Indice

CAPITOLO 1

Dispositivi di rete

1. Livello Fisico

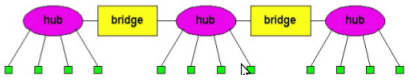
Repeater: Analogo all'*Hub*, ma ha solo un collegamento di input de output.

Hub: Sono dei *ripetitori multiporta* che operano a *livello fisico* (e quindi di segnale) che collegano i segnali di input verso tutti gli output. Questi hubs consentono inoltre l'interconnessione di LAN **omogenee** (ovvero adottanti la stessa tecnologia), creando quindi un'unica rete, ove ogni sottorete è identificata da un indirizzo unico: inoltre questa rete può essere gestita gerarchicamente ("multi-tier design") da una radice **backbone**.

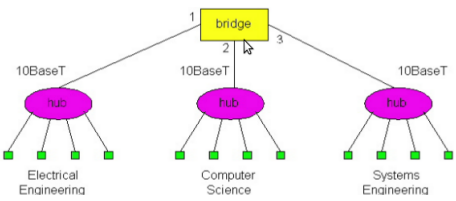
Uno dei vantaggi dell'applicazione di questa tecnologia, oltre ai costi limitati, è il *mantenimento di continuazione di servizio* (ovvero i "segmenti" continuano a funzionare) anche se un hub si guasta grazie alla riduzione graduale del servizio (*graceful degradation*) tramite il suddetto multi-tier design. Molti però sono i limiti di questa soluzione:gli hub non isolano le collisioni sul singolo segmento, creando quindi un singolo "dominio di collisione", e quindi in caso di errore tutti i segmenti attueranno in caso di conflitto la politica di *exponential backoff*. Inoltre, in questo modo non si aumenta il alcun modo la **bandwidth**, che invece potenzialmente con questa soluzione potrebbe essere moltiplicata, non permette l'interconnessione di tecnologie differenti, in quanto gli *Hub* non consentono la bufferizzazione dei dati (e quindi non possono essere gestite tecnologie a diverse velocità o con diversi pacchetti). Inoltre le limitazione di un singolo segmente comportano limitazioni a livello più globale nella rete.

2. Livello Datalink

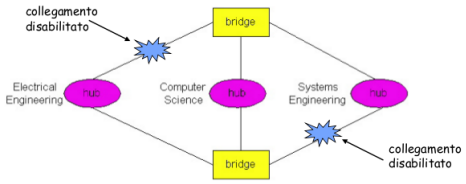
In questa sezione è presente unicamente il dispositivo **bridge**: si potrebbe quindi pensare una interconessione di LANs via bridge, ma in questo caso un singolo quasto potrebbe portare alla mancanza di comunicazione tra (es.) i due estremi della rete. Inoltre questa implementazione è inefficiente, in quanto se l'informazione deve avvenire tra i nodi estremi, essa passa anche per quelli intermedi che non hanno richiesto di ricevere.



Si potrebbe inoltre creare una *backbone bridge*, risolvendo sì il problema del traffico superfluo, ma un errore all'interno del bridge potrebbe comportare ad una impossibilità di effettuare la comunicazione tra tutte le reti locali



Una soluzione è quella di adottare un *bridge ridondante*, inserendo cammini alternativi (e quindi ridondanti) tra le varie sorgenti e destinazione collegati da bridges differenti; tuttavia, così facendo bisogna considerare il fatto che si possono creare dei cicli che possono provocare la replica e la trasmissione interminata dei frames. In questo caso si può risolvere il problema organizzando i bridges come in un grafico aciclico (tramite algoritmo di **spanning tree**, ovvero di albero di copertura: v. sezione 2 a pagina 45) diasbilitando alcune porte, in modo logico e non fisico. Essendo questo implementabile nel livello datalink, si possono in terconnettere reti differenti in quanto questi dispositivi sono in grado di tradurre i protocoli, introducendo tuttavia degli ovverhead di computazione.



3. Livello Network

Switch: gli switch trasmettono pacchetti a livello network tramite l'informazione di indirizzaemento, implementano la rete ed interagiscono tra reti omogenee che implementano gli stessi protocolli. Sono realizzati con hardware dedicato, e permettono l'ampliamento della rete permettendone la fruizione ad altri host, senza tuttavia interferire con le prestazioni degli host preesistenti.

Router: i routers invece sono utilizzati anche per connettere reti eterogenee, e quindi possono gestire differenze implementative di amministrazione. Una volta venivano chiamati "gateways", in quanto il loro unico compito era quello di tradurre i vari protocolli.

Gli switch inoltre hanno collegamenti d'entrata e di uscita ai quali possono essere connessi gli host, ovvero gli elaboratori ospiti. Essendo inoltre questi ultimi direttamente collegati ai *router*s, essi non si devono preoccupare della negozaiazione per l'accesso, ed in quanto hanno un rapporto dedicato, possono trasmettere con l'ampiezza di banda massima possibile. La loro interconnessione permette la creazione di reti geografiche, dove gli host sono identificati univocamente da un indirizzo globale.

Gateway: come descritto nella sottosezione 2.2.3 a pagina 30, è un particolare router IP che ha il compito, all'interno di un AS, di inoltrare i pacchetti ad altri AS

Parte 1

Codifica del segnale

CAPITOLO 2

Livello Fisico

A livello fisico possiamo analizzare unicamente la codifica del nostro flusso di bit come un segnale elettrico: da ciò segue che abbiamo differenti situazioni implementative:

NRZ (*Nonreturn to zero*): dato un certo segnale intermedio di base alla nostra decodifica detto *baseline*, associamo ad un segnale basso valore 0 ed ad uno alto un valore 1. Un problema di questa implementazione consta nel fatto che una serie di valori alti o bassi singolarmente prodotti in un tempo prolungato, portano ad un innalzamento della **variazione del valore di riferimento**, mentre nel secondo caso non si può decidere se il link o la sorgente sia guasta. Altro problema sorge dalla identificazione del ciclo di clock con il quale si sincronizzano le due macchine dialoganti: se il messaggio impiega un tempo eccessivo nella trasmissione, si può assistere alla *deriva*.

NRZI (*NRZ inverted*): si tenta di risolvere il problema della lunga sequenza di 1 invertendo il segnale tra due valori di 1 successivi

Codifica Manchester: Si effettua uno `XOR` tra valore del clock e valore del segnale in input, in modo da ottenere contemporaneamente valore del dato e ciclo di clock. Tuttavia con questo modo si raddoppia la velocità con la quale avviene la transizione del segnale, avendo quindi la metà del tempo per poterlo identificare: quando non avviene compunicazione quindi non avverrà alcuna transizione di segnale.

C. Manchester differenziale: 1 viene identificato ponendo la prima metà del segnale con un valore uguale a quella della metà del segnale precedente, mentre 0 la seconda metà del segnale avrà valore opposto a quello assunto dalla seconda metà del bit precedente.

4B/5B: Con questa codifica 4 bit vengono codificati come 5 (da cui il nome), in modo da avere alpiù un bit 0 in testa ed alpiù due bit in coda, in modo da inviare una sequena che non possiede più di 3 bit a 0 consecutivi: la sequenza così ottenuta viene poi codificata con NRZI. In quanto aggiungendo un bit aumentano le combinazioni possibili, alcuni valori assumono significati particolari, come `11111` che identifica una linea attiva (*idle*) mentre `00000` indica una linea fuori uso ed `00100` porta ad una interruzione del segnale (*halt*).

Per quanto riguarda invece la trasmissione wireless del segnale, si possono implementare due tecniche in modo da evitare le collisioni con altri segnali:

frequency hopping: il mittente invia al ricevente un *seed*, in base al quale il ricevente, conoscendo il valore del *seed*, può sintonizzarsi sulle frequenze casuali e variabili sulle quali trasmette la sorgente.

direct sequence: il mittente aggiunge ridondanza ai dati, dove ogni bit viene ripetuto n volte; i bit così ripetuti vengono trasmessi dopo aver effettuato lo `XOR` bit a bit con una sequenza pseudo-casuale creando così il **chipping code** ad n bit.

CAPITOLO 3

Livello Datalink (Reti a connessione diretta)

1. (b) Risoluzione del framing (Comunicazione punto-punto)

Si vuole riconoscere all'interno della sequenza di bit l'informazione da elaborare o comunque da analizzare. A questo livello lo scambio di bit viene percepito come lo scambio di blocchi di dati chiamati *frame*, che sono composti da una sequenza di campi contenenti un contenuto informativo con una propria semantica in riferimento alla sintassi di rappresentazione...

BISYNC: Il problema di framing viene risolto grazie all'utilizzo di un campo iniziale di `SYN` di 2 byte. Segue il campo `SOH` che indica l'inizio del campo `header`. Il campo del testo è racchiuso da i campi `STX` e `ETX`, che all'interno del campo testo è disambiguato da un *data-link escape* `DLE`. Inoltre è presente un campo `CRC` utilizzato per controllare la correttezza

PPP: Questa versione, più recente della precedente, è utilizzata per il trasporto di pacchetti IP. La sentinella (`Flag`) è caratterizzato dal campo con valore `01111110`. I campi `Address` e `Control` contengono dei campi predefiniti, in particolare il primo contiene valore `11111111` in quanto con questo valore tutte le stazioni possono accettare il frame, mentre di solito la comunicazione avviene solamente tra un emittente ed un singlo ricevente; il secondo campo vuol dire la modalità di controllo, ad esempio un valore `00000011` significa la disattivazione di ogni forma di controllo, ritenendo il contenuto affidabile. Nel campo `Protocol` si identifica il tipo di protocollo al livello superiore al quale è destinato il messaggio (ovvero **demultiplexing**). Anche in questo caso è presente un campo di **Checksum** di dimensione 2 byte (default), o 4. La dimensione del `Payload` è 1500 byte, anche se può essere negoziato. La dimensione degli altri campi può essere negoziato con il protocollo **LCP** come illustrato in Figura 1.

DDCMP: Questa implementazione utilizza invece di campi sentinella per identificare l'inizio e la fine di un file, memorizza all'interno di un campo specifico la lunghezza del campo `Body`, in questo caso all'interno del campo `Count`, ma in questo modo un

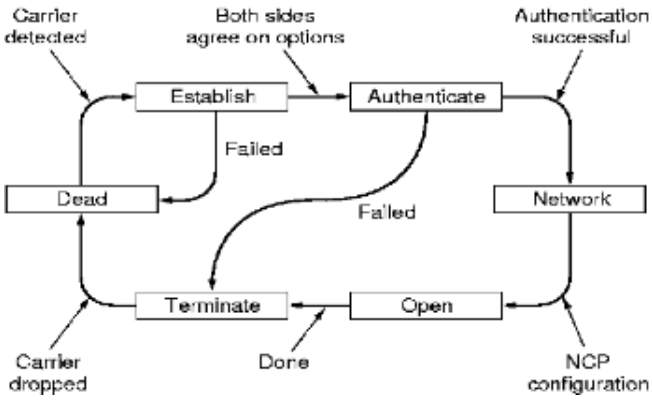


FIGURA 1. Protocollo *LCP* per stabilire la connessione PPP

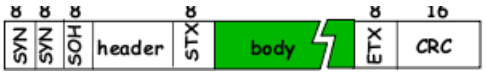
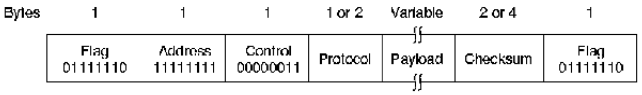
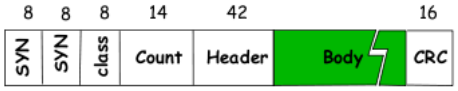
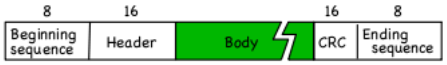
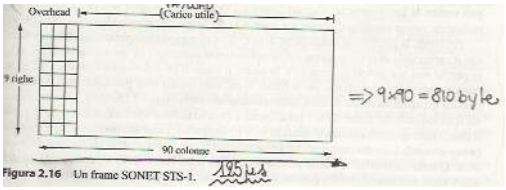
Nome	Livello	Rappresentazione
BISYNC	Orientati ai byte e sentinella	
PPP	Orientati ai byte e sentinella	
DDCMP	Orientati ai byte e conteggio b.	
HDLC	Orienati ai bit	
SONET	Orientati al clock	

TABELLA 1. Tabella dei protocolli incentrato sulla risoluzione del problema del framing

errore del campo può comportare una errata decifrazione della lunghezza effettiva del messaggio, metabolizzando quindi eventuali altir frame differenti dal corrente.

HDLC: Questa è la standardizzazione del protocollo SDLC sviluppato dalla IBM, e si propone di considerare l'informazione come una sequenza di bit. Come nei precedenti, è presente una sequenza iniziale ed una finale identici di valore, ovvero `01111110`; tuttavia bisogna disambiguare delle sequenze analoghe presenti nel campo con il **bit stuffing**: data la sequenza $011111xy$ vogliamo poi riconoscere se questa sia una sentinella nel seguente modo:

$$\begin{cases} x = 0 & \text{bit stuffing} \\ x = 1 & \begin{cases} y = 0 & \text{End Of Frame} \\ y = 1 & \texttt{Errore} \end{cases} \end{cases}$$

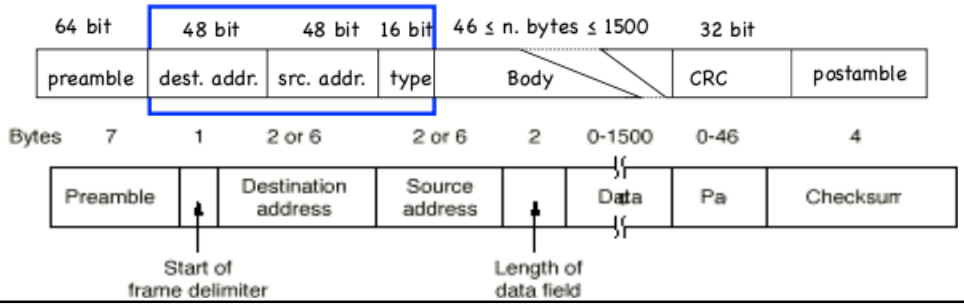
SONET: In questo caso la dimensione del dato dipende direttamente dalla velocità della linea (es.) telefonica: ad esempio una linea STS-1 opera a $51.84Mbps$, è possibile inviare in un tempo di $125\mu s$ 810B, dove i primi 3 byte di ogni *frame* costituiscono un **overhead** informativo del messaggio.

2. (e) Media Access Control (canale condiviso)

In questo caso si tenta di risolvere il problema di stabilire una connessione tra diversi componenti che utilizzano uno stesso mezzo di comunicazione, ovvero si vuole effettuare un *controllo di accesso al mezzo*.

2.1. Ethernet (Standard ed IEEE 802.3). In particolare tratteremo della rete **Ethernet**. La rete è strutturata nel seguente modo

- Tutti i nodi (o meglio il *transceiver* al quale è collegato direttamente l' *adattatore di rete* degli Host) riescono a riconoscere se la linea è inattiva (*idle*) o se questa è occupata (*busy*), "ascoltando" il segnale che viene trasmesso nel mezzo.



Campo	Dimensione (byte)	Descrizione
Preamble+SFD	7+1	Il preambolo serve per sincronizzare i due computer con il ciclo di clock prima di ricevere la sequenza seguente dei byte uguali `10101010`. Segue il delimitatore di inizio frame, `10101011`, che identifica una sequenza corretta
DA/SA	2/6	DA è il campo di destinazione del messaggio: mettendo questo campo inizialmente, il ricevente può decidere di scrtare il messaggio se non è ad essa indirizzato. SA è invece il campo mittente
Tipo/Lunghezza	2	Originariamente (*Ethernet*) indicava la chiave di demultiplexing per il livello superiore che aveva generato il Payload, mentre nello *IEEE 802.3* esso identifica generalmente la lunghezza in byte del campo dati
Payload	46…1500	Il campo dati deve contenere alpiù 1500*byte* ed almeno 46 per garantire la dimensione di 512*bit* minima come definito sopra: per garantire questa dimensione inferiore, viene riempito con una sequenza di riempimento detto **padding**, *che non è supportato dallo "standard Ethernet", dove quindi l'informazione deve essere di 46 bit; in quanto esiste questo campo, si può trasmettere anche un solo bit in "IEEE 802.3".*
CRC	4	É appunto un campo di checksum per controllare la correttezza del pacchetto informativo.
Postamble	1	*É presente solamente nello "standard Ethernet", non necessaria in "IEEE 802.3" in quanto la lunghezza è definita dal campo omonimo.*

FIGURA 2. Formato *(a)* "standard Ethernet" ed *(b)* "Ethernet IEEE *(802.3)*". É trasmesso in codifica *Manchester*

- Inoltre il nodo, rimanendo in ascolto, può eventualmente rilevare i conflitti nella rete, e quindi eventualmente nella trasmissione del pacchetto.

Bisogna sottolineare tuttavia che la parte algoritmica è implementata all'interno dell'adattatore e non nel transceiver. Ora invece è utile conoscere a fondo quali sono state le motivazioni per le quali si sono scelte particolari specifiche per questo sistema di comunicazione:

- Tutti i nodi condividono lo stesso *dominio di collisione*, in quanto la comunicazione viene effettuata in **broadcast**, e conseguentemente competono per l'accesso allo

stesso mezzo fisico da cui, se (es.) due sorgenti stabiliscono contemporaneamente che la linea è libera, possono emettere la comunicazione contemporaneamente. Si verifica la collisione quando l'emittente verifica che il dato ricevuto è differente da quello inizialmente inviato.

(1) Quando due nodi emittenti sono vicini, vengono trasmessi 96 *bit*, 64 *bit* di preambolo e 32 *bit* di *jamming sequence* quando viene rilevata la collisione; poi l'emittente che si è accorta dell'errore interrompe la comunicazione.
(2) Nel caso medio verranno quindi trasmessi almeno 96 *bit*.
(3) Nel caso in cui i due nodi si trovino agli antipodi della rete, bisogna valutare che:
   - Più distanti sono i nodi, più è il tempo necessario per raggiungere l'altro nodo, e quindi maggiore è il tempo per il quale la rete è sensibile alla collisione
   - Detto d il tempo impiegato a percorrere la rete per raggiungere l'estremo opposto, il tempo impiegato per ricevere il segnale di errore dopo l'emissione del pacchetto avverrà in un tempo $2d$. Scegliendo ora la lunghezza della rete di $2500m$, si stima un RTT di $51.2\mu s$, da cui segue che con una rete a $10Mbps$ corrispondono a 512 *bit*.v

da ciò segue che la dimensione del pacchetto deve essere quella di 512 *bit*.

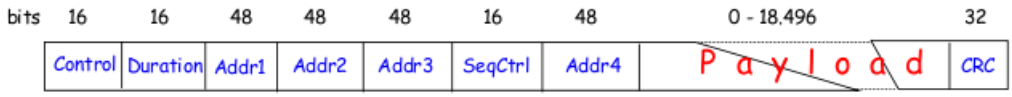
Ora nello specifico possiamo preoccuparci di alcune caratteristiche particolari di questa rete: ogni emittente può essere caratterizzato da 10Base$x$, dove:

(1) la rete ha una velocità di $10Mbps$
(2) *Base* sta per **sistema in banda base**, ovvero tutte le stazioni collegate trasmettono su di un'unica frequenza, ed ogni stazione partecipa alla comunicazione tramite *listening*
(3) la rete è formata da segmenti collegati da ripetitori (non più di 4), chiusi da un terminatore che assorbe il segnale, lunghi $x \cdot 100m$. Quando invece $x =$T, si ha un cavo *twisted pair*, ovvero un cavo in "doppino intrecciato".

Facendo riferimento alla tabella 2 nella pagina precedente, si può notare che per la prima volta parliamo di indirizzo della macchina alla quale è destinata il messaggio: questi sono "indirizzi macchina" detti **indirizzi MAC**, che identificano cioè la scheda di rete all'interno della rete locale: se il primo bit è `0` allora il messaggio è destinato ad un definito calcolatore, mentre se è `1` esso è diretto ad un gruppo di calcolatori, che diventa **broadcast** se nell'indirizzo sono presenti tutti valori `1`. Inoltre, per rendere univoco l'indirizzo MAC, alle ditte costruttrici dell'HW viene assegnato un prefisso MAC specifico, mentre l'altra parte dell'indirizzo é lasciato alla discrezione della ditta. Inoltre ogni volta che la trasmissione fallisce, si utilizza un tempo detto **backoff esponenziale**, con il quale si raddoppia continuamente il tempo di attesa rispetto al tempo atteso precedentemente, fino ad un massimo di 16 tentativi.

2.2. Wi-Fi (802.11). Trattiamo ora brevemente del formato di codice utilizzato nel campo del Wi-Fi: possiamo creare una rete dove la comunicazione avviene tra alcuni nodi mobili e degli **access point**, i quali sono collegati direttamente alla rete in qualche modo, fornendo così il servizio ai vari nodi mobili. La tecnica per la quale viene scelto un punto di accesso si chiama *scansione* e può essere di due modi:

S. passiva: viene chiamata in questo modo perhcè sono gli stessi punti di accesso ad inviare periodicamente un `beacon` che fornisce delle informazioni sulle sue caratteristiche, potendo in questo modo cambiare AP, risponendo con una `Association request`.



Campo	Dimensione (bit)	Descrizione
Control	16	Contiene tre campi: **Type:** Indica se il frame trasporta dati, se è un frame RTS o CTS o se è usato dall'algoritmo di scansione **ToDS, FromDS:** Indica se il frame deve essere inoltrato attraverso il sistema di distribuzione, e per identificare che il reale mittente non è il mittente che ha inviato il messaggio più recentemente.
Addr1,Addr2	48+48	Se entrambi i DS hanno bit 0, indicano rispettivamente l'indirizzo destinatario e mittente
Addr3,Addr4	48+48	Se entrambi i DS hanno bit 1, allora Addr1 rimane l'indirizzo di destinazione, Addr4 è l'indirizzo mittente, Addr2 è quello del trasmettitore attuale e Addr3 è la stazione intermedia alla quale deve essere inoltrato.
SeqCtrl	16	Identifica il numero della sequenza del frame.

FIGURA 3. Formato frame WiFi *(802.11)*

S. attiva: viene chiamata in questo modo perché viene ricercato attivamente il punto di accesso:
(1) il nodo effettua un messaggio di `Probe`[1] a tutti gli access point.
(2) gli Access point disponibili rispondono con un `Probe Response`.
(3) il nodo invia all'AP prescelto un `Association Request`
(4) se viene accettato, si risponde con un `Association Response`

Trattiamo ora per le reti wireless il problema della gestione dell'errore di inoltro del messaggio, come fatto per il protocollo Ethernet: anche in questo caso si vogliono evitare le collisioni tra i pacchetti che possono essere i seguenti:

- per il problema del **nodo nascosto**, se A e C sono collegati a B ma non si conoscono, in quanto i loro segnali non li raggiungono, potrebbero inviare a B il pacchetto ma non sapere che entrambi stanno trasmettendo a B, causando quindi una collisione tra i pacchetti;
- per il problema del **nodo esposto**, se B sta inviando ad A, C può comunque trasmettere a D in quanto i pacchetti non interferiranno tra di loro;

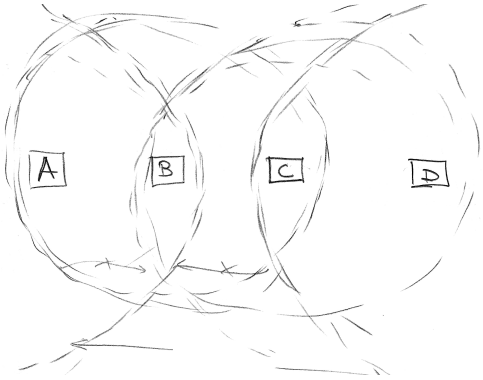
Per risolvere questi problemi è stato introdotto il protocollo **MACAW** strutturato nel seguente modo:

◇ Avviene uno scambio di messaggi tra mittente e destinazione, un mittente invia la `RTS` ("request to send") mentre il ricevente invia la `CTS` ("clear to send").
◇ Se un nodo vede il `CTS`, non potrà trasmettere in quanto i pacchetti potrebbero interferire alla ricezione del destinatario, ed allora per questo turno non potrà trasmettere.

[1](en.) esplorazione

◇ Se un nodo vede solo `RTS`, non interferirò con la trasmissione al mittente in quanto non si interferirà in alcun modo con l'altro ricevente, e quindi potrò inviare

Si identifica in questo caso la collisione se dopo l'invio di una `RTS` non si riceve il `CTS`: tale protocollo è presentato qui sotto:



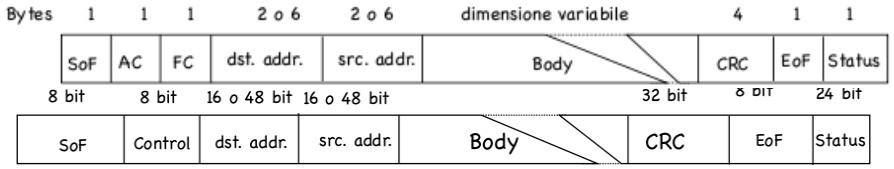
3. Reti ad anello ~ Token ring (802.5)

In questa implementazione si vuole creare una rete ove i dati circolano sempre in un'unica direzione, ed in quanto la comunicazione è visibile a tutti i frame, il cui contenuto è però copiato solo dal nodo di destinazione. Questi rete sono dette *token ring* in quanto il messaggi circolano incapsulati in un *token*, ovvero un testimone, che ciascun nodo riceve e trasmette: il nodo invece che aveva inviato il messaggio lo elimina dalla rete e vi inserisce il testimone. In questo modo si ha un modo *fair* di accesso al mezzo, in quanto ognuno a turno ha la possibilità di trasmettere; il ricevente può essere un singolo nodo, oppure può essere fatta in multicast o in broadcast. Il problema della trasmissione viene gestito dal THT (**token holding time**), il quale specifica il tempo durante il quale il nodo può inserire nel testimone tutte le informazioni che esso vuole inviare nel campo `Body`: tuttavia se si attribusce un tempo indeterminato a questo valore, si causerà una possibile starvation, oppure dare maggiore priorità ai nodi che vogliono inviare un maggiore numero di messaggi: questo viene risolto nel campo `AC` gestendo la priorità. Tuttavia da questa definizione può seguire che il testimone può essere smarrito o non essere presente perché:

(1) non è stato generato all'atto dell'inizializzazione
(2) l'alterazione di un bit lo rende irriconoscibile
(3) si è verificato il crash del nodo che detiene il token

É compito quindi di una stazione monitor verificare l'integrità del testimone, eventualmente di rimpiazzarlo, ed annuncia periodicamente la sua esistenza; mentre se questo messaggio non è prevenuto allora la prima stazione che verifica la sua assenza dopo un tempo $nstazioni \cdot THT + Latency$, emette un **claim token**, annunciando l'intenzione di diventare monitor; tuttavia se un altro nello stesso istante arriva alla stessa conclusione, allora è necessario dirimere la questione secondo una regola (es. l'indirizzo più alto vince).

Altro compito del "monitor" è quello di eliminare gli elementi corrotti, o quelli orfani che non possono più essere rimossi dalla emittente: quando il pacchetto viene emesso ha il `bit monitor` settato a zero, e quando questo passa tramite il monitor viene settato ad 1; conseguentemente, se al passaggio successivo il valore assunto è ancora 1, allora il pacchetto viene rimosso. Con l'implementazione della **FDDI** si migliora lo standard precedente creando una rete con un doppio anello, dove quello interno ha la funzione di backup se l'anello principale subisce una avaria. Con questa rete ogni nodo stima il proprio periodo di rotazione del testimone (**TRT**, token rotation time): se il testimone arriva in un tempo superiore alla stima, allora effettua una propria *claim token*, e la scelta del nuovo "monitor" avverrà per stima minore del suddetto tempo.



Campo	Dimensione (byte)	Descrizione *(con FDDI)*
SoF	1	Delimitatore di inizio frame
AC	1	Definisce il controllo all'accesso in questi termini: (1) priorità del frame (2) *prenotazione di priorità*: Il monitor tramite questo campo segnala la sua presenza con il valore `0000101`. *Non è presente nel frame* ***FDDI***
FC	1	Serve per distinguere il frame dati da quello di controllo, ovvero è una chiave di demultiplexing che identifica il protocollo di livello superiore. *Con FDDI, il primo bit se vero identifica un traffico sincrono, altrimenti asincrono; il secondo bit se 0 ha indirizzi a 16bit, altrimenti 48. I bit da 3 ad 8 identificano la chiave di demultiplexing.*
Dest/Src Addr	2/6	Campo indirizzo ricevente/mittente
Body	...	Ha valore *(a)* 4464 per il "Token Ring", e *(b)* 4532 per l'"FDDI".
CRC	4	Campo checksum
Eof	1	Delimitatore di fine frame
Frame status	1 ~ 3	Utilizzando i due ultimi bit, si può creare un servizio di consegna affidabile: detti `A` e `C` questi due bit, si ha $\begin{cases} \texttt{A=C=0} & \textit{init, guasto} \\ \texttt{A=1 C=1} & \textit{accettato} \\ \texttt{A=1 C=0} & \textit{respinto} \\ \texttt{A=0 C=1} & \textit{impossibile} \Rightarrow \textit{guasto} \end{cases}$ *Il campo ha valore 3 in FDDI*

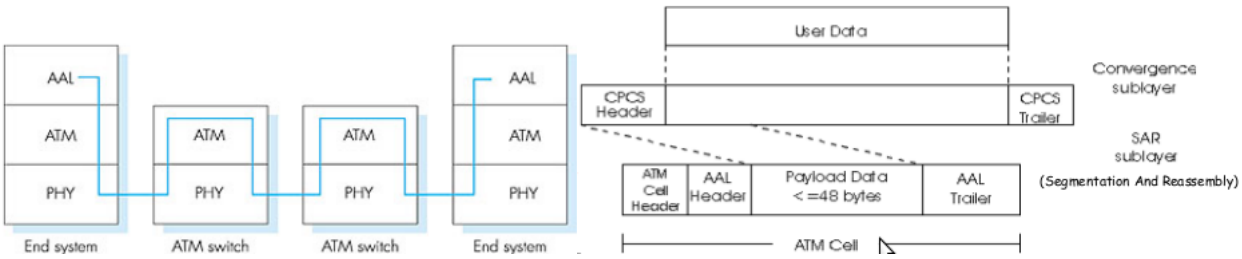
FIGURA 4. Formato *(a)* Token ring (802.5) e *(b)* FDDI

4. Asyncronous Transfer Mode (ATM)

Questa tecnologia è definita *commutazione di pacchetto orientata alla connessione*[2] ovvero instaurare tramite gli switch che collegano la rete in modo da permettere la comunicazione ai vari host, di instaurare tra questi ultimi una comunicazione virtuale punto-punto tramite un **circuito virutale** (VC) creato in una prima fase della comunicazione chiamato **connection setup**, dopo la quale inizia il vero e proprio trasferimento dei dati (**data transfer**).

Mentre per quanto abbiamo visto precedentemente i pacchetti potevano assumere una dimensione variabile con determinati valori di massimo e minimo, qui assistiamo ad un pacchetto, chiamato in questa sede **cella**, che hanno una dimensione fissa e che consente

[2]E quindi facciamo riferimento alla sezione di Network unicamente incentrata a tale commutazione di pacchetto



(A) *Architettura dei protocolli ATM.*

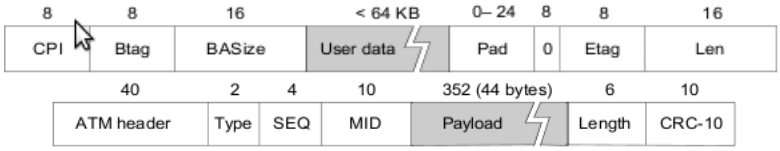
(B) *I dati utente passando dal livello AAL per essere poi frammentato nelle celle ATM.*

FIGURA 5. Analisi dei protocolli ai differenti livelli di ATM

eventualmente la suddivisione della informazione di dimensione maggiore in vari sotto-pacchetti.

- **Perchè utilizzare celle a dimensione fissa?**
  (1) In primo luogo può essere un modo per ottenere *throughput* elevati, poichè molti dispositivi di rete non sono limitati dalla dimensione dei pacchetti, ma dal loro numero. In seconda istanza, avendo pacchetti a dimensione fissa è più facile creare un Hardware scalabile, si può semplificare la loro gestione e la loro elaborazione è più semplice se ne conosciamo la dimensione. Inoltre, in quanto la dimensione è prefissata si può meglio gestire il parallelismo, quando si hanno molti elementi che effettuano la stessa operazione.
  (2) Si può calcolare esattamente il tempo di trasmissione e se il traffico è soggetto a vincoli stringenti di tempo, i pacchetti (es.) a maggiore priorità possono essere trattati immediatamente. Se si hanno invece dei pacchetti a dimensione variabile da trasmettere contenuti all'interno di code, il tempo di trasmissione di questi varierà molto a seconda della loro dimensione, ritardando eventualmente l'emissione di pacchetti più "leggeri" (es. se questi sono a maggiore priorità ma si stà trasmettendo un pacchetto di dimensioni maggiori a priorità minore, il primo dovrà attendere comunque molto tempo prima che il processo di invio sia terminato).
  (3) Conoscendo che vi saranno pacchetti di dimensione inferiore nelle code, in quanto dati di dimensione maggiore vengono suddivisi in queste celle, invece di effettuare l'invio solamente all'ultimazione della lettura del pacchetto entrante, possiamo trasmettere l'informazione già alla terminazione della ricezione della prima cella: in questo modo tendenzialmente si effettua una diminuzione della lunghezza delle code che tende a snellire il traffico smistando, grazie anche alla possibilità di effettuare più agevolmente attività in parallelo.
- **Qual è la dimensione giusta dei pacchetti?**
  L'unico difetto di questo sistema è che se si vuole inviare un singolo byte, è necessario effettuare un padding sufficiente per coprire i bit inutilizzati. Inoltre, se (come in questo caso), si sceglie una dimensione che non sia un multiplo di 2, risulterà difficile la gestione di questi nella memoria dei computer, come nelle pagine e nelle righe della memoria cache.

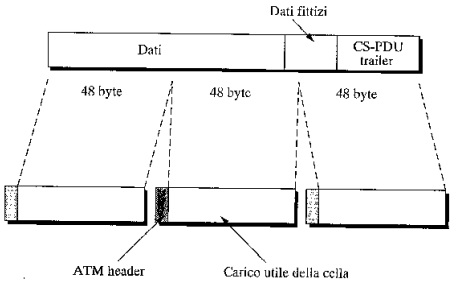
Il processo che abbiamo introdotto sopra per frammentare un messaggio (*PDU*) proveniente da un protocollo sovrastante è definito **segmentation and reassembly** (SAR, supportata dallo **strato di adattamento** `ATM 3/4`) ed in particolare, all'interno di AAL sono presenti le informazioni necessarie a ricreare il messaggio iniziale. In particolare vennero ideati AAL



Campo CS-PDU	Dimensione (byte)	Descrizione
CPI	1	Specifica la versione di CS-PDU in uso (per ora 0)
Btag/Etag	1	Assieme al corrispondente Etag, servono per delimitare l'inizio dalla fine del pacchetto, in modo da prevenire fusioni incontrollata con i pacchetti precedenti o/e seguenti
BAsize	2	Non deve essere per forza della dimensione del PDU, ma suggerisce la dimensione da utilizzare nel buffer per la ricostruzione del messaggio.
Userdata	<64*KB*	Payload PDU
Pad&Zerobyte	1…4	Assicurano l'allineamento a 32*bit* del campo che segue (trailer)
Len	2	Specifica la lunghezza del PDU contenuto nell'**Userdata**

Campo SAR-PDU	Dimensione (bit)	Descrizione
ATM header	40	
AAL header	2+4+10	Questo campo è formato da tre celle: **Type:** Identifica se la cella è l'inizio del messaggio (`BOM`: Begin Of Message), se sia una cella intermedia (`COM`), la sua fine (`EOM`), o essa rappresenti un unica cella costituente il messaggio (`SDU`: Single data unit). **SEQ:** Per identificare il numero di celle per il loro ordinamento o per controllare se precedentemente sono avvenute delle perdite **MID:** Ovvero *message ID*, usato per effettuare il multiplexing di più messaggi in un'unica connessione.
Payload	44*bytes*	Contiene il frammento di CS-PDU
AAL trailer	6+10	Mentre il primo campo `Lenght` contiene la lunghezza del messaggio, il secondo offre una protezione debole, poiché si presume che la rete sia affidabile e quindi che la possibilità di un errore sia remota

FIGURA 6. Strati di adattamento ATM 3/4: **Convergence Sublayer** (CS-PDU) e **Segmentation/Reassembly Sublayer** (SAR-PDU)



(A) *Incapsulamento e segmentazione.*

Campo	Dimensione (bit)
Data	<64*KB*
Pad	0...47
Reserved	16
Len	16
CRC-32	32

(B) *Formato del Pacchetto.*

◇ `Pad`: Per far in modo che il campo trailer sia in fondo al pacchetto.
◇ `Reserved`: Per ora zero.

FIGURA 7. ATM 5

differenti in modo da fornire, in base al tipo di dato che si voleva trasferire, servizi differenti, mappato poi completamente in una cella dedicata. Nello specidico di `AAL 3/4`, avviene una notevole incrementazione della quantità di informazioni che viene aggiunga a ciascuna cella a causa dell'incapsulamento effettuato in questa maniera.Parlando ora dello **strato di adattamento** `ATM 5`, il suo pacchetto è delineato dalle sezioni delineati nella figura 7, mentre per quanto riguarda l'`ATM header` *AAL 5* rimpiazza il campo `Type` di 2*bit* con un singolo bit, che quando è settato ad 1, identifica l'inizio del nostro frame, dove i seguenti saranno evidentemente conseguenti del primo, mentre l'ultimo è segnalato impostando il bit di `user signaling` ad 1.

CAPITOLO 4

Livello Network

A questo punto vogliamo introdurre un protocollo che permetta il collegamento tra reti di tipologie diversa, ovvero un **internetwork** (brevemente *internet*, da non confondere tuttavia con *Internet*), dove viene fornito un servizio di consegna di pacchetti tra host all'interno di una "rete di reti" più piccole: parliamo quindi di una sezione di **interlink**, ovvero di intercollegamento globale delle reti.

1. Concetti fondamentali: Datagrammi

Un datagramma è un particolare pacchetto che viene utilizzato all'interno delle reti connectionless per creare un servizio di consegna "non affidabile" dei suddetti; possono essere eventualmente frammentati, diventando quindi "pacchetti"[1]. Inoltre, differentemente ai pacchetti definiti nei livelli precedenti, in qualunque rete questo pacchetto transiti, deve essere possibile che ogni **switch** possa decidere come inoltrare l'informazione, facendo riferimento ad una **tabella di routing ovvero forwarding**, e quindi di inoltro del messaggio. La struttura è inoltre di tipo **best effort** in quanto sono implementate le seguenti caratteristiche:

(1) L'host mittente non sa in anticipo se il pacchetto giungerà o meno a destinazione, nè se l'host ricevente sia o meno operativo
(2) Non viene garantito al ricevente l'ordine di consegna dei pacchetti, che si dovrà preoccupare di riordinarli; inoltre questi potranno seguire anche percorsi differenti per giungere a destinaizone: conseguentemente il *breakdown* di un switch potenzialmente non comporta grossi rischi sulla consegna.

Da ciò segue che, se il pacchetto inviato non è stato consegnato, la rete non farà nulla per rimediare al problema. Questa implementazione segue la logica `KISS`, e quindi è sempre possibile garantire eventualmente ad un livello superiore la garanzia della consegna dei pacchetti. Da ciò che è stato detto precedentemente segue inoltre potrebbe essere possibile che uno stesso pacchetto possa essere consegnato più volte a destinazione.

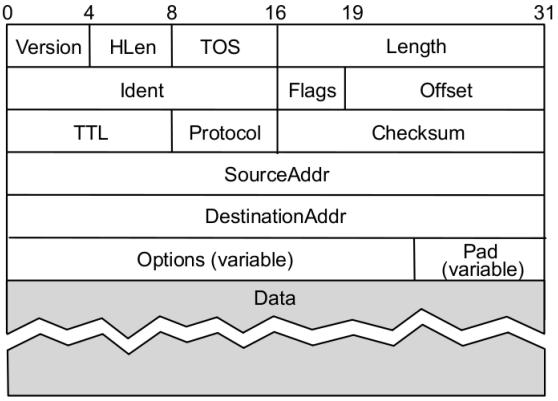
In particolare, quando si fa riferimento al "datagramma", spesso si fa riferimento al *datagramma IP*: anche se questo è ora diventato uno standard *de facto*, ciò non vuol dire che questo debba essere l'unico datagramma possibile.

2. Protocollo IP (IPv4 e IPv6)

Come si può notare dalla Tabella 1 a pagina 21, ogni rete ha una propria *MTU* (Maximum Transfer Unit), e quindi di dimensione del campo `Payload`: da qui la seguente soluzione, suddividere il dato IP tramite l'invio, nel protocollo sottostante, di più frame contenente il datagram sezionato. Durante l'invio dopo la frammentazione:

- se dovesse mancare qualche pacchetto durante la ricezione, l'host scarta tutti i pacchetti ed il protocollo IP non tenta di recuperare i pacchetti perduti.
- il processo di frammentazione può essere ripetuto anche successivamente se l'*MTU* è inferiore.

[1] Kurose, James F. & Ross, Keith W. (2007), Computer Networking: A Top-Down Approach. *en.Wikipedia*

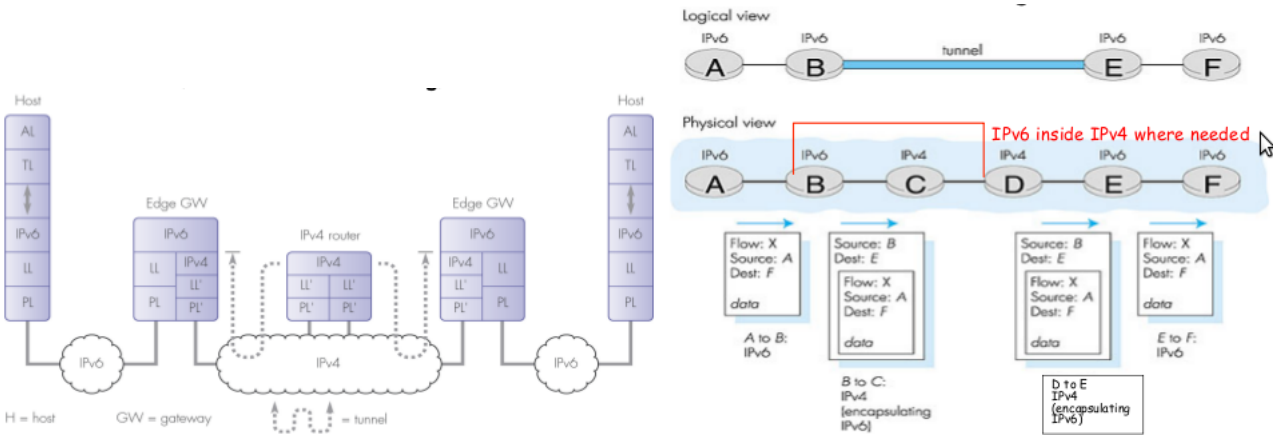


Campo	Dimensione (bit)	Descrizione
Version	4	Indica la versione utilizzata del pacchetto, attualmente viene utilizzata la 4 e la 6.
HLen	4	Indica la lunghezza dell'intestazione, che arriva fino al campo **Pad**., con indicazione a `word` di 32*bit*.
TOS	8	Fornisce una differenziazione tra i vari pacchetti e (conseguentemente) un trattamento diverso degli stessi in base al valore contenuto, ad esempio fornire servizi differenziati in base al valore dei pacchetti ricevuti a più alto livello.
Length	16	Indica la lunghezza di tutto il datagramma, e quindi la sua lunghezza ha valore massimo di $2^{16} - 1 = 65535$*byte*: in quanto la rete può non supportare pacchetti di queste dimensioni, vengono fornite informazioni sulla frammentazione e ricostruzione.
Ident	16	Identificativo del messaggio, con stesso valore per tutti i datagram correlati.
Flags	3	In particolare il campo `More`, viene impostato a `1` se seguono frame, altrimenti `0`.
Offset	13	Indica il numero dei dati inviati precedentemente, e quindi l'offset in byte rispetto al primo bit inviato.
TTL	8	Indica il tempo massimo per il quale il pacchetto può circolare, oltre il quale è necessario eliminare il datagram poichè ritenuto inutilizzato: inizialmente veniva contato il tempo di permanenza all'interno di un router in secondi, successivamente (ed ora) viene contato i **nexthop**, ovvero il nuemro di router attraversati.
Protocol	8	Indica il protocollo al livello superiore al quale deve essere consegnato il pacchetto IP.
Checksum	16	Questo campo contiene la somma di controllo dell'intera intestazione, da non confondere con un qualsiasi algoritmo di CRC: in quanto questo errore compromette l'indirizzo di destinazione, un pacchetto compromesso deve essere eliminato.
Source/Dest.Addr	32	Queste informazioni sono entrambe essenziali, perché il secondo serve ai routers per decidere sull'invio del datagram, mentre il primo per decidere se accettare tale datagram.
Options+Pad	0/32	Il primo è un campo che estende con informazioni dei campi precedenti. Quando queste informazioni sono assenti, l'intero header occupa una dimensione di 20*byte*, ovvero 5 parole di 32*bit*.
Payload	65535	...

FIGURA 1. Protocollo IPv4

Nome	Dimensione Payload (*bytes*)	Dimensione overrhead
HDLC	0-8·∞	6
PPP	0-1500	4+(1$\vee$2)+(2$\vee$4)
Ethernet Standard	46-1500	27
Ethernet (802.3)	46-1500	16/24
WiFi (802.11)	2312	34
Token Ring (802.5)	4464	13/21
FDDI	*4464*	14/22
CS-PDU	<64*K*	7+(1$\vee$4)
SAR-PDU	44	36

TABELLA 1. Il valore maggiore rappresenta la **Maximum Transmission Unit** dei protocolli indicati



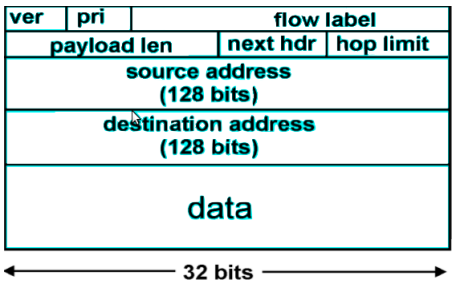
(A) *Particolarizzazione* ***Dual stack***. (B) *Particolarizzazione* ***Tunneling***.

FIGURA 2. Evidenziazione delle due componenti utilizzate nella soluzione *Dual stack e tunneling*

Inoltre, come vedremo di seguito, IP "lavora" coadiuvato da altri protocolli, come descritto dalla Tabella 2 a pagina 32

Ora dobbiamo parlare della gestione del nuovo protocollo *IPv6*, utilizzato per migliorare la gestione del protocollo della precedente versione, che è appunto la gestione della scalabilità della rete: tuttavia anche la gestione degli indirizzi tramite il protocollo **CIDR** (v. sottosezione 2.2.4 a pagina 30) porterà a breve l'esaurimento degli indirizzi IP disponibili, e pertanto si è scelto di aumentare il numero di bit del campo indirizzi da 32[2] a 128 bit, consentendo approssimativamente di identificare, sovrastimando un'efficenza di implementazione del 100%, 1500 indirizzi per 0.1m^2, e conseguentemente bisogna creare un nuovo protocollo IP. Conseguentemente bisognerà attuare delle politiche di conversione dei pacchetti durante la fase di transizione tra i due protocolli, in quanto non tutti i server supporteranno il nuovo protocollo, si metteranno quindi in atto le seguenti soluzioni:

[2]Il quale comporta ad una definizione di un indirizzo scrivibile con 4 caratteri di 256 valori possibili, da cui discende che $256^4 = 2^{32} = 4 \cdot 8^{10} \simeq 4 \cdot 10^9$



Campo	Dimensione (bit)	Descrizione
Version	4	come in IPv4, indica la versione utilizzata del pacchetto, attualmente viene utilizzata la 4 e la 6.
TrafficClass	8	v. `FlowLabel`: definisce la priorità del pacchetto in un flusso
FlowLabel	20	Identifica il flusso di pacchetti e, assieme a `FlowLabel`, servono per la qualità del servizio (**QoS**)
PayloadLen	16	Indica la lunghezza del pacchetto in byte senza l'intestazione
NextHeader	8	Sostituisce il campo di `Option` e di `Protocol`: ripsettivamente, se sono necessarie delle estensioni, allora queste vengono segnalate, altrimenti è una chiave di demultiplexing per identificare il protocollo eseguito al di sopra di IP: nella maggior parte dei casi però non vi saranno queste opzioni accessorie
HopLimit	8	Analogo al `TTL`, ha cambiato semplicemente nome per precisare il metodo che viene impiegato per contare il tempo di permanenza nella rete
Payload	65535	...

FIGURA 3. Protocollo IPv6

Protocolli duali: con questa implementazione si creano dei servers i quali supportano sia IPv4 sia IPv6 all'interno del livello network, in modo da poter rispondere ai datagram dei client nelle due versioni

Dual stack e tunneling: questa implementazione prevede l'utilizzo di due server che possano interpretare il protocollo IPv6, in quanto la rete che si vuole attraversare non supporta questo protocollo al livello network: da ciò segue che i due gateway alle due estremità dovranno supportare anche una differente identificazione di indirizzo a livello IPv4, ed inoltre dovrà essere implementato un sistema per poter identificare che il payload del messaggio è di tipo IPv6.

Traduttori: con questa implementazione si utilizza un processo di traduzione del datagramma IPv6 in IPv4, in quanto l'host ricevente è collegato ad una rete che supporta unicamente la versione inferiore: come si è accennato precedentemente, è necessario dunque convertire l'indirizzo con un **NAT** ("Network Address Translator") e di un **PT** ("Protocol Translator")[3]: mentre risulta abbastanza semplice tradurre l'indirizzo IPv4 in IPv6[4], rimane da risolvere la situazione opposta: all'interno di una stessa sessione, NAT alloca un nuovo indirizzo per la sorgente IPv6 in IPv4 all'interno di una tabella (questo indirizzo tradotto non è quindi globale, ma locale alla corrente area di destinazione). Ciò viene effettuato associando un *timeout*, scaduto il quale in assenza di datagram inviati, l'indirizzo IPv4 viene liberato. Tuttavia devono essere convertiti anche i messaggi **ICMPv6** in ICMPv4, ed inoltre non viene

[3]Introducendo conseguentemente un elevato *overrhead* di traduzione, e quindi una complessiva inefficienza

[4][...]

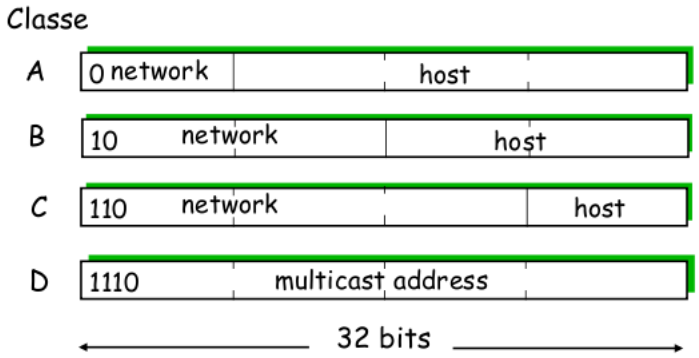


FIGURA 4. Attribuzione degli indirizzi *IPv4*

fatto alcun tentativo per cercare di tradurre i campi nella parte `Options` (ad esempio, verrà persa l'informazione di flusso, rendendo il pacchetto inutilizzabile nel caso in cui alla sorgente sia richiesta questa informazione). Nella traduzione sussiste un ulteriore problema quando il `Payload` IPv6 contiene informazioni sugli indirizzi IP: in tal caso si rivela necessario effettuare la traduzione di IPv6 a livello applicativo (**ALG**: "Application Level Gateway").

Bisogna inoltre considerare che IPv6 ha un campo di intestazione sempre lungo 40*byte*, mentre IPv4 senza opzioni è lungo

2.1. Gestione degli indirizzi. IPv4 in particolare ha una gestione *gerarchica* in quanto è presente una parte che corrisponde alla rete di riferimento (*network address*), ed un altro valore che corrisponde al particolare calcolatore di cui ne fa parte (*host address*), identificando così univocamente un calcolatore all'interno della rete: tuttavia è meglio precisare che un indirizzo non indica un particolare host, ma bensì una sua interfaccia di rete, in quanto se questo è collegato a due reti differenti, disporrà di due differenti indirizzi IP per ogni rete. Abbiamo quindi quattro possibili classi di indirizzi, identificando per ogni rete 2 indirizzi riservati:

(a) Gli indirizzi di questa classe iniziano con `0`, dove sono disponibili $2^7 - 2 = 126$ reti, con indirizzi che iniziano con `0` e `127` privati.
(b) Gli indirizzi di questa classe iniziano con `10` e sono disponibili 2^{14} reti
(c) Gli indirizzi di questa classe iniziano con `110` e sono disponibili 2^{21} reti
(d) Gli indirizzi di questa classe iniziano con `1110` e sono indirizzi di tipo **multicast**

L'**inoltro** è quel processo grazie al quale, mediante alle *tabelle di inoltro* create dal processo di **instradamento** ("routing") secondo particolari algoritmi distribuiti, si può inoltrare il pacchetto verso la porta corretta di uscita. Per effettuare questo discrimine possiamo definire un algoritmo come il seguente:

Host: Un host confronta se l'indirizzo della destinazione ha lo stesso numero di rete, allora viene richiesto (con una procedura che vedremo successivamente) l'indirizzo `MAC` della macchina, e quindi viene inviato il pacchetto tramite questo canale, altrimenti viene inviato direttamente al Router.

Router: Un router, se vede che il pacchetto ha la destinazione con lo stesso numero di rete di una sua interfaccia di rete, allora invia il pacchetto verso quella destinaizione come sopra, altrimenti se trovo l'indirizzo di rete all'interno della tabella di inoltro, invio il pacchetto tramite quella interfaccia, altrimenti inoltro al router di default.

Per quanto converne invece la gestione degli indirizzi in **IPv6**, si è già trattato in prima istanza nel punto 2 a pagina 22 la gestione della traduzione degli indirizzi: inoltre dopo la gestione CIDR (v. sezione 2.2.4 a pagina 30) non si sono utilizzate le classi, anche in questo caso all'interno di IP non verranno utilizzate le classi, ma è comunque utilizzata una suddivisione degli stessi:

(a) L'indirizzo `00...0` (128*bit*) non è specificato
(b) L'indirizzo `00...1` (128*bit*) è di **loopback** [5]
(c) Gli indirizzi con prefisso `111 111` identificano **indirizzi multicast**: essi ricoprono lo stesso ruolo degli indirizzi di classe D in IPv4, anche se in questo contesto si vuole dare agli indirizzi una attribuzione geografica, in modo che il loro indirizzo non dipenda dal ISP quale sono collegati.
(d) Gli indirizzi con prefisso `111 1110 10` identificano **link local unicats**: essi permettono di consentire ad un host di costruirsi autonomamnente un indirizzo che funzioni solamente nella rete a cui è connesso, senza preoccuparsi dell'unicità globale di tale indirizzo.
(e) Gli indirizzi con prefisso `111 1110 11` identificano **site local unicast**: essi, similmente ai precedenti, permettono di costruirsi indirizzi all'interno di una rete locale e quindi anche qui non è importante l'unicità globale.
(f) Tutti gli altri prefissi sono identificabili come **indirizzi unicast globali**: questi, per impedire che brevemente si possano esaurire, verranno:
  - ◇ assegnati come nell'assegnamento CIDR (v. sezione 2.2.4 a pagina 30), aggregando le informazioni di instradamento.
  - ◇ prevedendo che come in **EGP** (v. sezione 2.2.5 a pagina 31) la gerarchia non venga rigidamente rispettata (ad esempio la distinzione tra chi fornisce il servizio direttamente o indirettamente diventa incerta).

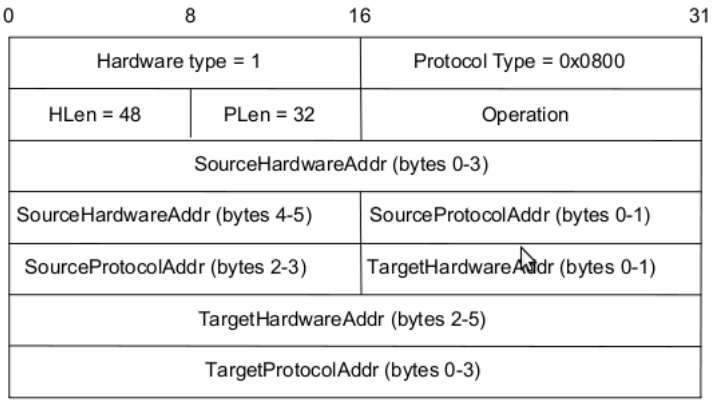
In particolare, all'interno degli indirizzi unicast globali, sono presenti anche gli indirizzi **anycast**, che non sono altro che un gruppo di host o di routers che hanno lo stesso indirizzo: questo potrebbe essere utile ad esempio quando questi forniscono globalmente uno stesso servizio, e non è necessario che sia individuabile ogni singolo calcolatore al suo interno: tuttavia ognuno di questi deve essere a conoscenza (es. da parte del gestore di rete) di essere membro di questo gruppo.

Si può inoltre notare dalla Figura 3 a pagina 22 che non è tra l'altro presente il campo **Checksum**: in quanto a livelli inferiori appesantirebbero la comunicazione, lasciando il controllo della correttezza delle informazioni trasportate ai livelli superiori.

Ora definiremo i protocolli che vengono utilizzati per risolvere le lacune che si sono presentate nel discorso precedente:

2.1.1. *ARP.* Con il protocollo *ARP* ("address resolution protocol") ci preoccupiamo di effettuare *la conversione tra indirizzo IP e indirizzo MAC della rete della controparte dell'host o del router di cui sopra*, in modo che ogni calcolatore possa ricavarsi la propria **tabella di inoltro**, nella quale le informazioni vengono periodicamente eliminate ogni 15 minuti. Ogni host (compreso il router) quindi, controlla se l'informazione è contenuta all'interno della nostra tabella, altrimenti viene inviato in broadcast il messaggio con il protocollo definito dalla immagine 5 nella pagina successiva, dal quale gli host "listeners" possono trarre informazioni

[5] `127.0.0.1` in IPv4: è l'indirizzo di **localhost**, «*può essere usato dalle applicazioni per comunicare con lo stesso sistema su cui sono in esecuzione. Essere in grado di comunicare con la propria macchina locale come se fosse una macchina remota è utile a scopo di test, nonché per contattare servizi che si trovano sulla propria macchina, ma che il client si aspetta siano remoti.*»(Wikipedia.it)



Campo	Spiegazione
Hardware type	Identifica il tipo di rete fisica
Protocol type	Specifica il protocollo del livello superiore
HLen	Identifica la lunghezza dell'indirizzo fisico
PLen	Identifica la lunghezza dell'indirizzo del livello superiore
Operation	Identifica se si tratta un messaggio di richiesta o uno di risposta
SourceHardwareAddr	Specifica l'indirizzo hardware del mittente (**Target**: ricevente)
SourceProtocolAddr	Specifica l'indirizzo alto livello del mittente (**Target**)

FIGURA 5. Protocollo ARP

utili sul mittente (se si ha già nella tabella un'informazione riguardante a tale calcolatore viene aggiornata tale record della tabella, altrimenti non viene in alcun modo aggiunto questo nuovo campo in quanto inutile alla loro comunicazione); il ricevente del messaggio, che è anche il destinatario del messaggio ARP, aggiunge comunque l'informazione del mittente (o eventualmente la aggiorna) sul mittente, e gli invia l'informazione riguardo al suo numero di MAC.

2.1.2. *DHCP.* Con questo protocollo ci preoccupiamo di *associare ad ogni calcolatore presente all'interno della rete un numero di indirizzo IP*: questo è necessario in quanto, mentre gli indirizzi MAC sono predefiniti dal costrutture, gli indirizzi IP devono essere riconfigurabili, in quanto un calcolatore potrebbe essere "delocalizzato" dalla rete originaria e spostato in un'altra. Per evitare una possibile gestione errata degli indirizzi da parte di un amministratore umano, si preferisce adottare questo protocollo per il quale, dato un certo "serbatoio" di indirizzi, viene effettuata o una **configurazione statica**, dove in una tabella del server DHCP vengono memorizzate le associazioni tra indirizzo hardware e IP, oppure viene effettuata una **configurazione dinamica**, secondo la quale dopo la richiesta di un calcolatore dell'indirizzo una volta che si è connesso alla rete, quando esso non lo adopererà più dovrà essere rilasciato, poichè eventualmente gli indirizzi si potrebbero esaurire. In questa seconda prospettiva, abbiamo due soluzioni implementative:

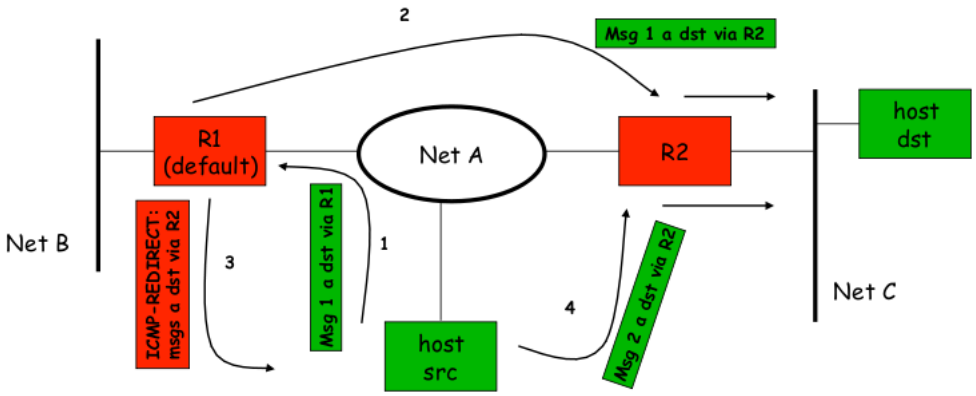
- L'host contatta il server DHCP con il messaggio `DHCPDISCOVER` inviato all'indirizzo broadcast del livello IP `255.255.255.255`[6], che risponde fornendogli l'indirizzo.
- Nel caso in cui esista un server centralizzato per tutte le nostre sottoreti, in modo da non utilizzare un server per ognuna di queste, si utilizza all'interno della rete un **relay agent** che, conoscendo già inizialmente l'indirizzo di tale server, quando riceverà nella sua sottorete un pacchetto come sopra, lo invierà al server.

[6]NOTA: il router tuttavia blocca il fluire di questo messaggio sulla rete, e quindi permane unicamente all'interno della rete locale

Tale servizio può essere affidato anche ad un servizio di *leasing* con il quale un calcolatore chiede di rinnovare periodicamente il suo indirizzo, altrimenti il server (in caso di quasto del primo) si preoccuperà di riallocare l'indirizzo.

> Tuttavia, per quanto riguarda il protocollo **IPv6**, si vuole creare una "operatività" del tipo *plug-and-play*, con la quale non sia necessario una configurazione ai client DHCP, e quindi senza la necessità di un tale server: si potrebbe quindi chiedere direttamente l'identificativo unico sulla propria linea di collegamento, fornendo ad esempio il proprio MAC address. In questo modo si possono limitare le configurazioni iniziali che erano necessarie ad ogni calcolarore, che fino ad ora deve invece possdere un IP valido, una maschera di sottorete e l'indirizzo di un server per la risoluzione dei nomi.

2.1.3. *ICMP.* Con questo protocollo possiamo identificare degli errori che possono essere segnalati all'host sorgente quando la controparte (es.) non è in grado di elaborare con successo un datagramma IP, oppure dei messaggi che un router può inviare al sorgente, come indicare l'esistenza di un percorso migliore verso la destinazione, come rappresentato dalla figura sottostante:



> Nella versione *IPv6*, ICMPv6 fornisce ulteriori messaggi (come "Packet too big") o funzioni per la funzione del multicasting

2.2. Instradamento interdominio. In questa sezione invece ci preoccupiamo di definire quei algoritmi che permettono la gestione della tabella di inoltro, necessaria (es.) al router per decidere verso quale porta effettuare l'inoltro del pacchetto, ovvero effettua la *corrispondenza tra numeri di rete e destinazioni successive, tramite il minimo cammino percorribile*. Per il momento tratteremo di protocolli o di algoritmi all'interno del nostro dominio *AS* ("sistema autonomo"). Questi protocolli-algoritmi hanno le seguenti caratteristiche in comune:

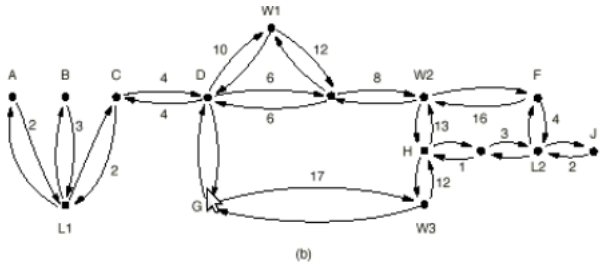


FIGURA 6. Rappresentazione di un sistema autonomo (AS) come grafo. Si sottolinea il fatto che uno stesso collegamento può avere costi differenti per ogni verso di percorrenza

- ogni nodo del nostro grafo può rappresentare un(v. sezione 2.2.4 a pagina 30) calcolarore (host, router...) oppure una rete (vedi Figura 6);
- si suppone che il costo di collegamento rimanga invariato nel tempo, non vengono considerate inclusioni di nuovi nodi o i loro eventuali guasti

In particolare, in base al contesto nel quale si vuole effettuare il *routing* si possono identificare due categorie di tali protocolli:

(1) **Interior Gateway Protocol**s, i quali vengono utilizzati per gestire il *routing* interdominio: a questa famiglia appartengono *RIP* e *OSPF*.
(2) **Exterior Gateway Protocol**s, i quali vengono utilizzati globalmente per gestire il *routing* "stadard" in Internet intradominio: a questa famiglia appartengono *EGP* e *BGP-4*.

Un'altra intestazione di estensione di *IPv6* è quella di instradamento, dove vengono designate quali aree topologiche devono essere attraversate: tuttavia se questa è assente, l'instradamento avviene come in CIDR (v. sezione 2.2.4 a pagina 30) di *IPv4*

Tuttavia è necessario sottolineare che, negli algoritmi che vedremo, si da per scontato di conoscere effettivamente il peso (o **metrica**) della linea che si sta considerando: tuttavia, senza informazioni aggiuntive, non possiamo conoscere la capacità della linea in quanto non conosciamo nè la loro latenza (*latency*) nè la loro ampiezza di banda (*bandwidth*). Nel tempo quindi si sono trovate le seguenti metriche del carico della rete:

New routing mechanism: bisogna premettere che, precedentemente, il meccanismo consisteva nel misurare il numero di pacchetti che erano presenti all'interno della coda di attesa, seppure questa era una misura artificiosa del **carico**. Con questo meccanismo invece si considerava sia l'ampiezza di banda, sia la latenza utilizzando il ritardo (*delay*) calcolando come segue:

$$\text{Delay} = (\text{DepartT} - \text{ArrivalT}) + \text{TransmissionT} + \text{Latency}$$

dove $\text{DepartT} - \text{ArrivalT}$ indica il ritardo del pacchetto subito durante l'attesa nella coda, con gli altri parametri costanti per quellaù linea. Tuttavia in questo modo, se una linea risulta molto congestionata, si troverà lo spostamenti dei pacchetti da un router all'altro, causando il loro sovraccaricamento e poi il loro inutilizzo ciclicamente.

Revised Arpanet routing metric: per diminuire la velocità con la quale si riducevano i costi, introducendo in prima istanza la percentuale di utilizza della linea, valore che veniva mediato con la percentuale precedente in modo da diminuire i cambiamenti bruschi di valore, riducendo così la probabilità per la quale i nodi potessero abbandonare un determinato percorso

2.2.1. *Definizione dell'internetwork globale.* Gli algoritmi progettati unicamente per funzioinare all'interno di singoli AS (IGP), non permettono una scalabilità ottimale all'interno di una rete globale. Tramite il meccanismo di **subnetting**, si vogliono distinguere all'interno di un stesso indirizzo di rete più sottoreti tramite una *maschera di sottorete*, che consentono la divisione dell'indirizzo IP nel campo dell'`Host number` come `Subnet ID` e `Host ID`. In quanto possiamo associare una stessa maschera a più sottoreti differenti, avremo che *tutti gli host appartenenti alla stessa rete fisica con la stessa maschera di rete avranno lo stesso indirizzo di rete, mentre host che si trovano su differenti reti possono anche avere uno stesso indirizzo di rete*. Un indirizzo di sottorete per i calcolatori appartenenti alla stessa sottorete si può calcolare prendendo l'indirizzo di un calcolatore ivi presente, ed effettuando l'`AND` bit a bit con la maschera

di sottorete. Con questa tecnica si può mantenere invariato l'algoritmo alla base di **ARP**, potendo così minimizzare il numero delle informazioni da mantenere all'interno della tabella di routing memorizzando non ogni singolo indirizzo IP dei calcolatori, ma unicamente il loro numero di sottorete e la maschera utilizzata:

◇ Se dato l'**indirizzo di sottorete** dell'host corrente, confrontato con quello della destinazione, ottenibile tramite l'`AND` bit a bit del suo indirizzo IP con tale maschera, essi sono uguali, allora il pacchetto può essere inoltrato direttamente all'interno della stessa sottorete, altrimenti deve essere inviato al router principale.

Per quanto riguarda i **router**s avremo ora per ogni riga della tabella una tripla costituita dal numero di sottorete, dalla sua maschera di sottorete e l'interfaccia di rete o il router verso il quale inviare il pacchetto ricevuto:

◇ Se dato l'indirizzo di destinazione, effettuando in serie per ogni riga della tabella confrontando il numero di sottorete contenuto con quello ottenuto tra l'`AND` bit a bit del numero di destinazione con la maschera di sottorete contenuta nel record si nota che sono identici, allora si spedisce il datagram verso l'interfaccia o il router di Next Hop, altrimenti lo consegno al router di default.

2.2.2. *RIP ~ (IGP).* Con questo meccanismo si vuole creare la tabella di instradamento, o meglio in questo caso il *vettore delle distanze* secondo il seguente algoritmo:

- Ogni nodo imposta a `0` la distanza con sè stesso e `1` i nodi adiacenti, setta ad ∞ gli altri non adiacenti
- Ogni nodo invia ai propri adiacenti le informazioni ottenute.
- Il nodo che riceve la comunicazione dal suo vicino somma 1 ai valori sul collegamento con i nodi e tra questi e quelli da lui memorizzati prende il valore minore.
- Prima o poi l'algoritmo termina ed ognuno sarà a conoscenza della distanza minima verso la destinazione prescelta

Tuttavia, mentre è facile pensare che, se si guasta un nodo collegato ad altri nodi, sia immediato ricostruire la topologia della rete, ciò non accate se il nodo dove è avvenuto il guasto è collegato solamente ad un altro nodo: in questo caso si assisterà ad un aumento progressivo delle distanze, fino ad un valore sufficientemente alto (che deve essere stabilito) che indica il valore infinito (viene usato il valore `16`). Un altro metodo per ovviare a questi cicli è la tecnica detta di **split horizon**, con la quale si evita di inviare ai nodi limitrofi informazioni che si sono apprese da quei stessi nodi: tuttavia questa tecnica non è successivamente drastica da far in modo di risolvere tutti i possibili conflitti.

2.2.3. *OSFP ~ (IGP).* Una caratteristica di *OSPF* (*Open Shortest Path First*) è il supporto per motivi di sicurezza della richiesta di autentificazione dei messaggi di informazione sulla rete, qualora si ritenga poco affidabile l'informazione pervenuta da un Host, richiedendo una chiave. Altre caratteristiche che non sono desumibili direttamante dagli algoritmi descritti sotto sui quali si appoggia, sono l'identificazione di una gerarchia, suddividendo il dominio in aree (invece di conoscere l'intero percorso, è utile sapere unicamente come raggiungere l'area giusta; di ciò parleremo meglio in seguito) e il **bilanciamento del carico**, ovvero l'assegnazione a tutti quei percorsi diretti verso la medesima destinazione di un medesimo costo, in modo da distribuirvi pariteticamente il traffico. Questo protocollo si basa quindi sui seguenti algoritmi:

Inondazione affidabile: L'affidabilità di questo algoritmo è dovuta al fatto che:

- il campo `ID` identifica chi ha creato (e quindi inviato) il pacchetto

- un *numero di sequenza* per poter identificare i pacchetti che contengono l'informazione più recente, contraddistinta da un calore alto di questo campo
- un *TTL*, ovvero il tempo di vita del pacchetto, in modo da garantire il raggiungimento di un pacchetto che contiene un'informazione recentemente trasmessa; altro motivo è l'eliminazione dei pacchetti che circolano nella rete inutilmente. Inoltre ogni terminale ha il compito di far "invecchiare" questo campo, facendo scadere i propri pacchetti.

Ogni noda accetta il pacchetto se è il più recente di quelli ricevuti, ed invia l'informazione a tutti i suoi vicini, tranne a quello dal quale l'aveva ricevuto precedentemente. Inoltre nuovi pacchetti vengono generati solamente allo scadere di un "cronometro", settato con temporizzazione elevata in modo da evitare eccessivi **overhead** della rete, o quando viene modificata la topologia della stessa: in particolare il guasto della linea viene identificato quando nello **strato di collegamento** o quando non viene più percepito un *hello packet* da parte dei vicini.

Algoritmo di Dijkstra: Con l'algoritmo di *Shortest path first* si vuole creare il minimo albero di copertura secondo il seguente algoritmo di tipo **Link State**[7]:

- ◇ Viene inizializzato l'algoritmo con la lista `tentative` vuota e la `confirmed`, detto l'elemento $\mathcal{A}$ il nodo corrente, con <$\mathcal{A}$, `0`, `-` > (all'inizio coincidente con il nodo di partenza $\mathcal{S}$).
- ◇ Dopo aver considerato il costo degli archi con tutti i vicini $\mathcal{N}$:
  - ○ se $\mathcal{N} \notin$`tentative`, lo metto in `tentative`
  - ○ se $\mathcal{N} \in$`tentative`, aggiorno il valore contenuto se quello trovato presenta un costo complessivo $\overrightarrow{\mathcal{AS}} + \overrightarrow{\mathcal{SN}}$ minore.
- ◇ Se `tentative` è vuota, l'algoritmo termina, altrimenti metto in `confirmed` il dato avente costo inferiore.

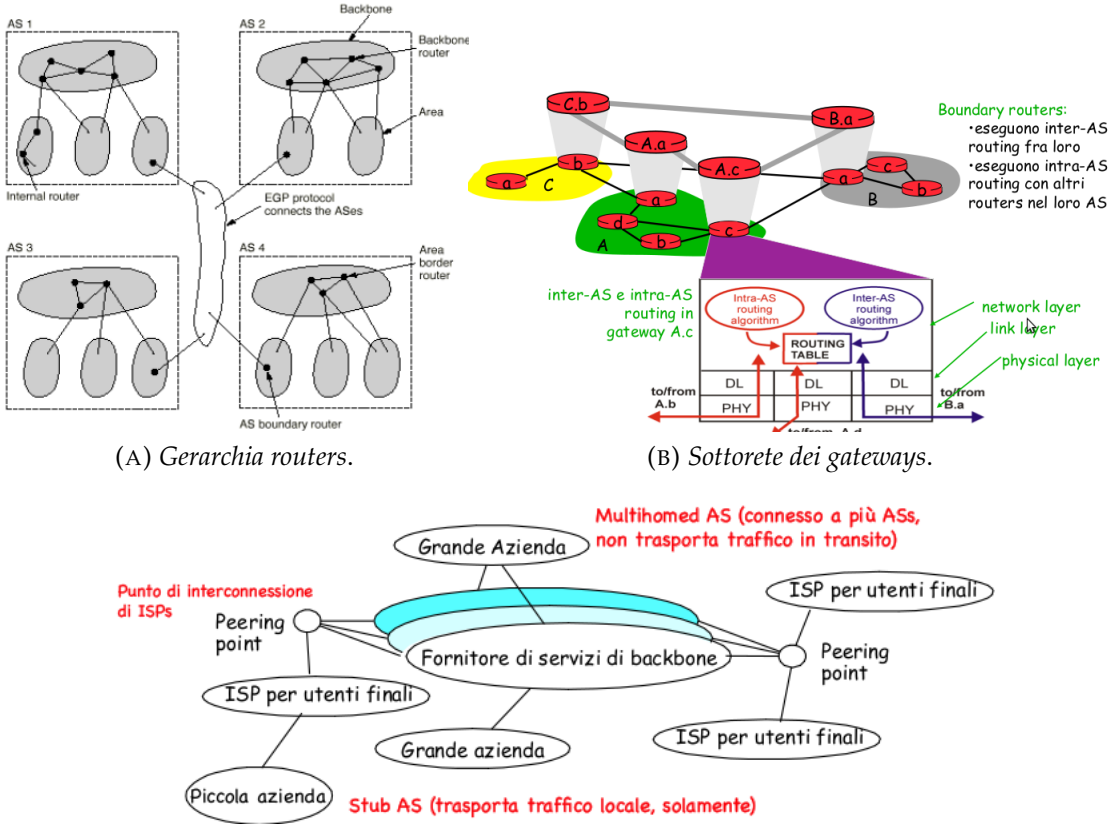
Tuttavia possiamo anche conoscere l'applicazione dei seguenti algoritmi anche all'interno di una rete intradominio quale Internet, con il quale introduciamo l'analisi delle reti globali. Facciamo riferimento ora alla suddivisione del sistema autonomo in aree piuttosto di identificazione di sottoreti alla quale si aveva accennato precedentemente:

(1) Ogni AS viene identificato con un numero univocamente
(2) Ogni area può identificare anche una rete od un insieme di reti contigue, dove i routers mantengono unicamente le informazioni della loro area, ed il flooding avviene solo in quelle aree. Da ciò segue che la comunicazione verso un'altra rete avviene inviando prima il messaggio all'area di backbone, poi all'area di destinazione, dove si può giungere alla destinazione.
(3) Ogni As ha un'area di **backbone** per mettere in collegamento le sottoreti: tale backbone è quindi unicamente un'area di collegamento.

Ora possiamo identificare la gerarchia dei routers all'interno dell'internetwork:

Internal routers: eseguono esclusivamente l'instradamento intradominio (ovvero nell'area)
Backbone routers: eseguono esclusivamente un protocollo all'iterno del backbone del domino
Area border routers: instradano l'informazione da e verso il backbone

[7] «*Un protocollo di routing Link State (routing basato sullo stato del collegamento), è un tipo di algoritmo in cui la topologia dell'intera rete e tutti i costi dei collegamenti sono noti.In un protocollo link state ogni nodo della rete acquisisce informazioni sullo stato dei collegamenti adiacenti ed inoltra queste informazioni a tutti gli altri nodi della rete tramite un Pacchetto Link State trasmesso tramite un algoritmo di link state broadcast.*». Wikipedia.it



(A) *Gerarchia routers.* (B) *Sottorete dei gateways.*

(C) *Esempio di internetwork globale.*

FIGURA 7. Esempi di internetwork globali

Boundary routers: oltre ad abilitare la connessione verso l'interno dell'area, devono anche essere in grado di comunicare verso l'esterno tramite protocolli comuni.

- ◇ Instradano l'informazione da e verso il loro AS
- ◇ Eseguono protocollo intra AS nel corrente AS
- ◇ Eseguono protocollo intra AS con gli altri Boundary routers

Questi particolari routers vengono anche chiamati **gateway**s in quanto sono il mezzo tramite il quale i calcolatori esterni possono accedere alle altre reti.

Come si può vedere dalla Figura 7, ai gateways è richiesto di poter leggere i protocolli delle reti attigue, e quindi devono essere in grado di implementare per ogni rete più protocolli in modo da poter effettuare la traduzione.

2.2.4. *CIDR.* Con questo metodo, si vuole *effettuare l'instradamento interdominio senza considerare le classi dell'indirizzo IP* (CIDR stà per *Classless InterDomain Routing*). Questa tecnica in particolare permette di risolvere il problema di scalabilità della rete permettendo di associare più indirizzi IP con un unico prefisso, ed in particolare questo scopo è perseguibile in un modo più semplice se prendiamo indirizzi IP con numeri di rete attigui. Possiamo in questo modo migliorare l'utilizzo degli indirizzi di rete, assegnando più indirizzi IP di una classe inferiore rispetto ad un indirizzo IP di classe superiore, «limitando conseguentemente l'esaurimento dello spazio di indirizzamnento IP». Tuttavia, per ovviare alla memorizzazione di più righe di `Network Number` per un solo AS, si possono **aggregare i percorsi**, utilizzando un'unica riga della tabella, dove viene il primo indirizzo IP (`X` con tre caratteri) ed il numero di bit in comune (`b`) presenti con tutti gli indirizzi successivi, scrivendo `X/b`. In questo modo otteniamo:

- ◇ assegnazione degli indirizzi in blocch più piccoli
- ◇ uni unico prefisso di rete da usare nelle tabelle di inoltro

Possiamo inoltre riscontrare tra somiglianza di CIDR con il procedimento di *subnetting*, in quanto anche in questo metodo si ha un metodo di aggregazione, che in questo caso è la condivisione di un indirizzo tra più reti fisiche, e chiamarla **supernetting**. Inoltre, se un ISP ha un'insieme di clienti ai quali ha assegnato a sua volta dei sottoindirizzi, in quanto questi sono accessibili tramite detto ISP, allora per il fornitore è sufficiente fornire alla rete globale per ogni cliente un'unico indirizzo aggregato che includa sè stesso ed i clienti. In questo modo, in quanto possiamo associare arbitrariamente gli indizizzi in base alla lunghezza dei bit che si sono aggregati, ed in quanto possiamo avere nelle tabelle di inoltro indirizzi aggregati di diversa lunghezza, scegliamo d'inoltrare il datagram tra quelli che possono corrispondere all'indirizzo della nostra destinazione, quello con il prefisso più lungo.

2.2.5. *BGP ed EGP ~ (EGP).* Effettivamente, all'interno di un nostro dominio amministrativo (AS) vorremmo determinare la nostra *routing policy*, ad esempio non vogliamo che le nostre informazioni possano passare all'interno di un nostro AS rivale, oppure scegliere il nostro percorso di istradamento dei pacchetti; inoltre vogliamo che tutto questo sia possibile senza l'aiuto di altri AS e prevenire loro errori o comportamenti avversi, senza la creazione di cicli.

- Il primo protocollo a tentare di risolvere questo problema era **EGP**, ma questo costringeva la rete globale ad assumere una strutturea ad albero, e quindi consentiva relazioni del tipo "padre-figlio" e non tra elementi paritetici.
- Il protocollo attualmente in uso è **BGP**, che è utilizzato come protocollo in uno strato superiore all'affidabile protocollo di trasporto TCP[8], e con questo vogliamo gestire AS arbitrariamente interconnessi e, diversamente da come era strutturata molti anni fa, sono presenti più *backbone*, dove gli AS possono essere di vari tipi (v. Figura 7 a fronte):

  AS stub: trasportano unicamente traffico locale
  AS multihomed: rifiuta di trasportare traffico di transito
  AS di transito: trasporta sia traffico locale sia di transito

  Altre particolarità di questo protocollo sono le seguenti:
  - ◇ per la configurazione del protocollo, ciascun AS sceglie un nodo come *BGP speaker*, il quale ha il compito di stabilire connessioni con gli altri BGP speaker delle altre reti per conoscere la loro raggiungibilità;
  - ◇ esistenza dei "gateway"s;
  - ◇ vengono notificati inoltre dei percorsi completi di indirizzi IP tramite i quali è possibile accedere a quella particolare rete;
  - ◇ devono essere in grado di cancellare dei percorsi precedentemente annunciati tramite una informazione negativa chiamata *withdrawal route* (ritiro del percorso)

  Inoltre, con questo protocollo viene garantito che gli AS utilizzati dal protocollo BGP devono essere univoci, mentre gli AS di tipo stub, ovvero la maggioranza, non necessitano di questo privilegio.

  Da queste informazioni possiamo ottenere dei percorsi senza cicli, ma è lasciato al singolo sistema autonomo la scelta della strategia per ottenere il percorso migliore.

[8]Per questo non è necessario inviare più volte gli stessi messaggi, ma è utile inviare in caso di conferma un messaggio di *keep-alive*: conseguentemente se questo messaggio non perviene più, vuol dire che chi aveva trasmesso non è più raggiungibile e quindi le informazioni precedentemente ottenute non sono più valide.

Protocollo	Riassunto
ARP	Determinazione dell'indirizzo fisico MAC di un host dato il suo indirizzo IP.
DHCP	Associa ad ogni calcolatore il suo indirizzo IP dinamico.
ICMP	Invia al mittente il messaggio di errore e la comunicazione del percorso migliore per il raggiungimento del router.
RIP	Calcolo di routing intradominio (creazione della tabella di istradamento) tramite il calcolo del vettore delle distanze.
OSPF	Calcolo di routing intradominio tramite l'algoritmo dei cammini minimi di Dijkstra.
CIDR	Calcolo di routing intradominio senza classi con aggregazione degli indirizzi.
BGP	Variante di EGP vetusta con rappresentazione della rete come albero (e quindi tramite una gerarchia) e non con un grafo.
EGP	Instradamento intradominio con rappresentazione della rete mediante grafo e determinazione della *routing policy* dei diversi domini amministrativi.

TABELLA 2. Elenco dei protocolli accessori IP

Protocollo	Dimensione header
ARP	28
IPv6	40
UDP	8
TCP	20+4

TABELLA 3. Elenco delle dimensioni dell'header di alcuni protocolli. Per i protocolli di trasporto, quando avviene l'incapsulazione in IPv**x**, viene aggiunto un campo Checksum di 4*byte*.

CAPITOLO 5

Livello di Trasporto

Lo scopo del livello di trasporto è quello di creare dei protocolli **end-to-end** in modo da realizzare l'astrazione della comunicazione diretta tra due calcolatori. Possono essere presenti all'interno di tali protocolli:

- garanzia di consegna del messaggio
- consegna dei messaggi nell'ordine dell'invio
- consegna di una sola copia dei messaggi
- messaggi di dimensione arbitraria
- sincronizzazione tra mittente e destinatario
- il ricevente può applicare una politica di controllo di flussi nei confronti del mittent
- supporto per più processi applicativi su ciascun host

1. UDP

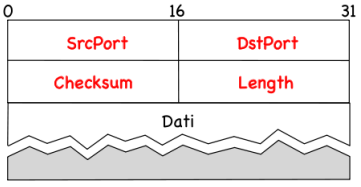
In questo protocollo ("User Datagram Protocol") opportuno aggiungere un livello di demultiplexing, in modo da consentire a più applicazioni di poter usufruire di questo protocollo: in particolare I processi vengono identificati l'un l'altro tramite una *porta*. tramite la quale ricevono le informazioni (tale tecnica è anche detta *sottoindirizzamento*: le porte possibili sono fino a $64K$ (2^{16}). Queste in particolare i riferiscono alle porte di un singolo host con la quale avviene la comunicazione; gli host possono venire a conoscenza della porta tramite la quale scambiare i messaggi in questo modo:

- ◇ Il server intrattiene la comunicazione tramite una **well-known port**, ovvero da una porta precedentemente pubblicizzata.
- ◇ La well-known port viene utilizzata come **port mapper**, tramite la quale fornisce in risposta la porta tramite la quale accedere al dato servizio.

Tuttavia tale protocollo non effettua le seguenti garanzie:

- non viene implementato il controllo del flusso
- non viene garantita la consegna nè l'ordine con la quale viene effettuata, che va quindi eventualmente implementato a livello applicativo.

Da ciò segue che tale protocollo può essere utilizzato per quei dispositivi applicativi dove è tollerata la perdita di informazione, ed inoltre controlla la correttezza del messaggio tramite la **somma di controllo**. I campi header del messaggio, così come si presentano nell'ordine di cui sotto, sono di dimensione $16bit$ ciascuno e sono:



UDP header

In questo caso particolare la primitiva di ricezione è bloccante. Il campo Checksum viene inoltre calcolato sull'intera intestazione, sul contenuto del messaggio e sullo `pseudoheader`, ovvero da tre campi dell'intestazione IP, quali:

- numero di protocollo
- indirizzo IP mittente e destinatario

dai quali si desume quindi che è sottointeso che il sottoprotocollo sia l'indirizzo IP.

2. TCP

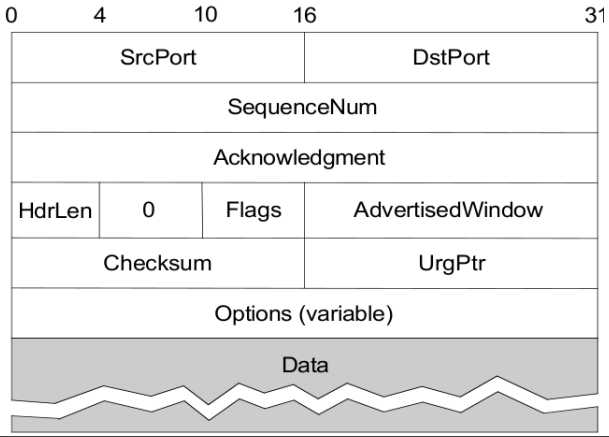
Con questo protocollo ("Transfer Control Protocol"), allo stesso livello di TCP, si tenta di risolvere i problemi non risolti nel protocollo precedente, quali:

- garanzia nella consegna (è affidabile) ed in sequenza;
- può essere utilizzato un **controllo di flusso** (v. sezione 2.1 a pagina 51), per consentire al ricevente di limitare i dati inviati dalla sorgente e quindi impedire che i mittenti esauriscano la capacità dei ricevitori;
- può effettuare un **controllo sulla congestione** (v. sezione 2.2 a pagina 53), ovvero può limitare l'invio di troppi dati all'interno della rete.

Inoltre questo protocollo è **full-duplex**, ovvero consente il doppio fluire delle informazioni in ciascuna direzione; inoltre, analogamente a quanto accade per il protocollo UDP, è consentito anche in questo caso il *demultiplexing*, che consente a più processi di dialogare con le altre entità. Tuttavia, in quanto questo protocollo utilizza l'algoritmo a *sliding window*, dobbiamo riscontrare alcune differenze tra astrazione end-to-end e comunicazione punto-punto:

- ◇ Per simulare la connessione punto punto è necessario instaurare prima una connessione stabilendo un accorto, e poi terminare la comunicazione.
- ◇ In quanto non si conosce l'effettiva struttura della rete che separa i due calcolatori, (`posizione`) ed in quanto bisogna considerare che il traffico che vi circola è variabile (`tempo`), non si può mai conoscere con certezza l'effettivo RTT della rete: da ciò segue che *l'algoritmo di sliding window, che il meccanismo di timeout per far scattare la ritrasmissione deve essere* ***adattativo***.
- ◇ Sempre per il fattore `tempo` e `posizione`, i pacchetti potrebbero subire il ritardo nella fase di consegna (e quindi conseguentemente i pacchetti potrebbero arrivare non nell'ordine voluto): si puuò inoltre garantire l'invio del pacchetto per un tempo di $120s$, detto *maximum segment lifetime* per impedire che un pacchetto possa essere scartato dopo un tempo esiguo.
- ◇ Il **controllo di flusso** deve essere impiegato perché i calcolatori in essa presenti possono supportare differenti capacità di memorizzazione.
- ◇ Inoltre si reputa opportuno conoscere l'**ampiezza di banda** complessiva, in quanto il mittente non è a conoscenza di evenutali altre connessioni, e quindi è opportuno effettuare il **controllo sulla congestione**.

TCP in particolare simula il flusso di byte memorizzando all'interno del campo dati finché il pacchetto non è stato riempito sufficientemente, ed inoltre il flusso di questi pacchetti è detto **segmento** in quanto ognuno di questi trasporta un segmento del flusso di byte di dimensione `MSS` ("maximum segment size"): altro modo con il quale può essere forzato l'invio dell'informazione è l'esplicita richiesta da parte del processo mittente tramite l'operazione `PSH`, svuotando completamente il buffer mittente, oppure viene effettuato l'invio allo scadere di un temporizzatore. Facciamo ora riferimento alla Figura 2 a pagina 36, dove guardiamo alla fase di instaurazione e terminazione della connessione:



Campo	Dimensione (*bit*)	Descrizione
SrcPort+DstPort	16+16	Identificano la porta mittente e la porta ricevente sulle quali si stabilisce e termina la connessione: in questo modo si acquisiscono e si rilasciano le porte. Quando questo avviene più di una volta sulle stesse due porte si parla di "incarnazione della stessa connessione".
SequenceNum	32	Contiene il numero di sequenza del primo byte memorizzato.
Acknowledgement	32	Contiene il numero dell'ultimo byte confermato.
HdrLen	4	Contiene la lunghezza dell'intestazione.
0	6	Campo riservato ad usi futuri.
Flags	6	Contiene dei valori la cui semantica sono dei comandi riguardanti la connessione TCP: **SYN:** quando si instaura la connessione **FIN:** quando terminal la connessione **ACK:** è `1` quando si conferma la validità del campo `Acknowledgement` **URG:** è `1` quando si conferma la presenza di dati urgenti (v. `UrgPtr`) **PSH:** indica l'invio di dati alla ricezione di questo segmento **RST:** è `1` quando si vuole terminare la connessione per un overflow o per la ricezione di un segmento inaspettato.
AdvertizedWindow	16	Indica la dimensione della finestra, ovvero il numero di byte di dati che il ricevitore è preparato ad accettare.
Checksum	16	Viene calcolato esattamente come UDP.
UrgPtr	16	Indica l'indirizzo dopo il quale sono presenti dati non urgenti.
Options	32	Questo campo viene utilizzato in fase di connessione per concordare il "Message Size" (MSS) che, se non specificato, è di $536B$, allora ogni macchina deve essere in grado di utilizzare questa dimensione di dati in memoria.

FIGURA 1. Protocollo TCP

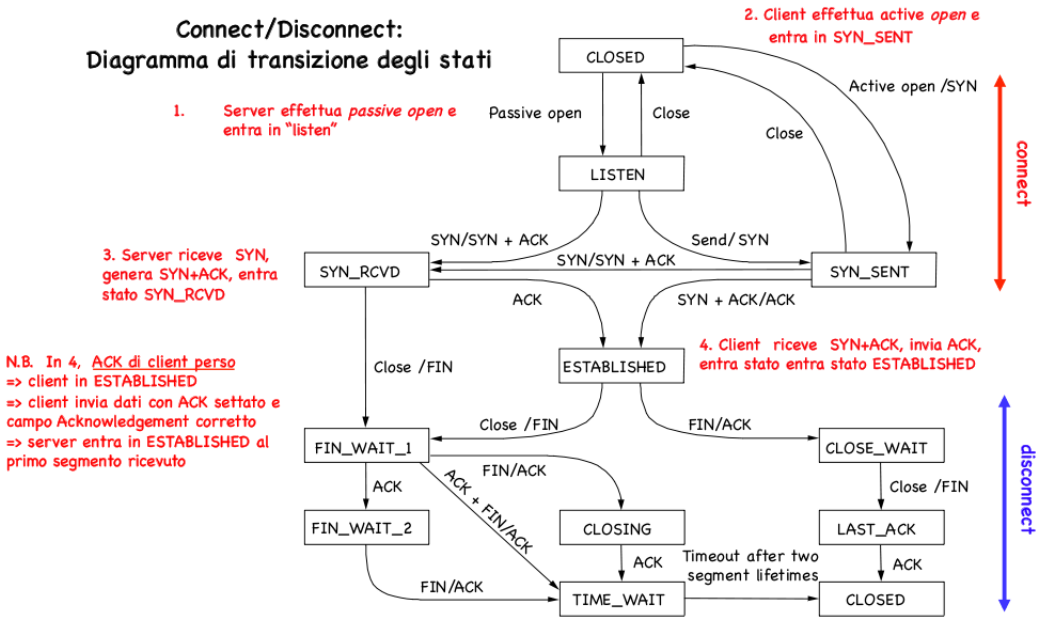


FIGURA 2. Diagramma delle transizioni di stato.

- ◇ Per stabilire la connessione:
  - *(i)* Il client ("attivo") invia al server ("passivo") invia il numero di sequenza che pensa di utilizzare (`<Flags=Syn, SequenceNum=x>`) dove `x` è un numero casuale[1].
  - *(ii)* In seguito il server conferma incrementando il campo `Acknowledgement` [2] con (`<Flags=Syn+Ack, SequenceNum=y, Acknowledgement=x+1>`).
  - *(iii)* Infine il client conferma con `<Flags=Ack, Acknowledgement=y+1>`
- ◇ Per rilasciare invece la connessione si fa in modo che i due processi agli estremi si disconnettano indipendentemente ("simplex"). Per rilasciare la connessione quindi ed evitare che non vengano recepiti altri datagrammi non ancora prevenuti, viene utilizzato il seguente handshake:
  - *(i)* $\mathcal{A}$ invia alla controparte ($\mathcal{B}$) il messaggio di terminazione `Fin` ed aspetta un messaggio analogo dal mittente entro un tempo $2\cdot$ TTL[3], dove TTL=120s.
  - *(ii)* $\mathcal{B}$ riceve `Fin` da $\mathcal{A}$ e gli invia `Fin` aspettando un tempo $2\cdot$ TTL.
  - *(iii)* $\mathcal{A}$ riceve il pacchetto di `Fin` e risponde con un `Ack`, e contemporaneamente termina la sua sessione.
  - *(iv)* $\mathcal{B}$ riceve `Ack` e termina la connessione

La caratterizzazione algoritmica di questo protocollo verrà trattata, assieme ad altri algoritmi, all'interno della successiva sezione.

2.1. Allocazione delle risorse. Parleremo in questa sezione della tecnica di allocazione delle risorse per gestire parzialmente il problema di controllo della congestione in quanto,

[1] Viene scelto un numero casuale per evitare il più possibile che, in una precedente incarnazione della connessione, si possa aver inviato un pacchetto di ACK, evitando quindi di rispondere a quel numero di sequenza.

[2] In quanto in questo modo indichiamo il valore che ci aspettiamo di ricevere, confermando implicitamente di aver ricevuto i valori precedenti

[3] É altresì necessario aspettare questo tempo, in quanto una successiva incarnazione (se si chiudesse immediatamente la connessione) porterebbe alla sua chiusura. Dopo questo tempo la comunicazione viene comunque terminata da quel lato

se si riesce ad ottimizzare il modo con il quale vengono gestiti i pacchetti in arrivo nel router, allora avremo facilitato il compito della congestione delle risorse da parte del mittente, tecnica descritta nella sezione . Più nel dettaglio, con allocazione delle risorse si intende «*il processo mediante il quale gli apparati di una rete cercano di soddisfare le richieste delle applicazioni in merito alle risorse della rete [...]in competizione tra loro [ovvero] l'ampiezza di banda delle linee e lo spazio di memorizzazione nei buffer di router e switch*». Più semplicemente si impedisce il veloce sovraccarico di un ricevitore lento, o comunque impedito da altri fattori.

Tuttavia si possono presentare una serie di situazioni che possono portare ad un rallentamento nella gestione della linea:

- Con un flusso singolo a banda fissa, considerando due host soggetti della comunicazione $\mathbb{A}$ e $\mathbb{B}$ con un router $\mathbb{C}$, si potrebbe avere una linea $\overrightarrow{\mathbb{AC}}$ ed una $\overrightarrow{\mathbb{CB}}$ a differenti velocità di trasmissione: l'unica soluzione sarebbe quella di gestire tale situazione bufferizzando i messaggi da inviare.
- Con un flusso singolo ma con banda variabile, avremo (es.) un $\overrightarrow{\mathbb{CB}}$ il cui valore dipenderà dal tempo, e quindi il DTR (**data transfer rate**) dovrà variare più o meno istantaneamente secondo le variazioni di Bandwidth.
- Con un flusso multiplo di dati, si deve *(a)* adattare il tempo di trasferimento dei dati per sfruttare al meglio la banda disponibile e *(b)* allocare la banda tra i diversi flussi in modo equo: tuttavia i flussi in entrata potrebbero avere differenti velocità, così come quello del flusso in uscita, costituendo in questo modo il problema del collo di bottiglia.

[...]

Per effettuare il controllo dell'efficacia della gestione delle code ci preoccupiamo che questa sia **efficace** e che sia **fair**: per la prima valutazione possiamo considerare la potenza ("power") della connessione, volendo ottenere il massimo **throughput** ed il minimo **delay**, anche se non necessariamente aumentare il primo fattore porta immediatamente alla riduzione del secondo, dove

$$\text{Power} = \text{Throughput}/\text{Delay}$$

Tuttavia bisogna considerare che questo rapporto dipende a sua volta dal **carico** della linea, ovvero dalla allocazione delle risorse; inoltre vogliamo che la rete sia stabile, ovvero che essa continui efficentemente a lavorare anche quando si trova "sotto pressione"

Si deve tuttavia sottolineare come la prenotazione delle risorse dipenda dalle risorse disponibili e dalla loro gestione, ed inoltre «*vorremmo che ogni flusso ricevesse [...]un'equa frazione di banda [in base ad una] eguale frazione di banda [di risorse allocate] ma, anche in assenza di prenotazioni, frazioni equali possono [in alcuni casi] non equivalere a frazioni eque*». Dobbiamo tuttavia premettere che il sottostrato della rete di cui stiamo discorrendo è di tipo best-effort, e quindi quando la coda del nostro "agente di collegamento" sarà piena, verranno eliminati alcuni pacchetti che sono stati bufferizzati in una **coda**[4], ed in particolare può essere utilizzato una strategia di eliminazione di tipo **tail-drop**: in questo modo i pacchetti perduti dovranno essere gestiti autonomamente dal protocollo TCP come si vedrà nella sezione 2.2 a pagina 53. Un altro metodo possibile è la **gestione prioritaria** delle code, utilizzando il campo `ToS` del protocollo IP: il router quindi implementerà più code FIFO una per ciascuna priorità, il quale si occuperà di trasmettere prima i pacchetti presenti nella coda a priorità maggiore; tuttavia in questo modo si può introdurre una *starvation* per le code a priorità inferiore. Per ovviare a questo inconveniente, si utilizza quindi una gestione *round-robin* delle code, implementata in modo tale da prevedere la ricezione di pacchetti di qualsiasi tipo,

[4]Questo tipo di struttura dati, essendo FIFO, permette un primo passo verso la gestione *fair* delle risorse

non segnalando qundi alcunché alle sorgenti, che continuano indisturbatamente ad inviare i pacchetti secondo il loro algoritmo prestabilito.

Anche introducendo questi accorgimenti, non si è arrivati ad una gestione completamente *fair* delle nostre code: si vuole inoltre ottenere un servizio che emuli la trasmissione del pacchetto bit a bit, simulando questo procedimento prevedendo «*quando terminerebbe la trasmissione di un pacchetto se venisse inviato usando un servizio a rotazione bit per bit*», considerando il tempo di arrivo dei pacchetti; secondo questo principio:

- ◇ tra tutti i pacchetti sopraggiunti, si invia sempre quello arrivato per primo (e quindi se è già stata iniziata la trasmissione di un pacchetto mentre ne sopraggiunge uno nuovo, bisognerà aspettare la terminazione di suddetta consegna);
- ◇ inoltre tra due pacchetti di due flussi diversi in attesa di essere trasmessi, viene scelto quello che ha lunghezza minore (ritardando di fatto l'invio del pacchetto di maggiori dimensioni);
- ◇ inoltre, se è presente una linea rimasta inattiva, la bandwidth della linea di uscita verrà gestita tra le code rimanenti non vuote.

Con questa soluzione si può comunque implementare la priorità delle singole code attribuendo ad esse dei pesi, che variano (es.) in base al valore di `ToS` nel campo IP, rendendo quindi una coda più veloce delle altre con una politica di **weighted fair-queueing**

Parte 2

Algoritmi distribuiti

CAPITOLO 6

Controllo degli errori

1. CRC

Detto $C(x)$ il polinomio di controllo per il messaggio $M(x)$, detto κ il grado del polinomio $C(x)$, ovvero la posizione del più alto bit non zero, si dovrà inviare il messaggio:

$$P(x) = (M(x) \ll k) \texttt{ xor } ((M(x) \ll k) \texttt{ xor } C(x))$$

Inoltre se la divisione di $P'(x)$ per $C(X)$ ha come resto zero, il messaggio ricevuto, con buona probabilità, non è stato alterato.

2. Checksum

Se effettuo il checksum a n bit, allora sommo, considerando per prime le cifre più significative, tutti i valori di cui calcolare la somma di controllo, cancellando i riporti nelle $n + 1$ cifre. A somma ultimata invertirò il valore dei bit.

3. Burst

Un tale errore inizia e finisce con un bit errato, ed è separato da un numero di bit corretti pari a quelli del burst.

CAPITOLO 7

Commutazione di circuito virtuale

Con questa tecnica il percorso viene inidilizzato prima dell'invio dei dati con una tecnica di **routing**: successivamente i pacchetti possono fluire dopo aver stabilito detta connessione: in questo caso si hanno degli overrhead iniziali, però verrà garantita all'interno della "intra-rete" locale la consegna dei datagrammi. Per stabilire detto circuito virtuale, è necessario pertanto instaurare la connesione mediante una **connection setup**, nella quale all'interno dello switch verrà memorizzato tramite una tabella, dove per ogni record è memorizzato un circuito dove verrà indicato:

(1) l'*interfaccia d'ingresso* attraverso la quale arrivano i pacchetti dalla sorgente;
(2) l'*identificatore del circuito virtuale* (VCI) d'ingresso che identifica all'interno dello switch in questione univocamente tale connessione;
(3) l'*interfaccia d'uscita* attraverso la quale escono i pacchetti verso la destinazione;
(4) l'*identificatore del circuito virtuale* d'uscita, ovvero l'identificatore che identifica nel router successivo tale connessione.

Per instaurare questo routing potrebbero essere utilizzati due approcci differenti:

PVC: con il *permanent virtual circuit* è l'amministratore di rete a determinare il circuito virtuale permamente: questa prospettiva è poco interessante dal punto di vista algoritmico;
SVC: con il *switched virtual circuit* è lo stesso invio di messaggi attraverso la rete che stabilisce e rilascia il circuito virtuale: tratteremo questa soluzione.

La composizione del setup avviene nelle seguenti fasi:

- ◇ All'attivazione del circuito, per ogni router per la quale passa il primo pacchetto, lo switch identifica la porta dalla quale è giunto il pacchetto ed associa un *VCI* d'ingresso, sapendo che la destinazione è raggiungibile tramite una porta già conosciuta [1]
- ◇ Quando il messaggio giunge finalmente a destinazione e, se vuole accettare di instaurare la connessione con la sorgente, anche questo assegna un VCI allo switch precedente, dove tale valore identificherà tutti i pacchetti provenienti da detta sorgente.
- ◇ Ora per comunicare a tutti gli host a ritroso i valori prescelti per il loro VCI che corrisponde al valore di VCI-successivo per lo switch precedente: questo viene effettuato tramite l'invio di una conferma.
- ◇ La connessione viene disattivata da parte del mittente inviando un messaggio di `teardown`, causando l'eliminazione del record della tabella dei circuiti virtuali in tutto il percorso precedentemente instaurato.

In questo algoritmo non è però stato descritto come faccia uno switch a conoscere dove sia situata la destinazione.

[1] scegliendo per quella porta un numero differente di VCIi rispetto a quelli utilizzati sia per VCIi e VCIo per la stessa porta

Nella specifica dell'**instradamento dalla sorgente** ("source routing") vogliamo inviare l'informazione completa del percorso da seguire per poter giungere a destinazione, memorizzando l'interfaccia di rete tramite la quale gli switch devono inviare i pacchetti, facendo poi ruotare l'elenco in modo che alla fine del percorso si possa utilizzare l'informazione di partenza al contrario per inviare un `Ack`. Tuttavia con questo approccio:

- si suppone che l'host mittente abbia sufficiente conoscenza della topologia della rete;
- non è possibile determinare la lunghezza dell'intestazione, in quanto non si è in grado di predire il numero massimo di switch;

Sono possibili quindi diversi approcci risolutivi, ma di limitato interesse algoritmico.[...]

CAPITOLO 8

Algoritmi per Lan estese (Bridge)

1. Learning Bridge

Con questo algoritmo si vogliono creare delle **tabelle di inoltro** ("forwarding"), ovvero indicando per ogni host di destinazione per il quale si riceve un pacchetto, verso quale porta inviare l'informazone. Come sempre, potremmo affidare ad un amministratore il compito di scrivere queste tabelle, però potremmo automatizzare la procedura nel modo seguente:

- ◇ Inizialmente le tabelle sono vuote
- ◇ Se arriva un pacchetto tramite la porta χ inviato dall'host $\mathcal{A}$, allora associo $<\chi, \mathcal{A}>$
- ◇ Se la destinazione non è nota, il pacchetto viene inviato in broadcasting su tutte le altre porte
- ◇ Dopo una certa temporizzazione, le informazioni vengono cancellate

2. Algoritmo ad albero di copertura

In questa sezione analizzeremo questo algoritmo di Radia Perlman, i cui obiettivi principali sono i seguenti:

(1) rendere *trasparente* ai nodi l'internetwork di LAN, ovvero far equiparare la LAN estesa ad una "normale" rete LAN;
(2) *utilizzare i bridges ridondanti* per far fronte ad eventuali guasti, posso disattivare momentaneamente le porte che porterebbero invece nei collegamenti cicli all'interno della rete, in modo da recuperarle successivamente;
(3) deve essere *autoconfigurante*, consentendo ai bridge di avere come unica informazione iniziale il proprio MAC (non è tuttavia un algoritmo di accesso al mezzo);
(4) l'algoritmo deve essere *scalabile*, ovvero non deve richiedere macciore memoria se si verifica l'espansione della rete;
(5) idealmente la bandwidth deve mantenersi costante;
(6) l'algoritmo deve *stabilizzarsi rapidamente*, ovvero deve portare ad un risultato unicamente pari a O(RTT), producendo quindi un albero DETERMINISTICO;
(7) deve effettuare il *caching* della locazione dei nodi nella LAN estesa, ovvero selezionando le porte dove verranno trasmessi i frames e mantenendo il database degli indirizzi conosciuti;
(8) scopo finale dell'algoritmo è quello di *eliminare loop permanenti* nella rete e di minimizzare eventuali loop temporanei di quei bridges che non implementano l'algoritmo;
(9) é basato su di un meccanismo connectionless, simile al "best effort" dei DATAGRAMS;
(10) é *configurabile* dinamicamente settando o modificando alcuni parametri.

Dopo aver visto gli obiettivi, analizziamo di seguito i principi di realizzazione dell'algoritmo:

(11) sono gli stessi bridge a scegliere le porte verso le quali inoltrare i frames;

(12) è necessario identificare la radice dell'albero secondo un prefissato criterio, (es.) si sceglie il bridge con l'identificatore minore;
(13) la *radice* invierà i frames sempre verso tutte le porte;
(14) ciascun bridge ha il compito di calcolare il percorso più breve verso la radice, ed in quale porta questo percorso è presente
(15) tutti i bridge connessi ad una stessa LAN nominano un unico *designato*
(16) i messaggi di configurazione hanno il formato:

$$< \texttt{root, hop ,mitt} >$$

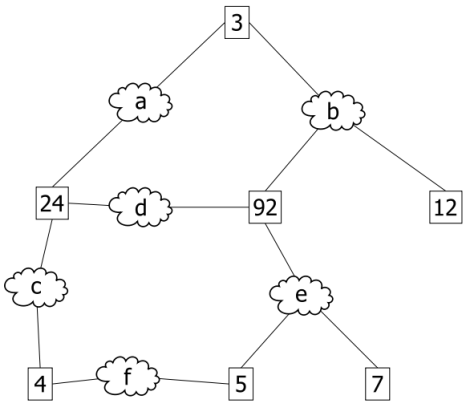
(17) ciascun bridge memorizza il messaggio di configurazione migliore da quelli che aveva ricevuto dalle sue porte

L'esecuzione dell'algoritmo avviene nelle seguenti fasi:

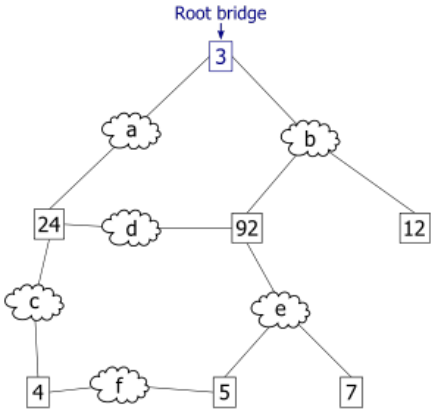
- Viene scelta come radice il bridge con ID minore
- Ogni bridge sceglie la **root port** scegliendo quella interfaccia per la quale si ha un percorso minimo verso la radice, o in caso di parità viene scelta la porta collegata ad un bridge di ID minore.
- Ogni rete locale determina la **designed port** del bridge tramite il quale si ha un costo mimore verso la radice, o in caso di parità viene scelta la porta collegata ad un bridge di ID minore.
- Se sono presenti molteplici collegamenti tra due differenti bridge, si sceglie quello collegato a quello di priorità minore
- Nessun'altro è una porta attiva.

Per evidenziare ora l'invio dei messaggi tra i bridge, si affina l'algoritmo nel modo seguente:

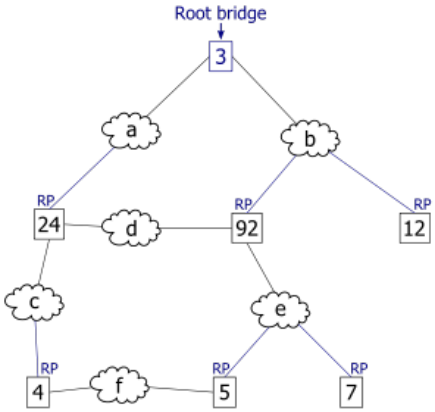
◇ Inizialmente ogni nodo si crede radice
◇ Se trovo un messaggio da un bridge di ID minore, allora il nodo attuale non sarà più radice
◇ Per ogni messaggio pervenuto da quella porta, salvo l'informazione migliore:
  ○ ID radice minore
  ○ stesso ID radice ma distanza minore
  ○ stesso ID radice e distanza, ma ID mittente minore
◇ Se il bridge riceve per quella porta un messaggio da uno con ID minore, non invierà più messaggi per quella porta (diventa **designed port**) e continua ad inoltrare pacchetti dai mittenti, incrementando di 1 l'`hop`.
◇ Se il bridge riceve per quella porta (di quella rete) riceve un messaggio che un altro ha una distanza minore dalla radice, allora disabilita quella porta, dalla quale non inoltrerà più alcun messaggio.



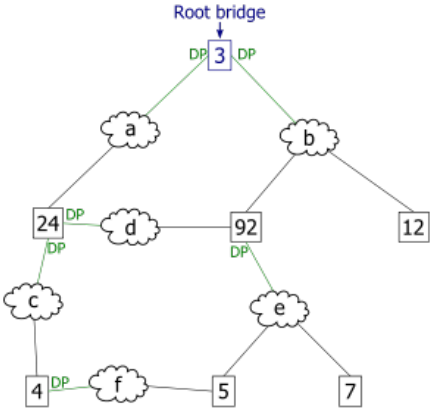
(A) *An example network. The numbered boxes represent bridges (the number represents the bridge ID). The lettered clouds represent network segments.*



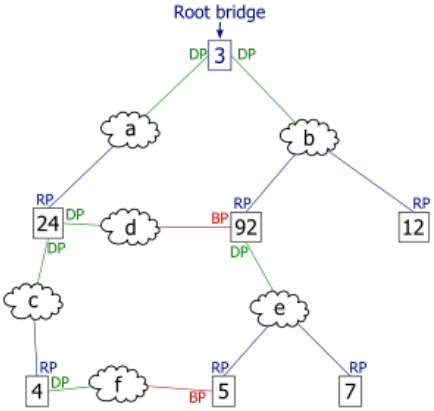
(B) *The smallest bridge ID is 3. Therefore, bridge 3 is the root bridge.*



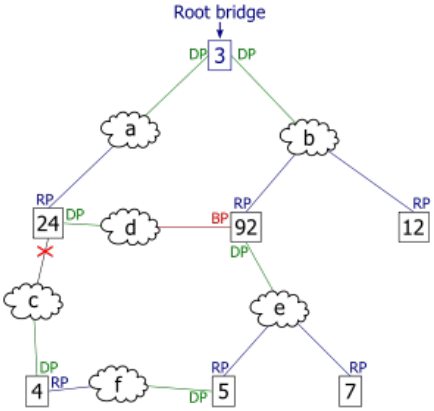
(C) *Assuming that the cost of traversing any network segment is 1, the least cost path from bridge 4 to the root bridge goes through network segment c. Therefore, the root port for bridge 4 is the one on network segment c.*



(D) *The least cost path to the root from network segment e goes through bridge 92. Therefore the designated port for network segment e is the port that connects bridge 92 to network segment e..*



(E) *This diagram illustrates all port states as computed by the spanning tree algorithm. Any active port that is not a root port or a designated port is a blocked port.*



(F) *After link failure the spanning tree algorithm computes and spans new least-cost tree.*

FIGURA 1. Esecuzione dell'algoritmo da Wikipedia.en

CAPITOLO 9

Algoritmi di Sliding Windows

1. Reti a connessione diretta

In questa sezione introdurremo i concetti di questi algoritmi, che nelle reti a connessioni diretta (analizzando il livello datalink), analizzando come questo algoritmo è stato ideato a questo livello per risolvere il problema della **trasmissione affidabile**: non a caso questo algoritmo si implementa anche nel protocollo di trasporto TCP, che tratteremo di seguito.

Principalmente questa tecnica viene utilizzata in quanto spesso i messaggi sono dotati di un overrhead talmente elevato da rendere impossibile l'effettivo riconoscimento di tutti gli errori presenti all'interno del messaggio, portando conseguentemente all'eliminazione dei pacchetti corrotti pervenuti. Per ricevere gestire questo algoritmo si utilizzano come contenuto informativo aggiunto:

- **ACK** (acknowledgement): é un pacchetto di piccole dimensioni utilizzato per segnalare l'avvenuta ricezione
- **timeout**: questo viene impiegato quando il mittente non riceve nessun ACK da parte del ricevente, in modo il quale (es.) viene ritrasmesso il messaggio.

1.1. Stop-And-Wait ~ (ARQ). Ciò introduce il più semplice algoritmo possibile basato sulla *richiesta di ripetizione automatica* (ARQ: "automatic repeat request")

◇ All'invio di un messaggio, il mittente aspetta un *timeout*: se entro questo tempo non arriva alcuna conferma, allora esso viene ritrasmesso

Tuttavia potrebbe succedere che tale conferma arrivi sempre dopo un tempo maggiore rispetto a quello prefissato per il timeout: da ciò segue che si può supporre, dopo un certo numero di tentativi a vuoto, che il server non sia più disponibile, oppure utilizzando un *exponential backoff*, ovvero duplicando ogni volta il tempo di attesa. Tuttavia, arrivando la conferma in ritardo, il mittente potrebbe anche pensare che sia sopraggiunta la conferma per il pacchetto inviato successivamente: questo problema può quindi essere parzialmente risolto assegnando sia al *frame* inviato sia all'`ACK` un bit per identificare la sequenza in trasmissione.

1.2. Sliding window. Tuttavia con l'implementazione di Stop-And-Wait non si riesce appieno ad utilizzare tutta la capacità del canale, ad esempio:

> Dato un canale con latenza $L = 50\ ms$ e con un $RTT = 100\ ms$, da ciò segue che possono essere consegnati alpiù 10 frames al secondo. Assumendo una bandwidth di $B = 2Mbps$ con la dimensione dei frame $1Kb$, si avrà una velocità di invio pari a $80Kbps$, utilizzando però solo il 4% della banda:
>
> $$\frac{80\ Kbps}{2\ Mbps} \sim \frac{8 \cdot 10^4}{2 \cdot 10^6} = 4 \cdot 10^{-2}$$
>
> Calcoliamo ora la capacità della rete adoperata: in quando abbiamo $C = 2\,Mbps \cdot 0.05\,s = 100\,kb$ che, assumendo l'ack di dimensione trascurabile ($1bit$), si potrebbe utilizzare fino a $2(D \times B)$ arrivando a $200kb$, occupando sostanzialmente anche qui

il 4% della pipe:

$$\frac{8}{200} = \frac{1}{25}$$

da cui possiamo evincere che invece di inviare un solo pacchetto, ne potremmo inviare 25 al secondo.

Data l'osservazione, possiamo confermare con un solo `ACK` più messaggi contemporaneamente. Definiamo quindi tre variabili che devono essere gestite dal mittente:

SWS: ("send window size") indica il limite superiore dei frame che il mittente può inviare prima di ricevere una conferma: questi sono memorizzati in un buffer in quanto devono essere sempre inviabili in caso di scadenza del *timeout*;

LAR: ("last acknowledgement received") indica il numero dell' ultima conferma ricevuta: viene incrementato quando viene ricevuto un `ACK`;

LFS: ("last frame sent") indica il numero dell'ultimo frame inviato;

SeqNum: è il numero di sequenza che viene assegnato dal mittente ad ogni frame;

che deve rispettare la seguente invariante:

$$LFS - LAR \leq SWS$$

Per quanto riguarda inoltre il ricevente, devono essere gestite le seguenti variabili:

RWS: ("receive window size") indica il massimo numero di frame fuori sequenza che possono essere accettati dal ricevitore;

LAF: ("largest acceptable size") indica il numero del frame accettabile più elevato;

LFR: ("largest frame received") indica il numero di sequenza dell'ultimo frame ricevuto;

SeqNumToAck: è una *conferma cumulativa* della effettuata ricezione con accettazione di tutti i pacchetti con valore ad esso minore od uguale;

ed anche in questo caso deve essere rispettata l'invariante:

$$LAF - LFR \leq RWS$$

ed agisce in base al seguente algoritmo:

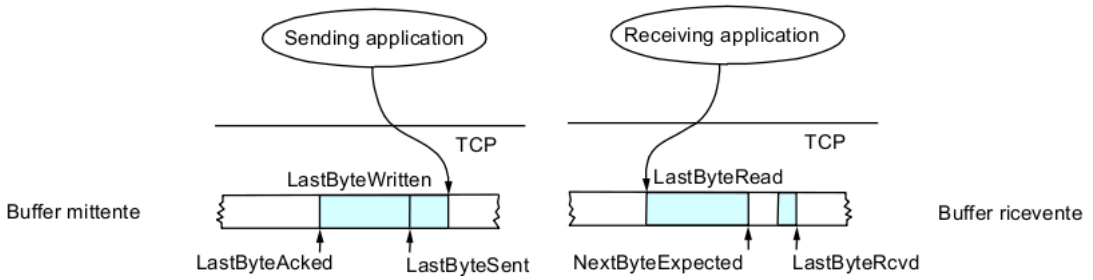
- ◇ se il frame è fuori dall'intervallo (**LFR**, **LAF**] allora viene rifiutato, altrimenti viene accettato, inviando come `ACK` **SeqNumToAck** se sono pervenuti tutti i pacchetti ad esso precedenti.

Tuttavia, anche in questo caso, potrebbe capitare che la conduttura non venga mantenuta piena, ad esempio quando si ha una errata consegna dei pacchetti. I possibili metodi risolutivi possono essere i seguenti:

- Inviare una converma negativa (NAK), che tuttavia può essere sostituito dal timeout, oppure la ricezione di conferme duplicate.
- Stimare SWS una volta calcolato il $D \times B$, oppure utilizzare RWS$=1$ memorizzando solamente il frame utile e non quelli fuori richiesta, oppure RWS=SWS.

Dobbiamo anche considerare che, all'interno dei pacchetti di informazione lavoriamo con numeri finiti, e quindi dobbiamo essere in grado di distinguere quando, limitando il range dei numeri di sequenza tra [0, **MaxSeqNum**], ed imponendo che:

$$\begin{cases} \textbf{MaxSeqNum} \geq \textbf{SWS} + 1 & \text{RWS} = 1 \\ \textbf{SWS} < (\textbf{MaxSeqNum} + 1)/2 & \text{SWS} = \text{RWS} \end{cases}$$



(A) *Buffer di trasmissione.* (B) *Buffer di ricezione.*

FIGURA 1. Puntatori dell'algoritmo Sliding Window in TCP

`AdvertizedWindow` in quanto nel secondo caso bisogna prevedere eventuali incarnazioni, che comportano il ritorno inatteso al valore iniziale di dati trasmessi (**wraparound**) traslando eventualmente le due metà della ricezione come avviene nello stop-and-wait.

2. TCP

Il protocollo TCP introduce numerosi miglioramenti per quanto riguarda la soluzione adottata nei problemi precedenti, e che sono già stati enunciati nell'introduzione di questo protocollo (v. sezione 2 a pagina 34).

- il **controllo di flusso** viene gestito tramite una rivisitazione dell'algoritmo di *sliding window*;
- il **controllo di congestione** viene gestito tramite modificazioni del protocollo "stop-and-wait".

2.1. Sliding Window TCP. Un primo cambiamento introdotto in questo algoritmo dal protocollo TCP è che *il ricevente comunica al mittente la dimensione della finestra* tramite il campo `AdvertisedWindow`, scelto in base alla memoria assegnatagli, e quindi impedendo il sovraccarico della macchina calcolatrice. Inoltre, è forse banale sottolineare come in questo caso considereremo buffer, definendo quindi le costanti **MaxSendBuffer** e **MaxRcvBuffer** le lunghezze rispettivamente del buffer mittente e ricevente, e numeri di sequenza di dimensioni limitate, in quanto stiamo discorrendo di una implementazione non teorica di questo algoritmo.

Per quanto riguarda il mittente, vengono gestiti i puntatori:

- `LastByteAcked`
- `LastByteSent`
- `LastByteWritten`, ovvero scritto all'interno del buffer dal mittente

che devono rispettare le seguenti invarianti:

$$\text{LastByteAcked} \leq \text{LastByteSent} \qquad \text{LastByteSent} \leq \text{LastByteWritten}$$

e quindi per garantire l'invio di solo quei byte permessi bisogna imporre:

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

Inoltre in quanto il numero dei dati inviati ma non confermati è $LastByteSent - LastByteAcked$, ed essendo $AdvertisedWindow$ quanti ne può accettare il ricevente, il mittente può effettivamente inviare i dati solamente se $EffectiveWindow$ ha valore maggiore di zero:

$$EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)$$

Inoltre per evitare che il buffer trabocchi bisogna fare in modo che:

$$\text{LastByteWritten - LastByteAcked} \leq \text{MaxSendBuffer}$$

ed inoltre, se la dimensione dei dati da inviare della applicazione supera il contenuto del buffer, allora TCP dovrà bloccare il processo trasmittente in modo da non permettergli di generare altri dati ingestibili, ovvero:

$$\text{(LastByteWritten - LastByteAcked)} + y > \text{MaxSendBuffer} \quad y > 0$$

Per quanto riguarda il ricevente, vengono gestiti analoghi puntatori:

- `LastByteRead`, incrementato quando viene metabolizzata l'informazione dal ricevente, aumentando la dimensione della finestra.
- `NextByteExpected`, che punta al byte successivo che si aspetta in quanto ricevuto fuori-fase
- `LastByteRcvd`

che devono rispettare le seguenti invarianti:

$$\text{LastByteRead} < \text{NextByteExpected} \qquad \text{NextByteExpected} \leq \text{LastByteRcvd+1}$$

e quindi, per evitare che il buffer trabocchi bisogna fare in modo che:

$$\text{LastByteRcv - LastByteRead} \leq \text{MaxRcvdBuffer}$$

Inoltre in quanto l'ultimo byte inviato è $NextByteExpected - 1$ e che quindi il numero dei byte già inviati sono $(NextByteExpected - 1) - LastByteRead$, segue che il numero dei byte che sono accettabili dal ricevente (indicati in `AdvertisedWindow`) sono:

$$AdvertisedWindow = MaxRcvBuffer - ((NextByteExpected - 1) - LastByteRead)$$

Da questa ultima relazione si sottolinea il fatto che se l'applicazione si attarda a metabolizzare dati (e quindi **NextByteExpected** è l'unico che viene incrementato in quanto sopraggiungono dati mentre **LastByteRead** rimane invariato)), **AdvertisedWindow** tenderà presto a zero.

Da ciò segue che un processo ricevente lento può provocare il blocco di un processo mittente: un modo per poter sempre controllare che il ricevente ora possa adempiere al suo compito è quello di inviare sempre a destinazione un byte di dati, in quanto come valore restituito si avranno i valori più aggiornati di `Acknowledge` e `AdvertizedWindow`, finché non si oterrà un `AdvertizedWindow`$\neq 0$.

Per evitare inoltre il "wraparound", bisogna fare in modo che il numero di sequenza non ritorni al suo valore originario in un tempo consigliato di $120s$, e per mantenere "pieno" il canale di comunicazione, dovremo fare in modo di avere un campo dati di `AdvertisedWindow` sufficientemente grande (il che però non è possibile per le connessioni più veloci) da avere un numero ingente di `SequenceNum`.

Come tuttavia si era già evidenziato precedentemente per l'algoritmo di Sliding Window per le reti punto-punto, se si utilizza un *timeout* molto superiore al RTT della rete, si avrà un'inefficienza in qunato non si trasmetteranno pacchetti quando ce ne sarà la possibilità; se invece in caso contrario il *timeout* è molto minore del RTT, l'`ACK` arriverà in ritardo, e quindi verrà trasmesso un'altra volta il pacchetto che era già stato ricevuto dalla sorgente. Per risolvere questo problema sono stati introdotti tre algoritmi, uno migliorativo del precedente:

A. originario: questo algoritmo non tiene conto della variazione del RTT, in quanto *SampleRTT* è altresì il tempo impiegato dall'invio del messaggio all'`ACK` di avvenuta ricezione: il valore viene quindi mediato tramite un valore α con il valore correntemente stimato di RTT (*EstimatedRTT*) effettuando quindi un "tuning", ovvero una media pesata:

$$EstimatedRTT = \alpha \cdot EstimatedRTT + (1 - \alpha) \cdot SampleRTT$$

dove si raccomanda di utilizzare un $\alpha \in [0.8, 0.9]$ per rallentare le modifiche al valore di $EstimatedRTT$. Tuttavia questo algoritmo non tiene conto a quale conferma si fa riferimento, in quanto si potrebbe considerare attribuire l'ACK all'ultimo pacchetto inviato, mentre questo voleva riferirsi al precedente.

A. Karn/Partridge: per risolvere il problema definito precedentemente, si restringe la misura di $SampleRTT$ per quei pacchetti che sono stati inviati una volta sola, ed invece di utilizzare l'$EstimatedRTT$ si applica l'algoritmo di backoff esponenziale. Tuttavia con questa soluzione non si ah una sufficente considerazione dei valori di RTT precedenti, anche se questo algoritmo, non essendo strettamente collegato a TCP, potrebbe essere impiegato per qualsiasi protocollo di trasporto.

A. Jacobson/Karels: Dobbiamo premettere che il tempo di *timeout* deve essere maggiore di $EstimatedRTT$, e quindi possiamo scrivere

$$timeout = EstimatedRTT + Difference$$

imponendo:

$$\begin{cases} Difference = SampleRTT - EstimatedRTT \\ EstimatedRTT_{n+1} = EstimatedRTT_n + (\delta \times Difference) \\ Deviation_{n+1} = Deviation_n + \delta(|Difference| - Deviation_n) \end{cases}$$

possiamo ottenere:

$$TimeOut = \mu \cdot EstimatedRTT + \phi \cdot Deviation$$

dove $delta \in [0, 1]$, e test empirici hanno ottenuto come valori consigliabili $\mu = 4$ e $\phi = 4$.

2.2. Temporizzazioni TCP. Con questi metodi di temporizzazione TCP si ha l'obiettivo di risolvere il problema di congestionamento della rete. In generale TCP affida ad ogni sorgente la responsabilità di determinare la capacità disponibile nella rete, in modo da essere certi di quanti pacchetti trasmettere. In quanto ACK viene utilizzato come conferma del successo di consegna, allora vuol dire che può accodare un altro pacchetto senza timore di congestionare la rete, allora ACK ha la funzione di temporizzare l'invio. Descriveremo quindi alcune tecniche adottate in questo protocollo. Più precisamente con controllo della congestione si intende trovare un metodo da parte del mittente per prevenire le condizioni di sovraccarico della rete, e quindi che un insieme di sorgenti invii troppi dati contemporaneamente all'interno della rete.

2.2.1. *AMID: aumento additivo, diminuzione moltiplicativa.* Utilizziamo in questo caso una "finestra di congestione" che chiameremo **CongestionWindow**, che è un valore basato sul valore di congestione della rete percepito: possiamo quindi operare con l'algoritmo di "sliding window" visto precedentemente, sostituendo nel calcolo di **EffectiveWindow** il valore di **AdvertisedWindow** con la seguente stima:

$$MaxWindow = \min\{CongestionWindow,\ AdvertisedWindow\}$$

Ora, assumendo la trasmissione affidabile, ogni volta che scade una temporizzazione, dimezziamo il valore della **CongestionWindow**, mentre ogni volta che si presente un successo nella consegna, incrementiamo il valore di **CongestionWindow**. Vogliamo in questo algoritmo effettuare un ampiamento cauto della nostra finestra in quanto «*avere una finestra troppo ampia ha conseguente assai peggiori del fatto di averla troppo piccola: [infatti nel secondo caso] i pacchetti eliminati devono essere ritrasmessi, rendono ancora peggiore la congestione*»

2.2.2. *Partenza lenta.* Con questa "partenza sarcastica"[1], viene utilizzato per aumentare velocemente la dimensione della finestra di congestione, aumentando la finestra esponenzialmente *raddoppiando il numero di pacchetti in viaggio ad ogni RTT*, mantenendo però la diminuzione moltiplicativa. Questo algoritmo può essere utilizzato per:

- All'inizio di una connessione, quando una rete non ha idea di quanti pacchetti possano essere inviati in una rete
- Premessa: con il nuovo metodo di `ACK` introdotto da TCP, si otterrà un messaggio "cumulativo" indicante i pacchetti da trasmettere, e quindi riaprendo la sessione di trasporto[2]. Nell'altra situazione in una condizione di perdita totale dei nostri pacchetti mentre la sorgente è in attesa della scadenza del *timeout*, in quanto abbiamo come valore valido il **CongestionWindow** precedente alla perdita di pacchetti, possiamo velocemente arrivare a questo valore utilizzando la crescita esponenziale, per poi utilizzare l'incremento del suo valore ("aumento additivo"): in questo modo possiamo ristabilire velocemente il valore della banda disponibile, riducendo quindi l'impatto necativo sul *throughput* di connessione.

2.2.3. *Ritrasmissione e recupero veloce.* Anche le due integrazioni precedenti tuttavia portano a lunghi tempi di attesa durante i quali si attendeva che scadesse la temporizzazione: con la **ritrasmissione veloce**, parte di questo algoritmo, invece ogni volta che si trasmette un pacchetto, viene inviato sempre l'ultimo valore del pacchetto che è arrivato in sequenza (e non viene considerato il valore di un pacchetto successivo ma arrivato fuori sequenza). Quindi il mittente, dal valore del pacchetto ricevuto, verrà anche a conoscere il valore dell'ultimo pacchetto inviato in sequenza, e quindi il valore del pacchetto non giunto a destinazione che si deve inviare: in quanto semplicemente tale pacchetto potrebbe essere anche stato ritardato dalla rete, il mittente attende un certo numero di pacchetti duplicati, dopo i quali soddisfa la richiesta del ricevente. In questo modo si eliminano i

Con la parte di **recupero veloce**, invece di riportare `CongestionWindow` al valore unitario per poi riprendere la partenza lenta, si possono utilizzare i `ACK` per pianificare l'invio dei pacchetti, si può partire dal valore dimezzato prima della perdita dei pacchetti per poi riprendere l'aumento adattivo con queste informazioni aggiuntive, utilizzando quindi una versione del primo algoritmo modificate, utile per non ripartire ex-novo con la trasmissione. Solamente all'inizio quindi si utilizzerà l'aumento adattivo per stabilire velocemente la capacità della linea.

[1] Notare che questo è semplicemente un termine ironico, sarebbe meglio parlare di "partenza esponenziale"

[2] In questo caso si fa riferimento al primitivo algoritmo di "sliding window", con il quale non si aveva alcun modo per sapere se la destinazione poteva accettare nuovi dati oppure no, in quanto non essendoci alcun invio di informazioni dalla sorgente, non potrà pervenire nemmeno la comunicazione `ACK`

Parte 3

Appendice

CAPITOLO 10

Formulario

Definiamo:

C: Capacità della rete
D: Delay o Tempo (ritardo) di trasmissione
B: Bandwidth
RTT: Round Trip Time
P: Tempo di propagazione
T: Tempo di trasmissione
A: Tempo speso nei buffer per effettuare *store-n-forward*

Sulla latenza:

$$L = P + D + A = \frac{Dist}{c} + \frac{PckSize}{B} + A$$

Sulla capacità e RTT:

$$C = D \cdot B = \frac{RTT}{2} \cdot B$$

Sulle conversioni:

$$1Mbps = 10^3 Kbps = 10^6 bps$$
$$1MB = 2^{10} KB = 2^{20} B = 2^{23} bit$$

Teorema 1 (di Nyquist)**.** *Il massimo tasso di trasmissione di bit (ovvero bitrate) ottenibile in un canale "perfetto" (dove ovvero non avvengono errori nella trasmissione: questo è quindi un limite superiore al dato reale) con bandwidth B misurato in kHz è*

$$D = 2B \cdot log_2 k$$

dove k è il numero di livelli discreti di voltaggio del segnale (da massimizzare per ottenere bitrate maggiori)

Teorema 2 (di Shannon)**.** *Il limite superiore alla capacità di una linea di connessione in bps è una funzione del rapporto della potenza media del segnale e della potenza media del rumore della linea stessa $\frac{S}{N}$, misurato in dB, è*

$$C = B \cdot log_2 \left(1 + \frac{S}{N}\right)$$