Tecnologie di rete e Internet

Augusto Ciuffoletti

Da Ethernet ad ATM: le principali tecnologie di interconnessione

Una guida ai protocolli Internet aderente alla documentazione ufficiale

Le primitive di accesso a Internet in BSD-UNIX

19 settembre 2007

II

Sommario

Prefazione

Ho affrontato la redazione di questo testo dopo alcuni anni di esperienza di insegnamento di corsi sulle reti di calcolatori, con riferimento ai protocolli Internet, presso il Dipartimento di Informatica dell'Università di Pisa.

In un caso si è trattato di un corso avanzato, che faceva seguito ad un corso tenuto da un altro docente che introduceva lo strato di data link ed alcune tecniche di trasporto. L'obiettivo del corso affidato a me era di collocare quelle nozioni in un ambito concreto.

Qualche tempo dopo mi è stato affidato anche il corso introduttivo, che ho riorganizzato portando parte del programma del corso avanzato nel corso introduttivo, ed arricchendo il corso avanzato con argomenti più impegnativi.

Il testo che propongo contiene buona parte del materiale utilizzato nei due corsi, ed è così organizzato:

- Un capitolo è dedicato alle implicazioni organizzative dell'esistenza di una rete globale – Si parte da considerazioni sulla opportunità di uno standard per arrivare alla struttura organizzativa che supporta la sua evoluzione (4 ore).

- Due capitoli sono dedicati allo strato *data link* – Distinguiamo la trattazione di reti locali (LAN) e geografiche (WAN) (18 ore).

- Due capitoli per illustrare il protocollo Internet – Vengono introdotti il protocollo di rete IP, ed i protocolli di trasporto UDP e TCP (16 ore).

- Un capitolo per illustrare l'uso di una interfaccia utente allo strato trasporto di Internet – Il capitolo si presta ad attività di laboratorio (4 ore).

- Due capitoli conclusivi, ciascuno dedicato ad una diversa applicazione (4+4 ore).

Ho riportato accanto a ciascuna parte il tempo che le ho dedicato nei corsi che ho tenuto: comprende anche la frazione dedicata alle esercitazioni. Come si vede, il materiale si presta a coprire circa 50 ore, ed è sovrabbondante per un singolo corso, che ne comprende circa 40. Esiste dunque margine per organizzare una parte del materiale per supportare una varietà di corsi:

- I capitoli 1 e 4 supportano un corso di 20 ore indirizzato specificamente ad Internet.
- I capitoli 2 e 3 supportano un corso di 20 ore indirizzato ai protocolli data link: da Ethernet ad ATM.
- I capitoli da 1 a 6 offrono supporto ad un corso completo di 40 ore, che si conclude in laboratorio.
- Il capitolo 1, insieme coi capitoli da 4 a 8 offrono supporto ad un corso orientato all'applicazione, che può concludersi con un progetto utilizzando i protocolli illustrati nell'ultima parte.
- L'intero contenuto del testo, affiancato da una trattazione completa degli argomenti solo introdotti nel capitolo 6, può supportare un corso di 40 ore cui sia collegato un modulo di laboratorio di circa 20 ore.

Ogni capitolo si conclude con alcuni suggerimenti relativi a esercizi o piccoli progetti; non si tratta di esercizi di "verifica di apprendimento", ma piuttosto di brevi spunti di riflessione su alcuni argomenti svolti nel capitolo.

I programmi presentati nel capitolo dedicato ai *socket* sono disponibili in Internet all'indirizzo `http://www.di.unipi.it/~augusto`.

Nella redazione del testo ho cercato di prestare molta attenzione alla terminologia, che ha la massima importanza nell'ambito delle reti di comunicazione: la comunità che sviluppa ed utilizza i protocolli di rete è vastissima, ed è necessario che la terminologia non produca gerghi compresi solo all'interno di una certa sottocomunità. La vera sfida si presenta al momento di decidere se un certo termine vada o meno tradotto dall'inglese. Non ho seguito nessuna regola in merito: talvolta ho tradotto, altra ho lasciato l'inglese nel testo, evitando però di introdurre mostri linguistici (bufferizzato?) anche se se ne incontrano spesso nel parlato. Ho inserito in appendice la tabella degli acronimi utilizzati nel testo, e nell'indice analitico riporto la traduzione italiana per le parole chiave introdotte nel testo, o l'originale inglese per quelle che ho deciso di tradurre.

Per tutta la parte relativa ad Internet ed ai protocolli di applicazione ho fatto fedele e puntuale riferimento ai documenti ufficiali di specifica dello standard, gli RFC. Per l'approfondimento delle nozioni introdotte nel testo è senz'altro indicato consultare questi documenti, che sono liberamente disponibili in Internet, ad esempio all'indirizzo `http://www.ietf.org`. Ho messo a disposizione quelli cui faccio riferimento nel testo all'indirizzo `http://www.di.unipi.it/~augusto`. Per quanto riguarda la parte dedicata alle reti LAN e WAN ho fatto spesso ricorso ai testi di A. Tanenbaum [26] e W. Stallings [24], [25].

Il testo è stato composto utilizzando esclusivamente strumenti software di pubblico dominio, supportati dal sistema operativo Linux: per la composizione ed impaginazione del testo è stato utilizzato LaTeX, per la realizzazione delle figure `Xfig`, per la correzione degli errori ortografici `ispell`, integrato nell'*editor* `Emacs`. Un ringraziamento particolare va a chi si impegna nella realizzazione di questi strumenti potenti ed affidabili.

Giugno 2002

Augusto Ciuffoletti
Affidatario del corso di Reti di Calcolatori II
presso il Dipartimento di Informatica
Università degli Studi Pisa

Capitolo 1

Panoramica sulla tecnologia delle reti di comunicazione

La tecnologia delle reti di comunicazione introduce concetti, e la relativa terminologia per esprimerli, che non si ritrovano in altri ambiti dell'informatica. La ragione va ricercata nel fatto che la tecnologia delle reti, e l'informatica distribuita in generale, si interessa della coordinazione di *molteplici* agenti, piuttosto che dell'attività di un *singolo* agente.

Prima di affrontare uno studio approfondito di alcune tecniche specifiche e molto diffuse, è quindi opportuno avere una visione d'insieme relativamente superficiale, che consenta, dopo, di collocare in un ambito allargato le tecniche specifiche che verremo studiando.

L'obiettivo di questo capitolo, dunque, è quello di dare una visione d'insieme degli argomenti attinenti le reti di calcolatori, e soprattutto di introdurre la necessaria terminologia, partendo dai concetti che dovrebbero già far parte del vostro bagaglio.

Un problema a parte, proprio dal punto di vista della terminologia, è la scelta, per i *termini chiave*, tra l'italianizzazione dei termini, la loro traduzione, o la conservazione dei termini in lingua inglese. Non seguirò una regola fissa, adattandomi piuttosto al mio gusto: tuttavia, nell'indice analitico trovate, tra parentesi accanto al termine italiano, quello in lingua inglese.

I termini chiave sono riportati in corsivo la prima volta che vengono utilizzati, ed in seguito il corsivo viene utilizzato per richiamare l'attenzione su concetti rilevanti. L'indice analitico riporta almeno la pagina dove un certo termine chiave compare la prima volta, e dove quindi viene definito: l'indice analitico può servire dunque a conoscere l'equivalente inglese.

1.1 Protocolli e dispositivi

Lo studio di una rete di comunicazione offre sempre due possibili punti di vista. Uno orientato ai *dispositivi*, che si concentra sui componenti fisici della rete (ad esempio server, modem, linee di comunicazione) ed uno orientata ai *protocolli* (ad esempio IP, TCP).

Le due rappresentazioni sono ortogonali, e descrivono aspetti diversi di una stessa realtà: la prima legata alla interconnessione di componenti fisici, e descritta nei termini della fisica del supporto di comunicazione scelto (elettronico o ottico), la seconda invece legata ai protocolli di comunicazione utilizzati, e descritta in termini di algoritmi e dati utilizzati nella comunicazione. I due aspetti vengono a contatto al momento della *implementazione* dei protocolli: in quel momento l'informazione dovrà essere convertita in un segnale in grado di essere oggetto di comunicazione.

Questo corso è orientato ai protocolli, e coprirà solo marginalmente gli aspetti legati ai dispositivi: infatti nella progettazione dei protocolli è necessario tenere conto delle caratteristiche dei dispositivi utilizzati, in modo che il progetto possa poi essere realisticamente implementato.

1.2 Funzionalità di un protocollo

Una definizione molto generica di protocollo potrebbe essere la seguente: un *protocollo* definisce la struttura dei dati scambiati, e le modalità con cui questi vengono scambiati.

La definizione tuttavia nasconde un fatto fondamentale: un protocollo interagisce con altri protocolli, offrendo a questi funzionalità complesse, ed utilizzando funzionalità offerte da altri protocolli.

La struttura che si viene a creare è abituale nella tecnologia informatica: una gerarchia di strati, a livelli di astrazione crescenti, in cui ciascuno *strato* utilizza le funzionalità offerte dallo strato sottostante, per realizzare nuove funzionalità che offre allo strato superiore.

La differenza sostanziale che distingue la tecnologia delle reti dall'informatica convenzionale sta nel fatto che l'oggetto di interesse è la *comunicazione*, piuttosto che la *elaborazione* dei dati. Quindi, in ciascuno strato il compito del protocollo sarà quello di controllare la comunicazione entro una molteplicità di *agenti* attivi. In analogia, nell'informatica convenzionale un

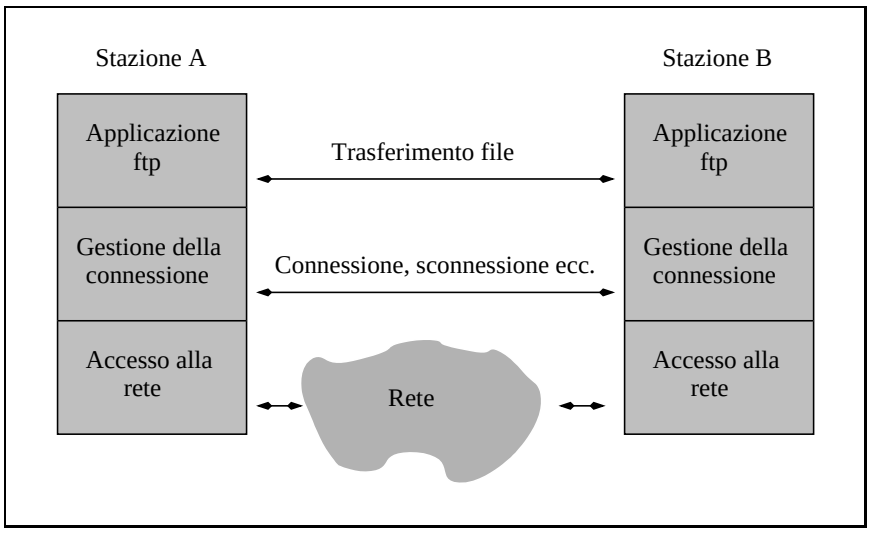


Figura 1.1 Rappresentazione a strati per un protocollo di trasferimento di file

programma coordina azioni semplici (assegnamenti, istruzioni di controllo ecc.) per ottenere funzionalità complesse.

Ciò che uno strato offre allo strato superiore è una *astrazione* della comunicazione che avviene tra i diversi agenti di quello strato: sono quindi le *caratteristiche* della comunicazione che si arricchiscono, di strato in strato. L'oggetto dell'astrazione finirà per essere sempre lo stesso: comunicazione. Il complesso delle funzionalità offerte da uno strato viene anche detto *interfaccia di comunicazione*.

In figura 1.1 vediamo una descrizione a strati di un protocollo per il trasferimento di file.

Possiamo rappresentare una *interfaccia di comunicazione* concretizzandola in un insieme di intestazioni di funzioni, un modulo *header* in C. Tra le funzioni offerte troveremo quasi invariabilmente una `send` ed una `receive`, che ad ogni strato troveranno connotazioni particolari, anche se tutte svolgeranno fondamentalmente lo stesso compito: comunicare.

La comunicazione avverrà dunque sempre tra *agenti* appartenenti allo stesso *strato*, utilizzando un *protocollo* comune: si parla in questo caso di comunicazione *peer-to-peer*.

Il protocollo quindi svolge la funzione di un *linguaggio comune* a tutti gli agenti che realizzano una certa *interfaccia di comunicazione*. Questo linguaggio comprende, come si diceva all'inizio, tanto la *struttura* dei dati

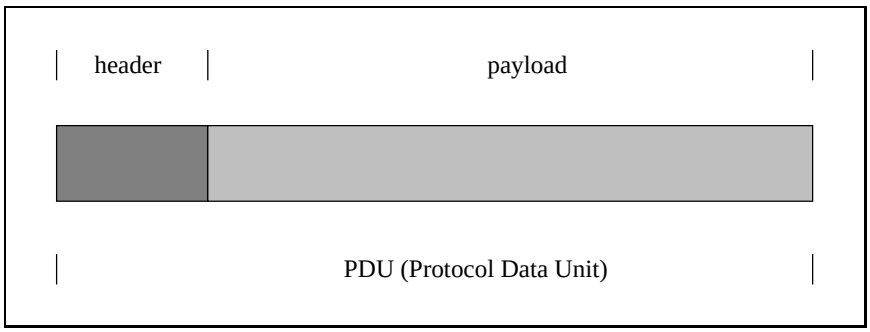


Figura 1.2 Header e payload di una PDU

scambiati, quanto le *modalità* dello scambio.

Il dato scambiato viene genericamente denominato *Protocol Data Unit* (PDU). Con poche eccezioni, una PDU è composta da (v. figura 1.2):

- uno *header*, che contiene informazioni che vengono controllate dal protocollo, e
- un *payload*, che invece è l'oggetto passivo della comunicazione.

Il contenuto dello *header* viene utilizzato dal protocollo per portare a termine la comunicazione: ad esempio può riportare l'indirizzo del mittente e del destinatario, può mettere in relazione la PDU con altre PDU per riordinarle, può contenere codici per la verifica del contenuto della PDU.

Al contrario, il contenuto del *payload* non viene utilizzato dal protocollo. Pur essendo manipolato in vario modo (ad esempio può essere tradotto in un'altra codifica, o frammentato), viene tuttavia restituito inalterato al suo arrivo a destinazione. Quindi è l'oggetto della comunicazione, e compare come parametro nelle funzioni che costituiscono l'interfaccia di comunicazione offerta dallo strato: tanto la `send` quanto la `receive` avranno un parametro che indicherà la collocazione del *payload* nella memoria locale. Nel primo caso il protocollo costruirà e spedirà una PDU prelevando il *payload* dalla memoria, mentre nel secondo il *payload* verrà estratto dalla PDU ricevuta e memorizzato nell'area indicata.

In genere si dice che il *payload* viene *incapsulato* nella PDU, come rappresentato in figura 1.3.

Ogni strato ha proprie regole per la costruzione dello *header* e per la manipolazione del *payload*: queste operazioni entrano a far parte del protocollo proprio di quello strato.

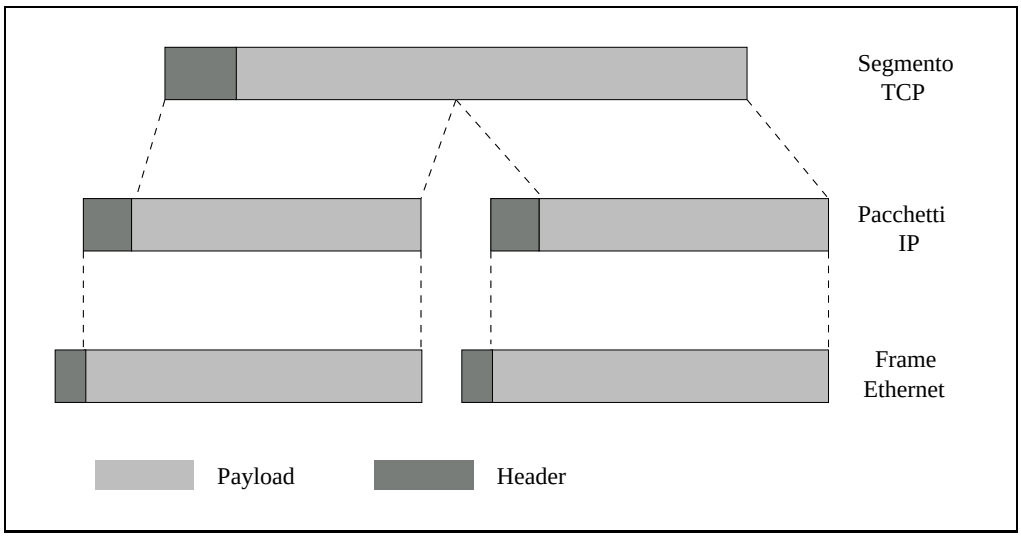


Figura 1.3 La PDU di uno strato è il payload per lo strato sottostante

Possiamo identificare altre funzioni che devono essere svolte da uno strato (v. figura 1.4), la cui realizzazione caratterizza il protocollo ([26] p.13).

Connessione e sconnessione Il modo in cui un *agente* inizia e termina la comunicazione.

Modalità di trasferimento dei dati L'orientamento della comunicazione, la sua temporizzazione ed altre proprietà.

Gestione degli errori Gli errori possono interessare il contenuto della comunicazione, o l'ordinamento dei messaggi, o la temporizzazione. La loro gestione dipende dal tipo di interfaccia offerta.

Controllo del flusso Può rendersi necessario, ad esempio, rallentare un mittente troppo veloce rispetto al ricevente.

Frammentazione/ricostruzione Serve ad adeguare la dimensione del *payload* offerto dall'interfaccia a quello reso disponibile dall'interfaccia utilizzata dal protocollo. Ad esempio, i *payload* provenienti dagli agenti che utilizzano l'interfaccia offerta da uno strato possono essere da questo ridotti in frammenti di dimensioni adatte per essere *payload* per l'interfaccia offerta dallo strato sottostante; una volta a destinazione, i frammenti verranno riordinati

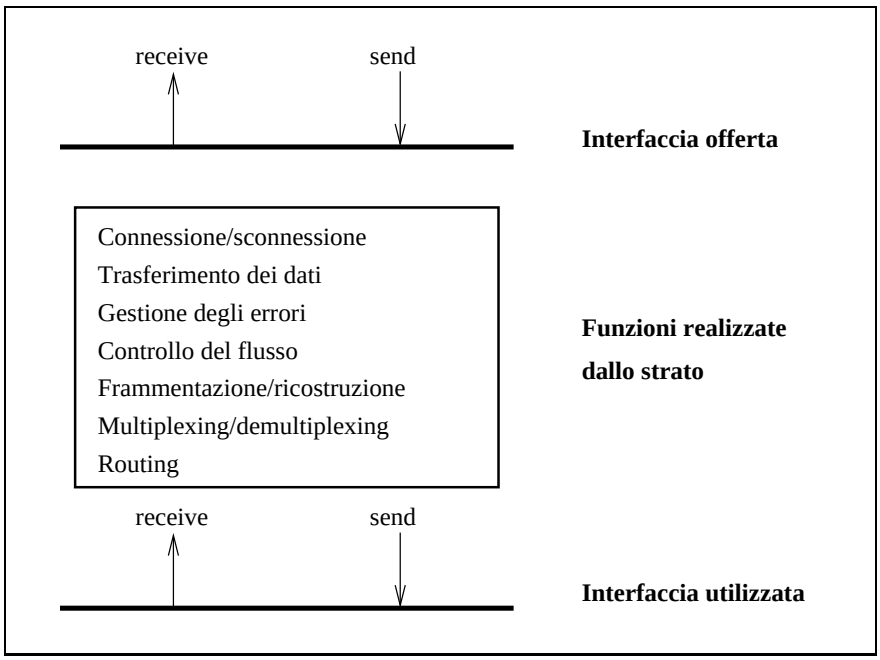


Figura 1.4 La struttura di uno strato e delle interfacce che lo interessano

e il *payload* originario ricostruito e consegnato agli agenti che utilizzano l'interfaccia. Può tuttavia avvenire il contrario, nel caso in cui la dimensione della PDU dello strato sottostante sia superiore a quella offerta.

Multiplexing/Demultiplexing Consente di realizzare una interfaccia condivisa da più agenti dello strato superiore; ad esempio, i *payload* provenienti dagli agenti che utilizzano l'interfaccia vengono opportunamente convogliati (*multiplexed*) verso altri agenti che li smisteranno (*demultiplex*) verso gli agenti che utilizzano l'interfaccia. Può tuttavia accadere il contrario quando un agente abbia a disposizione interfacce distinte verso la rete.

Routing Poiché ciascuno strato ha una propria topologia, si rende necessaria una funzionalità che trovi la strada (*route*) più adatta tra le due entità. Ad esempio, in un certo strato due *agenti* possono essere adiacenti, mentre nello strato sottostante la loro comunicazione può dover attraversare altri *agenti* intermedi.

Quindi ciascuno strato si trova a dover gestire una miriade di problemi impegnativi. L'impostazione di una soluzione non può essere frutto di de-

cisioni locali: è necessario che esistano solide convenzioni, che garantiscano che le soluzioni studiate per un agente riescano a coordinarsi con le altre.

1.3 La necessità della standardizzazione

È nella natura stessa della comunicazione la necessità di darsi delle regole universali, delle convenzioni a cui tutti si attengono per dare un significato a simboli altrimenti vuoti.

Gli standard non esistono solo in informatica. Ad esempio:

- Esiste uno standard per le dimensioni di dadi e bulloni. Altrimenti ogni ferramenta dovrebbe avere un numero indeterminato di dadi e bulloni, nonché di attrezzi per avvitarli o svitarli.
- Esiste uno standard per la sensibilità delle pellicole fotografiche. Altrimenti non esisterebbero le macchine fotografiche "automatiche" (e le modalità di impostazione della sensibilità sarebbe complicatissima).
- Esistono standard per i pesi e le misure: altrimenti non potrei ordinare per telefono 1 Kg di pane.

1.3.1 Gli standard: a chi servono

Gli standard servono a semplificare la comunicazione: corrispondono a concordare delle convenzioni.

- *Conviene aderire* ad uno standard per allargare il numero dei possibili utilizzatori (il *mercato*).
- *Conviene non aderire* ad uno standard per proteggere i propri prodotti dalla concorrenza.

Nel caso delle reti di calcolatori questa necessità si concretizza in un aspetto fondamentale: l'interoperabilità, cioè la capacità di agenti "diversi" di comunicare tra di loro. La diversità può riflettersi nell'uso di diversi linguaggi di programmazione (Java piuttosto che C), come nell'uso di diverso hardware (Intel piuttosto che Motorola).

Se lo standard viene seguito fedelmente dalla comunità degli utenti, l'interoperabilità è garantita. Ma appena questo "patto" si spezza, l'interoperabilità diventa critica, o naufraga, frammentando la comunità degli utenti in isole tra le quali la comunicazione diventa inaffidabile.

Al centro di questa complessa dinamica, da cui spesso dipende la fortuna di colossi finanziari come di piccole imprese, sta la gestione dello standard.

Uno standard non è una implementazione, e neppure una raccolta di implementazioni. È piuttosto una serie di regole che ogni implementazione deve rispettare. Quindi, almeno in linea di principio, uno standard deve essere indipendente dai dispositivi utilizzati per implementarlo. Quanto più lo standard è indipendente dal dispositivo, tanto maggiore sarà il suo successo, perché saprà adattarsi rapidamente a nuovi dispositivi. D'altra parte, quanto più uno standard è indipendente dal dispositivo, tanto cresce il rischio che la sua implementazione sia inefficiente: ad esempio, un certo standard, che introduce una forte ridondanza per garantire affidabilità nella comunicazione usa male un dispositivo che già offre una comunicazione estremamente affidabile.

1.3.2 Dove nasce uno standard

Quindi uno standard scaturisce da un gioco estremamente poco prevedibile che richiede continui compromessi. Alla base sta un lavoro di studio preliminare, che viene congelato nello standard. La coordinazione del lavoro di studio e sperimentazione è un dettaglio estremamente delicato, poiché è in questa sede che si incontrano o si scontrano gli interessi dei singoli produttori, o degli utenti.

Molto spesso accade che lo standard nasca per iniziativa di un gruppo di aziende od enti interessati ad uno sviluppo coordinato. Molto spesso si tratta di aziende produttrici, anche se tanto OSI (generato da ISO) quanto Internet (generato in ambito militare ed accademico) fanno eccezione. Un altro importante produttore di standard è l'*Institute of Electrical and Electronics Engineers, Inc.* (IEEE), non direttamente collegato ad interessi commerciali.

Analizziamo brevemente la storia di due standard, per cercare di capire come uno standard abbia successo mentre un altro fallisce.

1.4 Breve storia delle reti di comunicazione

L'obiettivo di una rete di comunicazione è la condivisione di risorse informatiche. Ma sul genere di risorsa condivisa si è giocata l'evoluzione delle reti di computer.

Anni '70 L'interesse era di pura condivisione di dati. Gli esempi tipici erano l'archivio dei *conti correnti in una banca*, le *prenotazioni di una compagnia aerea*, il *libro paghe di una ditta*, o *il magazzino di una industria*. Si parlava di database (basi di dati) accessibili da terminali distanti, e la topologia tipica era a stella, con un "cervellone" al centro ed i terminali alla periferia. La tecnologia della rete era generalmente telefonica (modem).

1975-1985 L'interesse si rivolge soprattutto alla condivisione di risorse di calcolo, oltre che di dati. Ad esempio una rete universitaria poteva disporre di un computer particolarmente potente per eseguire calcoli complessi, ed altri periferici per la memorizzazione o la visualizzazione, oppure un centro di calcolo di una grande azienda poteva disporre di un computer centrale con ampia disponibilità di memoria, ed altri periferici per elaborazioni complesse sui dati o per la loro acquisizione.

1985-1995 Si torna alla condivisione di dati, ma su una base sempre più estesa, sino a divenire mondiale con lo web, ma anche con le grandi basi di dati delle amministrazioni statali.

1995-... Si ritorna alla condivisione di servizi, tramite estensioni delle capacità dei server web, ormai alla portata di qualunque singolo o impresa, e dei corrispondenti clienti (i *browser*).

Accanto all'obiettivo primario della condivisione di risorse, un ruolo importante lo giocano alcuni obiettivi collaterali, estremamente popolari per un certo periodo, per poi diventare un requisito marginale quando la tecnologia o le applicazioni cambiavano. Anche in questo caso, possiamo collocarli nell'ultimo quarto di secolo.

Alta affidabilità Se più sistemi sono interconnessi, ed i loro guasti sono indipendenti, è possibile sfruttarne la ridondanza. Nel caso di un grande database degli anni '70, più copie di backup potevano essere mantenute in località distanti, per riparare ad eventi naturali o bellici devastanti.

Economicità Piuttosto che avere un unico "cervellone" con tanti terminali, meglio avere molte stazioni di lavoro, meno potenti. Il rapporto costo/prestazioni, per certe applicazioni, risulta migliore (anni '75-'85).

Comunicazione È possibile la cooperazione tra più utenti per la produzione di uno stesso prodotto. In particolare nella produzione scientifica, ragione che ha mosso lo sviluppo dello web, nato nella comunità dei ricercatori in Fisica (anni '85-'95).

Sicurezza La comunicazione bidirezionale consentita dalla tecnologia delle reti consente l'allargamento del mercato per qualsiasi azienda, con l'introduzione di demarcazione della proprietà e di transazioni finanziarie autenticate.

Dietro ciascun mutamento si nasconde una rivoluzione nella tecnologia informatica, spesso in sinergia.

1970-80 Il periodo è caratterizzato dalla transizione dai "grandi" computer degli anni '60, il cui costo li riservava alle grandi organizzazioni, ai mini-computer. Nello stesso periodo la tecnologia elettronica introduceva i microprocessori (8080 di Intel e Z80 di Zilog nel 1974).

1980-90 Il periodo è caratterizzato dalla transizione dai mini alle workstation, con costi accessibili ad un bilancio familiare per una unità di elaborazione completa; nello stesso periodo la densità di integrazione aumentava consentendo microprocessori più potenti e veloci (80386 di Intel nel 1985).

1990-... Superata una certa soglia di velocità di calcolo, diventano fattibili le interfacce grafiche (nel 1991, Intel 80486 con Windows 3.1, interfaccia grafica senza hardware dedicato). La facilità d'uso propria delle interfacce grafiche allarga il mercato del computer, quindi i prezzi cadono e gli investimenti crescono. Inoltre la tecnologia si auto-alimenta: migliori computer rendono possibile una progettazione estremamente raffinata, quindi la densità di componenti sui circuiti integrati cresce, e la velocità di elaborazione aumenta. Il computer diventa un elettrodomestico.

1.4.1 La tecnologia delle reti: il bambù e la quercia

Secondo un detto orientale, sotto il vento, il bambù si flette ma sopravvive e si raddrizza, mentre la quercia soccombe: la storia delle reti di comunicazione è

stata sottoposta a "venti" di cambiamento impetuosi, che hanno modificato completamente il modo in cui venivano intese. Almeno una quercia ne è rimasta vittima: lo standard OSI.

1.4.2 Lo standard OSI

Lo standard ISO-OSI è una espressione della ISO, fondata negli anni '40: da allora emana standard, dai bulloni alle pellicole fotografiche.

Lo standard OSI inizia la sua storia nel 1984. Si pone immediatamente come standard forte: è estremamente dettagliato, ma riflette una concezione della rete, quella della pura comunicazione (cfr. [26], p. 31-32), rapidamente sorpassata già negli anni '80.

Per descrivere quello che accadde, si usa spesso la metafora dell'*apocalisse dei due elefanti* (cfr. [26], p. 30-31). Le groppe dei due elefanti sono rappresentate da due picchi di attività su un certo argomento, che corrispondono alla fase di **ricerca**, che precede quella degli **investimenti**.

Tra i due picchi deve situarsi la formulazione dello *standard*, che consente lo sfruttamento ottimale degli investimenti, che si focalizzano su prodotti con un mercato molto ampio, garantito dalla presenza di uno standard.

L'evoluzione tecnologica sottopone lo standard a continui adattamenti, necessari ad incorporare nello standard le caratteristiche di nuovi dispositivi: lo standard deve sopportare queste sollecitazioni per garantire un ampio respiro alla fase degli investimenti. La conoscenza dell'argomento maturata durante la fase di ricerca dovrebbe garantire la flessibilità dello standard.

Se lo standard è **prematuro** il problema sarà compreso solo in parte, lo standard sarà di cattiva qualità e non riuscirà ad evolversi insieme alla tecnologia. Può accadere (talvolta) quando la ricerca insegue la tecnologia, senza riuscire a precederla, almeno in astratto.

Se lo standard è **tardivo** gli investimenti partiranno probabilmente prima che sia completo, eventualmente in direzioni diverse, ed il progresso tecnologico risulterà rallentato da una concorrenza commerciale disgregante.

OSI è un caso di standard prematuro: la poca (o cattiva) ricerca hanno prodotto uno standard che era già obsoleto prima di vedere la luce!

1.4.3 I difetti dell'OSI

Diversi autori hanno analizzato i problemi che hanno condotto ISO-OSI al fallimento. Vediamone alcuni, sempre ripresi da [26], p. 31-32.

Inefficienza Alcune funzioni (ad esempio la gestione degli errori) sono riprese in tutti gli strati, con grande inefficienza; il gran numero di strati (7) peggiora la situazione, visto che una comunicazione li deve attraversare tutti, prima di arrivare a destinazione, e ciascuno strato comporta un peso computazionale.

Collocazioni controverse Poca chiarezza sulla collocazione di determinate funzioni (e controversie nel comitato) in certi strati.

Superficialità Alcune scelte sono argomentate con superficialità.

Assenza di comunicazioni connectionless La popolarità di questo paradigma di comunicazione era invece crescente.

Assenza del concetto di elaborazione Invece lo standard si orientava alla pura comunicazione. Il modello tendeva al simmetrico sincrono, mentre l'architettura dei sistemi indicava come più adatto l'asimmetrico asincrono.

Tuttavia alcuni concetti basilari di ISO-OSI sono sopravvissuti, ed è in corso una rivalutazione di altri aspetti del modello.

Ad esempio, uno dei concetti fondamentali di ISO-OSI era la rappresentazione del software di comunicazione attraverso una gerarchia di *strati*, corrispondenti ad un livello di astrazione crescente, e vedremo come questa sia una forma di organizzazione del software ormai consolidata.

1.4.4 OSI è morto, viva Internet

Mentre OSI veniva progressivamente messo in secondo piano, la rete ARPANET conosceva crescente popolarità.

La rete ARPANET nasce nel 1969 da un progetto iniziato negli anni '60 dal dipartimento della difesa USA per avere una rete di computer in grado di sopravvivere ad eventi bellici. ARPANET acquisisce da UNIX (che nasce pure nel 1969) un approccio "composizionale". Ogni componente è progettato per:

- assolvere un compito specifico,
- avere una interfaccia chiara e ridotta.

ARPANET inizia dunque la sua storia nei primi anni '70, e formerà il nocciolo originario di Internet.

Nel 1983 la *Defense Communication Agency* consolida TCP/IP come protocollo standard nell'ambito di ARPANET, come esito della sperimentazione del protocollo *Network Control Protocol* (NCP).

Nello stesso anno il gruppo originario che conduceva la sperimentazione viene suddiviso in un numero di "task force", collettivamente note come *Internet Activities Board* (IAB).

Nel 1990 una nuova riorganizzazione allarga il numero di entità governative coinvolte nell'amministrazione delle risorse impegnate su Internet.

1.4.5 La gestione dello standard Internet

La crescita dello standard viene registrata in documenti che descrivono l'evoluzione dei diversi componenti dello standard, sino al loro "congelamento", quando diventano definitivi. Il principale strumento di documentazione dello IAB sono detti RFC.

Un RFC è un documento pubblico, e riporta risultati, problemi o suggerimenti di interesse per la comunità dei progettisti o degli utenti di Internet.

Gli RFC servono anche alla discussione ed alla divulgazione degli standard.

Attraverso gli RFC, Internet si arricchisce di nuovi protocolli, che rispettano i precedenti ed aggiungono nuove funzionalità: ad esempio, il protocollo di *File Transfer Protocol* (FTP) (il servizio che consente di "scaricare" file da rete) è specificato in un RFC [19].

Il programmatore che affronta la realizzazione o l'aggiornamento di un modulo che realizza un protocollo deve attenersi a due semplici *regole d'oro*:

> Essere restrittivi in quello che si fa,
> Essere permissivi in ciò che si accetta.

Quindi dovrebbe cercare di essere più possibile aderente alle prescrizioni più restrittive dello standard, ed accettare che i moduli con cui interagisce si comportino invece secondo l'interpretazione meno restrittiva del protocollo.

La terminologia adottata negli RFC consente infatti di specificare i protocolli con una certa elasticità, indicando in modo esplicito quali caratteristiche vanno interpretate in modo restrittivo, e quali in modo più rilassato: la specifica di questa terminologia si trova nell'RFC2119 [6].

`MUST` Indica un requisito primario.

`MUST NOT` Indica una proibizione primaria.

`SHOULD` Indica una raccomandazione: il programmatore dovrebbe seguire l'indicazione e realizzare un certo dettaglio, a meno che non esistano ottime ragioni per scegliere diversamente.

`SHOULD NOT` Indica una raccomandazione: il programmatore dovrebbe seguire l'indicazione, e non realizzare un certo dettaglio, a meno che questo non possa risultare opportuno o addirittura utile.

`MAY` Indica una caratteristica realmente opzionale. Può essere incluso od escluso anche su base semplicemente commerciale. Esistono alcuni `MUST` connessi ad una opzione specificata da un `MAY`: il modulo che non realizza una certa funzionalità deve (`MUST`) tuttavia essere in grado di operare con un modulo che le realizza, e viceversa, il modulo che realizza una certa funzione deve (`MUST`) essere in grado di operare con un modulo che non la realizza.

In conclusione, la combinazione delle *regole d'oro* e delle *parole chiave* per indicare la criticità dei requisiti garantisce un margine di elasticità allo standard, pur garantendo l'interoperabilità tra componenti realizzati da case diverse.

1.4.6 Gli RFC

Un protocollo Internet attraversa diversi stadi prima di diventare uno standard. In figura 1.5 sono rappresentati i diversi stadi evolutivi.

Un protocollo viene inizialmente proposto con un *draft* non incluso tra gli RFC. Può essere rifiutato, oppure accettato. Nel secondo caso attraverso varie fasi (tutte registrate come RFC) arriverà alla sua standardizzazione.

1.5 Confronto tra gli strati ISO-OSI e ARPANET

Anche se il concetto di *strato* è ben presente in ARPANET, questi non corrispondono a quelli di OSI.

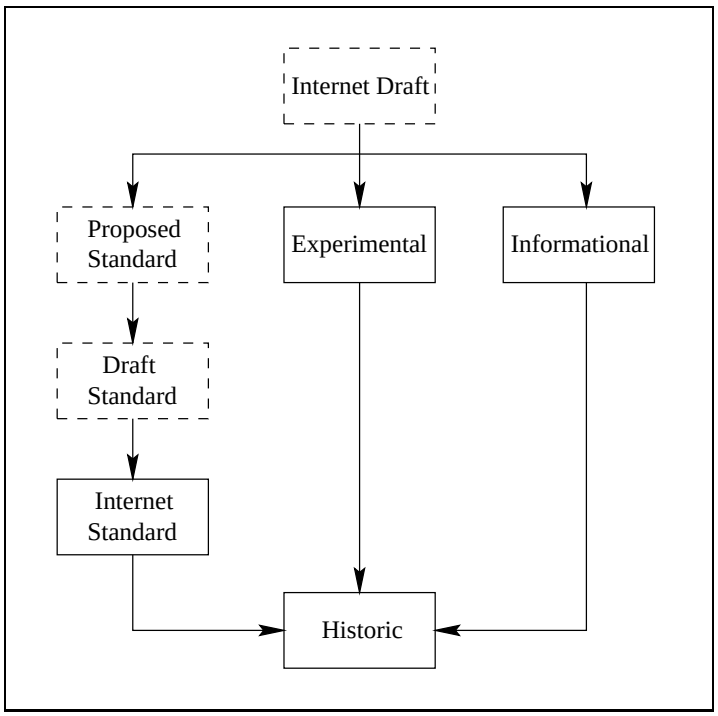


Figura 1.5 Evoluzione di un RFC

Un confronto diretto tra gli strati introdotti dai due modelli non è quindi possibile: i due modelli nascono da presupposti diversi, quindi seguono due strade diverse. Quello che faremo in questo capitolo sarà piuttosto ricollocare le funzionalità descritte nello standard ARPANET nella gerarchia OSI, ed eventualmente confrontare la collocazione di certe funzionalità nelle due gerarchie. Per questo analizzeremo brevemente il contenuto dei 3 strati ARPANET: rete, trasporto e applicazione.

Internet protocol

Il primo strato OSI (che specificava l'interfaccia fisica con la rete) non fa parte delle specifiche ARPANET. Quindi ARPANET nasce sganciato dal progresso nelle tecnologie di rete: un grande vantaggio.

Il primo strato ARPANET punta direttamente alla connessione tra reti: si chiama IP. Il concetto di connessione tra reti distinte era per sé poco adatto a OSI, che considerava piuttosto un'unica rete non strutturata. Il problema dell'interconnessione tra reti era trattato da OSI (con "riluttanza" usando il

termine usato in [26]) al terzo strato, che la terminologia OSI chiamava di *network*.

IP è dunque alla base della gerarchia ARPANET, ed è un protocollo *connectionless*: la comunicazione avviene tramite singoli messaggi (i *datagrammi*) che viaggiano dal mittente al destinatario. Protocolli *connection oriented*, dove invece la comunicazione avviene attraverso un flusso persistente di informazioni, vengono introdotti al livelli superiori.

Invece OSI introduceva già ai primi livelli protocolli *connection oriented*: infatti OSI, più legato a tecnologie telefoniche, si fondava su una astrazione di comunicazione vicina alla connessione telefonica, mentre ARPANET, anche per ragioni legate alle sua matrice militare, si affidava alla comunicazione attraverso pacchetti autocontenuti.

Protocolli di trasporto

Un servizio di trasporto, tanto in OSI quanto in ARPANET, è caratterizzato dal trasferire informazione dal mittente fino al destinatario finale (*end-to-end* nella terminologia inglese).

Il secondo strato ARPANET offre un servizio di trasporto *connection oriented*, ed uno *connectionless*: si chiamano rispettivamente TCP e UDP.

Lo standard OSI colloca i protocolli di trasporto al terzo livello: accanto ad un protocollo *connection oriented*, che può essere avvicinato a TCP, introduce un protocollo *connectionless* analogo a UDP.

TCP e UDP offrono una interfaccia adatta ad essere utilizzata nei linguaggi di programmazione ad alto livello.

Le applicazioni

Una *applicazione* utilizza la comunicazione offerta dallo strato precedente per offrire un *servizio* ad un processo utente. Quindi lo strato *applicazione* offre una interfaccia utilizzabile da un processo utente, mentre gli strati precedenti interfacciavano processi di sistema.

Rapportata allo standard OSI, una applicazione ARPANET copre tutti gli strati ARPANET superiori: sessione, presentazione e applicazione (nella terminologia OSI).

Infatti una applicazione gestisce la connessione realizzata al livello trasporto, ciò che OSI colloca al quinto livello, *sessione*. Ad esempio una appli-

cazione OSI può utilizzare più connessioni UDP o TCP, oppure utilizzarne una già stabilita da un'altra applicazione.

Inoltre l'applicazione ARPANET gestisce le diverse rappresentazioni dell'informazione sui diversi agenti: ad esempio, produttori diversi rappresentano diversamente il tipo di dato `int`, e l'applicazione ARPANET ne tiene conto. Questa funzionalità è associata, nello standard OSI, al sesto livello, *presentazione*.

Alcune operazioni attribuite allo strato di presentazione OSI vengono attualmente chiamate di *marshalling* e *unmarshalling*, per la codifica e decodifica di una sintassi comune.

Infine il livello applicazione di ARPANET copre gli aspetti specifici del servizio offerto tra i quali, quando si dia il caso, la realizzazione di azioni atomiche, la gestione della concorrenza e la *recovery* in caso di errore. Questi sono aspetti caratteristici pure del livello applicazione di OSI.

ARPANET, cioè Internet, offre una grande varietà di applicazioni di rete, ciascuna progettata con la solita ottica "composizionale", che segue quella di UNIX: ogni applicazione realizza una funzionalità semplice e circoscritta.

Ciascuna applicazione possiede inoltre il proprio standard, che ha una vita distinta da quella delle altre applicazioni.

1.5.1 Conclusione

In conclusione possiamo, con qualche cautela, affiancare le gerarchie OSI e ARPANET come in figura 1.6, sottolineando che, mentre ARPANET si fonda su comunicazioni *connectionless*, OSI parte da comunicazioni *connection oriented*.

Tanto per ARPANET quanto per OSI lo strato *trasporto* offre sia comunicazioni *connectionless* che *connection oriented* agli strati superiori.

1.6 Panoramica della tecnologia Internet

Si parla ormai di *Internet*, sostituendo l'originaria denominazione ARPANET, per intendere non solo il protocollo IP, Internet Protocol, ma il complesso di tutte le applicazioni, i protocolli ed i dispositivi che realizzano la rete mondiale: un termine più adatto forse sarebbe "tecnologia Internet".

Per avere un'idea di quale sia l'organizzazione interna di un nodo Internet, vediamo la gerarchia dei *moduli* che realizzano i vari strati Internet di un

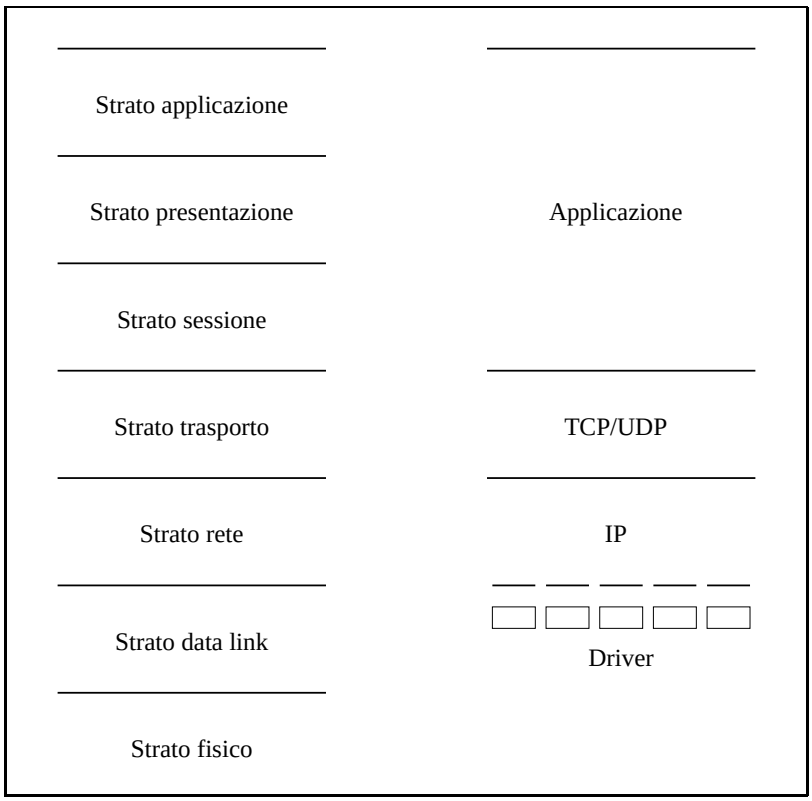


Figura 1.6 Confronto tra gerarchia OSI e ARPANET

nodo "tipico" che usi un singolo driver Ethernet per la comunicazione con gli altri nodi Internet. La gerarchia di strati che realizzano il supporto di comunicazione di un certo nodo vengono usualmente denominati lo *stack* del protocollo (v. figura 1.7) [23].

La linea orizzontale in basso rappresenta il cavo *Ethernet*. Il *transceiver* Ethernet, il circuito elettronico che trasferisce i dati dalla linea al computer, è rappresentato come un pallino nero sulla linea Ethernet.

Un *driver* è un programma che comunica direttamente con l'hardware. Nella figura è stato raffigurato il driver Ethernet. Il driver normalmente è implementato da uno o più processi che utilizzano tanto il supporto hardware fornito dalla scheda madre del *Personal Computer* (PC), che dunque è condiviso con tutti gli altri processi del nodo, quanto hardware specializzato.

Gli altri *moduli* sono programmi che supportano una funzionalità dello strato: ciascuno viene implementato da uno o più processi, generalmente

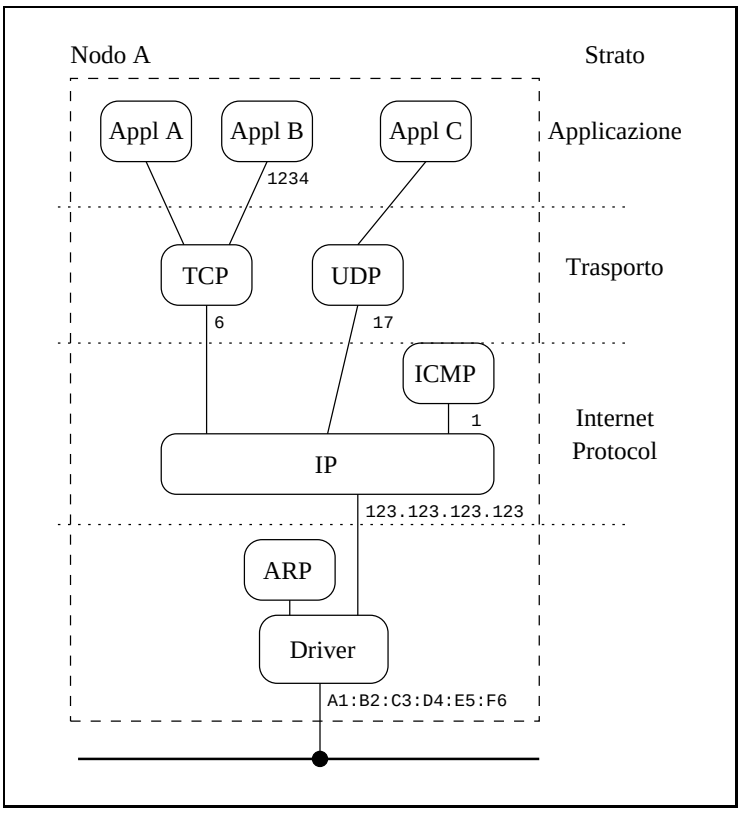


Figura 1.7 Lo stack Internet generico

supportati dalla stessa scheda madre del PC.

Il tipo di dato fondamentale di Internet è l'*ottetto*: si tratta di una sequenza di otto bit, in grado quindi di rappresentare 256 diversi valori. Nella definizione degli standard viene preferibilmente utilizzato il termine *ottetto* in sostituzione di *byte*.

Ogni strato possiede la propria forma di indirizzamento.

L'*indirizzo Ethernet* della scheda, in figura 1.7 accanto al collegamento tra il transceiver ed il modulo del driver, è composto di 6 ottetti che identificano la specifica scheda all'interno della rete Ethernet. L'indirizzo Ethernet è normalmente rappresentato come 6 numeri di due cifre esadecimali, separati da ":".

L'*indirizzo IP* del computer, in figura 1.7 accanto al collegamento tra il driver ed il modulo IP, è composto di 4 ottetti che identificano in modo univoco un computer di Internet. L'indirizzo IP è normalmente rappresentato

come 4 numeri in base 10, ciascuno compreso tra 0 e 255, separati da un ".".

Il *numero di porta*, in figura 1.7 accanto ad uno dei collegamenti tra un modulo della strato di trasporto e una applicazione, è composto da due ottetti, che identificano una specifica applicazione su quel nodo. La porta è normalmente rappresentata come un numero decimale tra 0 e 65535.

Una tripla costituita da un indirizzo di IP, da un protocollo di trasporto (in figura, TCP o UDP) e da una porta viene anche chiamato *indirizzo di trasporto*.

Ogni strato ed ogni protocollo utilizzano un termine proprio per indicare il formato della comunicazione. A seconda del protocollo che la utilizza, la PDU prende un nome differente:

- *frame* in Ethernet,
- *pacchetto* in IP,
- *datagramma* in UDP,
- *segmento* in TCP.

Ogni strato tratta i messaggi del proprio strato come *messaggi entranti* se provengono dallo strato sottostante, e come *messaggi uscenti* se provengono dallo strato soprastante. Ogni strato realizza una *interfaccia* alla rete per lo strato soprastante, utilizzando l'*interfaccia* offerta dallo strato sottostante.

Una generica applicazione utilizza uno stack di moduli: ad esempio, `ftp` (il comando UNIX che serve a trasferire file in rete) è un'applicazione che utilizza (in ordine discendente) FTP, TCP, IP ed infine un driver per l'accesso alla rete. Altre applicazioni utilizzano UDP invece di TCP: `procmail` (una applicazione per la gestione della posta elettronica) utilizza SMTP, UDP, IP ed il driver. Va ricordato che lo standard di Internet non descrive i driver, ma solo gli strati superiori.

1.6.1 Multiplexing e demultiplexing in Internet

I moduli UDP, TCP ed il driver Ethernet si comportano come *multiplexer* verso il basso, *demultiplexer* verso l'alto.

- I messaggi provenienti da più applicazioni vengono trasformati da TCP in pacchetti passati tutti allo stesso modulo IP, mentre TCP smista quelli generati dal modulo IP a diverse applicazioni.

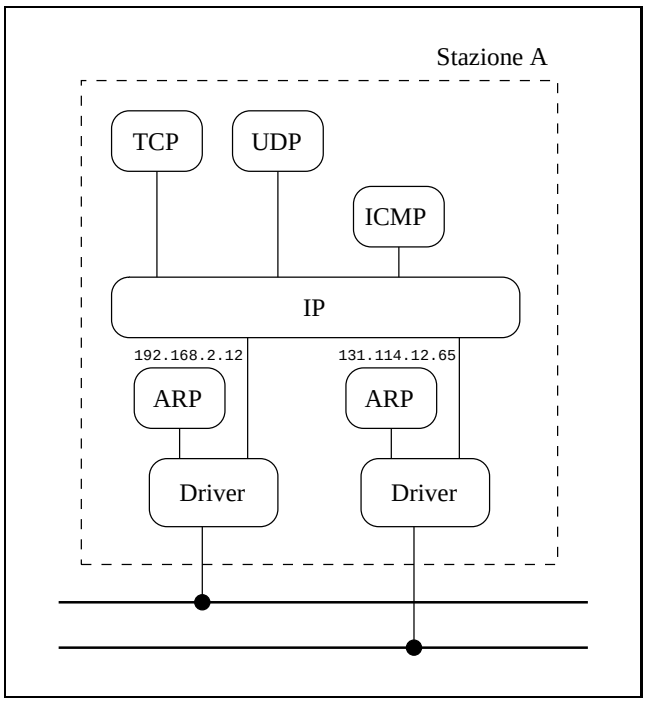


Figura 1.8 Lo stack Internet di un nodo collegato con due Ethernet

- I frame provenienti da Ethernet vengono smistati da/per il modulo IP o ARP (anche se questo è vero solo in astratto).

Invece il modulo IP può comportarsi come multiplexer e demultiplexer in ambedue le direzioni. La figura 1.8 illustra un caso di questo genere, che si presenta quando un nodo sia presente su due reti Ethernet diverse. In questo caso, il nodo ha due indirizzi IP e due indirizzi Ethernet.

Il modulo IP, in questo caso, ha funzione combinata multiplexer-demultiplexer sia verso l'alto che verso il basso: verso l'alto, il frame può provenire da uno dei due driver, e può essere smistato verso UDP o verso TCP. Dall'alto, il messaggio od il datagram può essere smistato sull'uno o sull'altro driver, a seconda della destinazione.

Un modulo IP che controlla più driver può avere anche una funzione complessa che non interessa gli strati superiori: può infatti servire da tramite per la interconnessione di due reti. In questo caso i frame non "emergono" allo strato superiore, ma vengono inoltrati sull'altra rete: questo genere di operazione viene denominata *forwarding*.

Un nodo che viene dedicato ad operazioni di *forwarding* (una funzione abbastanza gravosa) viene chiamato *router IP*.

Esercizi

- Identificare alcune attività umana che non usano standard.
- L'introduzione in un prodotto di una funzionalità non prevista dallo standard indebolisce lo standard: perché?
- Offrite un servizio di lettura telefonica di pagine televideo. Definite informalmente l'interfaccia di comunicazione offerta, ed il protocollo di comunicazione realizzato.
- Lo header IP conta almeno 20 ottetti, quello UDP ne conta 8. Un applicativo incapsula il proprio payload di 1000 ottetti in un pacchetto RTP, con uno header di 16 ottetti. RTP usa UDP come protocollo di trasporto. Quanto è lungo il pacchetto IP consegnato al driver? Qual è la percentuale di tale lunghezza dedicata agli header?

Capitolo 2

WAN: Wide Area Network

Le tecniche utilizzate per comunicare sono fortemente dipendenti dalla distanza che il segnale deve coprire: fenomeni dovuti alla velocità di propagazione, alle distorsioni o attenuazioni subite dal segnale durante il suo trasferimento, o semplicemente il costo del supporto di comunicazione condizionano pesantemente la scelta di una tecnica piuttosto di un'altra.

Quindi tradizionalmente si distinguono due problematiche: quella legata alla comunicazione su lunghe distanze, e quella legata a comunicazioni su brevi distanze.

Questo capitolo si interessa della prima, e quindi studieremo la realizzazione di reti destinate a coprire aree geografiche estese, comunemente dette *Wide Area Network* (WAN).

Si distinguono due approcci alternativi alla realizzazione di una rete su larga scala: a *commutazione di circuito*, oppure a *commutazione di pacchetto*. Questi corrispondono a due astrazioni molto diverse del concetto di comunicazione.

Nel primo caso i due agenti sono collegati tra di loro in modo persistente, attraverso l'allocazione di quello che si definisce *circuito*: per tutta la durata della *connessione* le risorse destinate alla realizzazione del *circuito* sono dunque impegnate, e non possono essere allocate ad altri *circuiti*. Il *circuito* servirà a trasferire un *flusso* di informazione, caratterizzato da una durata e da una continuità temporale.

La *commutazione di pacchetto* invece è basata sulla assenza di un collegamento persistente tra i due agenti: l'informazione da trasferire può essere racchiusa in un frammento atomico di comunicazione, il *pacchetto*. Le risorse

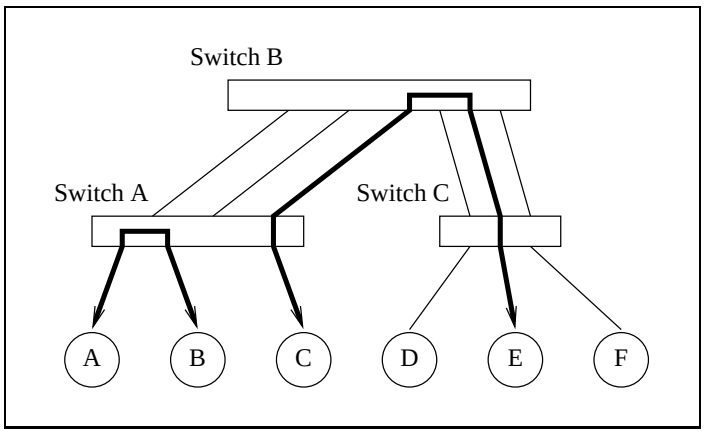


Figura 2.1 Un esempio di rete a commutazione di circuito organizzata gerarchicamente

sono dunque allocate al trasferimento del singolo *pacchetto*, solo per il tempo necessario al trasferimento del *pacchetto* stesso.

Si tratta di due paradigmi di comunicazione molto diversi, di cui vediamo le principali implementazioni.

2.1 Reti a commutazione di circuito

La tecnologia ha origini telefoniche: si tratta di una rete composta da nodi *commutatori* su cui confluiscono molte linee di comunicazione, e che hanno la funzione di interconnettere due delle linee connesse al nodo. Una volta operata la connessione, la comunicazione fluirà, in maniera mono- o bidirezionale, da una linea all'altra.

In telefonia il ruolo di *commutatore* viene svolto dalle *centraline*, che smistano le chiamate verso altre centraline che coprono aree via via maggiori (infatti esistono tariffe urbane, interurbane, internazionali ecc.) per poi raggiungere la centralina locale del destinatario. In figura 2.1 vediamo ad esempio sei nodi, indicati con le lettere maiuscole da A ad F, collegati tra di loro tramite due livelli di switch: il primo livello costituito dallo switch B, il secondo costituito dagli switch A e C. Le linee in grassetto con una freccia sui nodi indicano il percorso delle connessioni: la comunicazione tra il nodo A ed il nodo B attraversa un solo livello di switch, mentre quella tra C ed E ne attraversa due. In una rete telefonica, la prima potrebbe essere una comunicazione urbana, la seconda una interurbana. Gli agenti D ed F non

sono collegati (non è indicata la freccia), ed alcune linee disponibili restano inutilizzate.

Sappiamo tutti che la rete telefonica *circuit switching*, una volta riservata al traffico "in voce", ora viene largamente utilizzata anche per il traffico "dati": attraverso il modem possiamo codificare i dati con un segnale audio, con cui possiamo comunicare con un altro utente telefonico, ed in particolare con il nostro punto di accesso ad Internet.

L'approccio *circuit switching* ha pro e contro. Può essere molto inefficiente, in quanto la capacità della linea messa a disposizione è decisa staticamente al momento in cui viene stabilita la connessione, e non può essere modificata in seguito: se il mio interlocutore mi chiede un numero di telefono che ho sull'agenda, la trasmissione di qualche ottetto di informazione (il numero di telefono) potrà costare diversi minuti di impegno delle risorse: mentre cerco l'agenda, ed il numero nell'agenda, la linea sarà probabilmente silenziosa. D'altro canto il fatto stesso che la capacità venga assegnata staticamente renderà ben prevedibili i tempi di comunicazione, e questo può essere un vantaggio fondamentale per tutte le applicazione che richiedono tempi di reazione prevedibili (*real-time*).

La tecnologia più immediata per la realizzazione di un *commutatore* è il cosiddetto *crossbar* (v. figura 2.2). In questo caso si parla di *space division multiplexing*, in quanto il percorso seguito da diverse comunicazioni attraversa dispositivi interni fisicamente distinti. Questa tecnologia, inizialmente comune in telefonia, è stata in quel campo sostituita dalla tecnologia *time division*. Viene invece ancora impiegata per l'interconnessione veloce in sistemi multiprocessore.

La tecnologia attualmente utilizzata in molte centraline telefoniche è il *time division multiplexing*. Si tratta di istradare su una singola linea di comunicazione di grande capacità un certo numero di connessioni che utilizzano solo una frazione della capacità della linea (v. paragrafo 2.1.1 per la definizione di *capacità* di una linea).

Il funzionamento di uno switch in *time division multiplexing* somiglia molto a quello di un sistema operativo in *time sharing*: l'informazione da o per ciascuno dei canali viene gestita periodicamente. Nel caso di una centralina telefonica, l'informazione proveniente dai canali a bassa capacità viene digitalizzata e registrata in un buffer. Quando arriva il turno di un certo canale, il contenuto del buffer viene prelevato e riversato nel canale gestito in *time division* (v. figura 2.3). Poiché la somma delle capacità dei canali dedicati non supera la capacità del canale in *time-division*, tutta l'informazione entran-

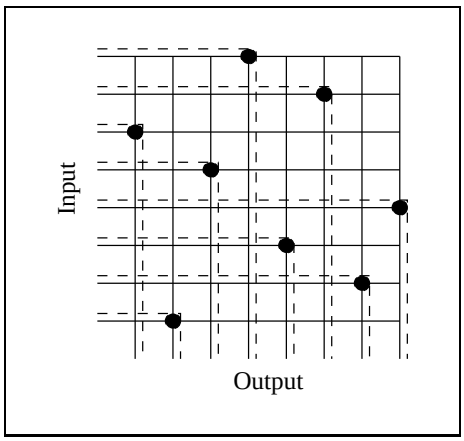


Figura 2.2 Il crossbar

te dai canali dedicati verrà convogliata nel canale uscente. Analogamente, i dati provenienti da un canale in *time-divisiono* vengono smistati, secondo uno scheduling fisso, verso i canali a bassa capacità in uscita: l'informazione digitalizzazione viene quindi memorizzata in un buffer e trasformata in forma analogica.

Ad esempio, consideriamo un canale con capacità di 500Kbps, utilizzato in *time-division* per realizzare 100 canali da 5Kbps per comunicazioni telefoniche. Assumiamo che un ciclo di polling di tutti e 100 i canali a bassa capacità avvenga ogni 0.1s: allora, durante questo intervallo, ciascun canale a bassa capacità produce 0.5Kbit di informazione nel buffer. Sul canale in uscita devono quindi trovare posto 100 frame da 0.5Kbit ogni 0.1s: quindi la capacità del canale deve essere almeno $\frac{100*0.5}{0.1} = 500$Kbps. I conti tornano.

Tuttavia l'esempio precedente mette in evidenza un punto debole degli switch in *Time Division Multiplexing* (TDM): dal momento in cui l'informazione entra nello switch, al momento in cui esce dall'altro switch all'altro capo della linea ad alta capacità, trascorre un tempo tra una e due volte il tempo di polling, nel nostro esempio tra 0.1 e 0.2 secondi. Quindi l'informazione subisce un ritardo che nel nostro esempio non è trascurabile: in una conversazione telefonica un ritardo di qualche decimo di secondo è già percettibile, e può causare errori di "sincronizzazione" tra gli interlocutori!

Anche lo standard di telefonia cellulare *Global System for Mobile communication* (GSM) utilizza una forma di *time-division* sui canali radio: in questo caso, la banda utilizzata è da 1850 a 1990 MHz (per una larghezza di banda di 140MHz) suddivisa in canali distanziati di 0.2MHz. La capacità di ciascun

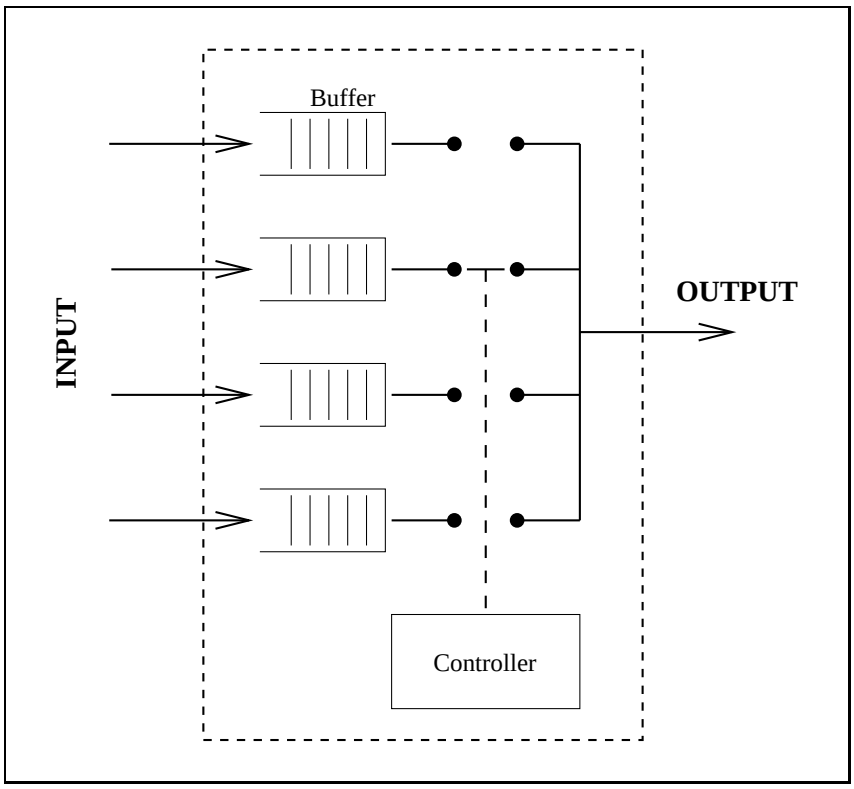


Figura 2.3 Un multiplexer in *time division*

canale è di 270Kbps, e la voce viene digitalizzata a 13Kbps. Ciascun canale è monodirezionale: per una conversazione vengono utilizzati due canali, distanziati di 80 MHz per evitare interferenze.

2.1.1 Digressione sulla capacità delle linee di comunicazione

Non voglio qui entrare nel dettaglio: l'argomento da solo necessiterebbe di un corso specifico per essere trattato esaurientemente. Ma alcuni semplici concetti sono necessari per giustificare certe soluzioni.

Una caratteristica fondamentale di una linea di comunicazione è, per noi, la sua *larghezza di banda*: indica l'ampiezza della *banda passante*, che corrisponde all'intervallo di frequenze in grado di attraversare la linea di comunicazione. Ad esempio, se una certa linea di comunicazione viene attraver-

sata "bene" da segnali sinusoidali di frequenza compresa tra 1MHz e 5MHz mentre frequenze di 0.5 MHz o di 6MHz vengono marcatamente indebolite (attenuate) diremo che quella linea ha una *larghezza di banda* di 4MHz (cioè $(5-1)$MHz), ed una *banda passante* tra 1 e 5MHz.

Il termine *larghezza di banda* si applica a segnali sinusoidali, ma sappiamo (il teorema di Fourier ce lo insegna) che ogni funzione può essere scomposta in un insieme di funzioni sinusoidali. Un segnale non è altro che una funzione che ha come dominio il tempo (si indica infatti con $s(t)$) e quindi può essere scomposto in tanti segnali sinusoidali: nella linea di comunicazione passeranno solo quelli entro la banda passante, ed alla sua uscita troveremo un segnale, tanto più distorto quanto più le frequenze che compongono il segnale cadono fuori dalla banda passante.

Se su un canale viaggia una forma d'onda perfettamente sinusoidale, il canale non porta informazione. Per ottenere che venga trasferita dell'informazione è necessario poter inviare segnali complessi, che quindi siano composti di più frequenze: maggiore sarà la larghezza di banda, maggiore la ricchezza del segnale che potrò trasmettere, quindi maggiore sarà il contenuto informativo che potrò inviare.

Il teorema di Nyquist fornisce in modo semplice la capacità del canale, data la larghezza di banda, quando vengono scambiati dati in codice binario, cioè con due soli valori di tensione, corrispondenti a 0 ed 1:

$$\text{Capacità} = 2 * (\text{Largh. banda})$$

Quindi un collegamento telefonico ordinario, con una banda passante da 0 a 3500 Hertz (la voce umana può essere effettivamente compressa entro quella larghezza di banda, restando identificabile ed intelligibile), necessita di una capacità di 7Kbps, cioè 7000 bit per secondo.

Il "doppino" comunemente usato nei collegamenti telefonici può arrivare a larghezze di banda dell'ordine del MHz, che corrispondono a qualche Mbps, e quindi potenzialmente potrebbe "contenere" qualche centinaio di comunicazioni telefoniche. Si ottengono risultati superiori con i cavi schermati, un conduttore interno circondato da una calza esterna come nei cavi di antenna della televisione. Si possono raggiungere così i 500MHz, corrispondenti ad 1 Gbps. Le fibre ottiche possono offrire ulteriori miglioramenti. In genere una fibra ottica è in grado di trasportare la luce entro un certo range di colori, cui corrisponde una larghezza di banda. Data l'altissima frequenza delle radiazioni luminose, avremo comunque larghezze di banda di molte centinaia di MHz, e corrispondentemente Gbps.

Per sfruttare la capacità del canale, si suddivide la banda disponibile in *canali*, a loro volta suddivisi in sottocanali. Abbiamo visto un esempio di canalizzazione parlando del *time-division multiplexing*. Si ottiene allora che da un singolo cavo coassiale da 100Mbps vengano ricavati 100 canali da 1Mbps. La convenienza economica è lampante: posando un singolo cavo si ottiene la capacità di un mazzo di 100 doppini da 1Mbps, che è ulteriormente canalizzabile, se utilizzato per comunicazioni telefoniche.

2.2 Reti a commutazione di pacchetto

La caratteristica principale di una rete *packet switching* sta nel fatto che le risorse per una comunicazione non vengono allocate staticamente a quella chiamata: invece, ogni elemento della comunicazione (il *pacchetto*) viaggia separatamente, e ciascun canale attraversato è assegnato a quella comunicazione solo per il tempo necessario a trasferire il pacchetto. In questo modo si ottengono un certo numero di vantaggi:

- Migliore sfruttamento delle risorse – Se non c'è comunicazione non vengono occupate risorse, diversamente dal caso *circuit switching*.
- Degradazione graduale delle prestazioni – In caso di congestione non si ha un crollo immediato delle prestazioni, ma queste degradano progressivamente, rendendo possibili diagnosi e regolazione.
- Possibilità di associare una priorità alle comunicazioni - Questo consente di offrire una migliore qualità del servizio alle comunicazioni prioritarie.

Nella realizzazione di una rete *packet switching* si distinguono due diversi approcci.

Datagram È rigorosamente aderente all'idea base del *packet switching*: ogni pacchetto viene trattato indipendentemente dagli altri. Poiché non esiste una correlazione tra i pacchetti, non è neppure possibile ordinarli, o rilevarne la perdita.

Circuito virtuale Realizza alcune caratteristiche del *circuit switching*: tra gli utenti viene stabilita una connessione persistente, controllando lo scambio dei pacchetti. Un *algoritmo di routing* potrà assegnare un percorso

stabilito a tutti i pacchetti della connessione. Diversamente dal *circuit switching*, non verrà riservata alla comunicazione parte della banda su ciascun ramo del percorso, ma i pacchetti saranno inoltrati al momento del loro arrivo. L'ordine di spedizione verrà ricostruito a destinazione, eventualmente rilevando la perdita di pacchetti.

I due approcci offrono caratteristiche complementari: il primo è più rapido, in quanto non è necessaria la fase di ricerca del percorso e di connessione, mentre il secondo offre sequenzializzazione e controllo degli errori. La flessibilità del datagram, che può dinamicamente aggirare punti di congestione o guasti, può renderlo maggiormente affidabile del circuito virtuale.

In figura 2.4 vediamo la temporizzazione di una comunicazione nei tre casi.

2.2.1 Il routing

Il problema del routing è più pressante nel caso del packet switching: in questo caso le decisioni devono essere prese rapidamente, e devono adattarsi dinamicamente al carico del sistema. Un algoritmo di routing deve possedere tutte le caratteristiche di un buon algoritmo distribuito.

Semplicità Incide sulla prevedibilità e sulla affidabilità.

Robustezza Garantisce la funzionalità anche in caso di guasto.

Stabilità Serve ad evitare che brusche modifiche nelle strategie locali possano portare a comportamenti oscillanti, che sotto-utilizzano le risorse. Vedremo un esempio tra poco.

Fairness È necessario garantire che le risorse siano distribuite equamente, anche in ragione della priorità.

Ottimalità Va valutata rispetto a quei parametri scelti per rappresentare le prestazioni della rete.

Efficienza Indica che l'*overhead* computazionale è ridotto, tanto sulle dimensioni dei messaggi quanto sul numero di nodi interessati.

Per arrivare ad un algoritmo che presenti queste caratteristiche dovranno essere valutate un gran numero di scelte realizzative.

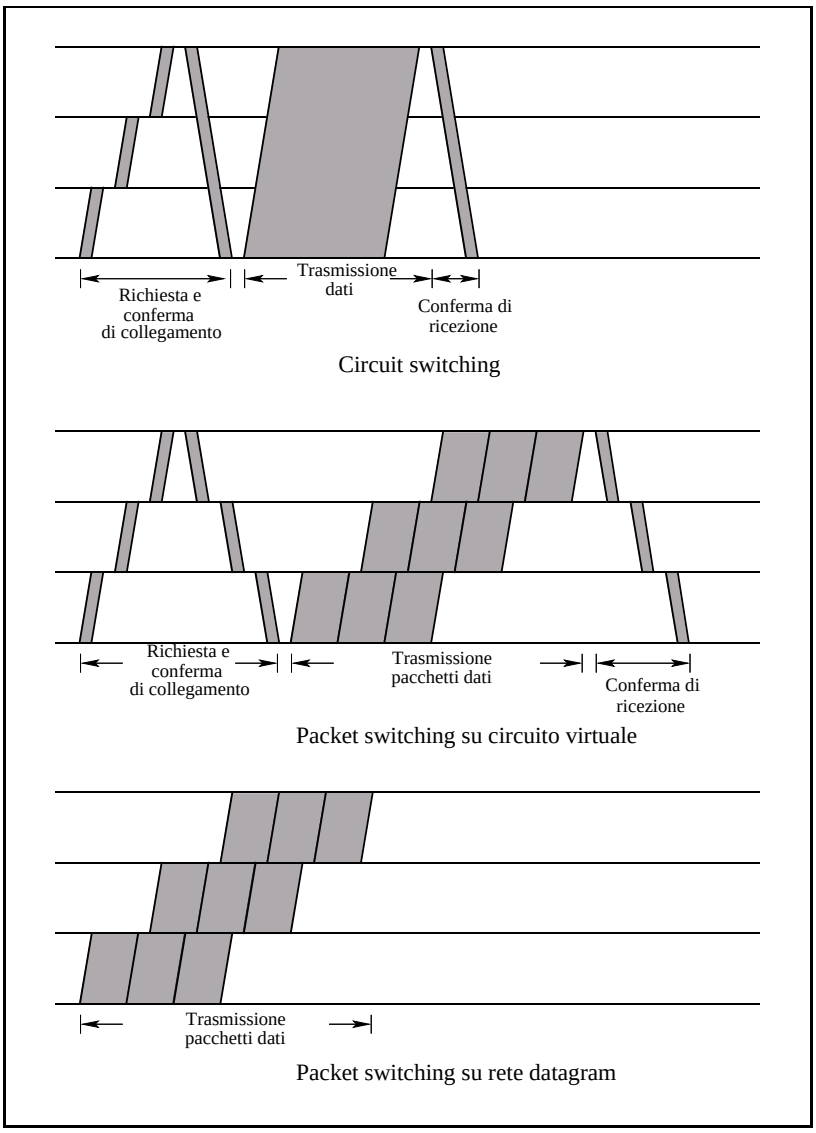


Figura 2.4 Confronto tra Circuit e Packet switching

Obiettivi di ottimizzazione In genere è necessario operare un compromesso tra i possibili parametri da ottimizzare. Sarà poi difficile modificare l'algoritmo se cambiano le priorità nelle prestazioni. Tra i parametri che possono essere oggetto di ottimizzazione ci sono il numero di canali interessati da una comunicazione, il costo di una connessione, quando l'utilizzazione dei canali abbia costo variabile da canale a canale, il ritardo, ottenuto cu-

mulando ritardi interni, di buffering e di rete, la quantità di informazione trasferita nell'unità di tempo (throughput).

Collocazione delle decisioni È necessario indicare chi controlla il routing, e quando è opportuno intraprendere questa operazione. Esistono almeno tre alternative, per quanto riguarda la localizzazione del controllo del routing:

Routing centralizzato Un solo nodo nella rete controlla il routing. È poco affidabile, visto che il guasto di quel nodo immobilizza la rete.

Routing distribuito Ogni nodo può controllare il proprio routing, coordinandosi con i vicini. Notevolmente più complesso, ma molto più robusto.

Source routing Il mittente determina, su base locale, il percorso che verrà seguito dalla comunicazione.

Il momento in cui viene determinato il routing dipende anche da altre decisioni: in una rete *virtual circuit* le decisioni verranno prese solo prima di stabilire la connessione, mentre in una rete *datagram* potranno essere modificate non appena cambino le condizioni di carico della rete.

Provenienza delle informazioni e temporizzazione delle decisioni In caso di routing distribuito, un nodo può ottenere informazioni da nodi adiacenti, o da nodi attraverso i quali transitano comunicazioni dirette a lui. In caso di routing centralizzato l'informazione viene raccolta da tutti i nodi. Se la raccolta delle informazioni richiede comunicazioni, è necessario determinare quando raccoglierle.

Una *strategia di routing* consiste in una serie di risposte alle scelte illustrate di sopra: vediamo alcune di queste strategie.

Il routing statico

È l'alternativa concettualmente più semplice. La rete è rappresentata come un grafo pesato (v. figura 2.5): per determinare il routing, per ogni coppia di nodi viene calcolato il cammino di costo minimo. Il risultato di questo calcolo risulta in una tabella simile alla 2.1. Questa tabella potrà risiedere su un nodo specializzato.

Quindi su ogni nodo viene registrata una tabella, che, per ciascun nodo della rete, indica verso quale dei vicini istradare un messaggio diretto a quel

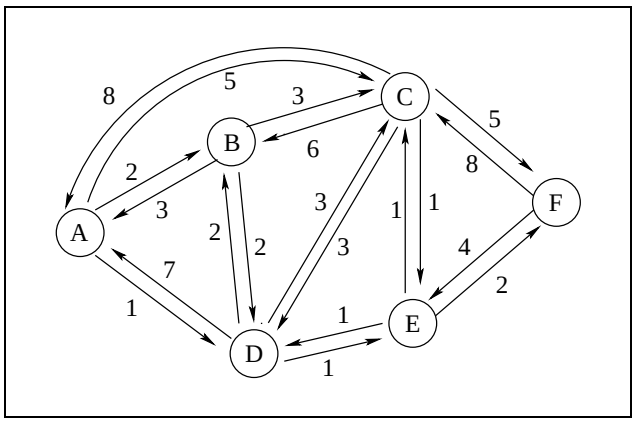


Figura 2.5 Una rete di comunicazione rappresentata con un grafo pesato

Tabella 2.1 Tabella di routing del sistema

origine	*destinazione*					
	A	B	C	D	E	F
A	-	B	D	D	D	D
B	A	-	C	D	D	D
C	E	E	-	E	E	E
D	B	B	E	-	E	E
E	D	D	C	D	-	E
F	E	E	E	E	E	-

nodo. In pratica si tratterà di una riga della matrice di partenza. La tabella 2.2 corrisponde alla *tabella di routing* per il nodo B.

Tabella 2.2 Tabella di routing per il nodo B

destinazione	*prossimo*
A	A
C	C
D	D
E	D
F	D

Quindi la tabella complessiva viene distribuita tra tutti i nodi del sistema.

Il routing statico ha il pregio di essere concettualmente semplice da realizzare. Tuttavia non è in grado di reagire alla congestione di un link. Una sem-

plice modifica consiste nel predisporre dei cammini alternativi, che possono essere utilizzati in caso di guasti o di congestione.

Flooding

Si tratta di una strategia che presenta vantaggi e svantaggi speculari rispetto al routing statico: consiste nel fatto che ogni nodo ritrasmette a tutti i propri vicini tutti i pacchetti che riceve. Quindi l'invio di un messaggio genera una valanga di messaggi che, prima o poi, raggiungono il destinatario finale: *flooding* sta appunto per "inondazione". Per arrestare o rallentare la moltiplicazione dei messaggi si possono utilizzare diverse tecniche:

- Non ritrasmettere il pacchetto verso il mittente.
- Identificare ogni messaggio e non ritrasmetterlo più di una volta.
- Inserire in ciascun messaggio un valore che indichi il massimo numero di ritrasmissioni (*hop-count*). Ad ogni ritrasmissione il campo deve essere decrementato di una unità, ed il messaggio non viene ritrasmesso quando l'*hop-count* raggiunge il valore 0.

La strategia di *flooding*, a fronte di un costo che può essere molto alto, offre diversi vantaggi:

- Se esiste almeno un cammino tra i due nodi, il pacchetto giunge a destinazione: quindi è eccezionalmente robusta.
- Il pacchetto giunge a destinazione nel più breve tempo possibile.
- Tutti i nodi connessi all'origine del messaggio (entro una certa distanza, se viene utilizzato l'*hop-count*) vengono raggiunti.

Questa ultima caratteristica rende estremamente attraente questa strategia nel caso in cui si debba realizzare un *broadcast*, cioè una comunicazione da un nodo verso tutti gli altri nodi.

Naturalmente il punto debole del *flooding* è il grande traffico generato, che è proporzionale al numero di link nella rete.

Routing casuale

Il nodo verso il quale ritrasmettere il pacchetto è scelto a caso. Eventualmente, può essere scelta una regola che privilegi alcuni nodi rispetto ad altri (ad esempio secondo la capacità del canale). Come nel caso precedente, la strategia può non avere necessità di informazioni sulla rete.

Routing adattivo

Consente alle regole di routing di adattarsi alle condizioni della rete, in particolare per quanto riguarda la *congestione* od il *guasto* di nodi o collegamenti. Tende anzi a prevenire le possibili cause di congestione, allontanandone quindi l'instaurazione prima di porvi rimedio.

Nelle intenzioni, dovrebbe quindi garantire prestazioni più equilibrate rispetto a quelle proprie delle altre strategie, ma gli algoritmi di adattamento possono essere notevolmente complessi e difficili da controllare.

Infatti la principale difficoltà di un algoritmo di routing adattivo sta proprio nel modo in cui viene ottenuta l'informazione "di controllo" riguardante la rete, e le modalità di reazione, cioè di adattamento locale.

Le informazioni possono ridursi a dati disponibili localmente (ad esempio lunghezza della coda di spedizione) o relative ai nodi vicini (ad esempio richiedendo periodicamente ai vicini una valutazione del proprio carico di lavoro). Ma le strategie adattive normalmente devono appoggiarsi a dati relativi all'intera rete, per ottenere le migliori prestazioni. Purtroppo non è semplice ottenere tali informazioni, che spesso non sono neppure attendibili.

Le maggiori reti di comunicazione hanno incontrato problemi con gli algoritmi di routing scelti: vediamo quale è stata l'evoluzione.

2.2.2 Evoluzione delle strategie di routing in ARPANET

L'algoritmo di routing in ARPANET (poi Internet) si è evoluto attraverso tre fasi, ciascuna delle quali ha goduto di un periodo di assestamento, durante il quale sembrava perfettamente adeguata. Successive evoluzioni nell'utilizzazione della rete rendevano tuttavia inadatta la soluzione, e forzavano una sua revisione che fosse compatibile con la precedente.

1a fase – Bellman-Ford con la lunghezza delle code

L'*algoritmo di Bellman-Ford* (1962) consente di calcolare presso ogni nodo della rete una tabella di routing locale, utilizzando informazioni su tutti i link della rete.

L'algoritmo è ricorsivo: sia $h_{origine}$ il nodo locale per cui vogliamo calcolare il miglior percorso lungo al più i link verso il nodo h_{dest}, avendo a disposizione i migliori percorsi lunghi al più $(i-1)$ da $h_{origine}$ verso tutti gli altri nodi, con i relativi costi. Se non esistono cammini lunghi al più $(i-1)$ per un certo nodo, al nodo viene associato un costo ∞. Questa informazione può essere rappresentata con una matrice, ottenuta dall'iterazione precedente (v. tabella 2.3 per la tabella del nodo B, cioè $h_{origine} = B$ al passo 2). Al primo passo (cioè per $i = 0$), la matrice contiene solo ∞, salvo per la riga del nodo $h_{origine}$ stesso, con associato un costo 0.

Tabella 2.3 Tabella di routing per il nodo B per cammini lunghi al più 2

h_{dest}	*cammino*	*costo*
A	B-A	3
B	B	0
C	B-C	3
D	B-D	2
E	B-D-E	3
F	B-C-F	8

Per ottenere la matrice per percorsi lunghi al più i, per ciascuno nodo h_{dest} della rete calcoleremo il miglior percorso $L_i[h_{dest}].cammino$ lungo al più i, ed il costo associato $L_i[h_{dest}].costo$. Otterremo questo calcolando, per ciascun nodo $h_{intermedio}$ della rete:

$$costo[h_{dest}] = L_{i-1}[h_{intermedio}].costo + c(h_{intermedio}, h_{dest})$$

dove la funzione $c()$ indica la funzione che associa ad un arco il suo costo, infinito se l'arco non esiste (ad es. $c(F,C) = 8$ e $c(B,E) = \infty$). Trovato il $costo[]$ minimo, questo corrisponderà ad un nodo intermedio ottimale $h^{ottimale}_{intermedio}$. Per concludere il minimo costo di un cammino lungo al più i da $h_{origine}$ a h_{dest} sarà:

$$L_i[h_{dest}].costo = L_{i-1}[h^{ottimale}_{intermedio}].costo + c(h^{ottimale}_{intermedio}, h_{dest})$$

ed il cammino corrispondente sarà dato dalla giustapposizione del cammino dall'origine sino al nodo intermedio ottimale, più l'arco dall'intermedio ottimale alla destinazione:

$$L_i[h_{dest}].cammino = L_{i-1}[h^{ottimale}_{intermedio}].cammino + (h^{ottimale}_{intermedio}, h_{dest})$$

L'iterazione prosegue sino a che la matrice resta inalterata. Allora siano certi di aver raggiunto un punto fisso, che corrisponde ad una soluzione del problema.

Il costo associato a ciascun link, nella prima versione del routing per ARPANET, è la lunghezza della coda di spedizione sul link. Tuttavia la rapida variabilità di questa informazione tende a rendere instabile il routing.

2a fase – L'algoritmo di Dijkstra

La versione successiva utilizza l'*algoritmo di Dijkstra*(1959), e si basa sui tempi di comunicazione con i vicini. Questi vengono stimati misurando il tempo trascorso dalla spedizione di un messaggio di controllo, trasmesso a ciascun vicino a scadenze regolari, alla ricezione del riscontro dal destinatario. Variazioni significative vengono comunicate in broadcast con un algoritmo di flooding, e portano alla modifica delle tabelle di routing.

Anche l'algoritmo di Dijkstra è ricorsivo: si assume di avere a disposizione gli $i-1$ nodi più vicini, in termini di costo, al nodo di partenza $h_{origine}$, e si calcola l'i-esimo nodo più vicino. Questa informazione può essere memorizzata in una lista, ciascun elemento della quale contiene tre informazioni: h_{dest}, il nodo destinatario, *cammino*, il cammino per raggiungere la destinazione, *costo*, il costo associato a quel cammino. La struttura ha la proprietà di contenere solo cammini a costi minimi, ed inizialmente ($i = 0$) contiene solo un elemento con il nodo originario, a cui si associa il cammino vuoto, ed un costo nullo.

Per calcolare l'i-esimo elemento della struttura, si esaminano tutti i nodi della rete non raggiunti dai precedenti elementi della struttura, ma adiacenti a uno o più di essi. Si ottengono così una serie di cammini del genere $cammino(h_{intermedio}) + (h_{intermedio}, h_{destinazione})$, dove $cammino(h_{intermedio})$ è già contenuto in uno degli elementi della lista, e quindi ne è noto il costo minimo, mentre $h_{destinazione}$ non è ancora raggiunto da nessun cammino nella lista. Di tutti i cammini ottenuti è calcolabile allora il costo, e tra tutti viene scelto quello di costo minimo, che costituirà l'i-esimo elemento della struttura.

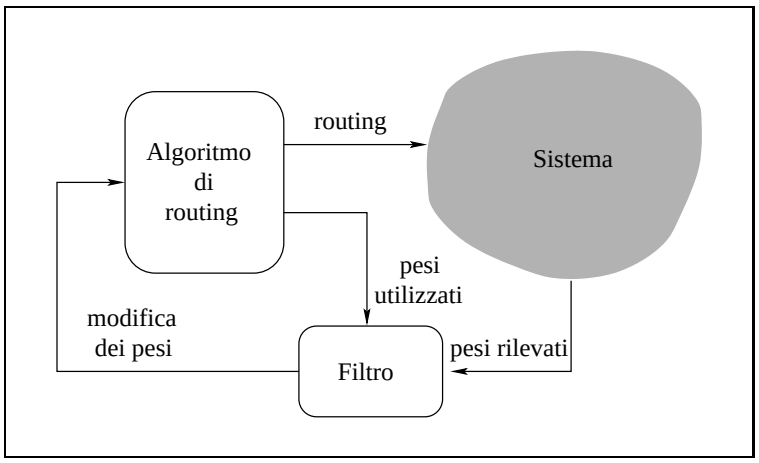


Figura 2.6 Un algoritmo di routing ed il *feedback* dalla rete

Si dimostra che il nodo raggiunto è il più vicino di quelli non ancora raggiunti (altrimenti i cammini nella lista non sarebbero i più brevi), e quindi che il cammino ottenuto è il minimo (altrimenti sarebbe già stato raggiunto). La prova dettagliata non rientra negli obiettivi del corso.

3a fase – Compensazione delle correzioni

L'aumento della velocità di comunicazione portò ad un ulteriore problema, che è tipico dei sistemi controllati da *feedback*.

Un sistema con feedback è rappresentato in figura 2.6: il funzionamento, oltre ai dati provenienti dal nodo stesso, è controllato anche da informazioni che arrivano dal sistema. Attraverso tali informazioni il nodo è in grado di adattare il suo funzionamento allo stato del sistema. Poiché lo stato del sistema a sua volta viene influenzato dal funzionamento del nodo, si instaura una interazione complessa tra i due. Se la reazione del nodo non è adeguata, il sistema tende ad entrare in oscillazione, con comportamenti estremamente lontani dall'ottimalità.

Ciò che accadde con la terza versione fu esattamente questo. Il fatto che l'intervallo tra le misurazioni dei ritardi fosse relativamente lungo (una decina di secondi) e che la risposta fosse invece istantanea portò alla conseguenza che le correzioni apportate dall'algoritmo di routing venissero ad essere eccessive rispetto al comportamento ottimale. In breve, una diminuzione di performance di un certo link portava a preferirne uno alternativo, scaricando

do quello meno efficiente. L'osservazione successiva riportava una situazione opposta, ma amplificata: quello meno efficiente ora mostrava ottime prestazione, mentre l'altro, sovraccaricato dalla modifica di routing, dava segni di congestione. Ciò portava ad una modifica dei pesi ancora più drastica, ed alla conseguente congestione dell'altro link. Una volta instaurata una alternanza di questo genere, il sistema si trovava in una situazione di persistente congestione.

Come in molti sistemi di controllo, la soluzione è stata l'introduzione di ritardi e la diluizione delle correzioni. In questo modo il nodo è in grado di sperimentare configurazioni intermedie, osservando la risposta del sistema.

In primo luogo, è stata modificata la stima del "costo" di un singolo link: la formula seguente incorpora sia la nozione di ritardo di comunicazione, sia quello di capacità del link. Infatti un link lento prossimo alla saturazione va distinto da un link ugualmente lento ma con la possibilità di gestire altro carico. Questa quantità è detta di *fattore di utilizzazione* :

$$\rho = \frac{2(T_s - T)}{T_s - 2T}$$

dove T_s è il tempo di servizio stimato, dipendente solo dalla dimensione del pacchetto e dalla capacità della linea, mentre T è il ritardo di comunicazione misurato, influenzato anche dai tempi di permanenza in coda.

Il *fattore di utilizzazione* impiegato per il calcolo del routing viene però mediato con quello utilizzato nell'intervallo precedente:

$$U(t) = \frac{\rho(t) + (k-1)U(t-1)}{k} \tag{2.1}$$

così che vengono evitate variazioni repentine. Le variazioni sono tanto più pigre quanto più è alto il valore della costante k, che viene determinata sperimentalmente.

2.3 I protocolli a commutazione di pacchetto su WAN

Con la crescente domanda di comunicazioni orientate al collegamento tra sistemi di calcolo, piuttosto che alle conversazioni telefoniche, le reti WAN si

stanno progressivamente orientando verso protocolli *packet switching*, piuttosto che *circuit switching*. Vediamo alcune delle tecnologie più impiegate nella realizzazione di questo genere di reti.

2.3.1 Il protocollo X.25

Il protocollo X.25 nasce negli anni '60, ed è stato uno dei protocolli di packet switching più diffusi. Entrava a far parte dello standard ISO-OSI, occupandone i primi tre livelli. Si articola dunque in tre strati: fisico, data link e network.

Si tratta di un protocollo orientato ai *circuiti virtuali*, quindi si distinguono tre fasi fondamentali: creazione del *circuito virtuale*, scambio di dati, rimozione del *circuito virtuale*.

La creazione del circuito consiste in un pacchetto dal nodo che inizia la connessione verso l'altro nodo (`Call Request`), seguita da una accettazione in senso opposto (`Call Accepted`). Lo scambio di dati viene poi controllato tramite un meccanismo che fa uso di *riscontri* per garantire che tutti i pacchetti spediti vengano ricevuti. Vedremo una tecnica simile in TCP. Data la tecnologia a circuito virtuale, i pacchetti vengono ricevuti nello stesso ordine in cui sono spediti. Infine i due nodi concludono la comunicazione: uno dei nodi spedisce una richiesta di conclusione (`Clear Request`), e l'altro conferma la rimozione del circuito (`Clear Confirmation`). Ciascun nodo può supportare un gran numero di circuiti, sino a 4095, e quindi implementa un opportuno *multiplexing* e *demultiplexing*.

Durante la comunicazione, il protocollo controlla tanto la regolarità del flusso di informazioni, quanto la presenza di errori: in questo riprende alcune caratteristiche di un altro protocollo, *High-level Data Link Control* (HDLC) (standard ISO 3009, ISO 4335). Al mittente viene spedito un riscontro per i pacchetti ricevuti: per questo si utilizza la numerazione progressiva dei pacchetti. Quindi il mittente ha la possibilità di sapere quando i pacchetti che ha spedito sono stati ricevuti. Insieme ai riscontri, il destinatario restituisce anche una informazione relativa a quanti altri pacchetti è preparato a ricevere: si parla di una *finestra scorrevole* (*sliding window*) che si sposta, man mano che i pacchetti vengono ricevuti. Tramite questo accorgimento il mittente può regolare la propria produzione di pacchetti, e si realizza così un efficace *controllo del flusso*. In questo modo X.25 risulta estremamente adatto a reti poco affidabili. Tuttavia, poiché questi controlli, relativamente complessi, vengono effettuati ad ogni stazione intermedia, ed ognuna di que-

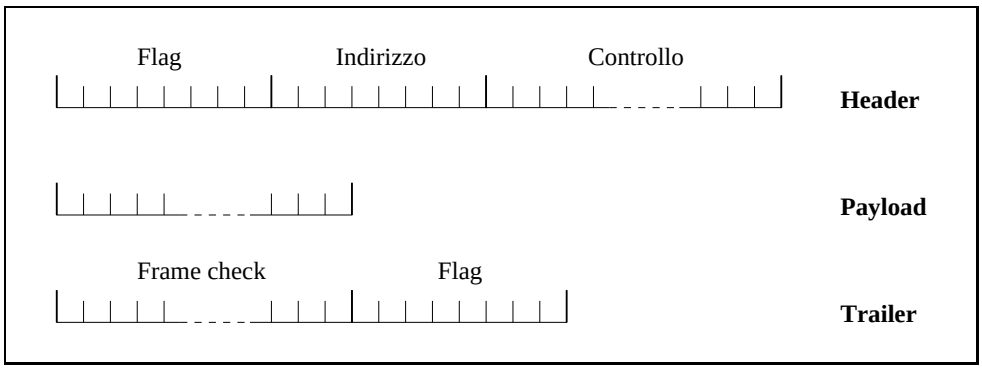


Figura 2.7 Formato del frame LAPB

ste deve essere predisposta per poter effettuare una ritrasmissione in caso di guasto, il costo dei dispositivi intermedi risulta elevato.

Il protocollo X.25 utilizza, per lo strato data link, un sottoinsieme del protocollo HDLC, chiamato *Link Access Procedure - Balanced* (LAPB).

Cenni sul protocollo LAPB

Il protocollo viene utilizzato su linee punto a punto: una caratteristica di questo protocollo è che ambedue le *stazioni* possono richiedere la creazione di un circuito virtuale.

La trasmissione avviene attraverso *frame*, che condividono un unico formato (v. figura 2.7).

Flag (8 bit) Il ricevente deve essere in grado di rilevare l'inizio di un nuovo frame: ricordiamo che LAPB si colloca ad uno strato molto basso dello stack, quindi le informazioni non sono in alcun modo strutturato. La sequenza di flag serve a questo scopo: si tratta di una sequenza di sei bit con valore uno compresa tra due bit di valore `0`: `01111110`. Questa sequenza identifica univocamente l'inizio di un frame. Bisogna tuttavia evitare che questa combinazione ricompaia altrove nel frame! Per fare questo si ricorre ad una tecnica detta *bit stuffing*: il resto del frame (in particolare, il *payload*) viene modificato in modo che ogni sequenza di sei o più bit a `1` venga interrotta da un bit a `0`. In pratica, il campo dati viene scandito ed al termine di ogni sequenza di cinque bit a `1` viene inserito uno `0`. In questo modo viene garantito che nel frame ci sarà un'unica sequenza di sei bit a `1`, il flag iniziale. Da parte del ricevente la rimozione dello `0` inserito dal mittente avverrà

con il seguente algoritmo: quando si incontra una sequenza di cinque `1`, si controllano i due bit successivi. Se il bit successivo è `0`, viene rimosso. Se i due bit successivi sono `10`, viene rilevata la presenza di un flag. Se sono due uno, viene segnalata la presenza di un errore.

Indirizzo (8 o più bit) Resta inutilizzato nel protocollo LAPB, ma è presente per compatibilità con gli altri protocollo HDLC.

Controllo (8 o 16 bit) Servono a controllare il protocollo. Vengono distinti tre tipi di frame di controllo: *I-frame* che trasportano un payload, ed eventuali informazioni di riscontro per frame già ricevuti, *S-frame* per le informazioni di riscontro non contenute nei precedenti, *U-frame* per altri frame di controllo e per segnalare errori. Il tipo del messaggio viene codificato nei primi due bit del campo controllo, mentre il resto del campo è utilizzato per informazioni dipendenti dal tipo (riscontri ecc.).

Payload (variabile) Presente negli *I-frame*.

Frame check (16 bit) Calcola una checksum per verificare, alla ricezione, la presenza di errori di trasmissione.

Flag (8 bit) Identica a quella introdotta in apertura, con la funzione di rilevare la fine del frame.

Il protocollo LAPB si articola in tre fasi:

Connessione Una delle due stazioni inizia il protocollo per stabilire una connessione, e l'altra segue. Vengono determinate le caratteristiche della connessione.

Trasferimento dei dati I dati vengono trasferiti con un meccanismo *sliding window*: il campo controllo di un *I-frame* o di un *S-frame* contiene il numero progressivo del frame atteso, oltre al numero progressivo del frame spedito (solo in caso di *I-frame*). In caso di rilevazione di errore in un pacchetto, il destinatario spedirà un *U-frame* con l'indicazione del numero d'ordine del pacchetto alterato. Il mittente ritrasmetterà il pacchetto ricevuto male e tutti gli altri successivi, compresi quelli che avesse già spedito (l'errore potrebbe aver alterato la rilevazione dell'inizio o della fine di un frame). I numeri d'ordine sono codificati con un numero limitato di bit: in fase di connessione si possono scegliere 3 o 7 bit per il numero d'ordine (8 o 128 numeri d'ordine disponibili).

Sconnessione Una delle due stazioni può decidere di interrompere la connessione, e l'altra segue. Alcuni frame possono andare perduti, visto che la stazione che inizia la sconnessione smette di ricevere all'atto della sconnessione.

Come si vede, dunque, il protocollo di *data link* utilizzato da X.25 è estremamente sofisticato e robusto. Ma, come si diceva, queste caratteristiche pesano su ciascun link di una connessione X.25, probabilmente composta dalla concatenazione di molti link.

Il protocollo X.25 è quindi particolarmente adatto a reti poco affidabili, su cui si voglia realizzare un servizio affidabile ad un costo non trascurabile. L'attuale tecnologia si sta tuttavia muovendo in direzione opposta, almeno per ciò che riguarda la rete fissa: collegamenti estremamente affidabili (come la fibra ottica), con componenti intermedi estremamente semplici. Di qui un progressivo abbandono di questa tecnologia a favore del *frame relay*, che opera a partire da ipotesi diametralmente opposte.

2.3.2 Frame relay

Dal punto di vista applicativo, l'approccio *frame relay* si distingue da X.25 in quanto parte dal presupposto di avere linee affidabili ad altissima capacità, e la necessità di avere poco *overhead* per ogni connessione sui nodi intermedi.

Dal punto di vista operativo, questi risultati vengono ottenuti con questi accorgimenti:

- Il controllo della connessione viene gestito tramite una connessione logica tra i due nodi interconnessi, piuttosto che su ciascuna delle connessioni fisiche. Questo libera le stazioni intermedie dal mantenere uno stato per ciascuna connessione, ed essere preparate alla ritrasmissione dei pacchetti perduti.

- Il *multiplexing* viene gestito a livello più basso rispetto a X.25, ed uno strato di software viene così eliminato.

Il compromesso tra le due tecnologie si gioca proprio sull'ipotesi di affidabilità o inaffidabilità della rete: nella tecnologia *frame relay*, il costo per la perdita di parte dei dati consiste nella ritrasmissione dei dati tra i due nodi interconnessi; quindi la ritrasmissione interesserà di fatto tutti i link fisici interessati dalla connessione. Invece, nella tecnologia X.25 la ritrasmissione

avviene solo sul link fisico interessato dalla perdita di informazione. In conclusione, X.25 è molto più efficiente nel trattamento della perdita di dati, a fronte tuttavia di un maggior costo dei dispositivi intermedi. Se la rete è inaffidabile, la tecnologia *frame relay* produce gravi perdite di prestazioni in caso di errore; ma se le perdite sono rare, l'investimento che X.25 fa sulla loro evenienza risulta ingiustificato, ed è preferibile l'utilizzo della tecnologia *frame relay*. Il miglioramento, secondo alcuni studi, è pari ad un ordine di grandezza o più.

La connessione viene dunque vista scomposta su due "piani": *control plane* e *user plane*. Sul *control plane* viene gestito un protocollo per lo scambio di informazioni di controllo riguardanti lo *user plane*, come la rilevazione degli errori ed il controllo del flusso. Le informazioni sullo *user plane* viaggiano invece senza overhead di controllo, visto che il loro controllo è già realizzato dal *control plane*.

2.3.3 ATM – Asynchronous Transfer Mode

Il protocollo ATM può essere visto come una evoluzione del protocollo *frame relay*, e sfrutta ulteriormente la disponibilità di una rete affidabile. Entra a far parte anche del protocollo Broadband-ISDN, una evoluzione del Narrowband-ISDN a 64 Kbps che conosciamo.

Nel protocollo ATM le PDU hanno dimensione fissa di 53 ottetti, dette *celle*: questo riduce l'*overhead* dovuto alla presenza di flag iniziali e terminali.

Inoltre viene introdotto un doppio livello di virtualizzazione: il circuito virtuale che il protocollo ATM offre all'utente – *Virtual Channel Connection* (VCC), viene infatti "ricavato" da un altro circuito virtuale il *Virtual Path Connection* (VPC).

I circuiti virtuali in ATM

Abbiamo detto che i circuiti virtuali vengono gestiti su due livelli: un nuovo VCC (canale) entra a far parte di un VPC (cammino) preesistente. Una delle ragioni che giustificano questa scelta è l'opportunità di ottimizzare le risorse impiegate nella realizzazione della connessione: in ATM viene risolto solo una volta il problema del routing e della costruzione del *cammino* tra due unità. Poi tutti i canali tra le due unità verranno creati utilizzando quel *cammino*.

Quindi di quel *cammino* sarà possibile mantenere molte informazioni, che vengono ottenute una volta sola e poi gestite per migliorare l'utilizzazione del *cammino*. Tra i risultati che si possono così ottenere c'è una chiara immagine della capacità disponibile, e di quella residua, su un certo *cammino*. In questo modo è possibile garantire la disponibilità di una certa capacità per soddisfare una certa richiesta, una volta che questa viene accettata.

Un *cammino* può connettere sia due nodi terminali, che nodi terminali con sottoreti: in quest'ultimo caso il cammino servirà a connettere il nodo utente con un nodo di controllo della sottorete, attraverso un unico cammino aggregato, e verranno scambiate anche informazioni per il controllo del flusso. Un cammino instaurato tra due sottoreti potrà servire invece per scambiare informazioni di routing.

Architettura del protocollo

Il protocollo è organizzato su due strati: sopra allo *strato fisico*, che non rientra nello standard, c'è un primo strato ATM destinato ad implementare sullo strato fisico la trasmissione delle celle, gestendo VPC e VCC. Sopra al primo strato ATM c'è uno strato adattatore *ATM Adaptation Layer* (AAL), che ha la funzione di realizzare su ATM il servizio richiesto dall'utente: ad esempio, realizzare packet switching per una rete IP o trasferire segnali audio. I requisiti delle due applicazioni sono contrastanti, ed ATM dovrà gestire le proprie risorse per soddisfare entrambe.

Il protocollo, come già il *frame relay*, è organizzato anche su più piani verticali (v. figura 2.9):

- Il *piano utente* che realizza il servizio.
- Il *piano controllo* che realizza il controllo delle connessioni.
- Il *piano amministrativo* che gestisce l'assegnazione delle risorse disponibili ai diversi piani ed ai diversi circuiti realizzati.

Il formato della cella ATM

La *cella* ATM è caratterizzata da una lunghezza fissa: 53 ottetti, di cui 5 formano lo header (in figura 2.8), mentre i rimanenti 48 ottetti sono il *payload*.

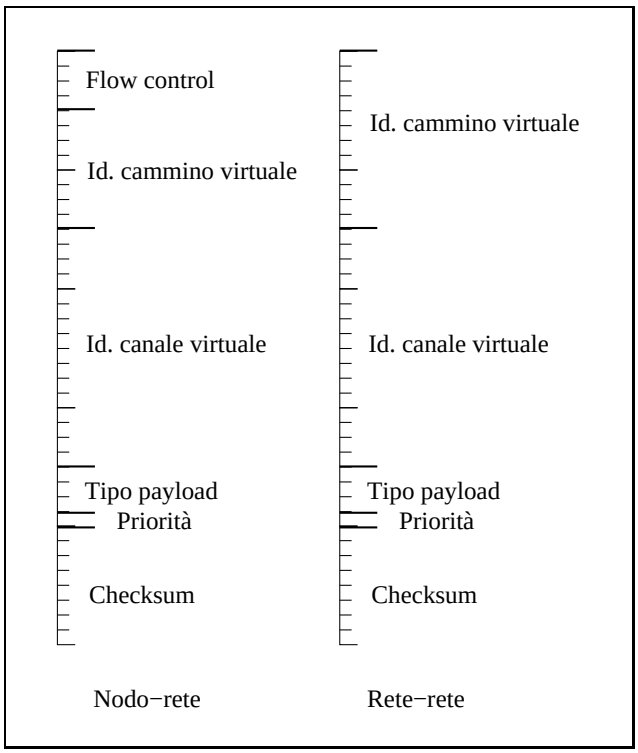


Figura 2.8 Formato dello header della cella ATM

La caratteristica di avere un formato a lunghezza fissa risulta conveniente da molti punti di vista:

- assenza di sequenze destinate al riconoscimento dell'inizio di una nuova PDU (sincronizzazione),
- semplicità degli algoritmi di buffering,
- riduzione dei tempi di attesa in coda per PDU ad alta priorità.

Il formato dipende dal tipo di VPC che viene utilizzata: se collega un utente ad una rete conterrà anche delle informazioni di controllo del flusso, che prenderanno posto nei primi 4 bit della cella:

Controllo del flusso e identificatore del VPC (12 bit) Come detto sopra, il campo sarà dedicato interamente all'identificatore del VPC se si tratta di un cammino da rete a rete, o interno ad una rete, mentre conterrà 4 bit

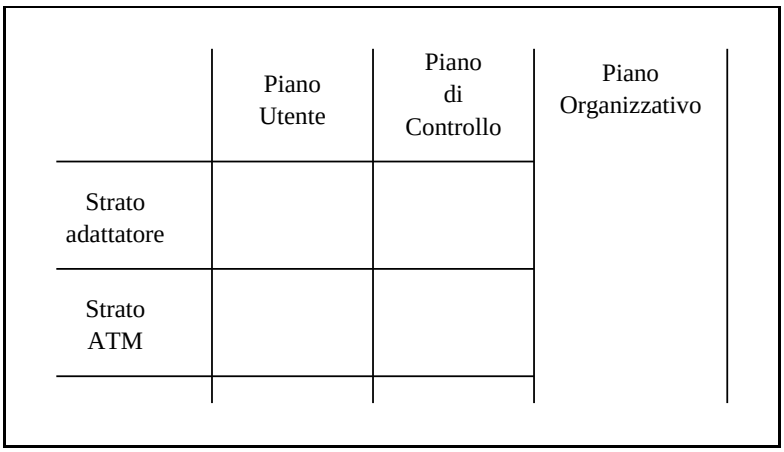


Figura 2.9 Architettura a "strati" e "piani" di ATM

per il controllo di flusso ed 8 di indirizzo se da nodo a rete. L'indirizzo consiste in un identificatore da utilizzare nel routing della cella.

Identificatore del VCC (12 bit) Serve all'identificazione del canale, all'interno del cammino.

Tipo del payload (3 bit) Il codice utilizzato è il seguente: il *primo bit* indica, se `0`, che si tratta di un payload del piano di utente; altrimenti fa parte del piano di controllo. Se si tratta di payload utente, il secondo bit indica la presenza di congestione, ed il terzo serve a discriminare due tipi di payload.

Priorità (1 bit) Rispetto alla possibilità di perdere la cella: se a `0` la cella non dovrebbe essere perduta, mentre può essere perduta, ad esempio in caso di congestione, se il bit è a `1`.

Checksum dello header (8 bit) Serve a controllare la correttezza dello header, e per sincronizzarsi sull'inizio della cella.

Il primi quattro bit sui cammini da nodo a rete vengono utilizzati per il *controllo del flusso*. L'intenzione, è quella di riuscire a supportare adattatori che offrano lunghezze di pacchetto variabili, con traffico soggetto a picchi di breve durata.

Il controllo del flusso si ottiene con un meccanismo a *sliding window*, che non vediamo nel dettaglio. I quattro bit non contengono, come avviene in

LAPB, la dimensione della finestra o un riscontro. Il valore della finestra viene invece memorizzato nel nodo utente (ricordo che i quattro bit sono presenti sui cammini da nodo a rete). Il meccanismo della finestra si applica soltanto ai canali che si definiscono *controllati*, ed è la rete che controlla il flusso dei dati proveniente dal nodo utente.

Una prima indicazione dà la disponibilità della rete a ricevere dal nodo utente: questo è un via libera incondizionato per tutte le celle appartenenti a *canali non controllati*. Per i *canali controllati*, ogni cella spedita decrementa un contatore interno. Le celle dei canali controllati non vengono spedite se il contatore è a zero, ma la sottorete può autorizzare il nodo utente a reinizializzare il contatore al suo valore massimo predeterminato.

Il campo destinato al *controllo degli errori* nello header consiste in un codice *Cyclic Redundancy Check* (CRC), generato sulla base del polinomio $X^8 + X^2 + X + 1$. Il codice prodotto è lungo 8 bit, ed è molto ridondante rispetto alla stringa codificata, lunga solo 32 bit. Quindi la rilevazione dell'errore è molto efficace, ed accade che per errori che coinvolgano un singolo bit si possa ricostruire lo header corretto.

Il campo per il controllo degli errori viene utilizzato anche per la sincronizzazione: l'inizio della cella viene infatti rilevato osservando una sequenza di 4 ottetti seguiti dal codice CRC corretto. La sincronizzazione viene confermata dopo che sono state ricevute correttamente un certo numero di celle.

Lo strato adattatore

Vengono definite 4 classi di *ATM Adaptation Layer* (AAL), ciascuno adatto a usi specifici. Per ciascuno di questi viene definito un formato del payload, estratto dalla cella ATM. Questo payload sarà a sua volta composto da uno header, e da un ulteriore payload, destinato all'applicazione.

Lo strato adattatore è composto a sua volta di due strati: uno che provvede alla *segmentazione* o al *riassemblaggio* delle celle per organizzarle in un frame di lunghezza superiore, ed uno strato di *convergenza* che realizza le diverse interfacce di adattamento.

Ciascuno dei due strati incapsula i payload dello strato o sottostrato superiore in una PDU caratterizzata da uno header e da un trailer, come rappresentato in figura 2.10.

Gli adattatori sono suddivisi in 4 classi: di ciascuna vediamo le caratteristiche principali, e la descrizione sommaria del formato della PDU del *sottostrato di convergenza* e del *sottostrato di segmentazione e riassemblaggio*.

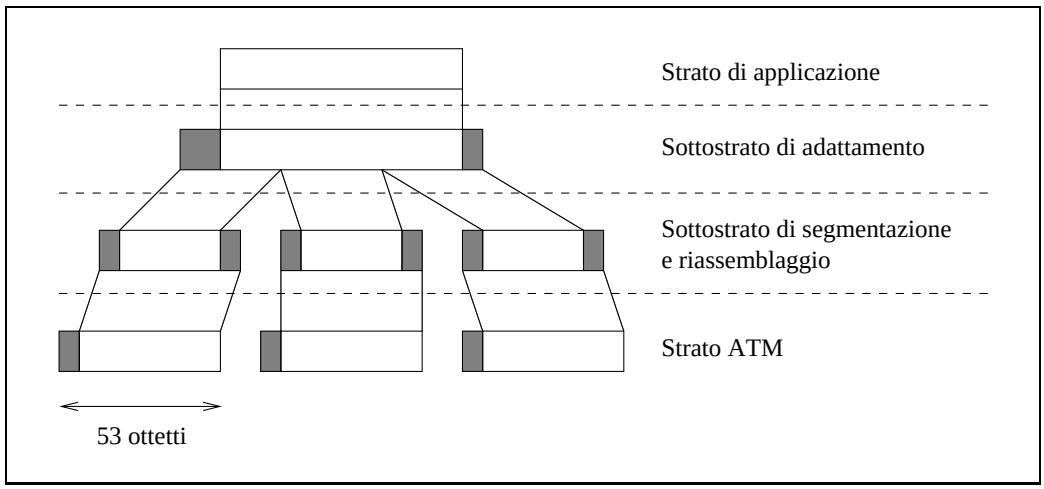


Figura 2.10 Relazione tra le PDU degli strati e sottostrati ATM

AAL1 Gestisce comunicazioni caratterizzate da un *bit rate costante* (*Constant Bit Rate* (CBR)), che quindi offrono una emulazione di un circuito in tempo reale. L'uso principale è per la trasmissione di voce su reti ATM, o per connessioni *Integrated Services Digital Network* (ISDN). Lo *header di convergenza* non è definito, e non dovrebbe essere necessario. Lo header di *segmentazione e riassemblaggio* contiene 4 bit che servono per la sequenzializzazione delle celle, ed altri 4 di codice di correzione e controllo per i quattro precedenti. I restanti 47 ottetti contengono un segmento della PDU prodotta dallo strato di convergenza.

AAL2 Anche in questo caso la comunicazione è in tempo reale, ma caratterizzata da un *bit rate variabile* (*Variable Bit Rate* (VBR)): l'applicazione tipica è la video conferenza. Non sono definiti i formati delle PDU dei due sottostrati.

AAL3 e 4 Si tratta di servizi che non richiedono risposte in tempo reale e che utilizzano *bit rate variabili*: il sistema utente specifica una capacità di picco, una media, e la durata massima dei picchi. Può essere utilizzata per servizi generici come transazioni bancarie, servizi di prenotazione, o per supportare il protocollo IP. La PDU dello strato di *convergenza* presenta tanto uno header, che contiene informazioni per l'identificazione del frame e per il dimensionamento dei buffer, quanto un trailer che replica l'identificazione

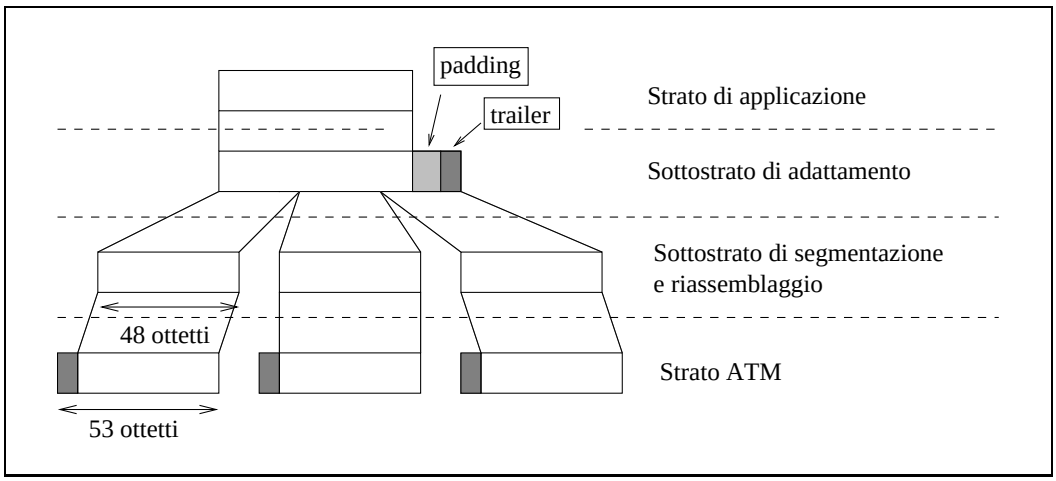


Figura 2.11 Le PDU nei diversi strati e sottostrati nello strato di adattamento 5

contenuta nello header e specifica la lunghezza del payload. Lo strato di *segmentazione e riassemblaggio* di questo tipo di adattatore realizza il *riassemblaggio* e la *segmentazione* di un frame in celle, ed inoltre è in grado di distinguere 4 diversi tipi di segmento, e di realizzare il multiplexing di 1024 comunicazioni sullo stesso canale. La dimensione del payload, che contiene un segmento della PDU prodotta dallo strato di convergenza, si riduce a 44 ottetti.

AAL5 Consente l'emulazione di una *Local Area Network* (LAN) ad alte prestazioni e ad alta affidabilità. Per limitare l'overhead di comunicazione, la PDU proveniente dallo strato di convergenza viene suddivisa in celle di 48 ottetti, che vengono spedite senza ulteriore incapsulamento (v. figura 2.11). L'ultima cella viene marcata nello header ATM, utilizzando un bit nel campo destinato al *tipo del payload*. La PDU del sottostrato di *convergenza* viene dunque portata ad una lunghezza multipla di 48 ottetti aggiungendo ottetti di *padding* al termine del payload. La PDU di convergenza non presenta header, ma un trailer che contiene 4 ottetti per l'identificazione e la dimensione del frame, e 4 ottetti per il codice di rilevazione di errore dell'intero frame.

Lo strato di adattamento AAL5 presenta una notevole efficienza, visto che ogni frame porta un overhead di soli 8 ottetti, indipendentemente dalla sua lunghezza. Questo a fronte di alcuni svantaggi:

- Le celle devono arrivare nello stesso ordine in cui sono spedite, pena

l'invalidamento dell'intero frame. Il codice di rilevazione d'errore rileva efficacemente questo genere di errori.

- Non è possibile intercalare la trasmissione di più frame.
- È necessario un padding, che porta il trailer ad occupare gli ultimi bit dell'ultima cella: altrimenti sarebbe difficile da localizzare.

Esercizi

- Utilizzando un linguaggio di programmazione o un foglio di calcolo scrivete un programma che realizzi la funzione $U(t)$ definita a pagina 39. Fornite quindi dei dati in ingresso che rappresentino una brusca variazione del fattore di utilizzazione misurato ρ. Qual è l'influenza del *guadagno* K?
- La tecnica del *bit stuffing* comporta una perdita di capacità della linea di comunicazione. Potete quantificarla?

Capitolo 3

LAN: Local Area Network

Quando le distanze da coprire si riducono, tecniche di comunicazione inutilizzabili sulle lunghe distanze diventano accessibili e convenienti: questa è una delle ragioni che giustifica la differenziazione tra tecnologie per reti a copertura geografica, le WAN, e quelle per installazioni localizzate, adatte a coprire distanze dell'ordine di pochi chilometri.

Un'altra ragione è tuttavia storica: mentre le reti a copertura geografica risentono tutt'ora dell'inerzia dovuta alla presenza di una rete telefonica già posata, le reti locali, nate da esigenze schiettamente informatiche, si sono subito rivolte a tecnologie più adatte alla trasmissione di dati.

Nel prossimi capitoli vedremo i due protocolli più studiati ed applicati, Ethernet (IEEE 802.3) e Token ring (IEEE 802.5), e accenneremo all'applicazione della tecnologia ATM nella progettazione di reti locali, che apre la prospettiva di far convergere tanto i servizi di telefonia quanto i servizi di trasmissione dati sulla stessa rete. Descriveremo infine le possibili applicazioni delle reti locali wireless.

3.1 IEEE 802.3 – Ethernet

Il protocollo IEEE 802.3 nasce da precedenti esperienze di protocolli che condividevano un unico dispositivo di comunicazione. Il problema fondamentale di questi protocolli erano le regole per risolvere i conflitti di accesso alla risorsa di condivisa.

Una prima soluzione venne introdotta con il protocollo ALOHA negli anni '70, ed era adatta alla comunicazione radio. In questo caso la risorsa da

condividere era la frequenza di trasmissione, che non poteva essere impegnata contemporaneamente da più trasmettitori.

Il protocollo consiste in questo: quando si rende necessaria la trasmissione di una PDU (o *frame*), questa viene spedita senz'altro. Ogni stazione è costantemente in ascolto, e riceve ogni frame spedito: se il contenuto del frame è corretto (controllato con un campo di rilevazione d'errore) e se il frame è destinato a quella stazione, la stazione consegna il *frame* allo *strato* superiore e spedisce un *riscontro* al mittente. Il mittente considera la spedizione riuscita se riceve il *riscontro* prima di un certo timeout, altrimenti rispedisce il frame.

Il problema principale di ALOHA è l'uso ridotto della risorsa: si raggiungono rapidamente condizioni di frequente collisione, e conseguente invalidazione, di frame. Utilizzando un certo modello probabilistico per la spedizione dei frame, risulta che viene utilizzato circa il 18% della banda disponibile [26]!

Per risolvere il problema, il protocollo *slotted ALOHA* fissa la dimensione del frame, e viene sincronizzato l'istante di trasmissione su tutte le stazioni. Viene così eliminata la possibilità di collisione *durante* la trasmissione: mentre in ALOHA la collisione con un certo frame avviene da parte di frame che possono essere spediti fino ad una lunghezza di slot prima o una dopo, nello *slotted ALOHA* si riduce questa finestra al solo slot precedente: se due *frame* risultano pronti alla spedizione nello stesso slot di tempo, collideranno all'inizio del prossimo slot.

Di conseguenza la massima utilizzazione viene raddoppiata ($\mathcal{U} \approx 36\%$). Ma, come nel caso di ALOHA, superata la soglia di traffico ottimale il sistema si avvia alla congestione, con una rapidissima perdita di prestazioni.

Il dato non è dunque entusiasmante, soprattutto perché queste tecniche sfruttano poco una importante caratteristica di una rete *locale* a bus: il tempo di propagazione di un segnale su tutta la rete è estremamente breve rispetto al tempo di trasmissione di un frame, almeno fintantoché la capacità della rete si mantiene sufficientemente bassa.

Ad esempio, per un frame di 1Kb, il *tempo di trasmissione* t_t su una rete a 10Mbps è di circa 100 μsec, mentre il *tempo di propagazione* t_p del segnale su un cavo di 2 Km è di circa 10 μsec (la velocità di propagazione del segnale in un cavo coassiale è circa la metà di quella della luce nel vuoto). È tuttavia da notare che queste considerazioni cessano di essere valide quando la *capacità* e la lunghezza della rete crescono, restando inalterata la velocità di propagazione del segnale: queste sono le condizioni che si verificano attualmente, con capacità dell'ordine del Gbps e coperture di chilometri.

Fintantoché il tempo di propagazione è molto superiore al tempo necessario a spedire un frame, è dunque opportuno ascoltare il segnale *prima della trasmissione* e così limitare la collisione entro un intervallo pari al tempo di propagazione. Questo procedimento viene detto *Carrier Sense Multiple Access* (CSMA).

Quindi, prima di iniziare la trasmissione del frame, la stazione verifica se sul bus è già presente un segnale, e solo se ciò non accade inizia a trasmettere: quindi la finestra di collisione è limitata a due volte il massimo ritardo di propagazione del segnale, e si elimina anche la necessità di sincronizzare gli *slot*.

Il *coefficiente di massima utilizzazione* ($\mathcal{U}$) della rete, corrispondente alla massima utilizzazione può essere calcolato con la semplice formula:

$$\mathcal{U} = \frac{1}{1 + \frac{t_p}{t_t}}$$

dove t_t è il tempo di trasmissione di un frame, mentre t_p è il tempo di propagazione del segnale: con i due dati dell'esempio precedente si ottiene a $\mathcal{U} = 90\%$. Inoltre non si presenta una degradazione catastrofica in caso di congestione.

Quando accade che due frame collidono, tuttavia, la disponibilità del canale viene perduta per tutto il tempo necessario a trasmettere l'intero frame: un ulteriore miglioramento, e qui arriviamo al *Carrier Sense Multiple Access - Collision Detection* (CSMA/CD) utilizzato nel protocollo 802.3, consiste nell'arrestare la trasmissione (*back-off*) appena si percepisce, ascoltando il canale *durante* la trasmissione, che si verifica una collisione. In questo modo la durata di una trasmissione inutile è limitato al tempo necessario a percepire la collisione, che è dello stesso ordine del tempo di propagazione del segnale.

Il protocollo IEEE 802.3 aggiunge ancora due dettagli:

- Quando rileva un conflitto su una propria trasmissione, la stazione invia un segnale di disturbo, in modo che tutti rilevino allo stesso modo la collisione.

- Prima della successiva ritrasmissione viene atteso un tempo, generato casualmente la prima volta, e poi raddoppiato, sino a sopprimere la spedizione del pacchetto.

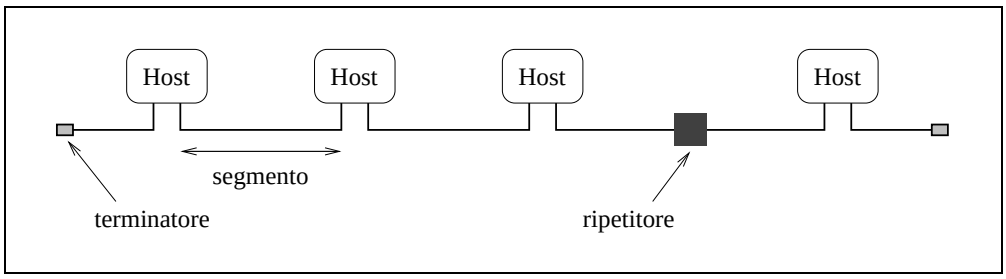


Figura 3.1 Cablaggio di una rete Ethernet a *bus*

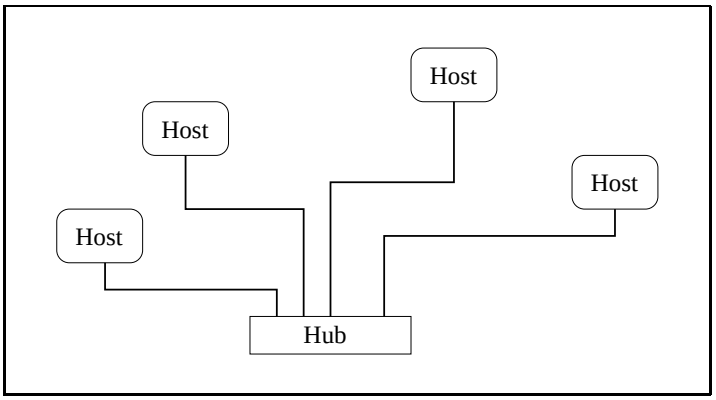


Figura 3.2 Cablaggio di una rete Ethernet a *stella*

La seconda tecnica evita che tentativi successivi siano sincronizzati, e quindi replichino la situazione di conflitto, ed allevia le situazioni di congestione rallentando progressivamente il ritmo di trasmissione.

La modalità con cui viene rilevata la collisione a livello fisico nel protocollo IEEE 802.3 è molto semplice. Poiché il segnale elettrico generato da un trasmettitore è di ampiezza nota, quando due o più trasmettitori trasmettono simultaneamente l'ampiezza del segnale raddoppia, e questo è un chiaro segnale di collisione.

Affinché tutto ciò sia vero, tuttavia, le caratteristiche del *bus*, ed in particolare la sua lunghezza, devono essere tali che l'attenuazione non porti l'ampiezza di due segnali in collisione ad essere equivocabile con un singolo segnale. Quindi la lunghezza del bus non deve superare una certa soglia, determinata dal tipo di cavo utilizzato e dalle frequenze in gioco. Nel caso di 10Mbps, il cavo di migliore qualità consente una lunghezza massima di 500 metri, che può essere estesa a 2500 metri con l'uso di ripetitori.

La topologia può essere a *bus* (v. figura 3.1), basata su un unica linea che

attraversa i *transceiver* collocati negli host, o a *stella* (v. figura 3.2), basata su un dispositivo elettronico, lo *hub*, a cui ciascun transceiver è collegato da una linea dedicata. Le diverse topologie incidono anche sulla logistica della rete, e sulla posatura delle linee necessarie.

Il protocollo IEEE 802.3 detta anche una semplice identificazione per le diverse implementazioni, proprio a partire dalle caratteristiche del protocollo fisico. L'identificazione è composta di tre parti: la prima indica la capacità operativa, il secondo il metodo di codifica dei segnali, ed il terzo la massima lunghezza del bus, in centinaia di metri. Questi sono gli identificatori più noti, con le rispettive caratteristiche della linea:

10BASE5 Cavo coassiale da 10mm di diametro, impedenza 50 Ohm, codifica Manchester – La massima lunghezza di un segmento di bus è di 500 metri, con un massimo di 100 nodi.

10BASE2 Cavo coassiale da 5mm di diametro, impedenza 50 Ohm, codifica Manchester – La massima lunghezza di un segmento di bus è di 185 metri, con un massimo di 30 nodi.

10BASE-T Doppino di categoria 3 (voce), codifica Manchester – La massima lunghezza di un segmento di bus è di 100 metri. La topologia è a stella, con un nodo centrale (*hub*) che ha solo la funzione di interconnessione.

10BASE-FP Fibra ottica – Un adattamento alla tecnologia a fibre ottiche (F sta per *fiber*), che consente 500 metri di lunghezza del segmento. Si tratta di nuovo di una connessione a stella: lo hub "riflette" i segnali provenienti dalle linee di ingresso su tutte le uscite. Anche in questo caso la codifica è Manchester, con lo `0` rappresentato da assenza di luce, e `1` rappresentato dalla presenza di luce. La codifica sottoutilizza la banda disponibile.

100BASE-TX Doppino di categoria 5, codifica MLT-3 – La massima lunghezza di un segmento di bus è di 200 metri. La topologia è a stella, con un nodo centrale (*hub*) che ha solo la funzione di interconnessione.

100BASE-FX Fibra ottica, codifica 4B5B NRZI – La topologia è a stella, con lunghezza massima del segmento di 400 metri.

100BASE-T4 Doppino di categoria 3, codifica 8B6T NRZ-L – Vengono usati 4 doppini di qualità inferiore (uno in trasmissione, uno in ricezione e due bidirezionali), per raggiungere i 100Mbps. La massima lunghezza di un

segmento di bus è di 200 metri. La topologia è a stella, con un nodo centrale (*hub*) che ha solo la funzione di interconnessione;

1000BASE-SX Fibra ottica multi-mode – La lunghezza del segmento va da 275 a 550 metri.

1000BASE-LX Fibra ottica multi-mode o single-mode – La lunghezza del segmento va da 500 metri a 5 chilometri.

1000BASE-CX Doppino schermato – La lunghezza del segmento è di 25 metri.

1000BASE-T Doppino di categoria 5. La lunghezza massima del segmento è di 100 metri. Vengono utilizzati 4 doppini per ciascuna linea.

Nel caso di protocollo 1000BASE, con capacità del canale dunque di 1 Gbps, si presenta il caso in cui il tempo di trasmissione di un frame è inferiore al tempo di propagazione di un segnale: infatti il tempo di propagazione resta condizionato alla velocità della luce, ma il tempo per la trasmissione del segnale si riduce di 100 volte, rispetto ad una rete 10BASE (nelle stesse condizioni dell'esempio a pag. 54 otteniamo $t_t = 0.5\mu$sec, contro $t_p = 5\mu$sec) e si tornerebbe a prestazioni ALOHA. Vengono quindi introdotti due artifici:

- frame corti vengono allungati opportunamente (padding), e
- se sono presenti più frame pronti, questi vengono inclusi nello stesso frame lungo.

Excursus – I cavi in fibra ottica

La tecnologia in fibra ottica offre grandi possibilità, ma ha i suoi limiti.

Il limite principale è dovuto alle riflessioni internamente alla guida. Infatti i segnali seguono percorsi diversi: quanto più è larga la guida, tanto più si smussano i fronti d'onda e si indeboliscono i segnali, per fenomeni di interferenza che si hanno quando ciascuna onda segue percorsi di diversa lunghezza. Si dice che tali fibre sono *multi-mode*.

Questo effetto viene in parte attenuato con le guide *graded-index multi-mode* il cui indice di diffrazione aumenta dal centro verso la periferia: in questo modo un'onda incidente viene gradualmente corretta.

Per finire, le guide d'onda *single-mode* giungono ad avere il diametro dell'ordine della lunghezza dell'onda stessa: i percorsi di onde riflesse sulla superficie interna della guida hanno lunghezze prossime a quelle delle onde che seguono la guida, mentre il basso angolo di incidenza riduce l'attenuazione del segnale.

Excursus – Migration path: evoluzione senza rivoluzione

L'evoluzione delle implementazioni del protocollo IEEE 802.3 esemplifica molto bene il concetto di *migration path*, che aiuta l'affermarsi di una tecnologia.

L'utente che introduce una certa tecnologia in un processo produttivo vuole avere garanzie circa la durata e l'entità dell'investimento che l'introduzione della tecnologia comporta. Se la tecnologia ha vita breve, non sarà possibile recuperare i costi. Se i costi sono eccessivi l'investimento potrà essere inaccessibile.

Nel nostro caso, Ethernet è stato per molte imprese il "punto di accesso" alla tecnologia dell'informatica distribuita. Posare un cavo 10BASE2 o una rete di doppini 10BASE-T è stata una opzione accessibile a molte imprese.

Il progresso verso tecnologie più evolute tuttavia non può richiedere investimenti troppo elevati: il rischio è che la nuova tecnologia venga rifiutata dal mercato, perché troppo costosa. Quindi una nuova tecnologia deve presentarsi come una evoluzione graduale, piuttosto che come una rivoluzione. Nel tempo, si afferma la stabilità di una certa tecnologia, e cresce anche la fiducia e la motivazione di chi, seguendo una strategia più prudente, aveva preferito non investire su tecnologie avanzate.

L'esistenza di un *migration path*, un percorso che l'utente (in questo caso l'impresa) può seguire per raffinare gradualmente il proprio sistema, è quindi essenziale, ed il successo di Ethernet forse è proprio dovuto ad un costo di ingresso molto contenuto, e ad un *migration path* ben delineato: vediamo una possibile "evoluzione", ma altre ne esistono.

Ingresso – Il sistema iniziale conta una decina di PC connessi con una rete Ethernet a basso costo 10BASE2.

Aumento del numero dei PC – Si rende opportuna la suddivisione della rete iniziale in 4 sottoreti 10BASE2 ed una dorsale 100BASE-TX

Aumento del traffico di rete – La sottorete contenente i server viene privilegiata ed implementata con 100BASE-TX.

Aumento del numero dei PC – Introduzione di nuove reti 100BASE-TX, agganciate alla stessa dorsale.

Aumento del traffico di rete – Sostituzione della dorsale 100BASE con un 1000BASE-CX.

Ora l'impresa potrebbe essere pronta per "entrare" nella tecnologia ATM, sostituendo la dorsale 1000BASE con uno switch ATM.

Excursus – Le codifiche dei dati

La codifica dei dati può svolgere un ruolo determinante nell'evoluzione di un protocollo.

Il modo più immediato per trasmettere i dati consiste nell'associare un certo livello di tensione all'`1` logico, ed un altro livello di tensione allo `0`. Una certa informazione binaria sarà dunque codificata con una sequenza di livelli di tensione di durata predeterminata, ciascuno rappresentante un bit. Esistono comunque molti modi di codificare i dati usando questa semplice tecnica: ad esempio, un livello alto può rappresentare `1`, ed uno basso uno `0` – codifica *NonReturn to Zero - Level* (NRZ-L), oppure un cambiamento di livello può rappresentare 1, ed un livello costante può rappresentare zero – codifica *NonReturn to Zero - Inverted* (NRZI).

Il difetto principale di questa tecnica sta nel fatto che il segnale può restare per lungo tempo costante: ad esempio, una lunga serie di `1` dà luogo ad un segnale a livello alto prolungato nella codifica NRZ-L. Questo può dare problemi di sincronizzazione: infatti il cambiamento di livello del segnale porta con sé una informazione temporale preziosa.

La *codifica bifase* elimina questo inconveniente: in questo caso ogni intervallo destinato a rappresentare un bit contiene sempre una transizione di stato al centro dell'intervallo. Nella *codifica Manchester*, un `1` viene rappresentato come una transizione intermedia da livello alto a basso, mentre uno 0 con una transizione intermedia da basso ad alto (secondo IEEE 802.3). Nella *codifica Manchester differenziale* la transizione intermedia è sempre presente, mentre una transizione all'inizio dell'intervallo indica lo `0`, mentre la sua assenza indica l'`1`.

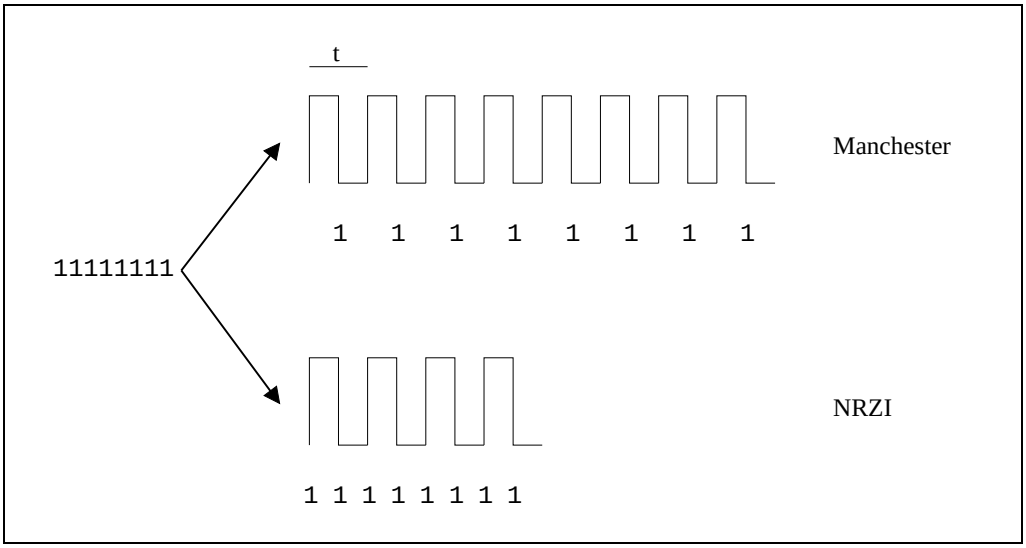


Figura 3.3 Confronto tra le codifiche Manchester ed NRZI

Purtroppo la codifica Manchester comporta un cattivo uso della capacità della linea: infatti per inviare un singolo bit può servire una completa alternanza `1-0`, quindi due bit, mentre per le codifiche NRZ è sempre sufficiente un singolo livello di segnale, quindi un bit. Ad esempio, in figura 3.3 confrontiamo i tempi necessari alla codifica Manchester ed alla codifica NRZI per trasmettere la stessa sequenza di bit, un ottetto composto di soli `1`: utilizzando lo stesso periodo τ per i due segnali otteniamo che la Manchester impiega un tempo doppio rispetto alla NRZI. In altri termini, con una certa larghezza di banda la codifica Manchester trasmette con un bit-rate che è la metà di quello della NRZI.

Per risolvere questo problema si ricorre ad una soluzione combinata, che consiste nel codificare tutte le sequenze di bit di lunghezza determinata in sequenze di bit leggermente più lunghe, ma con almeno una transizione di stato, e quindi a codificarle NRZ. In questo modo si ottiene il risultato desiderato di introdurre i cambiamenti di stato necessari alla sincronizzazione, evitando la ridondanza 2 a 1 della codifica Manchester. La *codifica 4B5B NRZI* vista sopra per 100BASE-FX, è di questo genere: 4B5B indica che 4 bit di dati vengono codificati in una sequenza di 5. I codici sono scelti in modo che, una volta che i dati sono stati ancora codificati NRZ, sia semplice ricostruire la sincronizzazione.

Si ottengono risultati simili con la codifica MLT-3 che tramite successi-

ve ricodifiche, riesce a generare un segnale di ampiezza di banda ristretta. Anche la codifica 8B6T ottiene risultati analoghi, utilizzando segnali ternari (livello alto, zero, livello negativo) in uscita: sequenze di 8 bit in ingresso (corrispondenti a 256 alternative) vengono tradotti in 6 simboli ternari in uscita (729 alternative). Utilizzato nella 100BASE4T, ciascuna delle tre coppie di bit di uscita viene immessa su uno dei tre doppini utilizzati, che quindi supporta 33 dei 100Mbps.

3.1.1 Formato dello header IEEE 802.3

Un *frame* IEEE 802.3 contiene i seguenti campi ([26, 11]):

Preambolo (7 ottetti) Ciascuno ottetto contiene una sequenza `10101010`, destinati a sincronizzare mittente e destinatario.

Delimitatore (1 ottetti) Contiene la sequenza fissa `10101011` destinata a indicare l'inizio della parte significativa del frame.

Destinatario (6 ottetti) È l'indirizzo Ethernet della scheda cui è diretto il frame.

Mittente (6 ottetti) È l'indirizzo Ethernet della scheda che ha generato il frame.

Lunghezza (2 ottetti) È la lunghezza del frame, espressa in ottetti. Questa nello standard non deve essere superiore a 1500 ottetti, e il frame può non contenere dati.

Dati Questo campo contiene il *payload* del frame.

Padding Si tratta di una sequenza di ottetti inutilizzati, necessari ad allungare il frame nel caso sia troppo corto. La lunghezza complessiva del payload e del padding deve essere superiore a 48 ottetti.

Checksum (4 ottetti) Un codice CRC per rilevare errori di trasmissione; viene calcolata sul frame escludendo il preambolo, il delimitatore e la checksum.

L'indirizzo Ethernet consiste di sei ottetti, rappresentati come una sequenza di sei numeri esadecimali di due cifre (da `00` a `FF`), separati da "`:`".

Ad esempio `08:00:39:00:2F:C3` è un indirizzo Ethernet valido. Ogni scheda Ethernet porta una *Electrically Programmable Read Only Memory* (EPROM) in cui è registrato alla fabbrica l'indirizzo Ethernet, che dunque identifica univocamente una certa scheda.

La lunghezza di un frame quindi, escludendo preambolo e delimitatore, varierà tra un minimo di 66 ottetti (18 di header e trailer, e 48 di dati e padding) ed un massimo di 1518 ottetti (18 di header e trailer, e 1500 di dati).

Ogni transceiver Ethernet ha il proprio indirizzo Ethernet, rappresentato da 6 ottetti, ed è in grado di selezionare la ricezione dei soli frame che nel campo destinatario riportino l'indirizzo proprio del transceiver, o l'indirizzo di *broadcast* `FF:FF:FF:FF:FF:FF`.

3.2 IEEE 802.5 – Token ring

Anche il protocollo *token ring* si pone il problema di coordinare l'accesso ad un unico dispositivo di comunicazione condiviso tra tutti gli *host*.

Il protocollo di arbitraggio si basa sull'uso di un piccolo frame, chiamato *token*. Ne esiste solo uno nella rete, configurata ad anello o a stella, e circola continuamente da un nodo al suo vicino. Quando un nodo ha un frame da inviare, attende di ricevere il token; quindi, invece di rispedirlo, spedisce il proprio frame. Questo verrà letto da tutti i nodi, per essere alla fine ricevuto nuovamente dal mittente, che lo ritirerà, rispedendo il *token*.

Più precisamente, il *token* verrà rispedito quando saranno verificate ambedue le condizioni seguenti:

- il frame è stato spedito interamente e
- il mittente riceve il preambolo del frame.

L'ordine in cui le due condizioni si verificano dipende dalla lunghezza della rete, e dalla lunghezza del frame: se il tempo di propagazione del frame (t_p) è inferiore al tempo necessario a trasmetterlo (t_t), la seconda condizione si verificherà per prima. Altrimenti si verificherà prima la seconda.

Ad ogni modo, dopo il reinserimento del token nell'anello, altri nodi potranno prelevarlo e trasmettere il proprio frame.

Notate come in condizioni di scarso carico, il protocollo è inefficiente: per trasmettere due frame, un nodo deve necessariamente intercalarli con

un periodo durante il quale fa circolare il token. Tuttavia le prestazioni migliorano in condizioni di carico massimo, proprio quando è necessario l'uso ottimale delle risorse. Infatti quando tutte le stazioni hanno un frame da spedire il protocollo segue una semplice politica *round-robin*, abilitando uno dopo l'altro, ciclicamente, tutti i nodi dell'anello.

Uno dei problemi principali del *token ring* consiste nella gestione del token: non deve essere duplicato, e non può andare perduto. Ciò che accade è che il compito di gestire il token viene assegnato ad una stazione. La perdita del token, così come il guasto del nodo responsabile della sua gestione, sono risolti da opportuni algoritmi che non vengono qui trattati. Il lettore interessato può trovare utili riferimenti in [26].

3.2.1 Il formato del frame nel protocollo IEEE 802.5

Descriviamo invece brevemente il formato del frame del protocollo *token ring*:

Delimitatore iniziale (1 ottetto) Per indicare l'inizio del frame. Ha un formato fisso `JK0JK000` dove `J` e `K` indicano due simboli che non corrispondono ad un dato valido nella codifica Manchester: si tratta di due livelli alti e due livelli bassi consecutivi.

Controllo dell'accesso (1 ottetto) I primi tre bit indicano la **priorità** del frame, un bit che indica se si tratta del **token**, ed in questo caso il prossimo ottetto è direttamente il delimitatore terminale, un bit che indica se si tratta del bit di **monitor**, e tre bit di **prenotazione**.

Controllo del frame (1 ottetto) Indica se si tratta di un frame contenente dati, o se si tratta di un frame per il controllo del protocollo. I primi due bit determinano il tipo, ed i sei restanti sono significativi se si tratta di frame di controllo.

Destinatario (6 ottetti) Come in IEEE 802.3.

Mittente (6 ottetti) Come in IEEE 802.3.

Dati In quantità non limitata.

CRC (4 ottetti) Come in IEEE 802.3.

Delimitatore terminale (1 ottetto) Per indicare la fine del frame. Ha un formato fisso `JK1JK1IE` dove `J` e `K` indicano due simboli che non corrispondono ad un dato valido, `I` viene messo ad `1` se il frame è un frame intermedio in una sequenza di frame, ed `E` viene impostato ad uno da una stazione che rilevi un errore.

Stato del frame (1 ottetto) I primi due bit (replicati anche nella quinta e sesta posizione per controllo) indicano se l'indirizzo del frame è stato riconosciuto, e se il frame è stato copiato;

La (poca) fortuna del protocollo token-ring è stata legata alla caratteristica di richiedere una elaborazione interna al nodo. Questo fatto lega le prestazioni del protocollo a quelle del nodo di elaborazione (o alla scheda utilizzata per il la gestione della rete), e quindi non può raggiungere le prestazioni di protocolli essenzialmente "passivi", come l'IEEE 802.3.

La variante di maggior successo, successiva al protocollo IEEE 802.5 è nota come *Fiber Distributed Data Interface* (FDDI), ed è adatta ad una rete con una capacità di 100 Mbps. Le sue caratteristiche sono solo marginalmente differenti da quelle descritte per l'IEEE 802.5, ed è considerato come un protocollo di "seconda generazione" rispetto all'Ethernet, che tuttavia riesce a raggiungere capacità superiori con le recenti estensioni (1Gbps).

3.3 Reti locali basate sul protocollo ATM

Il vero salto generazionale potrebbe tuttavia realizzarsi con l'affermazione di protocolli ATM per reti locali. Rispetto ad Ethernet, infatti, il protocollo ATM offre gli strumenti di base per gestire modalità di traffico diverse, caratterizzate da *qualità del servizio* specifiche. In particolare, il protocollo ATM, che abbiamo già visto nel capitolo dedicato alle reti WAN, offre modalità appropriate allo scambio di dati asincrono, tipico dell'uso "classico" della rete, così come per la comunicazione con requisiti di tempo reale, tipico di uso "convergente" della rete, adatta anche al trasporto di comunicazioni telefoniche.

L'uso della tecnologia ATM si può tradurre, allo stato attuale della tecnologia, in tre diversi modi:

- Gateway verso una WAN con tecnologia ATM – Per ottenere il meglio delle prestazioni di una rete ATM, il gateway verso tale rete può utilizzare la stessa tecnologia per fare da relay tra reti locali con tecnologie

convenzionali, ed una dorsale ATM. Il gateway può offrire, ad esempio, protocolli Ethernet virtualmente senza collisioni a ciascuno dei nodi connessi.

- Dorsale ATM – Uno o più nodi ATM possono collegare altre reti.
- Rete locale ATM – I nodi sono collegati direttamente allo *switch* ATM.

3.4 Reti locali wireless

Una rete locale *wireless* ha caratteristiche molto diverse da una rete locale convenzionale. In molti casi ha caratteristiche di una rete "provvisoria", realizzata temporaneamente per gestire una situazione transitoria. Ad esempio, sul luogo di una grave calamità che abbia messo fuori uso l'infrastruttura di collegamento ordinaria (telefonica), sul campo di battaglia, o per una riunione. In tutti questi casi, si richiedono le funzionalità di una rete di comunicazione dati, ma il suo allestimento con tecnologie "cablate" avrebbe costi o tempi proibitivi. In questo caso si parla anche di reti *ad hoc*.

Esercizi

- A quale caratteristica della rete è collegata l'evoluzione da X.25 a frame buffer ad ATM?
- Supponendo che la velocità di propagazione in un cavo sia pari al 90% di quella della luce, quanto è lungo un ottetto in una rete Ethernet a 10Mbps? Qual è la distanza, in ottetti, tra due host distanti 500 metri? E se la rete fosse a 100 Mbps?
- Una linea ha capacità di 10Mbps. Quanto tempo è necessario a trasmettere 1000 ottetti con codifica NRZI, con codifica Manchester e con codifica 4B5B rispettivamente?
- Qual è la ragione per l'introduzione dei simboli `J` e `K` nel *token ring*.

Capitolo 4

Lo strato di rete IP

Nei due capitoli precedenti abbiamo considerato lo *strato data link*, ed abbiamo esaminato alcune delle tecnologie disponibili per la sua realizzazione. Notiamo la grande varietà di tecnologie disponibili, la rapidità della loro evoluzione, la loro relazione con le caratteristiche fisiche della rete: distanze e collegamento fisico tra i componenti.

Il compito dello strato superiore, lo *strato di rete*, è quello di astrarre da questa molteplicità, e di produrre una interfaccia di comunicazione che non dipenda dalla tecnologia utilizzata dallo strato data link. Un aspetto di questo requisito è la necessità di nascondere la topologia dello strato data link, costituita da collegamenti fisici, ed offrire agli strati superiori la possibilità di comunicare con un qualunque altro nodo, purché connesso al primo attraverso una catena di link fisici.

In figura 4.1 vediamo un frammento di una rete più vasta. Lo strato data link consente la comunicazione solo tra i nodi che sono collegati dalla stessa rete fisica: nella figura, le reti fisiche sono rappresentate da linee continue tra i rettangoli, che rappresentano i nodi di elaborazione. Tuttavia sottoreti diverse possono avere nodi in comune, che possono quindi istradare le comunicazioni di sottorete in sottorete, realizzando la comunicazione anche tra sottoreti distanti. La realizzazione di questa funzionalità è il compito fondamentale dello *strato di rete*.

Il modulo IP ([1]) svolge, nella tecnologia Internet, questa funzione fondamentale: infatti è in questo strato che si fondono le molteplici sottoreti e le tecnologie utilizzate per la connessione fisica dei loro nodi, trasformandole in una unica rete, quello che poi viene chiamato Internet, che offre una interfaccia di comunicazione standardizzata.

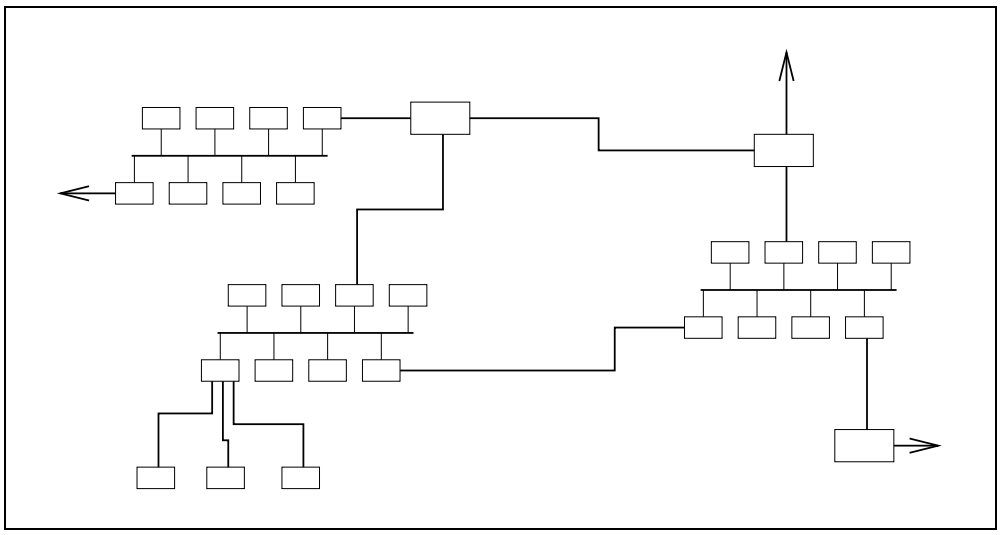


Figura 4.1 Esempio di un frammento di Internet

Una delle caratteristiche portanti di Internet è la sua *estendibilità*. Il modulo IP gioca un ruolo fondamentale nella realizzazione di questa caratteristica, poiché rende possibile una crescita progressiva dell'Internet attraverso il collegamento di nuove sottoreti alla rete preesistente, senza la necessità di coordinare le nuove sottoreti con la rete globale.

Poiché l'estensione non presenta problemi di consistenza globale, si è assistito ad una "esplosione" di Internet, con un andamento di crescita esponenziale dal 1993 (come si vede in figura 4.2), che è destinato a rallentare solo perché ormai prossimo alla saturazione.

Questa crescita vertiginosa può essere senz'altro attribuita anche all'assenza di informazione sullo stato globale della rete Internet nei singoli nodi. Secondo le specifiche di IP infatti, né il (singolo) modulo IP, né altri moduli degli strati superiori o inferiori conoscono l'effettivo percorso che un pacchetto ha seguito o deve seguire. Il percorso viene invece determinato dinamicamente dai moduli IP incontrati sul cammino, e non viene registrato (salvo in caso di *source routing*, come vedremo nel paragrafo 4.1.6). I moduli IP intermedi applicano euristiche basate sulle informazioni disponibili localmente o sui nodi immediatamente adiacenti.

Uno degli altri vantaggi di una euristica basata su dati locali, e non globali, consiste nel fatto che una nuova risorsa (ad esempio un nuovo link diretto o più veloce del precedente) diventa immediatamente disponibile a qualun-

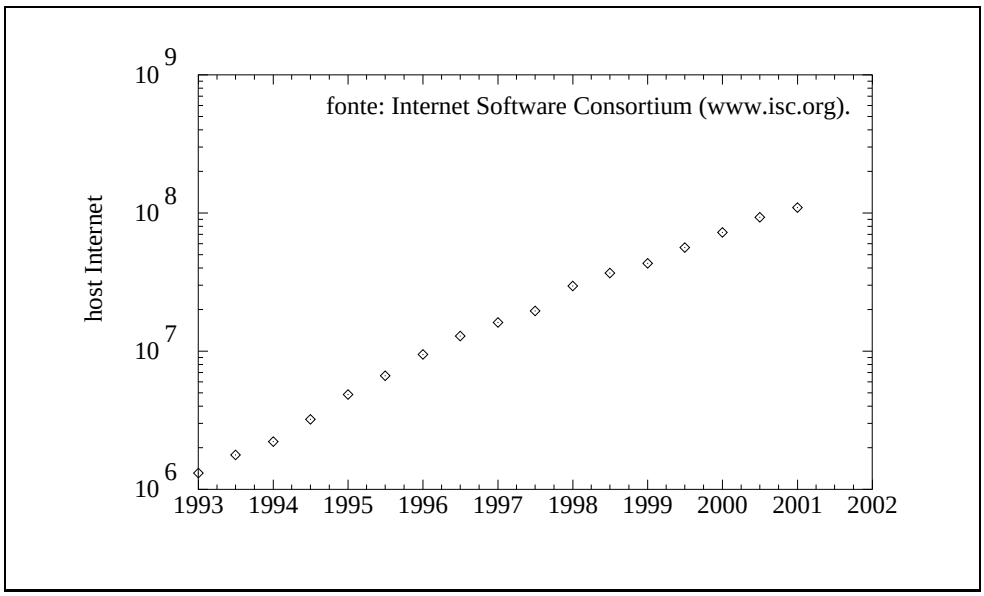


Figura 4.2 Crescita del numero di host in Internet (scala logaritmica sulle ordinate)

que altro nodo della rete, senza che per questo aggiornamento debba essere diffusa nessuna notizia.

Un'altra caratteristica importante di Internet è la possibilità di adeguarsi rapidamente e di trarre vantaggio dalle nuove tecnologie: infatti il modulo IP incapsula perfettamente l'hardware sottostante, e rende invisibili (ovvero trasparenti) agli strati superiori molte caratteristiche dell'hardware di comunicazione. Solo la qualità della comunicazione (più o meno affidabile, più o meno veloce) risulterà alterata.

Quindi una nuova tecnologia di comunicazione a livello hardware può essere resa immediatamente disponibile all'Internet, semplicemente realizzando un driver con l'opportuna interfaccia verso il modulo IP ed adattando i moduli IP per i quali è disponibile.

Un'altra caratteristica importante di Internet è la buona portabilità del software che lo utilizza: anche in questo caso IP gioca un ruolo importante. Poiché l'interfaccia offerta da IP agli strati superiori è standardizzata, è semplice "portare" una certa applicazione di rete su piattaforme hardware diverse, una volta che per queste sia stato realizzato il modulo IP.

La semplicità del **porting** è la base per la realizzazione della *interoperabilità*: una tecnologia facilmente riproducibile su molte piattaforme renderà possibili applicazioni interoperanti.

4.1 IP – Internet Protocol

Ogni strato dell'architettura Internet vista nel paragrafo 1.6 assolve alla funzione fondamentale di realizzare parte della astrazione di rete che viene utilizzata dagli strati superiori.

La specifica dell'*interfaccia* che il modulo IP offre allo strato trasporto consiste in un insieme di funzioni. Tali funzioni saranno rese disponibili ai programmi del modulo trasporto, che a loro volta le utilizzeranno per realizzare le loro funzionalità.

Le due funzioni principali consistono nella spedizione e ricezione di un *pacchetto*: una `send` ed una `receive`. Lo standard, che esamineremo dettagliatamente nel seguito, indica (cfr. [1], par. 3.3) una `send` asincrona e una `receive` bloccante, ma sottolinea che questo scenario è solo indicativo.

Tra i parametri di tali funzioni di interfaccia dovranno comparire l'*indirizzo IP* del mittente e del destinatario, ed il *buffer* contenente il pacchetto (cfr. [1], par. 3.3).

I parametri di ciascuna funzione verranno considerati dal modulo IP per produrre un pacchetto che incapsulerà come *payload* (che si può tradurre come "carico utile") i dati contenuti nel buffer. Oltre al *payload* il pacchetto comprenderà uno *header IP* (o "intestazione") contenente le informazioni necessarie per la gestione del pacchetto da parte dei moduli IP mittente, destinatario ed intermedi.

Il modulo IP invocherà a sua volta le funzioni del driver nello strato *data link*, che produrrà un frame da trasmettere. Il driver metterà dunque a disposizione del modulo IP altre funzioni di spedizione e ricezione.

Oltre all'interfaccia verso lo strato soprastante, il modulo IP di un nodo Internet realizza un'altra importante funzione: quella di determinare l'istradamento (o *routing*) su altre sottoreti dei pacchetti non diretti al nodo stesso, in modo da avvicinarli alla loro meta finale.

Per assolvere ai compiti di *interfaccia* e *routing*, il modulo IP realizza due funzioni fondamentali:

L'istradamento Questa funzione consente di raggiungere il destinatario indicato dal modulo trasporto mittente attraverso reti tra loro interconnesse. Ciascun passo nel percorso dal mittente al destinatario viene anche detto *hop*.

La frammentazione ed il riassemblaggio Le due funzioni complementari consistono rispettivamente nel trasmettere un pacchetto frammentandolo

in molti pacchetti, e nel ricostruire l'originale a destinazione. L'esistenza di queste funzioni è motivata dal fatto ogni supporto di comunicazione è caratterizzato da una quantità massima di informazione trasferibile con una singola operazione di comunicazione: questa quantità è detta MTU. Un certo pacchetto di dimensione x può incontrare sul suo cammino un supporto con MTU $< x$

Alla base della funzione di routing sta la rappresentazione dell'indirizzo associato ad un modulo IP.

4.1.1 Gli indirizzi IP

Un indirizzo IP è costituito da 4 ottetti, rappresentati con quattro numeri in base decimale nell'intervallo $[0, 255]$. Ad esempio `223.1.2.1` è un indirizzo di IP valido.

L'indirizzo IP assegnato ad un modulo dipende dalla rete sulla quale risiede il nodo che ospita il modulo: la parte più significativa dell'indirizzo IP identifica la rete in cui è collocato il nodo. Il resto corrisponde all'indirizzo del nodo, non diversamente da come il nome della via ed il numero civico costituiscono un indirizzo postale.

Ma il numero di ottetti considerati come indirizzo di rete può variare, corrispondentemente alla **classe** della rete: una rete è di classe C quando 3 ottetti sono destinati al numero di rete ed uno al numero di nodo, di classe B se 2 ottetti sono per il nome della rete, di classe A se 1 ottetto è per il nome della rete.

La classe è a sua volta determinata dal valore del primo ottetto dell'indirizzo IP, secondo lo schema seguente ([1]):

Classe A: `0nnnnnnn hhhhhhhh hhhhhhhh hhhhhhhh`
Classe B: `10nnnnnn nnnnnnnn hhhhhhhh hhhhhhhh`
Classe C: `110nnnnn nnnnnnnn nnnnnnnn hhhhhhhh`

dove `n` rappresenta un bit destinato a descrivere la rete, `h` un bit destinato a descrivere un host, `1` e `0` i due valori costanti.

Nell'esempio precedente, poiché 223 ha rappresentazione binaria `11011111`, l'indirizzo appartiene ad una rete di classe C: quindi i primi tre ottetti corrispondono all'indirizzo della rete (`223.1.2`) mentre l'ultimo ottetto (`1`) identifica un nodo.

Alcune combinazioni hanno significati particolari ([10]):

- La rete (di classe A) indicata da un numero il cui primo ottetto è `127` (`01111111`) indica l'host locale, ed un tale indirizzo non è mai presente in rete. In particolare l'indirizzo IP `127.0.0.1`, anche denominato *indirizzo di loopback*, e viene utilizzato per simulare una comunicazione in rete, utilizzando lo stesso nodo come mittente e destinatario della comunicazione.

- L'host indicato da un numero la cui parte host (di 1, 2 o 3 ottetti, a seconda della classe) è composta da ottetti con tutti i bit a `1` (corrispondenti al decimale `255`) indica tutti gli host della rete, ed è usato solo come destinatario per un *broadcast diretto*.

- L'indirizzo `255.255.255.255` indica tutti i nodi della rete fisica (indirizzo di *broadcast limitato*), ed è usato solo come destinatario.

- La rete o l'host composto da ottetti con tutti i bit a `0` non viene utilizzato (al più in modo transitorio durante l'inizializzazione).

Poiché il massimo numero di nodi che una rete può contenere dipende dalla sua classe, ad una rete viene assegnata una classe che cerca di anticipare la sua estensione: una rete di classe A dispone di più numeri per i suoi nodi di quanti non ne disponga una rete di classe C.

Lo spazio degli indirizzi è gestito da un unico ente sovranazionale, il *Network Information Center* (NIC), a cui è necessario rivolgersi quando vogliamo installare una nuova rete che sia visibile nell'Internet globale.

Ma, anche se non vogliamo che la nostra rete sia visibile globalmente, esiste una raccomandazione per l'indirizzo IP per la nostra rete: questo dovrebbe essere scelto tra i seguenti

`192.168.*.*`
`10.*.*.*`
`172.16-31.*.*`

Considerando attentamente il modo in cui vengono assegnati gli indirizzi IP, osserviamo che questo è in parziale contraddizione con un requisito base di IP: la trasparenza rispetto alla rete fisica. Infatti la distribuzione degli indirizzi IP ricalca proprio la struttura fisica della rete, ma un certo modulo IP risiede su un nodo i cui driver *data link* hanno pure un indirizzo proprio, che invece è indipendente dalla collocazione fisica del nodo!

La prima conseguenza di questa inconsistenza è che l'indirizzo IP e l'indirizzo *data link* pur essendo associati allo stesso oggetto fisico, non possono

essere correlati funzionalmente: proprio come il Codice di Avviamento Postale e la denominazione di una località non possono essere ricavati l'uno dall'altro (diversamente dal codice fiscale di una persona fisica. Per indicare entrambi senza errori è necessario disporre di una tabella che riporti la corrispondenza tra indirizzi *data link* ed indirizzi IP: è il protocollo *Address Resolution Protocol* (ARP) (*Address Resolution Protocol*) a gestire questa tabella.

4.1.2 Address Resolution Protocol

Consideriamo dunque brevemente la ridondanza che si viene a creare tra indirizzi *data link* ed IP: qui la trattiamo nel caso del protocollo Ethernet, ma presenta caratteristiche simili in altri protocolli *data link*.

In qualunque strato, di un messaggio si conosce l'indirizzo del mittente ed del destinatario in quello strato. In particolare, quando il modulo IP tratta un pacchetto IP è noto l'indirizzo IP del destinatario finale del pacchetto.

In qualunque strato, per ottenere il recapito di un messaggio è necessario invocare la funzione di spedizione predisposta dallo strato sottostante. Questa funzione deve essere a sua volta in grado di determinare l'indirizzo da utilizzare nel proprio strato. In particolare, è necessario che il modulo Ethernet conosca anche l'indirizzo *data link* del destinatario o di un nodo intermedio nel percorso verso il nodo destinatario. Infatti la tecnologia Ethernet è in grado di produrre una comunicazione tra due nodi collegati allo stesso cavo, cioè alla stessa sottorete fisica.

Dunque al modulo Ethernet deve essere comunicato l'indirizzo *data link*, che sappiamo essere correlato all'indirizzo IP. Ma la corrispondenza non può essere in alcun modo codificata in un algoritmo, così come non esiste modo di determinare che il CAP di Aosta è 11100, se non consultando una tabella. Bisogna dunque gestire una tabella che associ indirizzi IP ad indirizzi *data link*, nel caso in esame indirizzi Ethernet, come descritti nel paragrafo 3.1.1. La tabella 4.1 ne è un esempio.

Il modulo software che cura l'aggiornamento di questa tabella utilizza il protocollo ARP, ed assume la stessa denominazione. Il modulo ARP può essere realizzato come parte del driver Ethernet stesso o come modulo separato collegato al driver Ethernet ([15]).

Quando il driver Ethernet riceve un pacchetto da incapsulare e spedire, conosce l'indirizzo IP del destinatario: per poter costruire il frame da consegnare al dispositivo *hardware* deve conoscere l'indirizzo Ethernet del destina-

Tabella 4.1 La tabella ARP di conversione IP → Ethernet

Indirizzo IP	*Indirizzo Ethernet*
`223.1.2.1`	`08:00:39:00:2F:C3`
`223.1.2.3`	`08:00:5A:21:A7:22`
`223.1.2.4`	`08:00:10:99:AC:54`

tario (intermedio o finale). Quindi accede alla tabella ARP tramite il modulo ARP stesso, cercando la riga corrispondente all'indirizzo IP desiderato.

Ma chi costruisce la tabella ARP?

Salvo rare eccezioni, non può essere l'amministratore della rete a gestire la tabella ARP: sarebbe un lavoro troppo impegnativo e richiederebbe interventi troppo tempestivi, per far fronte a eventi di routine come lo spegnimento del nodo, o la sostituzione della scheda Ethernet. La tabella invece viene compilata automaticamente dal modulo ARP stesso.

L'inserimento di una nuova voce nella tabella viene innescata dalla richiesta di un indirizzo Ethernet corrispondente ad un IP non presente in tabella. In questo caso il modulo ARP trasmette in rete un *frame broadcast* contenente, nella parte riservata ai dati, l'indirizzo IP da risolvere.

Il campo destinatario di un *frame broadcast* è `FF:FF:FF:FF:FF:FF`, ed il campo mittente contiene l'indirizzo Ethernet del mittente. Il modulo ARP indicherà nel *tipo del frame* che si tratta di un frame ARP. In tabella 4.2 viene rappresentato il contenuto del frame di richiesta spedito da ARP.

Tabella 4.2 Frame di richiesta ARP

Tipo	*Richiesta ARP*
Indirizzo IP Mittente	`223.1.2.1`
Indirizzo Ethernet Mittente	`08:00:39:00:2F:C3`
Indirizzo IP Destinatario	`223.1.2.2`
Indirizzo Ethernet Destinatario	`FF:FF:FF:FF:FF:FF`

Una volta spedito il frame di richiesta ARP, il frame la cui spedizione richiedeva l'indirizzo Ethernet richiesto (il frame "in sospeso") viene messo in coda. Ogni driver Ethernet, ricevendo il frame in broadcast, riconoscerà il tipo del frame e lo consegnerà al proprio modulo ARP. Ma solo il modulo ARP che riconoscerà l'indirizzo IP destinatario come quello del nodo su cui risiede risponderà, con un frame che avrà come mittente se stesso, ora

completo del proprio indirizzo Ethernet, e come destinatario il mittente del precedente messaggio (v. tabella 4.3).

Tabella 4.3 Frame di risposta ARP

Tipo	*Risposta ARP*
Indirizzo IP Mittente	`223.1.2.2`
Indirizzo Ethernet Mittente	`08:00:28:00:38:A9`
Indirizzo IP Destinatario	`223.1.2.1`
Indirizzo Ethernet Destinatario	`08:00:39:00:2F:C3`

Ora il frame verrà ricevuto solo dal driver Ethernet interessato, e passato al modulo ARP residente, che completerà la tabella con il nuovo indirizzo (v. tabella 4.4).

Tabella 4.4 La tabella ARP di conversione IP → Ethernet

Indirizzo IP	*Indirizzo Ethernet*
`223.1.2.1`	`08:00:39:00:2F:C3`
`223.1.2.2`	`08:00:28:00:38:A9`
`223.1.2.3`	`08:00:5A:21:A7:22`
`223.1.2.4`	`08:00:10:99:AC:54`

Una possibile accortezza consiste nel memorizzare l'indirizzo Ethernet del mittente della richiesta anche nella tabella ARP del nodo che spedisce la risposta. Infatti, è probabile che i due nodi debbano successivamente scambiare pacchetti in entrambi i sensi, e quindi la predisposizione di entrambi gli indirizzi può risparmiare una successiva richiesta.

Affidabilità delle tabelle di ARP

Ogni protocollo Internet deve essere in grado di sopportare un funzionamento imperfetto, in particolare il guasto di nodi. Nel caso di ARP, se nessuno dei nodi che riceve la richiesta riconosce il proprio IP (ciò accade se il nodo in questione è guasto, o semplicemente spento), il messaggio ARP di richiesta rimarrà senza risposta.

Per fronteggiare questa evenienza, il mittente della richiesta predispone un timeout, in modo da rimuovere il pacchetto in sospeso dalla coda, se questo era stato accodato. Ma secondo una implementazione alternativa ugualmente

accettabile, il driver non accoda il pacchetto in sospeso, ma lo distrugge quando rileva che l'indirizzo ARP non è presente in tabella. Dipenderà dal protocollo allo strato trasporto rilevare la perdita del pacchetto e riproporne la trasmissione. Prima di allora il protocollo ARP avrà completato la tabella con l'indirizzo necessario (ma vedere anche [10], paragrafo 2.3.2.2).

Ciò corrisponde alle specifiche di IP, in quanto a IP non è richiesto di garantire il recapito dei dati che gli vengono consegnati dallo strato *trasporto*.

Ma le informazioni registrate nella tabella ARP possono essere invalidate da altri eventi, come lo spegnimento o la rimozione di un nodo.

Esistono alcune tecniche utili a fare fronte a questi eventi [10]:

- timeout – Scartare una voce nella tabella quando non viene utilizzata per un certo periodo, che si aggira intorno al minuto;
- polling – Interpellare periodicamente l'host, e rimuovere la sua voce in tabella se non risponde;
- rilevazione di errore – Se la successiva comunicazione con quell'host restituisce un errore, rimuovere la voce;
- rilevazione di errore – Come sopra ma tenendo conto di segnalazioni di errore dagli strati superiori.

4.1.3 Istradamento di un pacchetto IP

Il modulo IP è un po' il cuore della tecnologia Internet. Tramite questo modulo è possibile l'interconnessione tra più reti differenti. Il **routing**, cioè l'attività di istradare i pacchetti da una rete all'altra per farli giungere a destinazione, è tra le funzioni fondamentali di IP.

Alla base dell'attività di istradamento sta la gestione di tabelle opportune, che può essere compito tanto dell'**amministratore della rete** quanto di un algoritmo di routing distribuito.

Il routing diretto

Partiamo dal caso più semplice di una singola rete (figura 4.3), e poniamo che una comunicazione tra moduli TCP dia origine ad un pacchetto IP da A a B.

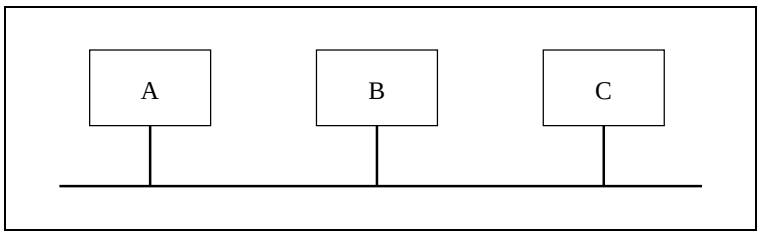


Figura 4.3 Una sottorete connessa allo strato *data link*

Il frame ed il pacchetto in esso incapsulato conterranno mittente e destinatario al rispettivo strato, come rappresentato in modo semplificato[1] nella tabella 4.5.

Tabella 4.5 Istradamento di un frame nel *routing diretto* – A e B sostituiscono gli indirizzi IP di A e B rispettivamente, a e b i corrispondenti indirizzi Ethernet

	Mittente	*Destinatario*
Frame Ethernet	`a`	`b`
Pacchetto IP	`A`	`B`

In questo caso il ruolo di IP è minimo: si tratta di *routing diretto*, ed il modulo di IP si limita ad incidere sul costo della comunicazione, in termini di tempo di elaborazione e di impegno della struttura di comunicazione.

Routing indiretto

Nella figura 4.4 vediamo un Internet più complesso: si tratta di tre reti, collegate insieme tramite il gateway D.

Tutte i nodi tranne D contengono lo *stack* Internet completo (quello di figura 1.7), mentre D potrà realizzare anche il solo modulo IP, ma avrà la struttura illustrata in figura 1.8, con tre driver ed un singolo modulo IP.

In questa rete un pacchetto spedito da B a F deve passare per D. Per poter effettuare il routing ciascuna rete Ethernet dovrà possedere un identificatore, l'indirizzo IP della rete. Sarà l'amministratore della rete a determinare il numero di rete, su indicazione di un ente sovranazionale. Infatti l'indirizzo IP deve individuare univocamente un nodo, su scala mondiale.

[1]Per semplificare la lettura sostituiamo gli indirizzi IP ed Ethernet con lettere, maiuscole per gli indirizzi IP, minuscole per gli indirizzi Ethernet; per un certo nodo viene utilizzata la stessa lettera (maiuscola o minuscola) eventualmente accompagnata da cifre.

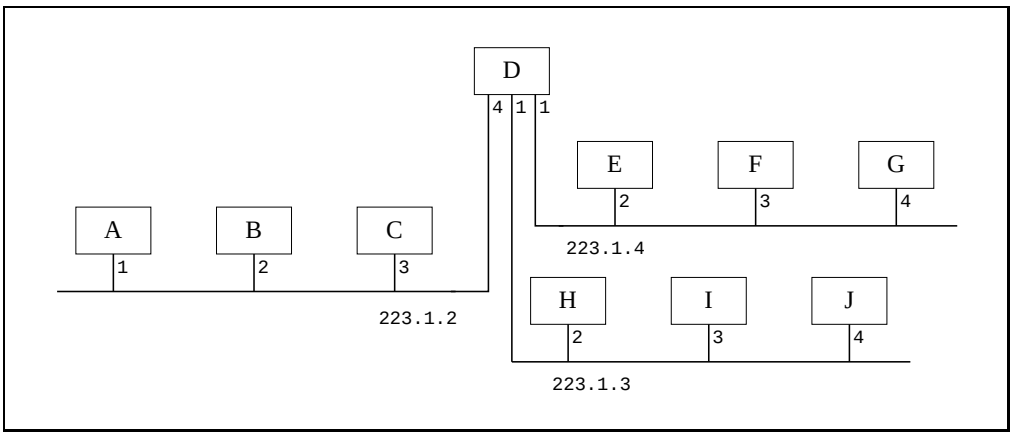


Figura 4.4 Una rete composta da tre sottoreti connesse tramite un gateway

Quando i due interlocutori (ad es. A e C) accedono alla stessa rete, la comunicazione può ancora avvenire per **routing diretto**: il meccanismo sarà lo stesso già visto in precedenza.

Il ruolo di IP diventa fondamentale quando la comunicazione avviene tra nodi appartenenti a reti diverse: ad esempio tra A ed E rispettivamente sulle reti `223.1.2` e `223.1.4`. In questo caso si sfrutta un routing diretto da A a D, e poi da D ad E, ed il gateway D decide che la comunicazione necessita della sua collaborazione, e "traghetta" il pacchetto tra i due Ethernet.

Tutto ciò avviene in maniera completamente trasparente agli strati superiori ad IP: quindi l'attraversamento del nodo D è invisibile tanto a TCP o UDP quanto ad altre applicazioni di strati ancora superiori.

Per ottenere la collaborazione del gateway, il modulo IP del mittente A deve sapere che il nodo E è collocato su una rete diversa dalla propria, e che il routing per quella rete è curato dal gateway D. Queste informazioni sono registrate nella **tabella di routing** di D. Esamineremo questa struttura nel seguito: ora accettiamo il fatto che, tramite la tabella, il nodo A determina che il pacchetto per E deve essere istradato al nodo D: il contenuto del pacchetto spedito da A ad E è indicato nella tabella 4.6.

Il destinatario del pacchetto (indicato dunque nel pacchetto IP) è E, ma il frame è diretto a D, cioè all'indirizzo Ethernet `d1`, corrispondente ad uno dei driver del nodo D sulla stessa rete dell'indirizzo Ethernet `a`, corrispondente al nodo A.

Il driver `d1` del nodo D riceverà il frame, e passerà il pacchetto al modulo IP del nodo D. Questo, utilizzando a sua volta le proprie tabelle di routing,

Tabella 4.6 Frame prodotto da A per E con routing indiretto via D

	Mittente	*Destinatario*
Frame Ethernet	`a`	`d1`
Pacchetto IP	`A`	`E`

determinerà che il pacchetto deve essere istradato al nodo E tramite il driver `d3`: il nuovo frame conterrà le informazioni in tabella 4.7.

Tabella 4.7 Frame prodotto da D per E (routing indiretto)

	Mittente	*Destinatario*
Frame Ethernet	`d3`	`e`
Pacchetto IP	`A`	`E`

Le regole di routing

Riassumiamo le regole di routing applicate da IP ad ogni pacchetto che riceve. Si definisce *pacchetto entrante* un pacchetto che proviene dallo strato *data link*, mentre un *pacchetto uscente* proviene dallo strato trasporto (v. [23], paragrafo 5.3).

- Per un pacchetto *uscente* il modulo IP deve decidere se il routing è diretto o indiretto, ed a quale driver indirizzarlo.
- Per un pacchetto *entrante* IP deve decidere se
  - consegnare il pacchetto allo strato superiore, o se
  - inoltrarlo ad un altro driver.

  Nel secondo caso verrà trattato come un pacchetto *uscente* (v. caso precedente).
- Un pacchetto *entrante* non è mai inoltrato sullo stesso driver da cui proviene.

Queste decisioni vengono prese consultando le tabelle di routing interne, prima di passare il frame al driver, e quindi prima di consultare la tabella di ARP.

4.1.4 La tabella di routing

Per decidere su quale rete fisica istradare un pacchetto il nodo accede alla **tabella di routing**.

Ogni riga della tabella di routing è intestata con un indirizzo IP: si può trattare tanto dell'indirizzo di un nodo, che di quello di una rete. Tra le informazioni contenute in ciascuna riga ci sono anche:

- il driver da utilizzare per l'istradamento di un pacchetto indirizzato a quel nodo o rete,
- un flag che indica se il routing è o meno diretto, e
- l'indirizzo del gateway, se il routing è indiretto.

La tabella di routing nel routing diretto

Vediamo il contenuto della tabella di routing nel caso più semplice di routing diretto: la tabella 4.8 illustra la riga della tabella nel nodo A relativa alla rete `223.1.2`.

Tabella 4.8 Routing diretto del nodo A sulla rete `223.1.2`

host/rete	*diretto/indiretto*	*gateway*	*interfaccia driver*
223.1.2	diretto	—	eth0

Quando il modulo IP del nodo A vuole spedire un pacchetto al nodo B, esegue le seguenti azioni:

1. Estrae la parte "numero di rete" dall'indirizzo IP del destinatario.
2. Ricerca nella tabella di routing la riga relativa a quella rete.
3. Rileva che la rete può essere raggiunta tramite *routing diretto*.
4. Accede alla tabella di ARP per ottenere l'indirizzo Ethernet del destinatario (v. protocollo ARP).
5. Invia il messaggio.

Nel caso che la seconda operazione (la ricerca nella tabella di routing) fallisca, si ottiene una segnalazione di errore del genere `Network unreachable`.

La tabella di routing nel routing indiretto

Prendiamo nuovamente in esame il sistema in figura 4.4. La tabella di routing di A è rappresentata in tabella 4.9.

Tabella 4.9 Tabella di routing del nodo A

host/rete	*diretto/indiretto*	*gateway*	*interfaccia driver*
223.1.2	diretto	—	eth0
223.1.3	indiretto	223.1.2.4	eth0
223.1.4	indiretto	223.1.2.4	eth0

Nel caso il modulo IP di A riceva dallo strato trasporto un pacchetto da inviare ad E, il cui indirizzo IP è `223.1.4.2`:

1. Il modulo IP ricerca l'indirizzo nella sua tabella di routing, e seleziona la riga corrispondente alla rete `223.1.4`.
2. Il modulo IP trova sulla riga selezionata l'indirizzo del gateway (`223.1.2.4`) ed il driver da utilizzare (`eth0`).
3. Il modulo IP consegna il pacchetto all'interfaccia del driver Ethernet `eth0`, e gli comunica che il pacchetto va recapitato all'indirizzo IP `223.1.2.4`.
4. Il driver Ethernet consulta la tabella ARP per ottenere l'indirizzo Ethernet del gateway `223.1.2.4`, ottenendo l'indirizzo Ethernet `08:00:10:99:AC:54`.
5. Infine il driver Ethernet di A costruisce il frame specificando come destinatario l'indirizzo Ethernet `08:00:10:99:AC:54` e lo consegna al transceiver. Il frame risultante avrà la forma illustrata nella tabella 4.10, che esplicita quello già illustrato in 4.6.

Il frame viene accettato dal driver Ethernet del modulo D, che riconosce come proprio l'indirizzo Ethernet, e da questo viene passato come *pacchetto entrante* al modulo IP. La tabella di routing di D è in tabella 4.11.

Le azioni intraprese dal nodo D sono dunque le seguenti:

1. Il driver Ethernet `eth0` riconosce il proprio indirizzo nel frame, riceve il pacchetto e lo passa al IP.

Tabella 4.10 Intestazione del frame prodotto da A per E con routing indiretto via D

	Mittente	*Destinatario*
Frame Ethernet	`08:00:39:00:2F:C3`	`08:00:10:99:AC:54`
Pacchetto IP	`223.1.2.1`	`223.1.4.2`

Tabella 4.11 Tabella di routing del nodo D

host/rete	*diretto/indiretto*	*gateway*	*interfaccia driver*
223.1.2	diretto	—	eth0
223.1.3	diretto	—	eth1
223.1.4	diretto	—	eth2

2. Il modulo IP di D tratta il pacchetto come *pacchetto entrante*: esamina il destinatario del pacchetto, e rileva che non coincide con il proprio. Il pacchetto quindi è da inoltrare nuovamente in rete, e viene trattato come *pacchetto uscente*.

3. Il modulo IP ricerca nella propria tabella di routing l'indirizzo del destinatario `223.1.4.2`, e trova la riga corrispondente alla rete `223.1.4`.

4. Il modulo IP trova nella riga selezionata l'indicazione che il routing è diretto, attraverso il driver `eth2`.

5. Il modulo IP consegna il pacchetto al driver Ethernet `eth2`, con l'indicazione del destinatario `223.1.4.2`.

6. Il driver Ethernet chiede ad ARP l'indirizzo Ethernet corrispondente a `223.1.4.2` ed ottiene `08:00:43:0F:12:73`.

7. Il driver Ethernet costruisce il frame che immette nella rete. Il frame avrà la forma illustrata nella tabella 4.12, che esplicita quello già illustrato in 4.7.

Infine il frame viene riconosciuto dal driver di E, e passato al modulo IP, che lo tratta come *pacchetto entrante*. Il modulo IP di E riconosce che l'indirizzo IP del destinatario corrisponde al proprio, estrae il messaggio contenuto nel pacchetto e lo consegna allo strato trasporto.

Tabella 4.12 Intestazione del frame prodotto da A per E con routing indiretto via D

	Mittente	*Destinatario*
Frame Ethernet	`08:00:10:99:AC:54`	`08:00:43:0F:12:73`
Pacchetto IP	`223.1.2.1`	`223.1.4.2`

Considerazioni conclusive sul routing

L'esempio considera un Internet di dimensioni limitate: nell'Internet globale un pacchetto può attraversare moltissimi gateway prima di giungere nella rete di destinazione.

Lo strato IP di ciascun gateway collabora nel viaggio il pacchetto, definendo soltanto la prossima destinazione. Quindi il routing complessivo risulta dalla composizione delle decisioni dei singoli gateway, senza l'intervento di una entità centralizzata.

Il routing in Internet è un esempio eccellente di algoritmo distribuito: non esiste un singolo punto di controllo (il cui guasto comporterebbe l'inutilizzabilità dell'intera rete), ma piuttosto il controllo è distribuito su un gran numero di nodi.

La struttura del routing in Internet non garantisce tuttavia alcuna ottimalità nelle scelte di routing: la cattiva configurazione delle tabelle di routing può portare a forti perdite di prestazioni, o a interruzioni difficili da diagnosticare.

Per questo esistono procedimenti per la compilazione automatica delle tabelle di routing, anche se per network di dimensioni contenute questo è alla portata dell'amministratore di rete.

Excursus: gli indirizzi IP mnemonici

Gli esseri umani preferiscono nomi significativi a sequenze di ottetti: quindi esiste l'esigenza di associare ai nodi, oltre all'indirizzo IP, anche una denominazione mnemonica.

Per la traduzione da nome a indirizzo o viceversa, si può utilizzare una tabella, memorizzata in un file nella memoria del nodo. Questa tabella viene mantenuta aggiornata manualmente, e dunque può risultare inconsistente in caso di modifiche alle denominazioni dei nodi.

Nei sistemi operativi della famiglia UNIX, questa tabella è registrata in un

file di configurazione spesso collocato in `/etc/hosts`, ed ha la forma indicata in tabella 4.13.

Tabella 4.13 Tabella di corrispondenza indirizzo IP ↔ nome

indirizzo	*nome*
223.1.2.1	A
223.1.2.2	B
223.1.2.3	C
223.1.2.4	D
223.1.3.2	I
223.1.4.2	E

L'indirizzo IP è nella prima colonna, il nome nella seconda.

Si può notare come il nodo D, il *gateway*, abbia in questa tabella una sola voce, anche se di fatto possederà tre indirizzi IP: essendo collegato a tre driver, ciascuna entrata al modulo dovrà essere caratterizzata da un indirizzo diverso. Ma le informazioni contenute nella tabella locale possono essere incomplete.

Esiste un file analogo per le reti: la corrispondenza qui sarà tra numeri di rete e nomi mnemonici per le reti (v. tabella 4.14). Nei sistemi operativi di tipo UNIX la tabella è normalmente contenuta in un file collocato in `/etc/networks`.

Tabella 4.14 Tabella di corrispondenza indirizzo IP ↔ nome

indirizzo	*nome*
223.1.2	laboratorio
223.1.3	magazzino
223.1.4	contabilita

È comunque necessario avere a disposizione la corrispondenza completa, cioè a copertura mondiale, tra nomi ed indirizzi IP; infatti questa informazione è necessaria a molte applicazioni. Ad esempio, i link nelle pagine web sono normalmente indicati da nomi, e non da indirizzi IP.

Quindi in ogni rete si predispone un nodo che mantenga una tabella di conversione molto estesa, e si chiede a lui il servizio di traduzione.

Anche in questo caso, come per le tabelle di ARP, la gestione di queste tabelle è critica, poiché l'indirizzo associato ad un nome può essere modificato

in qualsiasi momento e senza preavviso. Come per ARP, esiste un protocollo per il mantenimento delle tabelle di corrispondenza tra nomi ed indirizzi IP: il protocollo ed il servizio che offre sono detti *Domain Name Service* (DNS).

Nel caso del DNS la corrispondenza viene rappresentata su più tabelle, distribuite su molti nodi *server*. Per la rete dell'esempio, la corrispondenza potrà essere quella indicata in tabella 4.15. Da notare come ora siano esplicitati i tre indirizzi IP del nodo D, per il quale vengono altresì definiti due nomi, o alias: `D` e `labnetrouter`.

Tabella 4.15 Tabella di corrispondenza indirizzo IP ↔ nome

indirizzo	*nome*
`223.1.2.1`	A
`223.1.2.2`	B
`223.1.2.3`	C
`223.1.2.4`	labnetrouter D
`223.1.3.1`	magnetrouter
`223.1.3.3`	I
`223.1.4.1`	contnetrouter
`223.1.4.2`	E

4.1.5 Frammentazione di un pacchetto IP

La *frammentazione* di un pacchetto si rende necessaria quando le sue dimensioni siano superiori alla dimensione massima del frame del protocollo *data link* che deve trasferirlo: questo limite massimo è indicato come la *Maximum Transmission Unit* (MTU) del link. In genere si preferisce non incorrere nella frammentazione, ma esistono casi (che esamineremo nel paragrafo 4.3) in cui ciò si rende necessario per non evitare la perdita di un pacchetto.

Per quanto riguarda il *riassemblaggio* dei pacchetti frammentati si indica l'opportunità di riassemblare il pacchetto solo una volta che questo sia arrivato a destinazione (in [5], paragrafo 2.1). Esistono tuttavia delle eccezioni che non esaminiamo.

Per gestire il riassemblaggio di un frammento è necessario predisporre un buffer per la raccolta temporanea dei frammenti, in attesa che il pacchetto sia completo e venga trattato. La dimensione minima di questo buffer viene fissata a 576 ottetti (v. [10], paragrafo 3.3.2): quindi un pacchetto che abbia una dimensione di 576 ottetti è certamente in grado di essere recapitato,

eventualmente dopo essere stato riassemblato. Un pacchetto di dimensioni superiori potrebbe invece non essere recapitato.

Esiste inoltre un orientamento, consolidato nella prossima versione del protocollo IP, che porta ad escludere la frammentazione per pacchetti di dimensione di 576 ottetti, richiedendo che gli strati *data link* siano sempre in grado di offrire una MTU di almeno 576 ottetti: la frammentazione ed il riassemblaggio sotto questo limite verrebbero delegati, se necessari, allo strato *data link* (v. [8], capitolo 5). È l'orientamento seguito, ad esempio, dal protocollo ATM.

Queste considerazioni portano allora ad una raccomandazione per la **massima** dimensione di un pacchetto: se si vuole una comunicazione efficiente, è opportuno che ogni pacchetto IP abbia all'origine una dimensione massima di 576 ottetti. Come vedremo nel paragrafo 4.3, esistono protocolli per determinare l'opportunità di utilizzare dimensioni superiori per il pacchetto senza incorrere in frammentazione, migliorando le prestazioni.

La frammentazione avviene suddividendo il payload IP del pacchetto (entrante od uscente) in frammenti la cui lunghezza in ottetti sia multipla di 8, ed inserendo ciascun frammento in un nuovo pacchetto IP. Lo header dei nuovi pacchetti sarà identico allo header del pacchetto originario, salvo per i dati specifici del frammento.

Ad esempio (v. figura 4.5), un pacchetto di 1500 ottetti, suddivisi in uno header di 20 ottetti ed un payload di 1480 ottetti, dovendo transitare in un link la cui MTU è di 576 ottetti (corrispondente al requisito minimo in IP v6), verrà scomposto in due pacchetti di 576 ottetti, ciascuno con uno header di 20 ottetti ed un payload di 556 ottetti (multiplo di 8), ed un pacchetto di 388 ottetti, composto da uno header di 20 ottetti ed un payload di 368 ottetti.

Lo header del pacchetto IP che contiene un frammento dovrà contenere tutte le informazioni necessarie al riordinamento dei pacchetti ed al riassemblaggio del payload originario.

Queste sono le informazioni che compaiono nello *header* del pacchetto IP, e che sono riconducibili alle attività di frammentazione e riassemblaggio:

- Un **identificatore** che consente di distinguere i frammenti appartenenti ad un certo payload.

- Un **offset** che indica la posizione del frammento entro il pacchetto.

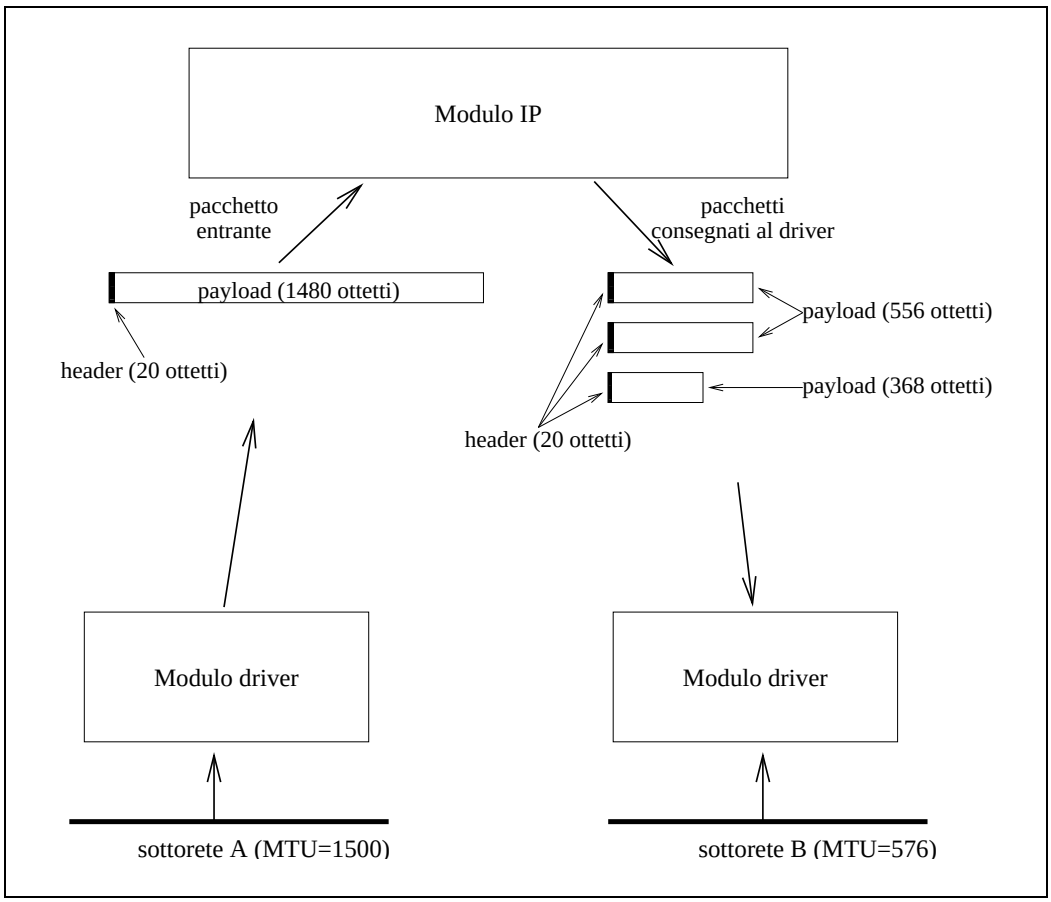


Figura 4.5 Un esempio di frammentazione

- Un **flag** che indica se il frammento è quello conclusivo del payload frammentato.

Riassumendo, tutti i frammenti, escluso l'ultimo, conterranno dati per un numero di ottetti che sia multiplo di 8 (64 bit). Il primo frammento avrà un offset `0`, che crescerà nei frammenti successivi corrispondendo alla lunghezza complessiva dei frammenti precedenti. Il flag sarà `1` in tutti i frammenti escluso l'ultimo, indicando la presenza di frammenti successivi.

Per evitare la frammentazione, lo strato IP fornisce allo strato trasporto il valore "consigliato" per la comunicazione con un certo destinatario. In questo modo la frammentazione può essere spostata, se opportuno, allo strato trasporto.

4.1.6 Il formato dei pacchetti IP

Entriamo ora nel dettaglio tecnico del protocollo. Per questo, come per altri protocolli che studieremo in seguito, l'analisi dello *header* dei messaggi prodotti dal protocollo si propone come una guida ideale, poiché la funzione dei diversi campi dello header è quella di supportare gli algoritmi realizzati dal protocollo.

Lo header di un pacchetto IP ([1]), dunque, deve contenere tutte le informazioni necessarie a determinare la gestione del pacchetto, tanto al nodo mittente ed al nodo destinatario quanto a tutti i nodi gateway intermedi, ed ha una lunghezza variabile da 20 a 60 ottetti. Il payload di un pacchetto di 576 ottetti è dunque compreso tra un minimo di 512 ed un massimo di 556 ottetti.

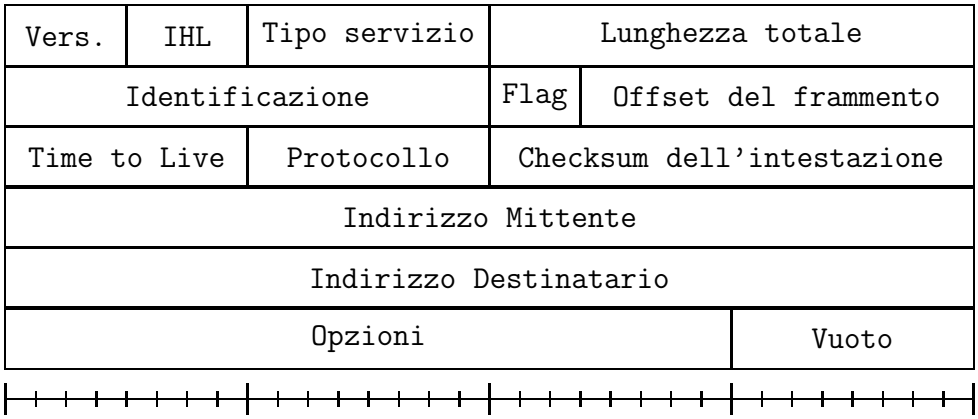


Figura 4.6 Intestazione di un pacchetto IP

In figura 4.6 è rappresentato lo *header* di un pacchetto IP. Vediamo la funzione dei diversi campi:

Vers. Indica la versione del modulo IP che ha prodotto il pacchetto: questo è necessario per fare in modo che chi riceve possa adattarsi alle caratteristiche di chi spedisce. La versione attuale è la 4, e deve scartare silenziosamente (cioè senza sollevare eccezioni) i pacchetti che non hanno questo numero di versione ([10], paragrafo 3.2.1.1).

IHL Indica la lunghezza (in parole di 32 bit, una riga nella figura 4.6, ovvero 4 ottetti). Può essere utilizzato per costruire un puntatore all'inizio della zona dati successiva allo header. La lunghezza minima dello header IP è

di 20 ottetti, che corrisponde ad un valore 5 nel campo *Internet Header Length* (IHL). Vedremo che i campi opzionali sono collocati oltre la 5a posizione (corrispondente al 20-esimo ottetto).

TOS Indica la "qualità del servizio" associata al pacchetto. Questo serve a fare in modo che sia possibile controllare alcuni parametri legati all'affidabilità, alla velocità di trasmissione, e al *throughput*. In genere si cerca un compromesso tra questi. Le tabelle 4.16 e 4.17 descrivono il contenuto del campo *Type of Service* (TOS).

Tabella 4.16 Il contenuto del campo TOS

Bit	*Significato*
`0--2`	Precedenza (v. tabella 4.17)
`3`	Ritardo (0 = Normal, 1 = Basso)
`4`	Throughput (0=Normale, 1 = Alto)
`5`	Affidabilità (0=normale, 1=Alta)
`6-7`	Riservati ad uso futuro

Tabella 4.17 Le priorità nel campo TOS

Valore	*Significato*
`111`	Network Control
`110`	Internetwork Control
`101`	CRITIC/ECP
`100`	Flash Override
`011`	Flash
`010`	Immediate
`001`	Priority
`000`	Routine

Generalmente le tre caratteristiche (ritardo, throughput ed affidabilità) non sono compatibili, e ciascuna va a scapito delle altre. È poco saggio chiederne più di due ...

Il tipo del servizio rimane associato al pacchetto attraverso tutto il percorso attraverso Internet.

L'amministrazione di ogni singola rete è responsabile della gestione delle indicazioni di "tipo del servizio" e priorità. Come regola generale, la priorità

Network Control dovrebbe essere limitata solo a pacchetti interni ad una singola rete.

Mentre il sotto-campo "Precedenza" ha un interesse confinato all'ambito militare, la parte restante potrebbe entrare nell'uso ([10]).

Lunghezza Totale È la lunghezza complessiva del pacchetto, header e dati, misurata in ottetti. Poiché il campo è lungo 16 bit, la massima lunghezza ammessa è di circa 65 Kbyte: una lunghezza che sembra generalmente eccessiva.

Identificatore È l'identificatore del pacchetto, che serve a riassemblarne i frammenti a destinazione. Viene generato dal modulo IP mittente, e conservato durante tutto il trasferimento in Internet. Insieme all'indirizzo del mittente ed a quello del destinatario fornisce un identificatore unico del pacchetto.

Flag Controllano alcuni dettagli della gestione della frammentazione (v. tabella 4.18).

Tabella 4.18 Il contenuto del campo "Flag"

Bit		*Significato*
0	-	Riservato per uso futuro
1	DF	Frammentabile (0 = No, 1 = Sì)
2	MF	Ultimo (0 = Sì, 1 = No)

Se il pacchetto è indicato dal flag *Don't fragment* (DF) come *non frammentabile*, verrà silenziosamente scartato qualora si renda necessaria la sua frammentazione.

Se il pacchetto è indicato dal flag *More Fragments* (MF) come *ultimo*, si intende che sia l'ultimo frammento di un pacchetto frammentato.

Offset del Frammento Indica la collocazione del frammento all'interno del pacchetto, ed è misurata in unità di 8 ottetti.

Time to Live Indica dopo quanto tempo il pacchetto deve essere rimosso dalla rete, ed è espresso in secondi.

In realtà, la regola applicata non rispetta (né potrebbe) l'indicazione in secondi. Ciò che accade è che ogni gateway decrementa il *Time To Live*

(TTL) di una unità, ed il pacchetto deve essere rimosso quando il campo raggiunge il valore 0. In questo modo si evita che pacchetti "vaghino" indefinitamente per la rete.

Protocollo Si tratta del codice corrispondente al protocollo dello strato superiore cui è diretta l'informazione. I codici dei protocolli sono riportati in [20].

Checksum dell'intestazione È un numero che viene utilizzato per verificare l'integrità dell'intestazione. Viene calcolato come segue (v. [1], cap. 3.1):

> Il checksum è il complemento a 1 della somma (in complemento a 1) di tutte le parole costituenti l'intestazione. Le parole sono considerate di 16 bit, e non si considera la parola destinata a contenere il checksum (ovvero la si considera a zero).

Questa semplice checksum è stata successivamente sostituita da un codice CRC che offre una migliore rilevazione degli errori.

Indirizzo del mittente È l'indirizzo IP del mittente, normalmente rappresentato in base 256 (ad es. `132.45.212.10`): per gli indirizzi IP, v. paragrafo 4.1.1.

Indirizzo del destinatario È l'indirizzo IP del destinatario: per gli indirizzi IP, v. paragrafo 4.1.1.

Opzioni Il campo destinato alle opzioni ha dimensioni variabili: ogni pacchetto può indicare un numero di opzioni variabili. Alcune opzioni vengono indicate con un campo di lunghezza fissa, altre hanno lunghezza variabile:

- Nel primo caso l'opzione occupa un ottetto, ed il *tipo dell'opzione* è codificato come indicato in tabella 4.19.
- Nel secondo caso, oltre all'ottetto che specifica il tipo dell'opzione, un secondo ottetto indica la lunghezza dell'opzione, a cui segue la descrizione (di lunghezza variabile) dell'opzione.

Il primo bit dell'ottetto indica se l'opzione va ricopiata in tutti i frammenti o meno.

Tabella 4.19 Il contenuto dell'ottetto *tipo dell'opzione*

Bit	*Significato*
`0`	Copiare (0 = No, 1 = Sì)
`1-2`	Classe
`3...7`	Numero

Tabella 4.20 Le *classi* ed i *numeri* definiti per il *tipo dell'opzione*

Classe	*Numero*	*Lunghezza*	*Descrizione*
`0`	`0`	—	fine opzioni
`0`	`1`	—	nulla
`0`	`2`	11	sicurezza (DoD)
`0`	`3`	variabile	indicazioni per il routing
`0`	`7`	variabile	registrazione del routing
`0`	`8`	4	stream ID: obsoleto ([10])
`0`	`9`	variabile	prescrizioni per il routing
`2`	`4`	variabile	timestamp

Delle molte classi e numeri disponibili molti sono lasciati inutilizzati. In tabella 4.20 sono riassunte quelle specificate nello standard ([1]).

L'opzione *fine opzioni* indica la fine delle opzioni, quando questa non sia consistente con il campo che indica la lunghezza dello header. L'opzione *opzione nulla* serve ad allineare le opzioni, quando necessario. L'opzione *sicurezza* non rientra nello standard IP, ed è gestita dalla singola rete. L'opzione *stream ID* è obsoleta, e viene ignorata in ricezione e mai inviata. Gli altri campi, *registrazioni*, *indicazioni* e *prescrizioni* di routing e *timestamp* sono utilizzati per la gestione del routing, e necessitano una descrizione più approfondita.

Le opzioni di controllo nello header del pacchetto IP

Le opzioni di controllo sono caratterizzate da una lunghezza variabile, e da un trattamento che può interessare anche tutti i gateway che vengono attraversati dal pacchetto.

Un primo gruppo è adatto a controllare il percorso seguito da un pacchetto, sia registrando quello che viene effettivamente seguito che condizionandolo.

L'interesse per la possibilità di controllare il percorso al mittente (o *source routing*) nasce dalla necessità, ad esempio, di far transitare determinati pacchetti in nodi che siano in grado di trattarli in modo appropriato.

L'opzione di source routing si è tuttavia rivelata inefficace per soddisfare questa necessità, poiché può introdurre problemi di sicurezza. Ad esempio, un cattivo uso delle opzioni di *source routing* può indurre il sovraccarico di un link o di un nodo (ma vedi anche [10], appendice "Security Considerations"). Quindi queste opzioni sono spesso ignorate dai moduli IP (più precisamente, il modulo IP scarta silenziosamente tali pacchetti). Vedremo nel paragrafo 4.4.1 una tecnica che ha sostituito l'uso delle opzioni di *source routing* nella versione corrente di IP, mentre nella prossima versione di IP, la numero 6, dovrebbero essere reintrodotte, migliorate. Il formato delle opzioni di *source routing* è quello rappresentato in figura 4.7.

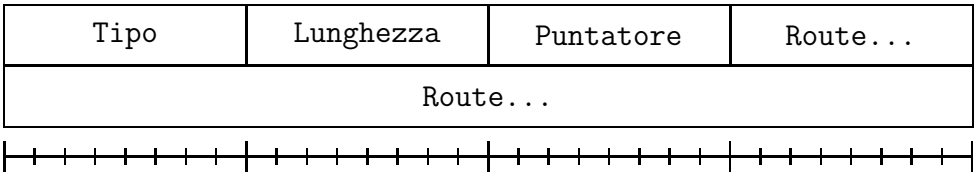


Figura 4.7 Formato di una opzione di controllo del routing

Esistono tre tipi di opzioni per il controllo del routing:

Routing indicativo e registrazione – L'opzione è indicata con il tipo `10000011`: viene indicata una route per raggiungere la destinazione. Il percorso consiste in una sequenza di numeri di IP, uno per ogni gateway da attraversare.

Ad ogni nuovo inoltro, il gateway indicato dal puntatore viene sostituito con l'indirizzo locale, e quello successivo viene inteso come suggerimento per il prossimo inoltro. Il suggerimento non verrà necessariamente preso in considerazione.

Arrivati in fondo alla lista l'inoltro non terrà più conto della lista di gateway, e considererà solo la destinazione finale.

In questo modo il pacchetto, all'arrivo, conterrà la traccia di tutti i gateway visitati.

Routing restrittivo e registrazione – L'opzione è indicata con il tipo `10001001`: come il precedente, ma il routing deve necessariamente segui-

re le indicazioni date nella parte variabile dell'opzione, pena il fallimento del routing.

Registrazione del routing – L'opzione è indicata con il tipo `00000111`: prende dai precedenti solo la registrazione del percorso, ma non è presente alcuna indicazione di quale percorso seguire.

Lo spazio destinato al percorso è riempito sino a che il puntatore non supera la lunghezza.

Esiste inoltre una opzione che consente il controllo della temporizzazione. Il pacchetto ha il formato indicato in figura 4.8

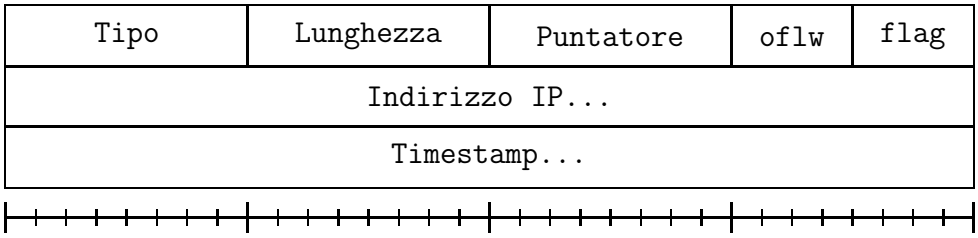


Figura 4.8 Formato di una opzione di controllo della temporizzazione

Questa opzione è caratterizzata dal tipo `01000100`, ed è destinata a contenere, per ciascun gateway attraversato, l'ora di attraversamento del gateway.

Il campo *lunghezza* indica lo spazio disponibile per queste registrazioni, ed il *puntatore* indica la prossima posizione libera. Il campo *oflw* viene incrementato quando un gateway non ha più spazio per riportare i propri dati.

Il campo flag può controllare il tipo di registrazione effettuata:

`0` vengono registrati i soli timestamp (4 ottetti ciascuno, consecutivamente)

`1` oltre ai timestamp vengono registrati anche gli indirizzi IP

`3` gli indirizzi IP sono indicati dal mittente: ogni gateway inserisce il proprio timestamp solo se il proprio IP coincide con il prossimo nell'opzione

4.1.7 Interfaccia con lo strato di trasporto

Per rappresentare l'interfaccia tra il modulo IP ed i moduli dello strato di trasporto, possiamo definire delle funzioni concettuali che possono essere poi rappresentate nel linguaggio di programmazione prescelto.

Una delle funzioni da mettere a disposizione è quella di spedizione di un pacchetto.

Tra i parametri della funzione, uno indicherà un buffer nel quale verrà depositato il dato da incapsulare nel pacchetto. Lo standard richiede inoltre che alcuni campi dell'intestazione del pacchetto da spedire siano controllati dallo strato di trasporto. Tra gli altri, questo è vero per i campi TOS, TTL, per il *Protocollo* e per le *Opzioni*.

Il prototipo, alla maniera del linguaggio `C`, della funzione di spedizione potrebbe essere il seguente ([10]):

`SEND (src, dst, prot, TOS, TTL, BufPTR, len, Id, DF, opt)`
$\Rightarrow$ `result` dove:

`src` : indirizzo mittente
`dst` : indirizzo destinatario
`prot` : protocollo
`TOS` : tipo del servizio
`TTL` : time to live
`BufPTR`: puntatore al payload
`len` : lunghezza del buffer
`Id` : identificatore (opzionale)
`DF` : non frammentare
`opt` : opzioni
`result`: esito
`OK`: datagram spedito
`Error`: argomenti errati o errore nella rete locale

La funzione `SEND` restituisce il controllo non appena termina, indicando se la costruzione del messaggio ed il suo inoltro sulla rete locale abbia avuto successo o meno. Si tratta quindi di una operazione **non bloccante**.

Un'altra funzione fondamentale è quella di ricezione. Questa può innescare o meno una attesa: nella specifica dello standard si precisa che il

destinatario può restare in attesa del pacchetto. Quindi si tratta di una operazione che può essere **bloccante**.

Lo strato di trasporto deve ricevere, insieme al payload del pacchetto, anche le informazioni relative alle *opzioni* presenti nel messaggio.

Il prototipo della funzione di ricezione può essere il seguente:

`RECV (BufPTR, prot)` ⇒ `src, dst, TOS, len, opt, result`
dove:

`src` : indirizzo mittente
`dst` : indirizzo destinatario
`prot` : protocollo
`TOS` : tipo del servizio
`BufPTR`: puntatore payload
`len` : lunghezza del buffer
`opt` : opzioni
`result`: esito
`OK`: datagram ricevuto
`Error`: argomenti errati o errore nella rete locale

Lo strato di trasporto deve essere in grado di determinare la dimensione massima del pacchetto (in spedizione e ricezione) che può essere scambiato con un certo destinatario senza incorrere in frammentazione IP (v. RFC879 [18]). Il prototipo della funzione che restituisce questa informazione può essere il seguente:

`GET_MAXSIZE(loc, rem, TOS)` ⇒ `MMS_R, MMS_S`
dove:

`loc` : indirizzo locale
`rem` : indirizzo remoto
`TOS` : tipo del servizio
`MMS_R` : massima dimensione del pacchetto in ricezione (Maximum Message Size – Receive)
`MMS_S` : massima dimensione del pacchetto in spedizione (Maximum Message Size – Send)

La funzione `GET_MAXSIZES` restituisce dunque due numeri: `MMS_S` rappresenta la lunghezza massima indicata per il payload di un pacchetto che non verrà frammentato nel percorso verso il nodo corrispondente all'indirizzo remoto, mentre `MMS_R` corrisponde alla lunghezza massima del payload di un pacchetto che può essere riassemblato dal nodo locale.

Infine, quando il nodo risieda su più reti (come nel caso di figura 1.8), è necessario che il modulo IP indichi allo strato di trasporto l'indirizzo IP da utilizzare per raggiungere la destinazione:

`GET_SRCADDR(rem, TOS)` $\Rightarrow$ `loc`
dove:

`rem` : indirizzo remoto
`TOS` : tipo del servizio
`loc` : indirizzo locale

4.2 ICMP – Internet Control Message Protocol

Il ruolo del protocollo ICMP è quello di verificare la funzionalità dei moduli IP. Questa verifica si esplica con due modalità:

- passiva – Consiste nel produrre e trasportare segnalazioni verso il mittente di un pacchetto che ha prodotto anomalie,
- attiva – Consiste nel trasportare pacchetti che generano sul ricevente un "eco" che viene rispedito al mittente.

Il ruolo di ICMP, dunque, *non è* quello di rendere in qualche modo più affidabile IP, ma piuttosto quello di notificare la presenza di problemi e di eseguire test mirati.

Il messaggio ICMP viene trasmesso come payload di un pacchetto IP. Lo header del pacchetto IP che contiene un messaggio ICMP presenta alcune particolarità: il campo *protocol* riporterà il numero `1`, ad indicare che il messaggio è indirizzato a ICMP, ed il campo *destinatario* viene impostato con il mittente del pacchetto anomalo. Il *tipo del servizio* per i pacchetti che contengono un messaggio ICMP dovrebbe essere 0 (normale).

Poiché ICMP stesso utilizza IP, si ammette che un messaggio di ICMP possa subire lo stesso genere di inconvenienti di cui soffre IP. Quindi è sempre possibile che, per esempio, un certo pacchetto generi una anomalia e che il mittente non riceva il messaggio di ICMP di segnalazione.

Per evitare loop infiniti, tuttavia, una anomalia generata da un messaggio di ICMP non dovrà mai dare luogo ad nuovo un messaggio ICMP. Inoltre, in caso di frammentazione, il messaggio ICMP verrà generato solo per il primo dei frammenti.

4.2.1 Funzionalità passiva di ICMP

In questa modalità la funzione di ICMP ([17]) è quella di accettare segnalazioni di eventi anomali da parte del modulo IP, e di generare messaggi ICMP di segnalazione diretti al mittente del pacchetto anomalo, o che ha determinato un'anomalia.

In [10] vengono indicati i casi in cui un evento anomalo deve essere o meno segnalato: questo in particolare si applica alla perdita o alla rimozione di messaggio. Nel caso in cui l'anomalia non debba essere notificata, la reazione viene normalmente indicata come *silent*, ad indicare che non dà luogo a segnalazione altro che nel modulo che rileva l'anomalia.

Si precisa inoltre che *non devono* dare luogo a messaggi ICMP i seguenti eventi:

- quelli concernenti un altro messaggio ICMP,
- quelli relativi a pacchetti inviati in broadcast o multicast,
- quelli relativi a frammenti di pacchetti che non siano il frammento iniziale.

Queste prescrizioni hanno l'effetto di evitare sovraccarichi della rete dovute a segnalazioni ridondanti o inopportune: ad esempio, le segnalazioni di errore provenienti dai destinatari di una comunicazione in broadcast, potenzialmente molto numerosi, potrebbero sovraccaricare il mittente, od i gateway prossimi a questo.

Negli altri casi può essere richiesta una segnalazione dell'anomalia, che comporta la spedizione di un messaggio di ICMP. In particolare sono previste le segnalazioni di:

Destinatario irraggiungibile – Nel caso in cui il gateway non sia in grado di indicare una via per raggiungere la rete destinataria, o indichi che non è raggiungibile, oppure nel caso che il nodo destinatario non supporti il protocollo indicato, oppure nel caso in cui il messaggio non sia frammentabile (v. pagina 4.1.6 la definizione del flag DF) ma debba essere frammentato per esigenze del gateway.

Tempo esaurito – Se il TTL del pacchetto è arrivato a zero, oppure se si esaurisce il tempo di attesa dei frammenti di un pacchetto (ed è stato ricevuto il primo!).

Parametri errati – Se il pacchetto deve essere scartato per una inconsistenza dei dati presenti nella sua intestazione (ad esempio nelle opzioni).

Rallentamento del mittente – Se il mittente genera troppi pacchetti rispetto alla banda disponibile: un gateway può generare questo messaggio se non ha più spazio di buffer disponibile, mentre il nodo destinatario può generarlo se l'applicazione non riesce ad elaborare i pacchetti. Ma il messaggio ICMP può anche essere spedito per prevenire questi inconvenienti.

Ridirezione – Viene spedito da un gateway al mittente del pacchetto, quando il gateway verso la rete di destinazione è raggiungibile direttamente dal mittente.

Vediamo nel dettaglio ciascuna di queste classi, ed il formato del messaggio che viene trasferito.

4.2.2 I formati dei messaggi di segnalazione

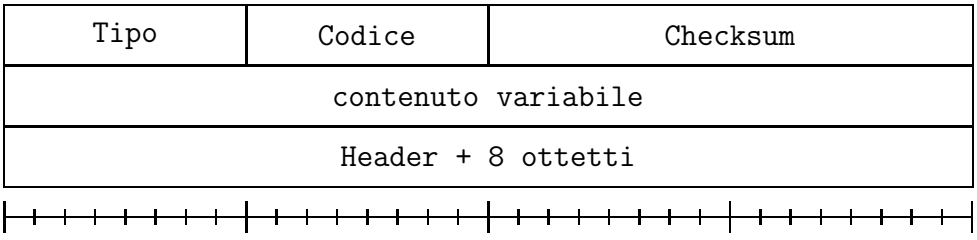


Figura 4.9 Formato di un messaggio ICMP di errore

Il messaggio ICMP di segnalazione ha il formato illustrato in figura 4.9. Contiene sempre lo header e 8 ottetti del pacchetto che ha causato l'anomalia:

questa informazione serve anche ad identificare il mittente a cui spedire il messaggio ICMP. Osserviamo inoltre che i primi ottetti del payload del messaggio che ha causato il problema conterranno, almeno in parte, lo header del messaggio allo strato trasporto. Questa informazione verrà utilizzata dal modulo ICMP ricevente per identificare il modulo dello strato trasporto cui inviare la segnalazione. Infatti, come vedremo, in genere si rende necessario informare lo strato trasporto circa le segnalazioni di ICMP.

Tuttavia poiché soli 8 ottetti possono essere insufficienti per identificare esattamente il mittente agli strati superiori, una recente estensione porta la lunghezza massima del pacchetto ICMP a 576 ottetti (v. [4] cap. 4.3.2.3).

Il genere di anomalia che viene segnalata è specificata da un *tipo* e da un *codice*, che occupano i primi due ottetti del messaggio. Vediamo per ciascun *tipo* le caratteristiche dell'anomalia che specifica, ed il modo in cui viene trattata dal destinatario.

Destinatario Irraggiungibile (tipo=3)

Tabella 4.21 I valori dei codici per il messaggio ICMP di destinazione irraggiungibile

codice	*segnalazione*
0	la rete non è raggiungibile
1	il nodo non è raggiungibile
2	protocollo non è raggiungibile
3	la porta non è raggiungibile
4	è necessaria la frammentazione di un pacchetto non frammentabile
5	la route specificata è interrotta
6	la rete non è nota ([10])
7	il nodo non è noto ([10])
8	il nodo mittente è isolato ([10])
9	l'accesso alla rete è riservato ([10])
10	l'accesso al nodo è riservato ([10])
11	la rete non offre il servizio richiesto ([10])
12	il nodo non offre il servizio richiesto ([10])

Il campo **codice** può avere uno dei valori elencati in tabella 4.21.

In particolare i codici `2` e `3` vengono generati rispettivamente quando il protocollo trasporto richiesto non è disponibile, e quando il protocollo trasporto non riesce a recapitare il messaggio, e non ha un meccanismo per informare il mittente.

Un messaggio di *destinatario irraggiungibile* viene sempre riportato allo strato trasporto del destinatario del messaggio ICMP, che lo gestisce nella maniera più opportuna. Questo accade anche se il protocollo trasporto possiede propri messaggi con lo stesso scopo.

I messaggi di tipo `0` (rete), `1` (nodo) e `5` (rotta) si intendono come indicazioni eventualmente transitorie: non vanno interpretate come segnale di guasto.

Il campo *variabile* è inutilizzato.

Tempo esaurito (tipo=11)

Tabella 4.22 I valori dei codici per il messaggio ICMP di tempo esaurito

codice	*segnalazione*
0	time to live scaduto
1	esaurimento del tempo limite per il riassemblaggio dei frammenti

Il campo **codice** può avere uno dei valori elencati nella tabella 4.22. Anche questo messaggio deve essere riportato allo strato trasporto del destinatario del messaggio ICMP.

Parametri Errati (tipo=12)

Il campo **codice** ha valore 0 (può avere valore 1 per usi militari [10]).

Il primo ottetto del campo variabile contiene un puntatore, che indica il numero d'ordine del campo dell'intestazione del pacchetto originario che ha causato il problema. Questo può essere anche tra le opzioni.

Il messaggio viene spedito solo se il pacchetto viene rimosso, e deve essere riportato allo strato trasporto del destinatario del messaggio ICMP.

Rallentamento del mittente (tipo=4)

Per questo messaggio il solo codice definito è `0`.

Questo messaggio può essere spedito anche prima che si verifichi la vera e propria rimozione o perdita di messaggi dovuta al sovraccarico generato dal mittente.

Il messaggio va riportato allo strato trasporto del destinatario del messaggio ICMP, il quale dovrebbe essere in grado di rallentare la trasmissione in modo da regolarizzare la comunicazione.

Inoltre lo strato trasporto deve essere in grado di mantenere sufficiente informazione per riprendere una interazione complessa anche dopo un messaggio di rallentamento.

Ridirezione (tipo=5)

Il campo **codice** può avere uno dei valori elencati in tabella 4.23. Il campo **variabile** contiene il gateway a cui ridirigere le comunicazioni.

Tabella 4.23 I valori dei codici per il messaggio ICMP di ridirezione

codice	*segnalazione*
0	ridirigi i pacchetti alla rete
1	ridirigi i pacchetti al nodo
2	ridirigi i pacchetti di un certo tipo di servizio al nodo
3	ridirigi i pacchetti di un certo tipo di servizio alla rete

Solo ai gateway è consentito trasmettere messaggi di questo tipo. Un host che riceve un messaggio di ridirezione può aggiornare le sue tabelle di routing, oppure scartare il messaggio se questo non è consistente (se il nuovo gateway non è sulla stessa rete su cui ha spedito il primo pacchetto, o se il mittente della ridirezione non è quello selezionato come prossimo per quella destinazione).

Ad esempio, una situazione in cui può essere spedito un messaggio di ridirezione è illustrata nella figura 4.10: il gateway `G1` riceve il pacchetto da `H1` per una certa destinazione `X`, e lo ridirige a `G2`. Il gateway `G1` può spedire un ICMP di ridirezione se si accorge che `H1` e `G2` sono sulla stessa rete.

4.2.3 La funzionalità attiva di ICMP

Il modulo ICMP di un nodo può produrre messaggi destinati a verificare la funzionalità del collegamento con un altro nodo. A questo scopo, il destinatario del messaggio di ICMP risponderà al mittente con un messaggio analogo. Quindi ogni formato del messaggio di andata è accompagnato anche dal formato dell'*eco*, il messaggio ICMP rispedito dal destinatario verso il mittente.

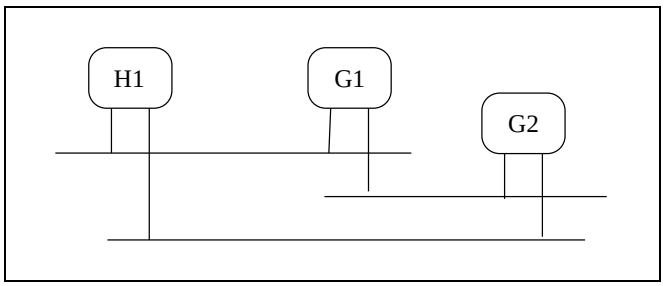


Figura 4.10 Un esempio di situazione in cui viene generato un messaggio ICMP di ridirezione

Lo standard prevede che un modulo ICMP sia in grado di trattare tre tipi di messaggi di verifica:

Richiesta di risposta – Si tratta semplicemente di un messaggio vuoto che verifica la sussistenza di un percorso per raggiungere la destinazione.

Sincronizzazione – È un messaggio che contiene informazione temporale.

Localizzazione – È un messaggio che viene utilizzato per scoprire il numero della rete su cui è collegato l'*host*.

4.2.4 I formati dei messaggi di verifica

Anche i messaggi ICMP di verifica sono incapsulati in un pacchetto IP: il mittente ed il destinatario corrispondono a mittente e destinatario reali, ed il tipo di protocollo indica ICMP(1).

I formati dei messaggi ICMP di verifica variano da un tipo all'altro. In comune con i messaggi di segnalazione ci sono i primi quattro ottetti: il *tipo*, il *codice* e la *checksum*.

Il messaggio di richiesta di risposta (tipo=8/0)

Il messaggio ICMP per la richiesta di risposta ha il formato illustrato in figura 4.11. Il **tipo** del messaggio è 8 per la richiesta, 0 per la risposta, ed il **codice** è 0.

I campi **identificatore** e **numero progressivo** possono servire per associare richieste e risposte.

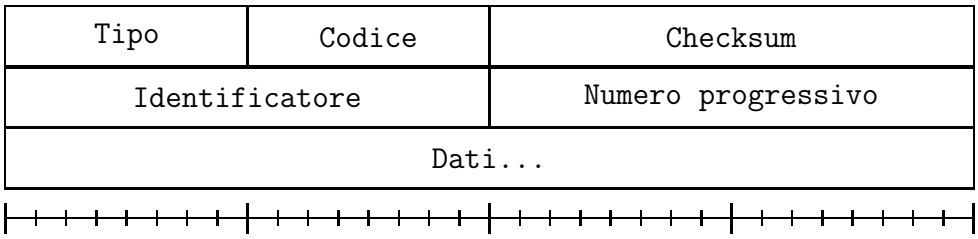


Figura 4.11 Formato di un messaggio ICMP di richiesta di risposta

Questo è il messaggio che viene generato dall'applicazione generalmente nota come "ping" nei sistemi operativi di tipo UNIX, molto utilizzata per un test minimale delle connessioni di rete.

Ogni host deve implementare la risposta ad una richiesta di un'eco di risposta, ma una risposta ad un indirizzo di broadcast o multicast può venire ignorata silenziosamente.

I dati devono essere passati dalla richiesta alla risposta intatti, salvo che il livello IP che spedisce la risposta non richieda la frammentazione della risposta. In questo caso il dati sono troncati al primo frammento.

Se il pacchetto IP richiede il tracciamento del percorso del pacchetto (opzione *record route* o *datagram*), questi dati vanno riportati nel messaggio di risposta, in modo che la traccia contenga tanto il percorso di andata, quanto quello di ritorno.

Se il pacchetto precisa una route specifica (opzione *source route*), lo stesso percorso va inserito, invertito, nel messaggio di risposta.

I formati dei messaggi di sincronizzazione (tipo=13/14)

Il messaggio di sincronizzazione è caratterizzato dal formato illustrato in figura 4.12. Il tipo è 13 per la **richiesta**, 14 per la **replica**, ed il **codice** è 0.

I tre campi identificati come "timestamp" rappresentano il tempo, in millisecondi, trascorso dalla mezzanotte in tempo universale (UTC).

Il messaggio di partenza viene composto impostando solo il timestamp `originario`. Il nodo che spedisce la risposta includerà il timestamp di *ricezione* appena riceve il messaggio, ed il timestamp di *spedizione* appena rispedisce la risposta.

La generazione di questo messaggio è facoltativa: inoltre è lasciato ampio margine per inserire tempi poco o per nulla significativi. Valgono requisiti analoghi a quelli specificati per la richiesta di eco per quanto riguarda la

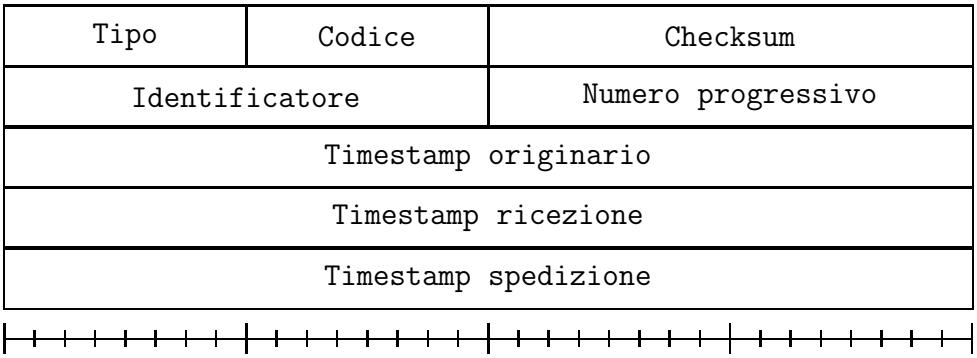


Figura 4.12 Formato di un messaggio ICMP di richiesta di sincronizzazione

possibilità di scartare quelli con un indirizzo di broadcast, il tracciamento del percorso di andata e ritorno per quelli con le opzioni di registrazione del percorso o dei timestamp, e la gestione del percorso per quelli con percorso specificato.

Per la sincronizzazione dei nodi è più indicato il protocollo *Network Time Protocol* (NTP) ([12]).

Messaggi di localizzazione (tipo=15/16)

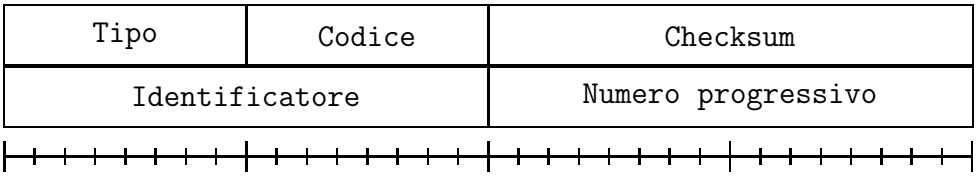


Figura 4.13 Formato di un messaggio ICMP di richiesta di localizzazione

Il messaggio di localizzazione è caratterizzato dal formato illustrato in figura 4.13. Il tipo è 15 per la **richiesta**, 16 per la **replica**, ed il **codice** è 0.

Questo messaggio può essere incapsulato in un pacchetto con indirizzo IP nullo di rete nullo (ad esempio `0.0.0.y` in una rete di tipo C), tanto per il mittente quanto per il destinatario. Il messaggio di ritorno riporterà l'indirizzo corretto dell'host.

Questo messaggio era utilizzato per il boot automatico di stazioni *diskless*.

Il comando è divenuto obsoleto con l'introduzione del protocolli *Reverse ARP* e *Dynamic Host Configuration Protocol* (DHCP).

4.2.5 Interfaccia al livello trasporto

Anche per ICMP è possibile descrivere una interfaccia a livello trasporto, come già per IP. L'interfaccia è rappresentata da due funzioni (v. [10], paragrafo 3.4): una di spedizione e l'altra di ricezione di un messaggio ICMP.

`SEND_ICMP (src, dst, prot, TOS, TTL, BufPTR, len, Id, DF, opt)`
`⇒ result`
dove:

`src` : indirizzo mittente
`dst` : indirizzo destinatario
`prot` : protocollo
`TOS` : tipo del servizio
`TTL` : time to live
`BufPTR`: puntatore al buffer contenente il messaggio
`len` : lunghezza del buffer
`Id` : identificatore (opzionale)
`DF` : non frammentare
`opt` : opzioni
`result`: esito
`OK`: datagram spedito
`Error`: argomenti errati o errore nella rete locale

Alcuni dei protocolli a livello trasporto infatti devono essere in grado di spedire alcuni messaggi ICMP che li riguardano esplicitamente: ad esempio, messaggi di errore riguardanti il numero di porta utilizzato.

La funzione, nella sua realizzazione pratica, può essere incorporata nella `SEND`.

`RECV_ICMP (BufPTR) ⇒ src, dst, len, result`
dove:

`src` : indirizzo mittente
`dst` : indirizzo destinatario

`BufPTR`: puntatore al buffer
`len` : lunghezza del buffer
`result`: esito
`OK`: datagram ricevuto
`Error`: argomenti errati o errore nella rete locale

Anche in questo caso, il protocollo al livello trasporto deve essere in grado di gestire opportunamente alcuni messaggi di ICMP che lo riguardano.

Come nel caso precedente, la funzione di ricezione del messaggio di ICMP può essere un caso particolare della ricezione di un pacchetto IP.

4.3 Determinazione della PMTU

La determinazione della corretta lunghezza di un pacchetto IP è di massima importanza per l'uso ottimale della rete. Un criterio di ottimalità appropriato consiste nel produrre pacchetti la cui lunghezza sia pari alla minima MTU tra quelle delle reti che dovranno essere attraversate dal pacchetto per arrivare a destinazione. Infatti, utilizzando una lunghezza inferiore a questa il traffico di rete aumenterebbe in quanto circolerebbero un numero di pacchetti superiore al necessario, mentre utilizzandone una superiore la frammentazione (necessaria per far transitare i pacchetti nelle reti che non possono trasmettere il pacchetto intero) aumenterebbe considerevolmente il carico dei router.

Quindi sarebbe opportuno trasmettere sempre pacchetti pari a questa lunghezza, che viene indicata come PMTU. Da qui nasce l'interesse per la determinazione della PMTU: questo dato, determinato dallo strato IP, potrebbe essere comunicato allo strato trasporto tramite la funzione `GET MAXSIZES` vista nell'interfaccia di IP; il parametro che contiene questa indicazione è quello indicato con `MMS_S` (Maximum Message Size – Send). Lo strato trasporto sarebbe quindi in grado di produrre payload che, una volta incapsulati, producano pacchetti di lunghezza pari alla PMTU.

Se il pacchetto è destinato a nodi residenti sulla stessa rete del mittente il problema è risolto semplicemente specificando nella tabella di routing la MTU per il supporto di comunicazione noto. Quando invece la destinazione è lontana, la determinazione della PMTU è problematica: ricordiamo infatti che le informazioni di routing sono distribuite, e che il mittente è in grado

di prevedere solo una piccola parte del cammino verso il destinatario. È in questo caso che si parla propriamente di PMTU.

Il modo tradizionalmente utilizzato per gestire questo problema consiste nell'applicare una semplicissima euristica: la PMTU è determinata come il valore minore tra la MTU della prima sottorete del path, ed il valore 576, che l'RFC1122 [10] indica come minima lunghezza per cui sia possibile la deframmentazione (e che condiziona la dimensione minima del buffer di ricezione, come visto a pag. 85). In questo modo si ottiene una valutazione della PMTU estremamente prudente, in quanto non supera la lunghezza del massimo pacchetto per cui è garantita la ricostruzione, ed evita la frammentazione almeno nella prima rete attraversata.

Questo modo è riconosciuto come estremamente inefficiente: infatti non esclude la frammentazione (la minima MTU è 68, e molti supporti utilizzano una MTU inferiore a 576), mentre non permette di approfittare di supporti con una MTU superiore a 576 (ad esempio, Ethernet supporta una MTU di 1500).

L'RFC1191 [13] propone un protocollo che può servire per risolvere questo problema, con un costo computazionale accettabile.

In sostanza si tratta di scegliere inizialmente per la MTU un valore approssimato, di solito quello della rete su cui viene inoltrato il pacchetto, e di inoltrare il messaggio con il *flag DF* (v. pagina 90) settato. Se viene ricevuta una ICMP di tipo 3 (*destination unreachable*) e con codice 4 (*fragmentation needed and DF set*), il mittente potrà provvedere a ridurre la PMTU per quel destinatario. Altrimenti potrà provare ad aumentarla.

4.3.1 Proposte di modifica del protocollo ICMP

L'RFC1191 [13] propone anche una modifica del protocollo ICMP originario, per consentire di diminuire il numero di messaggi necessari a convergere alla PMTU ottimale: si tratterebbe di inserire nel pacchetto ICMP di errore "*fragmentation needed and DF set*" la MTU della rete in cui si è verificato il problema. In questo modo, ogni ICMP contribuirebbe a determinare la MTU di una "strettoia" nel path.

Questa informazione verrebbe inserita nei due ottetti 7 ed 8, che nel protocollo originario sono inutilizzati, come si vede in figura 4.14.

Naturalmente, essendo la proposta una modifica di uno standard preesistente e diffusissimo, i proponenti indicano anche un modo per gestire messaggi ICMP che non contengano l'indicazione della MTU.

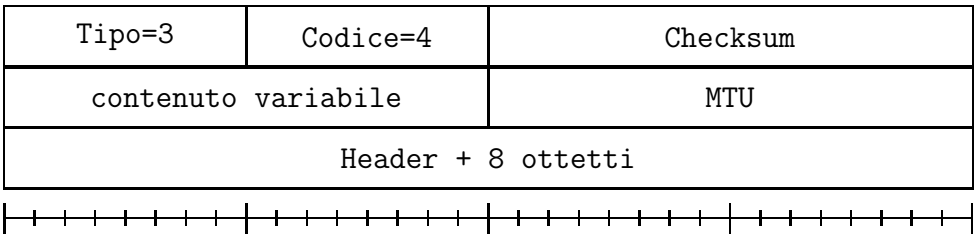


Figura 4.14 Formato di un messaggio ICMP che supporta *MTU discovery*

4.3.2 Trattamento dei messaggi ICMP da parte di router che non supportano MTU discovery

L'RFC indicato esplora la possibilità di acquisire ugualmente la migliore PMTU, anche nel caso in cui il router che invia il pacchetto ICMP non fornisca l'informazione sulla MTU del suo ramo. Vengono proposte e criticate alcune strategie.

Due semplici strategie di ricerca vengono sconsigliate: la prima decrementa la MTU del 25% ad ogni insuccesso, mentre la seconda effettua una ricerca binaria. Le ragioni che ne sconsigliano l'uso sono che la prima potrà sottostimare facilmente la MTU corretta, mentre la seconda si basa sull'effettivo raggiungimento della destinazione da parte di un certo messaggio. Come sappiamo, l'informazione fornita da ICMP a questo proposito è inattendibile, e in considerazione di ciò l'algoritmo di ricerca può essere estremamente complicato.

Quindi viene proposta una alternativa che sfrutta alcune MTU "note", ed un tentativo periodico di aumento della MTU. In tabella 4.24 sono raccolte le MTU caratteristiche di alcune tecnologie o protocolli di rete. Le informazioni circa la minima e massima dimensione di un pacchetto IP sono nell'RFC791 [1], paragrafo 3.2.

Poiché il procedimento è iterativo, sarà necessario che esista una memoria del valore di MTU tentato, per procedere con quello più piccolo. Questa memoria esiste implicitamente nel messaggio ICMP restituito: infatti questo conterrà lo header del pacchetto troppo lungo, completo del campo descrittore della lunghezza del pacchetto (ottetti 3 e 4 nello header IP).

Tabella 4.24 Le MTU di alcuni protocolli di rete molto diffusi

massimo	65535
prudenziale	32000
Token ring IBM	17914
IEEE 802.4	8166
IEEE 802.5 (token ring)	4464
FDDI	4352
IEEE 802.5	2002
IEEE 802.3 (Ethernet) - PtP	1500
SLIP	1006
X.25	508
PtP	296
minimo	68

4.3.3 Trattamento della informazione sulla PMTU

Le informazioni riguardanti la PMTU dovrebbero consistere in una tabella che associ una PMTU ad una certa destinazione e ad un certa qualità del servizio (la PMTU può variare a seconda della qualità di servizio richiesta).

La prima richiesta di spedizione di un pacchetto ad un host viene gestita utilizzando la tabella di routing, l'MTU calcolato con una regola di default, e abilitando il flag DF.

Quando viene ricevuto un ICMP "*fragmentation needed and DF set*", può essere creata una voce in una tabella che associa una PMTU specifica all'host mittente del messaggio ICMP. Questa tabella servirà a registrare la PMTU più opportuna per quell'host, e verrà modificata dal procedimento di PMTU discovery illustrato precedentemente.

Il livello trasporto sarà informato, anche in maniera asincrona, del nuovo valore di MTU opportuno per quella destinazione: ricordiamo che il dimensionamento dei pacchetti avviene sempre al livello trasporto. La notifica asincrona può essere evitata effettuando la comunicazione della nuova PMTU alla prossima richiesta di spedizione.

Dovrà pure esistere un meccanismo per verificare periodicamente le PMTU, visto che la grande dinamicità di Internet potrebbe consentire di aumentare la PMTU per l'apertura di una nuova rete, o il potenziamento di una esistente.

L'operazione di invalidazione non deve essere tuttavia troppo frequente, per non causare un eccessivo carico in rete. L'intero procedimento potrebbe

essere ripetuto con un periodo di 10 minuti circa (indicati nell'RFC1191), comunicando al livello superiore che la PMTU è ripristinata al valore della MTU del primo *hop*. Naturalmente, ogni installazione dovrebbe consentire di disabilitare questo periodo (fissandolo ad infinito).

4.4 Incapsulamento di un pacchetto IP dentro un pacchetto IP

Durante l'ultimo decennio sono emerse alcune applicazioni per le quali i soli meccanismi di routing offerti da IP non erano adatti. In particolare, il meccanismo di *source routing* (cioè di indicazione o prescrizione del percorso visti nel paragrafo 4.1.6) predisposto tra le opzioni del pacchetto IP si è rivelato poco utilizzabile. Infatti (v. RFC2003[14]):

- L'esigenza di una forzatura della rotta si può presentare in un nodo intermedio, e non in quello mittente.
- Il *source routing* presenta problemi di sicurezza.
- La presenza di opzioni (compresa quella di *source routing*) presenta problemi di efficienza dei router, dovuta alla dimensione variabile dello header ed alla sua gestione.
- Molti nodi gestiscono male le opzioni.
- La presenza di un firewall può arrestare pacchetti *source routed*.
- Se sul pacchetto è necessaria una forma di autenticazione, l'inserimento di opzioni può renderla complessa e poco sicura.
- Sarebbe possibile ovviare a molti di questi difetti modificando il pacchetto in transito, ma ciò è considerato inopportuno.

D'altra parte, la necessità di forzare il routing si presenta in diverse circostanze. Ad esempio:

- In una rete composta da utenti mobili (cioè che cambiano la loro collocazione nella rete) può risultare necessario che il gateway locale forzi il passaggio attraverso un altro gateway noto prima di raggiungere la destinazione. Il gateway noto potrebbe infatti essere delegato a conoscere la collocazione degli utenti mobili.

- Se il servizio utilizza protocolli non necessariamente implementati su tutti i router (ad esempio, il multicast utilizza il protocollo IGMP, non incluso tra quelli necessariamente presenti su un router), risulta necessario forzare il passaggio attraverso router che implementano quel protocollo.

- Per ragioni di sicurezza, può essere opportuno far transitare informazioni sensibili in siti e su reti affidabili.

In casi come questi si usa la tecnica del *tunnelling*, che consiste nell'incapsulare entro un altro pacchetto IP il pacchetto da istradare verso uno specifico router intermedio (v. figura 4.15). Lo *header* del pacchetto più esterno conterrà come destinazione del pacchetto il destinatario intermedio, cioè il punto di passaggio che si desidera forzare.

Ogni nodo intermedio su cui si vuole forzare il routing comporterà un allungamento del pacchetto pari alla lunghezza del nuovo header, ma si presume che in genere siano necessari pochi (o solo un) tunnel per raggiungere la destinazione finale. Tuttavia, i soli 20 ottetti necessari a incapsulare un pacchetto possono portare notevoli problemi, nel caso in cui un meccanismo di PMTU discovery abbia determinato una PMTU senza tener conto della presenza di un tunnel!

Vediamo i dettagli di questa tecnica.

4.4.1 Il tunnelling

Il caso generale di tunnelling può essere rappresentato dalla figura 4.16. Generalmente ciascun rettangolo rappresenta un nodo della rete differente, ma può darsi il caso che il destinatario abbia insieme il ruolo di decapsulatore, o che il mittente abbia pure il ruolo di incapsulatore. Il nodo indicato come *incapsulatore* corrisponde all'entrata nel tunnel, mentre quello etichettato con *decapsulatore* corrisponde all'uscita dal tunnel.

L'intestazione del pacchetto originario, precedente l'incapsulamento, è conforme ai requisiti di un pacchetto IP: quindi vi sono riportati l'indirizzo del nodo *mittente* iniziale e del nodo *destinatario* finale. Tutti gli altri elementi di questo header sono conformi a quanto visto nel paragrafo 4.1.6.

Lo header esterno viene inserito dal nodo incapsulatore. Anche questo header viene composto come una normale intestazione IP, ma è interessante esaminarne alcune caratteristiche:

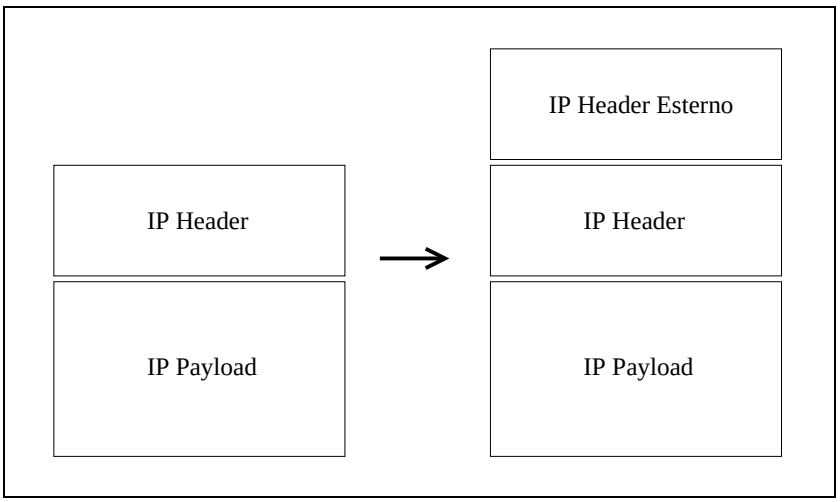


Figura 4.15 Incapsulamento di un pacchetto IP entro IP

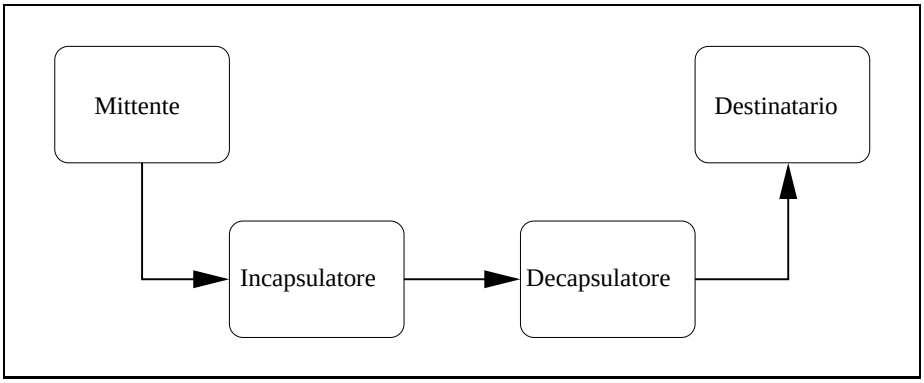


Figura 4.16 Un caso generico di *tunnelling*

- Il campo *Internet Header Length* corrisponde alla lunghezza dello header esterno.
- Il campo *Type of Service* viene ricopiato dall'analogo campo dello header interno.
- Il campo *Lunghezza Totale* corrisponde alla lunghezza di tutto il pacchetto, e quindi comprende la lunghezza dello header esterno, di quello interno, e del payload originario.
- L'*identificatore* e l'*offset* del frammento sono generati ex-novo: pure se il

pacchetto originario era un pacchetto non frammentato, è possibile una nuova frammentazione, per cui è necessario un identificatore ed un offset del frammento.

- Il valore del flag *Don't Fragment* viene ripreso dallo header interno.
- Il valore del TTL viene posto ad un valore appropriato per il percorso dall'incapsulatore al decapsulatore.
- Il codice del *Protocollo* di *incapsulamento* è 4.
- La *checksum* corrisponde a quella dello header esterno.
- L'indirizzo *mittente* e *destinatario* corrispondono agli indirizzi dell'incapsulatore e del decapsulatore, rispettivamente.
- Le *opzioni* non corrispondono in genere a quelle del pacchetto interno, ma sono adatte al percorso dall'incapsulatore al decapsulatore.

Nei confronti del pacchetto che viene incapsulato, il nodo incapsulatore si comporta come un nodo intermedio nella *route* che collega il mittente al destinatario. In particolare, decrementerà di una unità il TTL del pacchetto, e lo trasferirà, incapsulandolo, solo se questa quantità resta superiore a 0.

Un problema che viene amplificato dalla presenza di un tunnel è il rischio di un loop: potrebbe accadere che un errore di routing riporti il pacchetto già incapsulato al nodo incapsulatore, o che un pacchetto venga restituito al mittente dopo un incapsulamento. In questi casi il controllo sul *Time to Live* non sarà infatti efficace, visto che si produrrebbe un ripetuto incapsulamento, rinnovando ogni volta il *Time to Live*.

Queste sono le due prescrizioni che lo standard [14] applica a questo caso, descritte in figura 4.17:

- Il nodo che dovrebbe incapsulare un pacchetto controllerà se l'indirizzo del mittente del pacchetto corrisponde al proprio indirizzo, e in questo caso non incapsulerà il pacchetto, e preferibilmente lo scarterà. Infatti un loop nel routing potrebbe aver riportato il pacchetto al mittente, o all'ultimo incapsulatore, come nel caso del pacchetto A in figura 4.17.
- Il nodo che dovrebbe incapsulare un pacchetto controllerà che l'indirizzo del mittente non corrisponda a quello del decapsulatore a cui avrebbe

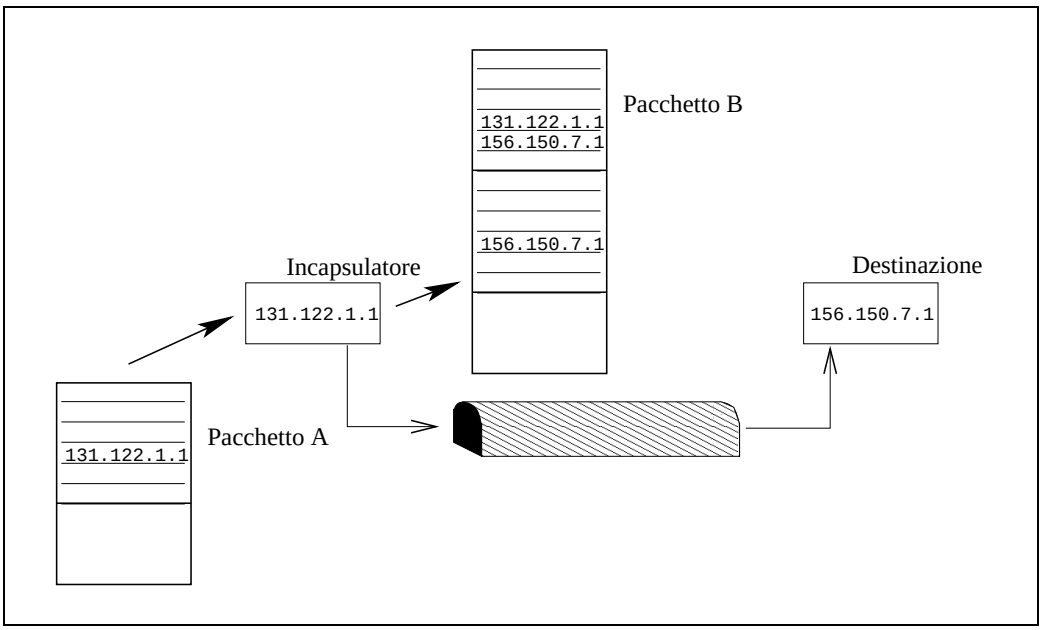


Figura 4.17 I due casi in cui l'incapsulatore scarta un pacchetto

mandato il pacchetto incapsulato, e in questo caso il pacchetto non sarà incapsulato, e sarà preferibilmente scartato. Infatti un errore nella tabella di routing del nodo potrebbe rimandare il pacchetto al mittente, mascherando lo header originario con quello di incapsulamento, come nel caso del pacchetto B in figura 4.17.

Messaggi ICMP dall'interno del tunnel

Il nodo incapsulatore può ricevere messaggi di ICMP da nodi incontrati nel percorso verso il decapsulatore. Questi messaggi subiscono un trattamento che dipende dal tipo del messaggio, in quanto sono relativi ad un pacchetto il cui header è generato dal nodo incapsulatore, e trasportano informazione proveniente dall'host interno al tunnel. Quindi il trattamento che devono subire è diverso da quello destinato ai messaggi ICMP ordinari: in primo luogo, è necessario distinguere se vadano "consumati" dall'incapsulatore, o dal mittente originario.

Ne vediamo alcuni casi interessanti (l'elenco completo è nell'RFC2003 [14]):

- I messaggi ICMP di tipo *Destination Unreachable* dovrebbero restituire l'astrazione più aderente alla realtà: quindi questi messaggi vengono riferiti dal nodo incapsulatore al mittente originario, con l'unica eccezione degli errori dovuti ad una prescrizione o indicazione di rotta inadatta.

- I messaggi di *Rallentamento del Mittente* non dovrebbero essere riferiti al mittente originario, ma trattati dall'incapsulatore.

- I messaggi di *Ridirezione* dovrebbero essere trattati dall'incapsulatore, senza riferirli al mittente originario.

- I messaggi di *Tempo Scaduto* rivelano l'impossibilità di raggiungere il nodo decapsulatore entro la scadenza. Quindi devono essere riferiti al nodo mittente.

- I messaggi di *Parametro Errato* possono essere riferiti al mittente, nel caso in cui il parametro sia tra quelli ripresi dal pacchetto incapsulato.

- Altri messaggi dovrebbero essere trattati come specificato nello standard ICMP@¡¿@¡¿@**!!** **(!!)**ICMPICMPICMP.

Un altro problema che riguarda i messaggi ICMP deriva dal fatto che tali pacchetti hanno una lunghezza predeterminata, e possono contenere poco più di uno header di lunghezza minima: quindi uno messaggio di ICMP proveniente dall'interno del tunnel conterrà certamente l'intestazione esterna, ma ben poco della intestazione del pacchetto incapsulato. Questo può condizionare la capacità del nodo incapsulante di restituire un messaggio ICMP completo al mittente originario.

Per risolvere questo problema, il nodo incapsulatore manterrà localmente alcune informazioni riguardanti lo stato del tunnel: tali informazioni vengono chiamate *soft state* del tunnel.

Dovrebbero fare parte del *soft state* del tunnel le seguenti informazioni:

- la MTU o MTU del tunnel,

- il TTL necessario a raggiungere il decapsulatore,

- la raggiungibiltà del decapsulatore.

Utilizzando queste informazioni, il nodo incapsulatore potrà generare lui stesso una parte dei messaggi di ICMP che sarebbero invece generati internamente al tunnel. In questo modo si risolve, almeno in parte, il problema della capienza del messaggio di ICMP. Contribuisce alla soluzione di questo genere di problemi anche il rilassamento delle restrizioni sulla lunghezza del pacchetto ICMP: abbiamo visto a pag. 100 che la lunghezza di tale pacchetto può arrivare sino a 576 ottetti.

Lo standard consiglia di incapsulare ed inviare ugualmente i messaggi che violano le restrizioni indicate dal *soft state* del tunnel, in modo da ottenere una immagine accurata della rete.

Vediamo ora alcune tecniche specifiche per la gestione del *soft state* di un tunnel, e brevemente il trattamento dei problemi di sicurezza introdotti dalla tecnica del tunnelling.

Determinazione della MTU di un tunnel

La determinazione dell'MTU del collegamento con un certo altro nodo è molto importante per evitare la frammentazione. Nel caso di un *tunnel*, fa parte dei compiti dell'incapsulatore determinare la MTU del tunnel, in modo da mantenere il *soft state* del tunnel. Per questa ragione l'incapsulatore abiliterà il flag DF dello header esterno, ciò che consentirà di determinare la dimensione dei pacchetti che possono passare senza essere frammentati.

Congestione del tunnel e rallentamento del mittente

Il nodo incapsulatore dovrebbe tenere conto di messaggi di rallentamento del mittente, aggiornando il *soft state* del tunnel con tali informazioni e prendendo adeguate contromisure. Tuttavia non dovrebbe riferire questo genere di messaggi al mittente originario.

Problemi di sicurezza

La tecnica dell'incapsulamento può comportare grossi problemi di sicurezza, e per questo i router che supportano l'incapsulamento ed il decapsulamento dovrebbero essere particolarmente preparati, per cercare il reale mittente e gli altri dati contenuti nello header incapsulato, piuttosto che fermarsi allo header esterno.

Questo è vero in particolare quando i router hanno funzione di filtraggio (è una delle tecniche per la realizzazione di firewall): infatti il filtraggio do-

vrebbe realizzarsi sui pacchetti incapsulati, piuttosto che sulle stesse capsule. Quindi i router di transito di pacchetti potenzialmente incapsulati dovrebbero essere preparati ad interpretare i pacchetti con indicazione di protocollo 4 nello header IP (v. figura 4.6).

D'altra parte, anche i decapsulatori dovrebbero essere in grado di autenticare (v. RFC1826 [3]) la provenienza dei pacchetti, in modo da non immettere nella rete di destinazione pacchetti indesiderati o provenienti da mittenti non affidabili.

Infine, i nodi in grado di decapsulare messaggi diretti a loro stessi dovrebbero ugualmente proporre alcune contromisure, e in particolare verificare l'autenticità e l'affidabilità dell'incapsulatore e del mittente originario.

È opportuno che questi controlli siano ridondanti, essendo realizzabili tanto sui router di separazione tra le sottoreti, quanto nei nodi incapsulatori e decapsulatori.

4.5 Il multicast in IP

L'idea del *multicast* nasce con l'esigenza di far arrivare a molti (ma non a tutti) determinati pacchetti: una teleconferenza è una delle situazioni in cui si presenta questa esigenza. In questa circostanza la stessa trasmissione deve essere ricevuta da più utenti: una analogia può essere quella di una trasmissione radiofonica, che può essere ascoltata da tutti quelli che si sintonizzano su una certa frequenza.

Il multicast comporta problemi che non possono essere risolti con le tecniche di unicast (cioè ci comunicazione da nodo a nodo) che abbiamo visto finora per IP. Infatti la realizzazione di collegamenti unicast con ciascuno dei partecipanti sarebbe proibitiva dal punto di vista del carico imposto alla rete di comunicazione: almeno nei router prossimi al mittente passerebbero tutti i datagrammi del servizio, e nel caso multimediale (come una conferenza) ciascuno di questi avrebbe già da sé un peso notevole.

D'altra parte, la modalità di broadcast generico supportata da IP può essere applicata in poche circostanze. Abbiamo visto a pagina 72 che gli indirizzi con la parte destinata all'host composta di soli `1` binari indicano tutti i nodi di una sottorete, e quindi vengono ricevuti da tutti questi: ad esempio `131.214.4.255` è l'indirizzo di *broadcast diretto* sulla rete `131.214.4`. Ma non sempre il multicast è indirizzato a *tutti* i nodi di una sottorete!

Quindi gli strumenti visti sino ad ora non sono in grado di realizzare il

multicast: servono degli strumenti diversi, anche se tutto deve in qualche modo restare contenuto entro i protocolli già esistenti.

4.5.1 Gli indirizzi multicast

L'idea di base è che, per realizzare il multicast, è necessario *identificare* una "sottorete" a parte: quella di tutti gli utenti che nell'analogia radiofonica sono sintonizzati su una certa frequenza, ovvero che vogliono ricevere una determinata trasmissione multicast.

L'iniziale ideazione degli indirizzi Internet ha lasciato liberi un certo numero di indirizzi IP, destinati ad un uso futuro. È una pratica prudente quella di progettare i protocolli lasciando spazio a futuri sviluppi.

Gli *indirizzi IP di classe D* sono tra quelli non assegnati, e contengono, nei primi 4 bit del primo ottetto, la combinazione `1110`. Quindi comprendono tutti gli indirizzi da 224.0.0.0 sino a 239.255.255.255, molti milioni di indirizzi. Ciascuno di questi indirizzi viene fatto corrispondere ad un *gruppo multicast*, a cui ciascun utente può aderire. Successivamente alla sua adesione, l'utente riceverà tanto i pacchetti indirizzati al suo indirizzo IP convenzionale, in classe A, B o C, quanto quelli indirizzati al gruppo multicast cui ha aderito.

Alcuni di questi gruppi hanno significati particolari.

224.0.0.1 È il gruppo che comprende tutti i nodi della sottorete locale (limitato a quelli che sanno gestire il multicast, come vedremo).

224.0.0.2 Come sopra, ma limitato ai soli gateway.

224.0.0.xxx Altri gruppi di router specifici per determinati servizi o attività.

239.0.0.xxx Altri gruppi di router specifici per l'amministrazione.

4.5.2 Livelli di conformità

Il routing, la ricezione e la spedizione di datagrammi in multicast sono funzionalità non specificate come necessarie nella attuale versione di IP (quella attuale è la versione 4, ma il multicast sarà specificato nella prossima, la 6). Quindi un nodo può essere conforme alle specifiche di Internet, ma non essere in grado di gestire il multicast. La conformità al multicast è una caratteristica opzionale di un nodo Internet. Per rendere flessibile la definizione della

conformità ai protocolli di multicast, si introducono dei *livelli di conformità* specifici, che indicano in che misura un certo nodo realizza la funzione di multicast.

I nodi al livello di conformità 0 non supportano in alcun modo il multicast: si limitano ad ignorare i pacchetti di classe D. Un nodo conforme alla specifica di Internet ha almeno conformità 0 rispetto al multicast: infatti la specifica di Internet richiede che il nodo scarti i pacchetti di cui non riconosce il nodo destinatario. Poiché gli indirizzi multicast sono confinati entro un intervallo in cui non esistono indirizzi convenzionali, un nodo che riconosca solo indirizzi convenzionali scarterà automaticamente i pacchetti multicast. La conformità di livello 0 propone quindi una caratterizzazione importante: ogni nodo Internet ricade almeno in questa classe di conformità. Quindi i protocolli di multicast, specificando il proprio comportamento verso i nodi di classe 0, sono in grado di funzionare correttamente in Internet.

I nodi al livello di conformità 1 possono spedire ma non ricevere datagrammi multicast. Si tratta di un gradino intermedio verso la piena operatività multicast, poiché è semplice modificare il modulo IP per poter spedire datagrammi multicast.

I nodi al livello di conformità 2 devono essere in grado di ricevere e spedire datagrammi multicast. L'attività di ricezione comporta la gestione dei gruppi, che è la parte più complessa, ed è regolata da un protocollo specifico (IGMP).

Vediamo ora nel dettaglio quali sono le modifiche da apportare al modulo IP per portarlo ai diversi livelli di conformità.

4.5.3 La spedizione dei datagrammi in multicast

Ci sono pochi dettagli da considerare quando si spedisce un datagramma IP in multicast:

- Il TTL deve essere adeguato alla "copertura" che si vuole fornire alla trasmissione. Normalmente si tratta di numeri più alti di quelli che si trovano nei normali datagrammi IP, e codificano il grado di copertura, piuttosto che specificare il numero di *hop* di routing: 32 o minore per coprire l'intero sito, 64 per coprire la regione, 128 per coprire l'intero continente, 255 per coprire il globo. Ma esistono altri strumenti più precisi per stabilire questi confini, in parte supportati dai gruppi di amministrazione (239.0.0.x).

- Il **loopback** (descritto a pagina 72) può servire per monitorare la trasmissione.

- La **selezione dell'interfaccia** di rete. Nei nodi con più di una interfaccia, dovrà essere selezionata una rete sulla quale sia disponibile un router a livello di conformità 2, che sia in grado di ricevere e trattare i pacchetti di multicast.

Il protocollo di *tunnelling*, illustrato nel paragrafo 4.4, può essere utilizzato da un mittente a livello di conformità 1 per raggiungere un router in grado di ricevere e gestire in maniera appropriata i pacchetti prodotti. Il mittente dovrà tuttavia essere autorizzato a raggiungere il router tramite tunnel, ed autenticare in maniera appropriata i propri pacchetti.

4.5.4 La ricezione dei datagrammi in multicast

Come anticipato, la ricezione dei datagrammi di multicast pone più problemi della spedizione.

Adesione ad un gruppo

L'operazione di *adesione ad un gruppo* ha effetti sia locali che remoti. Consideriamo ora solo quelli locali, mentre accenneremo a quelli remoti nel prossimo capitolo sul protocollo IGMP.

Il modulo IP, che prima doveva solo filtrare il datagrammi destinati al nodo stesso, dovrà ora selezionare anche i datagrammi relativi ai gruppi ai quali le applicazioni in esecuzione sul nodo hanno aderito. Il filtraggio sarà anche orientato ad una particolare interfaccia di rete, ove ve ne sia più di una.

Per essere in grado di gestire la spedizione di pacchetti in multicast, il nodo dovrà necessariamente aderire al gruppo `224.0.0.1`, che è quello attraverso il quale viene gestita, tramite il protocollo IGMP, l'adesione agli altri gruppi.

È opportuno sottolineare che l'adesione, come pure l'abbandono, di un gruppo sono eventi controllati da strati superiori ad IP, e che a tali livelli si riferiscono. Mentre la conoscenza degli indirizzi IP del nodo è di competenza esclusiva di IP, gli indirizzi dei gruppi multicast a cui il nodo aderisce sono dunque noti anche agli strati superiori.

Pre-filtraggio hardware

I frame Ethernet possono essere codificati in modo da contenere una indicazione del fatto che in essi è incapsulato un datagramma in multicast: se i primi tre ottetti dell'indirizzo Ethernet sono `01-00-5e`, il frame è interpretato come multicast, ed i restanti 3 ottetti sono interpretati come parte di un indirizzo IP del gruppo multicast a cui è destinata la trasmissione.

Questa forma di filtraggio è senz'altro più veloce del filtraggio software che può essere operato operato da IP, ma non altrettanto selettivo: infatti dei 28 bit destinati agli indirizzi di classe D, solo 23 (il primo bit è sempre 0) possono essere contenuti nell'indirizzo multicast Ethernet. Questi sono quelli meno significativi.

Quindi si ha un pre-filtraggio hardware eseguito sulla scheda di rete, ed uno successivo definitivo nel modulo IP.

4.5.5 IGMP – Internet Group Membership Protocol

L'effetto di una adesione ad un certo gruppo non può essere tuttavia limitato al solo nodo che supporta l'applicazione che aderisce al gruppo. Se questo fosse vero, tutto il traffico multicast mondiale dovrebbe passare attraverso tutti i router!

Quindi l'effetto di una adesione ad un gruppo viene ad avere anche effetti su altri nodi, e specificamente sui router posti sul cammino tra il nodo mittente dei datagrammi in multicast al gruppo, ed i destinatari che aderiscono al gruppo. La comunicazione nel gruppo si svolgerà attraverso una rete che potrà essere descritta come una foresta di alberi, con le radici sui mittenti del multicast, i router multicast collocati nei nodi intermedi dell'albero, e gli aderenti collocati nelle foglie o nei nodi intermedi della foresta. Parleremo dunque di *nodi a valle*, intendendo i nodi più lontani dalla radice rispetto ad un certo nodo. Rispettivamente, parleremo di *nodi a monte* per quelli più vicini alla radice di un certo albero.

Un router multicast aderisce ad un gruppo quando almeno uno dei nodi collegati ad una delle sottoreti da lui servite desidera iscriversi a quel gruppo, collocandosi a valle del router stesso.

Il protocollo utilizzato per gestire le adesioni dei router si chiama IGMP, ed è un requisito per la conformità di livello 2. La struttura del messaggio IGMP è in figura 4.18. I messaggi vengono incapsulati in datagrammi IP, come già avveniva per ICMP.

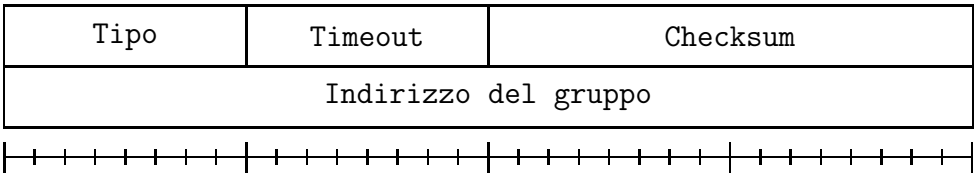


Figura 4.18 Formato di un messaggio IGMP

Alcuni router (vedremo in seguito quali) spediscono periodicamente (ogni pochi minuti) un messaggio IGMP di *general query* con un TTL di 1 ed un tipo `0x11` indirizzato al gruppo `224.0.0.1`, che sappiamo corrispondere a tutti i nodi della sottorete in grado di partecipare al protocollo IGMP. Il messaggio ha lo scopo di raccogliere le richieste di adesione a gruppi avanzate da nodi a valle del router.

I nodi vicini rispondono con un messaggio IGMP di tipo `0x16`, detto di *membership report*, per ciascun gruppo cui appartengono. Le risposte non sono inviate immediatamente, ma dopo intervalli casuali, e la prima ad essere inviata inibisce la spedizione delle successive. In questo modo nella sottorete circolerà solo un messaggio di *report* per ciascun gruppo, evitando sovraccarichi.

La ricezione di un messaggio di *membership report* relativo ad un certo gruppo, rappresenta per il router una richiesta di adesione ad un gruppo da parte di un destinatario finale, o di un router a valle. Se il router interessato già aderisce a quel gruppo, non viene intrapresa alcuna azione. Viceversa, il router innescherà un procedimento di adesione al gruppo, replicando con un *membership report* contenente l'indicazione del gruppo al prossimo messaggio di *general query* che riceverà.

Quindi l'adesione di un nodo ad un certo gruppo determina, attraverso i messaggi di *membership report*, una catena di adesioni che risalgono sino ad un router già aderente al gruppo. Ogni anello della catena si chiuderà quando un router a monte riceverà, in risposta ad un messaggio *general query* un messaggio *membership report* che specifica l'adesione al gruppo dei router a valle. La catena di adesioni costituirà un nuovo ramo nella foresta che rappresenta la diffusione del multicast. Le informazioni in multicast inizieranno a fluire nel nuovo ramo solo quando la catena raggiungerà un router già aderente al gruppo.

Risulta necessario che, entro una certa rete, solo uno dei nodi possa spedire il messaggio di *general query*. La semplice regola stabilisce che il router con indirizzo IP inferiore è quello che ha titolo a spedire il datagramma di interrogazione. Eventuali "usurpatori" possono produrre problemi temporanei, ma vengono immediatamente riconosciuti dagli altri router sulla stessa rete, e rileveranno essi stessi di usurpare il titolo. Questo avverrà non appena il router con IP minimo invierà il proprio messaggio di *general query*. Il protocollo, in apparenza semplice, nasconde un certo numero di tecniche di raggiungimento del consenso molto interessanti, spesso gestite tramite timeout. Per maggiori dettagli potete consultare l'RFC2236 [9].

Per quanto riguarda l'abbandono di un gruppo, nella versione 1 di IGMP, se nessuno aderisce ad un certo gruppo per un certo periodo di tempo, il gruppo viene cancellato da quelli di interesse per il router. Nella versione 2 esiste invece un tipo di messaggio appropriato a questa funzione (con tipo `0x17`), ed il procedimento di abbandono è reso più rapido.

4.5.6 L'Mbone

Quando una nuova tecnologia viene introdotta in una infrastruttura non preparata, può accadere che la nuova tecnologia trovi la strada sbarrata dalle carenze della infrastruttura esistente.

Questo è avvenuto con il multicast: i router ed nodi preparati alla gestione del multicast (cioè a livello di conformità superiore allo 0) sono una ristretta minoranza, apparentemente destinati a rimanere "isole" nell'Internet.

Per riuscire a collegare tra loro queste isole, si utilizza il meccanismo del tunnelling, che abbiamo già visto nel paragrafo 4.4: i messaggi con IP destinatario di classe D (cioè in multicast) vengono incapsulati entro un datagramma indirizzato ad un altro router che sia in grado di gestire il multicast.

Con questa tecnica viene realizzata una rete virtuale, che viene chiamata *Mbone*, che utilizza tunnel per trasferire datagrammi in multicast tra le "isole" Internet che sono in grado di trattare il multicast.

Esercizi

- Descrivete almeno tre eventi che possono portare all'inconsistenza della tabella ARP.
- Date almeno un pro ed un contro per la deframmentazione in un nodo intermedio, piuttosto che sul nodo destinazione, di un pacchetto che necessita di frammentazione.
- Esiste un caso di loop dovuto ad incapsulamento che non viene coperto dalle due tecniche illustrate dalla figura 4.17?
- Un albero multicast serve N nodi e ciascun nodo ha almeno 4 figli. Quanti nodi sono interessati, al massimo, da una operazione di *join*?

Capitolo 5

Lo strato trasporto di Internet

In una gerarchia di protocolli di comunicazione, lo strato trasporto deve consentire a più applicazioni di condividere le funzionalità offerte dall'interfaccia dello strato di rete, assegnando a ciascuna applicazione un indirizzo diverso e gestendo il multiplexing dei pacchetti uscenti ed il demultiplexing dei pacchetti entranti.

Queste due funzionalità di base vengono offerte tramite due generi di protocolli:

- quelli che trasferiscono datagrammi, e che non mantengono uno stato per l'associazione tra i nodi messi in comunicazione, e
- quelli che realizzano un circuito virtuale, e che mantengono uno stato per l'associazione tra i nodi messi in comunicazione.

I primi vengono anche detti *connectionless*, mentre i secondi *connection oriented*.

Un protocollo che non mantenga uno stato ha ben poche possibilità di controllo sulla comunicazione: ad esempio, non potrà ricostruire l'ordine di trasmissione dei datagrammi, non potrà rilevare la perdita di pacchetti, non sarà in grado di coordinare una "conversazione" tra i nodi che interconnette. In compenso offrirà migliori prestazioni, in quanto non dovrà gestire lo stato della connessione.

Quindi i due generi di protocolli offrono caratteristiche complementari, ed una architettura completa offre entrambe le modalità di trasporto.

L'architettura di Internet si basa su un *protocollo di rete*, IP, che offre un servizio a datagrammi: il *protocollo di trasporto a datagrammi* di Internet, denominato *User Datagram Protocol* (UDP), è quindi semplice nella sua

realizzazione, poiché si limita ad aggiungere il multiplexing/demultiplexing alla comunicazione *connectionless* offerta da IP. Invece il *protocollo di trasporto a circuito virtuale*, denominato *Transmission Control Protocol* (TCP), dovrà farsi carico della costruzione e gestione dello stato, e sarà dunque più complesso.

I due protocolli hanno in comune il meccanismo attraverso il quale si realizza il multiplexing: le *porte*. Si tratta di una forma di indirizzo interno ad un singolo nodo: un datagramma UDP od un segmento TCP vengono dunque recapitati ad un certo indirizzo IP, e internamente al nodo associato a questo indirizzo vengono inoltrati su una certa porta.

Una porta viene identificata da un numero: i numeri nell'intervallo 0-1024 assumono un significato particolare, in quanto le applicazioni collegate a queste porte realizzano funzioni standard. Queste porte vengono dette *note* (*well-known* nella terminologia inglese), riferendosi al fatto che l'applicazione che richiede una connessione su quelle porte sa di entrare in comunicazione con una certa altra applicazione, nota a priori.

Ad esempio l'applicazione `ftp` risponde sulla porta 21 di ogni host, `telnet` sulla porta 23, `http` sulla porta 80, e tutte utilizzano il protocollo di trasporto TCP. L'elenco delle porte note è pubblicato nell'RFC "Assigned Numbers", aggiornato periodicamente. L'ultimo RFC del genere è il numero 1700 [20].

In questo capitolo studieremo nel dettaglio i due protocolli UDP e TCP, mentre nel successivo utilizzeremo i protocolli di trasporto di Internet per costruire alcuni semplici esempi di applicazione. Per questo useremo i socket, introdotti dal sistema operativo UNIX ma poi estesi ad altri tipi di sistemi operativi quando Internet uscì dai laboratori di ricerca, che offrono una semplice interfaccia alle funzionalità offerte dallo strato trasporto.

5.1 UDP – User Datagram Protocol

Il protocollo UDP (definito dall'RFC768 [16]) consente la condivisione delle funzioni di base di IP tra più applicazioni, senza creare interferenze ed in maniera completamente trasparente.

Quindi il modulo UDP si incarica di realizzare il multiplex/demultiplex delle funzionalità offerte dal protocollo IP, ed ha le caratteristiche di un protocollo senza stato per lo scambio di datagrammi.

L'unica funzione di "controllo" effettuata da UDP consiste nella verifica della checksum dei dati contenuti in un singolo datagramma.

Rispetto a TCP, estremamente più ricco poiché offre una comunicazione a circuito virtuale, UDP può essere preferibile quando non sono necessarie le prestazioni di TCP. Inoltre UDP è specificamente indicato per le comunicazioni in *multicast*, per le quali TCP non è utilizzabile, in quanto vincolato alla realizzazione di un circuito virtuale bidirezionale tra due applicazioni. Un protocollo di trasporto multicast a circuito virtuale è in realtà ancora oggetto di ricerca, e vedremo una proposta nel capitolo dedicato ad RTP.

5.1.1 Il formato del datagramma UDP

Il formato (in figura 5.1) è molto semplice, e contiene solo il necessario per identificare la porta di arrivo, e per verificare la checksum.

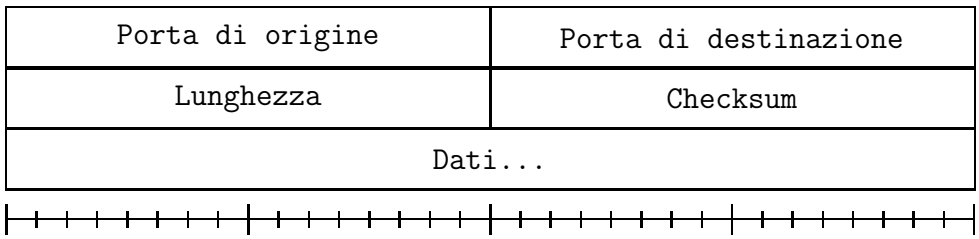


Figura 5.1 Formato del datagramma UDP

Il campo contenente la porta mittente è opzionale, e viene utilizzato solo se necessario a trasmettere, a cura dell'applicazione, un riscontro dal destinatario al mittente.

Il campo checksum può essere lasciato a 0, indicando così che la checksum non è stata calcolata.

Nell'RFC in cui viene definito UDP, per il calcolo della checksum viene utilizzato un algoritmo molto semplice: il campo checksum viene impostato calcolando il complemento a 1 della somma (in complemento a 1) di tutti i dati e dello header, più gli indirizzi IP di mittente e destinatario, la lunghezza del pacchetto, ed il numero di protocollo. In questo modo possono essere identificati errori di recapito. Se il calcolo restituisce il valore 0, la checksum viene impostati a tutti 1, così che la checksum non può mai avere valore 0: tale valore viene infatti utilizzato per indicare che il messaggio non contiene la checksum.

Un errore di checksum dà luogo alla rimozione del datagramma ([10]).

5.1.2 Funzionalità di UDP

Le funzionalità offerte da UDP all'applicazione soprastante si possono riassumere nei seguenti punti ([16]):

- possibilità di aprire nuove porte in ricezione – Questo corrisponde normalmente ad aprire un nuovo servizio, e l'applicazione che effettua l'apertura della nuova porta diventa il *server* per quel servizio. L'apertura di una porta ha molti punti in comune con l'apertura di un file.

- possibilità di ricevere datagrammi dalle porte aperte – Inoltre all'applicazione viene notificata la provenienza del messaggio (porta e indirizzo IP del mittente).

- possibilità di spedire datagrammi.

5.1.3 La trasparenza di UDP

Il protocollo UDP deve fornire all'applicazione soprastante un controllo quanto più possibile completo dello strato IP sottostante.

In particolare, UDP deve passare all'applicazione, lasciandole inalterate, le opzioni specificate nel datagramma IP. In particolare questo vale per la registrazione, l'indicazione e la restrizione del routing, e per la temporizzazione. Una applicazione che utilizza UDP dovrebbe provvedere ad impostare il cammino di ritorno di un eventuale datagramma di risposta invertendo la route indicata dal mittente con l'opzione di *source routing*.

L'interfaccia offerta da IP, sintetizzata nei capitoli precedenti con le funzioni `GET_SRCADDR` (la descrizione dell'indirizzo IP adatto per un certo destinatario) e `GET_MAXSIZES` (la lunghezza di datagramma adatta per un certo destinatario), deve pure essere offerta da UDP alle applicazioni dello strato superiore.

Nell'altro senso, la funzionalità di UDP deve fornire all'applicazione dello strato superiore la possibilità di indicare i campi TOS e TTL, e di indicare le opzioni da inserire nel datagramma IP.

Nel datagramma IP, il protocollo UDP viene indicato con il *numero di protocollo* 17.

5.1.4 Trasparenza rispetto a ICMP

La trasparenza di UDP deve applicarsi anche al protocollo ICMP: le comunicazioni ricevute tramite questo protocollo devono essere riportate integralmente allo strato superiore

Questo genere di comunicazione ha carattere asincrono, diversamente dalla `RECEIVE` che invece ha carattere sincrono.

5.2 TCP – Transmission Control Protocol

Il protocollo TCP è un protocollo che, utilizzando il servizio orientato ai *datagrammi* offerto da IP, produce invece un servizio orientato alla *connessione*. Questo genere di servizio viene anche chiamato, con una analogia alla tecnologia telefonica, *circuito virtuale*. La comunicazione è *asincrona* e *bidirezionale*, e ciascuna direzione di comunicazione viene gestita come una *connessione* distinta.

Possiamo considerare la *connessione* come un canale di comunicazione tra due processi, residenti su due host connessi allo stesso Internet. Il livello IP nasconde completamente qualsiasi dettaglio relativo alla collocazione dei due processi sulla rete, quindi ogni considerazione circa il routing o altre caratteristiche della rete non sono visibili al processo che utilizza l'interfaccia offerta da TCP.

Vediamo ora una descrizione delle caratteristiche funzionali del protocollo TCP, prima di entrare nei dettagli della sua realizzazione.

Lo stato di una connessione

Mentre il *datagramma* è semplicemente un blocco di ottetti che vengono trasferiti come tali, una *connessione* è un'entità complessa caratterizzata da uno stato interno. Ad esempio, una proprietà della *connessione* che richiede il mantenimento di uno stato è che i dati devono essere recapitati *nello stesso ordine in cui sono stati spediti*. Ma anche la modalità di controllo del flusso utilizzata da TCP richiede la presenza di uno stato della connessione.

Lo *stato di una connessione* viene memorizzato in una struttura dati predefinita, contenente gli indirizzi degli host coinvolti, i riferimenti per il trasferimento ordinato dei dati, le informazioni per il controllo del flusso. Questa struttura è detta *Transmission Control Block* (TCB), e racchiude tutti i dati necessari alla gestione di una certa connessione.

Oltre alle operazione di base di `send` e di `receive`, tra le funzioni fondamentali offerte dallo strato TCP ci saranno anche l'apertura (`open`) e la chiusura (`close`) di una connessione, destinate a creare ed inizializzare, e infine a distruggere, il TCB.

Il multiplexing

Il protocollo TCP deve poter essere disponibile a più *processi utente*: come già visto a pag. 128, per ottenere questo risultato viene utilizzata la tecnica di associare a ogni connessione anche una coppia di *numeri di porta*, uno per ciascuno dei due nodi connessi.

Il buffering

Il protocollo TCP utilizza un meccanismo di *buffering*: il modulo TCP mittente può infatti accumulare i dati in un *buffer* sul mittente prima di immetterli nella rete, frammentati nella PDU propria dello strato TCP, il *segmento*. Anche il modulo TCP ricevente può operare in modo analogo, accumulando i segmenti ricevuti in un *buffer* prima di recapitarli al processo utente. In ambedue i casi il buffer verrà svuotato quando il sistema operativo o il mittente lo riterranno opportuno. Il sistema operativo innescherà lo svuotamento dei *buffer* in concomitanza al riempimento del buffer, o allo scadere di un timeout o con altri criteri. Il mittente può forzare lo svuotamento dei buffer richiedendo al modulo TCP una operazione di `flush`, di cui non viene informato il destinatario.

Poiché i dati trasferiti da TCP attraverso una *connessione* sono recapitati nello stesso ordine in cui sono spediti, possono essere organizzati in un *flusso di dati*. Data la presenza della funzionalità di *buffering*, i processi utente non potranno tuttavia confidare che il contenuto di una singola `send` venga ricevuto con una singola `receive`: i processi utente dovranno quindi introdurre un proprio protocollo, se necessario, per interpretare le informazioni contenute nel flusso di dati che attraversa la connessione. Ad esempio, un protocollo come `telnet`, destinato a trattare comandi di una linea, interpreterà (salvo eccezioni) un "a capo" come chiusura di un comando ed inizio del successivo. Quindi suddividerà i dati ricevuti con le `receive` in modo da isolare sequenze di ottetti separate da caratteri di "a capo".

Nel capitolo 6 dedicato ai socket UNIX, vedremo un esempio di come dati spediti da `send` distinte possano essere letti con una singola `receive`.

La tolleranza ai malfunzionamenti della rete

Poiché una connessione TCP si presenta come un utente persistente della rete, è opportuno che sia in grado di sopportare certe modalità di malfunzionamento della rete. In tali circostanze i processi utenti possono osservare una momentanea degradazione delle prestazioni della rete, ma le proprietà funzionali della connessione, ad esempio il recapito ordinato dei dati, devono restare valide. Quindi è necessario garantire che le attività basate sulla connessione possano proseguire, e non debbano essere necessariamente azzerate da un malfunzionamento della rete.

TCP è in grado di sopportare e porre riparo (*recover*) a quattro circostanze di errore, le più frequenti tra quelle introdotte da IP:

- *danneggiamento* dei dati,
- *perdita* dei dati,
- *ripetizione* dei dati, e
- *consegna disordinata* dei dati.

Da notare che, da un punto di vista molto formale, solo nel primo caso si può parlare di *malfunzionamento* di IP: negli altri casi, si tratta di comportamenti ammessi, anche se preferibilmente infrequenti.

Per quanto riguarda il danneggiamento dei dati, per la sua *rilevazione* viene inserita una checksum in ciascun segmento: in caso venga rilevato un errore, l'evento verrà trattato come la perdita del pacchetto, che vediamo di seguito.

Per tollerare la perdita, la duplicazione (che può essere conseguenza dell'aver diagnosticato erroneamente una perdita) o il recapito disordinato dei segmenti, ogni ottetto trasferito attraverso la *connessione* deve essere identificato da un numero d'ordine [1].

Per realizzare ciò, con ogni segmento viene inviato anche il numero d'ordine del primo ottetto trasmesso nella sua parte dati: in questo modo è possibile associare a ciascun ottetto il proprio numero d'ordine.

Inoltre con un segmento può essere trasferito anche, come *riscontro* (`ACK`), il numero d'ordine del primo ottetto non ricevuto: l'applicazione che riceve

[1] **attenzione**: il numero d'ordine viene associato all'ottetto, non al segmento! Associare un numero all'intero segmento non consentirebbe, ad esempio, di ritrasmettere un segmento perduto scomponendolo in segmenti più piccoli

un riscontro sa che fino a quel numero d'ordine tutti gli ottetti sono stati ricevuti. Una tecnica suggerita dall'RFC consiste nel preparare, all'atto della trasmissione di un segmento, un analogo segmento in una *coda di ritrasmissione*, pronto per essere spedito dopo un certo *timeout* nel caso in cui non venga riscontrata la sua ricezione. Utilizzando questa tecnica è possibile tollerare la perdita di un segmento, introducendo tuttavia il rischio che il mittente riceva segmenti duplicati.

L'eliminazione dei duplicati si ottiene collocando l'informazione ricevuta entro un buffer che rispecchi i numeri d'ordine degli ottetti: l'informazione duplicata potrà essere rilevata, o semplicemente ricoprire quella già presente. Una scelta appropriata per il buffer di ingresso può essere un buffer circolare.

Con la stessa tecnica si ottiene il riordinamento dei dati: questi verranno subito collocati in maniera appropriata nel buffer.

Il controllo del flusso

Ancora in relazione al fatto che una *connessione* si presenta come un utente persistente della rete, è opportuno fornirla dei mezzi per utilizzare al meglio le risorse offerte dalla rete, evitando comportamenti che possano portare alla congestione: in primo luogo, si vuole evitare che il mittente sovraccarichi la rete ed il destinatario con dati che non possono ancora essere *consumati* dal destinatario. Questa situazione può prodursi come effetto di diverse cause: ad esempio il destinatario può essere lento a consumare i dati, oppure la perdita di alcuni dati intermedi, dovuta a problemi di comunicazione, può rendere necessario attenderne la ritrasmissione. A meno di un efficace *controllo del flusso*, i buffer sul percorso della connessione potrebbero esaurire la loro capacità.

Il *controllo del flusso* in TCP, che vedremo in dettaglio nel seguito, fa riferimento alla tecnica base della *sliding window*. Per realizzare questa tecnica, il destinatario, insieme al *riscontro* che conferma la ricezione dei dati, spedisce pure una *finestra*, che indica quali sono gli *ottetti* che è in grado di ricevere. Il mittente dovrà astenersi dall'inviare *ottetti* che non appartengono alla finestra indicata, mentre il destinatario potrà scartare gli *ottetti* che non ricadono nella finestra indicata.

Insieme al *riscontro* viene spedito con ogni segmento anche un numero d'ordine, che indica il limite superiore della *finestra*: in questo modo il mittente viene informato che gli ottetti con un numero d'ordine superiore potrebbero (non necessariamente) essere scartati.

Date tutte queste misure, e nell'ipotesi che le funzioni di *routing* siano efficaci, TCP funzionerà correttamente a meno che la rete non sia partizionata da un guasto per un periodo prolungato.

Efficienza nello scambio delle informazioni di controllo

Si vede che, in questo modo, le comunicazioni in una direzione serviranno a trasportare informazioni di controllo (riscontro e finestra) per le comunicazioni nell'altra direzione, rendendo particolarmente efficiente la gestione delle due connessioni.

Inoltre, essendo nato in ambito militare, il protocollo TCP è predisposto per gestire la trasmissione di informazioni confidenziali, e a diversi livelli di precedenza.

Per concludere, riassumiamo gli aspetti salienti del protocollo TCP:

- È utilizzabile da più processi utente sullo stesso host.
- Trasferisce una sequenza ordinata di ottetti, suddividendola in segmenti.
- È affidabile.
- Previene l'utilizzo inefficiente dei buffer.
- Gestisce precedenza e confidenzialità.

5.3 Realizzazione del protocollo TCP

Per esaminare il modo in cui vengono realizzate le specifiche funzionali del protocollo TCP esamineremo prima il formato dello header del segmento, poi il modo in cui vengono trattate le informazioni nello header, ed in particolare la gestione dei numeri d'ordine e della finestra. Infine vedremo nel dettaglio il modo in cui una connessione viene aperta e chiusa.

5.3.1 L'intestazione del segmento TCP

La struttura interna di un segmento è complessa: devono infatti trovare posto le informazioni necessarie al multiplexing, alla gestione del flusso ed alla realizzazione delle altre funzionalità di TCP.

Il segmento è dunque caratterizzato da una intestazione (in figura 5.2) che contiene le informazioni di controllo, ed un payload, che contiene una porzione di dati estratti dal buffer di spedizione.

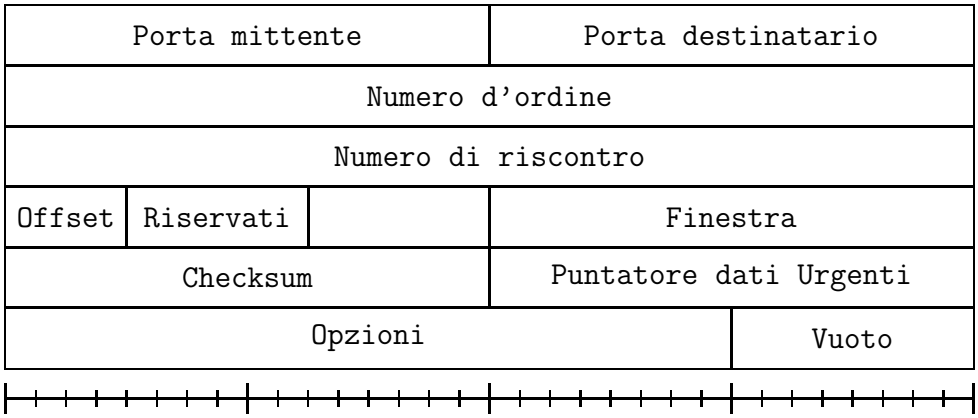


Figura 5.2 Intestazione di un datagramma TCP

I campi dell'intestazione sono così interpretati:

Porte Corrispondono ai numeri di porta mittente e destinatario.

Numero d'ordine È il numero d'ordine del primo ottetto di dati nel segmento.

Numero di riscontro È il numero d'ordine del prossimo ottetto atteso: una volta stabilita la connessione, questo dato viene sempre spedito in ambedue le direzioni.

Offset È la lunghezza dell'intestazione in parole da 32 bit (4 ottetti): corrisponde all'*offset* dei dati nel segmento.

Flag L'ottetto rappresenta, una per bit, le seguenti caratteristiche del segmento:

`URG` Segnala che il segmento contiene informazioni urgenti.

`ACK` Segnala che il segmento contiene un acknowledgement significativo.

`PSH` Richiede una operazione di `PUSH`: consiste nello svuotamento dei buffer al destinatario, e segue una *flush* del mittente.

`RST` Richiede il reset della connessione.

`SYN` Segnala la sincronizzazione dei numeri d'ordine.

`FIN` Richiede la conclusione della connessione.

Finestra Il numero di ottetti (successivi a quello indicato nel riscontro) che possono essere accettati. La dimensione di questo campo si rivela sempre meno adeguato, e ci si prepara ad estenderlo a 4 ottetti. Naturalmente il processo non può essere immediato, e si attende la prossima versione di TCP/IP ...

Checksum Calcolata, oltre sullo header, anche su uno "pseudo header", che contiene gli indirizzi IP, il numero di protocollo e la lunghezza del pacchetto TCP.

Puntatore ai dati urgenti Il segmento può contenere dati da elaborare prima degli altri. Questo campo indica il numero d'ordine dell'ottetto da cui iniziare l'elaborazione dei dati urgenti, ed è significativo solo se il flag `URG` è attivo.

Opzioni Una sola opzione è definita nello standard, e serve a informare il destinatario sulla lunghezza massima del segmento che il mittente potrà elaborare, cioè la `MMS_R` del mittente.

Vuoto L'allineamento deve sempre essere ai 4 ottetti.

5.3.2 I campi del Transmission Control Block

Lo stato di una connessione viene memorizzato in una struttura dati, detta TCB. Tra le componenti di questa struttura, la descrizione dei *socket* interessati, informazioni sulla precedenza e la confidenzialità, i puntatori ai buffer, alla coda di ritrasmissione e al segmento che si sta trattando, ricevuto o spedito.

Saranno presenti le variabili necessarie a coordinare lo scambio di informazione, tutte espresse in termini di *numeri d'ordine*. Nella descrizione che segue, che riprende l'RFC, quelle con prefisso `SND` indicano lo stato della connessione uscente dal modulo in esame, quelle con prefisso `RCV` indicano lo stato della connessione entrante. Si sottintende che gli ottetti vengono indicati tramite il loro *numero d'ordine*.

`SND.UNA` Indica l'ultimo ottetto spedito e non riscontrato.

`SND.NXT` Indica il prossimo ottetto da spedire.

`SND.WND` Indica ampiezza della finestra, in ottetti.

`SND.UP` Indica l'ottetto iniziale dell'informazione urgente.

`ISS` Indica il primo ottetto spedito nella connessione.

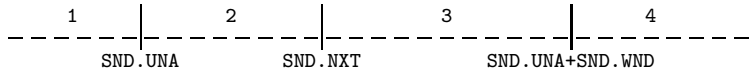
`RCV.NXT` Indica il prossimo ottetto da ricevere.

`RCV.WND` Indica l'ampiezza della finestra in ottetti.

`RCV.UP` Indica l'ottetto iniziale dell'informazione urgente.

`IRS` Indica il primo ottetto ricevuto nella connessione.

La rappresentazione grafica in figura 5.3 illustra le relazioni tra le variabili di spedizione, mentre la figura 5.4 le relazioni tra le variabili di ricezione. Il flusso di ottetti viene rappresentato come un segmento (si intende geometrico) orizzontale, e la coordinata di un punto sul segmento corrisponde al numero d'ordine di un ottetto. Vengono evidenziati con un tratto verticale i numeri d'ordine corrispondenti alle variabili descritte sopra, e le caratteristiche degli ottetti compresi in certi intervalli.



1 Numeri d'ordine di ottetti già riscontrati.

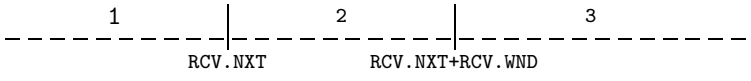
2 Numeri d'ordine di ottetti spediti ma non riscontrati.

3 Numeri d'ordine di ottetti che possono essere spediti.

4 Numeri d'ordine di ottetti che non possono essere ancora spediti.

Figura 5.3 Relazione tra le variabili `SND`

Per la natura sequenziale dei numeri d'ordine, un riscontro ha sempre valore cumulativo: insieme a quell'ottetto, si riscontrano anche tutti i precedenti. Tuttavia è necessario prestare grande attenzione al fatto che i numeri



1 Numeri d'ordine di ottetti già riscontrati.

2 Numeri d'ordine di ottetti che possono essere ricevuti.

3 Numeri d'ordine di ottetti che non possono essere ancora ricevuti.

Figura 5.4 Relazione tra le variabili `RCV`

d'ordine sono in numero limitato (2^{32}, limite dovuto alla lunghezza dei campi destinati ai numeri d'ordine nell'intestazione di TCP), mentre gli ottetti scambiati nel corso di una connessione sono in numero non limitato ed il numero d'ordine iniziale è arbitrario. Quindi tutte le operazioni devono essere effettuate in modulo 2^{32}!

L'eventualità che venga utilizzato lo stesso numero per due ottetti diversi è superata, nel funzionamento normale di una connessione, dal fatto che il periodo di rigenerazione degli stessi numeri d'ordine è senz'altro superiore al tempo di consegna, o di rimozione dalla rete in caso di errori di routing.

Il massimo tempo di permanenza di un segmento nella rete (*Maximum Segment Lifetime* (MSL)) corrisponde al tempo stimato necessario per consegnarlo al destinatario, o per rimuoverlo in caso di mancata consegna per problemi di connettività o routing. L'RFC consiglia di considerarlo di 2 minuti. Finché il ritmo di produzione di ottetti (e quindi la loro trasmissione) non eccede i 100 Mbit/sec il tempo che intercorre tra la generazione di due numeri d'ordine identici è superiore ai 5 minuti, dunque superiore alla MSL stimata. Ma il limite ormai è prossimo...

Ecco alcuni dei test che sarà necessario effettuare sui dati presenti nell'intestazione del segmento ricevuto. Consideriamo anche il segmento attualmente elaborato come parte dello stato, ed indichiamo così alcuni dei campi del suo header:

`SEG.SEQ` Il numero d'ordine del primo ottetto dei dati nel segmento.

`SEG.ACK` Il numero d'ordine riscontrato nel segmento.

`SEG.LEN` La lunghezza (calcolata) degli ottetti di dati nel segmento.

Il test seguente serve a verificare se il riscontro contenuto nel segmento ricevuto è significativo:

$$\texttt{SND.UNA} < \texttt{SEG.ACK} \leq \texttt{SND.NXT}$$

Se il numero d'ordine fosse inferiore al primo ottetto non riscontrato, potrebbe trattarsi di un riscontro allegato ad un segmento arrivato in ritardo, ed andrebbe trascurato. Se invece fosse relativo ad un ottetto non ancora spedito, potrebbe trattarsi di un errore.

Il test seguente serve a identificare la parte di segmento ricevuto che può essere accettata:

$$\texttt{RCV.NXT} \leq \texttt{SEG.SEQ} < \texttt{RCV.NXT} + \texttt{RCV.WND}$$

and

$$\texttt{RCV.NXT} \leq \texttt{SEG.SEQ} + \texttt{SEG.LEN} - 1 < \texttt{RCV.NXT} + \texttt{RCV.WND}$$

La prima condizione verifica se il primo ottetto nel segmento cade entro la finestra (l'intervallo 2 nella figura 5.4). Se il test fallisce perché il valore di `SEG.SEQ` cade a sinistra dell'intervallo, verrà eseguito anche il secondo test, altrimenti l'informazione ricevuta cade certamente fuori dalla finestra, ed il segmento verrà interamente scartato. Il secondo test verifica se l'ultimo ottetto del segmento cade entro la finestra. Se fallisce, potranno essere accettati solo gli ottetti del segmento che cadono entro la finestra, mentre gli altri dovrebbero essere scartati. Gli ottetti accettati saranno trasferiti nel buffer di ricezione.

5.3.3 Apertura di una connessione

L'apertura di una connessione è una operazione necessariamente asincrona: le applicazioni che saranno connesse non possono prendere la stessa decisione allo stesso istante!

In genere si distingue una apertura *passiva* ed una *attiva*: la *passiva* serve a richiedere l'apertura di una connessione in attesa che un'altra applicazione faccia una richiesta di apertura *attiva*. L'*apertura passiva* alloca una porta sul nodo che la richiede, ed eventualmente specifica la porta ed il nodo dal quale si attende la corrispondente apertura attiva. L'*apertura attiva* avverrà

specificando la porta sul nodo partner, ed allocando una porta sul nodo che la richiede. Quindi si tratta di due primitive simmetriche, l'una bloccante, l'altra no. Nel primo caso si dice anche che l'applicazione si prepara ad *accettare* connessioni su una certa porta.

Nel seguito, parleremo di *server* per indicare il processo che richiede l'apertura *passiva* della connessione, e *client* per indicare quello che richiede l'apertura *attiva*.

Un caso particolare di apertura passiva si ha quando il processo decide di accettare, su una certa porta, connessioni di cui non specifica la provenienza (porta o host). Questo caso si presenta in particolare quando un'applicazione intende presentarsi come *server* senza specificare l'identità del cliente. In questo caso la porta mittente indicata nella `OPEN` passiva sarà generica, rappresentata con un codice specifico al sistema operativo.

Determinate porte saranno associati a servizi "noti", ma il documento [2] indica la possibilità che esistano server specializzati nel fornire gli indirizzi di porta e l'indicazione dei nodi in grado di offrire un servizio specifico.

Anche se la modalità usuale per la creazione di una connessione avviene facendo corrispondere una apertura *passiva* con una *attiva*, lo standard prevede che anche la simultanea richiesta di due connessioni attive venga servita correttamente.

Il three way handshake

Una delle funzionalità fondamentali del protocollo di apertura di una connessione è la definizione del numero d'ordine del primo pacchetto inviato (*Initial Sequence Number* (ISN)) da ciascuno dei partner.

È considerato inopportuno che il numero d'ordine del primo ottetto inviato sia 0. Infatti la stessa connessione può essere riaperta e richiusa, un numero indeterminato di volte: restando identici gli host e le porte coinvolte, si tratterà di *reincarnazioni* successive della stessa connessione. Se allora ogni connessione iniziasse con ISN=0, una connessione chiusa appena aperta ed immediatamente (cioè entro MSL, v. pagina 139) ricostituita, presenterebbe numeri d'ordine duplicati.

Si raccomanda dunque di utilizzare un orologio per generare gli ISN per minimizzare la probabilità di generare, in incarnazioni successive, gli stessi numeri d'ordine. Legando la generazione dell'ISN ad un orologio interno, questa possibilità non si presenta se il tempo di ciclo del clock è ben superiore alla MSL. L'RFC consiglia un clock che avanzi di una unità ogni circa 4 μs:

il tempo di ciclo di questo clock sarebbe di circa 5 ore. Poiché questo tempo è sicuramente superiore a qualsiasi ragionevole MSL, concludiamo che non possono essere contemporaneamente in rete due aperture con lo stesso ISN.

La determinazione dell'ISN su una connessione necessita di una forma di accordo tra le due applicazioni coinvolte nella connessione. Infatti non esiste una nozione di tempo globale in rete, e quindi ciascuna applicazione non può considerare di conoscere il valore del clock che l'altro usa per generare il numero d'ordine iniziale.

L'accordo sui valori iniziali dei due numeri d'ordine (uno per ciascun processo) viene ottenuto tramite un procedimento indicato come *three way handshake*, nome che deriva dal fatto che sono tre i messaggi necessari per concludere l'operazione.

In sintesi, la comunicazione tra le due applicazioni che stanno stabilendo la connessione è la seguente (indichiamo con p_a il *client*, p_b il *server*):

1. $p_a \rightarrow p_b$ Il mio primo numero d'ordine è ISN_a
2. $p_a \leftarrow p_b$ Il tuo primo numero d'ordine è ISN_a
3. $p_a \leftarrow p_b$ Il mio primo numero d'ordine è ISN_b
4. $p_a \rightarrow p_b$ Il tuo primo numero d'ordine è ISN_b

Poiché il secondo ed il terzo passo possono essere incorporati nello stesso messaggio, i messaggi sono in realtà solo tre. Con solo due messaggi non è possibile ottenere un risultato analogo: infatti ciascuna applicazione necessita di ottenere un riscontro, mentre uno dei messaggi non può essere di riscontro.

I primi due messaggi coinvolti in questa fase iniziale hanno il flag `SYN` attivo. Il secondo ed il terzo hanno il flag di `ACK` settato. Il valore del primo numero d'ordine sarà indicato nel campo dell'intestazione destinato al numero d'ordine, mentre il riscontro sarà indicato nel campo destinato al riscontro. Il valore del flag `SYN` segnalerà che questi campi non vanno interpretati in maniera usuale, poiché è in corso il protocollo di sincronizzazione dei numeri d'ordine.

In figura 5.5 viene rappresentato il diagramma a stati dell'algoritmo. Gli stati hanno il seguente significato:

`CLOSED` La connessione non esiste.

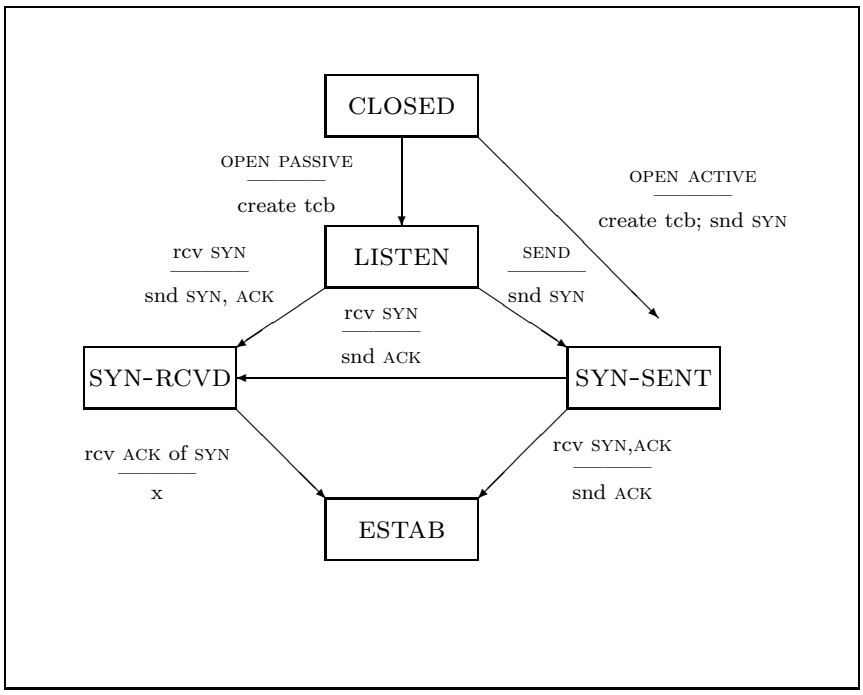


Figura 5.5 Diagramma a stati (semplificato) del three-way handshake

`LISTEN` La connessione è stata aperta in modalità passiva (server) e si attende la prossima richiesta.

`SYN-SENT` La connessione è stata richiesta con modalità attiva ed è stato spedito il messaggio `SYN` contenente il proprio `ISN`. Ora si attende il riscontro del partner, ed il suo `ISN`.

`SYN-RCVD` È lo stato di una connessione attiva ed utilizzabile per il trasferimento di informazione.

Le transizioni sono etichettate con un evento osservato (sopra) ed una azione intrapresa (sotto). Da notare che la transizione da `SYN-SENT` a `SYN-RCVD` si presenta solo nel caso in cui ambedue i partner abbiano operato una apertura attiva.

5.3.4 La chiusura "morbida"

La chiusura di una connessione richiede un protocollo simile a quello di apertura. Infatti l'operazione dovrebbe trovare il consenso di ambedue i processi

coinvolti: chi richiede la chiusura (indifferentemente, il *client* o il *server*) dovrebbe continuare a gestire la connessione, fino a che anche l'altro partner decida di concluderla.

Una delle circostanze che richiede particolare attenzione è proprio quella in cui una certa connessione viene prima chiusa e poi aperta, in rapida successione. In questo caso è ancora possibile, come già avevamo osservato a pagina 141, che ottetti appartenenti all'incarnazione precedente siano ricevuti erroneamente come parte dell'incarnazione successiva.

Per riparare a questa eventualità, è sufficiente che il modulo TCP ricordi l'ultimo numero d'ordine utilizzato in una certa incarnazione, ed utilizzi nell'incarnazione successiva un ISN successivo, eventualmente ottenuto utilizzando un orologio interno.

Esiste tuttavia la possibilità che la sconnessione avvenga in modo non controllato, ad esempio per il guasto di uno dei nodi, e che si perda la memoria degli ultimi numeri d'ordine utilizzati. In questo caso, dobbiamo introdurre l'ipotesi che un segmento venga eliminato dalla rete (anche grazie al TTL del modulo IP) entro un tempo limitato detto *Maximum Segment Lifetime (MSL)* (v. pagina 139). Allora saremo sicuri di non ricevere segmenti di una precedente incarnazione se, prima di riaprire la connessione, attenderemo un tempo corrispondente ad almeno MSL. Il suggerimento pratico dell'RFC è per una MSL di 2 minuti circa.

Un problema analogo sorge anche per i riscontri: i dati spediti ad un certo tempo t_0 potranno essere ricevuti al più al tempo $t_0 + \text{MSL}$, ed il relativo riscontro, spedito ancora a $t_0 + \text{MSL}$, potrà essere ricevuto sino al tempo $t_0 + 2 * \text{MSL}$.

Da qui l'opportunità di prolungare l'attesa, prima del rilascio della connessione dopo una conclusione anomala della connessione, fino a $2*$MSL: dopo tale istante, infatti in rete non saranno presenti più segmenti ne riscontri per gli ottetti inviati durante la connessione.

Il valore di MSL resta normalmente fissato entro i pochi minuti, e quindi una connessione rimane effettivamente impegnata per il doppio di quel valore dopo che ne è stata richiesta la chiusura.

Il protocollo per la chiusura di una connessione si basa dunque su un protocollo di *tree-way handshake*, ma è complicato dalla necessità di prevedere che tra la richiesta di chiusura e la chiusura di una connessione, può essere necessario continuare a gestire l'altra connessione.

Inizialmente il grafo (v. figura 5.6) presenta una biforcazione: a destra corrisponde al caso in cui la richiesta di chiusura arriva dal partner, con un

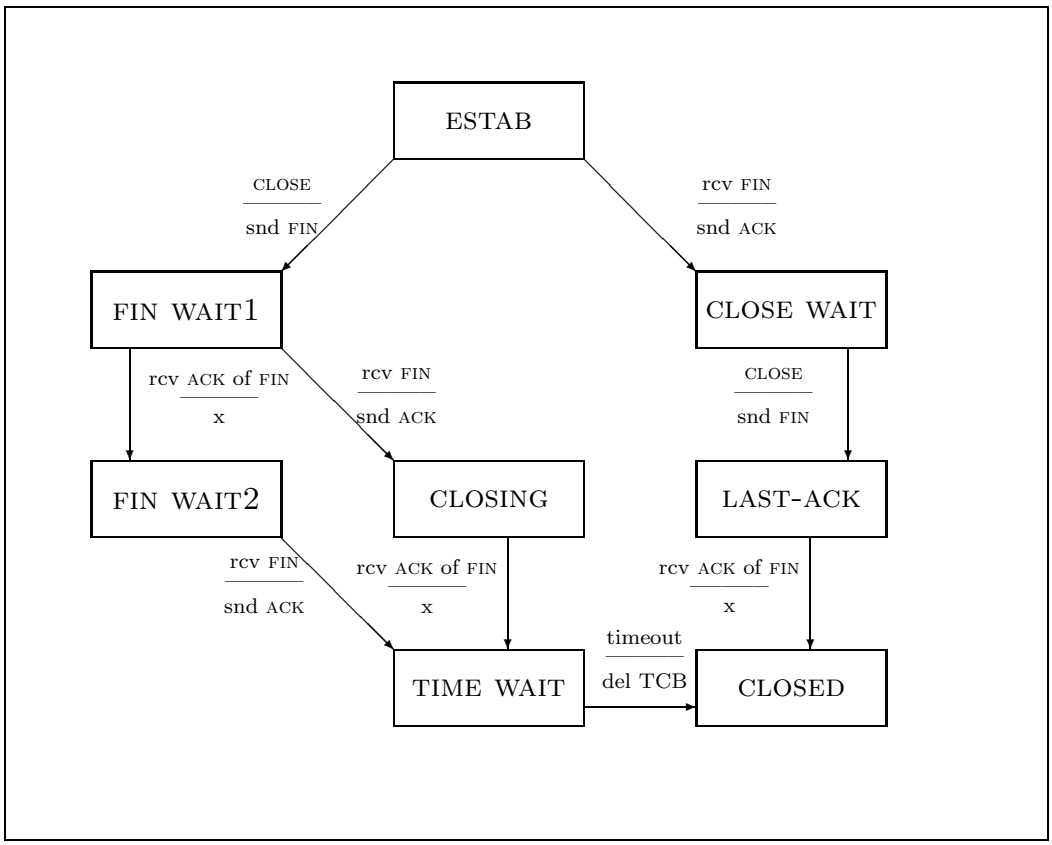


Figura 5.6 Diagramma a stati (semplificato) della chiusura di una connessione

segmento col flag `FIN` settato, mentre a sinistra la richiesta arriva dal processo utente, con l'invocazione della funzione **close**.

Nel primo caso, il protocollo richiede l'invio dell'acknowledgement e quindi il protocollo entra in uno stato in cui continua a gestire i segmenti in *uscita*, in attesa della richiesta di chiusura dal processo utente. Si tratta del percorso preferenziale per un *server*.

Nel secondo caso, il protocollo richiede l'invio del segmento `FIN`, e quindi il protocollo entra in uno stato in cui continua a gestire i segmenti in *ingresso*, in attesa della richiesta di chiusura e del riscontro della propria richiesta di chiusura. Al termine, il protocollo entra in una attesa lunga $2 * \text{MSL}$ prima di considerare la connessione chiusa. L'attesa è necessaria a garantire che il proprio riscontro sia ricevuto dal partner, che quindi provvederà alla chiusura per parte sua. Questa è la modalità di chiusura appropriata per un *client*.

Esiste la possibilità che ambedue i partner utilizzino il ramo sinistro, quando simultaneamente decidano di chiudere la connessione.

Quella che segue è la descrizione degli stati in cui si può trovare la connessione durante il protocollo di chiusura.

`FINWAIT-1` In chiusura di connessione, si attende la richiesta di chiusura o l'acknowledgement del partner.

`FINWAIT-2` In chiusura di connessione, si attende la richiesta di chiusura.

`CLOSE WAIT` In chiusura di connessione, si attende la richiesta di chiusura da parte dell'applicazione.

`CLOSING` Si attende il riscontro di una richiesta di chiusura, ma si continuano a ricevere segmenti.

`LAST-ACK` Si attende il riscontro di una richiesta di chiusura (che includeva il riscontro alla sua richiesta).

`TIME WAIT` La connessione è chiusa, ma si attende il timeout per l'esaurimento dei segmenti e dei riscontri.

5.3.5 Gli eventi

Possiamo suddividere gli eventi a cui deve fare fronte il modulo TCP in

- eventi prodotti da una applicazione dello strato superiore,
- eventi prodotti dal modulo IP, e
- eventi di timeout.

Li vediamo in qualche dettaglio: per una descrizione più approfondita rimando all'RFC793 [2].

Gli **eventi prodotti dall'applicazione** possono essere caratterizzati dai parametri di una chiamata di funzione: al solito, questo è solo un modo per rappresentare una interfaccia.

`OPEN (local port, foreign socket, active/passive)`
⇒ `local connection name` – richiede l'apertura della connessione.

Si distingue in `active` e `passive`. La prima corrisponde all'azione del *client* che richiede un servizio ad un server che considera già attivo, mentre la seconda corrisponde all'azione del server che offre un servizio, senza ancora prestarlo.

L'operazione di `OPEN` conduce sino allo stato `ESTAB`, e prende come parametri necessari la porta locale su cui si vuole instaurare la connessione, il socket remoto, e la modalità di apertura.

Nel caso la modalità sia passiva, l'apertura della connessione risulta bloccante, in attesa di una apertura dall'altro lato.

Nel caso la modalità sia attiva, l'operazione termina con successo se la connessione viene stabilita entro il `timeout` dato, altrimenti fallisce.

La connessione che viene restituita rappresenta una coppia di socket.

`SEND (local connection name, buffer address, byte count, PUSH flag, URGENT flag)` – richiesta di spedizione di un segmento.

Se viene indicato il flag `PUSH`, i dati vengono immediatamente trasmessi al destinatario (insieme agli altri accumulati nel buffer), ed il bit `PUSH` viene attivato nell'**ultimo** segmento inviato.

Se viene indicato il flag `URGENT`, i segmenti avranno il flag corrispondente attivato.

Nell'RFC dedicato al protocollo TCP [2], vengono esaminate diverse alternative per le caratteristiche *non funzionali* della `SEND`: in particolare, se debba o meno essere bloccante, se si debba verificare o meno un *rendez-vous* tra le applicazioni. Noi ci limitiamo a portare un esempio di una implementazione ormai dominante, i socket BSD-UNIX. In questa, la `SEND` non è bloccante.

`RECEIVE (local connection name, buffer address, byte count)` $\Rightarrow$ `byte count, urgent flag, push flag` – ricezione di un segmento.

Nell'RFC [2] viene specificato che il flag di `PUSH` va passato all'applicazione. Questo non è strettamente necessario, e [10] lo afferma facoltativo (paragrafo 4.2.2.2). È invece importante che la segnalazione `URG` venga recapitata all'applicazione destinataria ([10] paragrafo 4.2.2.4): in realtà l'applicazione dovrebbe essere immediatamente informata (tramite una trap o simile) della presenza di dati urgenti, e quindi continuerà a leggere segmenti sino a trovare quello contenente il flag, e tutti quelli con lo stesso flag attivo.

Anche in questo caso, l'RFC lascia notevole libertà nell'interpretare le caratteristiche non funzionali della `RECEIVE`.

`CLOSE (local connection name)` – chiusura della connessione.

La chiusura si intende sempre *morbida*: i dati relativi a `SEND` già richieste verranno completati, ed eventuali future `RECEIVE` verranno pure ricevute, sino alla `CLOSE` del partner.

Come abbiamo visto, l'operazione di chiusura non è istantanea: oltre a dover svuotare i buffer (una `CLOSE` implica sempre un'operazione di *flush*), la connessione rimarrà ancora attiva per il tempo necessario a non avere più messaggi obsoleti in circolazione.

`STATUS (local connection name)` $\Rightarrow$ `status data` – restituisce informazioni relative allo stato della connessione, in parte contenute o derivate dal TCB.

`ABORT (local connection name)` – richiede l'interruzione della connessione, con la perdita dei segmenti la cui trasmissione era incompleta.

L'**evento prodotto dallo strato inferiore** coincide con l'arrivo di un nuovo segmento: nel descrivere i protocolli di apertura e chiusura abbiamo trattato tutti gli eventi di questo tipo che rientrano nel funzionamento normale del protocollo. Abbiamo invece trascurato i casi in cui gli eventi si presentino "al momento sbagliato", probabilmente causati da malfunzionamenti hardware o errori di implementazione.

Tralasciando dunque i casi di inconsistenza (per una trattazione esaustiva si rimanda a [2]), il segmento generico verrà trattato controllandone la consistenza rispetto alla finestra, ed inserendolo in una coda di ricezione ordinata rispetto al numero d'ordine. In caso di sovrapposizione tra segmenti, viene trattenuto il valore del segmento successivo.

Infine gli **eventi di timeout** sono i seguenti:

`USER TIMEOUT` Si tratta di un timeout controllato dall'applicazione: al suo scadere si opera in modo simile ad un `ABORT`.

`RETRANSMISSION TIMEOUT` L'evento è determinato dalla mancata restituzione del riscontro per un segmento spedito. Si risponde comunque rispedendo il **primo** segmento nella coda di ritrasmissione.

`TIME-WAIT TIMEOUT` L'evento si presenta nel corso della chiusura della connessione, e corrisponde all'attesa di $2 * \texttt{MSL}$ prima dell'effettiva distruzione del TCB.

Esercizi

- La transizione dallo stato `SYN-SENT` allo stato `SYN-RCVD` può accadere solo in un caso particolare: quale?
- Che vantaggi si ottengono "raddoppiando" una connessione per trasferire dati ad una velocità superiore? Descrivete la struttura dell'applicazione che utilizza questa tecnica. Quante porte utilizza? Come le utilizza?
- Una linea transatlantica lunga 2000 Km è caratterizzata da una capacità di 100 Mbps. Qual è il tempo necessario a trasferire un ottetto dalla stazione di partenza a quella di arrivo, se la velocità nel cavo è il 90% di quella della luce? Cosa succede quando, in una connessione TCP, viene perduto un segmento? Come è opportuno configurare le finestre?

Capitolo 6

TCP e UDP dal punto di vista dell'applicazione

Nei capitoli precedenti abbiamo visto le caratteristiche funzionali dei protocolli di trasporto di Internet: TCP e UDP. Il punto di vista adottato era quello degli RFC, cioè di chi specifica le funzionalità e l'implementazione del protocollo, non di chi ne utilizza le funzionalità.

In questo capitolo invece consideriamo l'interfaccia offerta dallo strato trasporto ai processi utente che realizzano le applicazioni. Il punto di vista è quello del programmatore che deve utilizzare le primitive offerte dal livello trasporto per scrivere il codice dell'applicazione.

Ogni sistema operativo ed ogni linguaggio offrono una propria definizione dell'interfaccia alle funzionalità di trasporto, e quindi è opportuno limitarsi ad un caso: presenteremo le primitive offerte dalla libreria `C` di GNU. Si tratta di software di pubblico dominio, che è portabile su tutte i sistemi operativi UNIX ed è adattabile anche ad altri sistemi operativi. Gli esempi che vedremo nel seguito sono stati realizzati per la piattaforma Linux.

Avvertenza Non è tra gli obiettivi di questo testo fornire gli elementi del linguaggio `C` necessari a comprendere il contenuto di questo capitolo. D'altra parte, gli elementi di `C` propedeutici sono abbastanza superficiali: serve conoscere la sintassi di base del linguaggio, le direttive per l'inclusione di intestazioni, e dunque le possibilità di compilazione separata. Chi volesse provare ad eseguire o modificare i programmi presentati, dovrebbe preferibilmente avere a disposizione un computer con una installazione del sistema operativo Linux.

6.1 I socket

L'astrazione utilizzata per rappresentare l'interfaccia del livello trasporto è il *socket*: ad un socket è associata una coppia (indirizzo IP locale, numero di porta locale). Una comunicazione supportata da TCP o UDP avviene sempre tra due porte. Nel caso di TCP, una coppia di porte identificano una connessione.

I *socket* vengono utilizzati dai processi utente: per aprire ed utilizzare un *socket* non sono in genere necessari particolari privilegi di esecuzione.

Nella libreria GNU C un *socket* viene rappresentato con lo stesso tipo di dato associato ad un file: il *file descriptor*. Per operare sul socket si utilizzeranno dunque le stesse operazioni che si usano per operare su un file (apertura, chiusura, scrittura e lettura), più alcune specifiche per l'associazione del *file descriptor* del socket locale ad un altro socket remoto, un passo necessario tanto a TCP quanto a UDP.

Non tutti i sistemi operativi supportano i *socket*: nella libreria GNU (che esiste per molti sistemi operativi, non solo per UNIX) il file di intestazione `sys/socket.h` esiste sempre, a prescindere dal sistema operativo, e sono sempre definite le funzioni sui socket. Queste tuttavia falliscono sempre se il sistema operativo non è in grado di supportare i socket. Questo garantisce che un programma scritto utilizzando la libreria GNU C sia sempre compilabile nei sistemi operativi per cui esiste questa libreria, ma fallisca nel caso in cui il sistema operativo non realizzi lo strato trasporto di Internet.

6.1.1 I socket: concetti di base

Aprendo un socket, è necessario specificare che *genere di comunicazione* si desidera ed il *tipo di protocollo* che vogliamo utilizzare. Parlando di *genere di comunicazione* ci riferiamo all'astrazione di comunicazione che l'utente vuole avere a disposizione. Si tratta in breve di dare risposta alle seguenti domande:

- Quali sono le unità di informazione trasferite?

  I dati possono essere trasferiti come una sequenza di byte senza struttura, oppure essere raggruppati in *pacchetti.*

- I dati possono essere perduti durante il trasferimento?

Può essere opportuno avere garanzie sul recapito dei dati spediti, o sulla loro consegna ordinata. Oppure la consegna disordinata o duplicata dei pacchetti può non costituire un problema, ed essere gestita efficacemente dall'applicazione.

- La comunicazione è limitata a due soli corrispondenti?

  Un protocollo può consentire una comunicazione simile a quella offerta da una linea telefonica: i due corrispondenti ottengono un canale di comunicazione dedicato a loro. In altri casi, la comunicazione può essere simile ad una casella postale: non esiste un canale dedicato alla comunicazione e i messaggi possono arrivare da più parti.

La risposta a queste domanda indirizza alla scelta di un *protocollo*: TCP (non strutturato, affidabile, preserva l'ordine, bidirezionale, uno a uno) o UDP (strutturato, non affidabile, consegna duplicata o disordinata, comunicazione molti a molti).

Inoltre è necessario indicare in quale *spazio dei nomi* o *namespace* va collocato il socket: se si tratta di un socket destinato alla comunicazione in Internet, o se invece serve come struttura per la comunicazione tra processi interni ad un nodo.

Queste tre caratteristiche della comunicazione, genere di comunicazione, protocollo e spazio dei nomi, sono in realtà strettamente collegate tra di loro: un certo protocollo è strettamente legato ad un certo spazio di nomi e ad un certo genere di comunicazione.

Consideriamo per primi i *generi di comunicazione* disponibili, e passeremo in seguito agli *spazi dei nomi* ed infine ai *protocolli*.

6.1.2 Il genere di comunicazione

Il *genere di comunicazione* specifica l'astrazione di comunicazione che si vuole realizzata dal socket. Qui vediamo i due tipi che conosciamo: orientata al datagramma ed alla connessione. Ne esistono altre, e chi fosse interessato può vederle con il comando `man socket`.

Nella libreria GNU C vengono definite delle costanti del linguaggio, che nel programma vengono poi utilizzate ovunque venga richiesto di specificare una certa opzione, nel caso in esame il genere di comunicazione. Le costanti vengono indicate da un identificatore, che viene esportato da un file header

specifico, nel caso in esame `sys/socket.h`. Nella specifica della libreria troveremo l'indicazione dell'identificatore associato all'opzione, ma non il valore associato all'identificatore: questo potrà essere anche differente da sistema a sistema, senza che il programmatore ne debba tenere conto.

Noi consideriamo due costanti per l'indicazione del genere di comunicazione:

Sinossi:
`int SOCK_STREAM`
Descrizione:
la comunicazione avviene tramite una linea che connette i due socket, ed i dati vengono trasferiti in maniera affidabile. Corrisponde ad una comunicazione *connection oriented*.

Sinossi:
`int SOCK_DGRAM`
Descrizione:
viene trasferito un singolo pacchetto tra due socket. Il trasferimento non è affidabile: i pacchetti possono non essere consegnati o consegnati in maniera disordinata. Corrisponde ad una comunicazione *connectionless*.

6.1.3 Lo spazio dei nomi dei socket

Noi consideriamo due spazi dei nomi: quello che identifica comunicazioni locali, e quello che identifica comunicazioni tramite Internet versione 4. Ne esistono altri, che non consideriamo. Come nel caso dei *generi di comunicazione*, per ciascuna alternativa viene definita una costante, che viene indicata nella funzione che associa un nome ad un socket.

Sinossi:
`int AF_LOCAL`
Descrizione:
indica un nome nello spazio dei nomi locali. È limitato a comunicazioni all'interno della stessa macchina. Il file di header che contiene il tipo relativo agli indirizzi in questo spazio è `sys/un.h`.

Sinossi:

`int AF_INET`

Descrizione:

indica un nome nello spazio di Internet versione 4. Il file di header che contiene il tipo relativo agli indirizzi in questo spazio è `netinet/in.h`.

6.1.4 Lo spazio dei nomi dei socket Internet

Per assegnare un nome ad un socket nello spazio di Internet versione 4 dobbiamo fornire due dati:

- l'indirizzo IP della macchina locale, e
- un numero di porta sulla macchina locale.

Queste informazioni sono contenute in una struttura C, che quindi specifica il *nome* del socket come combinazione di questi dati:

```
struct sockaddr_in {
   short int sin_family;        /* il namespace, AF_INET */
   unsigned short int sin_port; /* il numero della porta */
   struct in_addr sin_addr;     /* l'indirizzo Internet */
}
```

Nel campo `sin_family` deve essere indicato il nome dello spazio dei nomi di Internet, quindi `AF_INET`.

Nel campo `sin_port` dovrà essere indicato il numero di porta da destinare al socket. Si tratta di un numero tra 0 e 65535, ma i numeri sino a 1024 sono riservati a servizi noti: ad esempio, 23 è utilizzato da `telnet`. Nel file `/etc/services` sono elencati tutti i numeri di socket riservati, insieme al servizio connesso.

Nel campo `sin_addr` dovrà essere indicato l'indirizzo di rete Internet verso cui si vuole aprire il socket. Vedremo tra poco i dettagli per la compilazione di questo campo.

6.1.5 Gli indirizzi di rete

L'indirizzo di rete viene rappresentato con la notazione `xxx.xxx.xxx.xxx` che conosciamo, e corrisponde ad un indirizzo IP: ad esempio, un PC nel nostro centro di calcolo ha indirizzo `131.114.11.18`. Questo genere di notazione in base 256, visto come stringa, può essere convertita in una struttura di tipo `struct in_addr` e viceversa. Quelle che seguono sono le due funzioni che operano quelle trasformazioni:

Sinossi:

```
int inet_aton(const char* name, struct in_addr *addr)
char *inet_ntoa(struct in_addr addr)
```

Descrizione:

La prima trasforma la stringa `name` e registra la struttura così costruita in `addr`. La seconda invece restituisce un puntatore ad una stringa che contiene l'indirizzo ottenuto elaborando la struttura puntata da `addr`.

Esistono anche delle costanti predefinite che possono essere indicate per il campo `s_addr` della struttura `struct in_addr`:

Sinossi:

```
unsigned long int INADDR_LOOPBACK
```

Descrizione:

Indica l'indirizzo IP `127.0.0.1`, per convenzione associato all'host locale.

Sinossi:

```
unsigned long int INADDR_ANY
```

Descrizione:

Sta ad indicare uno qualsiasi degli indirizzi IP definiti per l'host locale: ricordiamo infatti che un host può essere collegato a più reti, e quindi avere più indirizzi IP (v. figura 1.8).

6.2 Il protocollo

Dato il genere di comunicazione e lo spazio e dei nomi, la scelta del protocollo è conseguente: ogni standard offre al più un protocollo per una certa astrazio-

ne di comunicazione. Poiché ci limitiamo ai protocolli Internet, utilizzando il genere di comunicazione `SOCK_STREAM` avremo per default il protocollo `TCP`, mentre utilizzando `SOCK_DGRAM` avremo per default il protocollo `UDP`.

Comunque la libreria GNU C offre la possibilità per indicare, oltre al genere di comunicazione ed allo spazio dei nomi, anche il protocollo: se questo non viene indicato (ovvero viene specificata al suo posto la costante `NULL`), viene assegnato il protocollo conseguente le altre due scelte.

6.3 `socket()` – La creazione del socket

Il funzionamento del socket è fortemente asimmetrico, anche se consente comunicazioni bidirezionali: possiamo distinguere un terminale che attende che si stabilisca la comunicazione, ed uno che interviene per attivarla. Il primo è detto *server*: è permanentemente in attesa che arrivi una richiesta per il servizio che offre. Il secondo è detto *client* e scompare dalla scena dopo avere utilizzato il servizio.

Vedremo prima le funzioni utili a costruire un socket da parte del *server*, poi quelle utili al *client* per accedere al socket costruito dal server.

La creazione di un socket avviene specificandone il genere, lo spazio dei nomi, ed il protocollo:

Sinossi:

```
int socket(int namespace, int style, int protocol);
```

Descrizione:

Abbiamo già visto le costanti definite per i primi due parametri: la prima corrisponde allo *spazio dei nomi*, la seconda al *genere di comunicazione*. Assegnando NULL al terzo otteniamo l'assegnazione del protocollo di default.
La funzione restituisce il *file descriptor* relativo al socket, oppure -1 in caso di errore.

Per chiudere il socket utilizziamo la funzione `close()`, specificando come parametro il *file descriptor* del socket.

6.4 `bind()` – Assegnamento di un nome ad un socket

Abbiamo visto che ad un socket deve essere associata una porta e un indirizzo locali. A questo serve la seguente funzione:

Sinossi:

```
int bind(int socket, struct sockaddr *addr, size_t length);
```

Descrizione:

Il primo parametro indica il *file descriptor* che identifica il socket, ottenuto dall'invocazione della funzione `socket()`, mentre il secondo descrive l'indirizzo del socket, che abbiamo descritto nel paragrafo 6.1.4. Il terzo indica la lunghezza di tale descrizione.

A questo punto la gestione di un socket TCP e di uno UDP si differenziano: per quanto riguarda il socket UDP, questo è già utilizzabile per scambiare informazioni utilizzando le funzioni di `sendto()` e `rcvfrom()`. Invece il socket TCP necessita l'apertura della connessione, con l'esecuzione del protocollo di *three-way handshake*.

6.5 `listen()` – Apertura passiva della connessione

Quando il socket TCP è creato e collegato al socket partner, il processo si pone in ascolto: quindi si prepara ad eseguire il three-way handshake in modalità passiva. Questa operazione viene svolta dalla seguente funzione:

Sinossi:

```
int listen(int socket, unsigned int n)
```

Descrizione:

dove il primo parametro indica il socket da abilitare, mentre il secondo indica quante richieste possono essere lasciate in coda, mentre una viene servita, senza essere rifiutate.

6.6 `connect()` – Apertura attiva della connessione

Per aprire una connessione TCP, la libreria GNU C mette a disposizione la funzione `connect()`, che sostituisce la `bind()` e la `listen()` sul lato *client*:

Sinossi:

```
int connect(int socket, struct sockaddr *addr,
            size_t length)
```

Descrizione:

Come per la `bind()`, è necessario specificare il descrittore del file relativo al socket, l'indirizzo del socket e la lunghezza di tale descrizione.

Quando viene invocata, la funzione innesca il protocollo *three-way handshake* con il server.

6.7 Esempio – Una libreria per la gestione semplificata dei socket

In figura 6.1 vediamo una semplice libreria che contiene due funzioni che aprono un socket. Sono necessarie due funzioni, visto che il *client* apre il socket in un modo molto differente da un *server*: il primo richiede il servizio ed attende il risultato, mentre il secondo si predispone ad attendere una richiesta di servizio.

La funzione del lato *server* è la `make_socket()`. Viene prima creato un socket adatto al protocollo TCP, che viene poi associato con la `bind` ad una porta e ad un indirizzo locale.

Le tre istruzioni che impostano il contenuto della struttura `name` passata alla `bind()` descrivono infatti un indirizzo Internet (`AF_INET`), corrispondente a tutti gli indirizzi IP del nodo locale (`INADDR_ANY`), sulla porta che viene passata come parametro alla funzione `make_socket` (`port`).

L'unico parametro necessario alla nostra funzione `make_socket()` è quindi la porta su cui si intende rendere accessibile il servizio.

Dal lato *client* la funzione è la `connect_to_socket()`. Il socket viene creato come sul server, quindi viene costruita la struttura `name` associata al socket remoto: in questo caso la variabile `name` contiene l'indirizzo e la porta del *server* con cui si vuole stabilire la connessione.

La struttura corrispondente all'indirizzo del *socket* del *server* viene infine passata come parametro alla `connect()`, insieme alla sua lunghezza ed all'identificatore del socket locale.

```
#include <sys/socket.h>      /* per socket, bind, ecc */
#include <netinet/in.h>      /* per INADDR_ANY */
#include <arpa/inet.h>       /* per inet_aton */
#include "lisocket.h"

int
make_socket (int port)
{
  int sock;
  struct sockaddr_in indirizzo;

  /* Creazione del socket locale */
  sock = socket (PF_INET, SOCK_STREAM, 0);
  if (sock < 0)
    return -1;
  /* Creazione dell'indirizzo Internet locale */
  indirizzo.sin_family = AF_INET;
  indirizzo.sin_addr.s_addr = htonl (INADDR_ANY);
  indirizzo.sin_port = htons (port);
  /* Associazione del socket all'indirizzo Internet locale */
  if (bind (sock, (struct sockaddr *) &indirizzo, \
      sizeof (indirizzo)) < 0)
    return -1;
  /* Configurazione della coda delle richieste di connessione */
  if (listen (sock, 1) < 0)
    return -1;
  return sock;
}
```

Figura 6.1 Programma lisocket.c (cont.)

```
int
connect_to_socket (int port, char *hostname)
{
  int sock;
  struct sockaddr_in indirizzo;

  /* creazione del socket locale */
  sock = socket (PF_INET, SOCK_STREAM, 0);
  if (sock < 0)
    return -1;
  /* creazione dell'indirizzo Internet del socket remoto */
  indirizzo.sin_family = AF_INET;
  if (inet_aton (hostname, &indirizzo.sin_addr) == 0)
    return -1;
  indirizzo.sin_port = htons (port);
  /* connessione del socket locale al socket remoto */
  if (connect (sock, (struct sockaddr *) &indirizzo, \
      sizeof (indirizzo)) < 0)
    return -1;
  return sock;
}
```

Figura 6.1 Programma lisocket.c

6.8 Esempio – Apertura passiva di una connessione

L'esempio in figura 6.2 crea un socket predisposto per una connessione TCP passiva, utilizzando la funzione definita dalla nostra libreria, e subito lo distrugge: non sappiamo ancora come utilizzarlo...

```
#include <stdio.h>
#include <unistd.h> /* per la close */
#include "lisocket.h"

#define PORT 1200            /* questa e' una porta "libera" */

int
main ()
{
  int socket_mio;

  socket_mio = make_socket (PORT);
  if (socket_mio < 0)
    {
      perror ("socket");
      return -1;
    }
  puts ("A posto!");
  close(socket_mio);
  return 0;
}
```

Figura 6.2 Programma sock1.c

6.9 Esempio – Apertura attiva di una connessione

Il programma in figura 6.3 illustra invece una apertura attiva: viene aperta una connessione con un host generico (potete sostituire all'indirizzo nella costante `HOST` un indirizzo da voi raggiungibile), sulla porta 13. Se il nodo `HOST` non è particolarmente sospettoso, dovrebbe restituirvi il valore del suo clock di sistema.

La lettura dei dati provenienti dal *server* viene realizzata con una `read()`, a cui viene passato come *file descriptor* da cui leggere la risposta l'identificatore del socket.

Si presume, per semplicità, che tutta l'informazione arrivi in una singola

```
#include <stdio.h>
#include <unistd.h> /* per la read e la close */
#include "lisocket.h"

#define PORT 13              /* servizio daytime */
#define IPADDR "127.0.0.1"   /* sostituire un host noto */

int
main ()
{
  int socket_remoto;
  int caratteri;
  char buffer[256];

  /* Creazione del socket ed apertura della connessione */
  socket_remoto = connect_to_socket (PORT, IPADDR);
  if (socket_remoto < 0)
    {
      perror ("socket");
      return -1;
    }

  /* Lettura dei caratteri nel buffer di ricezione */
  caratteri = read (socket_remoto, buffer, sizeof (buffer));
  if (caratteri <= 0)
    {
      perror ("read");
      return -1;
    }

  /* Stampa dei caratteri ricevuti */
  buffer[caratteri] = '\0';
  puts (buffer);

  /* Chiusura del socket */
  close (socket_remoto);
  return 0;
}
```

Figura 6.3 Programma sock2.c

`read()`: sappiamo che questo è in genere sbagliato, e qui è giustificato solo dalla natura didattica del programma. La scelta più opportuna, in questo caso, sarebbe di utilizzare il protocollo UDP, non TCP: infatti si richiede l'invio di un singolo pacchetto contenente l'ora.

6.10 `accept()` – Il server accetta una connessione

Abbiamo visto nel paragrafo 6.5 come il *server* crea il *socket* dal proprio lato, per essere in grado di accettare connessioni da eventuali clienti.

Ora vediamo come fa per mettersi in attesa sul socket, cioè per mettersi in *attesa passiva* di una richiesta di un cliente.

La funzione destinata a questo è la `accept()`: attende una richiesta di connessione, e quando questa arriva comunica al server gli estremi di un *nuovo socket* su cui interagire con il cliente.

Tramite la `accept()` viene creata una connessione, quella specie di canale di comunicazione che collega i due processi corrispondenti e che è tipica del protocollo `TCP`. Quindi la funzione `accept` non può essere utilizzata su socket UDP.

Sinossi:

```
int accept(int socket, struct sockaddr *addr,
           size_t *length_ptr)
```

Descrizione:

Questa funzione è utilizzata per accettare una richiesta di connessione sul socket server *socket*. La funzione `accept` attende, se non ci sono connessioni in attesa.

I parametri `addr` e `lenth_ptr` sono utilizzati per restituire il nome del socket cliente che ha richiesto la connessione.

L'accettazione della connessione non rende il socket `socket` parte della successiva comunicazione. Invece viene costruito automaticamente un nuovo socket su cui si svolgerà la comunicazione. Il valore restituito dalla `accept`, a meno di errori, è il descrittore che corrisponde al nuovo socket.

Dopo l'esecuzione della `accept` il socket `socket` resta aperto e collegato: quindi è in grado di ricevere nuove connessioni sino a che non

viene chiuso. Può essere ancora utilizzando chiamando nuovamente una `accept`.
In caso di errore, la `accept` restituisce -1.

6.11 Esempio – Un server che accetta una connessione

```
#include <stdio.h>
#include <sys/socket.h>      /* per la accept */
#include <netinet/in.h>      /* per sockaddr_in */
#include <unistd.h>          /* per la write e la close*/
#include "lisocket.h"

#define PORT 1200            /* questa e' una porta "libera" */

int
main ()
{
  int socket_richiesta, socket_servizio;
  char stringa1[] = "prova di t";
  char stringa2[] = "rasmissione";
  struct sockaddr_in adCliente;
  int l, caratteri;

  /* Creazione del socket */
  socket_richiesta = make_socket (PORT);
  if (socket_richiesta < 0)
    {
      perror ("socket");
      return -1;
    }
  puts ("A posto!");
```

Figura 6.4 Programma sock3.c (cont.)

```
/* Loop indefinito di offerta di servizio */
while ( 1 ) {

  /* Attesa di una richiesta di connessione
     da parte di un cliente */
  socket_servizio = accept ( socket_richiesta,
                             (struct sockaddr *) &adCliente,
                             (unsigned int *) &l);
  if (socket_servizio < 0)
    {
      perror ("accept");
      return -1;
    }

  /* Invio della prima parte della stringa di prova */
  caratteri=write (socket_servizio, stringa1,
                   strlen (stringa1));
  if (caratteri != strlen(stringa1))
    {
      fprintf(stderr,"write failed\n");
      return -1;
    }
  printf("Spediti %d caratteri\n",caratteri);

  /* Ritardo (v. testo) */
  /* sleep(2); */

  /* Invio della seconda parte della stringa di prova */
  caratteri=write (socket_servizio, stringa2,
                   strlen (stringa2));
  if (caratteri != strlen(stringa2))
    {
      fprintf(stderr,"write failed\n");
      return -1;
    }
  printf("Spediti %d caratteri\n\n",caratteri);
```

Figura 6.4 Programma sock3.c (cont.)

```
    /* Chiusura del socket del servizio */
    close (socket_servizio);
  }

  /* Il processo non esce mai dal loop */
}
```

Figura 6.4 Programma sock3.c

In figura 6.4 ritroviamo il programma di figura 6.2, modificato per introdurre, all'atto della connessione, la spedizione di una stringa di prova. La struttura del programma ora presenta il *loop infinito* che caratterizza un *server*.

Anche in questo caso, abbiamo utilizzato TCP per finalità didattiche, visto che volevamo esemplificare una connessione TCP, ma per la funzionalità svolta dal server sarebbe stato più opportuno il protocollo UDP.

Per rendere più evidenti gli effetti del *buffer* di ricezione, abbiamo spezzato la spedizione della stringa di prova in due `write()` successive: le due stringhe verranno quindi ad occupare due segmenti separati. Discuteremo tra breve l'effetto di questo accorgimento.

6.12 Esempio – Un cliente che richiede una connessione

In figura 6.5 vediamo un cliente per il server di figura 6.4: rispetto al cliente visto nel programma 6.3 è stata modificata la costante `IPADDR`, e sono state aggiunte le istruzioni per ricevere e visualizzare i caratteri inviati dal server.

Avviando prima il *server* del programma `sock3.c`, e quindi il *client* `sock4.c`, osserviamo che viene eseguita solo una operazione di `read()`, e che questa viene ritardata rispetto alla costruzione della connessione con una chiamata alla funzione di `sleep()` che arresta l'esecuzione per un secondo.

Per effetto del ritardo ci sarà tempo per spedire, recapitare ed inserire nel buffer di ricezione del *client* il payload di ambedue i segmenti inviati dal *server* illustrato sopra: quindi la `read()` del *client* preleverà con un'unica operazione ambedue i payload, e li visualizzerà come un'unica stringa.

Abilitando l'istruzione `sleep(2)` tra le due `send()` del *server* (ora commentata) la situazione si capovolgerà: ora il *server* non sarà più in grado di spedire i due segmenti prima della `read()` del *client*, quindi visualizzerà solo il primo dei due payload, ed il secondo andrà semplicemente perduto.

```
#include <stdio.h>
#include <unistd.h>
#include "lisocket.h"

#define PORT 1200            /* Una porta libera */
#define IPADDR "127.0.0.1"     /* Indirizzo di loopback */

int
main ()
{
  int socket_remoto;
  int caratteri;
  char buffer[256];

  /* Apertura del socket */
  socket_remoto = connect_to_socket (PORT, IPADDR);
  if (socket_remoto < 0)
    {
      perror ("socket");
      return -1;
    }
  /* Ritardo (v. testo) */
  sleep(1);
```

Figura 6.5 Programma sock4.c (cont.)

```
  /* Lettura dei caratteri dal buffer di ingresso */
  caratteri = read (socket_remoto, buffer, sizeof (buffer));
  if (caratteri <= 0)
    {
      fprintf (stderr, "read failed\n");
      return -1;
    }
  printf("Ricevuti %d caratteri\n",caratteri);

  /* Stampa dei caratteri ricevuti */
  buffer[caratteri] = '\0';
  puts (buffer);

  /* Chiusura del socket */
  close (socket_remoto);
  return 0;
}
```

Figura 6.5 Programma sock4.c

Vale la pena di sottolineare che la struttura dei due programmi è stata studiata per evidenziare le caratteristiche della comunicazione tramite socket TCP, ma non va presa come esempio di buona programmazione: il software che dipende dalla temporizzazione dei propri componenti per il proprio funzionamento è, nel migliore dei casi, fonte di infiniti grattacapi. Nel caso in esame, l'adozione del protocollo UDP invece di TCP, e la spedizione del messaggio in un unico datagramma darebbero garanzia di ricezione del testo di prova integro, con un programma ancora più semplice di quello presentato.

Esercizi

- Fate in modo che la comunicazione tra il client ed il server dei due programmi precedenti avvenga tra due computer distinti (ad esempio tra il vostro e quello del vostro vicino). Per conoscere l'indirizzo IP di un computer il comando è

  ```
  $ host <nomehost>
  ```

  Ad esempio, per conoscere l'IP di host2:

  ```
  $ host host2
  ```

- Si tratta di scrivere un server collegato alla porta 1400 che invii al cliente che gli si connette la data attuale. Questa viene ottenuta effettuando una connessione con la porta 13 del nodo xxx.xxx.xxx.xxx (sostituite l'indirizzo IP di un nodo disponibile). L'idea è quella di realizzare un semplice "proxy".

- Modificate la libreria `lisocket` per aprire socket UDP, quindi riscrivete i due programmi sock3.c e sock4.c per UDP.

Capitolo 7

SNMP – Simple Network Management Protocol

Si tratta di un protocollo [7] semplice nella natura, ma complesso nelle funzioni che può realizzare: il suo scopo è consentire il controllo e la configurazione dei nodi di una rete da parte di un nodo di controllo. La presenza del modulo SNMP su un nodo è comunque opzionale.

L'architettura di SNMP vede la rete come contenente un certo numero di nodi *controllori*: il protocollo SNMP consente loro di osservare e modificare il funzionamento degli altri nodi. Il modulo SNMP dei nodi *controllati* dovrà essere in grado di realizzare le istruzioni prodotte dai nodi controllori.

Una caratteristica importante dell'architettura di SNMP è lo sbilanciamento del carico tra nodo controllato e nodo controllore: al primo vengono richieste operazioni estremamente semplici, mentre il controllo e la coordinazione di queste operazioni può essere arbitrariamente complesso, ma spostato sul nodo di controllo.

Poiché si richiede ad SNMP una estrema flessibilità, il protocollo è stato progettato per fornire solamente due operazioni: la lettura (`GET`) e la scrittura (`SET`) dei parametri di funzionamento di un nodo.

Poiché la comunicazione in SNMP consiste in uno scambio di messaggi contenenti le operazioni da eseguire e le relative risposte, il protocollo SNMP è tradizionalmente realizzato sopra UDP. Dato il tipo di prestazioni attese, una caratteristica importante delle operazioni controllate da questo protocollo è la *idempotenza*: possono essere ripetute più volte senza avere effetti cumulativi, e l'ordine di esecuzione non modifica l'effetto finale. Inoltre la perdita di un messaggio SNMP non comporta gravi conseguenze.

La descrizione (formale) dei parametri di funzionamento è realizzata attraverso dei documenti cartacei che definiscono la *Management Information Base* (MIB), il complesso delle variabili che influenzano il comportamento del nodo.

7.1 Una notazione standard per i dati

Uno dei problemi fondamentali per l'interconnessione di una rete è l'interoperabilità, cioè la caratteristica che il protocollo deve avere per consentire la collaborazione di macchine basate su piattaforme hardware e software diverse.

Un esempio eclatante di questo problema è nella notazione degli interi. Infatti due architetture hardware estremamente diffuse rappresentano gli interi di 2 o più ottetti in modo diametralmente opposto: in un caso (Intel/DOS) gli ottetti con indirizzo inferiore sono i meno rappresentativi, nell'altro (Motorola/Apple) lo sono quelli di indirizzo inferiore. Si parla nel primo caso di architetture *little endian*, nel secondo *big endian* (curiosa e indicativa la provenienza dei termini, che trovate in [26]).

La soluzione di Internet consiste nel trasferire tutti i dati in forma numerica (come abbiamo visto studiando gli header dei principali protocolli): in questo modo si limita il problema alla codifica degli interi nella forma *big endian/little endian* corretta. La modalità *di rete* (ovvero il "Network Byte Order") viene poi fissata in *big endian*: quindi un nodo *little endian* dovrà invertire gli interi di due o più ottetti (ad esempio, la lunghezza del datagramma in IP) prima di inserirli nell'intestazione. Per questa operazione vengono fornite da UNIX due funzioni di sistema, `hton()` e `ntoh()` che hanno lo scopo rispettivamente di tradurre un dato nella modalità locale a quella di rete e viceversa. In questo modo tutti i formati per le intestazioni definiti precedentemente acquistano un valore indipendente dall'architettura, ed ogni programma che debba interoperare con altri di architettura non nota deve utilizzare opportunamente le due funzioni.

Tuttavia la soluzione di Internet funziona solo quando i dati si limitino ad interi: non è applicabile a situazioni in cui i dati abbiano una forma più complessa o strutturata. L'*Abstract Syntax Notation - 1* (ASN-1) è la soluzione di ISO-OSI al problema di garantire l'interoperabilità: tutti i dati trasferiti vengono identificati con un tipo.

L'identificazione del tipo consente ad ogni nodo di *ricostruire* il dato se-

```
ipDefaultTTL OBJECT-TYPE
    SYNTAX      INTEGER (1..255)
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
            "The default value inserted into the Time-To-Live
            field of the IP header of datagrams originated at
            this entity, whenever a TTL value is not supplied
            by the transport layer protocol."
    ::= { ip 2 }
```

Figura 7.1 La rappresentazione MIB del *time to live* di default

condo le proprie regole: una coppia di funzioni provvedono a ricostruire la forma interna un dato proveniente dalla rete, ed a codificare nella forma *di rete* un dato nella rappresentazione interna.

La ASN-1 (ancora opportunamente semplificata) trova comunque applicazione in Internet: viene utilizzata per descrivere in maniera standard i parametri che condizionano il funzionamento, ed SNMP la utilizza nelle sue operazioni di `SET` e `GET`. Le informazioni destinate a configurare il funzionamento di un host vengono indicate collettivamente MIB.

La sintassi (originariamente estesa a tipi aggregati, come gli insiemi) è stata limitata ai tipi più semplici: vediamola attraverso alcuni esempi, tratti da [21].

In figura 7.1 vediamo la descrizione ASN-1 del valore default del TTL inserito da un nodo nei propri pacchetti IP: si tratta di un semplice intero. L'identificatore del *tipo* associato dato compare in fondo: nel caso dell'esempio è `ip 2`.

Nella figura 7.2 vediamo invece un dato di tipo più complesso: si tratta della tabella degli indirizzi IP associati ad un nodo. In questo caso viene prima definita la struttura `ipAddrTable`, come sequenza di descrizioni di indirizzi, e poi viene definita la singola voce nella sequenza `IpAddrEntry`.

```
ipAddrTable OBJECT-TYPE
    SYNTAX SEQUENCE OF IpAddrEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
           "The table of addressing information relevant
           to this entity's IP addresses."
    ::= { ip 20 }

IpAddrEntry ::= SEQUENCE {
        ipAdEntAddr          IpAddress,
        ipAdEntIfIndex       INTEGER,
        ipAdEntNetMask       IpAddress,
        ipAdEntBcastAddr     INTEGER,
        ipAdEntReasmMaxSize  INTEGER
    }
```

Figura 7.2 La rappresentazione MIB degli indirizzi Internet associati ad un certo nodo

7.2 Il protocollo SNMP

Per definire il funzionamento di SNMP è necessario definire alcuni concetti fondamentali del suo funzionamento. Questi ruotano attorno all'esistenza di un certo numero di *entità* che sono coinvolte nell'azione di controllo, ed in un patrimonio di *informazioni di controllo* che determinano l'azione di controllo.

Quindi nel seguito definiamo:

- il genere di informazioni di controllo che possono essere comunicate attraverso SNMP,
- come queste informazioni vengono rappresentate nel protocollo,
- quali siano le operazioni che possono essere realizzate attraverso il protocollo su tali informazioni,
- come avviene lo scambio di informazioni tra le entità coinvolte,
- che relazioni sussistono tra tali entità, e infine

- come vengono identificate le informazioni di controllo,

Il *genere di informazioni di controllo* che può essere comunicato coincide con le informazioni codificate nella Management Information Base (MIB). Le informazioni di controllo vengono indicate anche come *oggetti*, cui viene associato un *tipo*. Le informazioni di controllo per i nodi Internet sono definite nell'RFC [21].

La *rappresentazione delle informazioni di controllo* viene definita in uno standard esterno a SNMP: consiste in regole per tradurre una certa istanza di un oggetto in un dato trasferibile in rete.

Le *operazioni* che possono essere effettuate sulle *informazioni di controllo* si risolvono nell'assegnamento di un valore ad un oggetto, o nella lettura del valore di un oggetto.

Questa caratteristica ha l'effetto di rendere il protocollo estremamente semplice e flessibile la parte di chi deve eseguire le operazioni (cioè l'entità controllata), mentre l'entità di controllo può applicare politiche arbitrariamente complesse ed in evoluzione.

Le *modalità di scambio* delle informazioni verranno descritte nel seguito, descrivendo il protocollo. Qui si sottolinea solo che si presume che questo poggi sul servizio UDP: non solo per l'esiguità delle risorse che richiede, in linea con l'essenzialità del protocollo, ma anche perché è quello più adatto a supportare comunicazioni non orientate alla connessione, come possono essere la lettura o l'impostazione di un parametro, o la spedizione di uno stesso messaggio a tutti i nodi sulla stessa rete.

Le *relazioni tra le entità* chiariscono quali hanno il ruolo di controllo, e quali vengono controllate: nel complesso si parla di una *comunità* SNMP per un insieme di entità che sono in grado di controllare un *agente SNMP*. L'agente riconosce come *autentici* i messaggi prodotti da entità di quella comunità. L'identificazione di un messaggio come autentico è problema rilevante.

Ciascun agente è caratterizzato da un certo numero di oggetti, per i quali, rispetto ad una certa *comunità*, sono definiti dei diritti di accesso: per il genere di operazioni ammesse, questi sono limitati alla **lettura** ed alla **scrittura**. Per la comunità di cui l'agente stesso fa parte, questi vengono detti collettivamente *politica di accesso*.

L'*identificazione delle informazioni* di controllo viene utilizzata la codifica fornita dalle MIB, ed illustrata sopra nelle linee essenziali. Nel seguito fare-

```
Message ::= SEQUENCE {
                        version    INTEGER,
                        community  OCTET STRING,
                        data       PDUs,
                      }
```

Figura 7.3 Struttura di messaggio SNMP

mo esplicito riferimento a quella sintassi, ma focalizziamo sulle potenzialità offerte.

7.3 Specifica del protocollo

Il protocollo di SNMP si appoggia su UDP: ogni **messaggio** di SNMP viene incapsulato in un singolo datagramma UDP. La codifica del messaggio è quella specificata nella ASN-1: quindi le descrizioni che seguono non hanno la caratteristica rappresentazione come sequenze di ottetti allineate ai quattro ottetti, ma sono più vicine alle descrizioni di una struttura C.

Il protocollo SNMP riceve dalla porta 161 (con il protocollo UDP) tutti i messaggi eccetto quelli che invocano una **trap**, che vengono ricevuti dalla porta 162. La forma del messaggio è quella visibile in figura 7.3: è necessario specificare a che versione di SNMP si fa riferimento, la *comunità* cui si riferisce il messaggio. Seguono i dati, nella forma di PDU.

Le PDU che devono essere necessariamente incluse in una realizzazione del protocollo SNMP sono di 5 tipi, e sono descritte in ASN-1 in figura 7.4. Vedremo nel seguito le caratteristiche di ogni tipo di messaggio.

I passi per la generazione di un messaggio, comuni ad ogni tipo di messaggio, sono i seguenti:

1. Costruzione del `Message` da incapsulare nel messaggio.
2. Invocazione del servizio di **autenticazione** – Dati il `Message`, l'identificazione di mittente, il destinatario e la comunità, si costruisce un messaggio che il destinatario riconoscerà come autentico, per esempio crittografando il messaggio originario.
3. Costruzione del messaggio, completo di numero di versione e comunità.

```
PDUs ::=  CHOICE {
                   get-request      GetRequest-PDU,
                   get-next-request GetNextRequest-PDU,
                   get-response     GetResponse-PDU,
                   set-request      SetRequest-PDU,
                   trap             Trap-PDU
                  }
```

Figura 7.4 Struttura di messaggio SNMP

4. **Serializzazione** del `Message` così generato – La serializzazione si ottiene traducendo le PDU dal formato locale al formato di rete.

Il messaggio così generato verrà consegnato al modulo UDP per essere recapitato al destinatario. D'altro canto, il ricevente eseguirà le azioni seguenti:

1. Una prima **analisi** del `Message` – Ha lo scopo di ricostruire la struttura originaria e di verificare che il numero di versione corrisponda ad una versione compatibile.

2. Autenticazione del `Message` – Se l'autenticazione ha successo il messaggio viene pure decrittato, altrimenti il protocollo deve sospendere l'elaborazione del messaggio, e può registrarne la mancata autenticazione e spedire una *trap* al mittente.

3. Estrazione della PDU dal `Message`.

4. Trattamento, dopo ulteriori verifiche, della PDU – L'elaborazione dalle PDU dipenderà dal suo tipo.

5. Se il passo precedente dà luogo ad un messaggio di risposta, questo viene spedito al mittente della PDU appena elaborata.

Vediamo ora i diversi tipi di PDU che possono essere incluse nel messaggio. Per definirle, è necessario definire prima alcuni tipi, che verranno poi utili nella definizione delle PDU.

```
RequestID ::=   INTEGER
ErrorStatus ::= INTEGER  {
                          noError(0),
                          tooBig(1),
                          noSuchName(2),
                          badValue(3),
                          readOnly(4)
                          genErr(5)
                         }
ErrorIndex ::=  INTEGER
VarBind ::=     SEQUENCE {
                          name  ObjectName,
                          value ObjectSyntax
                         }
VarBindList ::= SEQUENCE OF VarBind
```

Figura 7.5 Alcuni tipi ausiliari

`RequestId` È un tipo di PDU che serve a identificare una richiesta, per poter poi riconoscere la risposta.

`ErrorStatus` In questa PDU viene restituito, in caso di errore, lo stato prodotto dall'errore.

`ErrorIndex` Può servire a localizzare l'errore.

`VarBind` La PDU specifica una associazione tra variabile e valore nel modo che vedremo.

`VarBindList` La PDU specifica una sequenza di associazioni variabile-valore.

Il contenuto dei diversi tipi di PDU sono descritti in figura 7.5. Vediamo ora le diverse PDU.

7.3.1 GetRequest-PDU e GetResponse-PDU

Per la `GetRequest` si tratta della richiesta dei valori assegnati alle variabili indicate nell'ultimo parametro (v. figura 7.6). I valori attinenti agli errori `error-status` ed `error-index` sono impostati a 0.

```
GetRequest-PDU ::=
     [0]
     IMPLICIT SEQUENCE {
                          request-id        RequestID,
                          error-status      ErrorStatus,
                          error-index       ErrorIndex,
                          variable-bindings VarBindList
                         }
```

Figura 7.6 Descrizione della PDU `GetRequest`

Il **destinatario**, alla ricezione del messaggio contenente la `GetRequest` decide il messaggio da restituire:

1. Se il nome indicato per una delle variabili non corrisponde ad una variabile locale, o se questa non è di tipo "non aggregato" (cioè corrisponde ad una tabella o altro dato complesso), restituisce la stessa PDU, ma descritta come `GetResponse`, con la segnalazione di errore `noSuchValue`.

2. Se il messaggio è troppo lungo per essere elaborato, risponde come sopra, ma indicando `tooBig`.

3. Se per un'altra ragione il messaggio non viene elaborato, risponde come sopra, ma indicando `genErr`.

4. Se non si verifica nessuna delle situazione descritte sopra, il messaggio viene elaborato ed il campo valore di ciascun `VarBind` viene impostato con il valore opportuno.

Il messaggio restituito è del tipo `GetResponse` in figura 7.7.

Alla esecuzione del `GetResponse` il contenuto delle variabili viene riportato al livello superiore.

7.3.2 GetNextRequest-PDU

Bisogna premettere che le variabili semplici (quelle non aggregate) sono accessibili successivamente secondo l'ordine lessicografico della loro rappresentazione. D'altra parte, il modo in cui vengono definiti gli identificatori garantisce che le voci di una tabella (un tipo aggregato) sono tutte adiacenti

```
GetResponse-PDU ::=
     [2]
     IMPLICIT SEQUENCE {
                          request-id        RequestID,
                          error-status      ErrorStatus,
                          error-index       ErrorIndex,
                          variable-bindings VarBindList
                         }
```

Figura 7.7 Descrizione della PDU `GetResponse`

```
GetNextRequest-PDU ::=
     [1]
     IMPLICIT SEQUENCE {
                          request-id        RequestID,
                          error-status      ErrorStatus,
                          error-index       ErrorIndex,
                          variable-bindings VarBindList
                         }
```

Figura 7.8 Descrizione della PDU `GetNextRequest`

nell'ordinamento lessicografico. La `GetNext` utilizza questa proprietà per fornire la possibilità di scandire una struttura aggregata.

Il tipo, definito in figura 7.8 è identico alla `GetRequest`, salvo per l'identificatore del tipo.

Ciascun `VarBind` porterà tanto l'identificatore dell'oggetto quanto (eventualmente) uno dei suoi valori, ed il `VarBind` restituito dal **destinatario** corrisponderà al successivo, nello stato del destinatario, rispetto a quello inviato. La generazione di errori verrà modificata di conseguenza: l'errore `noSuchName` verrà inviato nel caso in cui non esista un successore del `VarBind` indicato.

Il messaggio restituito è del tipo `GetResponse` in figura 7.7.

7.3.3 Esempio di scansione di una tabella

Prendiamo ad esempio la tabella di routing in tabella 7.1: ciascuna riga rappresenta una `SEQUENCE`.

Lo scambio di messaggi tra una applicazione SNMP che scandisce la tabella ed un elemento della rete che risponde potrebbe quello raffigurato in figura 7.9.

Tabella 7.1 Una tabella di routing

IpRouteDest	*IpRouteNextHop*	*IpMetric1*
`9.1.2.3`	`99.0.0.3`	3
`10.0.0.51`	`89.1.1.42`	5
`10.0.0.99`	`89.1.1.42`	5

7.4 SetRequest

La descrizione è analoga alle precedenti, salvo per l'identificatore (v. figura 7.10).

Viene restituito un errore se per una delle variabili indicate si verifica uno dei seguenti casi (tra parentesi il tipo di evento segnalato):

- la variabile indicata non può essere alterata (o non esiste) (`noSuchValue`),
- il valore indicato non è consistente con la variabile (`badValue`),
- la dimensione del messaggio è eccessiva (`tooBig`), oppure
- in qualunque altro caso l'operazione di assegnamento non abbia successo (`genErr`).

Altrimenti, i valori indicati vengono assegnati alle corrispondenti variabili.

Il protocollo specifica che l'operazione deve essere **atomica**: quindi tutti gli assegnamenti devono essere eseguiti in modo apparentemente simultaneo. Per questo è necessario organizzare un opportuno blocco delle variabili interessate.

```
GetNextRequest( ipRouteDest,
                ipRouteNextHop,
                ipRouteMetric1 )

   GetResponse (( ipRouteDest.9.1.2.3 =  "9.1.2.3" ),
                ( ipRouteNextHop.9.1.2.3 = "99.0.0.3" ),
                ( ipRouteMetric1.9.1.2.3 = 3 )

GetNextRequest ( ipRouteDest.9.1.2.3,
                 ipRouteNextHop.9.1.2.3,
                 ipRouteMetric1.9.1.2.3 )

   GetResponse (( ipRouteDest.10.0.0.51 =  "10.0.0.51" ),
                ( ipRouteNextHop.10.0.0.51 = "89.1.1.42" ),
                ( ipRouteMetric1.10.0.0.51 = 5 )

GetNextRequest ( ipRouteDest.10.0.0.51 ,
                 ipRouteNextHop.10.0.0.51 ,
                 ipRouteMetric1.10.0.0.51   )

   GetResponse (( ipRouteDest.10.0.0.99 =  "10.0.0.51" ),
                ( ipRouteNextHop.10.0.0.99 = "89.1.1.42" ),
                ( ipRouteMetric1.10.0.0.99 = 5 )

GetNextRequest ( ipRouteDest.10.0.0.99 ,
                 ipRouteNextHop.10.0.0.99 ,
                 ipRouteMetric1.10.0.0.99   )

   GetResponse ( noSuchName )
```

Figura 7.9 Scambio di messaggi per la scansione di una tabella

7.5 Trap

Una trap è una forma di segnalazione asincrona. Il tipo di PDU corrispondente è descritto in figura 7.11.

Il ruolo della `trap` è dunque quello di informare di eventi asincroni. I

```
SetRequest-PDU ::=
     [3]
     IMPLICIT SEQUENCE {
                          request-id        RequestID,
                          error-status      ErrorStatus,
                          error-index       ErrorIndex,
                          variable-bindings VarBindList
                         }
```

Figura 7.10 Descrizione della PDU `SetRequest`

```
Trap-PDU ::=
         [4]
         IMPLICIT SEQUENCE {
                 enterprise       OBJECT IDENTIFIER,
                 agent-addr       NetworkAddress,
                 generic-trap INTEGER {
                                 coldStart(0),
                                 warmStart(1),
                                 linkDown(2),
                                 linkUp(3),
                                 authenticationFailure(4),
                                 egpNeighborLoss(5),
                                 enterpriseSpecific(6)
                                  },
                 specific-trap INTEGER,
                 time-stamp  TimeTicks,
                 variable-bindings VarBindList
          }
```

Figura 7.11 Descrizione della PDU Trap

primi due campi descrivono la provenienza della `Trap`, mentre il terzo indica il genere di evento che viene segnalato.

Sommariamente, i tipi di evento generici hanno il seguente significato:

`coldStart` Il mittente riparte ma la configurazione precedente è andata perduta.

`warmStart` Il mittente riparte e la configurazione precedente è stata conservata.

`linkDown` Il collegamento (del destinatario) si è interrotto. Il collegamento interrotto viene indicato tra le `VarBind` successive.

`linkUp` Il collegamento (del destinatario) è stato ripristinato. Il collegamento interrotto viene indicato tra le `VarBind` successive.

`authenticationFailure` Il mittente non ha autenticato il messaggio ricevuto.

`egpNeighborLoss` Il mittente ha perduto uno dei suoi "vicini" nel protocollo `Exterior Gateway Protocol`, destinato al controllo del routing.

`enterpriseSpecific` La definizione di questa trap dipende dall'amministrazione del sistema: il campo successivo (`specific-trap`) indica il tipo di trap.

Il campo `time-stamp` indica l'intervallo di tempo trascorso dall'ultima trap generata da quel mittente, e `variable-bindings` indica altre variabili utili a specificare la trap (per esempio nella `linkDown`).

Capitolo 8

RTP – Real Time Protocol

Il protocollo *Real Time Protocol* (RTP) è destinato a trasferire dati cui sono associate delle caratteristiche temporali (un *sincronismo*) che sono parte determinante dell'informazione trasferita.

Quindi il termine *real-time* non è associato alla qualità del servizio offerto dal protocollo (né potrebbe esserlo, visto che la sua definizione non fa ipotesi sui protocolli e sull'hardware sottostanti), ma piuttosto al fatto che il protocollo è adatto a trasportare informazioni cui viene associato anche un istante di generazione.

Gli esempi normalmente vengono tratti dalle applicazioni multimediali: trasferendo dati audio, può essere opportuno associare a ciascun dato l'istante in cui un certo suono viene generato. A questo modo, sarà poi possibile *miscelare* e sequenzializzare correttamente suoni provenienti da fonti diverse. Una circostanza che può rendere necessario questo genere di prestazioni è una teleconferenza.

Un'altra caratteristica di RTP è nella sua architettura software: infatti non viene concepito come un modulo chiuso a sé stante, con una interfaccia funzionale rispetto al resto del sistema, ma piuttosto come un insieme di funzionalità "aperte", componibili con altre per la creazione di un modulo completo. Questo genere di approccio è stato accettato da chi ha progettato `vic`, `vat` ed altri strumenti multimediali di rete che utilizzano RTP.

8.1 Caratteristiche generali

Il protocollo RTP si colloca al livello trasporto, e considera di avere a disposizione le funzioni offerte da UDP: quindi una comunicazione *connectionless* orientata ai *datagrammi*, completa di un controllo sul contenuto dei pacchetti, ottenuto attraverso una *checksum*, e di funzionalità di multiplexing alla sorgente ed alla destinazione, ottenute attraverso le *porte*.

Poiché il protocollo deve essere in grado di integrarsi in protocolli specifici, è necessario che la descrizione del contenuto informativo del pacchetto (il *payload* nella terminologia RTP) possa essere definito dinamicamente, e comunque rientrare nella parte descrittiva del pacchetto, lo *header*. Questa descrizione potrà essere non standard, ed essere specifica per l'applicazione in cui RTP viene integrato.

Le informazioni trasferite devono essere collocate rispetto a due distinte sequenze: la sequenza di spedizione, che deve essere in grado di riprodurre caratteristiche simili a quelle di un circuito virtuale in cui l'ordine di spedizione dei dati viene ricostruito all'arrivo, e la sequenza temporale, potenzialmente indipendente ed incoerente rispetto alla prima, e che tiene conto solo dell'istante in cui l'informazione è stata prodotta dalle sorgenti.

Infine, seppure il protocollo non fornisca garanzie circa la temporizzazione delle comunicazioni, nondimeno deve fornire il modo per poterle controllare. Al protocollo RTP è quindi affiancato un protocollo di controllo, RTCP, con intenzioni simili a ICMP: controllare, attraverso messaggi di eco e simili, che le prestazioni della rete siano quelle attese.

8.2 Il funzionamento del miscelatore

Il funzionamento di un host che operi da *miscelatore* illustra bene le potenzialità del protocollo RTP.

I flussi di informazione multimediali possono generalmente essere miscelati: non solo su domini diversi (ad esempio audio e video) ma anche sullo stesso dominio. Ad esempio più immagini possono essere presentate simultaneamente sullo schermo, o più suoni possono essere riprodotti simultaneamente attraverso un altoparlante.

Per ottenere questo è necessario che i diversi flussi siano sincronizzati, e che l'effetto della miscelazione sia di presentare insieme eventi effettiva-

mente simultanei! Altrimenti il significato della comunicazione può essere assolutamente falsato.

Utilizzando RTP è possibile miscelare flussi di informazione proveniente da fonti diverse: inoltre questa funzionalità può essere associata ad uno specifico host, che riceve i flussi provenienti dai diversi partecipanti ad una videoconferenza (ad esempio), e fornisce a ciascuno una riproduzione "miscelata" di ciò che avviene nei diversi siti coinvolti.

Questa soluzione consente anche di ridurre notevolmente il carico di rete, ciò che è sempre opportuno.

Un'altra funzione simile alla miscelazione è la *traduzione*. L'organizzazione della rete, statica o dinamica che sia, può rendere un protocollo di codifica dell'informazione più opportuno di altri su determinate connessioni.

Un host *traduttore* può trasformare il flusso informativo ricodificandolo nella forma più adatta. In questo caso, oltre alla necessaria informazione di sincronizzazione, sarà anche necessaria una descrizione del genere di codifica utilizzata per rappresentare il flusso informativo. Questa descrizione potrà essere specifica per l'applicazione, ma dovrà essere presente nella parte descrittiva di ogni singolo pacchetto ricevuto.

Questa funzione è associata, in RTP, ad un campo specifico nell'intestazione del pacchetto, che descrive la codifica del *payload*, l'informazione trasferita nel pacchetto.

8.3 L'intestazione dei pacchetti RTP

Possiamo vedere l'intestazione di un pacchetto RTP in figura 8.1.

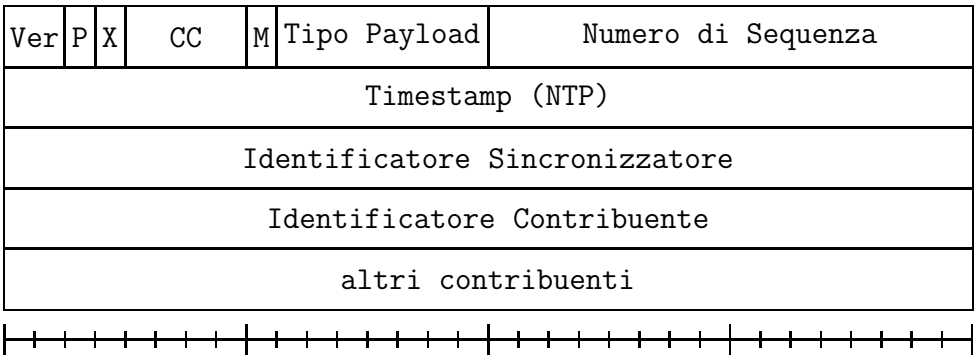


Figura 8.1 Intestazione di un pacchetto RTP

V Il campo indica il numero di versione, attualmente il 2.

P Il campo segnala la presenza di ottetti di *padding*, necessari a riallineare i dati al termine del pacchetto. Si rende necessario per alcuni algoritmi di crittazione.

X Il campo segnala che lo header RTP è esteso con informazioni di intestazione specifiche per l'applicazione in cui RTP è stato integrato. Come abbiamo visto RTP non è concepito come un protocollo a se stante, ma come un modulo che può essere *integrato* in una applicazione più complessa. Questo bit segnala che, oltre lo header RTP, è presente anche uno header per l'applicazione. La semplice organizzazione dell'estensione è illustrata nel paragrafo 5.3.1 di [22].

CC Il valore del campo corrisponde al numero di host che contribuiscono al flusso informativo trasportato dal pacchetto, fino ad un massimo di 16.

M È un marker generico, al quale l'applicazione specifica può attribuire un significato determinato. Anche questo dettaglio è in vista della *integrazione* del protocollo RTP con una applicazione.

Tipo Payload Il campo specifica il tipo di payload contenuto nella PDU. Alcuni codici sono descritti da uno standard, ma il significato di questo campo viene ancora lasciato all'applicazione, e quindi definito solo dopo l'*integrazione* di RTP nella applicazione.

Numero d'ordine Il numero d'ordine è riferito al flusso di dati, e viene incrementato di una unità ad ogni pacchetto spedito.

Timestamp Nel campo è indicato un valore numerico che corrisponde ad una misura dell'istante in cui il primo ottetto nel campo dati viene prodotto. La misurazione del tempo deve avere caratteristiche di crescita lineare nel tempo, ma non deve effettivamente aderire ad uno standard di rappresentazione del tempo: anche questa può dipendere dal tipo di carico utile. La granularità di rappresentazione del tempo deve essere tale da consentire la corretta sincronizzazione della comunicazione.

Identificatore Sincronizzatore Il campo contiene l'identificatore della sorgente che ha fornito la sincronizzazione per il flusso di informazione: nel caso l'informazione sia frutto di un'operazione di miscelazione, qui si tratterà del

miscelatore. Il numero è scelto a caso, ma le probabilità di collisione sono molto basse: quindi non coincide con un indirizzo Internet, e lo stesso host può essere sorgente di flussi diversi. Ad esempio, se lo stesso host genera audio e video, per ciascuno di questi sarà scelto un `SSRC` differente, ed i due flussi distinti. Questa soluzione è preferibile a miscelare audio e video nello stesso flusso, come descritto nel paragrafo 5.2 di [22].

Identificatore Contribuente Il campo contiene l'identificatore di una delle sorgenti che contribuiscono al flusso: infatti questo può essere frutto di una *miscelazione* di più flussi. La lista, la cui lunghezza è indicata nel campo `CC`, non può contenere più di 16 identificatori.

8.4 Il protocollo RTCP

Il protocollo RTCP serve a scambiare informazioni riguardanti il funzionamento di RTP. Il funzionamento di questo protocollo avviene su una porta distinta da quella utilizzata da RTP. Quindi il funzionamento di RTP comporta l'uso di due porte: una per RTP e l'altra per RTCP.

I messaggi di RTCP vengono sempre diffusi a *tutti* i partecipanti alla sessione: quindi diventa particolarmente interessante un supporto che consenta il multicast.

Lo scopo primario di RTCP è quello di consentire al mittente di conoscere la *reazione* del destinatario, in modo che il primo abbia informazioni sulla qualità della comunicazione. Infatti una funzione essenziale di un protocollo a livello trasporto è proprio il *controllo del flusso*, che si ottiene tramite questo genere di meccanismo, detto *feedback*. Un ruolo simile svolgeva anche ICMP.

Tra l'altro, la diffusione di questi messaggi rende possibile anche il controllo di determinati aspetti della comunicazione (ad esempio codifica o routing) da parte di *entità terze* (*third party* nella terminologia, anche legale, inglese), non direttamente interessate nelle informazioni scambiate, ma responsabili della comunicazione.

Un'altra funzione del protocollo RTCP consiste nell'associare agli identificatori delle sorgenti di sincronismo uno specifico *indirizzo al livello trasporto* (in Internet una coppia costituita da un indirizzo Internet e una porta). L'informazione può essere necessaria per collocare la sorgente di un flusso di informazioni nell'Internet, invece che nello spazio anonimo dei generatori di sincronismo.

Inoltre, il multicast dei messaggi di RTCP consente ad ogni partecipante di *conoscere il numero di partecipanti* alla sessione: questo può essere indispensabile, ad esempio, per moderare il numero di messaggi di controllo, che rischierebbero altrimenti di saturare la rete!

Una ulteriore funzionalità associata a RTCP consiste nello scambiare informazioni identificative dei partecipanti, collocandoli nel mondo fisico: nome, indirizzo di posta elettronica, collocazione geografica ecc.

Poiché i pacchetti RTCP non vengono spediti in risposta ad eventi specifici, ma periodicamente, è possibile economizzare rispetto al carico imposto alla rete racchiudendo, quando possibile, messaggi provenienti da diversi partecipanti.

8.5 Il formato dei pacchetti RTCP

I pacchetti RTCP sono di 5 tipi diversi:

`SR` Sta per *sender report* e contiene il rapporto di ricezione e trasmissione da parte di partecipanti che trasmettono.

`RR` Sta per *receiver report* e contiene il rapporto di ricezione da parte di partecipanti che *non* trasmettono.

`SDES` Sta per *source description* e contiene la descrizione (compreso l'indirizzo al livello trasporto, come detto sopra) di un partecipante.

`BYE` Indica l'abbandono della sessione.

`APP` Indica che il tipo è specifico per una certa applicazione.

Tutti si aprono con una identica sequenza, simile a quella di RTP, e rappresentata in figura 8.2.

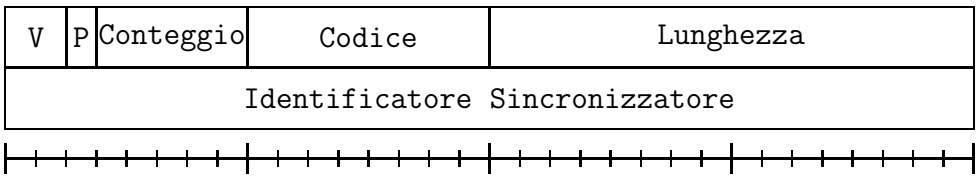


Figura 8.2 Intestazione di un pacchetto RTCP

V Il campo indica il numero di versione, sempre la 2.

P Segnala la presenza di padding, come per lo header RTP.

Conteggio Il campo indica il numero di elementi racchiusi in questa PDU: infatti un pacchetto RTCP può contenere un numero variabile di elementi, tutti dello stesso tipo.

Codice Il campo codifica in quale delle categorie dette sopra ricade il pacchetto: `SR` (codice=200), `RR`(201), `SDES`(202), `BYE`(203) e `APP`(204). In quest'ultimo caso il campo `conteggio` assume un significato diverso da quello detto sopra.

Lunghezza Il campo indica la lunghezza del pacchetto, corrispondente al numero di parole di quattro ottetti comprese nel pacchetto (compreso lo header) decrementato di una unità.

Identificatore Sincronizzatore Il campo indica il mittente del pacchetto.

Ci interessiamo prevalentemente dei messaggi di tipo `RR` e `SR`.

8.6 I messaggi `RR`

Questo tipo di messaggio è composto di un numero variabile di blocchi, di lunghezza costante e corrispondente a 6 parole composte da 4 ottetti ciascuna. Ciascun blocco contiene un rapporto sui pacchetti ricevuti da un certo mittente. Il formato corrispondente è illustrato in figura 8.3.

Identificatore di SSRC L'identificatore del partecipante a cui si riferisce il rapporto.

Pacchetti persi Il campo contiene la frazione di pacchetti che il destinatario non ha ricevuto: il mittente verrà dunque a sapere in quale misura i suoi messaggi vengono perduti senza raggiungere quello specifico destinatario. La frazione è rappresentata in virgola fissa, con la virgola collocata all'estrema sinistra. Quindi la cifra indicata corrisponde a 256-esimi, e per ottenere la frazione decimale è necessario dividere il valore del campo per 256.

Vengono conteggiati tutti i pacchetti perduti, a partire dal numero d'ordine iniziale sino a quello che viene riscontrato con questo pacchetto.

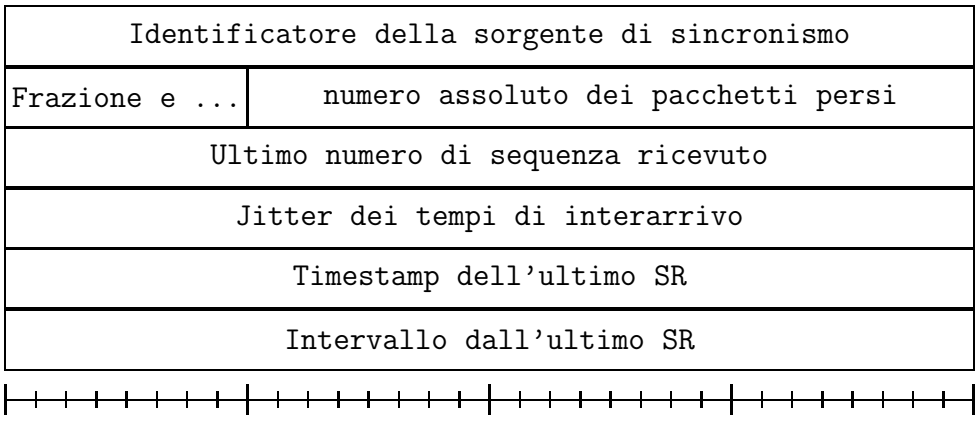


Figura 8.3 Formato di un blocco `RR`

Ultimo numero d'ordine ricevuto Questo campo serve da riscontro, come in TCP. Il numero d'ordine (che abbiamo visto essere di *soli* 16 bit) viene esteso con un prefisso che consente sessioni prolungate senza ripetizioni di numeri d'ordine.

Jitter dei tempi di interarrivo Questo campo contiene un parametro numerico che serve a valutare la regolarità della comunicazione. Il *jitter* quantifica la differenza media tra i ritardi di propagazione dei pacchetti. Quindi si tratta di una valutazione che non tiene conto del *ritardo di propagazione* in assoluto, ma della *regolarità* di tale ritardo. Se S_i ed S_j sono i tempi di spedizione di due pacchetti, e R_i ed R_j sono i tempi di ricezione degli stessi pacchetti, la formula per il calcolo della deviazione è la seguente:

$$\text{Deviazione}(i, j) = |(R_j - S_j) - (R_i - S_i)|$$

Per ottenere il jitter, si accumulano i valori di deviazioni successive, dando maggior peso a quelli più recenti. La formula è la seguente:

$$Jitter(i) = Jitter(i-1) + \frac{\text{Deviazione}(i-1, i) - Jitter(i-i)}{16}$$

L'effetto è quello di ottenere una valutazione media della variabilità dei tempi di comunicazione che dia maggior peso ai dati più recenti, e meno quelli lontani nel passato.

Timestamp dell'ultimo `SR` Questo campo contiene i 32 bit centrali della rappresentazione NTP del tempo di ricezione dell'ultimo pacchetto RTCP ricevuto da quella sorgente. Da notare che questa regola non è necessariamente la stessa utilizzata per assegnare i timestamp dei pacchetti RTP dei dati. Il timestamp così prodotto ha una risoluzione di 15 μs circa.

Intervallo dall'ultimo `SR` In questo campo viene indicato il ritardo, espresso con la stessa notazione del campo precedente, tra la ricezione dell'ultimo pacchetto da quella sorgente, e la spedizione di questo pacchetto, misurati sul clock locale.

Utilizzando il campo precedente e questo campo, la sorgente di sincronismo che riceve questo messaggio può ottenere un campione del ritardo di comunicazione del messaggio: se il rapporto riporta un `timestamp dell'ultimo SR` S, ed un `intervallo dall'ultimo SR` d, la sorgente di sincronismo, ricevendo il rapporto e leggendo il clock interno t potrà stimare il tempo di andata e ritorno (*roundtrip*):

$$\text{Roundtrip} = (t - S) - d$$

8.7 I messaggi `SR`

Il messaggi `SR` si distinguono dagli `RR` perché si aprono con un rapporto sull'attività del mittente del messaggio. Poi proseguono con una sequenza di rapporti di ricezione analoghi a quelli descritti per i messaggi `RR`.

Il blocco del messaggio `SR` relativo al mittente ha il formato illustrato nella figura 8.4

Timestamp NTP Il campo indica il tempo trascorso dal 1 Gennaio 1970. La *rappresentazione del tempo* secondo lo standard [12] occupa 64 bit, corrispondenti ad 8 ottetti. I primi 4 indicano i secondi interi dall'epoca iniziale, mentre i successivi 4 indicano la frazione di secondo, ancora in virgola mobile collocata all'estrema sinistra. Quindi la risoluzione è di $2^{-4}\mu$s circa. Da qui viene estratto il valore S utilizzato per la determinazione del roundtrip.

Timestamp RTP Il campo contiene il timestamp prodotto con la stessa regola utilizzata per produrre i timestamp nei pacchetti RTP. Questa informazione, insieme alla precedente, consente di sincronizzare tra di loro i

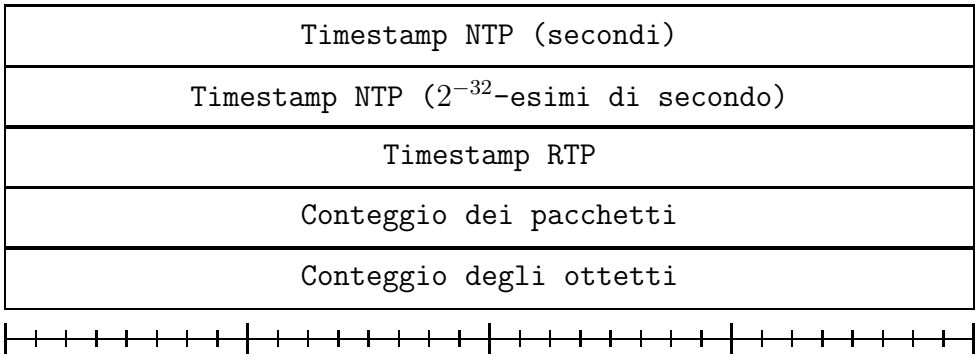


Figura 8.4 Formato di un blocco `SR`

flussi provenienti da sorgenti distinte, a condizione che il loro meccanismo di generazione dei timestamp NTP sia sincronizzato.

Conteggio dei pacchetti Il campo indica il numero di pacchetti generati dalla sorgente, dall'inizio della trasmissione.

Conteggio ottetti Il campo indica il numero totale di ottetti di *carico utile* prodotti dalla sorgente, dall'inizio della trasmissione. Può essere utilizzato per valutare la velocità con cui la sorgente produce dati.

Esercizi

- Perché non viene indicato TCP per supportare RTP?
- In quale circostanza può essere opportuno che il numero di sequenza non rispecchi il timestamp?
- In che anno il timestamp NTP andrà in *overflow*?

Bibliografia

[1] Internet Protocol – DARPA Internet program protocol specification. Request for Comment 791, Defense Advanced Research Projects Agency, September 1981.

[2] Transmission Control Protocol – DARPA Internet program protocol specification. Request for Comment 793, Defense Advanced Research Projects Agency, September 1981.

[3] R. Atkinson. IP authentication header. Request for Comment 1826, Network Working Group, August 1995.

[4] F. Baker. Requirements for IP Version 4 routers. Request for Comment 1812, Network Working Group, June 1995.

[5] R. Braden and J. Postel. Requirements for Internet gateways. Request for Comment 1009, Network Working Group, June 1987.

[6] S. Bradner. Key words for use in RFCs to indicate requirement levels. Request for Comment 2119, Network Working Group, March 1997.

[7] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A Simple Network Management Protocol (SNMP). Request for Comment 1157, Network Working Group, May 1990.

[8] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) specification. Request for Comment 1883, Network Working Group, 1995.

[9] W. Fenner. Internet group management protocol, Version 2. Request for Comment 2236, Network Working Group, November 1997.

[10] Internet Engineering Task Force. Requirements for Internet hosts – Communication layers. Request for Comment 1122, Network Working Group, October 1989.

[11] Charles Hornig. A standard for the transmission of IP datagrams over Ethernet networks. Request for Comment 894, Network Working Group, April 1984.

[12] D.L. Mills. Network time protocol (version 3). Request for Comment 1305, Network Working Group, March 1992.

[13] J. Mogul and S. Deering. Path MTU discovery. Request for Comment 1191, Network Working Group, November 1990.

[14] C. Perkins. IP encapsulation within IP. Request for Comment 2003, Network Working Group, October 1996.

[15] David C. Plummer. An Ethernet address resolution protocol. Request for Comment 826, Network Working Group, November 1982.

[16] J. Postel. User datagram protocol. Request for Comment 768, Network Working Group, August 1980.

[17] J. Postel. Internet control message protocol. Request for Comment 792, Network Working Group, September 1981.

[18] J. Postel. The TCP maximum segment size and related topics. Request for Comment 879, Network Working Group, November 1983.

[19] J. Postel and J. Reynolds. File Transfer Protocol (FTP). Request for Comment 959, Network Working Group, October 1985.

[20] J. Reynolds and J. Postel. Assigned numbers. Request for Comment 1700, Network Working Group, October 1994.

[21] M. Rose and K. McCloghrie. Structure and identification of management information for TCP/IP-based internets. Request for Comment 1155, Defense Advanced Research Projects Agency, May 1990.

[22] H. Shultzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. Request for Comment 1889, Audio-Video Transport Working Group, January 1996.

[23] T. Socolofsky and C. Kale. A TCP/IP tutorial. Request for Comment 1180, Network Working Group, January 1991.

[24] William Stallings. *Data and computer communications*. Prentice Hall, 6th edition, 2000.

[25] William Stallings. *Trasmissione dati e reti di computer*. Gruppo Editoriale Futura, 2000.

[26] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall International Editions, 1989.

Tabella degli acronimi

AAL *ATM Adaptation Layer* – Lo strato superiore della gerarchia ATM. I cinque AAL ottimizzano l'uso del supporto ATM rispetto a diverse classi di utenza.

ARP *Address Resolution Protocol* – Il protocollo Internet per ottenere l'indirizzo fisico corrispondente ad un indirizzo IP.

ASN-1 *Abstract Syntax Notation - 1* – Notazione indipendente dalla macchina utilizzata per descrivere i dati trasferiti dai protocolli di comunicazione.

ATM *Asynchronous Transfer Mode* – Supporto di rete basato sul trasferimento di celle di lunghezza costante su un mezzo affidabile.

BSD *Berkeley Software Distribution* – Organizzazione che cura la distribuzione del software prodotto presso l'Università di Berkeley.

CBR *Constant Bit Rate* – Una delle modalità di trasferimento dati offerta da ATM.

CRC *Cyclic Redundancy Check* – Un codice numerico calcolato a partire da un dato di partenza. Può rilevare certe alterazioni del dato ed anche ricostruirlo.

CSMA *Carrier Sense Multiple Access* – vedi CSMA/CD

CSMA/CD *Carrier Sense Multiple Access - Collision Detection* – La modalità di arbitraggio di un supporto Ethernet.

DF *Don't fragment* – Modalità di trattamento di un pacchetto IP, che consiste nel rifiutarlo se richiede frammentazione. Indicata da un *flag* nello *header* IP.

DHCP *Dynamic Host Configuration Protocol* - Protocollo Internet per la configurazione automatica dei parametri di rete di un host.

DNS *Domain Name Service* – Servizio Internet destinato a tradurre indirizzi mnemonici (hostname) in indirizzi IP.

DOS *Disk Operating System* – Sistema operativo Microsoft per PC.

EPROM *Electrically Programmable Read Only Memory* – Memoria a sola lettura che può essere impostata una sola volta con apparecchiature speciali.

FDDI *Fiber Distributed Data Interface* – Supporto di rete che utilizza uno schema *token ring*.

FTP *File Transfer Protocol* – Protocollo Internet per il trasferimento di file.

GNU *GNU is Not UNIX* – Progetto per la produzione di applicativi tipici del sistema operativo UNIX (ed es. il compilatore C) per sistemi operativi diversi da UNIX.

GSM *Global System for Mobile communication* – Standard europeo per la comunicazione telefonica cellulare.

HDLC *High-level Data Link Control* – Un protocollo *data link* sincrono, utilizzato come base per altri protocolli.

IAB *Internet Activities Board* – Il comitato che coordina i progetti che afferiscono ad Internet.

IBM *International Business Machine corporation* – La IBM...

ICMP *Internet Control Message Protocol* – Il protocollo destinato a trasferire i messaggi di controllo di IP.

IEEE *Institute of Electrical and Electronics Engineers, Inc.* – Associazione statunitense di supporto all'innovazione tecnologica.

IGMP *Internet Group Management Protocol* – Il protocollo Internet destinato a coordinare i processi che fanno parte di un gruppo multicast.

IHL *Internet Header Length* – Uno dei campi nello header IP.

IP *Internet Protocol* – Il protocollo Internet destinato a gestire l'interconnessione di reti distinte tramite *routing* attraverso nodi in comune tra le reti.

ISDN *Integrated Services Digital Network* – Un protocollo in grado di supportare, sullo stesso mezzo, diversi canali e modalità di comunicazione.

ISN *Initial Sequence Number* – Il numero d'ordine del primo ottetto scambiato in una connessione TCP.

ISO *International Organization for Standardization* – Acronimo non-standard per l'organizzazione che emana standard internazionali.

LAN *Local Area Network* – Rete di comunicazione in grado di coprire aree limitate (ad es. un edificio).

LAPB *Link Access Procedure - Balanced* – Protocollo *data link* derivato da HDLC ed adottato da ISDN.

MF *More Fragments* – Uno dei campi nello header IP, indica che il pacchetto originario è stato frammentato, e che il frammento contentuto nel pacchetto non è quello conclusivo.

MIB *Management Information Base* – La registrazione dei principali parametri di funzionamento di un *host*, la cui sintassi è descritta utilizzando la ASN-1.

MSL *Maximum Segment Lifetime* – Quantità di tempo che corrisponde alla stima del massimo tempo di permanenza di un segmento TCP in rete.

MTU *Maximum Transmission Unit* – La massima dimensione di un pacchetto che può transitare su un link senza essere frammentato.

NCP *Network Control Protocol* – Protocollo predecessore di Internet.

NIC *Network Information Center* – Rete di organizzazioni destinate a coordinare l'uso di Internet.

NRZ-L *NonReturn to Zero - Level* – Una codifica di dati binari con un segnale elettrico, basata sul livello di tensione.

NRZI *NonReturn to Zero - Inverted* – Una codifica di dati binari con un segnale elettrico, basata sulla variazione di livello di tensione.

NTP *Network Time Protocol* – Protocollo Internet per la sincronizzazione del clock locale utilizzando un clock remoto di riferimento.

OSI *Open System Interconnection* – Lo standard per la connessione di una rete di computer emanato da OSI.

PC *Personal Computer* – Computer che, per costo e dimensione, è adatto ad un singlo utente.

PDU *Protocol Data Unit* – L'unità di informazione trattata da un protocollo di comunicazione.

PMTU *Path MTU* – La massima dimensione di un pacchetto che può transitare su un *path* composto da più *link* tra due *host* senza essere frammentato.

RFC *Request For Comment* – Il rapporto tecnico che contiene indicazioni per la realizzazione dei protocolli Internet.

RTCP *Real Time Control Protocol* – Il protocollo Internet destinato a trasferire i messaggi di controllo di RTP.

RTP *Real Time Protocol* – Il protocollo Internet di trasporto orientato alle applicazioni con requisiti di risposta in tempo reale.

SLIP *Serial Line Interface Protocol* – Un protocollo *data link* su linea seriale.

SMTP *Simple Mail Transfer Protocol* – Un protocollo il trasporto della posta elettronica.

SNMP *Simple Network Management Protocol* – Un protocollo per il controllo della configurazione dei nodi di una rete.

TCB *Transmission Control Block* – La struttura dati che contiene le informazioni riguardanti una connessione TCP.

TCP *Transmission Control Protocol* – Un protocollo Internet di trasporto orientato alla connessione.

TDM *Time Division Multiplexing* – Una modalità di condivisione di una risorsa basata sull'assegnamento della risorsa a ciascun utente per periodi di tempo limitati.

TOS *Type of Service* – Uno dei campi nello header IP.

TTL *Time To Live* – Il contatore che limita il numero di *hop* di un pacchetto IP.

UDP *User Datagram Protocol* – Un protocollo Internet di trasporto orientato al pacchetto.

UTC *Universal Time Coordinate* – Uno degli standard per l'indicazione del tempo. Corrisponde, con piccola approssimazione, all'ora di Greenwich.

VBR *Variable Bit Rate* – Una delle modalità di trasferimento dati offerta da ATM.

VCC *Virtual Channel Connection* – Una serie di canali virtuali ATM destinati alla comunicazione unidirezionale tra due utenti.

VPC *Virtual Path Connection* – Una serie di cammini virtuali ATM destinati alla comunicazione unidirezionale tra due utenti.

WAN *Wide Area Network* – Rete di comunicazione senza limiti di estensione geografica.

Indice analitico