CAPITOLO 2

COMPRESSIONE DELL'HEADER E ARCHITETTURA TCP/IP

2.1 Architettura TCP/IP

L'*architettura TCP/IP* fu introdotta per la prima volta nel 1974 da Cerf e Kahn [16] come modello di riferimento per la rete ARPANET, commissionata dal Dipartimento della Difesa statunitense. Il nome "TCP/IP" derivava dai due protocolli di base utilizzati: l'**Internet Protocol** (IP) [12] ed il **Transmission Control Protocol** (TCP) [4]. La filosofia di progettazione legata al modello attualmente utilizzato è stata presentata da Clark [17] nel 1988.

2.1.1 Architetture a "livelli"

Due requisiti alla base del grande successo della architettura TCP/IP sono la capacità di collegare reti con tecnologie differenti e quella, per le varie unità interconnesse, di "sopravvivere" ai guasti delle sottoreti di collegamento. Per garantire questi ed altri requisiti, il modello TCP/IP si sviluppa su una *struttura gerarchica a livelli*, compatibile, sia pure con le proprie differenze, con quella proposta per la prima volta con il *modello OSI* (*Open System Interconnection*) dell'*International Standard Organization* (*ISO*) e schematizzata in figura 1.2.

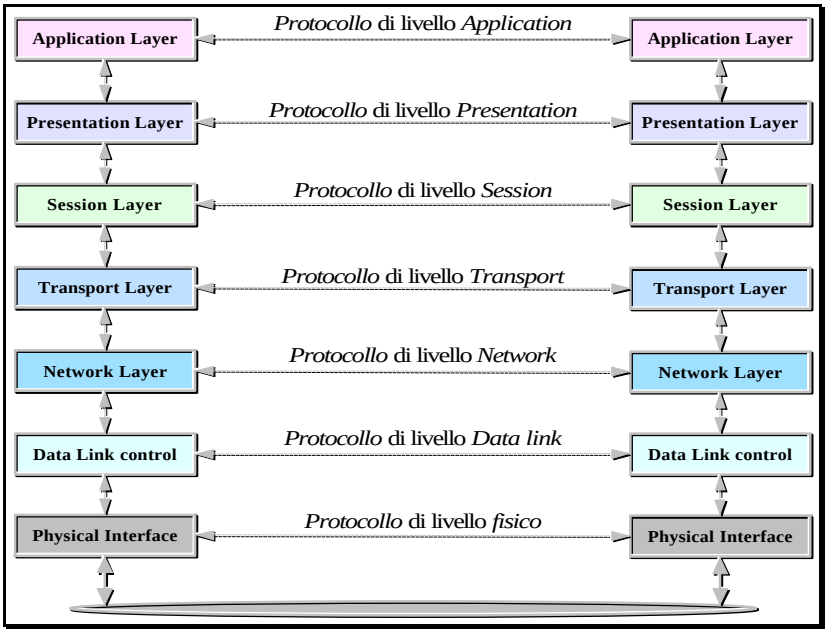

Figura 1.2 - Struttura a livelli del modello ISO/OSI

Nelle strutture a livelli, ogni livello è responsabile di aspetti diversi della comunicazione; il livello *n* su una macchina permette una conversazione con il

livello *n* su un'altra macchina. Il numero dei livelli, il nome di ciascun livello, il suo contenuto e le sue funzionalità differiscono da una rete all'altra.

La funzione di ogni livello è quella di fornire dei **servizi** al livello superiore. Gli elementi attivi di ogni livello sono spesso chiamati **entità** e può trattarsi di software (un processo) o moduli hardware (un componente intelligente di gestione dell'input/output). Le entità al livello *n* implementano i servizi usati al livello *n+1*: il livello *n* è perciò detto *fornitore di servizi* e il livello *n+1* è detto *utente di servizi*.

Fra ogni coppia di livelli adiacenti esiste una **interfaccia**, la quale definisce quali operazioni primitive e quali servizi sono offerti da ciascun livello al livello superiore.

Le regole e le convenzioni usate per le "conversazioni" tra livelli corrispondenti sono racchiuse nei corrispondenti **protocolli**: ciascuna entità usa, per la comunicazione con la pari entità (*peer-entity*) su un altro nodo, il protocollo corrispondente al proprio livello.

Un insieme di livelli e protocolli prende il nome di **architettura di rete**. Una lista di protocolli (uno per ciascun livello) usati da un certo sistema è invece chiamata **pila di protocolli**.

I livelli possono offrire due diversi tipi di servizi ai livelli superiori: i *servizi orientati alla connessione* (connection-oriented) e i *servizi non orientati alla connessione* (connection-less).

I *servizi orientati alla connessione* sono modellati sul sistema telefonico: l'utente del servizio prima stabilisce una connessione, poi la utilizza e infine la rilascia. La connessione agisce dunque come una specie di "tubatura": il mittente introduce oggetti (bit) a partire da un'estremità e il ricevente li riprende nello stesso ordine all'altra estremità; nessun'altro può usare la stessa tubatura.

I *servizi privi di connessione* sono invece modellati sul sistema postale: ogni messaggio porta con sé l'indirizzo completo di destinazione e viene condotto lungo il sistema indipendentemente da tutti gli altri. Normalmente, quando due messaggi sono inviati alla stessa destinazione, arrivano nello stesso ordine con cui sono partiti; tuttavia, è anche possibile che il primo inviato possa essere ritardato a tal punto che il secondo arriva prima. Con un servizio orientato alla connessione questo non è invece possibile.

Alcuni servizi sono *affidabili*, nel senso che garantiscono l'arrivo corretto a destinazione di tutti i dati. Viceversa, sono *non affidabili* i servizi che non forniscono questa garanzia.

Normalmente, un servizio affidabile è realizzato attraverso un *messaggio di avvenuta ricezione*, inviato dai destinatari ogni qualvolta ricevono un nuovo messaggio, in modo tale che il mittente si assicuri della corretta trasmissione. Tale processo introduce però lavoro ulteriore e ritardi, che sono utili ma a volte indesiderati: ad esempio, le garanzie di corretta ricezione sono necessarie nel trasferimento di archivi, mentre invece i relativi ritardi introdotti sono inaccettabili per alcune applicazioni, quali la trasmissione di voce digitalizzata, con stringenti requisiti di tipo real-time.

2.1.2 Il modello TCP/IP

I requisiti di progetto indicati dal Ministero della Difesa statunitense portarono alla scelta di una *rete a commutazione di pacchetto* basata su un *livello non orientato alla connessione*. Questo livello, chiamato **livello internet**, è il perno che mantiene assieme tutta l'architettura. Il suo compito è permettere a un host di inserire pacchetti in una qualsiasi rete in modo che questi possano viaggiare, indipendentemente uno dall'altro, verso la destinazione. Essi possono arrivare anche in un ordine diverso rispetto a quello con cui erano stati inviati: sarà compito di qualche livello superiore riordinarli.

L'architettura progettata è riportata nella figura 2.2.

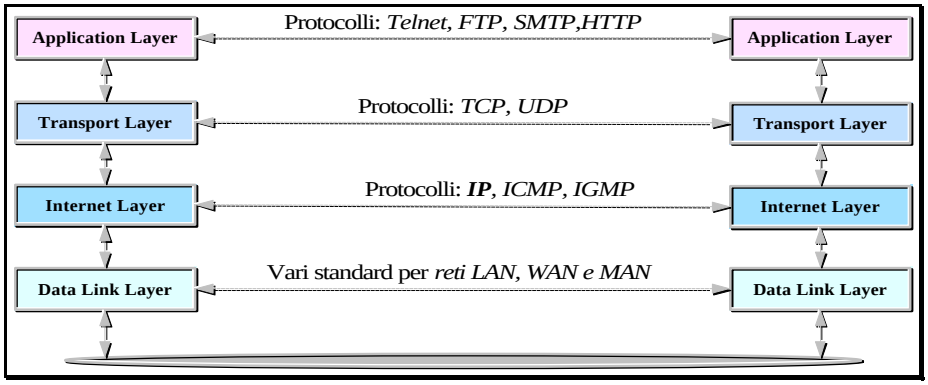

Figura 2.2 - Livelli e relativi principali protocolli per una architettura TCP/IP

Al posto dei sette livelli previsti dal modello generale OSI (figura 1.2), il modello TCP/IP prevede solo 4 livelli, con i seguenti compiti fondamentali:

- il *livello di applicazione* (*application*) si occupa dei dettagli legati alla particolare applicazione; un insieme limitato ma significativo delle possibili applicazioni è riportato nella tabella 1.2;
- il *livello di trasporto* (*trasport*) è il primo che si occupa di gestire il flusso di dati a livello end-to-end (ossia tra i due estremi della connessione). Il modello TCP/IP prevede due differenti protocolli per questo livello: il **TCP** (*Transmission Control Protocol*) è un protocollo affidabile ed orientato alla connessione, mentre invece **UDP** (*User Datagram Protocol*) [18] è un protocollo non affidabile e non orientato alla connessione;
- il *livello internet* si occupa del "movimento" dei pacchetti all'interno della rete. In particolare, questo compito è affidato primariamente al protocollo **IP** (*Internet Protocol*), eventualmente affiancato dall'**ICMP** (*Internet Control Message Protocol*) e dall'**IGMP** (*Internet Group Management Protocol*) [19]. Il protocollo IP deve fondamentalmente scegliere il cammino dei pacchetti verso la destinazione; secondariamente, deve prendere anche alcuni provvedimenti per evitare condizioni di congestione della rete;
- il *livello fisico* (*data link*) si occupa infine di tutti i dettagli per l'interfaccia con il mezzo fisico utilizzato per la comunicazione; il modello TCP/IP non dice molto a proposito di quanto avviene a questo livello, limitandosi a specificare che l'host deve connettersi alla rete utilizzando un qualunque protocollo in grado di inviare pacchetti IP lungo di essa; questo protocollo varia da host a host e da rete a rete.

Tabella 1.2 - Applicazioni tipiche per le reti TCP/IP

Applicazione	***Protocollo di livello application***
Terminale virtuale	**TELNET**
Trasferimento di archivi	**FTP** (File Transfer Protocol)
Posta Elettronica	**SMPT** (Simple Mail Transfer Protocol)
Trasferimento ipertesti	**HTTP** (HyperText Transfer Protocol)
Gestione di una rete di computer	**SNMP** (Simple Network Management Protocol)

Compiti del livello network

Il *livello network* (che, nel modello TCP/IP, prende il nome di *livello internet*) si occupa di trasmettere pacchetti dalla sorgente verso la destinazione. Per raggiungere la destinazione, può essere necessario attraversare diversi nodi intermedi lungo il percorso. Tale livello assolve dunque ad una funzione chiaramente diversa da quella del livello inferiore (*data link*), il quale ha il compito più modesto di trasportare pacchetti da un estremo all'altro di un cavo. Quindi, il livello rete è il più basso tra i livelli che si occupano di trasmissione punto-a-punto.

Per raggiungere i suoi obiettivi, il livello rete deve avere informazioni sulla topologia della rete di comunicazione, in modo da scegliere i percorsi appropriati attraverso di essa. Deve inoltre fare attenzione a scegliere i percorsi in modo tale da non sovraccaricare parte delle linee di comunicazione e dei nodi intermedi, così da evitare situazioni di congestione. Inoltre, qualora il mittente e il destinatario appartengano a reti differenti, è compito del livello rete tenere conto di questa differenza e risolvere i problemi causati da essa.

I *servizi* del livello rete vengono generalmente progettati tenendo conto di due fondamentali obiettivi: devono essere indipendenti dalla tecnologia e dalla rete ed il livello di trasporto deve essere indipendente dal numero, dal tipo e dalla topologia delle reti presenti.

Nel caso specifico del *livello internet* previsto dal modello TCP/IP, esso ha il compito fondamentale di fornire un meccanismo *best-effort* per trasportare pacchetti dalla sorgente alla destinazione, indipendentemente dall'esistenza di reti intermedie lungo il percorso e dal fatto che le singole "macchine" appartengano oppure no alla stessa destinazione.

2.1.3 Esempio di funzionamento di una rete TCP/IP

Il principale punto di forza del modello TCP/IP è nel garantire la comunicazione tra due utenti qualunque sia la rete che li interconnette. Si consideri, per esempio, il trasferimento di un file tramite *protocollo FTP*, da un computer B (detto *server*, in quanto "fornisce" il servizio) ad un computer A (detto *client*, in quanto "riceve" il servizio), come illustrato nella figura 3.2.

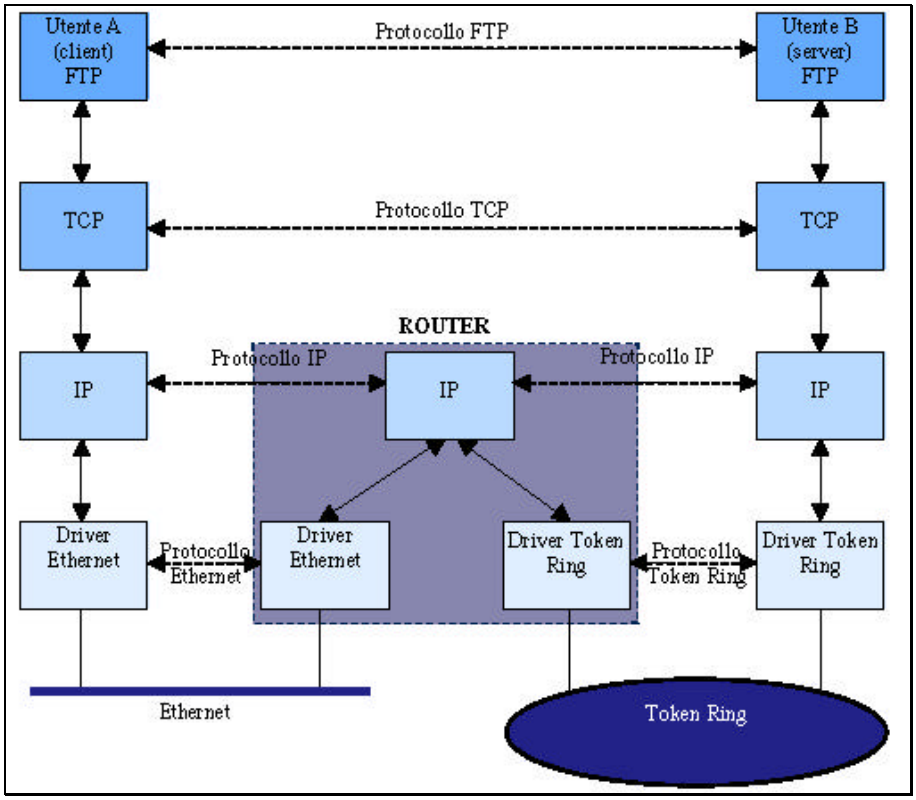

Figura 3.2 - Esempio di connessione tra due reti eterogenee mediante un router

Viene considerato il caso in cui i due computer sono collegati tra loro mediante un *router* ([1]) e due reti fisiche diverse (Ethernet e Token Ring).

Il livello application dell'utente A "vede" solo la comunicazione end-to-end con lo stesso livello dell'utente B, mentre invece ignora del tutto quanto avviene nei livelli sottostanti. Stesso discorso vale per gli altri livelli.

Per ogni livello, quindi, esistono due tipi di collegamento: il *collegamento virtuale* è quello che si realizza tra livelli corrispondenti dei due computer collegati; il *collegamento fisico* è invece quello presente all'interfaccia di ogni livello sulla stessa macchina e corrisponde allo scambio di dati tra livelli adiacenti.

Il concetto di *collegamento virtuale* è proprio ciò che svincola la connessione dalla rete sottostante, rendendola universale.

La comunicazione tra i vari livelli di ciascun nodo avviene attraverso la tecnica dell'**incapsulamento** in trasmissione e del **demultiplexing** in ricezione (figura 4.2):

[1] Un router è un dispositivo che consente di collegare due o più reti, anche eterogenee tra loro. Ogni pacchetto al suo ingresso viene esaminato fino a livello rete e viene successivamente "adattato" alle caratteristiche della rete in uscita su cui deve essere spedito.

- in trasmissione, il protocollo di ciascun livello riceve dati dal protocollo di livello superiore (**payload**) ed aggiunge ad essi una propria intestazione (**header**), dopodiché passa il tutto al livello inferiore; il processo termina in corrispondenza del livello più basso (livello fisico), dove viene generato il *frame* finale da trasmettere sul mezzo di comunicazione;
- in ricezione, avviene il processo inverso, per cui ogni protocollo riceve dati dal protocollo inferiore, estrae l'header di competenza e lo utilizza per compiere le dovute elaborazioni, dopodiché passa i dati rimanenti al protocollo superiore; il processo si arresta quando l'applicazione destinataria riceve i dati utente.

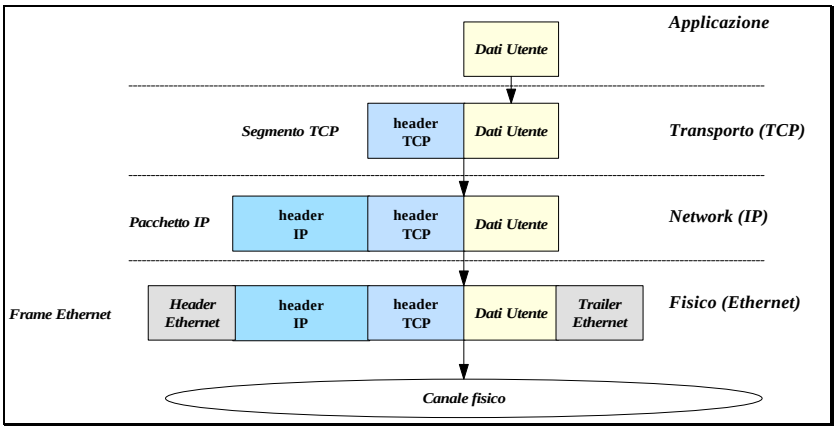

Figura 4.2 - Procedura di incapsulamento dei pacchetti in una architettura TCP/IP. Si considera la "sezione di trasmissione"di una rete Ethernet

Per poter recapitare i dati generati da un processo sorgente ad una ben precisa applicazione destinataria su una ben definita macchina, vengono usati appositi *sistemi di indirizzamento* delle macchine e, su ciascuna macchina, vengono allocate delle *porte* ciascuna corrispondente ad una determinata applicazione ricevente.

Nella figura 4.2 è stato illustrato quanto avviene per la trasmissione mediante TCP, ma lo schema conserva la sua validità anche usando il protocollo UDP: in questo secondo caso, cambierebbero la dimensione dell'header (20 byte per il TCP e 8 byte per UDP) e il nome dell'unità di informazione passata al protocollo IP (detta *segmento TCP* nel primo caso e *datagramma UDP* nel secondo).

2.1.4 Principali protocolli TCP/IP

Nell'ambito delle reti a commutazione di pacchetto, un **protocollo** rappresenta, nella sua concezione più generica, un insieme di regole che determinano il formato e la semantica dei pacchetti generati al livello cui il protocollo "appartiene".

Ogni pacchetto è costituito da un **header**, che contiene tutte le informazioni "di controllo" utili all'espletamento delle funzioni cui è preposto il protocollo, e da un **payload**, corrispondente all'informazione utile da trasmettere (in trasmissione) oppure da trasferire al protocollo di livello superiore (in ricezione).

Di seguito vengono esaminati i protocolli di interesse per questa tesi: *TCP*, *IP versione 4*, *IP versione 6*.

2.1.4.1 TCP (Transmission Control Protocol)

Il **protocollo TCP** [4] fu progettato esplicitamente per fornire un flusso affidabile, a livello end-to-end, utilizzando una rete (intesa eventualmente come "collezione" di sottoreti) intrinsecamente inaffidabile. Esso è in grado di adattarsi dinamicamente alle proprietà della rete ed è robusto nei confronti di molte tipologie di guasto.

Per rispettare questi requisiti, il protocollo TCP fornisce un servizio *orientato alla connessione* ed *affidabile*:

- *orientato alla connessione*: la generica coppia di applicazioni (client e server), che vuole comunicare utilizzando TCP, deve necessariamente stabilire una connessione (una sorta di "collegamento virtuale") prima di poter scambiare dati;
- *affidabile*: le due applicazioni, una volta stabilita la connessione, devono assicurarsi che tutti i pacchetti trasmessi dal mittente (server) vengano consegnati, immuni da errori e in ordine, al ricevente (client).

Ogni macchina che supporta TCP possiede un'*entità di trasporto TCP*, che gestisce i flussi di dati TCP e si interfaccia con il livello IP. L'entità TCP riceve i dati dai processi locali, li suddivide in unità (dette **segmenti TCP**) lunghe al più 64 Kbyte e spedisce queste unità come *datagrammi IP* separati (in base all'incapsulamento precedentemente descritto). Quando un datagramma IP

contenente dati TCP giunge ad una macchina, i dati vengono passati all'entità TCP, la quale ricostruisce il flusso originale di byte.

Il livello IP non fornisce alcuna garanzia sulla consegna corretta dei datagrammi, per cui è compito di TCP ritrasmetterli quando necessario. I datagrammi in arrivo possono inoltre essere nell'ordine sbagliato: è ancora compito di TCP riassemblare i messaggi nella sequenza corretta. In pratica, dunque, TCP deve fornire l'affidabilità che molte applicazioni richiedono e che la rete sottostante non possiede.

Modello di servizio TCP

I *servizi* del TCP si ottengono tramite la creazione, da parte del mittente e del ricevente, di punti di accesso chiamati *socket***.** Ogni socket consiste nell'indirizzo IP dell'host e in un numero di 16 bit, locale all'host, detto *porta*. Per ottenere un servizio TCP, si deve creare esplicitamente una connessione fra un socket della macchina mittente e un socket della macchina ricevente.

Tutte le connessioni TCP sono full-duplex (il traffico può scorrere in entrambe le direzioni contemporaneamente) e punto-a-punto (ogni connessione ha esattamente due estremi). TCP non supporta il multitasking e il broadcasting.

Il TCP prevede il metodo del *piggybacking* per integrare le informazioni di controllo della trasmissione in una direzione con i dati diretti nella direzione opposta.

Il TCP non fa particolari assunzioni per quanto riguarda la rete sottostante, per cui funziona, con prestazioni diverse, su reti diverse: collegamenti telefonici commutati, reti locali, reti ad alta velocità in fibra ottica, collegamenti punto-punto a bassa velocità su lunghe distanze, reti miste wireless-wired.

Protocollo TCP

Le entità TCP mittente e ricevente si scambiano dati sotto forma di *segmenti TCP*. Ogni segmento è formato da un *header* ed un *payload*; l'header comincia sempre con una parte fissa da 20 byte ed una parte opzionale di lunghezza variabile (multipla di parole da 32 bit). La dimensione dei segmenti viene fissata dal software TCP tenendo conto di due vincoli: ogni segmento deve entrare in un pacchetto IP di 64 Kbyte e la dimensione di quest'ultimo non deve superare la **MTU** (*Maximum Transfer Unit*, unità massima di trasferimento) delle rete o delle reti su cui dovrà transitare.

Il principale protocollo utilizzato dall'entità TCP per modulare la propria frequenza di trasmissione è di tipo *sliding window* (finestra scorrevole): al momento di trasmettere un segmento, il mittente inizializza un timer (detto *timer di ritrasmissione*); quando il segmento arriva a destinazione, l'entità TCP ricevente spedisce indietro un nuovo segmento (contenente eventualmente anche dati) di *riscontro della ricezione* (**ACK**). Se però il timer del mittente scade prima che l'ACK sia ricevuto, il segmento viene ritrasmesso. In ogni caso, il TCP mittente non aspetta l'ACK di ciascun segmento per trasmettere il segmento successivo: esso invia più segmenti in sequenza senza attendere il loro riscontro, fin quando non si raggiunge il numero massimo di segmenti inviabili, pari alla dimensione della finestra mobile; a questo punto, ogni riscontro ricevuto produce lo scorrimento della finestra mobile in modo che nuovi segmenti possano essere trasmessi.

I segmenti ACK inviati dal TCP ricevente sono dunque usati dal TCP mittente per regolare la propria politica di trasmissione. La durata dei timeout e la modalità di ritrasmissione sono stabilite attraverso un *algoritmo di controllo della congestione*.

Checksum TCP

Per garantire l'integrità dei dati consegnati alla destinazione, il protocollo TCP prevede, nell'header di base dei segmenti, un opportuno campo denominato *TCP checksum*: esso contiene il risultato di una somma di controllo fatta sull'intero segmento TCP e su alcuni campi dell'header IP ([2]).

Se, a destinazione, il controllo della checksum fallisce, il TCP trasmette un ACK riferito all'ultimo segmento ricevuto correttamente, richiedendo così implicitamente la ritrasmissione del segmento risultato errato.

Connessioni TCP

Ogni host è costantemente in attesa di una richiesta di connessione su un determinato *numero di porta* e, quando essa arriva, decide se accettarla o meno. Quando un host (*client*) vuole attivare una connessione TCP con un altro host (*server*), usa il protocollo **three-way handshake** (figura 5.2).

[2] Per maggiori informazioni, si veda il paragrafo 3.2.4.

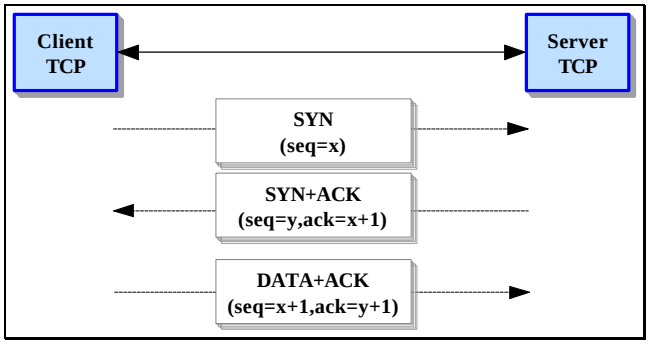

Figura 5.2 - Three-way handshake per l'instaurazione di una connessione TCP con dettaglio sui valori dei campi TCP Sequence Number e ACK Number

Il client invia al server una richiesta tramite un *segmento SYN* in cui specifica i propri parametri di connessione (indirizzo IP, porta, dimensione massima dei segmenti TCP, numero di sequenza iniziale e altro). Se il server è disponibile, risponde con un *segmento SYN+ACK*, nel quale può sia confermare i parametri "proposti" dal client sia proporne di nuovi. A sua volta, il client può confermare il pacchetto ricevuto tramite un segmento di riscontro (ACK) oppure, se non concorda con i parametri di connessione suggeriti dal server, può abortire la procedura. Se accetta quanto indicato dal server, la connessione si considera stabilita e può quindi iniziare il flusso di dati.

Da questo punto in poi, la connessione è fatta di un "dialogo" tra client e server, fortemente "sbilanciato" verso quest'ultimo: il client effettua le proprie richieste ed il server risponde inviando le informazioni (se disponibili). Quindi, almeno in linea generale, il *flusso dati* (*data stream*) va nella direzione client→server quando il client effettua le proprie richieste e nella direzione server→client quando il server risponde a tali richieste; il discorso inverso vale invece per i flussi dei riscontri (*ACK stream*).

Il TCP, inoltre, prevede la tecnica del *piggybacking* per ridurre l'overhead dell'header ([3]): questo significa che un segmento di riscontro può anche contenere dati per la sua destinazione, in modo da gestire, con un unico segmento (e quindi un unico header), sia il flusso di riscontri sia il flusso dati.

Dato il forte sbilanciamento del traffico nelle due direzioni, sono rare le situazioni con pacchetti che trasportano sia un ACK sia dati, il che significa che, tranne in casi eccezionali, i flussi da considerare sono solo due, come schematizzato in figura 6.2.

[3] Per "overhead dell'header" si intende la frazione di bit occupata dall'header rispetto all'intera dimensione del pacchetto. Analiticamente, quindi, esso corrisponde al rapporto tra la dimensione dell'header e la dimensione totale (header+payload) del pacchetto. Si tratta di un concetto fondamentale in questa tesi, per cui sarà ampiamente approfondito.

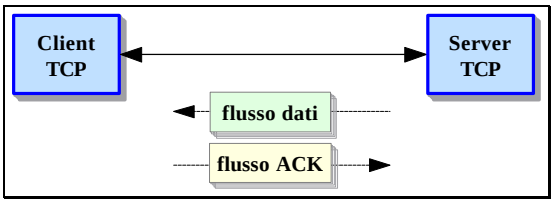

Figura 6.2 – Tipico flusso TCP

L'entità TCP del server suddivide il flusso di dati al suo ingresso (provenienti da una applicazione, ad esempio FTP) in segmenti TCP e li invia al client; nell'header di ciascun segmento, l'apposito campo *Sequence Number* riporta il numero d'ordine del primo byte del payload ([4]).

Dall'altra parte, l'entità TCP sul client, per ogni segmento TCP ricevuto, invia un segmento di riscontro, nel cui header inserisce, tramite l'apposito campo *ACK Number*, il numero di sequenza del primo byte di payload del prossimo segmento che aspetta di ricevere.

Il rilascio di una connessione TCP avviene considerando la connessione full-duplex come una coppia di connessioni simplex indipendenti: quando una delle due parti non ha più nulla da trasmettere, invia un *segmento FYN*; quando tale segmento viene confermato dall'altra parte, la connessione in uscita viene rilasciata, ma è ancora possibile ricevere dati; quando anche l'altra parte completa lo stesso procedimento e rilascia la connessione nell'altra direzione, la connessione full-duplex termina.

Politica di trasmissione del TCP

La gestione delle finestre in TCP è organizzata nel modo seguente: quando il ricevente riceve correttamente un segmento, lo conferma inviando al mittente un segmento ACK, nel quale indica sia il numero di sequenza del successivo segmento atteso (campo *ACK Number*) sia la porzione ancora libera del proprio buffer di ricezione (campo *Window Size*). Il mittente, quindi, continua a trasmettere tenendo conto di tali informazioni.

Quando il ricevente segnala un buffer vuoto, generalmente il mittente non può spedire segmenti, con due eccezioni: può spedire *dati urgenti* oppure può

[4] Per maggiori dettagli sui campi dell'header TCP si veda il paragrafo 3.2.4

spedire un segmento di 1 byte per fare in modo che il ricevente annunci nuovamente il prossimo byte atteso e la dimensione della finestra ([5]).

Non è richiesto che i mittenti trasmettano i dati appena questi giungono dall'applicazione, né si richiede che i riceventi spediscano gli ACK il più presto possibile. In quelle applicazioni in cui un dato deve essere trasferito subito, anche se non riempie il buffer, è previsto un meccanismo di *push* che forza la trasmissione.

Controllo di congestione di TCP

Sebbene anche il livello di rete si occupi delle congestioni, la maggior parte del lavoro è eseguito da TCP, in quanto l'unica soluzione per una congestione è ridurre il flusso di dati.

In linea del tutto teorica, la congestione andrebbe affrontata nel modo seguente: non si inserisce un nuovo pacchetto nella rete fino a quando uno vecchio non ne esce. TCP cerca di raggiungere questo obiettivo manipolando dinamicamente la dimensione delle proprie *finestre*.

Nelle reti attuali su cavo, la perdita di pacchetti dovuta ad errori di trasmissione è relativamente rara, in quanto gran parte dei canali a lunga distanza è su fibra ottica e quindi garantiscono altissima affidabilità. La maggior parte dei timeout di trasmissione scaduti sono quindi dovuti alle congestioni e tutti gli algoritmi TCP usati in simili reti partono da questa consapevolezza per stabilire il da farsi.

Quando viene creata una connessione, si sceglie una "dimensione di finestra accettabile". Il ricevente può specificare una finestra basata sulla dimensione del proprio buffer; se il mittente accetta questa dimensione di finestra, non vi saranno problemi causati da esaurimento del buffer del ricevente; i problemi potrebbero però essere causati dalla congestione interna della rete. La soluzione adottata consiste allora nel tenere conto dell'esistenza dei due problemi potenziali (capacità del ricevente e capacità della rete) ed affrontarli separatamente.

Ogni mittente mantiene due finestre: una relativa al ricevente (*finestra del ricevente*) e l'altra relativa alla congestione (*finestra di congestione*). Il numero di byte che può essere spedito è il valore minimo tra le due finestre, il che

[5] Questa opzione serve a prevenire lo stallo nel caso in cui un segmento contenente la dimensione della finestra sia andato perso.

significa che la finestra effettiva corrisponde al minimo tra ciò che è ritenuto adatto dal mittente e ciò che è ritenuto adatto dal ricevente.

Quando viene creata una connessione, il mittente pone la dimensione della finestra di congestione uguale alla dimensione del segmento di dimensione massima che può essere spedito e spedisce quindi un segmento di tale dimensione. Se il corrispondente ACK giunge prima che il relativo timer scada, il valore della finestra di congestione viene incrementato della dimensione massima di un segmento e il mittente spedisce due segmenti. Ogni qualvolta uno di questi segmenti viene confermato, la dimensione della finestra viene aumentata della dimensione massima di un segmento. In tal modo, quando la finestra di congestione è uguale a n segmenti, se tutti ed n vengono confermati in tempo, essa viene portata a $2 \cdot n$ segmenti di dimensione massima. La finestra di congestione continua dunque a crescere in modo esponenziale (in quanto ogni trasmissione che viene confermata ne raddoppia la dimensione), fino a quando scade un timeout oppure si raggiunge il limite corrispondente alla finestra del ricevente. Questo algoritmo viene detto *slow start* (partenza lenta), anche se non è affatto lento, essendo esponenziale. Tutte le implementazioni TCP lo supportano.

L'algoritmo di slow start prevede inoltre un parametro denominato *soglia* (*congestion threshold* o anche *slow start threshold*), posto inizialmente a 64 kB. La crescita esponenziale si ferma quando la soglia viene raggiunta: da quel punto in poi, trasmissioni con successo fanno crescere linearmente la finestra di congestione (della dimensione di un segmento massimo alla volta) invece di raddoppiarla.

Quando invece scade un timeout, la soglia viene posta a metà della finestra di congestione corrente, la finestra di congestione viene riportata alla dimensione massima di un segmento e l'algoritmo di slow start viene fatto ripartire.

Versioni TCP: Tahoe e Reno

Si vuole ora fornire una rapida descrizione della versione TCP utilizzata per le simulazioni oggetto di questa tesi. Per fare questo, è necessaria una rapida sintesi delle versioni precedenti [20].

La prima versione modificata rispetto al TCP originale [4] fu denominata 4.3 BSD Tahoe TCP (brevemente *TCP Tahoe*) e risale al 1988: in essa fu introdotto un nuovo algoritmo di controllo della congestione, ideato da Jacobson [21], che

consisteva fondamentalmente nei meccanismi di *Slow Start*, *Congestion Avoidance* e *Fast Retransmit*.

Quando si inizia una nuova sessione, il protocollo inizia la fase di slow start precedentemente descritta, durante la quale la finestra di congestione viene posta inizialmente ad uno e poi incrementata alla ricezione di ogni ACK, determinando così una crescita di tipo esponenziale.

Ogni fase di slow start termina, qualora non scada alcun timeout, quando la dimensione della finestra di congestione raggiunge la congestion threshold. A questo punto, il protocollo inizia la fase di *Congestion Avoidance*, durante la quale esso aumenta la finestra corrente di un pacchetto per ogni finestra completa di dati che viene confermata. L'aumento della finestra è dunque adesso di tipo lineare.

Nel meccanismo di *Fast Retransmit*, l'arrivo di un numero di *ACK duplicati* ([6]) pari ad una prefissata soglia (normalmente pari a 3) fa scattare una ritrasmissione senza aspettare che trascorra il timeout associato, consentendo così di migliorare l'efficienza di utilizzo del canale. In risposta a tale "timeout anticipato", Tahoe intraprende la stessa azione che avrebbe avuto per un timeout regolare, settando la congestion threshold a metà della finestra di congestione corrente ed iniziando lo slow start.

Il meccanismo di slow start non è sempre efficiente, specialmente quando l'errore non è causato dalla congestione della rete, ma ad esempio dalla perdita di un pacchetto sulla tratta radio. In tali casi, la riduzione della finestra di congestione non è necessaria e può portare ad una bassa utilizzazione della banda. La versione successiva a Tahoe, denominata 4.3 BSD TCP Reno (brevemente *TCP Reno*) e risalente al 1990 [22], ha allora introdotto l'algoritmo di *Fast Recovery*, utilizzato insieme al *Fast Retransmit*, per ovviare ai limiti dell'algoritmo di slow start.

L'algoritmo di Fast Recovery interpreta la ricezione di ACK duplicati come una indicazione di banda disponibile (in quanto ciascuno di essi testimonia che almeno un segmento è "uscito" dalla rete) e agisce di conseguenza. All'arrivo di ACK duplicati, il mittente Reno si comporta così come il mittente Tahoe, fin quando il loro numero non raggiunge una soglia prefissata (generalmente indicata come *tcprexmtthresh* e impostata al valore 3). Quando la soglia viene raggiunta, il mittente ritrasmette il pacchetto andato perso e entra in Fast

[6] Per "ACK duplicati" si intendono due o più segmenti di ACK recanti, nel campo ACK Number, lo stesso valore: essi indicano che non è stato correttamente ricevuto il segmento con numero di sequenza pari al valore recato nel campo ACK Number, mentre i segmenti precedenti sono stati tutti correttamente ricevuti.

Recovery: riduce la finestra di congestione di metà, ma, al posto di avviare l'algoritmo di slow start come avviene in Tahoe, usa gli addizionali ACK duplicati in arrivo per predisporre i prossimi pacchetti in uscita. Infatti, esso pone la dimensione della propria finestra di trasmissione pari al minimo tra *awin* (la finestra di avvertimento del ricevente) e *cwnd+ndup*, dove *cwnd* è la finestra di congestione del mittente stesso, mentre *ndup* è un numero che viene mantenuto al valore 0 fin quando il numero di ACK duplicati ricevuti non raggiunge il valore *tcprexmtthresh*, dopodiché viene posto pari al numero di ACK duplicati ricevuti

In questo modo, durante il Fast Recovery, il mittente allarga la propria finestra proporzionalmente al numero di ACK duplicati ricevuti, ottenendo un incremento del throughput.

La fase di Fast Recovery viene conclusa quando il mittente riceve il riscontro per tutti i dati inviati durante tale fase. A questo punto, esso setta l'ampiezza della congestion window al valore della congestion threshold e pone nuovamente *ndup=0*.

Comparato a Tahoe, Reno usa dunque una strategia di recupero dagli errori più aggressiva e risulta quindi più indicato per sistemi misti wired-wireless: infatti, quando riceve il numero di soglia di ACK duplicati, esso non intraprende lo Slow Start, che ridurrebbe la finestra ad un singolo pacchetto, ma efficacemente la setta a metà del suo valore precedente. Tuttavia, è stato verificato [23] che l'algoritmo di Fast Recovery per la versione Reno è realmente ottimizzato solo per il caso in cui un singolo pacchetto viene perso all'interno di una finestra di dati; viceversa, esso può soffrire problemi di prestazione non del tutto trascurabili quando vengono persi più pacchetti nella stessa finestra di dati.

La versione TCP Reno è quella utilizzata nelle simulazioni discusse nel capitolo 6 di questa tesi.

2.1.4.2 IP (Internet Protocol)

Questo è il protocollo principale che opera al livello di rete in una qualunque architettura TCP/IP. Viene comunemente detto **IP**.

Si tratta di un protocollo di tipo *connectionless* (non orientato alla connessione), che si preoccupa di inviare *datagrammi* (così si chiamano i pacchetti a livello di rete) dalla sorgente verso la destinazione.

Il livello di rete riceve un flusso di dati dal livello di trasporto: mediante IP, tale flusso viene frammentato in datagrammi e trasmesso. IP non mantiene alcuna informazione per i successivi datagrammi, per cui ciascuno di essi viene instradato indipendentemente dagli altri. Questo comporta che i datagrammi possano essere recapitati anche in ordine diverso rispetto a come sono stati spediti.

Il protocollo è inoltre di tipo *non affidabile*, nel senso che non fornisce alcuna garanzia che i datagrammi giungano a destinazione. Si dice allora che IP è un protocollo di tipo *best effort*: se qualcosa va male, ad esempio un datagramma trova pieno il buffer di ingresso su un router, il protocollo scarta il datagramma e si limita a inviare un messaggio di errore al mittente (tramite il protocollo ICMP).

IP prevede anche un semplice algoritmo di frammentazione dei datagrammi, qualora la loro dimensione superi quella massima accettabile dal livello di interfaccia al canale fisico.

IPv4 e IPv6

La **versione 4** di IP fu rilasciata nel settembre 1981 [12]. Tuttavia, sin dagli inizi degli anni '90, IPv4 ha cominciato a mostrare segni di obsolescenza, sia per l'esaurimento dello spazio di indirizzamento sia per i mutamenti profondi dello senario in cui sono venute ad operare le architetture TCP/IP. Consapevole di ciò, l'IETF ha in quegli anni iniziato il progetto di un nuovo protocollo IP che permettesse di superare i limiti evidenziati da IPv4. Nel dicembre 1995, fu così rilasciata la nuovissima **versione 6** [24], che poi fu ulteriormente perfezionata [25] nel dicembre 1998.

IPv6 si basa su una struttura più semplice rispetto ad IPv4, dispone di uno spazio di indirizzamento praticamente illimitato (in quanto utilizza indirizzi da 16 byte anziché da 4 come in IPv4), dedica maggiore attenzione alle problematiche inerenti alla mobilità (*Mobile IP*), alla sicurezza ed al trasporto di traffico dati di tipo *real time*.

2.2 La compressione dell'header

La comunità scientifica internazionale sta attualmente producendo un notevole sforzo per l'individuazione e la sperimentazione di possibili tecniche con cui ottimizzare il traffico di informazioni sui link wireless, che soffrono notoriamente di grosse limitazioni in termini di banda disponibile e rumorosità, ma che, al contempo, risultano di importanza critica per rendere le reti efficienti e quindi competitive sul mercato.

La tendenza attuale verso la realizzazione di reti di tipo "**all-IP**" (paragrafo 1.3) ha accentuato ulteriormente l'importanza del problema e l'urgenza della sua soluzione.

In questa tesi, ci si concentra specificamente su uno degli aspetti di ottimizzazione per questo tipo di reti: la **compressione dell'header sulla tratta radio**.

Per comprendere a pieno l'opportunità di comprimere l'header, si consideri uno scenario del tipo riportato in figura 7.2.

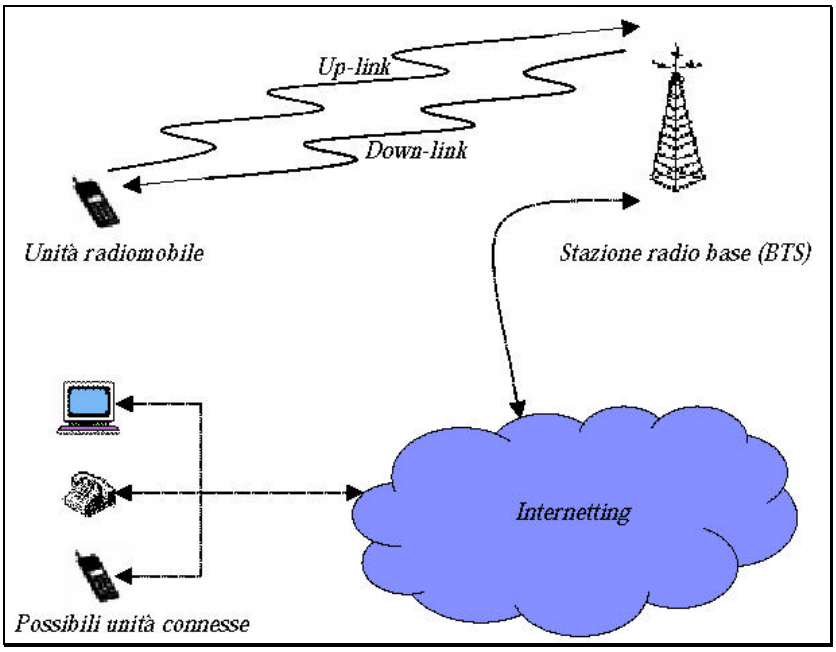

Figura 7.2 - Tipico scenario di connessione su rete mista wired-wireless

La connessione raffigurata avviene tra un *host mobile* ed un *host fisso*, messi in comunicazione tramite una opportuna *rete di trasporto*, costituita da *link cablati* in numero variabile e da almeno un *link wireless*, che collega l'host mobile alla *stazione radio base* incaricata di "servirlo".

In una rete di questo tipo (spesso definita *rete mista wired-wireless*), valgono le considerazioni già fatte sulle reti "all-IP": l'uso del protocollo IP potrebbe essere teoricamente limitato alla sola rete di trasporto, ma questo finirebbe inevitabilmente col limitare la flessibilità del sistema, imponendo l'adozione di tecniche di trasmissione, codifica e instradamento diverse, sul link wireless, a seconda del tipo di comunicazione effettuata (voce, dati, entrambi). Di conseguenza, la tendenza attuale degli enti di standardizzazione è quella di uniformare l'intera connessione, utilizzando il protocollo IP dalla sorgente fino alla destinazione.

Considerando, del resto, che la larghezza di banda del canale radio costituisce la risorse più limitata e al tempo stesso più costosa del sistema, è importante che ne venga ottimizzato l'uso. In quest'ottica, l'uso di una architettura basata su piattaforma IP non sempre si rileva ottimale.

Nel paragrafo 2.1.2 si è detto infatti che l'architettura TCP/IP è organizzata secondo una "struttura gerarchica a livelli", in cui ogni livello fornisce servizi a quello superiore, nascondendo però le modalità con cui li realizza. Il passaggio dei dati tra livelli adiacenti di uno stesso host avviene per mezzo dell'**incapsulamento** (figura 4.2): in trasmissione, ogni protocollo riceve dati (payload) dal protocollo di livello superiore e aggiunge ad essi una intestazione di controllo (header), dopodiché trasmette il tutto al protocollo di livello inferiore, fino al protocollo di livello fisico incaricato della trasmissione vera e propria. In ricezione, invece, avviene il processo inverso (**demultiplexing**).

In questo modo, sul canale fisico viaggia un pacchetto costituito in parte da *informazione d'utente* (payload) e in parte da *informazione di controllo* (corrispondente alla sequenza di header introdotti dai protocolli ai vari livelli), che risulta però "inutile" per l'utente finale. Questa caratteristica dell'architettura TCP/IP penalizza maggiormente quelle applicazioni che generano un payload di dimensioni confrontabili se non addirittura inferiori rispetto all'header: infatti, buona parte della banda richiesta dal servizio viene usata, in questi casi, per la trasmissione di informazione inutile per l'utente.

Fin quando il pacchetto deve essere trasmesso su un link cablato a larga banda e con buone se non ottime caratteristiche di rumorosità, il "peso" dell'informazione di controllo rappresenta senz'altro un problema trascurabile. L'esatto contrario avviene invece quando il canale impiegato è wireless, nel qual caso la banda non solo è limitata, ma è anche costosa per l'operatore che deve fornirla all'utente e quindi, di riflesso, anche per l'utente.

L'ambito nel quale questa "penalizzazione" dovuta all'header è più forte è senz'altro quello dei servizi di tipo real-time. Un esempio classico è il servizio

di voce su IP (*VoIP*, Voice Over IP): lo stack di protocolli su cui si implementa un simile servizio prevede l'uso dei protocolli RTP [26,27] e UDP [18] al di sopra di IP, producendo pacchetti di livello rete con header lungo complessivamente 40 byte (IPv4) o 60 byte (IPv6); a fronte di un header così lungo, il payload generato dall'applicazione a monte non supera generalmente i 20 byte ([7]), nel qual caso il 67% con IPv4 (80% con IPv6) della banda richiesta dal servizio viene consumata per la sola trasmissione dell'header.

Questo problema trova una valida soluzione nella **compressione dell'header**. Infatti, se è vero che l'informazione di controllo inserita negli header è necessaria per consentire al pacchetto di "viaggiare" nella rete in modo indipendente rispetto agli altri pacchetti, è anche vero che alcuni di questi header (dal livello rete in su) non esplicano alcuna funzione sul canale radio, per cui si può senz'altro pensare di ridurne il peso, tramite una forma opportuna di compressione, prima di inviare il pacchetto sul canale.

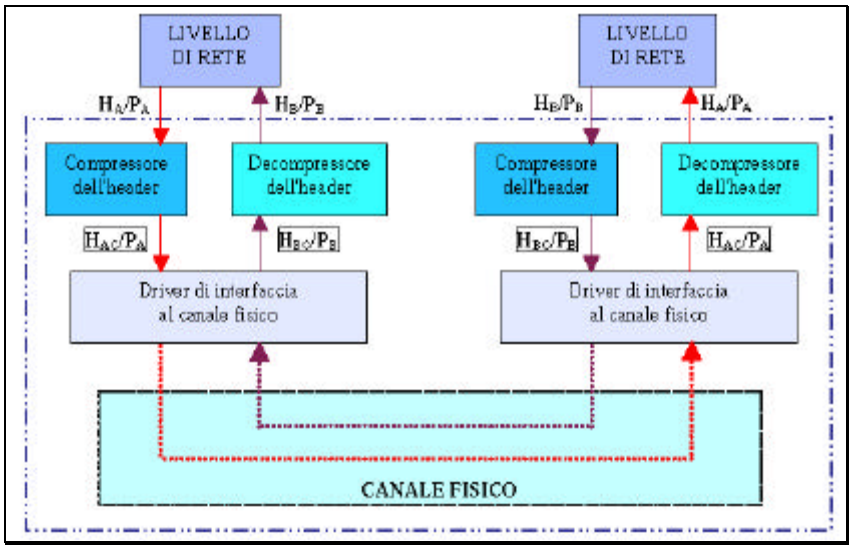

Figura 8.2 - Introduzione delle unità di compressione e decompressione dell'header nell'architettura TCP.

*P_A = payload generato dal processo A; H_A = sequenza di header (fino al livello rete) nel pacchetto IP finale; H_{AC}/P_A = pacchetto con header compresso generato dal compressore. La regione tratteggiata (**regione di comprimibilità dell'header**) è quella in cui gli header fino a livello rete non svolgono alcuna funzione*

Il compito di comprimere l'header prima della trasmissione sul canale viene affidato ad una nuova unità funzionale dell'architettura TCP/IP (figura 8.2), che

[7] Questo valore si ottiene ipotizzando un tasso di emissione dei pacchetti di 50 pkt/s e la generazione dei bit di payload al ritmo di 8 kbit/s.

si inserisce tra il livello rete ed il livello di interfaccia al canale: il **compressore dell'header**.

Dualmente, il compito di ricostruire l'header originale, partendo dalla sua versione compressa ricevuta dal canale, è affidato al **decompressore dell'header**.

I prossimi paragrafi sono dedicati ad esaminare più nel dettaglio i benefici ottenibili con la compressione dell'header, evidenziando l'importanza di ridurre l'overhead dell'header ma non solo. A partire dal capitolo successivo, invece, si esaminerà nel dettaglio l'algoritmo di compressione proposto in questa tesi.

2.2.1 Overhead dell'header nei pacchetti IP

Si è detto che la compressione dell'header risulta particolarmente attraente per i servizi real-time basati su IP, come ad esempio il VoIP, in quanto in tali servizi è molto frequente la generazione di pacchetti IP dove l'header ha un lunghezza uguale se non addirittura superiore a quella del payload.

In effetti, però, si ritiene che la compressione dell'header sia un valido ed opportuno strumento di ottimizzazione non solo per connessioni di tipo real-time, ma anche per connessioni basate su un protocollo di trasporto di tipo affidabile e connection-oriented come è TCP.

Ad esempio, si consideri una pila protocollare che preveda appunto l'uso del protocollo TCP al di sopra di IP, come riportato in figura 4.2: i pacchetti generati dal livello rete, in base all'incapsulamento, comprendono in questo caso un header TCP e un header IP, le cui dimensioni minime sono 20 byte per TCP e 20 byte per IPv4 oppure 40 byte per IPv6. In alcune connessioni, come ad esempio quelle per il protocollo TELNET, un segmento TCP può contenere anche un solo byte di dati, il che significa che l'overhead dell'header è prossimo al 100%. Lo stesso problema, sia pure in modo meno rilevante, si presenta anche per trasferimenti di file con il protocollo FTP: in questi casi, infatti, la dimensione tipica dei segmenti TCP è di 576 byte, sui quali quindi l'overhead ammonta al 6.9 % per IPv4 e 10.4 % per IPv6, che sono quantità ancora significative considerando le scarse risorse a disposizione.

L'entità dell'overhead dell'header diventa poi molto maggiore quando si considera il *tunneling*, ossia quel meccanismo per cui un pacchetto IP viene inserito in un altro pacchetto IP al fine di fargli seguire un percorso prefissato

all'interno della rete di trasporto ([8]): in questi casi, ciascun pacchetto consegnato al livello fisico comprende almeno due header di livello IP, con conseguente aumento dell'overhead rispetto ai valori citati prima. Ad esempio, un caso frequente di tunneling è quello che prevede l'incapsulamento di un pacchetto TCP/IPv4 in un pacchetto IPv6: in questo caso, la dimensione minima dell'header ammonta a 100 byte, che sale poi a 120 byte nel caso di TCP/IPv6/IPv6. Se i segmenti fossero lunghi 576 byte, l'overhead dell'header sarebbe del 17.4% nel primo caso e del 20.8% nel secondo.

L'algoritmo di compressione proposto in questa tesi è in grado di comprimere pacchetti che includano uno o due header a livello IP, raggiungendo valori di overhead molto vicini all'1%.

Overhead e occupazione di banda

Esiste uno stretto legame tra l'overhead dell'header nei pacchetti IP e la banda occupata per la loro trasmissione. L'overhead esprime, in termini percentuali, la quota costituita dall'header rispetto all'intero volume del pacchetto trasmesso. Esso è infatti definito come rapporto tra la dimensione, in byte, dell'header (H) e quella dell'intero pacchetto trasmesso, header + payload ($H+P$):

$$OVERHEAD\% = \frac{\mathrm{H}}{\mathrm{H}+\mathrm{P}} \cdot 100$$

Questa espressione mostra la prevedibile relazione di inversa proporzionalità tra l'overhead dell'header e la dimensione del payload.

Si può facilmente dimostrare che l'overhead rappresenta la quota della banda richiesta per la trasmissione dell'informazione inutile per l'utente finale, rispetto a quella totale richiesta dal servizio ($Banda_{TOTsc}$: banda totale senza compressione):

$$Banda_{TOTsc} = \frac{\mathrm{H}+\mathrm{P}}{\mathrm{T}} = (\mathrm{H}+\mathrm{P}) \cdot f_{\mathrm{r}} = \mathrm{H} \cdot f_{\mathrm{r}} + \mathrm{P} \cdot f_{\mathrm{r}} = Banda_{Header} + Banda_{Utile}$$

$$OVERHEAD\% = \frac{\mathrm{H}}{\mathrm{H}+\mathrm{P}} \cdot 100 = \frac{\mathrm{H}}{\mathrm{H}+\mathrm{P}} \cdot 100 \cdot \frac{f_{\mathrm{r}}}{f_{\mathrm{r}}} = \frac{Banda_{Header}}{Banda_{TOTsc}} \cdot 100$$

[8] Si veda, in proposito, quanto detto nel paragrafo 1.5.3.2

In base a questo semplice modello, l'overhead diventa, di fatto, un indicatore del grado di efficienza con cui si utilizza il canale, dato che, quanto più basso è il suo valore, tanto più la banda richiesta per la trasmissione degli header ($Banda_{Header}$) costituisce una quota poco significativa della banda totale richiesta dal servizio.

2.2.2 Influenza della compressione sul Frame Error Rate

Una seconda importante ragione che rende opportuna la compressione dell'header su sistemi misti wired-wireless, a prescindere dal protocollo usato a livello di trasporto, è la possibilità di ridurre il tasso di perdita dei pacchetti rispetto al caso di compressione assente. Infatti, dato che diminuisce il numero medio di bit inviati in ciascun pacchetto, per un BER (Bit Error Rate) prefissato il tasso di perdita dei pacchetti (FER, Frame Error Rate) risulta necessariamente inferiore. Questo determina un maggiore throughput della connessione, che, nel caso specifico del TCP, è dovuto fondamentalmente alla migliore crescita della finestra di trasmissione del mittente, conseguente al minor numero di ritrasmissioni.

2.2.3 Protocollo TCP su reti wireless

Le considerazioni del precedente paragrafo, circa i miglioramenti che la compressione dell'header può indurre su una connessione TCP grazie alla riduzione del FER, meritano di essere approfondite.

Le reti di telecomunicazioni più "tradizionali" sono quelle costituite da link cablati (*wired links*) e *host fissi*. Tipicamente, i link cablati hanno tassi di errore estremamente bassi: questo comporta che la stragrande maggioranza delle perdite di pacchetti sia causata da problemi di congestione della rete più che da errori di trasmissione. I protocolli di trasporto come TCP, almeno nelle versioni meno recenti, sono stati allora messi a punto per fornire ottime prestazioni proprio su reti di questo tipo, usando appositi meccanismi di adattamento ai ritardi end-to-end ed alle perdite di pacchetti.

Il mittente TCP deduce la perdita di un pacchetto sia quando arrivano diversi ACK con lo stesso valore dell'ACK Number (*ACK duplicati*) sia quando non arriva alcun riscontro per il pacchetto entro la scadenza di un *timeout* opportunamente calcolato. Una volta rilevata la perdita, TCP reagisce sia

ritrasmettendo il segmento ritenuto perso sia diminuendo l'ampiezza della sua finestra di trasmissione, intraprendendo l'azione di *congestion control* o *congestion avoidance* e resettando il timer di ritrasmissione. Nel fare questo, il TCP assume che le perdite siano dovute solo alla congestione: l'esito dei provvedimenti presi è dunque fondamentalmente quello di ridurre temporaneamente (salvo poi ad incrementarlo dinamicamente) il tasso di emissione dei pacchetti, in modo da ridurre il carico della rete ed annullare la congestione.

Tuttavia, quando i pacchetti vengono persi nella rete per ragioni diverse dalla congestione, come ad esempio gli errori di trasmissione, i meccanismi citati contribuiscono solo a peggiorare il throughput: infatti, la sorgente TCP riduce il numero di pacchetti in uscita nonostante invece la rete sia ancora perfettamente in grado di gestirli.

Un simile peggioramento è molto frequente nelle reti che utilizzano *link wireless*. Questi ultimi presentano caratteristiche radicalmente diverse dai link wired: la banda disponibile è generalmente minore, i RTT sono più elevati e, soprattutto, si manifestano sporadici ed alti tassi di errore sui bit. Di conseguenza, le prestazioni del TCP in queste reti risentono della significativa degradazione del throughput e della presenza di ritardi molto alti (che rendono più frequente la scadenza dei timeout).

Ulteriori complicazioni si aggiungono nelle reti che garantiscono agli utenti il requisito della *mobilità*, ossia la possibilità di mantenere attive le proprie connessioni nonostante essi continuino a muoversi lungo il territorio di copertura della rete: in questi casi, le perdite di pacchetti sono dovute non solo alla rumorosità dei canali, ma anche all'espletamento delle procedure di handover per il cambio di cella, durante le quali il canale può risultare completamente "assente".

Sono state proposte varie soluzioni in letteratura [28] per migliorare le prestazioni del TCP su reti di questo tipo. In questa tesi, non si è interessati alla loro analisi, ma si ritiene indiscutibile un fatto: qualunque sia la versione del TCP utilizzata, l'uso di un algoritmo di compressione dell'header non può che essere vantaggioso, visto che consente di ridurre il FER e quindi di "avvicinare" le caratteristiche reali della connessione a quelle teoriche che il TCP si attende.