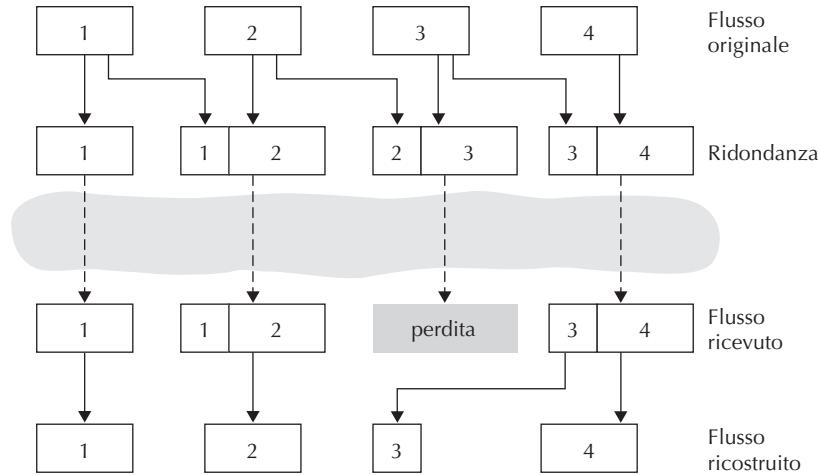
### 9.3.3 Recupero dei pacchetti persi

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.

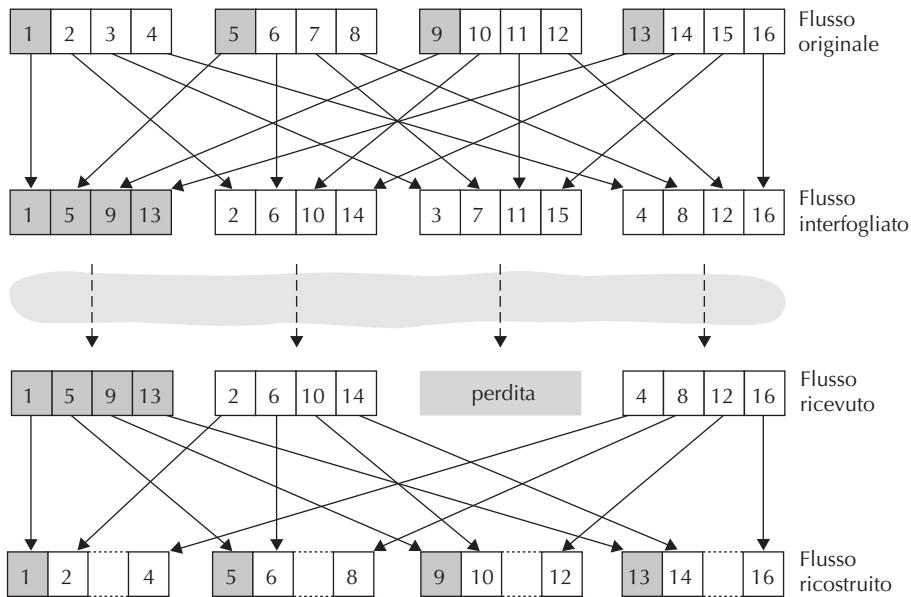


Figura 9.6 Invio di audio interfogliato.

### 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].

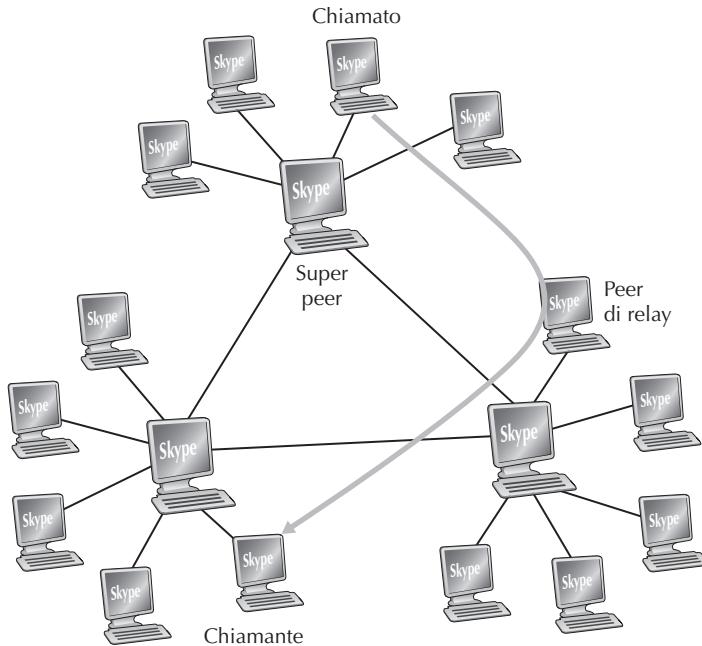
### 9.3.4 Un caso di studio: VoIP con Skype

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 tramite 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 estrae 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 estrae 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 trmissivo 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.

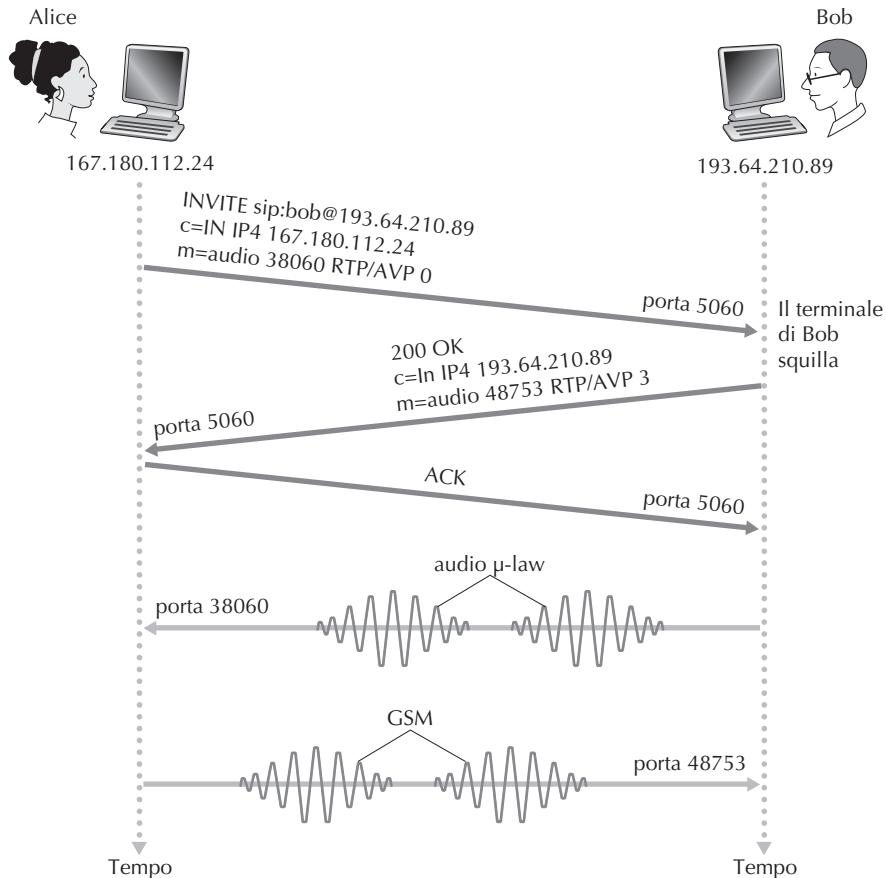
### Inizializzazione di una chiamata verso un indirizzo IP

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 dovrebbero 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 disposesse 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.hereway.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.

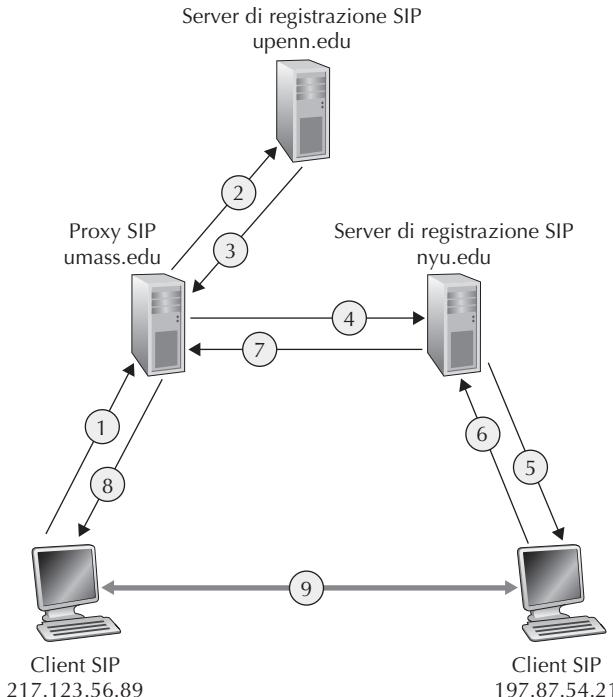
A questo punto vi starete probabilmente chiedendo in che modo il server proxy riesce a individuare l’attuale indirizzo IP di sip:bob@domain.com. Occorre sapere che a ciascun utente è associato un **server di registrazione SIP** (*SIP registrar*) al quale l’applicazione SIP su un dispositivo, quando viene 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

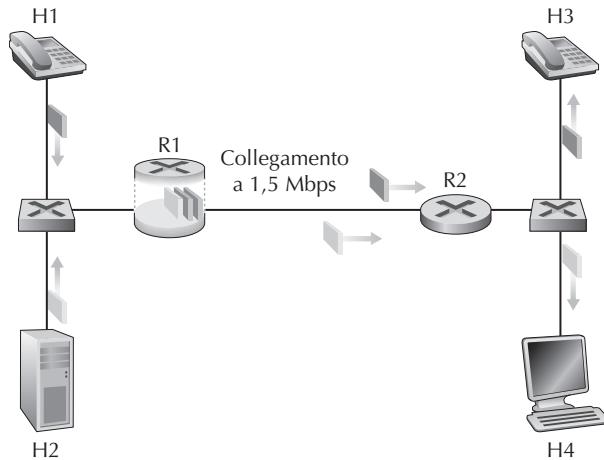
economiche e organizzative. Da un punto di vista economico, gli utenti sarebbero disposti a pagare i propri ISP, in modo tale che questi ultimi installino banda sufficiente per supportare applicazioni multimediali in Internet con un servizio best-effort? Le questioni amministrative sono ancora più scoraggianti. 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.

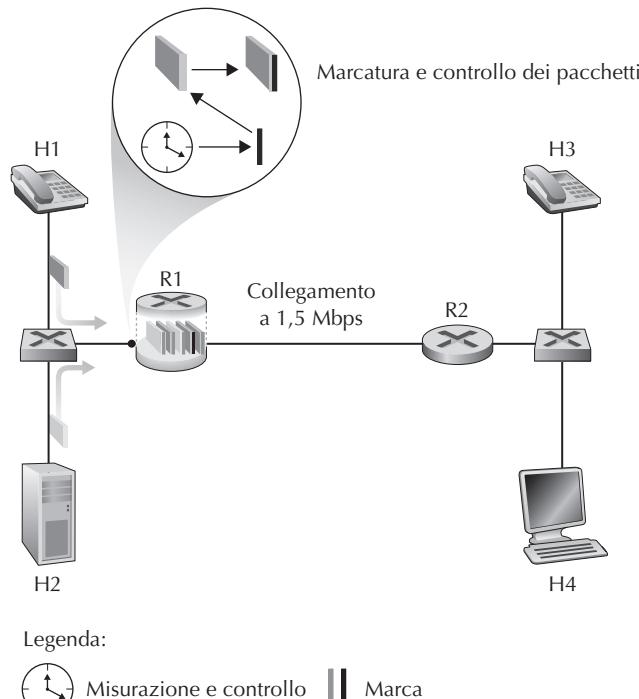
Si noti che, nonostante questo esempio consideri flussi multimediali e non in competizione, lo stesso principio può essere applicato a tipi di servizio differenziati dalla tariffazione: un sistema di marcatura dei pacchetti è ancora necessario per indicare a quale classe un pacchetto appartiene.

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 assicurarcene 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

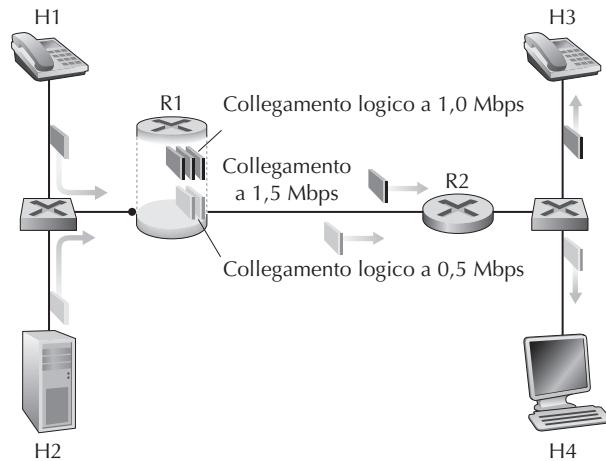


**Figura 9.12** Controllo del profilo (e marcatura) dei flussi di traffico audio e HTTP.

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.

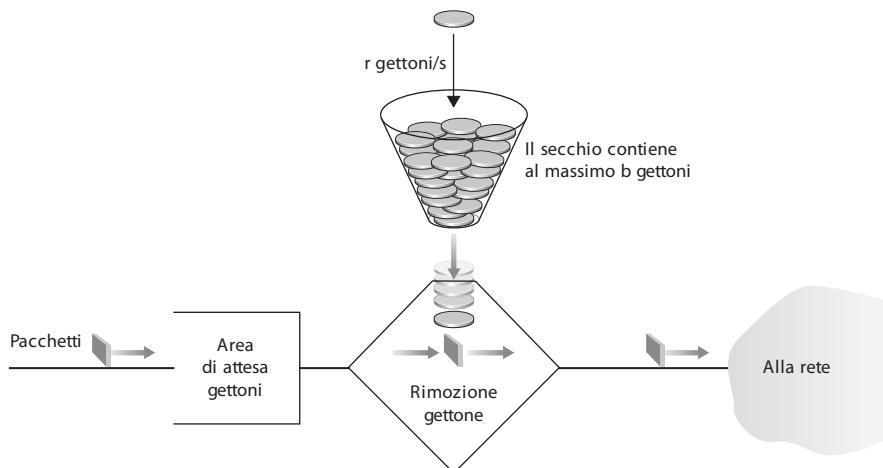
- **Tasso medio.** La rete può voler limitare la frequenza trasmissiva media, ossia il numero di pacchetti in un dato intervallo di tempo a lungo termine, alla quale il flusso può essere inviato. Un aspetto cruciale è qui rappresentato dall'intervallo di tempo rispetto al quale calcolare il tasso medio. Un flusso con una frequenza trasmissiva 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 trasmittivo 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.

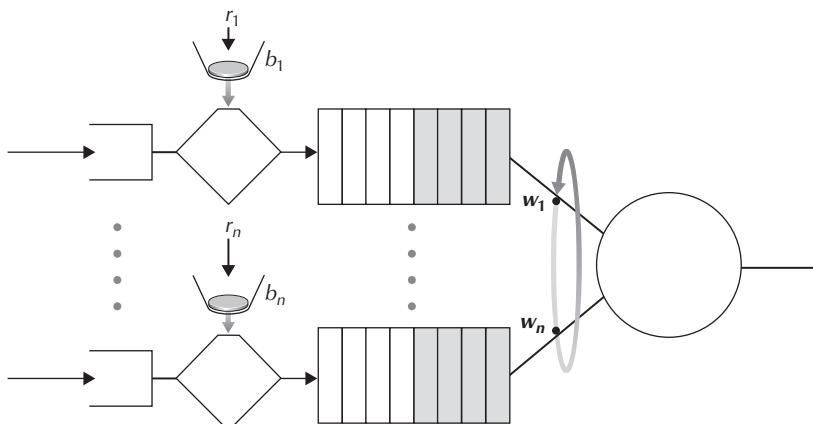
### Leaky bucket + WFQ = ritardo massimo dimostrabile in una coda

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, dove

$$d_{\max} = \frac{b_1}{R \cdot w_1 / \sum w_j}$$

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.

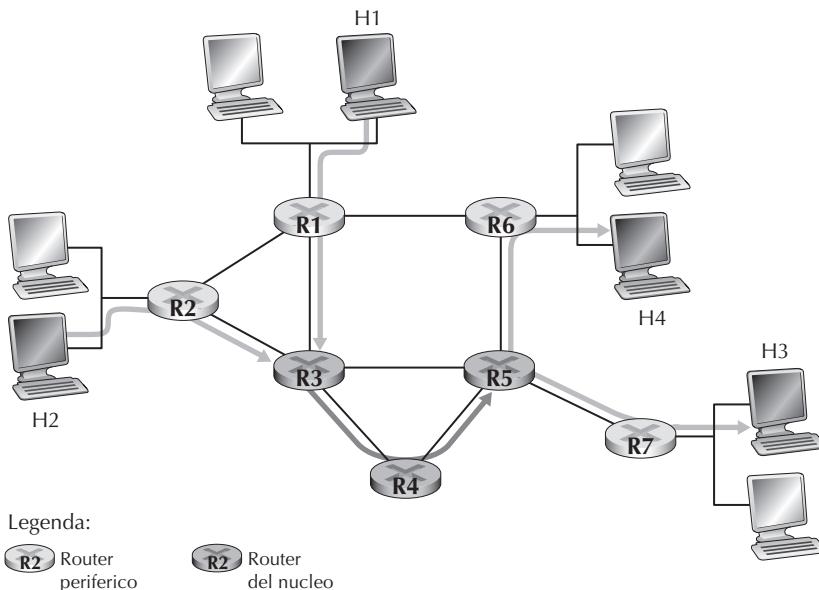
### 9.5.3 Diffserv

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.

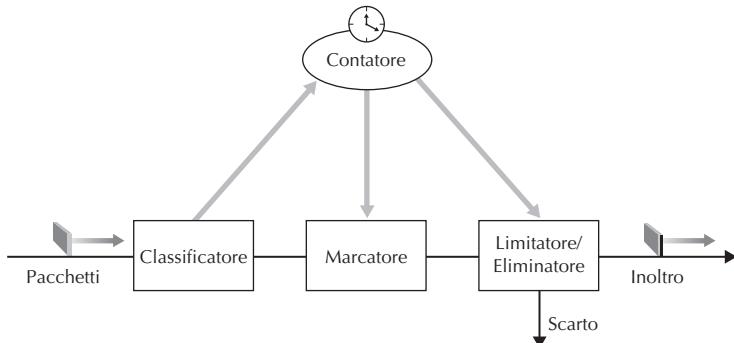
definizioni del campo ToS in IPv4 e del campo relativo alle classi di traffico di IPv6, discusse nel Capitolo 4. Per esempio (Figura 9.16), i pacchetti spediti da H1 a H3 potrebbero essere contrassegnati con R1, e quelli da H2 a H4 con R2. Diverse classi di traffico riceveranno servizi diversi nel nucleo 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 trasmisivo 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.

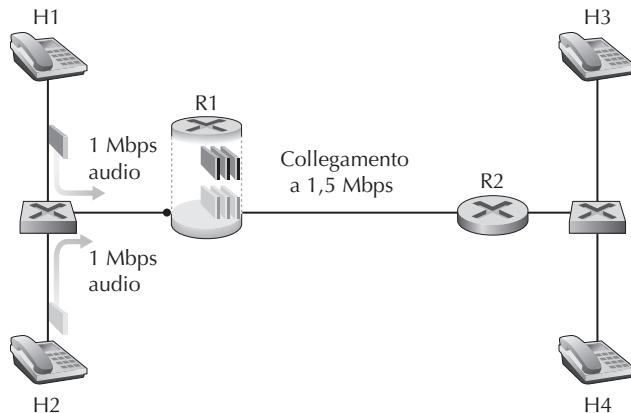
Attualmente sono state definite due tipologie di PHB: expedited forwarding (EF, inoltro rapido) [RFC 3246] e assured forwarding (AF, inoltro assicurato) [RFC 2597]. **Expedited forwarding** specifica che il tasso trasmissivo di una classe di traffico dal router deve essere uguale o superiore a un valore 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 trasmissivi 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.

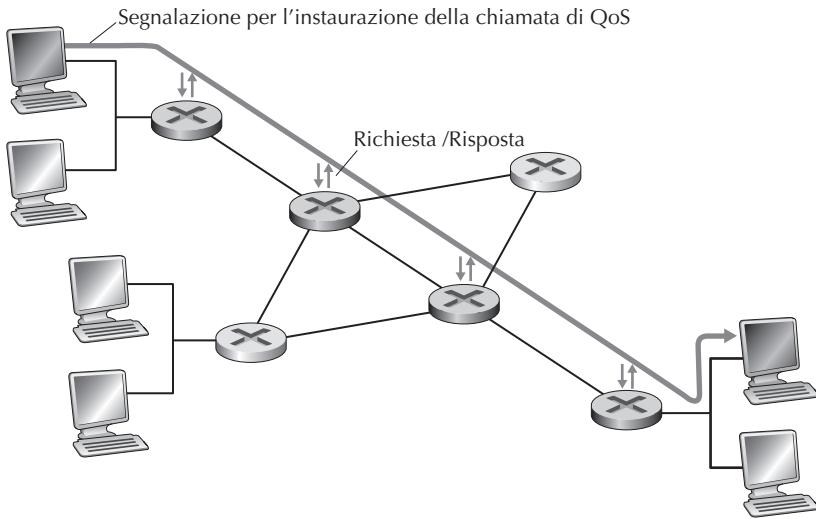
Ammettendo o bloccando esplicitamente un flusso, in base ai requisiti sulle risorse e ai requisiti delle sorgenti dei flussi già ammessi, la rete può garantire a un flusso ammesso che sarà in grado di ricevere la QoS richiesta. Occorre quindi che il flusso indichi quali sono i requisiti di servizio che 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 sottrarre 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.