

Reti di Calcolatori

Raccolta appunti, anno 2013/14

Sito:

<http://nptlab.di.unimi.it/>

Libro:

Fred Halsall, "Computer Networking and the Internet" — 5th edition, Addison-Wesley, 2005

Questa dispensa è una raccolta di tutti gli appunti del corso di reti tenuto dal Prof. Rossi del Dipartimento di Informatica della Università Statale di Milano. Il documento può presentare errori e NON PUO' sostituire il libro.

Introduzione

Una **rete di calcolatori** è un sistema interconnesso di calcolatori.

Questo sistema è il più distribuito del pianeta ed è costituito da due parti fondamentali:

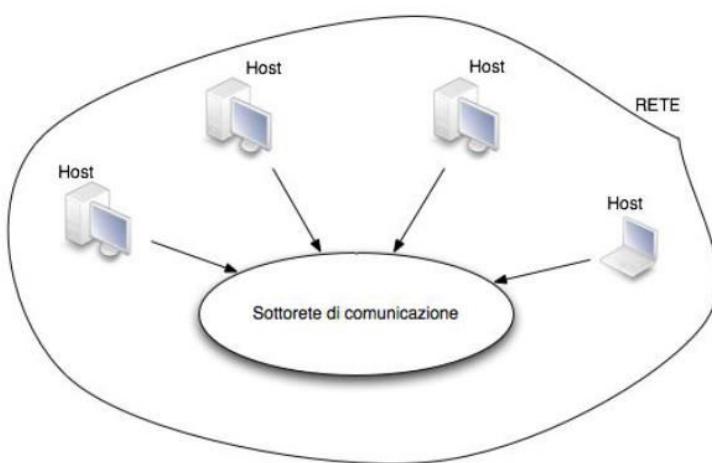
- le **macchine utente**, dette host computer (o *end system*) sono i pc, smartphone, e tutti i dispositivi connessi alla rete per specifiche funzionalità come mail, web ... Gli host possono comunicare solo tramite le sottoreti di comunicazione. Ogni host per svolgere tali attività deve essere dotato di un software di rete necessario per comunicare in rete con altri computer. Ogni host è un elaboratore che non dipende dal servizio offerto dalla rete.

- la **sottorete di comunicazione**, è un sistema che si occupa della comunicazione delle macchine tra loro. Questo sistema è composto da un insieme di protocolli software e un insieme di infrastrutture fisiche (cavi e apparati di rete) per la comunicazione.

L'oggetto di questo corso è lo studio del sistema che consente la comunicazione, dal punto di vista delle sue componenti.

Sottorete di comunicazione

Insieme di link e nodi che consentono alle macchine (per quanto remote) di comunicare l'una con l'altra. È il sottosistema.



Sottoreti nelle diverse tecnologie

I sistemi di comunicazione possono essere classificati in base all'area geografica che coprono, cioè in base all'estensione territoriale

- **LAN** (Local Area Network), rete locale che opera su uno spazio fisico omogeneo (scuola, casa, ..)
- **MAN**, rete su area geografica molto più ampia
- **WAN**, reti per città (wifi)
- **Reti interconnesse**: rete composta da elementi utilizzanti tecnologie potenzialmente diverse

Un esempio è Internet

Grande rete interconnessa. Insieme di milioni e milioni di piccole o grandi reti. Ogni rete connessa a Internet è una parte di Internet.

TCP/IP è l'elemento unificante di tutta Internet, permette di collegare tutte le reti anche molto diverse tra loro.

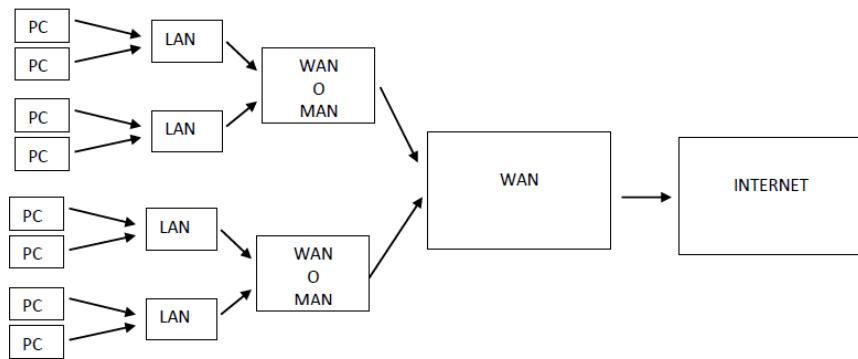
Il collegamento delle reti già esistenti avviene usando protocolli (regole)

- IP : comunicazione tra macchine
 - TCP: comunicazione tra i processi all'interno delle macchine
- (vedremo meglio in seguito quando parleremo dei livelli 3 e 4)

Qual è la differenza concettuale tra rete interconnessa e una WAN? Implica diverse tecnologie? Diversa estensione territoriale?

Si può fare una distinzione basata su estensione territoriale che in realtà implica anche una distinzione funzionale: inequivocabilmente con le tecnologie attuali sarebbe impossibile creare una rete WAN usando le stesse tecnologie usate per creare un sistema LAN.

Quindi potremmo immaginare la rete interconnessa come una o più WAN.

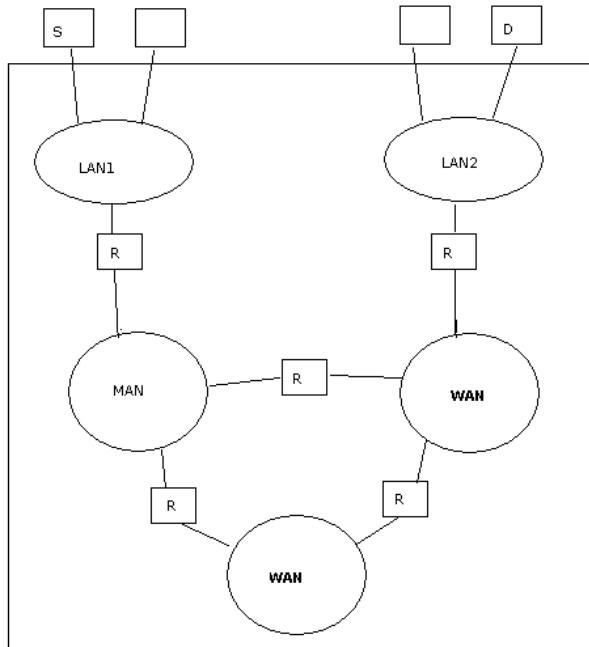


Gateway: ponte d'accesso. Un gateway (dall'inglese, *portone, passaggio*) è un dispositivo di rete che opera al livello di rete e superiori del modello ISO/OSI. Il suo scopo principale è quello di veicolare i pacchetti di rete all'esterno di una rete locale (LAN).

Gateway è un termine generico che indica il servizio di inoltro dei pacchetti verso l'esterno; il dispositivo hardware che porterà a termine questo compito è tipicamente un router.

Router: apparati diretti che permettono di passare da una rete ad un'altra rete, (dall'inglese *intradattore*) è un dispositivo elettronico che, in una rete informatica a commutazione di pacchetto, si occupa di instradare i dati, suddivisi in pacchetti, fra reti diverse. È quindi, a livello logico, un nodo interno di rete deputato alla commutazione di livello 3 del modello OSI o del *livello internet* nel modello TCP/IP.

Due host non saranno collegati direttamente, ma ci sarà un cammino nella sottorete. Bisognerà scegliere il cammino migliore.



Questa è una gerarchia di sistemi interconnessi.

TCP/IP è il modello di omogeneizzazione ed è presente su tutte le piattaforme per permettere la comunicazione.

La diversità tra le reti si materializza anche per l'aspetto topologico. Una **topologia di rete** rappresenta un modello geometrico (grafo) di una rete di telecomunicazioni i cui elementi costitutivi sono i nodi e i rami. Un nodo individua un elemento della rete connotato da specifiche funzionalità e un ramo costituisce un elemento di connessione fisica fra due nodi. Il significato di queste entità geometriche è diverso a seconda del tipo di operatività che si considera.

Le modalità di connessione dei calcolatori possono essere raggruppate in due famiglie principali:

- **reti magliate** (o punto-punto)
- **reti broadcast**

Confrontiamo la topologia punto-punto e la topologia broadcast.

Topologia punto-punto

Un link collega solo una coppia di nodi. Rappresentata come grafo, ha come nodi i router e come archi i link fisici. Ad ogni router sono collegati uno o più host. Se il grafo è completo, si parla di maglia totalmente connessa; altrimenti, di maglia parzialmente connessa.

Maglia totalmente connessa, Ogni nodo è direttamente connesso all'altro, non ci sono intermediari.

Tutti gli host possono parlare con gli altri in modo diretto, ma è difficilmente realizzabile per un numero elevato di nodi: la distanza e il costo dei link sarebbe insostenibile, e inoltre ogni nodo dovrebbe avere un numero esponenziale di interfacce. Quindi la comunicazione è diretta ma ci sono troppi cavi.

Maglia parzialmente connessa

Non tutti i nodi sono collegati fisicamente, ma ognuno è raggiungibile da un qualsiasi altro nodo con un cammino di n passi detti hop, con $n \geq 1$. Il 50% di internet usa questa topologia.

Ci sono meno cavi della maglia totalmente connessa. Non avendo un cavo diretto tra tutti i nodi, devo far sì che ogni nodo non debba preoccuparsi solo del proprio traffico, ma anche del traffico che passa attraverso di lui per raggiungere un altro host. Poiché alcuni nodi dovranno anche instradare i pacchetti, essi devono fornire la funzionalità di **routing**. La funzione di instradamento o *routing*, viene svolta dal router attraverso il protocollo IP (si sceglie la strada giusta per arrivare a destinazione). Servono algoritmi per collegare i nodi e servirà un'euristica di instradamento ("devo calcolare il cammino in base al numero di nodi o in base al tempo?").

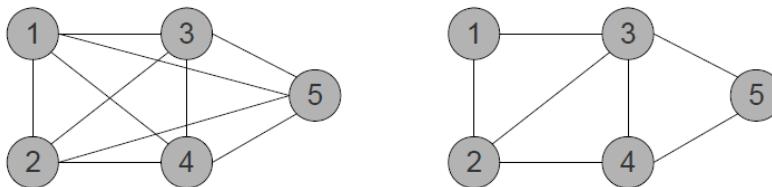


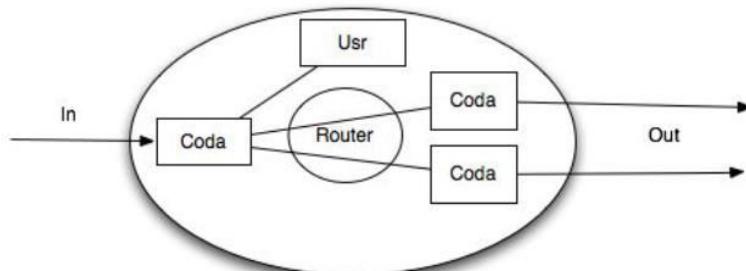
Figura 1.2: reti a tipologia magliata totalmente connessa (a sinistra) e parzialmente connessa (a destra).

Nodo di commutazione

Un nodo di commutazione (o router) presenta una linea di ingresso e una o più linee di uscita. È presente un buffer di ingresso che, esaminando un pacchetto alla volta, instrada il pacchetto verso il destinatario.

Ogni coda ha un buffer su cui vengono copiati i dati trasmessi sulla rete. Sulla linea "In" arrivano i dati destinati o per l'host/nodo corrente o per altri host/nodi a lui adiacenti. In tal caso i dati devono essere instradati nelle code opportune (buffer della macchina locale). In questo esempio i dati possono uscire verso due host diversi (ha due linee di "Out"). Il cuore del nodo, il router, è fondamentale per scegliere l'instradamento migliore, con un algoritmo opportuno. La scelta dell'instradamento migliore è dettata dalla metrica del mezzo scelto. Una delle metriche possibili potrebbe essere la velocità di connessione, il tempo necessario per il trasferimento dei dati sul link o il numero di link per arrivare a destinazione. **Hop count** indica il numero di dispositivi che bisogna attraversare per arrivare dalla sorgente alla destinazione.

Si aggiunge inoltre il problema del *Multiplexing*: sul lato in input del nodo passano informazioni di diversi nodi, queste informazioni dovranno essere temporaneamente memorizzate e poi essere inoltrate sulla giusta uscita ("store-and-forward").



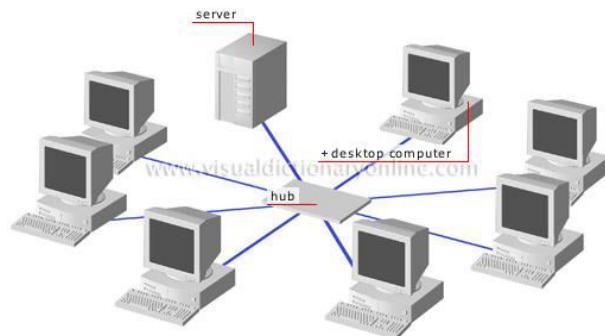
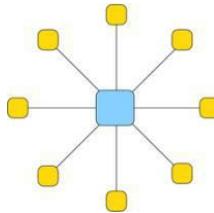
Topologia Broadcast

È un network che riceve un segnale da una linea in ingresso e lo propaga su tutte le linee d'uscita, compresa la linea d'ingresso attiva; il routing non esiste perché i dati arrivano a tutti. Quindi tutti sentono quello che viene trasmesso dalla sorgente. A sua volta anche la sorgente sente quello che dicono gli altri.

Studiamo ora alcuni modelli:

Centro stella

Sempre rappresentata come un grafo, ma i nodi sono tutti connessi a un "centro stella". Esso è definito **passivo** quando, ricevendo una informazione da una linea di ingresso, la propaga a tutte le linee di uscita, compresa quella dalla quale ha ricevuto l'informazione. Un link collega ciascun nodo. Un nodo manda un messaggio al centro che viene rimandato a tutti (intradattamento non esiste, multiplexing sì). Non sempre il messaggio deve arrivare a tutti! Si può rendere "sordo" chi non deve ricevere il messaggio.



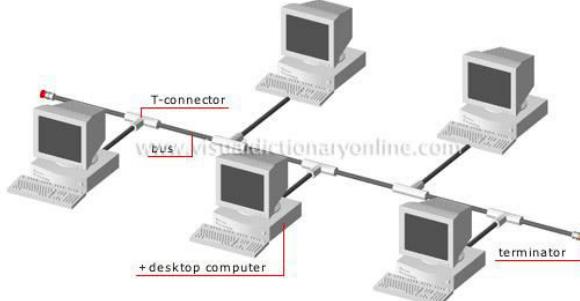
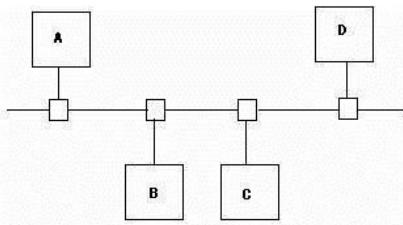
Ethernet

Il restante 50% di Internet, è caratterizzata da bus lineare, l'informazione viaggia ed entra in tutte le stazioni tramite cavo coassiale, che ha un' anima conduttrice che propaga i dati in tutte le direzioni. Prima o poi tutte le stazioni ricevono ciò che A ha trasmesso.

Parleremo della possibilità delle collisioni.

In questo tipo di rete non c'è bisogno di un router perché il flusso dei dati viene trasmesso a tutti gli host, anche quelli che non devono ricevere. Il router serve se voglio connettere la mia rete locale a Internet (è il router che collega la LAN a Internet).

La topologia dominante in una rete locale è quindi broadcast. Le altre reti sono a maglia (MAN, WAN).

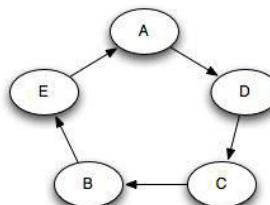


terminatore

Un tipo particolare di connettore (fatto a tappo) che chiude il percorso elettrico ai due estremi di una rete Ethernet su cavo coassiale. Monta al proprio interno un resistore che s'innesta a cavallo dei due poli del cavo coassiale al fine di scaricare il segnale che arriva e impedire che si rifletta tornando indietro sulla rete e provocando collisioni.

Anello unidirezionale (o token-ring)

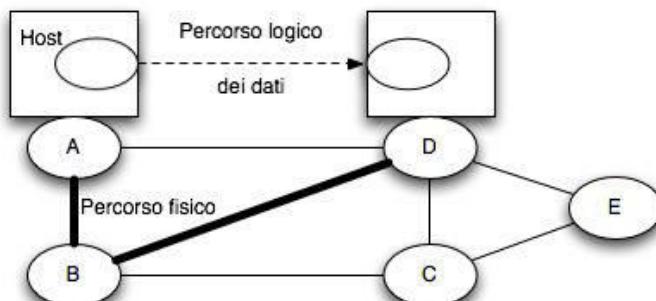
Ogni stazione è collegata alla successiva ad anello. Non ci sono collisioni. Sviluppata da IBM ma non ha avuto successo.



Commutazione in una maglia parzialmente connessa

Riprendiamo la topologia magliata (punto-punto). Dobbiamo fare una importante distinzione:

- *percorso logico*: se mando un messaggio, penso che in un passo vada dal mittente al destinatario
 - *percorso fisico*: come abbiamo già visto, in una maglia parzialmente connessa, il messaggio non è detto che arrivi in un solo passo, ma potrebbe attraversare molti nodi intermedi.
- Quindi il percorso logico \neq dal percorso fisico.

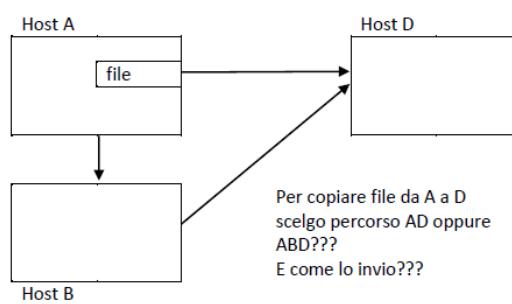


Per spedire un messaggio si commuta (*sceglie*) da una linea di uscita a una linea di ingresso (da qui il termine commutazione).

Che cosa instradiamo all'interno di un router/host? O un **messaggio**, o un **pacchetto**.

La commutazione è la funzione di instradamento del messaggio o pacchetto.

Commutazione = smistamento



Commutazione di messaggio

Un messaggio è un'unità dati, cioè un'unità elementare di informazione che viaggia sulla rete. È una serie di bit che ha un'inizio e una fine ben definiti e contenente le informazioni da trasmettere. È composto da:



- un'**intestazione**, che contiene la sequenza di bit che identificano il mittente e il destinatario (in figura: **S**=Sorgente, **D**=Destinazione),
- il **payload** = DATI
- una **coda** (T) che funge da codice rilevatore di errore (una funzione che, applicata sul messaggio ricevuto, restituisce zero se il messaggio non si è corrotto nell'invio).

[C = ulteriori informazioni di controllo, per esempio il tipo di protocollo che dovrà essere utilizzato al livello superiore. Vedremo alcuni casi per i frame ethernet, vlan ..e vedremo anche per i pacchetti di IP e segmenti di TCP]

Complessivamente i messaggi sono composti da un header (sorg, dest), una parte dati (payload) e il codice rilevatore di errori (Trailer). Essendo i dati variabili, anche il messaggio è a lunghezza variabile.

Svantaggi della commutazione di messaggio

1) Essendo un messaggio un'unità elementare (quindi non divisibile), esso può raggiungere dimensioni molto grandi e in tal caso terrebbe molto impegnata la rete. Al raggiungimento del nodo destinazione, se l'input è FIFO, un messaggio corto dovrà attendere lo smaltimento del messaggio molto lungo ricevuto precedentemente.

La gestione delle code diventa complessa: anche se decidessi di dare la precedenza ai messaggi corti, il messaggio lungo rischia di dover aspettare indefinitivamente a causa delle continue precedenze ai successivi messaggi corti che arrivano. Da qui nasce il termine **starvation** sul messaggio lungo (nella coda di un router, vi è occupazione della rete per un tempo imponente. Quello che si tenta di fare è modificare lo scheduler interno del router per prevenire questo fenomeno).

2) Difficoltà nella gestione e allocazione della memoria interna dei nodi, perché i messaggi sono partizioni variabili.

Se i messaggi fossero di dimensione predefinita si potrebbe avere una gestione efficiente e comoda di partizioni fisse.

3) Se il messaggio si corrompe durante l'invio (anche minimamente - un solo bit), il messaggio deve essere rispedito completamente. Non vengono usati codici correttori nelle reti, ma solo rilevatori.

Packet Error Rate

Invio un file intero, quindi se è 1 MB invio 1 MB.

- facile da implementare, devo però aggiungere informazioni al file: source e destination

SOURCE	DESTINATION	FILE
Header		Dato

Devo creare regole (*protocolli*), per esempio, la dimensione dell'header deve essere uguale per tutti.

In base alla tabella di instradamento invio sul percorso giusto. Quando il nodo riceve il file, controlla se è lui il destinatario:

- se sì, elabora il file
- se no, ricontrolla la tabella di instradamento e riinvia il dato

Problema:

- corruzione per errore (se un solo bit è rovinato, tutto il file è da considerarsi rovinato)

Soluzione:

- rimando il file più volte
- oppure spezzo il file in più pacchetti

Oggi, in media, ho una probabilità di errore P di 10^{-12} :

Probabilità di errore su un blocco di dimensione N è:

$$\text{Packet Error Rate, } P_{er} = 1 - (1 - b_{er})^n \cong n \cdot b_{er}$$

b_{er} , indica la probabilità di errore sul singolo bit

n descrive il numero di bit trasmessi

Usare i messaggi è svantaggioso, si preferisce usare i pacchetti.

Commutazione di pacchetto

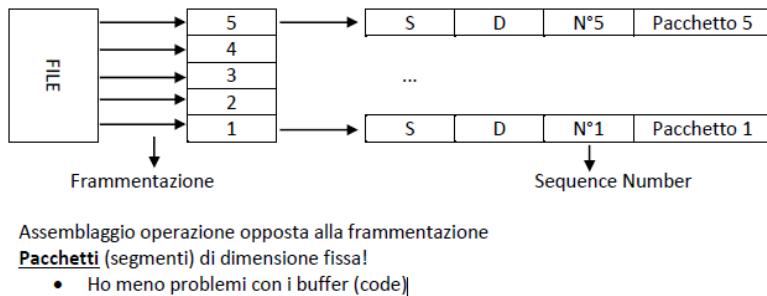
I pacchetti sono migliori sotto ogni punto di vista, infatti:

- 1) Garantiscono meglio la *fairness* nelle code (*fairness* = equità) tutti i pacchetti hanno uguale dimensione, quindi spedisco in modo imparziale e non ho problemi di gestione della memoria
- 2) L'allocazione della memoria è implementata in partizioni fisse (i pacchetti hanno grandezza predefinita)
- 3) In caso di errore, non bisogna ritrasmettere tutto il messaggio, ma solo il pacchetto danneggiato.

Abbiamo comunque qualche svantaggio:

- necessaria funzionalità di decomposizione/ricomposizione
- ogni pacchetto ha la stessa intestazione, quindi genero traffico ridondante
- tempo di arrivo e ordine dei pacchetti non predicibile

Nel trasferimento, i nodi della maglia (che sono le entità che forniscono la funzionalità di routing, e a cui sono collegati uno o più host) si occupano quindi di instradare i pacchetti, mentre, host sorgente e host destinazione si occupano della frammentazione/composizione. È necessario un buffer a destinazione per ricomporre, nell'ordine corretto, tutti i pacchetti ricevuti, perché non è certo che i pacchetti arrivino nel giusto ordine (possono prendere strade diverse oppure possono essere corrotti e quindi devono essere rispediti).



Un pacchetto è formato da:

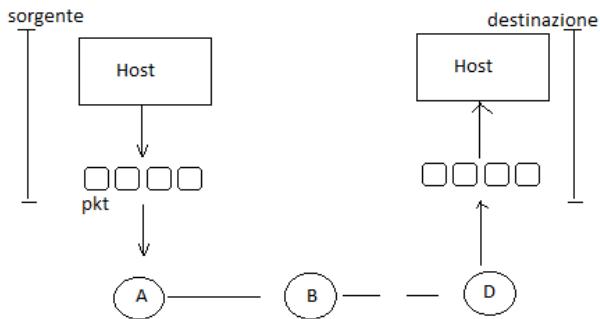
- **header**, intestazione
- **payload**, campo dati

Servono due nuove funzioni:

- Frammentare il messaggio, lato sorgente
- Ricomporre il messaggio, lato destinazione (inoltre bufferizza il contenuto dei pacchetti per ricomporre l'intero messaggio).

Studiamo i tempi di invio e ricezione dei pacchetti

Prendiamo in considerazione la seguente situazione dei nodi: A → B → D

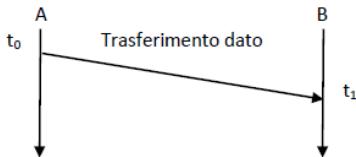


Posso calcolare il tempo in cui tutti i pacchetti arrivano a D, partendo da A?

In generale, da un nodo A a un nodo B ...



passa un tempo:



$t_1 > t_0$, infatti:

$$t_1 - t_0 = t_x + t_{elaborazione} + t_p$$

Dove, $t_x \rightarrow$ tempo di trasmissione (proporzionale alle dimensioni del pacchetto)

$$T = t_{coda} + t_{elaborazione} + t_{trasmissione(tx)} + t_{propagazione}$$

T è detta "Latenza di trasmissione": tempo trascorso tra invio e ricezione del messaggio. È variabile e non determinabile a priori a causa del numero di hop della rete, dalla distanza tra sorgente e destinazione e dal carico/traffico della rete (con tanto traffico aumenta il tempo di permanenza in coda).

T rappresenta il tempo di attesa tra trasmissione e ricezione di tutti i pacchetti di un messaggio.

Si può definire anche come $t_1 - t_0$, dove t_0 è il tempo in cui il mittente ha inviato il primo pacchetto e t_1 è il tempo in cui il destinatario ha ricevuto l'intero messaggio.

$t_{propagazione}$ è il tempo che ci impiega il pacchetto ad arrivare a destinazione attraverso il mezzo fisico (ciascuno dei quali ha una specifica velocità di propagazione)

t_{coda} è il tempo di permanenza in coda nel nodo (sia input che output)

$t_{elaborazione}$ è il tempo che serve al nodo per decidere su quale strada inoltrare il pkt

$t_{trasmissione(tx)}$ tempo per trasferire i dati sul mezzo fisico. I cavi della rete sono cavi seriali, a ogni clock di trasmissione viene inserito un bit nel canale di comunicazione che collega un nodo a un altro.

Notiamo che:

- t_{coda} e $t_{elaborazione}$ sono generalmente trascurabili

$$- t_{propagazione} = \frac{distanza}{vel.mezzo}$$

dove :

$vel. mezzo = 3 \cdot 10^8$ se si utilizza fibra ottica

$vel. mezzo = 2 \cdot 10^8$ se si utilizza cavo in rame

Esempio

$$t_p = \frac{\text{lunghezza}}{\text{propagazione mezzo}} \quad \text{es: } \frac{10^5}{2 \cdot 10^8} = 5 \cdot 10^{-3}$$

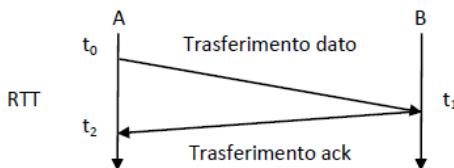
Il tempo di trasmissione è il tempo che serve per mettere i bit sul mezzo fisico. È dato da:

$$T_x = \frac{\text{dim pacchetto}}{\text{vel tx}} \quad , \text{ dove vel. tx (o bwth) incide pesantemente sul tempo finale}$$

$$t_x = \frac{P_{\text{size}}}{\text{bitrate}} \quad \text{es: } \frac{1000 \text{ bit}}{10 \frac{\text{MB}}{\text{s}}} = \frac{10^3}{10^7} = 10^{-4}$$

Come facciamo a far sapere al mittente che il pacchetto è arrivato?

- Mandiamo un ricevuta di lettura (messaggio di ack)



All'arrivo del pacchetto, il destinatario invia un pacchetto particolare chiamato ACK, che informa A che il pacchetto è giunto a destinazione. In questo caso l'ACK arriva ad A al tempo t_2 .

Il tempo $t_2 - t_0$ è chiamato Round Trip Time (RTT), detto anche tempo di andata e ritorno (PING).

Allora $(RTT/2)$ sarà circa il tempo di latenza (T) della rete in quel momento.

Esercizio [Ex.1.4]

Un blocco di dati da 1000 bit deve essere trasmesso tra due calcolatori. Determinare T_p e T_x per i seguenti link:

- 100 mt. di doppino con transmission rate di 10 Kbps
2. 10 Km di cavo coassiale con transmission rate di 1 Mbps
3. 50000 Km di canale satellitare con transmission rate di 10 Mbps

SOL: $T_p = \text{lungh.mezzo} / \text{vel.prop.}$; $T_x = \text{bit dati} / \text{bitrate link}$. Perciò:

$$1. T_p = 0.100 / 200000 = 5 \times 10^{-7} \text{ s.}$$

$$Tx = 1000 / 10000 = 0.1 \text{ s.}$$

$$2. T_p = 10 / 200000 = 5 \times 10^{-5} \text{ s.}$$

$$Tx = 1000 / 1000000 = 0.001 \text{ s.}$$

$$3. T_p = 50000 / 300000 = 0.167 \text{ s.}$$

$$Tx = 1000 / 10000000 = 0.0001 \text{ s.}$$

Esercizio

[Ex.1.2] Determinare massimo ritardo di propagazione associato con i seguenti canali di comunicazione:

1. connessione attraverso linea telefonica di 1 Km
2. connessione attraverso PSTN di 200 Km
3. connessione via canale satellitare di 50000 Km

SOL: Nei primi due casi il mezzo è in rame, perciò $v_p = 200000 \text{ Km/s}$; nel terzo caso è $v_p = 300000 \text{ Km/s}$. Perciò:

$$1. T_p = 1 \text{ Km} / 200000 \text{ Km/s} = 5 \times 10^{-6} \text{ s.}$$

$$2. T_p = 200 \text{ Km} / 200000 \text{ Km/s} = 10^{-3} \text{ s.}$$

$$3. T_p = 50000 \text{ Km} / 300000 \text{ Km/s} = 1.67 \times 10^{-1} \text{ s.}$$

Esercizio

Una stazione deve trasmettere un msg di 1.5 Mb su un path di lunghezza 4 hop che la collega al destinatario, e ogni hop è costituito da un link

con bwth 1 Mbps. Si supponga che i tempi di propagazione, elaborazione dati e accodamento ai router siano trascurabili. Qual è il ritardo di trasmissione end-to-end se la rete adotta rispettivamente commutazione di messaggio o di pkt?

Nel secondo caso si assuma che i dati vengano frammentati in pkt da 1500 bit.

SOL:



$$(1.5 \text{ Mb} = 1.5 \times 10^6 \text{ b})$$

• *msg switching*: $[Tx_1msg \cdot \#hop]$

Calcolo Tx per un hop:

$$1.5 \text{ Mb} / 1 \text{ Mbps} = 1.5 \text{ sec}$$

Ora calcolo per i 4 hop:

$$1.5 \text{ sec} \cdot 4 \text{ hop} = 6 \text{ sec.} = T$$

• *pkt switching*:

Calcolo quanti pacchetti trasmettere: $[frame/fragment_size=\#_pkt]$.

$$1.5 \text{ Mb} / 1500 \text{ b} = 1000 \text{ pkt da trasmettere.}$$

Calcolo Tx per un pacchetto per un hop: $[Tx_1pkt = f/b]$

$$\text{Ogni pkt ha tempo di trasmissione } 1500 \text{ b} / 1 \text{ Mbps} = 1.5 \text{ ms.}$$

Devo calcolare il Tx dell'ultimo pkt:

1- Primo pkt inviato a tempo 0;

2- Secondo pkt inviato a tempo 1.5 ms;

3- Terzo pkt inviato a 3 ms

4- ...

$$1000 - \text{Millesimo pkt inviato a } 999 \cdot 1.5 \text{ ms} = 1498,5 \text{ msec. } [Tx_last_pkt = (\#_pkt - 1) \cdot Tx_1pkt]$$

[come da testo, il tempo di propagazione è trascurato]

Calcolando per 4 hop :

$$\text{il 1000esimo pkt è ricevuto a destinazione entro: } [Tx_last_pkt + (\#_hop \cdot Tx_pkt)]$$

$$T = 1.498,5 \text{ msec} + (4 \cdot 1.5 \text{ ms}) = 1504,5 \text{ ms} = 1,5 \text{ s}$$

Esercizio

Sia data una rete tale che ha un path di 10 hop da sorgente a destinazione, e un ritardo di propagazione di 100 μ sec per hop. La banda dei canali è di 2 Mbps. Un'applicazione deve mandare 50 Mb di dati. Il ritardo e2e (end-to-end) è inferiore se la rete è a commutazione di circuito (assumendo 10 ms per il set-up) o se è a commutazione di pkt (con pkt size 1 Kb)?

SOL:

- *Commutazione di circuito:*

$$T = 10 \text{ msec} + (50 \text{ Mb} / 2 \text{ Mbps}) + (10 \cdot 10^{-4} \text{ sec}) = 25.011 \text{ sec}$$

$$[T = setup + (Tx) + (\#hop \cdot Tp)]$$

$$\text{NB. } 1 \text{ usec} = 1 \cdot 10^{-6} \text{ s}$$

$$100 \text{ usec} = 10^{-4} \text{ s} = 0.0001 \text{ s}$$

- *Commutazione di Pkt:*

Tempo trasmissione di 1 pkt è :

$$1 \text{ Kb} / 2 \text{ Mbps} = 0.5 \text{ msec; } [Tx_pkt= pkt-size / bwth]$$

$$(\text{cioè: } 1.000 \text{ b} / 2 \cdot 1.000.000 \text{ b/s} = 0,0005 \text{ s} = 0.5 \text{ msec})$$

Questo è il tempo necessario per trasmettere dalla sorgente il primo pkt, ma sarà necessario un ulteriore ritardo per essere ricevuto a destinazione.

A ogni hop, un pkt spende 0.5 msec per ritrasmissione (Tx) e 0.0001 sec per propagazione (Tp), infatti $Tp = 100 \text{ usec} = (10 \cdot 10^{-4} \text{ sec})$

Vediamo quanti pacchetti devo inviare: $50 \text{ Mb} / 1 \text{ Kb} = 50.000 \text{ pkt in totale.}$

$$\text{L'ultimo pkt è trasmesso a } 49999 \cdot (0.5 \text{ msec}) = 24.99 \text{ sec. } [\# \text{ pkt} - 1 \cdot (Tx_pkt)]$$

$$\text{La destinazione riceve l'ultimo pkt al tempo } T = 24.99 \text{ s} + (0.0005 \text{ s} \cdot 10) + (0.0001 \text{ s} \cdot 10) = 24.996 \text{ sec}$$

$$[T = setup + (Tx_last_pkt) + (Tx_pkt \cdot \#hop) + (Tp_pkt \cdot \#hop)]$$

Conviene pkt switching. Infatti: 25.011 sec > 24.996 sec

Se il tempo di set-up per commutazione di circuito è minore di $(k-1)p/b$ [con k #pkt, p pkt size e b bwth], allora conviene circuit-switching.

0.001 s (setup-time) < ?? 24.99 sec $(k-1)p/b$

Es: $k=100 \dots ? ? ?$ [DA SISTEMARE]

Esercizio

A 256 Kbps quanti bit passano in 10 ms?

SOL

$$Tx = f/b$$

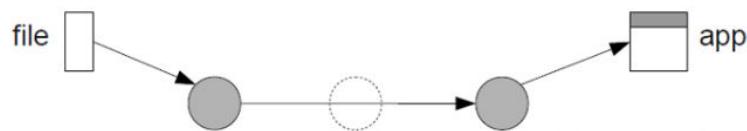
$$F = Tx \times b = (10 \times 10^{-3}) \times (256 \times 10^3 \text{ b/s})$$

$256 \times 10 = 2560$ bit che è circa 2,5 Kb per ogni 10 ms

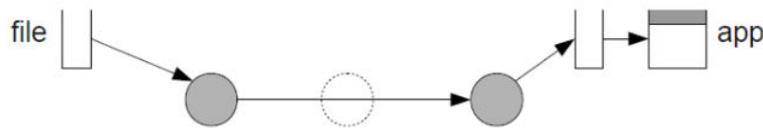
Traffico in rete e jitter

Il traffico realtime è un particolare tipo di traffico che richiede erogazione di pacchetti a velocità costante. Si pensi ad esempio ad una telefonata via Skype: se un pacchetto arriva in ritardo si viene a creare un "buco" nella ricezione; in questo caso la varianza ammissibile è nell'ordine del millisecondo.

In una trasmissione realtime i pacchetti sono direttamente elaborati dall'applicazione client man mano che sono ricevuti.



Il traffico non realtime riguarda invece il tradizionale scambio di dati sulla rete, come files, posta, web... Questi tipi di transazione non pongono particolari vincoli sui jitter di trasmissione; infatti, i files devono in ogni caso essere scaricati per intero prima di essere elaborati da un'applicazione. Quindi l'applicazione non elabora i singoli pacchetti, ma l'intero file, che deve quindi essere completamente ricostruito prima di poter essere aperto.



Il traffico in rete dei contenuti multimediali (audio e video) è molto più complesso da gestire rispetto a quello di dati, file, email ... Il traffico più complicato da gestire è quello vocale.

La voce è campionata e scomposta in bit. Quindi anche i messaggi audio sono scomposti in pacchetti.

Vincolo: se ho un'unità informativa X e la si processa in Y che è la rappresentazione digitale di X, ogni campionamento per secondo deve essere erogato esattamente ogni μs .

È impossibile che Internet garantisca un tempo di latenza costante. Ci sarà un delay tra i vari vincoli. Questa varianza si chiama Jitter.

Esempio

Supponiamo che la sorgente invii in un tempo T_1 il primo pacchetto di un file multimediale

In un tempo T_2 verrà inviato il secondo pacchetto.

Ovviamente $T_1 \neq T_2$ in quanto non sono predibili i tempi di arrivo alla destinazione.

Possiamo allora scrivere che $T_1 = T_2 + a$, dove a è detto jitter.

Jitter

parametro dominante delle trasmissioni streaming audio/video. È la varianza sul ritardo Tx. Non si tengono in considerazione i pacchetti errati, viene tollerato un possibile disturbo).

Per compensare al jitter si usano nella destinazione dei buffer di *play-out* per immagazzinare alcuni pacchetti, nel caso ci siano dei ritardi nella trasmissione.

Nella trasmissione audio-video, se un pacchetto è corrotto, esso viene eliminato (*package drop*), in quanto la perdita di un singolo pacchetto non comporta effetti collaterali all'utente finale (es. voce smorzata in una conversazione, piccola attenuazione di colore in un video ecc...).

Constant Bit Rate e Variable Bit Rate

Dal punto di vista delle loro caratteristiche di emissione, le sorgenti di informazione possono essere a ritmo binario costante (CBR – Constant Bit Rate) e a ritmo binario variabile (VBR – Variable Bit Rate).

Per una sorgente VBR il ritmo binario emesso varia nel tempo tra un valore massimo (il ritmo binario di picco) e un valore minimo, che può essere anche nullo.

Una sorgente CBR è caratterizzata dal solo ritmo binario di picco.

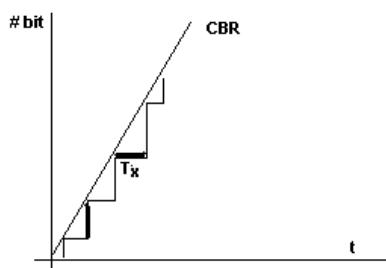
La variabilità di una sorgente VBR è dovuta alla possibile variazione nel tempo del contenuto informativo all'ingresso del codificatore.

Conviene quindi realizzare una codifica che tiene conto di questa variazione in modo da utilizzare in modo più efficiente la capacità richiesta per trasferire l'informazione: quando il contenuto informativo in ingresso si riduce, il codificatore riduce il ritmo binario prodotto in uscita.

In una trasmissione, le sorgenti CBR sono adottate generalmente per la fonia digitalizzata e il traffico realtime, mentre le sorgenti VBR per flussi video digitale (streaming), dove si alternano immagini a basso movimento e colore abbastanza uniforme ad immagini ad alto movimento e colore vario, con audio associato intermittente.

La banda massima necessaria per trasportare flussi multimediali o solamente video è sempre molto maggiore della banda massima per trasportare il solo flusso vocale a causa della differente quantità di informazione tra i due flussi.

Tecniche di compressione dati (ovvero codifica di sorgente) sono in grado di abbassare, anche notevolmente, la banda effettiva necessaria alla trasmissione.



Trasmissione digitale

Un **sistema di rete** consiste essenzialmente in due o più apparati in grado di trasmettere una sequenza di bit su un mezzo fisico. Una trasmissione via cavo può essere schematizzata con la seguente rappresentazione:

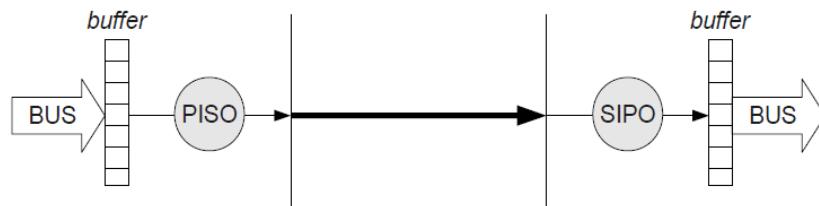


Figura 1.7: due apparati connessi in rete con un cavo seriale

Il segnale passa dal *bus* dati del primo apparato (si osservi che il *bus* trasferisce, in parallelo, un gruppo di bit per ogni colpo di clock) ad un *buffer* di uscita; un'unità PISO (*Parallel Input Serial Output*) serializza il contenuto del *buffer* e lo pone sul cavo. Il secondo apparato riceve, parallelizza l'*input* con un'unità SIPO (*Serial Input Parallel Output*), e pone i dati sul *bus* interno della macchina.

L'obiettivo della trasmissione è far giungere una sequenza ordinata di bit da un estremo ad un altro del mezzo fisico scelto (in questo caso il cavo di rete). In una **trasmissione digitale** (o **trasmissione banda base**) [sincrona?] la portante è un segnale nel discreto, ovvero può assumere i soli valori zero e uno; al contrario, in una **trasmissione analogica** [asincrona?] la portante è caratterizzata da valori nel continuo.

Nonostante le comunicazioni di rete siano per natura trasmissioni digitali, è frequente che si verifichi una distorsione del segnale nel mezzo fisico proporzionale alla lunghezza del tratto da percorrere. Con la tecnologia della fibra ottica si riescono ad ottenere delle comunicazioni anche in banda base.

Una trasmissione digitale può essere rappresentata in questo modo:

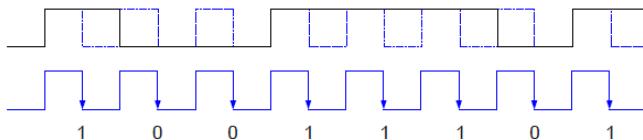


Figura 1.8: trasmissione digitale della sequenza di bit "10011101"; la linea blu rappresenta la portante, mentre quella nera il segnale effettivamente trasmesso.

Si nota che, leggendo il segnale (nero) ad ogni fronte di discesa della portante (blu) si ottiene la sequenza di bit trasmessa.

Sincronizzazione

Nella trasmissione asincrona i due sistemi che stanno dialogando fra loro usano due clock diversi per cui non si può garantire che i due clock siano in fase o siano alla stessa frequenza. Il ricevitore non ha alcuna informazione sulla temporizzazione con cui arrivano i dati. Per garantire che trasmettitore e ricevitore siano sincronizzati, le informazioni da inviare sono suddivise in piccoli blocchi di bit che sono precedute e seguite da bit di sincronizzazione detti bit di start e di stop.

Il periodo della portante che viene detto *clock di trasmissione*, rappresenta la frequenza di invio dei dati, e determina quindi la velocità di connessione. Spesso si usa il termine *baud rate* per riferirsi alla frequenza del clock. Si osservi che, per come abbiamo descritto la trasmissione digitale, si potrebbe creare un'equivalenza tra il termine **baud rate** ed il termine **bit rate**; infatti fino ad ora abbiamo trasferito un bit ad ogni *baud* (ovvero ad ogni periodo del clock). Questo tuttavia non è scontato: si potrebbe, ad esempio, immaginare un sistema di trasmissione in cui si differenziano 8 diversi valori di tensione. In questo caso, ad ogni colpo di clock della portante, il segnale trasmesso può variare entro 8 valori e quindi può codificare $\log_2 8$ (=3) bit.

Anche la ricezione dei dati avviene sulla base di una portante, il clock di ricezione. È intuitivo che perché si verifichi una comunicazione tra un mittente e un ricevente, il clock di trasmissione deve essere uguale a quello di ricezione.

È necessario allora un metodo per sincronizzare i due clock in modo che il fronte di discesa si trovi il più possibile al centro del segnale trasmesso. Questo è particolarmente importante, in quanto come già detto, i segnali subiscono un'inevitabile distorsione durante la trasmissione e i bordi risultano spesso imprecisi.

Una semplice tecnica di sincronizzazione potrebbe essere quella di impostare il clock di ricezione ad una frequenza più alta rispetto a quella di trasmissione. Le trasmissioni ethernet utilizzano una famosa codifica per le trasmissioni in banda base, detta *codifica Manchester*. In questo caso è previsto che la velocità del clock sia doppia rispetto ai bit trasmessi. Manchester infatti garantisce una transizione di stato a metà di ogni bit.

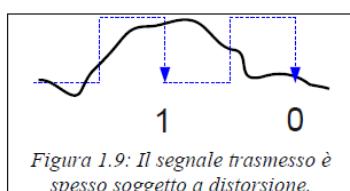


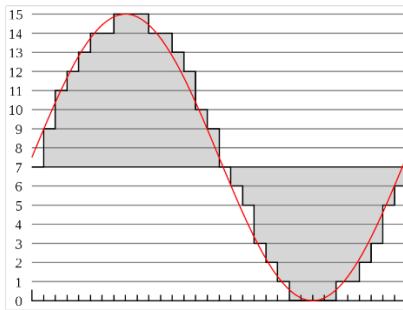
Figura 1.9: Il segnale trasmesso è spesso soggetto a distorsione.

Conversione di segnali analogici: campionamento PCM

PCM – Pulse Code Modulation

In elettronica e telecomunicazioni la pulse-code modulation (letteralmente "modulazione a codice di impulsi"), in acronimo PCM, è un metodo di rappresentazione digitale di un segnale analogico. Il metodo utilizza un campionamento dell'ascissa del segnale a intervalli regolari; i valori letti vengono poi quantizzati in ordinata ed infine digitalizzati (in genere codificati in forma binaria). La PCM è ampiamente utilizzata nei sistemi di telefonia, ma si basano su questo principio anche molti standard video, come l'ITU-R BT.601. Poiché la PCM pura richiede un bitrate molto

elevato, gli standard video di consumo come DVD o DVR sono basati su sue varianti che fanno uso di tecniche di compressione. Molto frequentemente, la codifica PCM viene impiegata per facilitare le trasmissioni digitali in forma seriale.



La modulazione a impulso, si occupa di trasformare un segnale analogico in uno digitale.

Si effettua per mezzo del campionamento del segnale al doppio della velocità del segnale.

Se ho una velocità di 4000 b/s devo eseguire una campionatura di 8000 volte. Quindi 1 per ogni 125 us x 8 bit = 64 Kbps.

$$1/4 = 0,25$$

$$0,25/2 = 0,125$$

$$1/8000 = 0,125$$

$$8000 \text{ volte/s} \times 8 \text{ bit} = 64 \text{ Kbps}$$

Nb. 1 campionatura ogni 125 us significa

$$0,125 \text{ sec} \times 10^{-3} \times 8 \times 10^3 = 1 \text{ s}$$

Per avere un segnale in uscita uguale al segnale in entrata, devo eseguire una de campionatura ogni 125 us.

Ogni campione è un byte e viene prodotto ogni 125 us.

Il motivo principale che portò alla campionatura dei segnali era la necessità di far passare su un unico cavo diversi campioni provenienti da diverse fonti telegrafiche. La TDM (time-division multiplexing) fu inventata nel [1853](#), dallo statunitense M.B. Farmer.

ATM (Asynchronous Transfer Mode - modalità di trasferimento asincrona)

Una tecnologia a commutazione di cella capace di trasmettere dati, voce e video. Le velocità variano da 1,5 Mbps a 622 Mbps. Si tratta in altri termini di un sistema di asynchronous time division multiplexing, vale a dire multiplazione statistica a divisione di tempo in modalità asincrona.

Time Division Multiplexing

In telecomunicazioni la **multiplazione a divisione di tempo**, più conosciuta come **time-division multiplexing** (corrispondente termine di lingua inglese), acronimo **TDM**, è una tecnica di multiplazione ovvero di condivisione di un canale di comunicazione secondo la quale ogni dispositivo ricetrasmettente ottiene a turno l'uso esclusivo del canale (e delle risorse ad esso dedicate cioè ad esempio l'intera banda) per un breve lasso di tempo (125 µs nello standard E0).

Per rafforzare il fatto che l'allocatione del tempo del canale è rigida si usano anche i termini **multiplazione a divisione di tempo sincrono** (STDM) o **multiplazione a divisione di tempo quantizzato** (QTDM).

Divido la mia banda del canale di trasmissione. Faccio condividere tante connessioni su un unico cavo ottico con parole lunghe 193 bit che passano alla frequenza di 125 us.

$$192/8 = 24 \text{ canali}$$

In 192 bit ci stanno 24 canali da 8 bit per cui 24 campionamenti.

Il 193-esimo bit è un bit di controllo.

Ogni canale trasmette 1,544 Mbps (=24 x 8 bit x 8000 campionamenti)

Posso cambiare anche il numero di canali mantenendo lo slot time sempre di 125 us.

$$T2=T1\times 4$$

$$T3=T2\times 7$$

In europa si usa E1 a 32 canali

$$E1=32 \text{ canali} \times 8 \text{ bit} \times 8000 \text{ campionamenti} = 2,046 \text{ Mbps}$$

In molte tecnologie basate sulla multiplazione a divisione di tempo, sono necessari appositi apparati detti multiplatore/demultiplatori (o MUX/DEMUX), posti agli estremi del circuito ad alta capacità da gestire, che si occupano di collegare il flusso di dati ad alta capacità con i numerosi circuiti in cui esso è diviso (detti flussi tributari).

Questi apparati devono copiare dati numerici a velocità anche molto alte, e la velocità dei circuiti elettronici per eseguire queste operazioni costituisce un limite alle velocità dei collegamenti che possono essere gestiti con questa tecnica.

In altre tecnologie, tipicamente utilizzate in ambito di rete locale, non esistono apparati di MUX/DEMUX, e l'accesso sincrono al canale condiviso è governato mediante algoritmi di coordinamento tra le stazioni che devono trasmettere come gli algoritmi di CSMA.

Nel caso telefonico vengono utilizzati sistemi mux a livelli gerarchici (portante-E): il primo livello base permette di gestire 30 utenti circa, i livelli successivi sono realizzati da più livelli precedenti usati in gruppi da 4.

Una prerogativa di questo tipo di multiplazione è la necessità di sincronizzazione da parte del ricevitore/trasmettitore nella trama e nello slot di competenza.

Sottorete di comunicazione

Vogliamo descrivere le sottoreti in base alle funzionalità necessarie alla loro realizzazione. Abbiamo bisogno di diversi *componenti* e di diverse *funzionalità* (o meglio *protocolli*).

Parlando prima dei componenti abbiamo:

- Host
Essi hanno protocolli orientati alle applicazioni: *frammentazione/ricomposizione* dei pacchetti
- Router/nodo
Essi hanno protocolli orientati alla sottorete: *intradamento*
I router permettono agli host di comunicare
Uno o più host sono collegati a un router/nodo

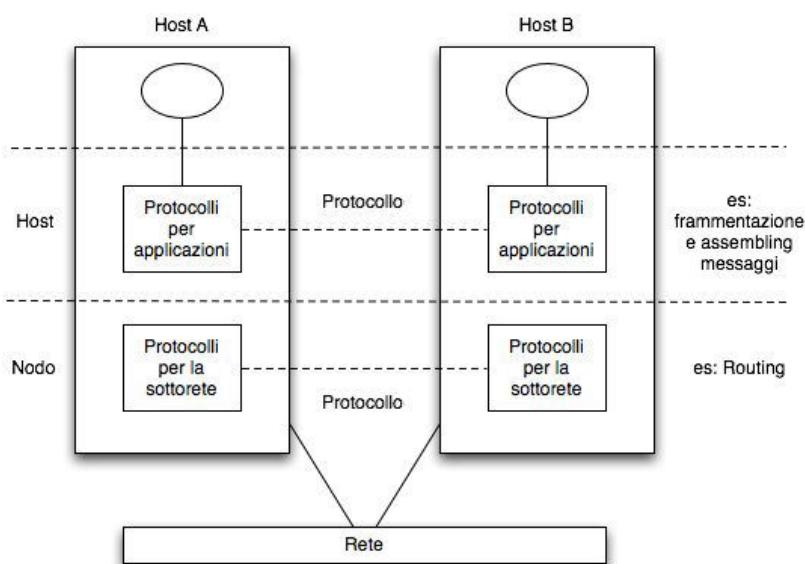
Un **protocollo** è un insieme di regole per la realizzazione di una funzionalità.

Queste regole devono essere rispettate da tutti i partner (nodi e host) che comunicano fra di loro.

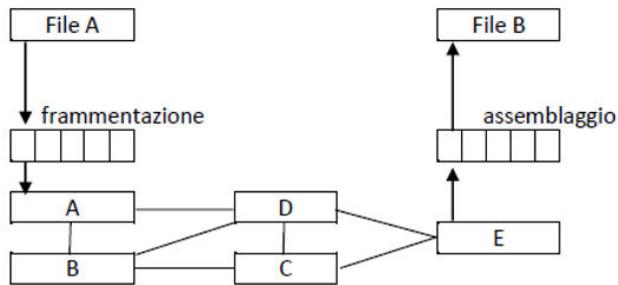
I protocolli permettono a due o più entità diverse di comunicare in una “lingua” che viene capita e rispettata da tutte le parti in gioco. I protocolli sono indispensabili per il coordinamento di tutti gli apparati della rete.

Un protocollo può essere inoltre visto come una macchina a stati che gestisce un insieme di eventi.

Noi inizieremo dai protocolli più a basso livello, più vicini al mezzo fisico, per poi salire fino alle applicazioni finali che l'utente utilizza.



Possiamo definire una sottorete come un insieme di nodi e archi, mentre una rete come un insieme di host e nodi. Ogni host ha una parte per far comunicare processi e una per comunicare con altri nodi: possono essere usati 2 protocolli diversi. La cosa importante è che la comunicazione tra nodi sia uguale per ciascuno di essi.



Chi si occupa della correttezza della trasmissione (ovvero che i pacchetti arrivino tutti alla destinazione, non duplicati e in ordine)?

Ho due alternative:

1. Gestione del traffico da parte della sottorete

In questo caso bisognerà fare in modo che i router possano comunicare tra loro, ma soprattutto che per ogni sessione utente all'interno di ogni router, e per ogni link, dev'essere presente un descrittore per tener traccia delle informazioni dei pacchetti ed eventuali copie

2. Gestione del traffico da parte degli hosts

Ho solo 2 interlocutori che sono anche i diretti interessati, quindi la comunicazione risulta molto più semplificata rispetto a prima, e soprattutto ho un uso minimo di risorse. I due hosts infatti hanno già il buffer da riempire con il messaggio (risparmio sulle risorse). Il vantaggio principale di questa politica è che la sottorete risulta essere molto più leggera.

Una sottorete di questo tipo è detta *datagram* (o best-effort) e si occupa solo dell'instradamento dei pacchetti.

Si evince che la sottorete può allora essere di due tipi:

1 - sottorete affidabile

- Il *circuito virtuale* (daremo la definizione) deve rispettare l'ordine dei pacchetti
- Ogni nodo intermedio deve controllare l'integrità e l'ordine dei pacchetti
- Metodo utilizzato in passato, quando la tecnologia non era efficiente

2 - sottorete non affidabile

- La sottorete *datagram* trasmette soltanto
- Controlli, frammentazione e assemblaggio sono compito degli host
- Ogni singolo pacchetto è mandato come entità autonoma
- Metodo utilizzato oggi in quanto la tecnologia del mezzo utilizzata (fibra) è molto più efficiente

Circuito virtuale

La sottorete può garantire la correttezza dei pacchetti trasmessi, l'ordine con cui vengono ricevuti a destinazione e che non ci siano duplicazioni. Questo può essere realizzato per mezzo di circuiti virtuali in cui ogni pacchetto è identificato dalla sessione utente, ogni nodo costruisce strutture dati dedicate e tutti i pacchetti percorrono lo stesso percorso. In questo modo il circuito è fissato. Alla fine della trasmissione bisogna chiudere il circuito. Questo metodo è giustificato soltanto se il tasso di errore dei link è alto.

Sottorete Datagram

Le sottoreti di oggi sono datagram e l'affidabilità è a livello transport.

L'affidabilità è implementata a livello di host da TCP, non è più un compito della sottorete.

La sottorete non garantisce la correttezza dei dati (consegna in ordine e non duplicati). Ogni pacchetto abbiamo detto che viene considerato come entità autonoma e ciascuno di essi può seguire percorsi diversi per arrivare a destinazione.

Fino a qualche anno fa, la funzione di affidabilità apparteneva anche al livello 2 perché si utilizzavano ancora i doppini telefonici. La principale differenza tra rendere affidabile a livello 2 o a livello 4 è il tempo di ricezione dello scambio avvenuto.

A livello 2 so contestualizzare il RTT che a livello 4 non posso invece stabilire.

Questa scelta porta ad avere una struttura della sottorete leggera, facile da gestire e che concentra le sue funzionalità sull'unico vero scopo che è l'instradamento dei pacchetti (disinteressandosi della sessione utente).

Una sottorete IP si definisce allora sottorete con servizio *datagram*, o rete *best-effort*.

Questa scelta è motivata anche dal fatto che le sottoreti attuali hanno link più affidabili e bassa probabilità di errore (fibre ottiche). Piuttosto ora la probabilità più alta di perdita di pacchetti è causata dai buffer pieni dei nodi (congestione della rete).

Durante la trasmissione, la coda dei router potrebbe essere piena e quindi il pacchetto verrebbe scartato. Il router, accorgendosi che la coda è piena, potrebbe avvertire il mittente di rallentare. Ma chi sviluppa questa funzionalità e la mette in atto?

La forniscono gli host, in quanto, al contrario, sarebbe necessario che ogni router intermedio contenga un descrittore (per ogni scambio tra due host) e quindi la rete diventerebbe molto pesante. Il protocollo che fornisce questa funzionalità è sempre TCP.

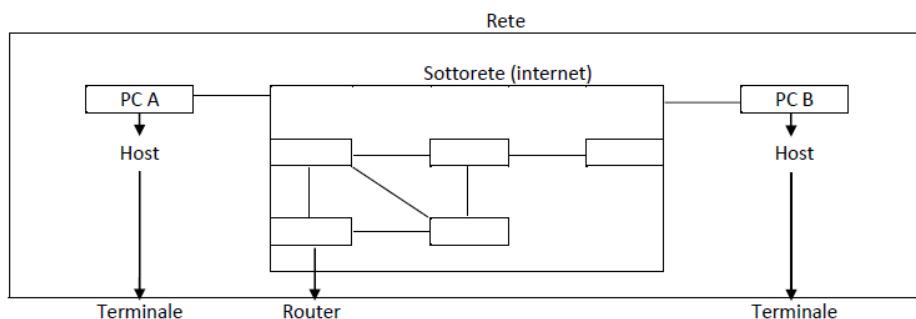
Schema:

Come gestiamo l'Affidabilità?

- Non vogliamo errori
- Non volgiamo duplicazione di pacchetti
- Pacchetti in ordine

Abbiamo preferito sottorete NON affidabile

- se no obbligherei ogni coppia di nodi a svolgere attività di controllo
- meglio lasciarli agli host attraverso protocollo TCP



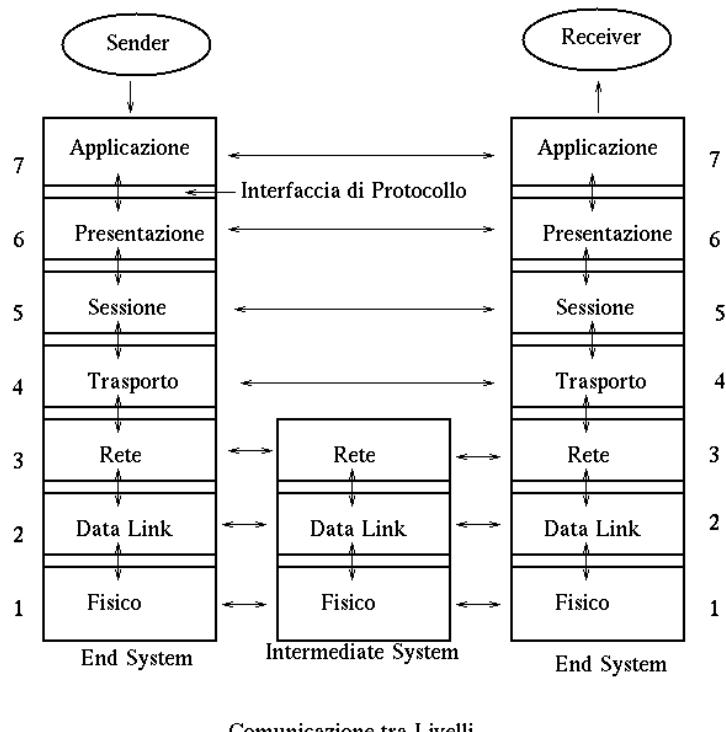
Consideriamo anche che se la sottorete fosse affidabile, il TCP farebbe cose ridondanti (overhead).

Ma allora se considerassimo la sottorete affidabile, esistono alternative per il livello transport?

Sì, il protocollo UDP trasporta senza affidabilità a livello 4 ed è molto più semplice di TCP (ogni macchina possiede sia TCP che UDP). È interessante notare che se la sottorete non è affidabile e si vuol comunque usare UDP come per esempio per applicazioni di interrogazione a database (query), sarà il livello 7 a occuparsi dell'affidabilità.

In questo modo non avremo alcuna gestione dell'affidabilità a livello di rete e sottorete ma dato che le informazioni da trasferire sono modeste, si accetta il compromesso ottenendo buoni risultati prestazionali.

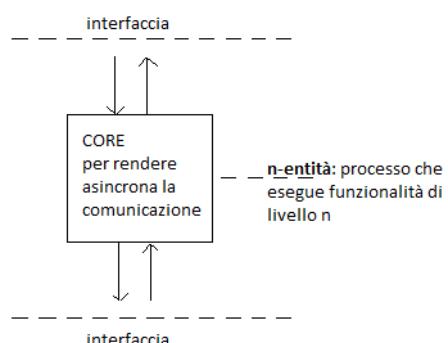
Modello architetturale ISO/OSI



OSI (*Open System Interconnection*), definisce l'architettura funzionale della macchina.

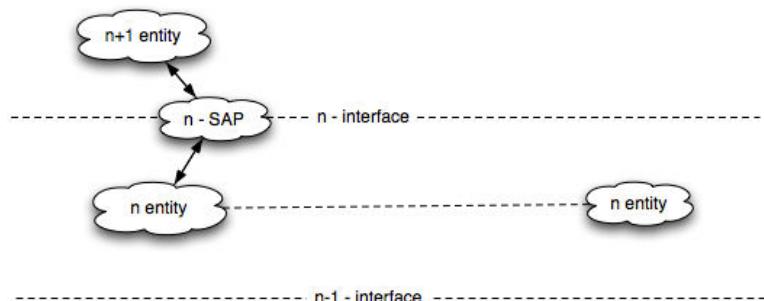
I livelli descritti sono organizzati come uno stack, e ogni livello ha due interfacce: una con il livello soprastante e una con il sottostante. Per ogni livello funzionale abbiamo uno o più protocolli.

I protocolli si interfacciano con i livelli superiori o inferiori tramite code, con tutti i problemi di concorrenza ad esse legati.



SAP = Service Access Point

Entità = esecuzione funzionale di livello, entità dello stesso livello comunicano, tramite protocollo, solo con altre entità dello stesso livello.



Dai livelli più alti e scendendo, ogni unità funzionale aggiunge un Header e un Trailer (come una specie di scatola cinese).

Ogni livello non entra in merito al contenuto inviato dal livello superiore.

Lato destinazione invece si riceve l'informazione dal livello più basso e risalendo i livelli superiori, avviene uno "sbustamento": vengono eliminati gli H e T.

In modo del tutto trasparente all'architettura, possiamo astrarre la comunicazione come se ciascun livello parlasse con il livello *peer* (livello paritetico, o livello dell'host/nodo ricevente).

In tutto ci sono 7 livelli e possiamo separarli in base all'appartenenza tra nodo e host.

Livelli gestiti dai nodi della sottorete

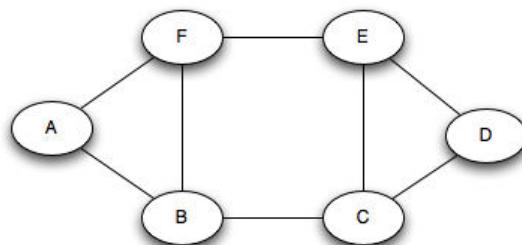
I livelli gestiti dalla sottorete sono:

- 1- Fisico
- 2- DataLink
- 3- Network

1. Fisico

Livello più basso, si occupa del canale fisico e del trasferimento in modo seriale.

È quindi sia un trasmettitore che un ricevitore e la trasmissione è regolata da un segnale di clock proprio del mezzo fisico.



In questa rete a maglia, il nodo A ha 2 schede (una per ogni collegamento agli altri nodi), mentre il nodo B ne ha 3, etc etc... Le funzionalità di primo livello sono **identiche** per tutti i nodi della stessa rete, la loro implementazione invece varia a seconda della tecnologia utilizzata per la trasmissione. Il livello fisico è molto "povero": trasmette bit per bit le informazioni che gli vengono passate dal livello superiore e non è assolutamente in grado di preparare i pacchetti.

2. DataLink

Gestisce frame (anch'essi come i pacchetti del livello superiore, sono dotati di header, payload e tail).

Viene data una formattazione corretta ai dati da trasmettere e viene fornita una struttura ai dati per la comunicazione in uno e un solo link (nell'esempio appena visto del livello fisico il nodo A ha 2 data-link).

Si occupa di trasferire i frame da un capo all'altro della connessione (connessione punto-punto del livello fisico) e se, è una rete a circuito virtuale, deve garantire affidabilità in quella tratta (esegue tutti i controlli in quella tratta).

Se il servizio della sottorete è non affidabile, datalink allora prende il pacchetto, lo bufferizza e lo passa al livello superiore (senza effettuare alcun controllo). Ancora oggi, si ha in ogni tipo di rete esistente la possibilità di scegliere una o l'altra soluzione.

Osservazione: il numero di link in entrata o in uscita in uno stesso nodo equivale al numero di livelli fisici che sono montati su di esso, ed è anche equivalente al numero di link in quel nodo.

Il livello link passa un *frame* al livello fisico come una *stringa di bit*.

L'unico obiettivo di questo livello è quello di trasmettere un frame tra due nodi.

3. Network

Produce i pacchetti che poi vengono incapsulati nel livello DataLink

Si occupa di routing e dell'indirizzamento. È l'unico livello che ha una visione globale della sottorete.

È diviso in due sottolivelli (3A e 3B):

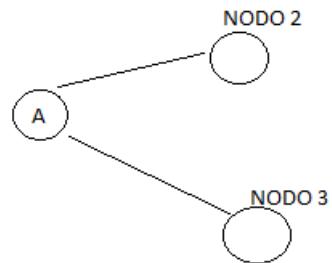
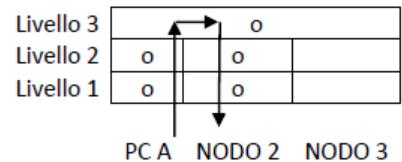
- sottolivello 3A è quello che ha la visione globale della sottorete ed effettua l'instradamento intranet.
- sottolivello 3B è quello che ha la visione globale della rete ed effettua l'instradamento internet

Livello 3B	Network	HW+SW
Livello 3A		
Livello 2	Datalink	HW
Livello 1	Physic	HW

"Livello ROUTER": potremmo così rinominare il livello network in quanto è l'unico che conosce tutti i nodi e la tipologia della sottorete (il livello link conosce solo la destinazione a cui il proprio link fisico è connesso).

Esempio di rappresentazione di un nodo

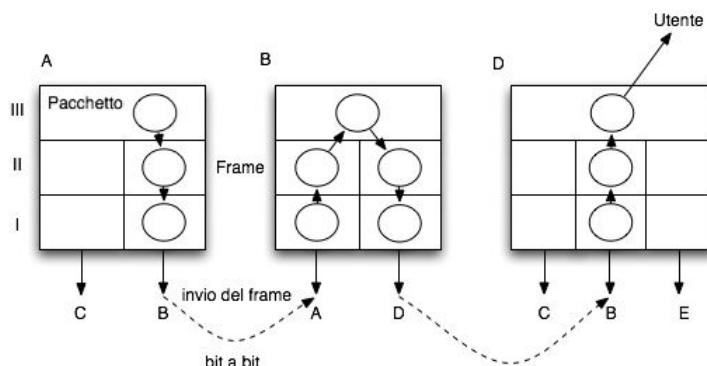
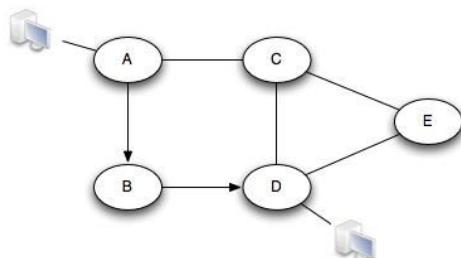
Il nodo A in questo caso è collegato a un PC e ad altri due nodi:



Ogni nodo della rete ha tanti canali fisici (e interfacce di rete) quanti ne servono, questo perché non è detto che i canali a cui è collegato siano della stessa tecnologia.

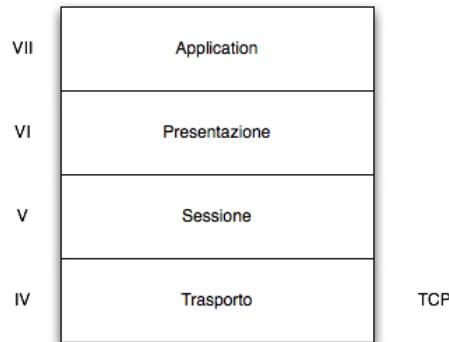
Esempio di instradamento di un nodo

Devo spedire un pacchetto dall'host in A all'host in B. Il router di A ha scelto la strada fino a D, passando per B.



Livelli gestiti dagli host di rete

I protocolli lato host parlano solo con l'interlocutore finale, non ci sono intermediari (e si dice comunicazione "end-to-end", a differenza di ciò che accade per i protocolli della sottorete). I livelli gestiti dagli host sono:



4. Transport

È il primo livello funzionale lato host, che interagisce con i livelli sottostanti.

Siccome il livello Network instaura una rete best-effort, è in questo livello che, se necessaria, si garantisce affidabilità.

TCP è il protocollo che garantisce l'affidabilità a livello 4.

Se si dispone di una sottorete a circuito virtuale ha ancora senso utilizzare il TCP?

No, perché la correttezza è gestita dalla sottorete. Con una rete a circuito virtuale conviene usare una connessione UDP a livello 4, che garantisce solo il trasporto e non la consegna.

Inoltre è preferibile usare UDP piuttosto che TCP anche se la sottorete è datagram qualora le unità dati siano piccole. UDP infatti, evitando di controllare la correttezza, è molto più veloce a trasmettere ed il controllo sulla correttezza dei dati verrà demandato a livello applicazione. In caso di errore, si provvederà a ritrasmettere il pacchetto, ma ciò non influirà troppo sulle prestazioni della rete, in quanto UDP è appunto molto veloce.

Riepilogando: la scelta del protocollo migliore tra TCP e UDP dipende da:

- Affidabilità della sottorete (datagram o CV)
- Il tipo di applicazioni che usano la rete

5. Session (DEPRECATO)

Originariamente era l'apertura di una sessione tra due utenti, ormai tutto è risolto con le socket

6. Presentation (DEPRECATO)

Al tempo in cui ogni costruttore aveva una propria architettura di rete, ogni macchina di diversi produttori aveva formati proprietari diversi (file system diversi, ecc...). Era quindi impossibile la comunicazione tra due macchine di due ditte diverse. Per risolvere questo problema il livello di presentazione convertiva le informazioni da spedire in un formato comune.

Oggi il problema è risolto in maniera diversa portando all'utilizzo del livello di presentazione:

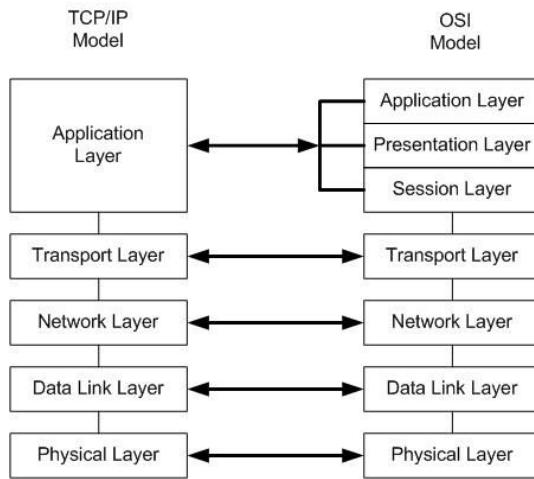
- Sono stati adottati gli standard comuni
- Le applicazioni stesse si preoccupano di convertire il loro formato particolare in un formato standard (per esempio: stampanti ed editor di testo → postscript)

7. Application

Protocolli che forniscono supporto di comunicazione all'utente finale. Si hanno tanti protocolli in base a quali sono le applicazioni (es. mail, web, ftp, dns, telnet..). Questo livello non descrive le applicazioni vere e proprie ma i loro protocolli di comunicazione, potremmo immaginare per questo un livello 8 dove risiede l'utente.

Visto che i livelli 5 e 6 sono deprecati, come riferimento viene usato lo stack dei protocolli TCP/IP di 5 livelli.

È lo standard de facto in contrapposizione allo standard de iure rappresentato invece dal modello ISO/OSI. Il modello Internet è stato prodotto come una soluzione ad un problema ingegneristico pratico in quanto si è trattato di aggiungere via strati protocollari all'architettura di rete delle reti locali per ottenere un'interconnessione efficiente ed affidabile. Il modello OSI, in un altro senso, invece è stato l'approccio più teorico-deduttivo ed è stato anche prodotto nel più vecchio modello di rete.

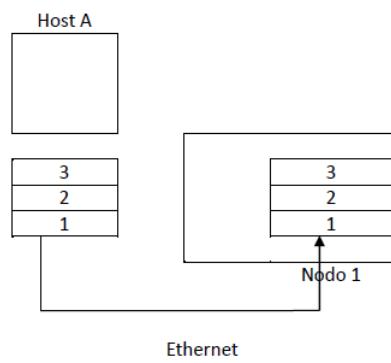


Funzionalità a confronto

Ogni host è composto dai quattro livelli funzionali appena esaminati più i tre livelli inferiori (quelli visti precedentemente per i nodi della sottorete). Questi ultimi 3 livelli saranno più semplici rispetto a quelli di un nodo di commutazione. Per esempio, il livello di rete dell'host è molto più limitato e semplificato in quanto deve solo indirizzare la strada verso il nodo più vicino (non deve calcolare cammini complessi, ma solo diretti). C'è più funzionalità di indirizzamento che instradamento.

Quindi, se la rete è Ethernet (cioè vale a dire broadcast), il livello 3 di un host è "quasi" vuoto, nel senso che il livello 3 si occuperà principalmente di indirizzamento. Il protocollo utilizzato dal link è comunque IP. Ogni apparato di rete avrà uno e un solo IP per tutta la sua sessione in rete.

Schema: In un host di una rete Ethernet vi sono i 3 livelli di sottorete. Il livello 3A però non esiste (che era quello che ha la visione globale della sottorete ed effettua l'instradamento intranet)

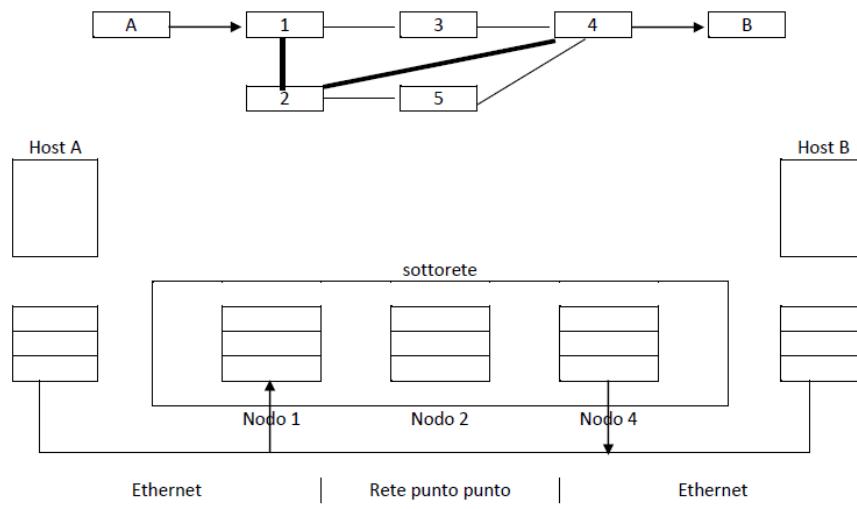


Esempio di interfacciamento tra parte host e nodo

Tornando a una normale sottorete... Come attacciamo un host al proprio nodo della sottorete?

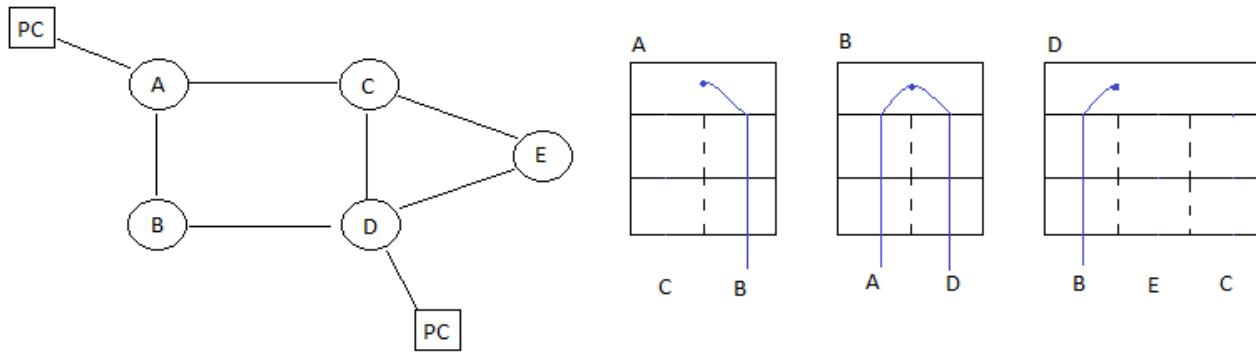
Non è un collegamento con un filo, a bordo del mio host dovrò avere i 3 livelli funzionali per la gestione della sottorete.

Ovviamente gli host non posseggono le funzionalità dei router, ma si interfacciano ai nodi i quali si occuperanno dell'instradamento necessario per far arrivare il messaggio a destinazione.



Gli host svolgono tutte e 7 le funzionalità previste, mentre i nodi solo le prime 3.

Esempio di funzionalità di instradamento di un nodo



Cammino: A → B → D , deciso dal livello 3

1. Nodo A: livello 3 produce un pacchetto (con formato che è capito da tutti, regola del protocollo)
2. Il pacchetto viene passato al livello 2 (quello che va verso B)
3. Nodo A: Data-Link produce frame che vengono quindi passati al livello fisico
4. Nodo A: Livello fisico mette nel link bit per bit
5. Nodo B: Livello fisico riceve una stringa di bit. Ciascun bit viene passato al livello DataLink
6. Nodo B: Livello DataLink riconosce la frame, la processa ed estrae il pacchetto IP togliendo H e T
7. Nodo B: Livello Network riceve pacchetto IP e saprà determinare la strada di inoltro
8. ..
9. Quando il frame raggiunge D, questi riconoscerà di essere il destinatario e invierà alla parte host il pacchetto.

Osservazioni:

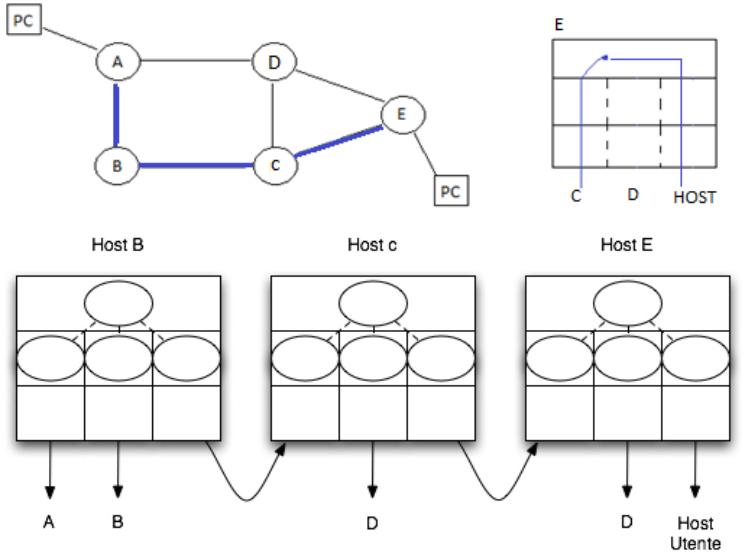
- Il livello datalink conosce solo il nodo adiacente.
- Il livello network si prende cura della topologia dell'intera rete, il protocollo IP garantisce che ogni apparato Internet sia identificato univocamente da uno e un solo indirizzo.
- Il livello fisico non ha percezioni, si limita a mandare bit.
- Se la sottorete fosse affidabile, ci sarebbero continui "rimpalli" di informazioni tra i nodi per i controlli.
- A livello 2 si trasmettono sempre FRAME
- A livello 3 si trasmettono sempre PACCHETTI
- A livello 4 si trasmettono sempre SEGMENTI
- I livelli principali (effettivamente implementati/utilizzati), che studieremo, sono i livelli 1,2,3,4,7.

Livello DataLink

Abbiamo già detto che ogni nodo collegato ad un host (utente) ha un link anche per la macchina host.

Esempio

E ha tre link: verso C, D e l'utente. Mettiamo il caso che il pc collegato al nodo A voglia comunicare al pc collegato al nodo E.



Come tutti i livelli, il livello 2 della sorgente parla con il livello 2 del destinatario, in questo caso trasferendo frame.

Vediamo però in dettaglio che cosa succede:

Il livello 3 del nodo A conosce l'intera topologia della rete, quindi passerà il pacchetto al livello data link del nodo giusto.
Il livello 3 del nodo A è anche in grado di capire come inoltrare sul cammino corretto e se il pacchetto che gli è arrivato è giunto a destinazione.

Il livello 2 invece, lavora sempre su una coppia di nodi per volta, quindi su un solo link.

Il livello 2 ha un protocollo punto-punto e non ho modo di sbagliare la destinazione!

Questo spiega perché nel frame non abbiamo i riferimenti del sorgente e del destinatario.

Una cosa importante da capire è quando un frame inizia e quando finisce. Allora l'Header ed il Tail saranno delle parole convenzionali. Quando il livello fisico del nodo A sorgente riconoscerà tali sequenze, sarà in grado di riconoscere il frame e di mandarlo interamente al livello 2 del prossimo nodo. Analogamente il destinatario sarà in grado di ricomporre il frame e inoltrarlo al suo livello superiore.

Frame in dettaglio

Un frame è una struttura dati con un header, un payload e una coda. Vediamoli in dettaglio.



Frame Header

Bisogna mettere solo lo stretto necessario per far funzionare il livello 2, perché ogni bit inutile è un bit sprecato sulla rete e ne diminuisce l'efficienza. Dobbiamo cioè inserire il necessario per far capire al destinatario che sta iniziando un frame.

Per far capire al livello 1 quando inizia e quando finisce il frame da spedire, occorre usare una keyword. Le parole chiave sono STX e ETX (Start_Of_Text e End_Of_Text).

All'inizio di ogni frame inseriremo come header il carattere ASCII *STX*.

Frame Tail

Dovrà contenere una o più parole che delimitano la fine del frame. Alla fine di ogni frame viene inserito come tail il carattere ASCII *ETX*. Il frame sarà quindi così caratterizzato:

STX _____ dati _____ ETX

Character Stuffing

Notiamo però che *STX* e *ETX* possono essere presenti anche nel campo data. Come capiamo allora che non sono header o tail ma bit nel campo data?

La prima soluzione trovata, che veniva svolta a livello fisico, eseguiva una procedura di **stuffing**:

- Frame Header diventa *DLE STX*
- Frame Tail diventa *DLE ETX*
- Tutti i caratteri *DLE* all'interno del payload vengono raddoppiati ottenendo la sequenza *DLE DLE*

Nel trasmettitore, mentre viene copiato il messaggio nel buffer, se viene trovata una parola che corrisponde alla parola *DLE*, viene aggiunta un'altra parola *DLE*. Questa procedura è detta stuffing. Alla fine, l'ultimo *DLE* prima di *ETX* non viene duplicato (perché viene riconosciuto essere la fine del frame e non più il contenuto della parte dati).

Osservazione: tutto questo viene fatto a livello 1 (Hardware).

In ricezione bisogna eliminare i *DLE* duplicati. Ogni volta che viene letto un *DLE*, se il carattere successivo è *ETX*, allora significa che sono arrivato alla fine, altrimenti, se viene letto un altro *DLE*, viene eliminato il primo e si continua a leggere i dati.

Osservazione: nel campo dati, non esiste un carattere *DLE* da solo.

Esempi: il carattere '-' rappresenta bit dati

-----DLE -----	diventa	-----DLEDLE -----
-----DLEDLEETX -----	diventa	-----DLEDLEDLEETX -----
-----ETX -----	diventa	-----ETX -----

Esercizio

Se data link layer usa character stuffing e preambolo frame è *DLE STX*, mostrare quale sequenza di caratteri è trasmessa a partire dai dati A B DLE C DLE STX D STX E ETX F DLE ETX J H

SOL: <**DLE STX**> A B **DLE** DLE C **DLE** DLE STX D STX E ETX F **DLE** DLE ETX J H <**DLE ETX**>

Questa procedura è più veloce di un counter al livello 2 ed è più veloce di aggiungere nell'header del frame un campo length. Quando ci si è accorti però che questo stuffing aveva un altissimo overhead (nel peggior dei casi il dato potrebbe raddoppiare la sua dimensione), si è passati a uno stuffing di bit.

Bit Stuffing

Il campo dati viene delimitato da un flag:

Flag _____ Flag

- Frame Header e Frame Tail sono uguali a $F = 01111110$ (*il numero di 1 presenti in questo flag determina il tipo di stuffing*)
- Lo stuffing prevede quindi di aggiungere uno 0 se leggo 5 uni consecutivi nel campo data
- Non ho da aspettare (con character stuffing il trasmettitore prima di inviare un byte deve controllare anche il byte successivo)

Esempi:

F-----01111110 -----F diventa F-----011111010 -----F
F-----001111100 -----F diventa F-----0011111000 -----F

Esercizio

Se assumiamo che il carattere di flag sia 11111, come sarà trasformata la sequenza 010101011111011110 nel caso in cui si applichi il bit stuffing?

SOL Il bit stuffing aggiunge zeri o uni tutte le volte che la stringa di ingresso contiene il carattere di flag in modo da impedire che la sequenza di dati contenga il carattere di flag, quindi la stringa di uscita potrebbe essere cambiata in

010101011110110111100 aggiungendo uno zero dopo ogni sequenza di quattro uni.

Esercizio

Se data link layer usa bit stuffing e preambolo di frame è 01110, mostrare quale sequenza bit è trasmessa a partire dai dati 0 1 1 1 0 0 1 0 1 1 1 1 1

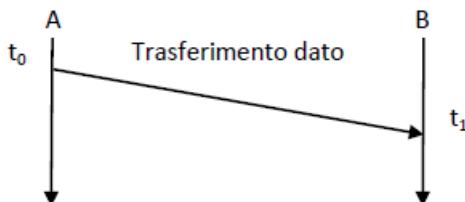
SOL: 01110 0 1 1 0 1 0 0 1 0 1 1 0 1 1 0 1 1 0 01110

Data Transfer a livello 2

Analizziamo ora il trasferimento dati a livello 2:

- connectionless = non affidabile (**datagram**)
- connection = affidabile (**circuito virtuale**)

Non ci soffermiamo sullo schema connectionless in quanto è semplice:



Affidabilità a livello 2

Sottorete a circuito virtuale

Deve assicurare non duplicazione dei pacchetti, ordine nella ricostruzione e integrità.

A livello data link ogni nodo ha visione solo delle sue adiacenze, ovvero dei nodi che sono direttamente collegati ad una delle sue porte di I/O. Anche se non è usato nella pratica, è possibile costruire dei protocolli che, già a livello 2, garantiscono l'affidabilità della comunicazione. Per ottenere questo risultato, tali protocolli fanno uso di un metodo noto come ARQ (*Automatic Repeat Request*), che prevede la ritrasmissione di un pacchetto, fino a che il destinatario non ne conferma la ricezione.

I protocolli che considereremo sono:

Idle RQ (*Stop-and-wait ARQ*)

Continuous RQ:

- Go-back-N ARQ
- Selective Repeat ARQ

Idle RQ

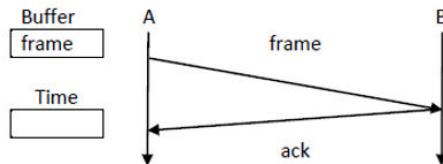
Idle-RQ è un semplice protocollo di livello 2, che regola la comunicazione tra due nodi adiacenti, garantendone l'affidabilità, attraverso il sistema Stop-And-Wait ARQ. Si osservi che tale protocollo ha una valenza unicamente didattica: abbiamo già detto che nelle reti odiere l'affidabilità delle comunicazioni è garantita a livello di host (livello 4).

Il problema che Idle-RQ si propone di risolvere è quello di consegnare un'unità dati (la frame) da A a B, dove A e B sono nodi in una topologia di rete magliata (reti punto-punto).

Il funzionamento di base del protocollo è illustrato nella seguente figura, dove la dimensione verticale rappresenta il tempo.

Il nodo A invia per prima cosa la sua frame a B; B, ricevuta la frame, segnala ad A di aver ricevuto correttamente i dati inviando un ACK (Acknowledgement). Si noti che ogni frame termina con un codice di validazione (CRC) in base al quale si determina la bontà dei dati ricevuti. Alla ricezione dell'ACK, A saprà che i suoi dati sono giunti correttamente a destinazione.

Il nodo sorgente ha un buffer dove tiene copia dell'ultimo frame inviato. Nel caso dovesse riinviare il frame ce l'ha già pronto. Quando poi riceverà l'ACK allora svuoterà il buffer.



Timer di ritrasmissione

Supponiamo ora che il frame inviato da A venga perso. Come conseguenza avremo che non tornerà indietro l'ACK ed il nodo A dovrà riinviare il frame. Ma come fa A ad accorgersi che il frame è stato perso?

Il mittente dovrà dimensionare un time-out in modo tale che, una volta scaduto, venga ritrasmesso il frame. Ma in che modo?

Sappiamo che:

$$RTT = t_{x,FRAME} + t_{p,FRAME} + t_{x,ACK} + t_{p,ACK} + t_{el}$$

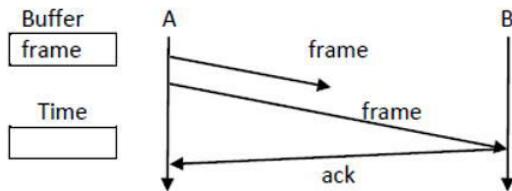
Supponendo che $t_{x,ACK}$ e t_{el} siano trascurabili..

$$RTT \approx t_{x,FRAME} + 2t_p$$

Visto che a livello 2 il cavo è corto e la varianza di RTT è bassa, posso dimensionare il time-out poco al di sopra di RTT. Quindi $t \geq RTT$.

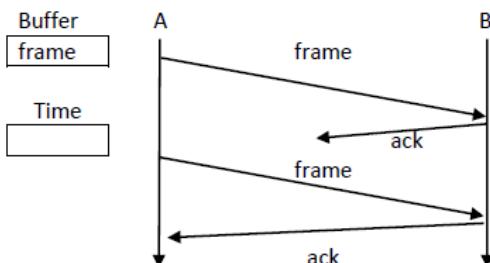
Vedremo quali complicazioni sorgeranno quando tratteremo questo argomento a livello 4.

Inoltre, ogni volta che si riceve l'ACK il timer dovrà essere riavviato, perché ciascun timer si riferirà alla trasmissione del frame corrente.



Numeri di sequenza

Se è il frame ACK a non arrivare al mittente?



Prendiamo il caso in esempio:

- 1- A manda frame
- 2- B lo riceve e manda ACK ad A
- 3- L'ACK però non arriva ad A
- 4- Scaduto il timeout, A rimanda lo stesso frame
- 5- Al destinatario B arriva nuovamente lo stesso frame
- 6- Il destinatario se non sa riconoscere che si tratta dello stesso frame già ricevuto lo accetterebbe.

Serve allora che i frame abbiano un numero che li identifichi

Il mittente, allo scadere del timer, rimanda il frame che crede che non sia mai arrivato al destinatario; sarà poi il destinatario a capire che è un duplicato e a dropparlo (punto 4 e 5). Ma in che modo avviene questo? Il mittente e il destinatario memorizzano una variabile V che, per il mittente, indica il numero di sequenza dell'ultimo pacchetto inviato, e per il destinatario indica il numero di sequenza del prossimo pacchetto che si aspetterà di ricevere. Viene quindi usata una variabile di trasmissione (VT) dal nodo trasmettitore e una variabile di ricezione (VR) dal nodo ricevitore:

- 1- In fase di setup le variabili vengono inizializzate a 0
- 2- A invia frame aggiungendogli un numero di sequenza uguale al valore VT
- 3- B riceve il frame e controlla se il numero di sequenza corrisponde al valore di VR
 - Se sì, elabora dato, incrementa VR e manda ACK al mittente
 - Se no scarta frame e non fa nulla*
- 4- A riceve l'ACK, svuota buffer, azzera timer e incrementa VT .

**Notare che, anche quando droppe, il destinatario, manda ACK per desincronizzare il mittente.*

Un campo nella struttura del frame sarà quindi dedicato a memorizzare il numero di sequenza. Si cercherà allora di avere una dimensione del numero di sequenza che sia la più piccola possibile.

Per il tipo idle-RQ, se il frame ha un numero di sequenza dispari, qualsiasi frame con numero pari ci garantisce che non sia lo stesso.

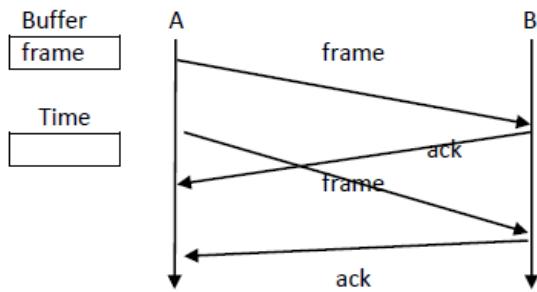
Possiamo dunque usare semplicemente 1 bit (valori 0 – 1) per rendere diversi 2 frame consecutivi.

Infatti a noi non interessa l'ordine numerico, ci interessa solo poter distinguere i frame in modo tale che il frame non sia ricevuto 2 volte di fila!

$$V_T \text{ e } V_R \in \{0,1\}$$

Se il timer di ritrasmissione del mittente è sbagliato?

Vediamo un esempio in cui il timer è più breve dell'RTT.



- 1- A manda frame 0
- 2- B riceve frame 0 e lo elabora perché $VR(0) == 0$
- 3- B cambia $VR(1)$ e invia ACK
- 4- Scade timeout quindi A riinvia frame 0
- 5- A riceve ACK con $VR(1)$
- 6- B riceve frame 0 (di nuovo)
- 7- B invia ACK con $VR(0)$
- 8- *I due nodi non sono più sincronizzati*

Quindi, anche se arriva l'ACK, ho già riinvia il frame.

Verranno inviati due pacchetti e quindi si riceveranno due ACK che faranno commutare due volte il mittente. Quindi il pacchetto duplicato viene scartato, e la sincronizzazione viene persa.

Il problema è che l'ACK del destinatario non specifica la classe corretta, cioè quella del pacchetto ricevuto correttamente.

Allora per ogni ACK ricevuto, *il mittente controlla che sia l'ACK della stessa classe del pacchetto precedentemente mandato e se arrivano due ACK della stessa classe il mittente ignora il secondo (ridondanza)*.

Conviene quindi mettere un campo di sequenza anche per l'ACK.

Piggybacking

Se ho frame da mandare al mittente posso unire frame e ACK.

Sarà necessario un timer anche a destinazione. In questo caso, il ricevente sta aspettando un frame che dovrà inviare al mittente e sta per inviare un ACK al frame ricevuto dal mittente. Se scade il timer lato destinazione, e il frame non è pronto, mando solo l'ACK.

Per fare piggybacking, il frame da trasmettere al mittente e l'ACK dovranno essere necessariamente pronti in un tempo minore del timer di attesa della ritrasmissione del mittente.

Questa possibilità di invio frame + ACK ci porta a considerare che ogni unità frame conterrà due numeri di sequenza distinti: uno per il frame e uno per l'ACK.

Il termine "piggybacking" significa "saltare in groppa".

Anche TCP adotta questa tecnica.

Considerazioni su RTT

Se il RTT è dominato dal tempo di trasmissione, la stazione trasmittente non può migliorare di molto la velocità totale.

Se invece domina il tempo di propagazione, ci troviamo in una situazione in cui la stazione trasmittente potrebbe inviare più di un frame prima che il primo arrivi a destinazione.

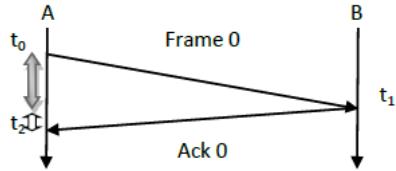
Esempio 1:

Lunghezza frame = 1000 bit
 Velocità canale = 1 Mb/s
 Lunghezza canale = 2 Km

$$t_x = \frac{1000 \text{ bit}}{1 \text{ Mb/s}} = 10^{-3} \text{ s} = 1 \text{ ms}$$

$$t_p = 10^{-5} \text{ s} = 10 \text{ us}$$

$$RTT = 1,02 \text{ ms}$$



In questo caso RTT è costituito per la maggior parte dal tempo di trasmissione. La rete è sfruttata bene e non ha lunghe pause di trasmissione.



RTT

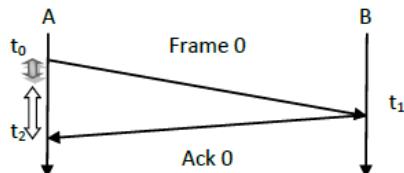
Esempio 2:

Lunghezza frame = 1000 bit
 Velocità canale = 1 Mb/s
 Lunghezza canale = 200 Km

$$t_x = \frac{1000 \text{ bit}}{1 \text{ Mb/s}} = 10^{-3} \text{ s} = 1 \text{ ms}$$

$$t_p = 10^{-3} \text{ s} = 1 \text{ ms}$$

$$RTT = 3 \text{ ms}$$



Ho un tempo di propagazione elevato, in cui il mio sistema non sfrutta le risorse. In quel tempo (pari a 2/3 di RTT) potrei inviare altri frame (altri 2 in questo caso).



RTT

1/3 del tempo viene impiegato per trasmettere il frame

2/3 del tempo il mittente resta ad aspettare l'ACK senza fare niente, POSSO OTTIMIZZARE!

Utilizzo del canale

$$U = \frac{t_x}{RTT} = \frac{t_x}{t_x + 2t_p}$$

U è l'utilizzo totale della rete: più il valore è alto, meglio viene usato il canale.

Con distanze molto alte e con valori di Tp alti, si rischia di penalizzare U (c'è una forte dipendenza con la lunghezza del canale fisico), ma in generale, bisogna considerare i parametri Tx e Tp.

Caso 1:

U ≈ 100 %

Caso 2:

U = 1/3 ≈ 33%

Per sfruttare meglio la rete nel caso 2 dovrei trasmettere più frame durante RTT del primo frame inviato

- quanti?

$$U = K * \frac{t_x}{t_x + 2t_p}$$

se voglio uno sfruttamento U del 100%

$$k = \frac{t_x + 2t_p}{t_x}$$

in questo caso si parlerà di protocollo a finestra di dimensione k

Quindi introduciamo una serie di protocolli che cercano di ottimizzare l'utilizzo della rete mandando una finestra di frame sul canale in base al rapporto tra Tx e Tp.

I protocolli a finestra del Circuito virtuale Data Link

Definizione: la finestra di trasmissione è il numero di frame che il protocollo riesce a spedire prima di ricevere un acknowledge.

La dimensione della finestra rappresenta il fattore moltiplicativo K tale che K·U=1, cioè il numero di frame che riesco a mandare sfruttando il tempo di propagazione all'interno del RTT.

Esempio:

se U = 1/3 → ottimizzo mandando 3 Frame durante il RTT

se U = 20% → ottimizzo mandando 4 Frame durante il RTT

Il buffer del mittente (e/o del destinatario) può essere dimensionato in base alla dimensione della finestra.

L'RTT dei frame possono essere contemporanei: quando un frame è inviato posso subito iniziare l'invio di un altro frame in modo che ci siano sempre al più k frame in trasmissione.

Allora ho bisogno di:

- k buffer nel trasmettitore (e in alcuni casi anche nel ricevente)
- k timer
- VT e VR non sono più variabili binarie (0,1).

Finché non perdo niente, tutto va bene e migliori le prestazioni. Ma quando perdo un pacchetto?

Una ottimizzazione è il segnale di NACK che serve ad anticipare il mittente per avvertirlo che un frame:

- è arrivato ma non è corretto. Il vantaggio è che non devo aspettare la scadenza del timer per far riinviare il frame dal mittente.
- oppure, il frame viene perso mentre gli altri arrivano (il ricevitore si accorge che i frame non sono in ordine)

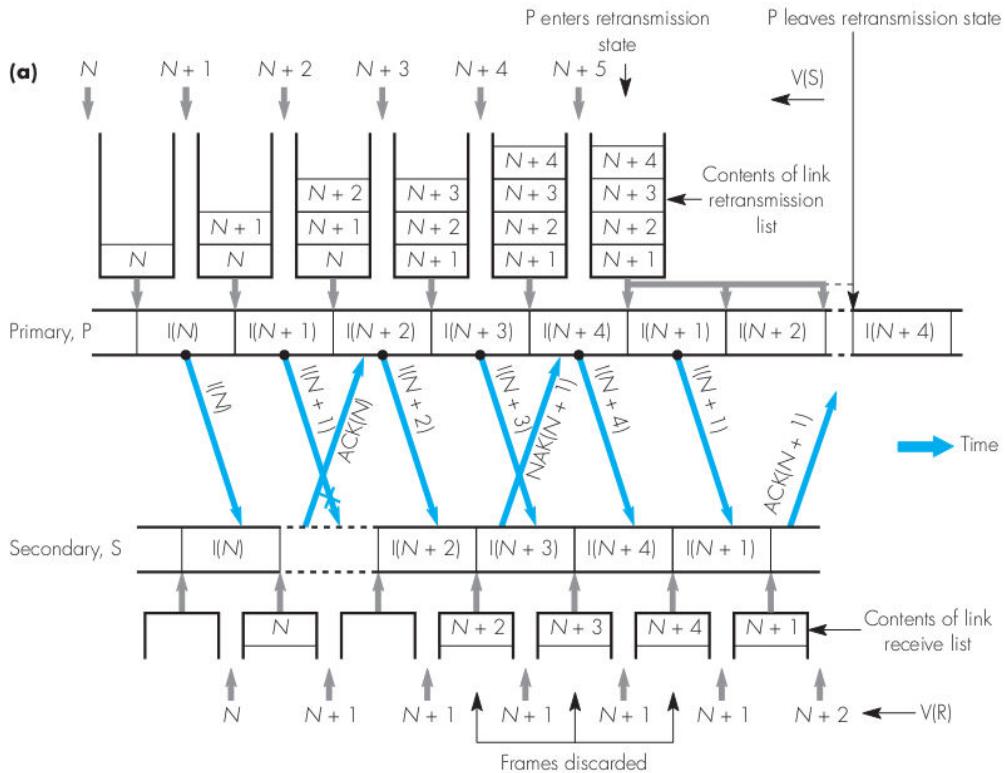
Il tipo di trasmissione visto in precedenza (Idle-RQ) con 2 soli numeri di sequenza (0,1) ha dimensione della finestra k=1.

Il trasmettitore però può agire secondo 2 algoritmi differenti:

- Go back n
- Selective Repeat

Go-back-N: Retransmission strategy

Data una sequenza di frame $N, N+1, N+2, N+3, N+4, \dots, N+k$, se in tale sequenza si perde o arriva un frame corrotto a destinazione, occorre ritrasmettere l'intera sequenza a partire dal frame corrotto. Per segnalare la perdita/corruzione di un frame, il ricevitore trasmette un segnale NAK (negative o non ack): è un segnale che rileva un fallimento prima dello scadere del timer del sender. Tale politica è più efficiente dell'uso del timer, in quanto si rivela prima il fallimento.



Il messaggio NAK che il Secondario invia al Primario lo avvisa che un pacchetto è andato perso. Allora P dovrà rimandare tutti i pacchetti ripartendo dal primo perso (go back n). Questo per garantire la corretta sequenza dei pacchetti in arrivo, perdendo però un sacco di tempo in ritrasmissione. C'è da dire però che così il ricevitore non deve allocare memoria se non per il pacchetto appena arrivato che è da mandare al livello superiore.

In genere si utilizza quando ho un ricevitore con poca memoria, quando ho finestre piccole oppure quando ho finestre grandi ma con un tasso di errore piccolo.

Osservazione: la dimensione della finestra determina i numeri di sequenza che mi occorrono.

In Go Back N, la finestra lato trasmissione ha dimensione k , lato ricezione 1.

Quindi se indichiamo con K la dimensione della finestra lato trasmissione, diremo che $K = 2^n - 1$

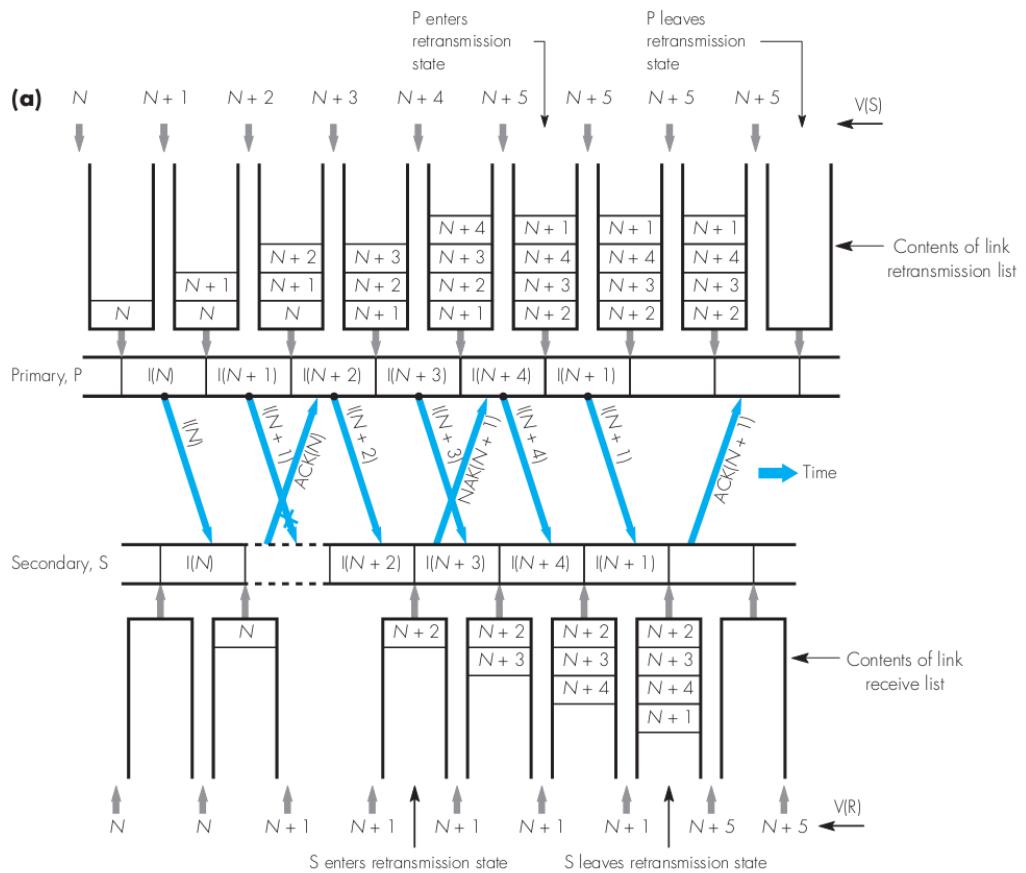
Mentre per indicare quanti numeri di sequenza saranno necessari, avremo: $\text{MAX_SEQ} = 2^n$

Possibili miglioramenti

Mettiamo un buffer dimensionato come la finestra del trasmittente anche al ricevitore, in modo tale che si possano tenere anche i pacchetti arrivati dopo uno perso. Quando arriva il pacchetto perso scarico il buffer seguendo la sequenza corretta. Parliamo di Selective Repeat.

Selective Repeat

Sia trasmittitore che ricevitore hanno un buffer grande K, mentre prima (Go-Back-N) il ricevitore aveva un buffer a cella singola.



Il trasmittitore riinvia solo il frame che non è arrivato (solo N+1). I frame successivi a quello perso, che sono stati correttamente ricevuti, vengono conservati nel buffer. E' come Go Back N, solo che non si ritrasmettono tutti i frame a partire dall'ultimo corretto, ma solo il frame corrotto, identificato dal segnale di NAK. Quando entrano in "retransmission state", P ed S smettono di mandare rispettivamente frame ACK.

Note: vedere se in retransmission state il protocollo funziona correttamente continuando a inviare le altre frame.

Quando un frame perso e ritrasmesso arriva al ricevitore, si hanno due strategie per validare il frame:

- ACK selettivo

Convalido usando un ACK per il singolo frame, sia quello del frame ritrasmesso che quelli dei pacchetti arrivati prima del NAK.

- ACK cumulativo

Aspetto a mandare tutti gli ACK successivi al frame perso. Quando viene ricevuto il frame mancante ed è corretto, mando un ACK dell'ultimo frame buono presente nel buffer (l'ultimo che nel frattempo è stato ricevuto correttamente), validando così anche tutti i frame precedenti (spedizione del solo ACK di N+4, con la semantica "ricevuti correttamente tutti i frame fino a N+4"). Con ACK cumulativo risparmio frame di controllo.

Retransmission mode

Un nodo trasmittitore entra in *retransmission mode* quando riceve un NAK.

Un nodo ricevente entra in *retransmission mode* quando invia un NAK, cioè quando si accorge che è arrivato un pacchetto fuori sequenza (come abbiamo visto in selective repeat il nodo continua comunque a ricevere, mantenendo nel buffer i pacchetti successivi a quello corrotto).

Per ripartire ho tre possibilità:

1. **Ack selettivo:** quando viene ricevuto un pacchetto corrotto, il nodo ricevente entra in retransmission mode. Viene scartato il pacchetto sbagliato ed inviato un NAK. I pacchetti successivi sono bufferizzati perché corretti. Quando viene ricevuto il pacchetto corretto che era stato scartato, per ogni altro pacchetto ricevuto in questo lasso di tempo, viene inviata una raffica di ack che li convalida tutti.

2. **Ack cumulativo:** un solo segnale di ack convalida automaticamente tutti i pacchetti ricevuti precedentemente

(3. Un'altra opzione è quella in cui il nodo ricevente non entra in retransmission mode e convalida tutti i pacchetti intermedi)

Attenzione: se il nodo trasmettitore non riceve un segnale di ack nei due casi precedentemente descritti?

1. ack cumulativi: il recovery non avrà problemi in quanto vengono convalidati tutti i pacchetti precedenti

2. ack selettivi: potrebbe essere un problema.

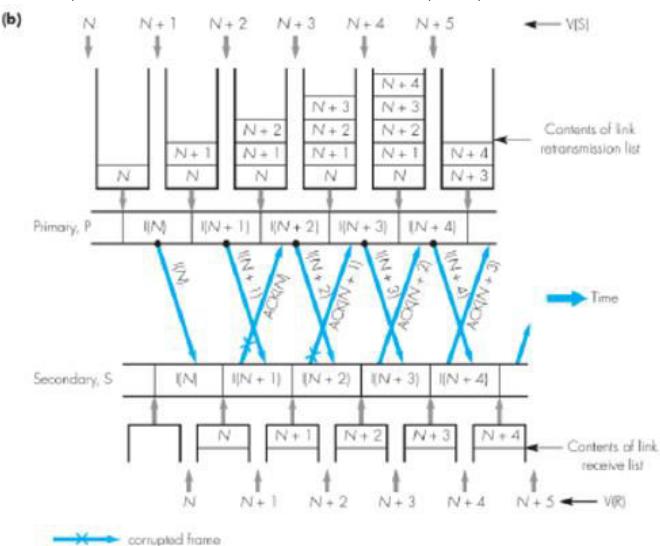
Vantaggi di ack selettivi e cumulativi:

-cumulativi: sono più economici, ma rischiano di far scattare il timer del trasmettitore

-selettivi: evitano di far scattare i timer

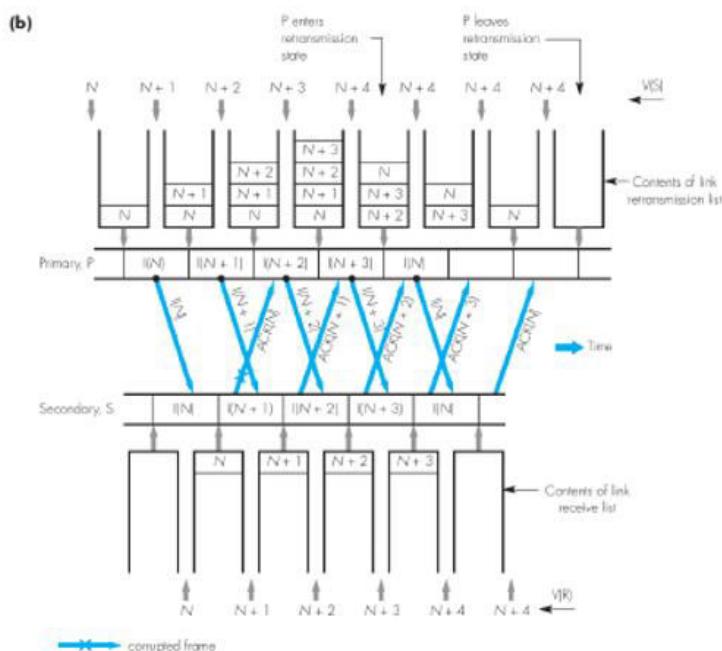
Esempio di perdita ACK – Go back n

Nessun problema, si ferma il meccanismo e poi riparte



Esempio di perdita ACK – Selective Repeat

Con ack cumulativo ho il vantaggio che mandando l'ultimo ack è come se mandassi tutti i precedenti insieme.



Numeri di sequenza (vedi p.58)

La dimensione della finestra determina la quantità dei numeri di sequenza di cui si ha bisogno.

Qualche esempio esplicativo:

- IDLE RQ, con finestra $K = 1$

Mi servono 2 numeri di sequenza, perché con un solo numero il ricevitore non saprebbe distinguere tra nuovo frame e frame rispedito (si comporta come un go-back-n con finestra 1)

Conclusione

La larghezza della finestra d'invio è fissa a $k = 1$ frame

$\text{MAX_SEQ} = K+1 = 2$ (servono due numeri di sequenza al massimo)

Possibili valori sono: 0,1

In un determinato momento della trasmissione il sender avrà come valore di sequenza uno 0 e il receiver un 1, o viceversa.
Sia sender che receiver possono memorizzare un solo valore di numero di sequenza alla volta.

- CONTINUOUS RQ, con finestra $K = 2^n - 1$

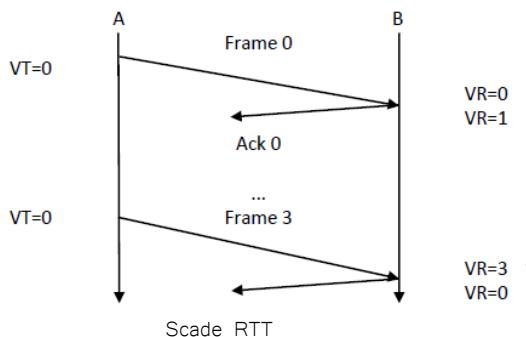
- Go-Back-N

Caso peggiore

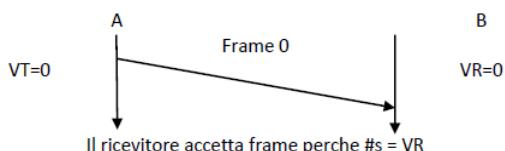
Se si perdono tutti e K gli ACK, mi basta un solo numero in più in modo tale da disaccoppiare la nuova finestra da quella vecchia

Esempio:

Vediamo cosa accadrebbe se $\text{MAX_SEQ} = K$

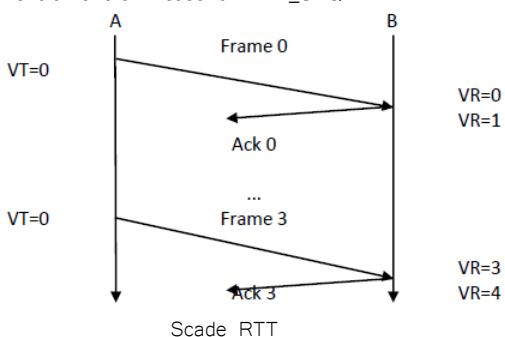


Il timeout scade e poiché non sono arrivati gli ACK, il trasmettitore riinvia i pacchetti. Il ricevitore (che non sa che i suoi ACK sono andati persi) vedrebbe ora i nuovi pacchetti come nuovi frame validi!



Esempio

Prendiamo ora il caso di $\text{MAX_SEQ} = K + 1$



Il timeout scade e il trasmettitore riinvia pacchetti partendo dal frame 0. Questa volta $#S \neq VR$, infatti è $VR(4)$ anziché $VR(0)$. Il ricevitore avrà la sequenza 0,1,2,3,0. A questo punto si interrompe il processo perché si accorge che manca il frame 4, di cui chiede la ritrasmissione.

Conclusione

Se la dimensione della finestra d'invio è $k = 2^n - 1 = (2^3)-1 = 7$ frame

Con $n=3$ bit riservati nel frame del trasmettitore per memorizzare il numero di sequenza d'invio

$$\text{MAX_SEQ} = K+1 = 2^n = 8$$

In totale servono otto numeri di sequenza al massimo;

Possibili numeri di sequenza che il sender può mantenere in memoria: 0,1,2,3,4,5,6,7

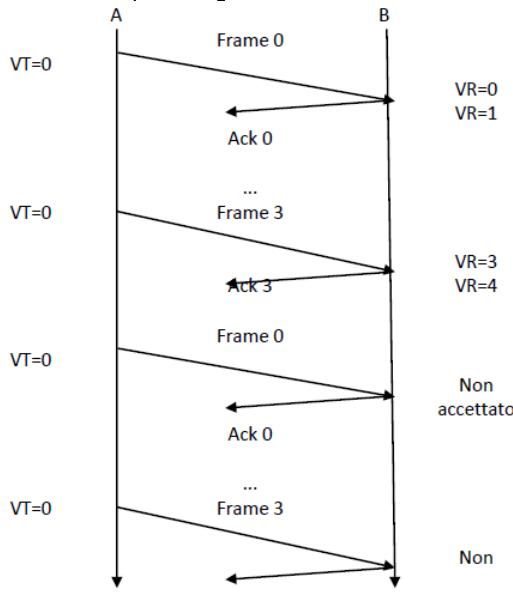
il ricevitore ha un solo bit per memorizzare un solo numero di seq. alla volta

- Selective Repeat, con finestra $K = (2^n)/2$

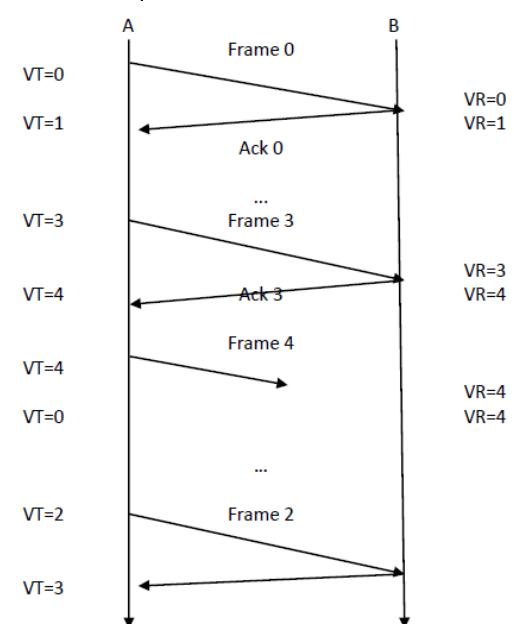
Caso peggiore: si perdono tutti e K gli ACK

La sequenza massima che mi può servire è $2K$, perché se si perdono tutti i pacchetti il ricevitore deve poter capire che i prossimi sono una ripetizione.

Caso 1 – si perdono gli ACK



Caso 2 – si perde un frame



Nel caso di perdita di un frame, il ricevente fa in tempo a ricevere i successivi 0,1,2 frame e li bufferizza. A questo punto il ricevitore non può più sapere se si tratta dei pacchetti di una nuova finestra o dei vecchi pacchetti ritrasmessi dal mittente.

Conclusione

La dimensione della finestra di trasmissione è $k = (2^n)/2 = 8$ frame

Con $n=4$ bit riservati nel frame del trasmettitore per memorizzare il numero di sequenza d'invio;

il ricevitore ha anche lui 4 bit per memorizzare il suo numero di seq

Possibili valori:

0,1,2,3, 4,5,6,7, lato sender

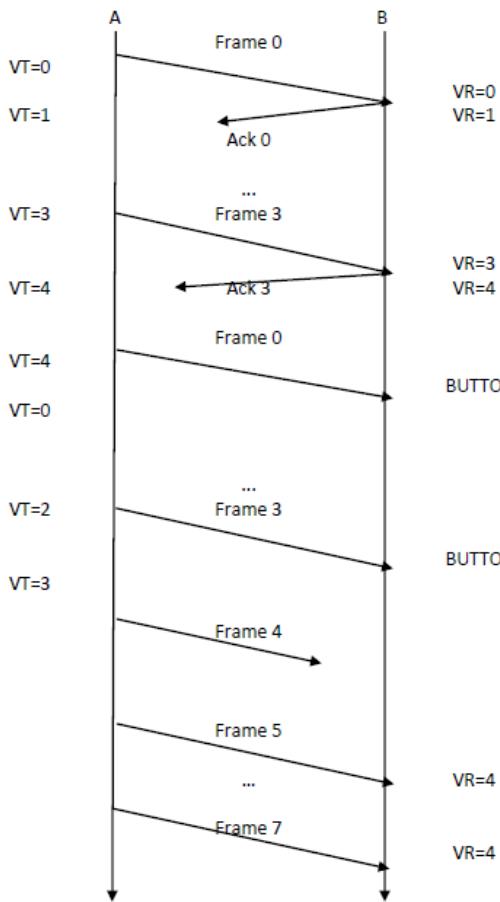
0,1,2,3, 4,5,6 lato receiver

$$\text{MAX_SEQ} = (2K) - 1 = (2^n) - 1 = 15 \text{ (servono in totale 15 numeri di sequenza al massimo)}$$

La grandezza del numero di sequenza implica un diverso numero di bit nel campo sequenza del pacchetto.

Segue un esempio per Selective Repeat con $\text{MAX_SEQ} = 16$

Il ricevitore riconosce i frame dal 5 al 7 come nuovi e quindi li bufferizza in attesa che sia rimandato il 4



Esercizio

Calcolare l'utilizzo di un canale avente banda di 6 Mbps e lunghezza del cavo in rame di 4 Km, ottenuto da un protocollo di livello Data Link che genera frame di taglia fissa 1.5 Kb e che adotta un approccio di tipo stop-and-wait (Idle RQ).

SOL:

Tempo di propagazione $T_p = \text{lunghezza cavo} / \text{velocità propagazione del segnale} = 4 \text{ Km} / 200000 \text{ Km/s} = 2 \times 10^{-5} \text{ s}$.

Tempo di trasmissione $T_x = \text{frame size} / \text{bwth} = 1.5 \text{ Kb} / 6000 \text{ Kbps} = 2.5 \times 10^{-4} \text{ s}$.

Utilizzo per Idle RQ è $U = T_x / (T_x + 2T_p) = (2.5 \times 10^{-4}) / (2.5 \times 10^{-4} + 2 \times 2 \times 10^{-5}) = 86.21\%$

Esercizio

Un blocco di dati da 4 Kb deve essere trasmesso tra due calcolatori connessi da un canale di comunicazione in fibra lungo 50 Km e avente banda di 250 Mbps. Calcolare l'utilizzo del canale se il livello data-link usa un protocollo Selective-Repeat con numeri di sequenza rappresentati con 4 bit.

SOL:

$T_p = 50/300000 = 1.67 \times 10^{-4} \text{ s}$.

$T_x = 4/250000 = 1.6 \times 10^{-5} \text{ s}$.

Con 4 bit per numero sequenza, il MAX-SEQ è $(2^4)-1=15$ e la dimensione K della finestra di invio è $(2^4)/2=8$.

L'utilizzo è $U=K T_x / (T_x + 2 T_p) = 8 \times 1.6 \times 10^{-5} / (1.6 \times 10^{-5} + 2 \times 1.67 \times 10^{-4}) = 36.57\%$

Esercizio

Un blocco di dati da 3 Kb deve essere trasmesso tra due calcolatori connessi da un canale di comunicazione in rame lungo 70 Km e avente banda di 60 Mbps. Calcolare l'utilizzo del canale se il livello data-link usa un protocollo Go-Back-N con numeri di sequenza rappresentati con 3 bit.

SOL:

$T_p = 70 \text{ Km} / 200000 \text{ Km/s} = 70 \times 10^3 \text{ m} / 2 \times 10^5 \times 10^3 \text{ m/s} = 35 \times 10^{-5} \text{ s} = 3.5 \times 10^{-4} \text{ s}$

$T_x = 3 \text{ Kb} / 60000 \text{ Kbps} = 3 \times 10^3 \text{ b} / 6 \times 10^4 \times 10^3 \text{ bps} = 0.5 \times 10^{-4} \text{ s}$

Con 3 bit per numero di sequenza, il MAX-SEQ è $(2^3)=8$ e la dimensione K della finestra di invio è $(2^3)-1=7$.

$$\begin{aligned}
 \text{L'utilizzo è } U &= K \cdot T_x / (T_x + 2 \cdot T_p) \\
 &= 7 \times 0.5 \times 10^{-4} \text{ s} / (0.5 \times 10^{-4} \text{ s} + 2 \times 3.5 \times 10^{-4} \text{ s}) \\
 &= 3.5 \times 10^{-4} / (0.5 + 7) \times 10^{-4} \\
 &= 3.5 / 7.5 \\
 &= 0.4666 \\
 &= 46.66\%
 \end{aligned}$$

Meglio non fare i conti con i numeri con troppe cifre decimali, usare unità di misura più piccole!

Usare la stessa unità di misura per Tx e Tp.

Esercizio

Un blocco di dati da 3Mb deve essere trasmesso tra due calcolatori connessi da un canale di comunicazione in rame lungo 7 Km e avente banda 600 Kbps. Calcolare l'utilizzo del canale se il livello Data-Link usa un protocollo Go-Back-N con numeri di sequenza rappresentati con 3 bit.

$$\begin{aligned}
 K &= 2^3 - 1 = 7 \\
 T_x &= 3 \text{ Mb} / 600 \text{ kbps} = 3 \times 10^6 \text{ b} / 6 \times 10^2 \times 10^3 \text{ bps} = 30 \times 10^5 / 6 \times 10^5 = 5 \text{ s} \\
 T_p &= 7 \text{ km} / 200.000 \text{ km/s} = 7 \times 10^3 \text{ m} / 2 \times 10^5 \times 10^3 \text{ m/s} = 70 \times 10^2 \text{ m} / 2 \times 10^8 \text{ m/s} = 35 \times 10^{-6} \text{ s} \\
 U_{gbn} &= K [(T_x) / (T_x + 2T_p)] = \\
 &= 7 \times 5 / (5 + 2 \times 35 \times 10^{-6}) \\
 &= 35 / (5 + 70 \times 10^{-6}) \\
 &= 35 / 5.000070 \\
 &= 6.999 \\
 \text{Calcolando in percentuale} \\
 &= 6.999 \times 100 \\
 &=? \text{ sbagliati i dati}
 \end{aligned}$$

Esercizio

Un blocco di dati da 4Mb deve essere trasmesso tra due calcolatori connessi da un canale di comunicazione in rame lungo 5 Km e avente banda di 250Kbps. Calcolare l'utilizzo del canale se il livello Data-Link usa un protocollo Selective-Repeat con numeri di sequenza rappresentati con 4 bit.

$$\begin{aligned}
 \text{MaxSeq} &= 2^4 - 1 = 15 \\
 T_x &= 4 \text{ Mb} / 250 \text{ kbps} = 4 \times 10^6 \text{ b} / 250 \times 10^3 \text{ bps} = 40 \times 10^5 \text{ b} / 25 \times 10^4 \text{ bps} = 16 \text{ s} \\
 T_p &= 5 \text{ km} / 200.000 \text{ km/s} = 5 \times 10^3 \text{ m} / 2 \times 10^5 \times 10^3 \text{ m/s} = 2.5 \times 10^{-5} \text{ s} \\
 \text{dim finestra: } K &= 16/2=8 \\
 U &= K \cdot T_x / (T_x + 2 \cdot T_p) \\
 &= 8 \times 16 \text{ s} / (16 \text{ s} + 2 \times 2.5 \times 10^{-5} \text{ s}) \\
 &= 128 \text{ s} / (16 \text{ s} + 5 \times 10^{-5} \text{ s}) \\
 &= 128 / 16.00005 \\
 &= 7.999
 \end{aligned}$$

Esercizio

$$\begin{aligned}
 v &= 6 \text{ MB/s} \\
 \text{lunghezza } l &= 4 \text{ km} \\
 f &= 1.5 \text{ kb}
 \end{aligned}$$

trovare sfruttamento del canale U

$$t_p = \frac{l}{2 * 10^8} = \frac{4 * 10^3}{2 * 10^8} = 2 * 10^{-5}$$

$$t_x = \frac{f}{v} = \frac{1.5 * 10^3}{6 * 10^6} = 0.25 * 10^{-3}$$

$$U = \frac{2.5 * 10^{-4}}{2.5 * 10^{-4} + 2 * 2 * 10^{-5}} = \frac{2.5}{2.9} =$$

$$= 86\%$$

$$\begin{aligned}
 U &= 2.5 \times 10^{-4} / 2.5 \times 10^{-4} + 2 \times 2 \times 10^{-5} \\
 &= 2.5 \times 10^{-4} / 2.5 \times 10^{-4} + 4 \times 10^{-5} \\
 &= 2.5 \times 10^{-4} / 2.5 \times 10^{-4} + 0.4 \times 10^{-4} \\
 &= 2.5 / 2.9 \\
 &= 0.862
 \end{aligned}$$

= 86%

Esercizio

$$\begin{aligned}f &= 300 \text{ bit} \\l &= 70 \text{ km} \\v &= 60 \text{ Mb/s}\end{aligned}$$

Trovare sfruttamento rete U
con un protocollo di livello 2 **go back n** con numero di sequenza di 3 bit

$$U = K * \frac{t_x}{t_x + 2t_p}$$

Quanto vale K (cioè quanto è grande la finestra?)

Con protocollo **go back n** sappiamo che mi servono $k+1$ numeri di sequenza affinché funzioni!
Con 3 bit rappresento massimo 8 numeri di sequenza -> $k = 8-1 = 7$

$$t_x = \frac{3 * 10^3}{6 * 10^7} = 0.5 * 10^{-4}$$

$$t_p = \frac{7 * 10^4}{2 * 10^8} = 3.5 * 10^{-4}$$

$$U = 7 * \frac{0.5 * 10^{-4}}{0.5 * 10^{-4} + 2 * 3.5 * 10^{-4}} = 7 * \frac{0.5}{7.5} =$$

= 46%

Esercizio

Due stazioni comunicano attraverso una connessione su cavo coassiale di 4 Km alla velocità di 10 Mbps. I pacchetti sono di 500 bit. Determinare l'utilizzo del canale con finestra di trasmissione 2.

SOL $T_p = (4 * 10^3) / (2 * 10^8) = 2 * 10^{-5} = 20 \text{ us}$

$T_x = (5 * 10^2) / (10^7) = 5 * 10^{-5} = 50 \text{ us}$

$U = 2 * (50 / 50 + 2 * 20) = 2 * (50/90) = 1,1$

L'utilizzo del canale è di circa 1, quindi il sistema garantisce un utilizzo pari a circa il 100%

Esercizio

Qual è la massima window-size lato sender di un protocollo a finestra a livello datalink che opera su una rete con round trip delay (rtt) 100 msec, con link da 50 Kbps su cui si trasmettono pacchetti da 500 bit?

SOL Dai dati disponibili il tempo richiesto per trasmettere un pacchetto è di 10 ms

[Infatti: $T_x = f/bwth = 500 \text{ b} / 50 * 10^3 \text{ bps} = 10 * 10^{-3} \text{ s} = 10 \text{ ms}$]

Se una stazione sender trasmette al tempo $t=0$ riceverà il pacchetto ACK al tempo $t=110 \text{ ms}$. [$T_x + (rtt = 2 T_p)$]

Per meglio utilizzare il canale disponibile, il sender può trasmettere fino a 11 pacchetti [$110 / 10 = \# \text{pacchetti}$] dopo che i pacchetti ACK arriveranno più o meno regolarmente ogni 100 msec. [$T_x = 0 \text{ per gli ACK}$] consentendo al sender di trasmettere in base alle sue esigenze. La finestra massima lato sender è quindi 11.

[Dovrebbe essere: $Rtd=2T_p$, mentre: $rtt=T_x+2T_p$]

Esercizio

In una rete con max TPDU 128 B, max tempo di vita di una TPDU 30 sec e #seq di 8 bit, qual è il max bit rate per connessione?

SOL:

$$\# \text{seq} = 8 \text{ bit} \rightarrow K = (2^8) - 1 = 255;$$

perciò il sender non può inviare più di 255 segmenti in 30 sec.

Il max bit rate è $(255 * 128 * 8 \text{ bit}) / 30 = 8704 \text{ bps}$. [è massimo n° di bit al secondo che possono essere trasmessi]

Esercizio

Un carrier T1 lungo 3000 Km è usato per trasmettere frame di 64 B con protocollo Go-Back-N. Se la velocità di propagazione è 6 μ sec/Km, di quanti bit dovrebbe essere il #seq?

SOL: bwth T1 per dati è 1.288 Mbps.

$$Tx = (64 \cdot 8) / 1.288 \text{ Mbps} = 0.397 \text{ ms.}$$

Sia k la taglia della finestra, U tende a 1 se $k \cdot Tx / (Tx + 2 \cdot Tp) = 1$,

$$\text{Quindi: } k = (Tx + 2 \cdot Tp) / Tx$$

$$k = (0.397 + 2 \cdot 0.006 \cdot 3000 \text{ Km}) / 0.397$$

$$k = (0.397 + 36) / 0.397 = 36.397 / 0.397 = 91.68 \text{ frame.}$$

$$\text{Quindi i #seq sono di 7 bit: } (2^7) - 1 = 127.$$

Usando 7 bit mi assicuro di avere dei numeri di sequenza a sufficienza per 91 frame.

Nb. 6 μ sec/Km = $6 \cdot 10^{-6} / 10^{-3} = 6 \cdot 10^{-3} = 0.006 \text{ s.}$

$$Tp = d / v = (3 \cdot 10^6 \text{ m}) / (2 \cdot 10^8 \text{ m/s}) = 1.5 \cdot 10^{-2} = 0.015 \text{ s NO!}$$

In questo esercizio Tp viene calcolato in modo molto semplice: cioè per ogni Km viene impiegato un tempo 0.006 s. Quindi Tp = 0.006 s • 3000 Km = 18 s

bwth T1 per dati è 1.288 Mbps

T1 ha bwth 1.544 Mbps, compresi bit di controllo

Carrier T1 invia un frame ogni 125 μ sec

Esercizio

Qual è il ritardo di trasmissione di 3 Mb di dati se viene utilizzato il carrier T1?

SOL: T1 ha bwth 1.544 Mbps, compresi bit di controllo. Il frame è costituito da 1 bit di sincronizzazione, e da 24 canali multiplexed, per ognuno dei quali vi sono a disposizione 8 bit. 1 bit di ogni canale è riservato per controllo e sincronizzazione tra gli apparati. Se usato per il solo invio dati, il 24esimo canale è interamente usato per sincronizzazione; perciò i bit di controllo sono in totale $(1 + 1 \cdot 23 + 8) = 32$ in ogni frame da 125 μ sec e i dati occupano $(193 - 32) = 161$ bit per frame.

La bwth disponibile per dati è $161 / (125 \cdot 10^{-6}) = 1.288 \text{ Mbps}$. Perciò $3 \text{ Mb} / 1.288 \text{ Mbps} = 2.33 \text{ sec.}$

Esercizio

Un cavo lungo 100 Km trasporta un carrier T1. La velocità di propagazione nel cavo è 2/3 della velocità della luce. Quanti bit ci stanno nel cavo?

SOL:

Carrier T1 invia un frame ogni 125 μ sec.

Segnale si propaga a 200000 Km/s; perciò 100 Km sono percorsi dal segnale in 0.5 msec.

$$Tp = 100 \text{ Km} / 200000 \text{ Km/s} = 50 \times 10^{-5} \text{ s} = 0.5 \text{ msec}$$

NB.

Vedi cap 3.2 p. 152: la lunghezza massima del cavo è fissata a 2,5 Km e considerando cavo in rame ho:

$$Tp = d/v = 2,5 \cdot 10^3 \text{ m} / 2 \cdot 10^8 \text{ m/s} = 12,5 \text{ } \mu\text{sec}$$

$$\text{Quindi dato che ho 100 Km: } 100 \text{ Km} / 2,5 \text{ Km} = 40 \text{ (ripetitori), quindi } 40 \cdot 12,5 \text{ } \mu\text{sec} = 500 \text{ } \mu\text{sec} = 0.5 \text{ msec}$$

Questo tempo corrisponde all'invio di 4 frame, ovvero $193 \cdot 4 = 772$ bit (Vedere esercizio 14).

NB.

$$0.5 \text{ msec} = 500 \text{ } \mu\text{sec}$$

$$500 / 125 = 4 \text{ frame}$$

Esercizio

Frame di 1000 bit sono inviati su un canale satellitare con bwth 1 Mbps e ritardo di propagazione 270 msec. Gli ack sono sempre trasmessi in piggyback; gli header sono di lunghezza trascurabile. Si usano #seq di 3 bit. Qual è il massimo utilizzo del canale ottenibile con (i) Idle RQ; (ii) Go-Back-N e (iii) Selective Repeat?

SOL: ritardo di trasmissione $Tx = 1000 \text{ bit} / 1 \text{ Mbps} = 1 \text{ msec.}$

$$\cdot \text{Idle RQ: } U = Tx / (Tx + 2 \cdot Tp) = 1 / (1 + 2 \cdot 270) = 0.18\%$$

$$\cdot \text{Go-Back-N ha finestra grande } K = (2^3) - 1 = 7; \text{ perciò } U = k \cdot Tx / (Tx + 2 \cdot Tp) = 7 \cdot 1 / (1 + 2 \cdot 270) = 0.0129 = 1.3\%$$

$$\cdot \text{Selective Repeat ha finestra } K = (2^n) / 2 = 8 / 2 = 4; \text{ perciò } U = k \cdot Tx / (Tx + 2 \cdot Tp) = 4 \cdot 1 / (1 + 2 \cdot 270) = 0.74\%$$

Esercizio

2000 bit di data frame

1 Mbps (canale), il quale introduce 20 ms di Tp

3 bit di segnale

Ipotizzando IDLE_RQ, calcolare utilizzo U

SOL: $U = Tx / (Tx + 2Tp)$

$$Tx = f/b = 2000 \text{ bit} / (10^6 \text{ b/s}) = 2 \cdot 10^{-3} \text{ s} = 2 \text{ ms}$$

Poiché $k=1$ in idleRQ,

$$\begin{aligned} U &= (2 \cdot 10^{-3} \text{ ms}) / (2 \cdot 10^{-3} \text{ ms}) + 2 (20 \cdot 10^{-3}) \\ &= 2 / (2 + 40) \\ &= 2 / 42 = 0,047 \rightarrow 4,7\% \end{aligned}$$

GO_BACK_N:

$$K = (2^n) - 1$$

$$\text{Quindi: } U = [(2^3) - 1] \cdot (2/42)$$

SELECTIVE_REPEAT:

$$K = (2^n)/2$$

$$\text{Quindi: } [(2^3)/2] \cdot (2/42)$$

Esercizio

2000 bit data, canale 50 Km, 100 Mbps

4 bit per #seq.

Calcolare U in GoBackN

$$U = K \cdot (Tx/Tx + 2Tp)$$

$$Tx = 2000 \text{ bit} / (100 \cdot 10^6 \text{ bps}) = 2 \cdot 10^{-5} \text{ s} = 20 \text{ us (microsec)}$$

$$Tp = 50 \cdot 10^3 \text{ m} / (2 \cdot 10^8 \text{ m/s}) = 25 \cdot 10^{-5} \text{ s} = 250 \text{ us}$$

$$K = (2^4) - 1 = 15$$

$$\begin{aligned} Tx/(Tx + 2Tp) &= 2 \cdot 10^{-5} \text{ s} / [2 \cdot 10^{-5} \text{ s} + (2 \cdot 25 \cdot 10^{-5} \text{ s})] \\ &= 2 / (2 + 2 \cdot 250) = 2 / (502) = 0.4\% \end{aligned}$$

Esercizio

Si consideri un canale satellitare di 64 Kbps privo di errore, usato per trasmettere frame di 512 B in una direzione, con ack di taglia trascurabile trasmessi nell'altra. Qual è il max throughput per taglie di finestra di 1, 7, 15 e 127?

SOL: $Tx = 512 \cdot 8 / 64000 = 0.064 \text{ sec. } [Tx = \text{bit dati}/\text{bitrate link}]$

$Tp = 270 \text{ msec. } [\text{Dato fornito ??}]$

Con finestra 1:

si inviano $(512 \cdot 8) = 4096 \text{ bit in } (0.064 + 2 \cdot 0.27) = 0.604 \text{ sec } [Tx + 2Tp]$,

perciò throughput è :

$S1 = 4096 / 0.604 = 6781.46 \text{ bps } [f/ Tx + 2Tp]$

$U = 0.064 / 0.604 = 10.59\% \quad [U = Tx / (Tx + 2Tp)]$

Con finestra grande 7 il throughput è:

$S7 = 7 \cdot S1 = 47470 \text{ bps.}$

Ora per vedere quanto deve essere grande la finestra affinché si abbia il miglior utilizzo possibile:

$[U = k \cdot Tx / (Tx + 2Tp)]$

Poniamo $U=1$ e $k=(Tx+2Tp)/Tx$

$k = (0.064 + 2 \cdot 0.27) / 0.064 = 0.604 / 0.064 = 9$,

Con finestra grande 9:

throughput $S9 = 9 \cdot S1 = 61 \text{ Kbps.}$

Quindi finestre più grandi saranno inutili, perché la banda massima è 64Kbps.

Esercizio

Un canale ha bit rate 4 Kbps e ritardo di propagazione di 20 msec. Per quale range di frame size Idle RQ ha efficienza almeno 50%?

SOL: $U = T_x / (T_x + 2 T_p)$; $T_x = \text{frame size} / \text{bwth} = f/b$

$U \geq 50\%$ se

$$\frac{\frac{f}{b}}{\left(\frac{f}{b} + 2 T_p\right)} \geq 0.5 \rightarrow \frac{f}{(f + 2 * b * T_p)} \geq 0.5$$

(ho moltiplicato il primo membro per b, sia a numeratore che a denominatore)

$$\frac{1}{(f + 2 * b * T_p)} \geq \frac{0.5}{f} \rightarrow f + 2 * b * T_p \leq \frac{f}{0.5}$$

(diviso per f entrambi i membri e ribalto numeratori e denominatori)

$$0.5(f + 2 * b * T_p) \geq f \rightarrow 0.5f + 1 * b * T_p \geq f$$

(moltiplicato per 0.5 entrambi i membri, poi svolgo la moltiplicazione a primo membro)

$$b * T_p \geq f - 0.5f \rightarrow b * T_p \geq 0.5f$$

$$f \geq 2 * b * T_p = 2 * 4000 * 0.020 = 160 \text{ bit}$$

Esercizio

velocità $v = 4kb/s$

$t_p = 20 \text{ ms}$

trovare dimensione del frame f che garantisce uno sfruttamento U del 50%

$$U = 0.5 = \frac{t_x}{t_x + 2 t_p}$$

$$t_x = \frac{f}{v}$$

$$0.5 * \left(\frac{f}{v} + 2 t_p \right) = \frac{f}{V}$$

$$0.5 * (f + 2 t_p V) = f$$

$$\frac{1}{2}f + t_p V = f$$

$$t_p V = \frac{1}{2}f$$

$$f = 2 t_p V = 2 * 20 \text{ ms} * \frac{4kb}{s} = 2 * 4 * 10^3 * 2 * 10^{-2} =$$

= 160 bit

Esercizio

8 Kbps su canale con $T_p = 40 \text{ ms}$.

In Idle RQ, quale frame size (f) per avere $U=50\%$?

SOL:

$U = T_x / (T_x + 2 T_p)$

$T_p = d/v = 40 \text{ ms}$

$50\% = T_x / (T_x + 2 T_p)$

$50\% (T_x + 2 T_p) = T_x$

Nb.finestra ha dim=1 perche siamo in IdleRQ, quindi k=1.

$$0.5 * (T_x + 2 * 40 \text{ ms}) = T_x$$

$$0.5 T_x + 0.04 \text{ s} = T_x$$

$$0.04 \text{ s} = T_x - 0.5 T_x$$

$$0.04 \text{ s} = 0.5 T_x$$

$$T_x = 0.08 \text{ s}$$

$$T_x = f/b \text{ quindi: } f = 0.08 * (8 * 10^3) = 640 \text{ bit.}$$

Protocollo HDLC (p.71)

Come abbiamo già detto a livello 2 abbiamo sia un protocollo per rete affidabile, sia uno per rete non affidabile.

Finora abbiamo esaminato la comunicazione di livello 2 affidabile.

I frame dati HDLC possono essere trasmessi attraverso collegamenti sincroni o asincroni.

Abbiamo visto come HDLC sincronizzi la trasmissione dei dati per mezzo di un protocollo di riempimento di bit (la tecnica di bit stuffing) evitando inoltre che le sequenze di terminazione compaiano all'interno dei frame.

HDLC può utilizzare o meno la modalità connessa.

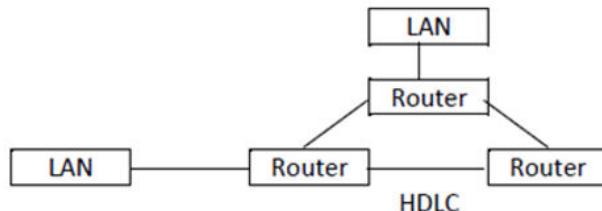


Il protocollo HDLC (High-Level Data Link Control) è uno standard internazionale definito per essere utilizzato con connessioni punto-punto, ma può lavorare con un certo numero di configurazioni e tipi di reti:

- link duplex punto-punto ISDN (PPP) - "Best effort"

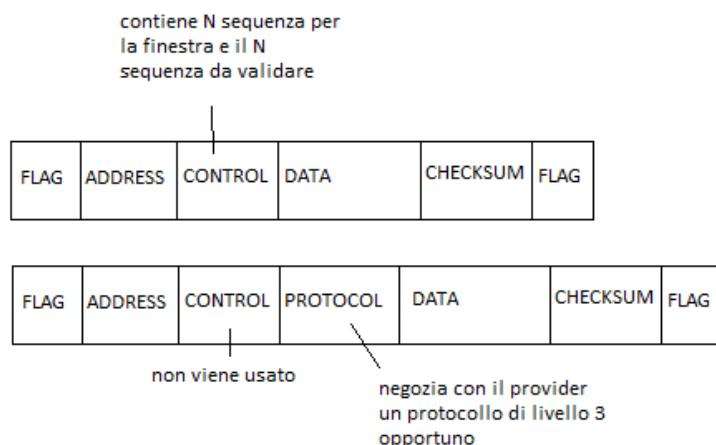


- link half-duplex multi-punto (broadcast) in una LAN (Ethernet).



Il protocollo HDLC originale è composto da alcune varianti, i frame si differenziano per header, come:

- LAPD (Link Access Procedure D-channel) su ISDN
- LLC (Logical Link Control) su LAN.



Un po' di terminologia HDLC:

- **comandi**, cioè frame inviati dalla sorgente alla destinazione
- **risposte**, cioè frame inviati dalla destinazione alla sorgente

Inoltre, se la connessione è affidabile:

Supervisory frame, è il nome con cui vengono indicati tutti i frame del controllo di flusso e di errore (es. ACK)

Unnumbered frame, è il nome con cui vengono indicati i frame necessari al setup e alla disconnessione del link

Formato frame HDLC (p. 71)

Flag	Indirizzo	Controllo	Dati	FCS	Flag
8 bit	8 bit	8 o 16 bit	Lunghezza variabile, 0 o più bit a multipli di 8	16 o 32 bit	8 bit

Analizziamo il campo **control** (sia nel caso normale o supervisor).

Nella versione a 8 bit, i bit per i numeri di sequenza sono 3, e quindi la dimensione massima della finestra di invio è 7.



N(S) = send sequence number

N(R) = receive sequence number

P/F = poll/final bit



I 4 frame di supervisione usati per implementare uno schema RQ continuo per il controllo degli errori sono:

1. RR : è un Ack in Go Back N. Ogni frame RR contiene un numero di sequenza che conferma la ricezione corretta.

2. RIJ : è un Nack in Go Back N

3. SREJ: è un Nack nel Selective Repeat

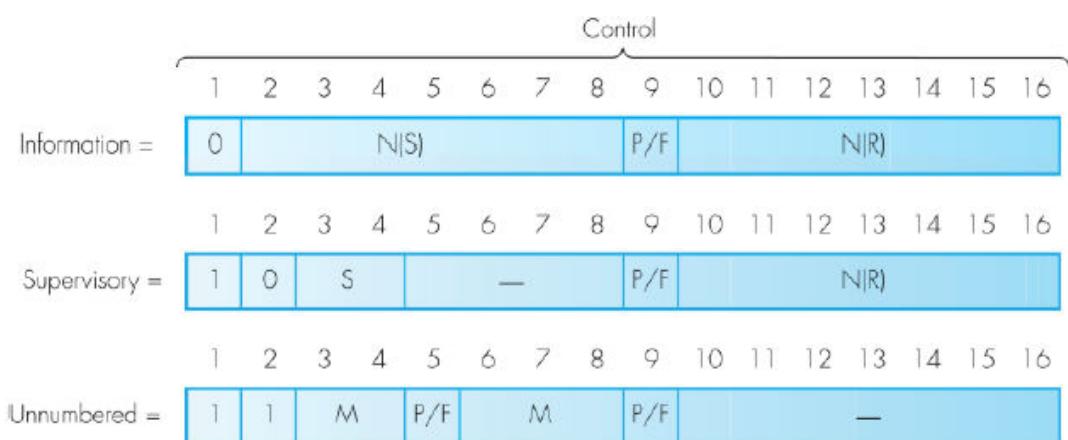
4. RNR: funzione che può essere usata dal secondario per comunicare al primario di interrompere la trasmissione di nuovi frame.



In command frames, M indicates operational mode and (unnumbered) frame type (e.g. SABM and DISC).
In response frames, M indicates frame type (e.g. UA).



Nella versione a 16 bit, i bit per i numeri di sequenza sono 7; la finestra è quindi ingrandita fino a una dimensione massima di 127. Questa versione è applicabile a situazioni particolari, come link con satelliti, che richiedono link molto lunghi o bit-rate molto alto.



Il campo P/F, noto come Poll/Final bit, quando settato a 1 in un comando, ha la semantica di una richiesta di ACK. Questo ACK viene indicato con una specifica risposta con il bit P/F settato a sua volta a 1.

Esercizio

Una entità HDLC deve trasmettere la sequenza di bit 011111111101. Quale stringa di bit produrrà in output?

SOL La sequenza in output è: 011111011111001

PPP: Point-to-Point Protocol (p.141)

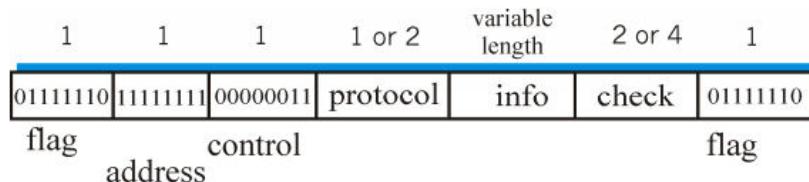
Il protocollo PPP (Point to Point – RFC 1661/2/3 e RFC 2153) nasce dall'esigenza di evitare una proliferazione eccessiva di protocolli di livello 2. È una variante di HDLC, la differenza principale è che PPP fornisce un metodo standard per trasmettere sullo stesso canale pacchetti generati da diversi protocolli di livello superiore (supporto multiprotocollo).

Per dare a PPP la necessaria flessibilità per operare su vari tipi di link, ha delle caratteristiche che gli permettono di supportare un protocollo di livello Network per il trasferimento di pacchetti Internet. Per esempio, può operare sia in connessioni affidabili (*reliable*) sia connectionless (*best-effort*).

Per dargli questa flessibilità, PPP è composto da 2 parti:

- **LCP** – *Link Control Protocol*, si occupa di stabilire un link tra le due parti, con le opzioni richieste
- **NCP** – *Network Control Protocol*, si occupa di selezionare e configurare un protocollo di rete comune tra sorgente e destinazione

L'uso originario di PPP era quello di connettere due computer usando una linea telefonica, ed è largamente utilizzato dai provider per connettere gli utenti ad Internet con una connessione dial-up, rimpiazzando il più vecchio protocollo SLIP. Ora il suo utilizzo è più frequente per le linee DSL, dove viene utilizzato sopra un livello ATM (PPPoA) o Ethernet (PPPoE). Molto utilizzato anche nelle connessioni GPRS



Flag = delimitatore di trama (byte stuffing)

Address = l'unico valore possibile di questo campo è 11111111

Control = l'unico valore possibile è la sequenza 00000011

Protocol = serve a negoziare con il provider il protocollo di livello 3 disponibile sulla macchina dell'utente. Indica il protocollo che ha generato la PDU contenuta nel campo Info (ad es. IP, IPX, AppleTalk, Decnet, LCP...)

Info = contiene la PDU generata dal protocollo specificato nel campo Protocol (la maximum frame size di default è pari a 1500 byte)

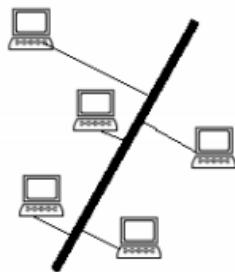
Check = codice CRC (Cyclic Redundancy Code) per l'error detection.

Procedura

1. Inizia una connessione PPP
2. Manda un frame (LCP) di livello 2 per settare i parametri
3. Manda un frame (NCP) di livello 3 per negoziare l'IP e il protocollo che ho a bordo con il provider
4. Il PC diventa host internet (identificabile tramite IP)
5. Invio frame PPP
6. Disconnetto inviando prima un frame NCP e poi un frame LCP

Dopo aver visto i collegamenti punto-punto, adesso trattiamo reti private, dove oltre alla possibilità di parlare con Internet, esiste anche la possibilità di parlare con gli altri utenti connessi alla stessa LAN. Parliamo quindi di reti broadcast.

DataLink nelle Reti broadcast



Nelle reti broadcast il frame viene inviato a tutte le stazioni della LAN. L'idea è quella di emulare un collegamento punto-punto, facendo in modo che tutti i ricevitori tranne il destinatario droppino il frame.

Quindi un frame contiene src e dst. Inoltre, visto che il canale può essere usato contemporaneamente da più nodi, può esserci collisione e deve essere quindi implementato un protocollo di *mutua esclusione* o di *accesso unico al canale condiviso*.

Esistono diversi protocolli di accesso multiplo al canale e che utilizzano uno due approcci:

- Approccio deterministico
- Casualità

Deterministico

Garantisce l'accesso multiplo con un "token". La stazione che in quel momento ha il token può trasmettere. Condizione necessaria è che deve esistere uno ed un solo token nella rete. Il token si può passare in modo circolare, ovvero ciclicamente ogni stazione riceve il token insieme al frame. In questo modo garantisco fairness nella rete.

Vantaggi:

- risolve la mutua esclusione
- garantisce fairness

Svantaggi di un approccio deterministico:

- se solo un nodo nella rete deve trasmettere, devo comunque passare il token (ho overhead nella rete)
 - se ho tanti nodi in rete, ogni nodo deve attendere che il token faccia il giro dell'intera rete prima di poter trasmettere ancora
 - il token è l'unica cosa che garantisce il protocollo, se viene perso? Si blocca l'intera rete. Per rilevare la perdita del token si potrebbe usare un timer che controlla per quanto tempo non viene usato il canale. Si presenta un nuovo problema: chi genera un nuovo token?
- Meccanismo di elezione della macchina che deterrà il nuovo token
- Se aggiungo un nodo, devo riconfigurare l'ordine logico dei nodi per ogni nodo della rete.

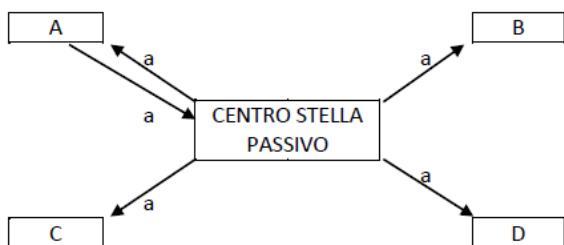
Token ring

Un esempio è la rete **Token-ring**, una rete locale sviluppata da IBM.

Protocollo deterministico che aggiunge un token alla rete

Solo chi ha il token può parlare

Quando l'host che detiene il token ha finito di parlare rilascia il token, che continua in modo circolare



Se nodo B vuole inviare un messaggio, devo evitare che si sovrapponga a quello mandato da A (collisione)

Con un metodo come questo impongo che ci sia un centro che rimetta a posto le cose in caso di errore.

2 proposte:

- con centro stella attivo, in ogni ingresso ho una coda e il centro stella visita una di essa alla volta
- temporizzo i nodi, ogni nodo può parlare in un determinato tempo

Sono proposte deterministiche e che garantiscono equità ma danno troppo potere al centro.

Controllo centralizzato: improponibile per una rete locale con molti nodi

Controllo distribuito: il controllo è affidato ai nodi.

Reti Broadcast con controllo totalmente distribuito

Ogni nodo ha le stesse funzionalità di rete degli altri, così ogni nodo è indipendente dagli altri e ha in se tutti gli elementi per poter accedere alla risorsa, indipendentemente dagli altri nodi.

Ogni nodo trasmette sul mezzo in comune. Se solo un nodo trasmette, non ho nessun problema di concorrenza.

Se due o più nodi trasmettono nello stesso istante, o comunque in tempi tali che i pacchetti spediti risultano sovrapposti, i frame o una parte di essi risultano corrotti. Ci si accorge dell'errore solo dopo un timeout.

Un algoritmo di questo tipo (vedi ALOHA) è ideale per una rete con tante stazioni, ma in cui solo poche trasmettono. Un miglioramento è il Carrier-Sense: prima di trasmettere sul canale ascolto che sia libero. Il CS si può attuare solo se il tempo di propagazione è basso.

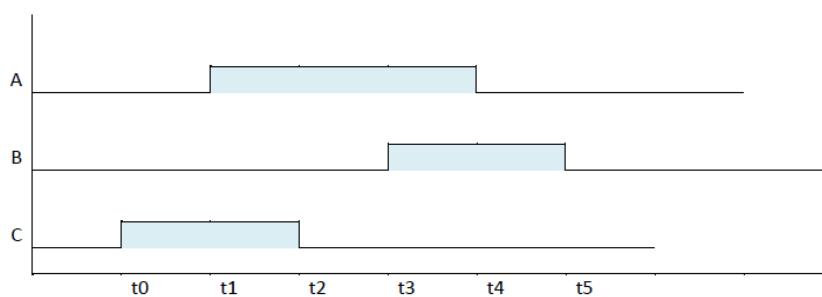
ALOHA

Il controllo è distribuito su tutti i nodi.

Ogni stazione risolve il problema di mutua esclusione. Più efficienza e Fairness ("equità", tutti possono accedere al canale).

L'accesso non è deterministico, ma casuale, quindi ci affidiamo alla probabilità

Ogni nodo è libero di trasmettere quando vuole, con la speranza che non ci sia un altro nodo che sta trasmettendo



Frame A collide con C da t1 a t2 e con B da t3 a t4!

Nella prima rete, sviluppata così, la rete ALOHA (si usava un satellite e si facevano comunicare i nodi sulla terra) la collisione era tollerata. Posso accettare questo compromesso se ho pochi nodi e se ho poco traffico.

Il caso maggiore di throughput (sfruttamento) della rete era del 18%

Migliorando questo approccio si è arrivati quasi al 100%.

ALOHA con Carrier Sense

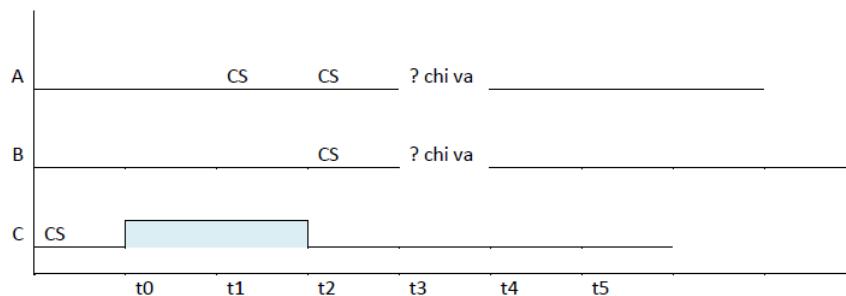
È possibile solo se i tempi di propagazione sono piccoli.

Una stazione, se vuole trasmettere, deve prima ascoltare il canale, verificando se è libero.

La stazione effettua la verifica:

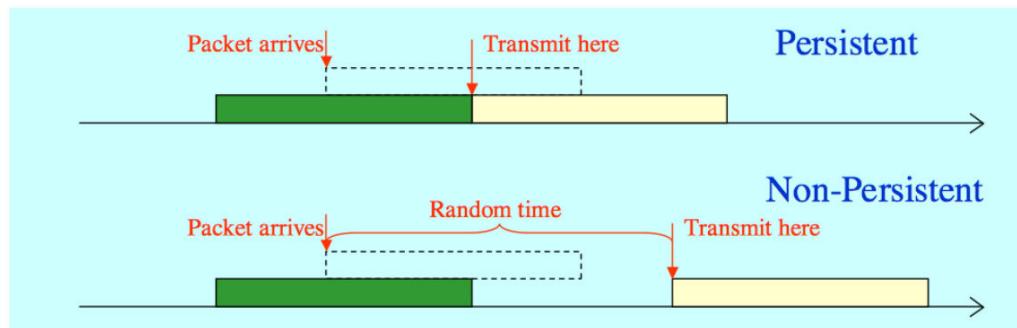
- facendo un polling continuo finché non è libero → *continuo ad ascoltare il canale (1 persistent)* → Ethernet

Se due stazioni fanno polling insieme, inizieranno a trasmettere insieme e si verificherà sicuramente una collisione.



Quindi si aspetta un tempo casuale per ascoltare il canale e poi si riprova a controllare (Protocollo non-persistent)

È difficile che 2 nodi si trovino contemporaneamente a richiedere il canale. La connessione però viene molto "spalmata nel tempo" e risulta lenta e poco efficiente.



CSMA 1 persistente: continuo ad ascoltare il canale, appena trovo libero trasmetto

CSMA non-persistente: non trasmetto appena il canale si libera ma aspetto un tempo casuale (*backoff*)

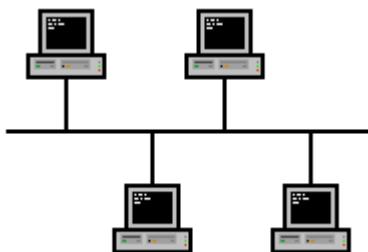
CSMA p-persistente: quando si libera il canale trasmetto con una probabilità p , mentre con probabilità $(1-p)$ aspetto un tempo casuale.

Ethernet è di tipo CSMA persistente e quando si verifica una collisione attende per un tempo casuale a riascoltare il canale.

Protocollo CSMA CD 1p

Si controlla continuamente se il canale è libero, appena è libero si trasmette; in caso di collisione si attende un intervallo casuale prima di ritrasmettere.

È il protocollo usato da Ethernet.



Le caratteristiche di 802.3 sono ben riassunte nell'acronimo CSMA/CD:

- **Carrier Sense:** ogni stazione sulla rete locale ascolta continuamente il mezzo trasmisivo;
- **Multiple Access:** il mezzo trasmisivo è condiviso da tutte le stazioni, ognuna delle quali vi accede da un punto differente;
- **Collision detection:** Il problema delle collisioni si risolve grazie alla natura stessa delle reti broadcast. Se ogni nodo che trasmette è in grado anche di sentire cosa c'è nel canale, allora posso confrontare ciò che sto trasmettendo con ciò che ricevo. Se le sequenze di bit sono identiche allora sto spedendo il frame correttamente e senza interferenze, altrimenti significa che qualche altra stazione sta trasmettendo in contemporanea. Se i bit sono diversi, si interrompe ogni trasmissione e per de-sincronizzarsi ogni stazione genera un ritardo casuale.

Ma qual è il range di questo ritardo casuale?

Binary Exponential Backoff (BEB)

Parto da un ritardo piccolo perché spero che le stazioni che hanno generato una collisione siano solo due, se collido ancora il range casuale si allarga e così via per i successivi tentativi, abbassando sempre di più la probabilità di collisione:

$$r = \text{rand}(0, 2^i - 1)$$

dove i è il numero delle collisioni che sono avvenute.

Il numero r indica il numero di slot di tempo che la stazione deve aspettare prima di ritentare la trasmissione.

Uno slot di BEB è 51.2 us (microsec.), che come vedremo successivamente è il tempo minimo con cui sono sicuro di rilevare una possibile collisione. Le collisioni vengono minimizzate perché tra uno slot di attesa e l'altro, c'è il tempo per l'invio di un frame.

Esercizio

Un nodo causa 3 collisioni consecutive.

Qual'è l'intervallo di tempo di attesa per il riinvio del frame?

l'intervallo è definito dalla regola:

$$(0: 2^i - 1) \text{ dove } i \text{ è il numero di collisioni consecutive}$$

$$(0: 2^3 - 1) = (0: 7)$$

vanno però intesi come multipli del tempo base 51.2μs

il tempo di attesa è un valore random che va da

0μs a 7 * 51.2μs

Esercizio

Una stazione connessa ad una rete Ethernet subisce 5 collisioni consecutive nel tentativo di spedire un frame. In quale range temporale avverrà la successiva ritrasmissione secondo l'algoritmo BEB ?

SOL:

Dopo la prima collisione, la ritrasmissione è generata nell'intervallo [0;1]

dopo la seconda nell'intervallo [0; (2^2)-1=3]

dopo la quinta nell'intervallo [0; (2^5)-1=31]

Quindi da $(31 \times 51.2) = 1587.2$, l'intervallo è [0 ; 1587.2 microsec.]

Nb. BEB: $I=[0;(2^N) - 1]$

TIME SLOT per un frame Ethernet è 51.2 microsec.

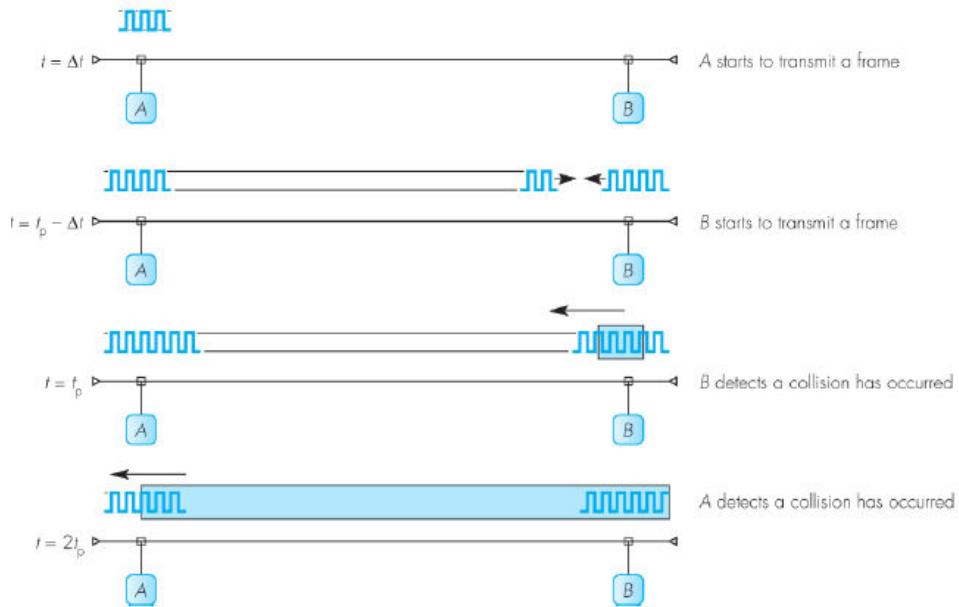
Esercizio

Se in una rete Aloha non sincronizzata i pacchetti durano t , quant'è lungo il tempo di vulnerabilità, cioè il tempo nel quale un pacchetto potrebbe generare una collisione?

SOL Dato che una stazione non ascolta prima di trasmettere, un pacchetto la cui trasmissione è iniziata in un tempo inferiore a t , e prima dell'istante di trasmissione, allora causerà una collisione. Inoltre, ogni pacchetto iniziato durante il tempo di trasmissione causerà anche una collisione. Quindi il tempo di vulnerabilità è $2t$.

Come rilevo la collisione? Per poter controllare che i miei bit arrivino integri, devo avere $t_x = 2t_p$

Solo così sono sicuro di inviare il mio frame e fare in modo che, se ha colliso con la stazione più remota, ho il tempo necessario di accorgermene quando mi torna indietro. In questo modo io posso controllare e nel caso fermarmi, mentre sto ancora inviando.



Se il tempo di trasmissione non fosse correttamente dimensionato, si potrebbero verificare casi in cui potrei aver terminato l'invio dell'intero frame prima che i bit per la verifica della collisione siano tornati indietro.

In tal caso, quando ricevo, non saprei più riconoscere se :

- i bit che ho trasmesso sono bit corrotti a causa di una collisione
- i bit sono integri ma non sono i miei, poiché provenienti da un'altra stazione che nel frattempo ha trasmesso.

Inoltre, quando una stazione rileva una collisione, invia ancora 32 bit detti **jam sequence**, per essere certi che tutte le stazioni coinvolte nella collisione si accorgano dell'imprevisto.

Lo standard IEEE 802.3 impone che t_x sia maggiore o uguale a 51.2 us, chiamato **slot-time**, che include, oltre a $2t_p$, il ritardo dei repeater e una maggiorazione "per stare tranquilli". È calcolato pensando alla massima lunghezza di una rete Ethernet, cioè 2.5 Km. In 51.2 us riesco a mandare (minimo) 64 Byte, quindi tutti i frame di Ethernet devono avere almeno questa lunghezza.

Esempio

STANDARD IEEE 802.3

Lunghezza massima 2.5 Km

Usati almeno **4 RIPETITORI** per riuscire a ottenere tale lunghezza.

- Un ripetitore è uno strumento puramente fisico che rigenera il segnale elettrico (bit) che altrimenti si esaurirebbe prima!

Velocità di trasmissione 10 MB/s (impulso ogni 100 ns)

Esempio con frame di 1 MB

$$t_x = \frac{10^3}{10^7} = 10^{-4} = 100 \text{ ns}$$

$$t_p = \frac{2,5 * 10^3}{2 * 10^8} = 12,5 \text{ ns}$$

i ripetitori generano un ritardo di propagazione che porta tale tempo nel caso peggiore a $t_p = 25 \text{ ns}$

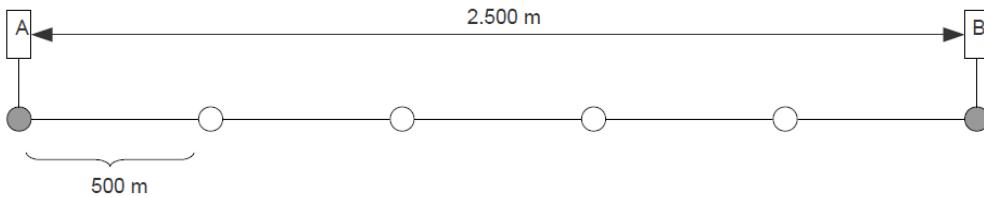
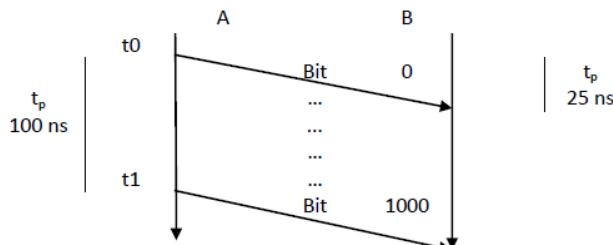
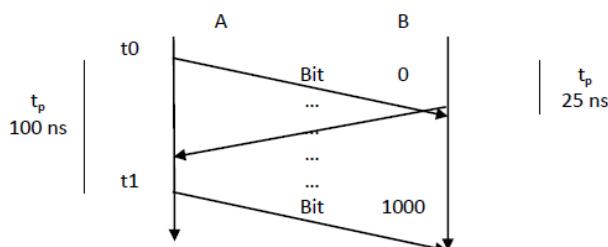


Figura 3.2: vincoli sulla lunghezza dei cavi Ethernet

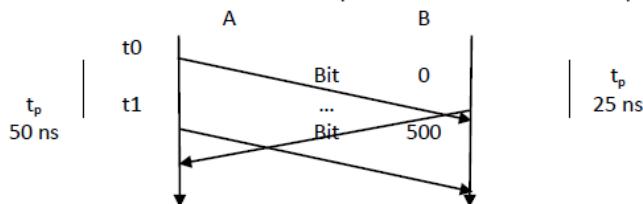


Ok, ma ammettiamo che B un attimo prima che gli arrivi il messaggio di A faccia CS. Troverebbe il canale libero e quindi potrebbe trasmettere. Creando quindi una collisione:



A questo punto il nodo A si accorge che riceve qualcosa che non manda lui, capisce quindi che è avvenuta una collisione.

Proviamo ora a vedere se il frame al posto di 1000 B fosse ad esempio di 500 B



Mentre B riceve il frame di A, lo stesso A non sta più trasmettendo e dunque non è in grado di rilevare la collisione.

Bisogna dunque porre un vincolo sulla lunghezza minima del frame:

Con i parametri definiti sopra, nell'esempio 1 la collisione veniva rilevata dopo un tempo:

$$t_0 + 2t_p = 0 + 2 * 25\text{ns} = 50\text{ ns}$$

Quanti bit invio in 50 ns alla velocità di 10 MB/s ?

$$500 \text{ bit} \sim 512 \text{ bit} = 64 \text{ B}$$

$$\text{e } k = 51,2 \text{ ms}$$

Un frame secondo lo Standard IEEE 802.3 deve avere una lunghezza non minore a 64 B!

Esercizio

Qual è la dimensione minima in bit di un frame 802.3 ? Nel caso un frame subisca 3 collisioni consecutive, qual è il tempo massimo atteso dal protocollo CSMA prima di ritentare l'accesso al canale in base all'algoritmo binary-exponential-backoff ?

SOL La dimensione minima del frame 802.3 è di 512 bit, o 51,2 us.

Nel caso di tre collisioni consecutive, in base all'algoritmo BEB, il tempo di attesa è al massimo:

$$(2^3)-1 \text{ SLOT-TIME}$$

$$\text{Cioè } 7 \cdot 51,2 \text{ us} = 358,4 \text{ us.}$$

Esercizio

Abbiamo 2 nodi distanti 360 m collegati tramite 2 ripetitori

Ogni ripetitore produce un ritardo di 40 μ s su una banda con velocità di 8 Mb/s

Trovare la lunghezza minima del frame.

$$t_p = \frac{\text{lunghezza canale}}{2 * 10^8} + \text{ritardi} = \frac{3.6 * 10^2}{2 * 10^8} + 2 * 40 * 10^{-6} = 1.8 * 10^{-6} + 80 * 10^{-6}$$

$$= 81.8 * 10^{-6}$$

$$t_x = 2t_p \rightarrow \frac{f}{v} = 2t_p \rightarrow f = 2t_p * v = 2 * 81.8 * 10^{-6} * 8 * 10^6 = 163.6 * 8 = \mathbf{1308.8 \text{ bit}}$$

Esercizio

Abbiamo 2 nodi distanti 1 km collegati su una banda con velocità di 1 Gb/s

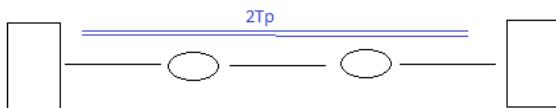
Trovare la lunghezza minima del frame.

(identico sopra, lo svolgo in modo simile)

$$t_x = 2t_p \rightarrow \frac{f}{v} = 2 * \frac{\text{lunghezza}}{2 * 10^8} \rightarrow f = 2 * \frac{\text{lunghezza}}{2 * 10^8} * v = 2 * \frac{10^3}{2 * 10^8} * 10^9 = 10^4 = \mathbf{1000 \text{ bit}}$$

Esercizio

CSMA/CD su tratte da 200 m con 2 repeater che producono in tot. 20 us di ritardo, 8 Mbps su rame. Trovare taglia frame f (minima).



Incomincio a determinare la distanza.. $3 * 200 \text{ m} = 600 \text{ m}$

$T_p = 600 \text{ m} / (2 * 10^8 \text{ m/s}) = 3 * 10^{-6} \text{ s} = 3 \text{ us}$

$T_p (\text{tot}) = 3 \text{ us} + 20 \text{ us} = 23 \text{ us}$ [da un estremo all'altro del canale]

Per CSMA/CD è necessario trasmettere per un tempo $T_x \geq 2T_p$

$2T_p = 2 * 23 = 46 \text{ us}$

$F = 2T_p * bwth = 46 \text{ us} * (8 * 10^6 \text{ bps}) = 46 * 8 = \mathbf{368 \text{ bit}}$

Esercizio

Si consideri una rete con tratte di lunghezza massima 120 mt., dove si possono inserire al massimo due ripetitori consecutivi tra due dispositivi, con ritardo introdotto dai ripetitori 40 μ sec., banda dei canali 8 Mbps e mezzo in rame. In tale rete si vuole usare CSMA/CD; determinare la minima taglia di frame necessaria.

SOL:

Se si considera come limite massimo 2 ripetitori nella tratta, allora la massima distanza tra due dispositivi è $(120 * 3) = 360 \text{ mt.}$

Il tempo di propagazione sul mezzo è

$T_p = 360 \text{ mt.} / 2 * 10^8 \text{ mt/s} = 1.8 * 10^{-6}$,

A questo valore si devono aggiungere $(40 * 2) = 80 \mu\text{sec.}$ di ritardo indotto dai ripetitori.

In totale $T_p = 81.8 * 10^{-6} \text{ sec.}$

Il tempo di trasmissione del frame $T_x = \text{frame size} / bwth$ deve essere \sim pari a $2 T_p$, da cui :

$\text{frame size} = 2 T_p * bwth = 2 * (81.8 * 10^{-6}) * (8 * 10^6) = 1309 \text{ bit.}$

Esercizio

Qual è la minima dimensione di frame per una rete CSMA/CD con bwth 1 Gbps su un cavo di 1 Km senza ripetitori, tale che la velocità di propagazione è 2/3 della velocità della luce?

SOL:

$T_p = 1 \text{ Km} / 200000 \text{ Km/s} = 5 * 10^{-6}$,

Cioè $(1 * 10^3 \text{ m}) / (2 * 10^8 \text{ m/s}) = 0.5 * 10^{-5}$

Perciò il T_x è 10^{-5} , infatti: $2 T_p = 2 * 0.5 * 10^{-5}$

Perchè CSMA/CD funzioni, non deve essere possibile inviare un frame intero in un tempo inferiore a questo valore.

$T_x = f/b$, perciò $f = T_x * b = 10^{-5} * 10^9 = 104 \text{ bit}$, che è la taglia minima dei frame.

Formato frame LAN (p.157)

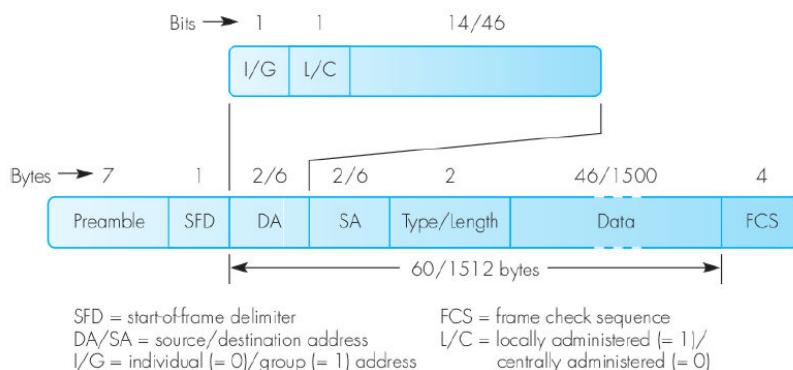
Precedentemente abbiamo visto come è composto un frame per le reti che utilizzano PPP.

Poi abbiamo studiato il livello 2 per le reti LAN e abbiamo visto che mentre prima la comunicazione era diretta, ora ci si trova a dover comunicare con più macchine contemporaneamente.

Dopo aver risolto il problema delle collisioni e aver visto quali regole devono essere rispettate (distanze tra nodi, dimensione del frame e tempo $T_p=2T_x$), vediamo come è fatto il formato di questi frame.

Nonostante Ethernet abbia diverse tipologie, l'elemento comune è nella struttura del pacchetto Ethernet detto *frame*, che è rimasto fedele alla versione originale.

La dimensione minima della frame in Ethernet è 512 bit, 64 byte. Questo è il motivo per cui il campo *Data* ha una dimensione minima di 46: la somma di tutti gli altri campi, nel caso peggiore, è 18, e $18 + 46 = 64$.



Questo è il *frame* ovvero il pacchetto dati ricevuto dallo strato di datalink nella pila di protocolli.

Gli elementi sono:

- **Preamble** (preambolo), di 7 byte: questi primi byte hanno valore 10101010 e servono a "svegliare" gli adattatori del ricevente e a sincronizzare gli oscillatori con quelli del mittente.
 - **Start Frame Delimiter (SFD)**, di 1 byte: questo byte ha valore 10101011 e la serie dei due bit a 1 indica al destinatario che sta arrivando del contenuto importante; è utilizzato il codice Manchester; svolge la stessa funzione del campo *flag* della trama HDLC;
 - **Destination MAC address** (indirizzo di destinazione), di 6 byte: questo campo contiene l'indirizzo LAN dell'adattatore di destinazione; se l'indirizzo non corrisponde, il livello fisico del protocollo lo scarta e non lo invia ai livelli superiori;
 - **Source MAC address** (indirizzo sorgente), di 6 byte;
 - **Type/Length** (campo tipo), di 2 byte: questo campo indicava il tipo di protocollo del livello di rete in uso durante la trasmissione per le frame PPP; nel caso di frame IEEE 802.3 indica la lunghezza del campo dati (payload);
 - **Payload** (campo dati), da 46 a 1500 byte: contiene i dati reali, che possono essere di lunghezza variabile in base alla MTU della Ethernet; se i dati superano la capacità massima, vengono suddivisi in più frame, mentre se i dati non raggiungono la lunghezza minima di 46 byte, viene aggiunto del *padding* (riempitivo) della lunghezza opportuna. Almeno 46 B del campo DATA devono essere presenti, perché altrimenti non si raggiungerebbero i 64 byte necessari per riempire lo slot time (18 B header + 46 = 64 B min payload length);
 - **Frame Check Sequence (FCS)**, o controllo a ridondanza ciclica (CRC), di 4 byte:

fcs (*Frame Check Sequence* - verifica di sequenza della trama)

La parte finale di una trama Ethernet o Token Ring che contiene il valore di controllo per quella trama. Viene anche chiamato crc (dal nome della particolare operazione matematica che viene utilizzata per definire il contenuto). La fcs viene calcolata prima della trasmissione eseguendo una particolare operazione matematica sui bit contenuti nella trama. Il valore ricavato viene accodato alla trama medesima così che all'altro estremo il destinatario possa eseguire la stessa operazione sui dati ricevuti, confrontarne il risultato con il valore contenuto nel campo fcs (parte finale della trama) e accertarsi che non ci siano stati errori di trasmissione. Se il valore di fcs calcolato dal destinatario è diverso da quello contenuto nella trama, quest'ultima viene scartata.

Indirizzo Ethernet MAC

Gli indirizzi sono tutti a 6 byte, in quanto Ethernet definisce uno schema di indirizzamento a 48 bit: ogni nodo collegato, quindi, ha un indirizzo Ethernet *univoco* di questa lunghezza, che è l'indirizzo fisico della macchina ed è associato all'hardware.

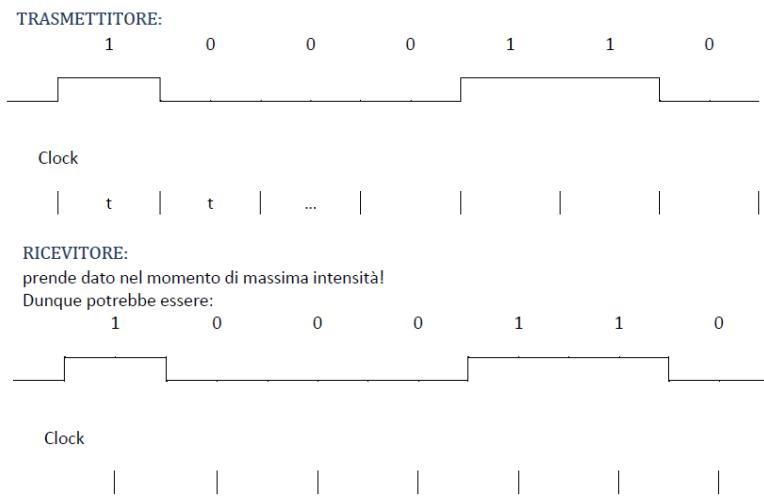
Tali indirizzi sono anche detti *indirizzi hardware*, *indirizzi MAC* o *MAC address*, oppure *indirizzi di livello 2*.

Manchester Encoding (p.40)

Per fare in modo che una stazione sia sicura di riconoscere che qualcuno sta trasmettendo, bisogna che il segnale di trasmissione dati sia riconoscibile dal segnale idle del canale libero.

Come si differenzia un dato di tutti 0 da un segnale assente (IDLE) ?

Finora ..



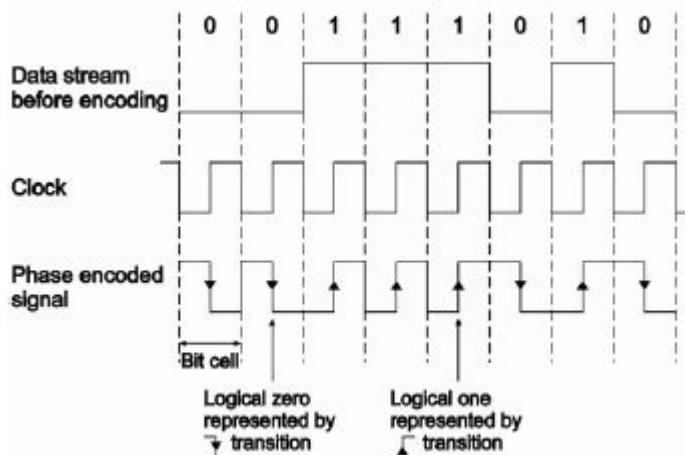
Come si vede, i due clock non sono sincronizzati. Devo trovare un modo per mandare al ricevitore, oltre al dato, indicazioni riguardo al clock del trasmettitore.

Si usa quindi una specifica codifica.

Manchester Encoding

Due impulsi nello stesso tempo t :

- Ogni 1 è rappresentato da una transazione da 0 a 1
- Ogni 0 è rappresentato da una transazione da 1 a 0
- IDLE è rappresentato dall'assenza di transazioni



Il ricevitore per capire quando arriva un dato, utilizza i bit del preamble (7 byte), cioè la sequenza di 01010101010101..

SEQUENZA NORMALE

0 1 0 1 0 1



MANCHESTER ENCODING

0 1 0 1 0 1



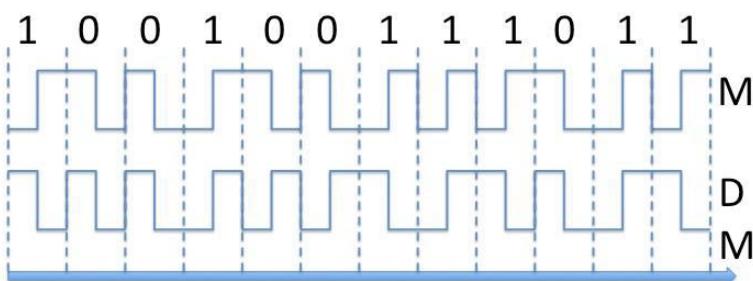
Si vede che è la stessa sequenza di bit traslata di un tempo $t/2$.

Il clock di Ethernet deve quindi raddoppiare. Gli orologi dei nodi possono però sincronizzarsi ad ogni bit.

Esercizio

Data la stringa di bit 1 0 0 1 0 0 1 1 1 0 1 1, rappresentare il segnale prodotto usando codifica Manchester e NRZI. Nel secondo caso, si assuma che inizialmente la linea si trovi in stato alto.

SOL:



Manchester: 1 sale ; 0 scende (potrebbe essere al contrario)

Manchester Differenziale: l'inverso del Manchester.

NRZI: lo 0 mantiene il verso stabilito per il segnale 1 (in questo caso alto-basso); ogni volta che incontro 1, il segnale si inverte. L'1 comanda! [cit. Sara]

Il preambolo del frame Ethernet serve proprio a sincronizzare gli orologi di sorgente e destinazione, ed è formata da una serie di bit codificati per rappresentare il clock (vedremo le codifiche più avanti quando parleremo dei tipi di cavi Ethernet, paragrafo 5.4).

Sottolivelli del livello 2 (vedi anche p. 61)

Nella pila di protocolli di rete del modello di riferimento ISO/OSI, 802.3 occupa il livello fisico e la parte inferiore del livello datalink. IEEE ha infatti ritenuto opportuno suddividere il livello data-link in due parti: LLC, *Logical Link Control* e MAC, *Media Access Control*. Il sottolivello LLC è comune a tutti gli standard della famiglia IEEE 802, mentre il sottolivello MAC è più strettamente legato al livello fisico, e le sue diverse implementazioni hanno il compito di offrire un'interfaccia comune al livello LLC. Fra queste implementazioni vanno ricordate in particolare 802.4, token bus e 802.5, token ring.

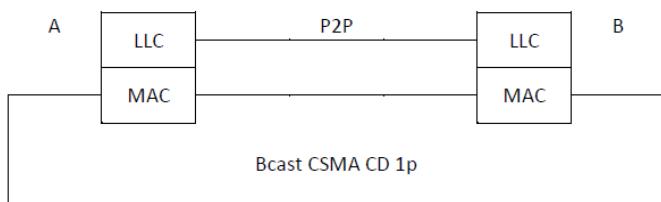
2B) LLC – Logical Link Control

Per gestione rete punto-punto

2A) MAC – Multiple Access Control

Per gestione rete broadcast

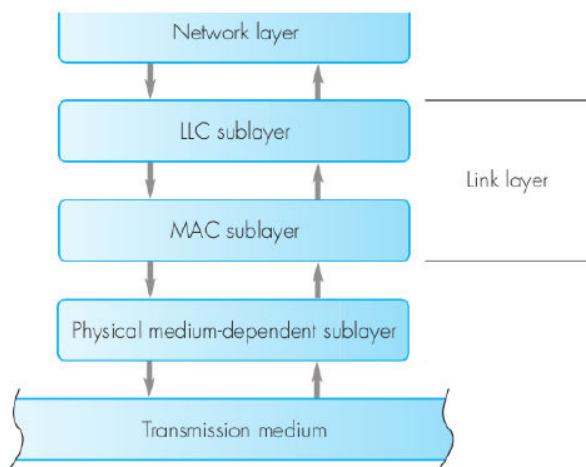
Ogni scheda di rete possiede un numero identificativo della macchina che è unico a livello mondiale (MAC-address su 48 bit)



MAC, garantisce la mutua esclusione sul canale condiviso (uso di CSMA/CD)

LLC, fornisce l'affidabilità se richiesta e le funzionalità punto-punto (PPP)

In questo livello mi comporto come se avessi la funzionalità punto-punto, poiché MAC ha già provveduto al livello inferiore a tutte le problematiche del canale condiviso.



A seconda delle implementazioni, l'LLC può fornire al livello superiore distinte modalità di servizio:

- **Tipo 1 (logical data link)** è un servizio *non affidabile* e *non orientato alla connessione*, costituito da singoli datagrammi che vengono trasmessi in modo indipendente l'uno dall'altro e senza richiedere alcuna comunicazione preliminare. È possibile la trasmissione verso una sola stazione (*unicast*), più stazioni (*multicast*) o l'intera rete (*broadcast*). D'altra parte, non è garantita la consegna dei singoli elementi e neppure che sia rispettata la sequenza di trasmissione. Inoltre, non è prevista alcuna forma di correzione degli errori né di controllo di flusso; qualora tali funzioni siano necessarie, devono essere fornite dai protocolli di livello superiore.
- **Tipo 2 (data link connection)** è un servizio *affidabile* e *orientato alla connessione*, che richiede l'apertura di un canale di comunicazione tra una stazione sorgente e una destinazione (*unicast* o punto-punto) per consentire lo scambio dei dati. Sono presenti meccanismi di correzione degli errori e di sequenziamento che garantiscono la consegna dei dati inviati nella sequenza di trasmissione. Si tratta di una connessione simmetrica, in cui ciascuno dei due interlocutori è responsabile dei dati di cui è l'origine. È concettualmente il protocollo HDLC.

Domini di collisione

In Ethernet si può usare un bus lineare e nella fattispecie un cavo coassiale per collegare i vari dispositivi. Anziché usare il cavo coassiale però potremmo usare un hub.

Hub

È un centro stella passivo di livello fisico (prende segnale e lo riinvia), permettendo la connessione di più host.

Un hub incorpora anche le funzionalità di repeater. Il controllo è sulle singole stazioni.

Infatti, il dominio di collisione è unico, l'hub non interviene nel protocollo, il lavoro viene fatto dal MAC che sta nelle schede di rete dei PC.

Ogni PC collegato tramite uno o più hub si trovano nello stesso dominio di collisione.

Un hub non rileva le collisioni ma le propaga. Il cavo collegante un nodo ad un hub è detto cavo di drop.

Esempio: canale a 10 Mbps

A ----- 10 km ----- B

La distanza di 10 km è molto maggiore dello standard (2.5 km).

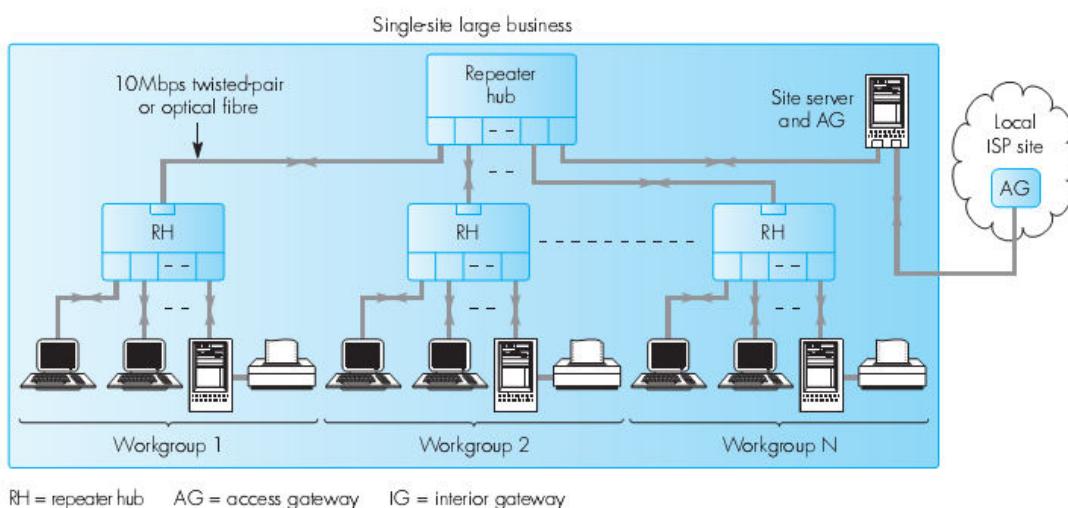
Sono ardito, voglio continuare ad usare CSMA CD. Però devo garantire la correttezza, quindi aumento la lunghezza dei frame:

$$t_p \approx 4 \cdot 51.2 \mu s \approx 20.0 \mu s$$

Quindi la lunghezza del frame: 64 byte \times 4 = 256 byte di frame (perché $4 \times 2.5 = 10$ Km)

Osservazione: dimensione della frame, lunghezza del cavo e velocità di trasmissione sono strettamente dipendenti.

Per dividere i PC collegati alla rete in più domini di collisione (utile da fare quando ho troppe collisioni), mi devo servire di un Bridge.



In un dominio di collisione, se le collisioni sono troppe la rete è poco utilizzata. Una soluzione plausibile è adoperare un Bridge.

Si collegano così due domini di collisione facendo passare il traffico da un dominio di collisione ad un altro.

Il numero di collisioni è dimezzato e l'utilizzo della rete aumenta.

Il bridge è una specie di router di livello 2.

Class I repeater (ripetitore di Classe I)

Un tipo di hub Fast Ethernet che esegue una traduzione del segnale entrante prima di ritrasmetterlo al fine di consentire l'interconnessione di diversi mezzi trasmissivi all'interno dello stesso dominio di collisione.

Class II repeater (ripetitore di Classe II)

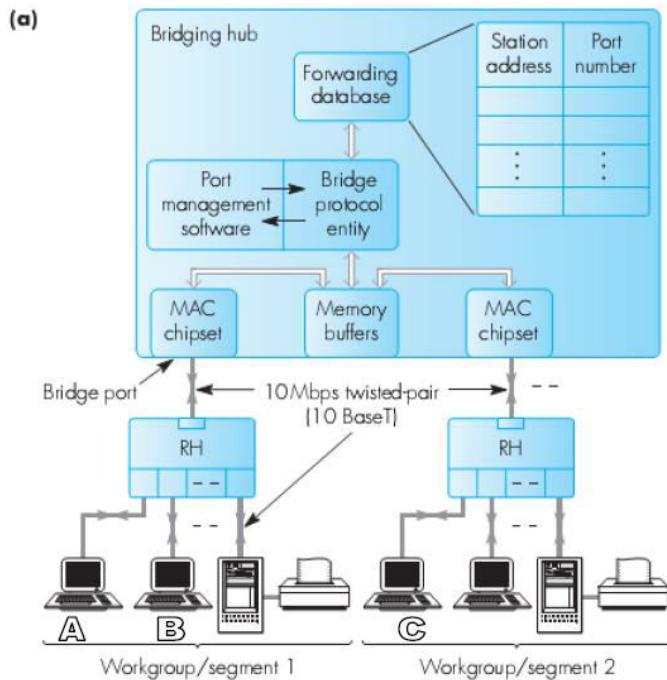
Un tipo di ripetitore che si limita a ritrasmettere ossia a ripetere il segnale entrante, inviandolo ad altri dispositivi che usano tutti lo stesso mezzo trasmissivo. Non esegue nessun tipo di traduzione.

Bridge

Usiamo quindi i Bridge, che dividono i domini di collisione.

Un bridge possiede una scheda di rete con relativo MAC (quindi può collidere anche lui).

Un bridge opera anche a livello 2 e controlla, tramite l'indirizzo MAC della destinazione, in che dominio di collisione, e quindi in quale porta di uscita, instradare il frame che arriva. Controlla anche i pacchetti corrotti, che evita di inoltrare.



In questa figura, se nodo A vuole parlare con nodo B, il bridge non fa nulla perché i nodi appartengono alla stessa porta, è quindi l'hub a farli parlare. Se invece A vuole parlare con C, è il bridge che deve occuparsi della comunicazione. Come capisce se il nodo è collegato a una porta piuttosto che un'altra? Attraverso una tabella (tabella di Forwarding).

Tabella di Forwarding

Mette in relazione tutti i nodi della rete con le porte del bridge a cui è connesso.

1. All'inizio la tabella del Bridge è vuota
2. Al Bridge arriva un messaggio da A
3. Il bridge salva nella tabella la porta relativa ad A
4. Instradamento (*forwarding*) verso B

Come fa il bridge a instradare verso B ?

-Se porta di B non è nota, il frame viene inoltrato a tutte le porte tranne quella di A (flooding). Questo perché l'hub non sa ancora a quale dominio di collisione appartiene il nodo B (quindi a quale porta è associato).

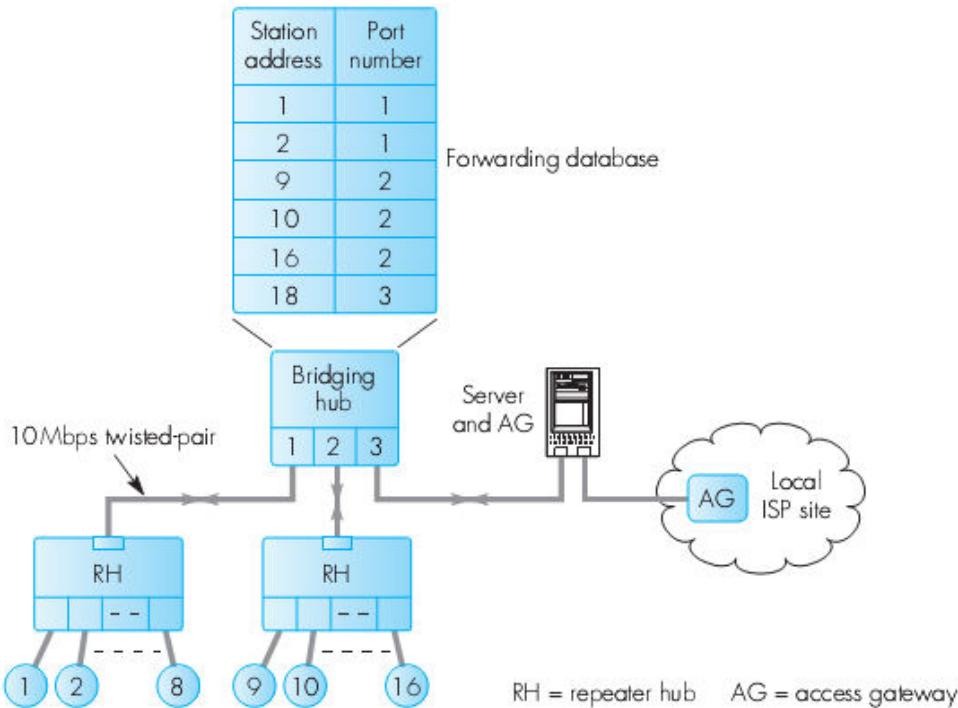
-Se porta di B è nota ed è la stessa di A, non viene inoltrato.

-Se porta di B è nota e non appartiene allo stesso dominio di collisione di A, il frame viene mandato solo sulla porta di B

5. Nel caso B risponde, si riparte dal punto 2, questa volta però il bridge conosce già a quale porta inoltrare il frame per A.

Dopo un certo periodo la tabella sarà resettata e ricostruita per tenerla aggiornata.

(b)



Quindi il Bridge separa i domini di collisione ed è invisibile agli utenti.

Il Bridge è un esempio di strumento "Store and Forwarding", infatti:

- 1- riceve frame
- 2- lo salva nel buffer (store)
- 3- controlla la tabella
- 4- invia a seconda dei casi analizzati precedentemente (*forwarding*)

Il bridge è capace di autoconfigurarsi senza l'intervento umano. Possiede un algoritmo di configurazione della rete.

Anche il router si può dire che faccia la stessa cosa ma lui a livello 3 (*routing*). Dal punto di vista strutturale sono due apparecchi molto simili.

Un bridge ha una scheda di rete per ogni dominio di collisione a cui è connesso.

Il bridge è una stazione a tutti gli effetti, come quelle nei domini di collisione a cui è collegato.

Il bridge deve quindi conoscere la configurazione della rete per permettere il passaggio di frame da una sottorete a un'altra.

Il bridge trasparente è un bridge che apprende col tempo la dislocazione delle macchine nelle varie LAN.

Spanning tree algorithm (algoritmo della diramazione ad albero)

I bridge devono assicurarsi che nella rete non esistano circoli viziosi e cioè che la stessa trama non continui a girare eternamente in tondo mantenendo la rete inutilmente occupata alla ricerca di un percorso per arrivare alla propria destinazione. Questo succede quando tra due reti o due segmenti esistono più percorsi possibili. Le trame continuano a propagarsi su tutti i segmenti possibili e a volte tornano indietro creando circoli viziosi. La trama esce "dalla porta" e rientra "dalla finestra" di continuo. Per intercettare queste "trame vagabonde" si sceglie uno tra i vari percorsi possibili e s'indirizzano tutte le trame indirizzate a una certa destinazione su quel percorso. Il percorso che viene scelto deve essere quello più corto tra tutti quelli disponibili. La decisione viene presa seguendo un algoritmo che si chiama spanning-tree (diramazione ad albero). Se il percorso originariamente scelto venisse a mancare per un difetto della rete, il bridge passerebbe al secondo così da mantenere una continuità di connessione. Il risultato finale è che la rete appare strutturata come un grande albero gerarchico senza percorsi doppi tra nessuna delle sue stazioni.

L'efficacia di questo metodo è limitata alle reti locali e non funziona bene su una connessione geografica poiché comporta il continuo scambio d'informazioni tra i bridge.

Source routing (instradamento dalla sorgente)

Detta anche source route o source-route bridging. Una tecnica d'instradamento tipica delle reti Token Ring, in cui la stazione sorgente (source) determina il percorso necessario per arrivare a una certa destinazione e inserisce tale informazione all'interno dei pacchetti che invia.

In questo modo il pacchetto trasporta con sé non solo l'indirizzo di destinazione, ma anche le informazioni su come arrivarci. È sinonimo di end-to-end routing (instradamento da punto terminale a punto terminale). L'intero percorso viene determinato prima di trasmettere il pacchetto. La tecnica è alternativa a quella di transparent bridge rispetto alla quale prevede anche una maggiore complessità sul fronte del bridge, tuttavia è più efficiente dopo che il percorso è stato trovato la prima volta poiché evita al bridge di ricercarlo da capo ogni volta.

Switch

La differenza tra switch e hub è che i link che collegano i bridge agli switch non sono necessariamente link con CSMA/CD, la capacità di processing dello switch è elevatissima e di conseguenza non abbiamo più collisioni.

Lo switch gestisce anche frame che arrivano contemporaneamente su porte di input diverse. Ho solo bisogno di garantire che le frame siano del tipo di CSMA/CD per quando verranno inoltrate nei vari dominii.

Inoltre, per garantire la velocità di cui è capace, lo switch richiede collegamenti full-duplex, e quindi non possiamo collegare direttamente hub (vedi figura).

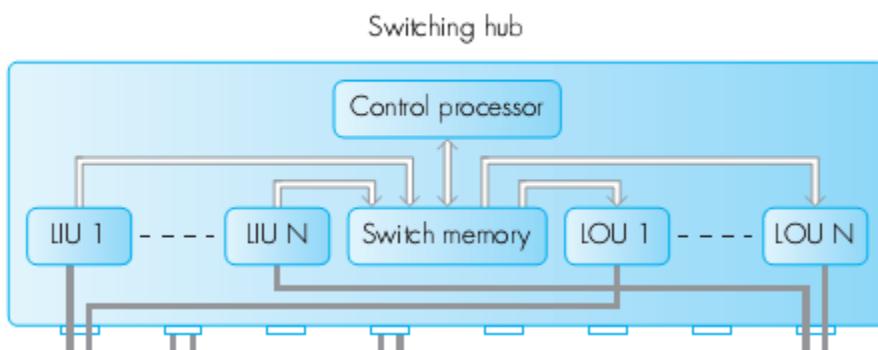
Quindi:

- il bridge

- Ha MAC
- Fa carrier sense
- Fa collision detection

- lo switch

- Opera in modalità full-duplex (due cavi, uno di andata e uno di ritorno)
- Ha collegamenti punto-punto (non ci sono collisioni)
- Tutti i vincoli di lunghezza massima del cavo e velocità di trasmissione non ci sono più, rimangono i vincoli per il formato del frame
- Apparecchio di livello 2 molto veloce grazie alla sua architettura



Lo switch è stato introdotto perché su un server possono essere spedite tantissime frame provenienti dagli host, e se il server fosse direttamente accessibile tutto colliderebbe. Quindi si mette uno switch di mezzo che bufferizza il tutto.

Avendo abbastanza soldi, si potrebbero usare solo switch e non avere più nessun problema di collisione tra tutte le stazioni ad esso collegate.

Lo switch è uno strumento compatibile al 100% con ethernet, si parla infatti di switch ethernet.

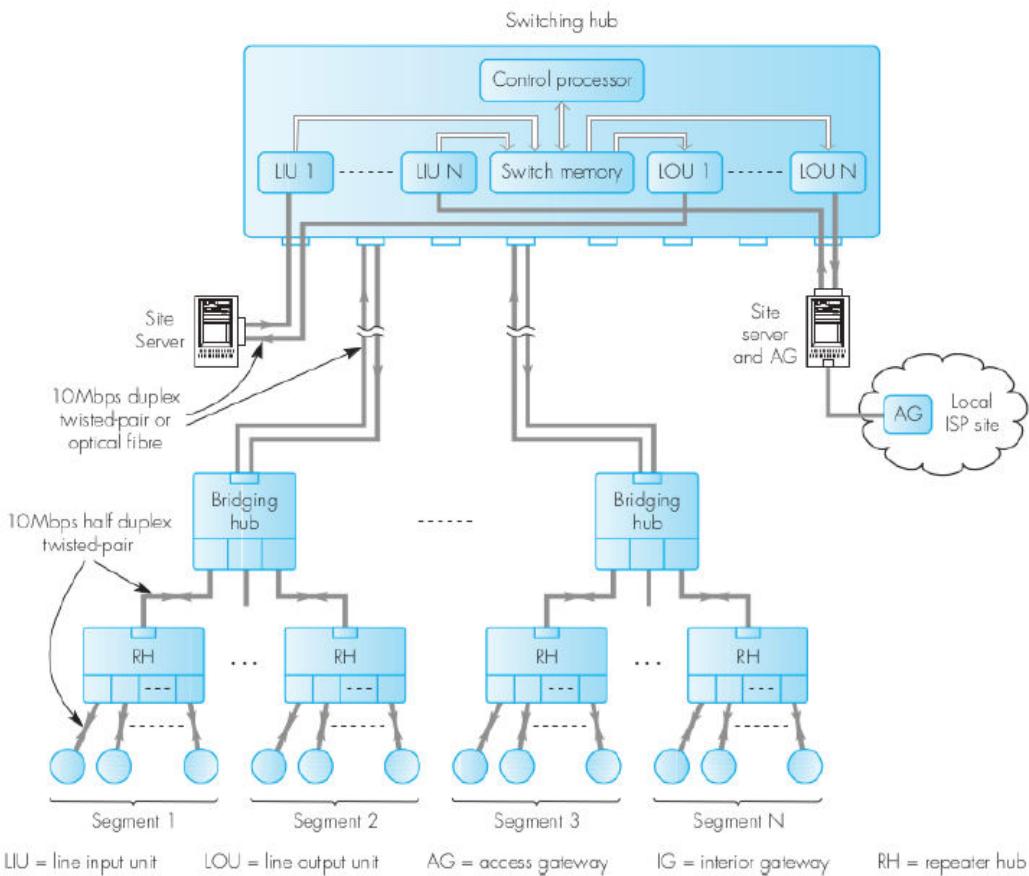
Al giorno d'oggi gli switch sono diventati economici e hanno più o meno gli stessi prezzi dei bridge. Si creano dunque reti con solo switch semplici, che possiedono una scheda ethernet dotata di livello MAC e LLC ma sui quali non si fa Carrier Sense. La compatibilità del frame deve però essere garantita anche da switch economici.

Lo switch interpreta anche lui l'indirizzo MAC ma è diverso dal bridge.

Commuta la linea d'ingresso con una d'uscita, è trasparente e crea la tabella di forwarding.

In fase di apprendimento, lo switch, quando non conosce la porta di uscita, propaga su tutte le porte compresa quella da cui ha ricevuto il frame (tipo broadcast).

Uno switch è un bridge con le porte non CSMA CD e con alta potenza di calcolo.



Rispetto al bridge, lo switch esegue tutte le proprie elaborazioni in hardware e non via software, perciò non rallenta il fluire del traffico tra i segmenti. In gergo tecnico si dice che la connessione sia *wire speed* cioè lasci transitare i pacchetti alla velocità massima consentita dal tipo di conduttore usato per il cablaggio. Nella realtà, un rallentamento esiste sempre, anche se marginale, e la sua entità dipende dal modo in cui lo switch funziona. La primissima tecnica di switching, che eredita in toto la modalità operativa dei bridge, si chiama *store-and-forward*. Ogni trama che arriva su una delle porte dello switch viene incamerata per intero in una speciale porzione di memoria (buffer) e quindi scartata o trasferita a un altro segmento a seconda dell'indirizzo di destinazione (mac address) indicato al suo interno. L'operazione è velocissima, ma comporta in ogni caso un certo rallentamento perché la trama deve arrivare per intero nel buffer dello switch prima di cominciare a essere ritrasmessa su un'altra porta (a cui corrisponde un altro segmento, appunto).

È la tecnica di commutazione più affidabile, poiché prima di rispedire il pacchetto ci si accerta di averlo per intero e se ne verifica la correttezza attraverso il calcolo del crc (*Cyclic Redundancy Check*), ed è l'unica utilizzabile quando si collegano segmenti funzionanti a velocità diverse, come Ethernet e Fast Ethernet, per esempio. Tuttavia su impianti molto veloci, come nel caso di una dorsale che funziona tutta a 100 Mbps o più, il numero di trame in circolazione è molto elevato e il ritardo che si accumula per la registrazione di ciascuna si fa sentire.

L'alternativa ideata per eliminare quest'ultimo inconveniente si chiama commutazione *cut-through*. La parola significa "tagliare attraverso", "prendere una scorciatoia" e in effetti è proprio quello che accade. Non appena lo switch comincia a ricevere una trama su una qualsiasi delle sue porte, ne legge l'indirizzo di destinazione e, se questo corrisponde a un segmento collegato a un'altra porta, inizia immediatamente a trasmettere la trama senza aspettare che questa sia arrivata per intero.

In questo modo, dopo aver letto l'indirizzo, la trasmissione in uscita avviene quasi in contemporanea con la ricezione, e il ritardo è minimo (fino a 20 volte inferiore a quello della tecnica store-and-forward). Benché molto efficace sotto il profilo della velocità, questa tecnica presenta il difetto di far passare tutto quel che segue senza controllo alcuno.

Inoltre il beneficio del sistema cut-through diminuisce quando il traffico diventa molto intenso e continuo.

Ethernet

Una rete locale a 10 Mbit per secondo inizialmente sviluppata dal Palo Alto Research Center di Xerox e successivamente rifinita da Digital Equipment, Intel e Xerox. È la più diffusa tra le reti locali nel mondo e il suo standard finale porta la firma del comitato 802.3 dell'IEEE (*Institute of Electrical and Electronics Engineers*).

Breve introduzione

Ai tempi delle prime installazioni di Ethernet l'unico mezzo di trasmissione che poteva operare a 10Mbps erano i cavi coassiali, quindi inizialmente fu utilizzato un cavo coassiale con fili spessi; la lunghezza dei segmenti poteva essere al massimo di 500 m, ma siccome andava utilizzato su lunghezze ben maggiori era obbligatorio l'uso di ripetitori. Il numero massimo di ripetitori che potevano essere utilizzati era 4, dovuto al fatto che il segnale arrivava attenuato alle estremità del segmento ed andava ogni volta amplificato e riportato alla forma originale. Inoltre questo fenomeno limitava la lunghezza massima a 2,5 Km (diametro della rete).

Il vero problema di questi cavi era però il loro spessore, essendo infatti elevato non potevano essere piegati e questo rendeva difficoltosa l'installazione. Furono così utilizzati cavi coassiali più fini, ma a causa dell'aumento della resistenza elettrica interna, la lunghezza massima dei segmenti dovette essere abbassata a 185 m.

Quando si scoprì che l'utilizzo di una coppia di cavi intrecciati permetteva di raggiungere velocità di 10 Mbps esse divennero lo standard per Ethernet. Venne utilizzata una topologia con hub a stella. La lunghezza massima di 100 m imposta non si rivelò un problema in quanto nella maggior parte degli uffici era sufficiente per effettuare il collegamento.

Ora, per connettere gli hub tra di loro, viene utilizzata la fibra ottica in quanto ha un limite maggiore sulla lunghezza e maggiore velocità.

Al livello fisico del modello ISO/OSI, 802.3 prevede esclusivamente trasmissioni via cavo in **banda base**, a velocità di 10, 100 e 1000 Mbps, su cavi coassiali, doppini intrecciati (schermati e non) e fibre ottiche. Queste e altre caratteristiche sono riassunte negli acronimi usati per le varie implementazioni del livello fisico, tutti del tipo *NBaseA*, essendo *N* la velocità di trasmissione, *Base* indica che l'implementazione opera in banda base, ed *A* è una sigla legata al tipo di cavo utilizzato e ad altre caratteristiche salienti. Fra le implementazioni ormai non più installate vanno ricordate [10Base5](#) (*thick Ethernet*), [10Base2](#) (*thin Ethernet*), [10Base-T](#).

- **10Base5**, cavo coassiale (fili spessi) con lunghezza massima dei segmenti di 500m
- **10Base2**, cavo coassiale (fili sottili) con lunghezza massima dei segmenti di 185 m
- **10BaseT**, topologia hub con una coppia di cavi intrecciati di lunghezza massima dei segmenti di 100 m
- **10BaseF**, topologia hub con fibra ottica di lunghezza massima dei segmenti di 2 Km

Attualmente vengono installate soprattutto le varianti a 100 Mbps e superiori, come [100Base-T](#), al momento certamente fra le più diffuse.

Tipi di Ethernet

Esistono tre principali versioni di Ethernet a 10 Mbit per secondo, distinte per tipo di cavo utilizzato:

10Base-5 (10 Mbps Baseband 500 metri - 10 Mbps banda base a 500 metri su coassiale grosso)

Formato originale delle reti Ethernet che impiega un cavo coassiale del diametro di circa un centimetro (3/8 di pollice) e speciali oggetti, detti transceiver, per collegare il computer alla rete. I dati vengono trasmessi a 10 Mbit per secondo in modalità broadband (a impulsi) su una distanza massima di 500 metri. Il cavo è pesante e abbastanza rigido e non può essere tagliato in nessun punto. La connessione avviene per mezzo di un attacco a *vampiro*: una speciale punta contenuta nel transceiver buca la guaina, attraversa la calza di rame esterna (che costituisce il secondo elettrodo per un cavo coassiale) e penetra fino a raggiungere la parte conduttrice interna (anima). Data la delicatezza dell'operazione, è facile creare cortocircuiti che disturbano il funzionamento dell'intero segmento. Una parte della distanza utile viene sprecata perché, data la sua rigidità, il cavo non può descrivere curve con un raggio inferiore a una cinquantina di centimetri e la distanza minima tra due stazioni è di 2,5 metri, il che significa che nel caso esistano diverse macchine molto vicine tra loro è necessario avvolgere il cavo su se stesso creando una sorta di anello. Questo sistema di cablaggio è stato rapidamente soppiantato da un cavo coassiale sottile che non presenta problemi di curvature o di peso, non richiede l'uso di transceiver e impone una distanza minima tra le macchine di soli 60 centimetri. Il cavo coassiale grosso viene ormai utilizzato solo per le dorsali, cioè quei segmenti che attraversano tutto l'edificio per unire tra loro altri segmenti, e anche in questa funzione sta progressivamente cedendo il passo alla fibra ottica.

10Base-2 (10 Mbps Baseband 200 metri - 10 Mbps banda base a 200 metri su coassiale sottile)

Uno degli standard per le reti Ethernet, particolarmente diffuso alla fine degli anni Ottanta. Definisce le modalità per trasmettere dati a 10 Mbit per secondo con modalità baseband (a impulsi) su un cavo coassiale a sezione sottile (3/16 di pollice equivalenti a circa 5 millimetri di diametro, chiamato in codice RG 58) su una distanza massima di 200 metri (185 volendo rispettare rigorosamente le specifiche dello standard). Il cavo passa da una macchina all'altra come un lungo serpente flessibile e si connette in ogni fermata a uno speciale connettore a baionetta (bnc in sigla) sagomato a T. La base della T s'inserisce nella scheda del computer, mentre alle altre due estremità troviamo lo spezzone di coassiale entrante e lo spezzone di coassiale uscente. Ai due estremi del segmento vanno posti due tappi terminatori che

impediscono al segnale elettrico di essere riflesso e di tornare indietro mescolandosi con quello in transito. Se s'interrompe il cavo in qualsiasi punto, l'intero segmento cade. Si tratta di un sistema di cablaggio economico e abbastanza flessibile: per aggiungere una macchina alla rete basta aggiungere un pezzo di coassiale e non bisogna comperare concentratori (hub) come nel caso del 10Base-T. Può integrarsi con una rete 10Base-T o 100Base-T con l'aggiunta di un dispositivo ponte (bridge).

10Base-T (10 Mbps Baseband Twisted pair - 10 Mbps banda base su doppino ritorto)

Una versione di rete locale Ethernet funzionante su doppino di tipo telefonico approvata come standard nel 1990. Il nome significa che funziona a 10 Mbit per secondo con una trasmissione di tipo baseband (trasmissione a impulsi) su doppino ritorto (twisted pair) non schermato. La topologia fisica ha la forma di una stella: tutti i computer si collegano a un concentratore (hub) con tratte di filo individuali, perciò se una tratta s'interrompe l'unica a soffrirne è la macchina interessata, anziché tutto il segmento come invece accade con l'Ethernet su cavo coassiale. La topologia elettrica è tuttavia quella tipica delle reti Ethernet: un bus (percorso continuo e comune) che corre da una macchina alla successiva con una terminazione elettrica a entrambi gli estremi. È il concentratore che collega i singoli spezzoni di filo in entrata e in uscita in modo che siano uno la continuazione elettrica dell'altro. La lunghezza massima della tratta di doppino che unisce la singola stazione all'hub è di 100 metri. È possibile collegare in cascata fino a 4 hub per un totale di 500 metri di diametro massimo (la distanza tra le due stazioni agli estremi). Il doppino (simile al cavo usato in telefonia) è preferibile perché, oltre a essere più economico, consente maggiore sicurezza. Infatti se si trancia il cavo che collega un computer al concentratore viene isolata solo quella macchina e tutte le altre continuano a trasmettere. Se invece s'interrompe uno spezzone di coassiale tutto il segmento cessa di funzionare.

Fast Ethernet

Un tipo particolare di rete Ethernet che funziona a 100 Mbit per secondo invece che a 10 Mbps. Come cavo trasmissivo si usa un doppino ritorto non schermato (simile al cavo telefonico ma di qualità superiore) per tratte massime di 100 metri. Usa un concentratore (hub) diverso dall'Ethernet a 10 Mbps e funziona su reti più corte in ragione dell'alta velocità: 205 metri complessivi invece di 500 metri complessivi che si raggiungono a 10 Mbps (la distanza aumenta a seconda della qualità dei ripetitori - hub impiegati - e della combinazione di doppino e fibra ottica e si arriva anche a più di 300 metri). Per il resto funziona esattamente come un Ethernet convenzionale e per questo motivo quasi tutte le schede Fast Ethernet in circolazione possono anche funzionare a 10 Mbps.

100Base-FX (100 Mbps Baseband Fiber X - 100 Mbps banda base su fibra)

Una delle tre tecniche di trasmissione previste dalle reti Fast Ethernet. Usa un cavo in fibra multimodale (62,5/125 micron oppure 50/125 micron) per trasmettere su distanze maggiori. Combinando una connessione su doppino verso la stazione di lavoro e una connessione in fibra ottica dall'hub alla dorsale si arriva alla distanza massima di 308 metri, contro i 205 consentiti dallo standard Fast Ethernet nel caso in cui si usi il doppino anche per la connessione con la dorsale.

100Base-T4 (100 Mbps Baseband Twisted pair 4 - 100 Mbps banda base su doppino ritorto a quattro coppie)

Una versione di Fast Ethernet che funziona su doppino ritorto di categoria 3 (il meno pregiato) usando quattro coppie invece di due per inviare e ricevere i dati da ciascuna workstation. Tre coppie sono la ricezione e la trasmissione, mentre la quarta viene usata per segnalare eventuali collisioni. Non permette la connessione di due hub in cascata visto che gli unici hub usati per le reti 100Base-T4 sono tipicamente di Classe I perché li si vuole rendere contemporaneamente utilizzabili anche per le reti 100Base-TX, molto più diffuse. La trasmissione 100Base-T4 serve a riutilizzare un vecchio cablaggio già realizzato per 10Base-T. Può tuttavia accadere che non esistano abbastanza coppie per servire ciascun posto di lavoro e perciò sia in ogni caso necessario cablare di nuovo. In questo caso è consigliato il passaggio al doppino di categoria 5 e a 100Base-TX.

Caso di studio

Rivediamo ora l'architettura ethernet appena presentata: cavi da 10 Mbps e topologia a stella con hub.

Quando abbiamo introdotto gli hub abbiamo detto che la distanza tra l'hub e il nodo può essere al massimo di 100 m.

Quindi due nodi collegati allo stesso hub avranno una distanza di 200 m.

$$t_p = \frac{2 \cdot 10^2}{2 \cdot 10^8} = 10^{-6} = 1 \mu\text{s}$$

Ricordando che il tempo critico (per collisioni) era RTT:

$$t_0 + 2t_p = 2 \mu\text{s}$$

Aggiungendo il ritardo dovuto all'hub, possiamo approssimare a 5 μs



Il nostro obiettivo è quello di aumentare di 10 volte la velocità di trasmissione che da 10 MBps diventa 100 MBps

Consideriamo ora quale sarà il nuovo tempo di trasmissione mantenendo le frame da 512 bit studiate precedentemente per 10BaseT

$$Tx = f / b =$$

$$\frac{512 \text{ bi } \square}{10^8} = 512 \cdot 10^{-8} = 512 \mu\text{s}$$

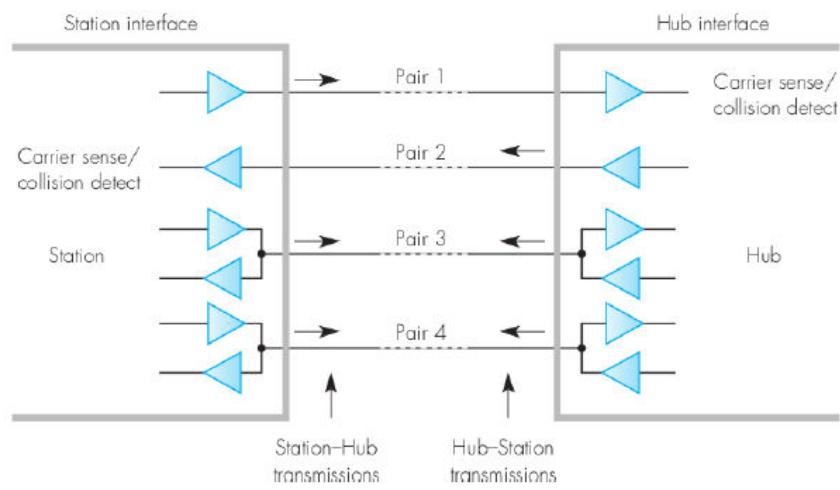
Per avere ancora un sistema funzionante in grado di rilevare le collisioni, dovremo inviare frame in un tempo di 512 microsec.

Rimane ancora un problema:

Dato che i cavi utilizzati sono 4 e che la nuova velocità di trasmissione è di 100 Mbps, lo stesso cavo del 10BaseT non è in grado di trasferire dati a una velocità massima di 30 MB/s dunque è stata introdotta una nuova architettura chiamata appunto **100Base4T** (o 100B4T).

100B4T è nato per incrementare la velocità di Ethernet, portandolo a 100 Mbit/s ma usando gli stessi cavi.

Vengono usate 4 coppie di cavi di cui due consentono la trasmissione full-duplex CSMA/CD (sono bidirezionali)

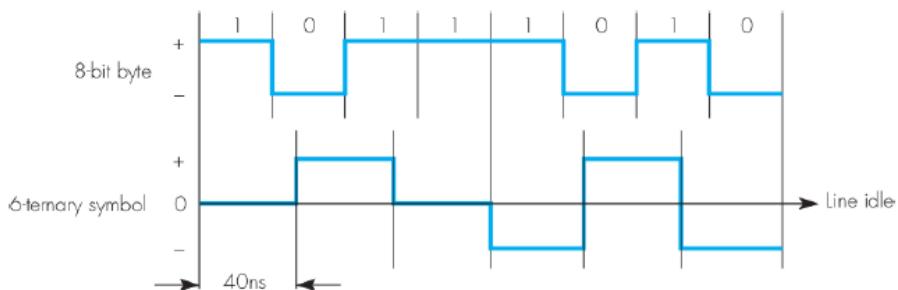


Si usano tre cavi per spedire :

- i cavi 1,3,4 per la stazione
 - i cavi 2,3,4 per l'hub
- e il rimanente in ricezione per rilevare le collisioni → raggiungo i 100 Mbps.

Ognuno dei tre cavi ha una velocità di 33 Mbps .

Con queste velocità, la codifica Manchester non è però più applicabile; si usa quindi la codifica **8B6T**, che converte 8 bit in 6 simboli in base 3, basati sulla fase del segnale.



2^8 mappato in 3^6

8B6T (8 binario 6 ternario)

100Base-TX (*100 Mbps Baseband Twisted pair X* - 100 Mbps banda base su doppino ritorno a due coppie)

Una versione di Fast Ethernet che funziona su doppino ritorno di categoria 5 (più pregiato) usando due coppie di doppino, una per inviare e una per ricevere i dati da ciascuna workstation. Costituisce il metodo di trasmissione tipico delle reti Fast Ethernet, oltre che il più conveniente, inoltre è l'unica a consentire trasmissioni full duplex su doppino a 100 Mbps.

La 100BaseTX è diventata un Ulteriore standard per la velocità 100 Mbps, differisce da 100Base4T perché necessita di cavi di qualità superiore, compensando però questo vincolo con la possibilità di migliorare le trasmissioni di file audio/video. Usa la codifica **4B5B**.

Data symbols

4-bit data group 5-bit symbol

0000 -----	11110
0001 -----	01001
0010 -----	10100
0011 -----	10101
0100 -----	01010
0101 -----	01011
0110 -----	01110
0111 -----	01111
1000 -----	10010
1001 -----	10011
1010 -----	10110
1011 -----	10111
1100 -----	11010
1101 -----	11011
1110 -----	11100
1111 -----	11101

Control symbols

IDLE	----- 11111
J	----- 11000
K	----- 10001
T	----- 01101
R	----- 00111
S	----- 11001
QUIET	----- 00000
HALT	----- 00100

1000BaseF – Gigabit Ethernet

Se volessi una connessione di 1 GB/s? avrei una lunghezza massima del cavo di 10 metri!

1000BT

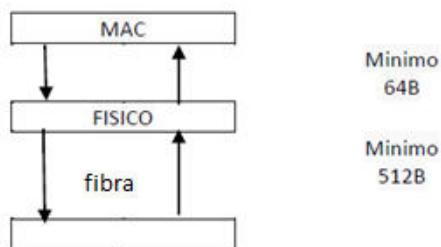
È stato realizzato un cavo ibrido che consente una distanza tra un nodo e un hub di 200 m (400 da nodo a nodo).

$$t_p = \frac{4 \cdot 10^2}{2 \cdot 10^8} = 2 \cdot 10^{-6}$$

$$2t_p = 4 \mu s \approx 5 \mu s$$

Calcolando la frame size scopriamo che viaggiano dunque 5120 bit in questo tempo!

Frame passa da min 64 B a min 512 B.



Se l'informazione non raggiunge la taglia minima, sono state trovate due soluzioni:

- *Carrier Extension*, viene aggiunto un padding che verrà poi rimosso in ricezione (Aggiungo bit spazzatura a livello fisico)
- *Clustering*, aggrego più frame da 64 byte per raggiungere i 512 byte e poi mando tutto in un singolo frame. Allo scadere di un tempo prefissato mando quello che ho bufferizzato, se necessario applicando anche Carrier Extension.

Esercizio

Si consideri una LAN che usa CSMA/CD su un canale fisico (bus) da 1 Gbps. Il cavo è lungo 1 Km. Si assuma che la velocità del segnale sia circa 0,7 la velocità massima teorica. Quale saranno le dimensioni minime di un frame?

SOL prima calcolo la velocità del segnale:

$$v = 0,7 \cdot 3 \cdot 100000 \text{ Km/s} = 0,7 \cdot 3 \cdot 10^8 \text{ m/s} = 2,1 \cdot 10^8 \text{ m/s}$$

Le dimensioni minime del pacchetto devono essere sufficientemente grandi da richiedere un tempo di trasmissione minimo di $2t$ dove t è il tempo di propagazione (T_p) tra le due stazioni più lontane.

Se la LAN è lunga 1 Km avremo:

$$T_p = d/v = (1 \cdot 10^3 \text{ m}) / (2,1 \cdot 10^8 \text{ m/s}) = 4,76 \cdot 10^{-6} \text{ s}$$

Ora dato che la velocità di trasmissione è 1 Gbps, il numero di bit che devono essere trasmessi per trasmettere per un tempo $2t$ sarà:

[ricordiamo la formula $T_x = f / bwth$]

$$f = T_x \cdot bwth = 2T_p \cdot bwth = 2 \cdot (4,76 \cdot 10^{-6} \text{ s}) \cdot (1 \cdot 10^9 \text{ bps}) = (9,6 \cdot 10^{-6}) \cdot (1 \cdot 10^9) = 9,6 \cdot 10^3 = 9600 \text{ bit} = 9,6 \text{ Kbit} = 1200 \text{ B} (= 9600 \text{ bit} / 8)$$

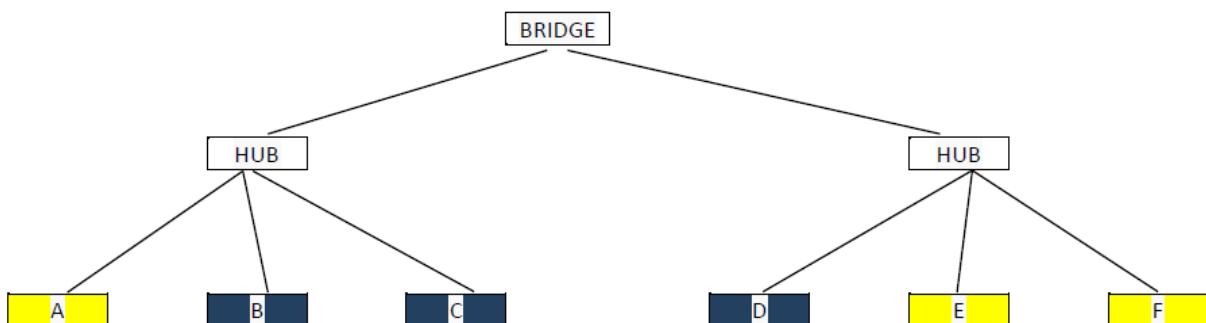
VLAN – Virtual LAN (IEEE 802.1Q)

Nei capitoli precedenti abbiamo parlato dei formati dei frame di una LAN (avevamo confrontato con lo standard HDLC del livello 2). Abbiamo poi visto che il livello 2 è suddiviso in due parti: LLC e MAC. Per quest'ultimo sottolivello, abbiamo studiato il CSMA e abbiamo visto che Ethernet è di tipo CSMA CD 1p. Abbiamo quindi visto come è possibile suddividere i domini di collisione per mezzo di apparati di rete come bridge e switch. Infine abbiamo analizzato alcune implementazioni di reti LAN come fastEthernet e Gigabit Ethernet.

Ora non ci resta che analizzare un altro tipo di reti LAN: le VLAN.

È una LAN che collega più stazioni solamente dal punto di vista funzionale, non in base al dominio di collisione o alla sottorete a cui le stazioni appartengono. Anche il cablaggio quindi può variare da stazione a stazione.

Una VLAN aggiunge sicurezza nella comunicazione, perché è possibile filtrare i frame per raggiungere solo le stazioni che appartengono funzionalmente a un determinato gruppo.



Raggruppiamo diversi nodi non in base a dove sono localizzati ma in base a motivi organizzativi:

- A, E, F appartengono ad una certa area organizzativa;
- B, C, D ad un'altra area organizzativa

Se A vuole comunicare con E non ci sono problemi.

Come facciamo a impedire la comunicazione tra A e D?

Nelle schede MAC dei bridge viene aggiunta l'informazione riguardo al **gruppo di appartenenza del nodo**, identificatore di VLAN (detto colore – MAC-VLAN compatibili) e tengo l'informazione nella tabella di forwarding del bridge che diventerà:

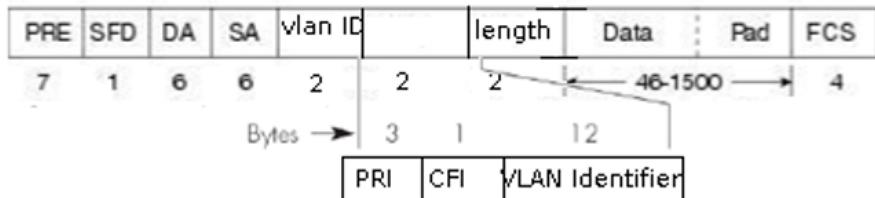
Nodo	Porta	Vlan
A	1	Yellow
B	1	Dark Blue
C	1	Dark Blue
D	2	Dark Blue
E	2	Yellow
F	2	Yellow

Anche il frame deve contenere l'informazione del gruppo di VLAN a cui appartiene il nodo.

Modifiche al formato (IEEE 802.1Q):

- Le frame dovranno includere l'informazione sul gruppo di appartenenza, e infatti ci sono delle modifiche

Il frame risulta 4 B più lungo. Non si è scelto di inglobare queste informazioni nel padding per far sì che i livelli superiori non siano costretti a sapere del cambiamento di dimensione del campo dati.



DA = Destination MAC Address

CFI: canonical format identifier - used to enable a frame relating to a token ring LAN to be embedded within the data field of the frame

SA = Source MAC Address

VLAN protocol ID = 8100 Hex Note: is greater than 1500 and hence is interpreted as a type value

VLAN identifier: this identifies the VLAN to which the frame belongs (colors)

PRI: priority field (for possible future use)

MTU (Maximum Frame Size) is now 1522 bytes

Osservazione: i frame VLAN hanno 4 byte in più, allora il padding è al massimo 42 byte.

PRI, priorità sul traffico tra VLAN

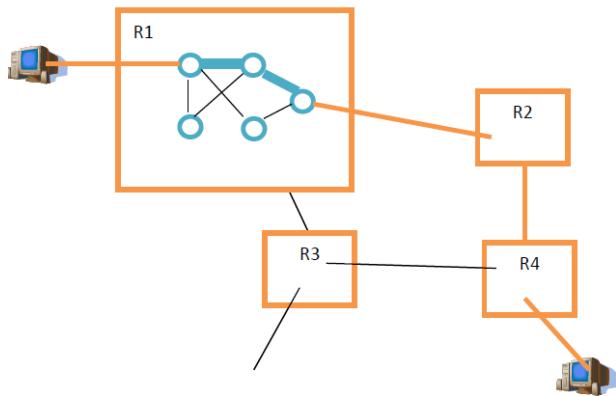
CFI, selettore formato frame per compatibilità Token ring

VLAN ID, identificatore del gruppo funzionale, è un valore maggiore di 1500 così non rischio di scambiarlo per la possibile lunghezza del dato del IEEE 802.3 (il valore è 8100, esadecimale)

- La scheda Ethernet cambia perché cambia lo standard, ma devo garantire comunque la compatibilità

- Viene aggiunto alla tabella MAC del bridge un nuovo campo, che per ogni stazione indica il gruppo di appartenenza (trucco per garantire la compatibilità è il doppio utilizzo del campo length)

Non introduce costi aggiuntivi nel processing delle frame.



È il livello che assegna un'indirizzo all' host, questo indirizzo sarà valido per tutta la rete.

Sappiamo che la topologia è magliata, e questo livello è funzionalmente strategico per collegare più nodi o più sottoreti.

Principalmente si occupa di:

- **Addressing**, (indirizzamento) devo essere in grado di identificare univocamente un nodo di una sottorete
- **Routing**, devo capire come instradare un pacchetto tra sottoreti diverse

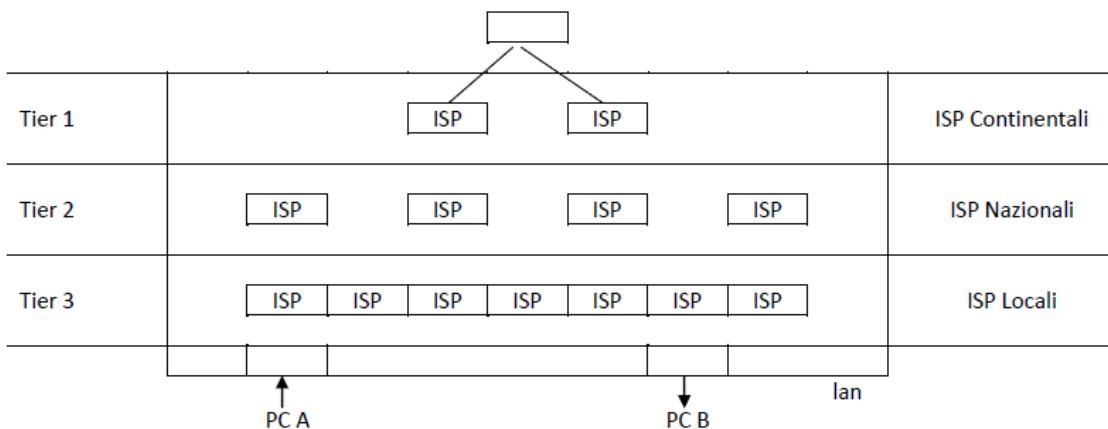
Il livello 3, come il livello 2, si divide in due livelli:

- 3a: **Rete (intra-rete)**, instrada i pacchetti all'interno della sottorete. Se la sottorete è broadcast non serve; se invece è magliata, serve per instradare al di fuori della sottorete locale
- 3b: **Inter-rete (IP – InterNetwork Protocol)**, fornisce l'indirizzo IP che è univoco e specifica i protocolli per instradare tra sottoreti. Notare che non specifica esplicitamente il protocollo da usare, ma solo le regole perché funzioni tramite gli indirizzi IP.

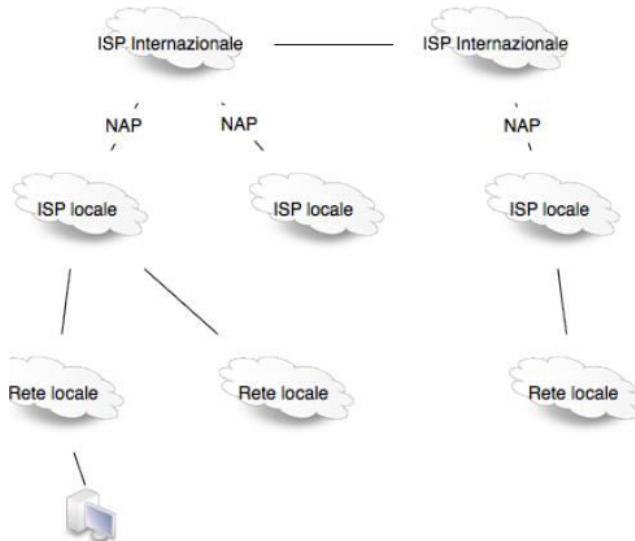
Livelli di accesso a Internet

Esistono diversi livelli (Tier1, Tier2 e Tier3) tramite i quali Internet è raggiungibile:

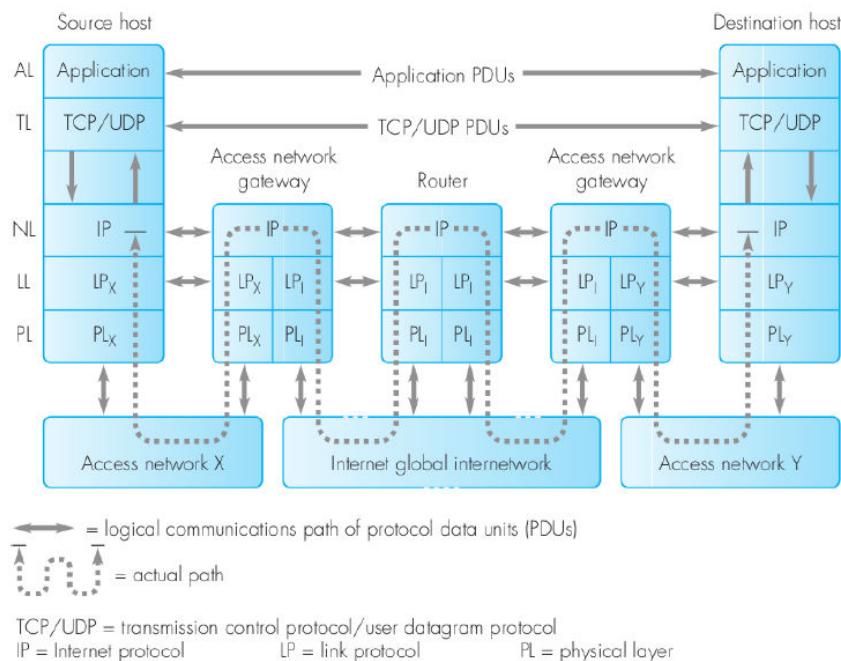
- **Livello di accesso (livello 3)**, tutte le sottoreti locali, come aziende o il nostro Dipartimento
- **Livello 2**, gli ISP nazionali o regionali uniscono le diverse sottoreti del livello di accesso
- **Livello 1**, ISP internazionali uniscono gli ISP nazionali. Uniscono cioè continenti e nazioni diverse. Sono una ventina al mondo.



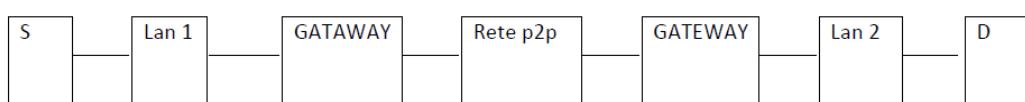
Tra il livello 2 e il livello 1 ci sono i NAP, switch a 10 Gigabit che smistano tutto il traffico nazionale.



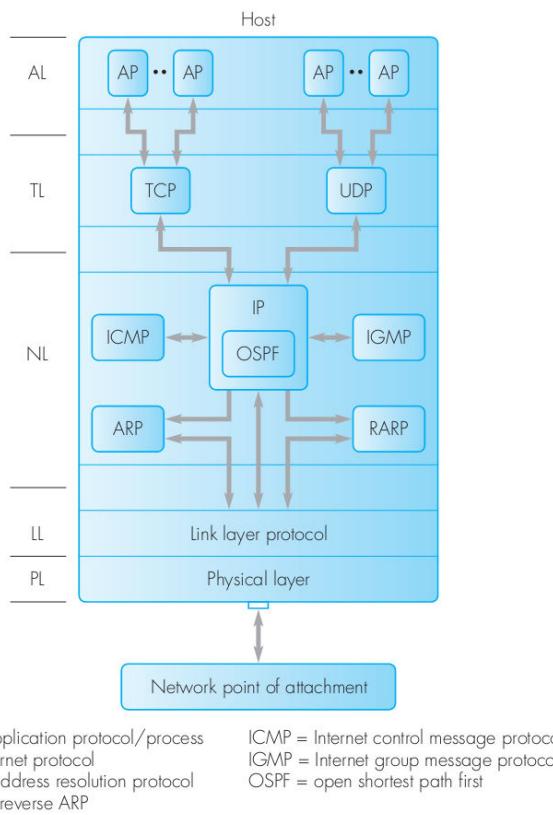
Ricordiamo il modello generale:



I gateway sono strutture che da una parte hanno protocollo CSMA/CD e dall'altra una topologia P2P



Approfondiremo il network level incontrando diversi protocolli:



Noi ci occuperemo dei seguenti protocolli :

- IP
- OSPF
- ICMP
- ARP
- RARP
- -----
- TCP (livello 4)
- UDP (livello 4)
-

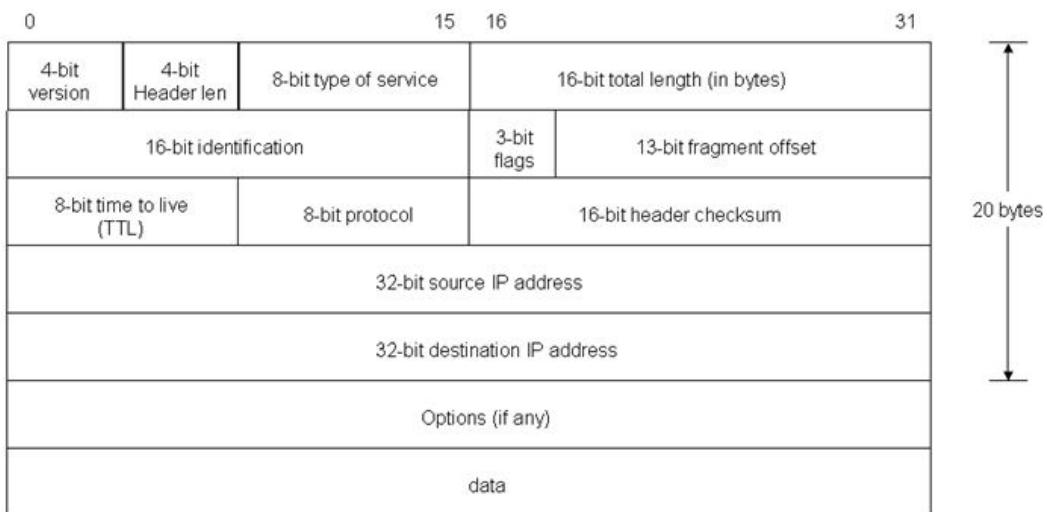
In sintesi, il ruolo del protocollo IP è ristretto :

- indirizzamento
- instradamento, inglobando uno tra i 2 possibili tipi di protocolli:
 - OSPF: instradamento livello intra e inter-rete (oggi le reti usano MPLS)
 - RIP: instradamento solo a livello intra-rete
- specifica e controlla formato dei pacchetti

IPv4 Packet

Parlamo di pacchetti e non più di frame. IP fornisce un servizio datagram, cioè non affidabile.

L'header di un pacchetto (o datagram) IP, è organizzato da 5 parole di 32 bit obbligatorie e un campo opzionale (quindi header IP = 20 B, escludendo options):



Version, specifica la versione di IP (in questo caso 4) e istruisce quindi gateway, router e host su come interpretare il pacchetto

IHL (Internet Header Length), la lunghezza dell'header può essere variabile (campo option), IHL specifica la lunghezza dell'header in multipli di 32 bit: la lunghezza minima è 5 (default), mentre la massima è 15 (il campo IHL è di 4 bit)

Type Of Service (ToS), fornisce una priorità al pacchetto IP, cosicché i router possano instradarlo nel modo migliore

Total Length, indica la lunghezza totale del pacchetto IP, comprendendo quindi sia header che payload. La dim massima è $2^{16} - 1 = 65535$ (circa 65 KB, troppo rispetto alla dimensione massima dei sotto livelli che era max 1500 B per 802.3, bisognerà frammentare!)

Parola per la frammentazione dei pacchetti, 32 bit {

Identification, identificatore del pacchetto originale, è quindi uguale in ogni pacchetto dello messaggio originale

Reserved bit

D (Don't Fragment, DF), non frammentare il pacchetto, quindi cerca di mandarlo in una rete che può farlo passare intero

M (More Fragment, MF), indica che esistono altri pacchetti che devono arrivare (1), o se questo è l'ultimo frammento (0)

Fragment offset, Indica quanti bit sono stati già inviati dai frammenti precedenti; non vengono contati i bit ma cluster di 8 byte, massimizzando così l'uso dei 13 bit (non esiste quindi un frammento più piccolo di 8 byte). Indicizza $2^{13} = 8192$ gruppi di 8 byte , quindi $(8192 \times 8) - 1 = 65536$ B (che è la maximum segment size di un segmento di livello 4)

}

Parola per controllo, 32 bit {

TTL (Time to Live), tempo massimo di vita di un pacchetto nella rete, se scaduto tale tempo e il pacchetto non è stato consegnato, esso deve essere eliminato. La durata del pacchetto all'interno della rete può essere espressa in termini di hop (tecnica più comunemente utilizzata).

Protocol Selector, indica il protocollo di livello 4 usato dal mittente, cosicché il destinatario possa passarlo allo stesso protocollo

Header Checksum, usato per efficienza operativa, poiché se la checksum è sbagliata evito subito di instradare il pacchetto, (verifica integrità solo dell'header, per questo il livello 4 ricalcolerà la crc anche per il contenuto IP)

}

- **Options**, campo variabile (ecco motivato il campo IHL) nel quale vengono specificate opzioni su: Sicurezza, Source Routing ..

L'indirizzamento è a 32 bit, quindi si ha 2^{32} indirizzi possibili (il numero non è più sufficiente, problema risolto con IPv6).

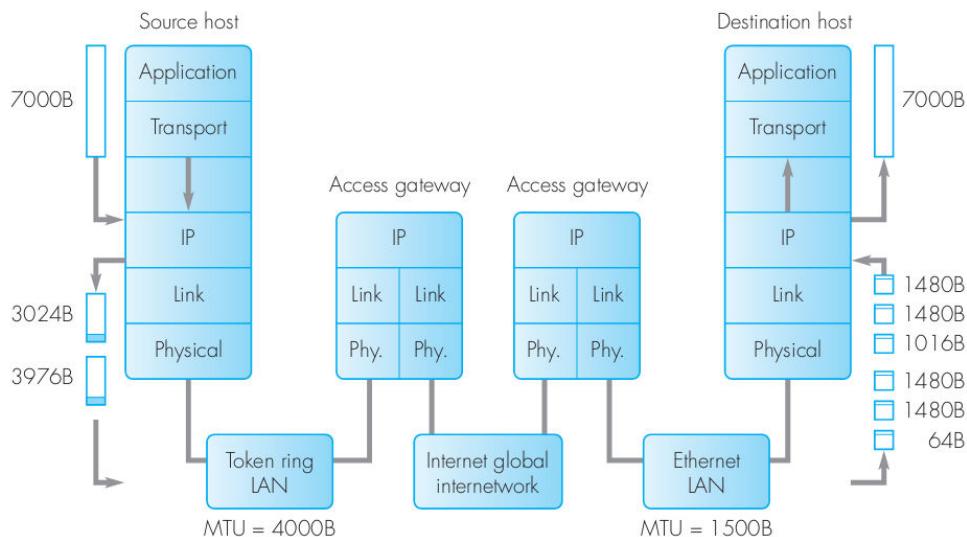
Ogni macchina connessa a Internet è indirizzabile da un indirizzo IP univoco (non esistono 2 macchine con stesso IP connesse contemporaneamente).

Frammentazione e ricostruzione di un pacchetto IP

Supponiamo di dover trasferire un pacchetto di 7000 byte (7 KB) da una LAN token ring attraverso Internet fino ad un host di una LAN Ethernet. Assumiamo che la quantità massima di trasmissione (MTU) su token ring sia di 4000 byte, mentre quella Ethernet 1500 byte. In una classica Ethernet i frame contengono da un min di 64 B fino a 1500 B di dati; il livello IP aggiunge ai dati veri e propri il proprio header di 20B.

Abbiamo quindi $4000 - 20 = 3980$ byte disponibile per il payload e dobbiamo creare frammenti con payload multipli di 8 byte. Vengono inviati quindi due frammenti, che contengono, nell'header, le informazioni per la ricostruzione (indicati in figura sotto Token ring LAN).

Quando questi due pacchetti giungono sulla rete di destinazione, devono essere ulteriormente riframmentati per poter viaggiare in Ethernet, ed infatti entrambi i pacchetti vengono divisi in altri 3 pacchetti, che contengono nell'header le informazioni per il destinatario (indicati in figura sotto Ethernet LAN).



Note: All values shown are the amounts of user data in each packet/frame in bytes

Token ring LAN:

	Frame 1	Frame 2
ID	20	20
Total Length	7000B	7000
Fragment Offset	0	497
User Data	3976	3024
M	1	0

Ethernet LAN

	Frame 1	Frame 2	Frame 3	Frame 4	Frame 5	Frame 6
ID	20	20	20	20	20	20
Total Length	7000B	7000	7000	7000	7000	7000
F O	0	185	370	497	682	867
User Data	1480	1480	1016	1480	1480	64
M	1	1	1	1	1	0

Esercizio

Il protocollo TCP consegna al protocollo IP un segmento di 3720 byte da trasferire. Considerando che il pacchetto risultante dovrà essere trasmesso attraverso una tratta di sottorete Ethernet, indicare il contenuto dei campi fragment offset e total length dei frammenti generati e specificare il numero di byte contenuto nel payload.

SOL: Fragment offset nello header IP indica la posizione dei dati contenuti nel frammento – in multipli di 8B – relativamente all'inizio del datagram. Total length indica quantità totale di dati nel datagram. 3720B producono:

- 1 pkt con 1480B nel payload, fragment offset 0, total length 3720
- 1 pkt con 1480B nel payload, fragment offset $1480/8=185$, total length 3720
- 1 pkt con $(3720-1480-1480)=760$ B nel payload, fragment offset $(1480+1480)/8=370$, total length=3720

Esercizio

Si supponga che un IP access gateway debba instradare su una LAN Ethernet tre pacchetti provenienti da una LAN in una rete non IP, contenenti rispettivamente 2875, 3862 e 1877 B di dati. Quanti frame Ethernet vengono generati, e qual è il loro contenuto di dati?

SOL: In una classica Ethernet i frame contengono da un min di 64 B fino a 1500 B di dati; il livello IP aggiunge ai dati veri e propri il proprio header di 20B. Pertanto:

- il primo pacchetto viene spezzato in 2 frame contenenti rispettivamente :
(1480 B dati + 20 B header IP);
 $(2875 - 1480) = (1395 \text{ B dati} + 20 \text{ B header IP})$
- il secondo pacchetto viene spezzato in 3 frame contenenti rispettivamente :
(1480 B dati + 20 B header IP);
 $/* 3862 - 1480 = 2382 > 1480 */$
(1480 B dati + 20 B header IP);
 $/* 2382 - 1480 = 902 */$
(902 B dati + 20 B header IP)
- il terzo pacchetto viene spezzato in 2 frame contenenti rispettivamente :
(1480 B dati + 20 B header IP);
 $(1877 - 1480) = (397 \text{ B dati} + 20 \text{ B header IP})$

Indirizzamento

Ogni stazione possiede un indirizzo IP, che è uno tra i 2^{32} possibili. Questi indirizzi erano, inizialmente, divisi in 4 classi:

- classe A

8 bit per identificare la rete, 24 per gli host

Non tutti e 8 i bit per la rete sono utilizzati perché il bit più significativo è settato a 0

Gli indirizzi IP di classe A vanno quindi da 1.0.0.0 a 127.255.255.255

- classe B

16 bit per la rete, 16 per gli host

Non tutti e 16 i bit per la rete sono utilizzati perché i due bit più significativi sono settati a 10

Gli indirizzi IP di classe B vanno quindi da 128.0.0.0 a 191.255.255.255

- classe C

24 bit per la rete, 8 per gli host

Non tutti e 24 i bit per la rete sono utilizzati perché i tre bit più significativi sono settati a 110

Gli indirizzi IP di classe C vanno quindi da 192.0.0.0 a 223.255.255.255

- classe D

È utilizzata per il multicast

- classe E

È utilizzata per esperimenti

Come si vede dalla figura che segue, i primi bit sono selettori della classe:

se il primo bit è 0 siamo nella classe A

se il primo bit è 1 :

 se il secondo bit è 0 siamo nella classe B

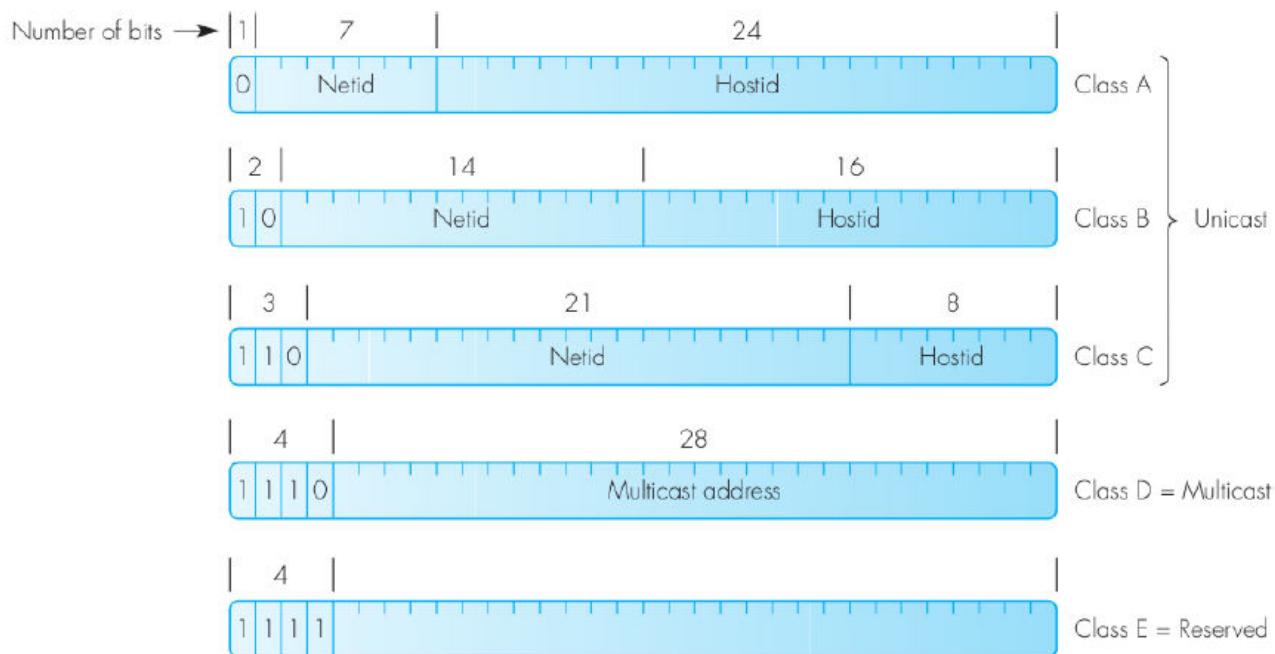
 se il secondo bit è 1 :

 se il terzo bit è 0 siamo nella classe C

 se il terzo bit è 1 siamo in altre classi che noi non consideriamo

Il secondo gruppo di bit identifica la rete

Il terzo gruppo di bit identifica l'host



Esercizio

Convertire l'indirizzo IPv4 esadecimale C22F1582 alla dotted notation

SOL: Si considerano le cifre esadecimali a coppie: max FF = $16 \times 15 + 15 = 255$

$$C2 = 16 \times 12 + 2 = 194;$$

$$2F = 16 \times 2 + 15 = 47;$$

$$15 = 16 \times 1 + 5 = 21;$$

$$82 = 16 \times 8 + 2 = 130 .$$

Quindi 194.47.21.130

NB. Conversione da esadecimale a decimale

$$\begin{array}{c} EA_{(16)} \\ \searrow \quad \swarrow \\ Ex16^1 + Ax16^0 = \\ 14 \times 16 + 10 \times 1 = \\ 224 + 10 = \\ 234_{(10)} \end{array}$$

Ricorda A=10

Esercizio

Trovare l'IPv4 address che corrisponde ai seguenti 32 bit: C0201010

SOL 192.32.16.16

Esercizio

Considera l'IPv4 address FF051610 e trasformalo nella notazione abitualmente usata.

SOL In IP i network address sono numeri di 32 bit abitualmente scritti in notazione decimale con uso del punto come separatore. L'IP address specificato corrisponde in notazione decimale all'indirizzo 255.5.22.16 e non corrisponde ad un indirizzo attualmente usato in Internet in quanto il range di numerazione superiore a 240.0.0.0 è lasciato per usi futuri e/o particolari.

Tabelle di routing

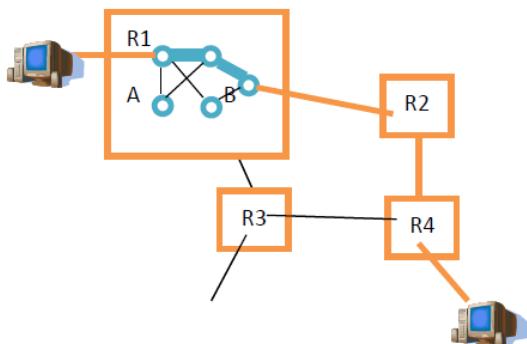
Tiene traccia di tutte le possibili "strade" per far arrivare a destinazione il pacchetto.

Nel caso ad esempio della classe C sono 2^9 possibili informazioni da memorizzare. Sono troppe!

Quindi il router tiene in memoria solo gli indirizzi della sua sottorete; nel caso arrivi un pacchetto per un'altra rete, lo affida ad un router di livello superiore.

Esempio

Nella figura A conosce solo gli indirizzi della sottorete R1. Se deve inviare un pacchetto ad un nodo in R4, lo affida a B.



Subnetting

Ogni router di una sottorete dovrebbe quindi avere una entry per ogni host della sottorete, e per la classe A o B diventerebbe oneroso. Si introduce allora il concetto di subnetting, cioè una tecnica di organizzazione degli indirizzi all'interno di una singola sottorete.

Classe B

Nell'indirizzare gli Host ID, formati da 2^{16} bit, viene introdotto il subnet, cioè un ulteriore livello gerarchico per dividere in "zone" la rete (è solo una questione organizzativa, non centra con sicurezza o altri fattori).

-senza subnet

Arriva al router del nodo A un pacchetto:

Se NetID non corrisponde con la rete di A, mando pacchetto verso la rete giusta

Se Net corrisponde allora rintraccio host a cui consegnare il pacchetto

- con subnet

Arriva al router del nodo A un pacchetto:

se net non corrisponde con la rete di A, mando pacchetto verso la rete giusta

se net corrisponde, allora effettuo un ulteriore controllo:

- se subnet non corrisponde con la subnet di A, mando pacchetto verso la sottorete giusta
- se subnet corrisponde, allora rintraccio host a cui consegnare il pacchetto

Invece di dividere l'indirizzo IP solo in NetID e HostID, aggiungiamo anche un subnetID. Il router principale dovrà quindi confrontare solo il netID, poi indirizzerà il pacchetto al router dedicato alla Subnet indicata nel subnetID. Esso indirizzerà poi al singolo host della sua porzione di rete.

Il SubnetID è sempre ricavato da una porzione dell'indirizzo IP dedicata agli host, scelta a piacere (è una scelta interna alla mia sottorete).

Come capisco quanti bit sono stati assegnati alla subnet? Chi gestisce il router deve fornire ad esso una maschera.

Per capire a quale sottorete deve essere indirizzato il pacchetto, viene fatto un **and** logico con la **subnet mask**, formata da tutti 1 nella porzione del NetID e quella scelta per il SubnetID, e 0 altrove.

Esempio

Arriva indirizzo 130.50.15.6

Che il router interpreta:

net: 130.15 (8bit+8bit)

130 = 10000010, i primi bit sono la sequenza 10, quindi siamo nella classe B, avremo 16 bit di net_id e 16 bit di host_id
50 = 00110010

Host: 15.6 (8bit+8bit)

15 = 00001111

6 = 00000110

E quindi l'indirizzo binario sarà:

Net	Host
1 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0	0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 0

Se però nell'indirizzo è compreso anche l'indicazione di una subnet di 2^{16} possibili valori, esso sarà così composto:

Net	Subnet	Host
1 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0	0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 0	

Che indica:

- indirizzo di rete: 100001000110010 = 16946
- indirizzo subnet: 000011 = 3
- indirizzo host: 110000110 = 390

La maschera sarà composta da una stringa binaria di 1 per i 16 bit del net e i 6 bit della subnet e da 0 nei 10 bit dell'host:

Net	Subnet	Host
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		

Che messi in AND danno:

1 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 0 AND		
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	=	
1 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 16946 3 0		

Conosco quindi l'indirizzo della rete e della sottorete, poi recupero l'indirizzo dell'host.

Nb. L'indirizzo 130.50.15.6 si trova più facilmente scritto nel formato CIDR in cui vengono specificati quanti bit a 1 sono presenti per la maschera: 130.50.12.0/22

Esercizio

Una rete di classe B ha subnet mask 255.255.240.0 Qual è il max# host per subnet?

SOL: $240 = 128 + 64 + 32 + 16 \rightarrow 11110000$ perciò ho $4+8=12$ bit per host ID, pertanto $2^{12}-1=4095$ host ID possibili.

Esercizio

Un amministratore di rete deve gestire una campus LAN a cui è assegnato l'indirizzo di classe B 150.10.0.0. Assumendo che la rete comprenda 100 sottoreti ognuna delle quali connessa ad uno switch FastEthernet usando un router, definire una appropriata subnet mask se il numero massimo di host connessi ad ogni subnet è 70.

SOL: Per assegnare un netID ad almeno 100 subnet servono 7 bit (127 subnet).

Per assegnare un hostID ad almeno 70 host servono 7 bit. ($2^6=64$ troppo pochi!)

Quindi qualsiasi suddivisione che soddisfi tali vincoli è appropriata:

(7 bit subnet + 9 bit hostID); oppure

(8 bit subnet + 8 bit hostID); oppure

(9 bit subnet + 7 bit hostID).

NAT, CIDR, IPv6

Nate principalmente per due ragioni:

- la paura che gli indirizzi IPv4 non siano sufficienti a coprire tutte le richieste, ha portato a cercare nuovi metodi per indirizzare la singola macchina (paura incoraggiata anche dal fatto che gli indirizzi IPv4 sono divisi in classi; alcuni indirizzi vengono assegnati a qualche ente, che magari non li usa tutti e quindi vanno sprecati);

- Far gestire gli indirizzi assegnati ad una organizzazione è scomodo, meglio farli gestire all'associazione per indirizzi privati che può sceglierli liberamente

NAT

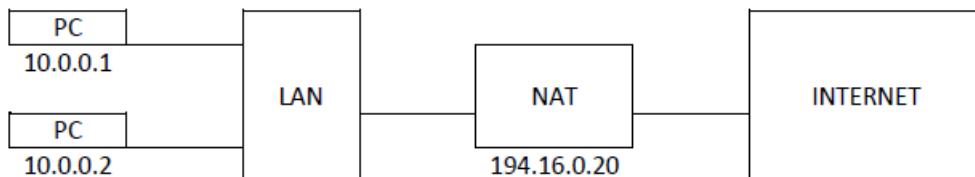
Network Address Translation (o PAT – Port Address Translation)

All'interno di una rete privata, connessa con un router ad Internet, ogni computer deve essere associato a un indirizzo IP per poter essere indirizzabile. Assegnare un diverso indirizzo IP ad ogni apparato di ogni sottorete è impensabile e molto scomodo, quindi IP fornisce la possibilità di assegnare degli indirizzi IP privati che sono però univoci solo all'interno della propria rete.

Gli indirizzi IP privati sono indirizzi statici, associati ad un insieme di macchine facenti parte di uno stesso gruppo, che violano il vincolo di unicità globale per ogni macchina connessa alla rete.

"Non è più possibile parlare da Internet ai miei computer?"

Il problema viene risolto dai router, che trovandosi funzionalmente nel mezzo tra i computer privati e Internet, forniscono un unico indirizzo IP pubblico. In questo modo, il **router con funzionalità NAT** parla con Internet da una lato e con tutti i computer della LAN dall'altro. Tutte le risposte agli host della LAN passano dal router, che dovrà quindi provvedere a indirizzarle all'host destinatario.



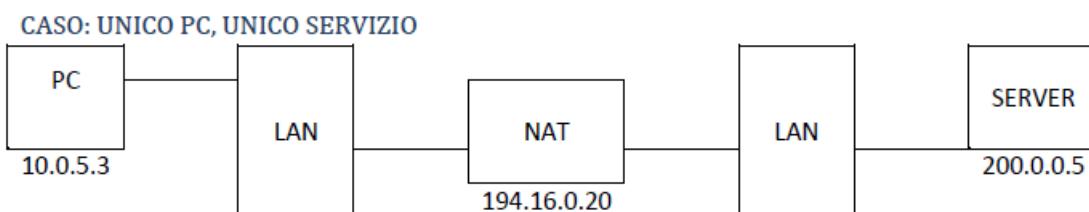
NAT ha un IP univoco, e ad esso posso collegare più macchine che possiederanno un IP privato.

E se volessi comunicare con un servizio?

1- se un host con IP privato 10.0.0.1 manda un pacchetto a 200.15.0.8 (server all'esterno della LAN), NAT cambia sorgente da 10.0.0.1 a 193.16.0.1 (che è un indirizzo IP pubblico associato al router). Il router dovrà quindi contenere una tabella dove è indicato che 10.0.0.1 sta comunicando con 200.15.0.8 attraverso l'IP pubblico 193.16.0.1 . E' necessario specificare anche l'indirizzo IP pubblico perché NAT consente anche di usare un pool di indirizzi pubblici, e non uno solo.

2- In risposta, sempre NAT , cambia la destinazione da 193.16.0.1 a 10.0.0.1 usando la entry della tabella creata.

Esempio

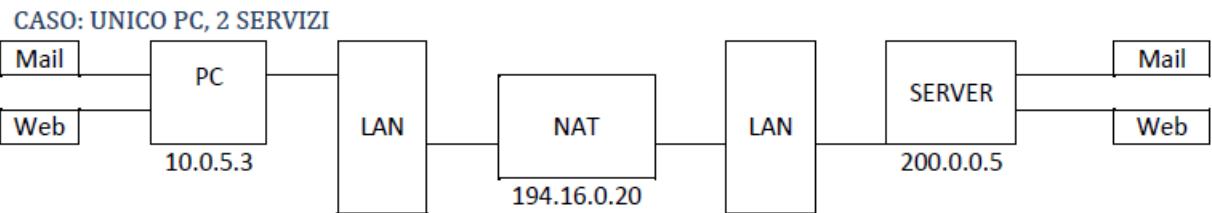


Se il PC vuole comunicare con il server sarà sufficiente inviare il pacchetto all'indirizzo del server. Il server però non vede e non può sapere l'indirizzo del PC e quindi il destinatario del pacchetto di ritorno corrisponderà all'indirizzo NAT. Nella tabella NAT dovrà quindi esserci salvato che il messaggio proveniente dal server è destinato al PC.

IP interno	IP esterno	IP Destinazione
10.0.5.3	194.16.0.20	200.0.0.5

C'è però un problema, se più stazioni *local* (interne al NAT) comunicano con la stessa stazione *global* , come si fa a indirizzare le risposte del server alle giuste macchine? Il NAT non può basarsi solo sull'indirizzo IP, ma deve basarsi su un'informazione aggiuntiva: la **porta**. Come IP distingue una macchina in Internet, la porta distingue il processo sulla singola macchina. La coppia IP:porta (chiamata socket) mi rende quindi univoco il processo della macchina.

Esempio



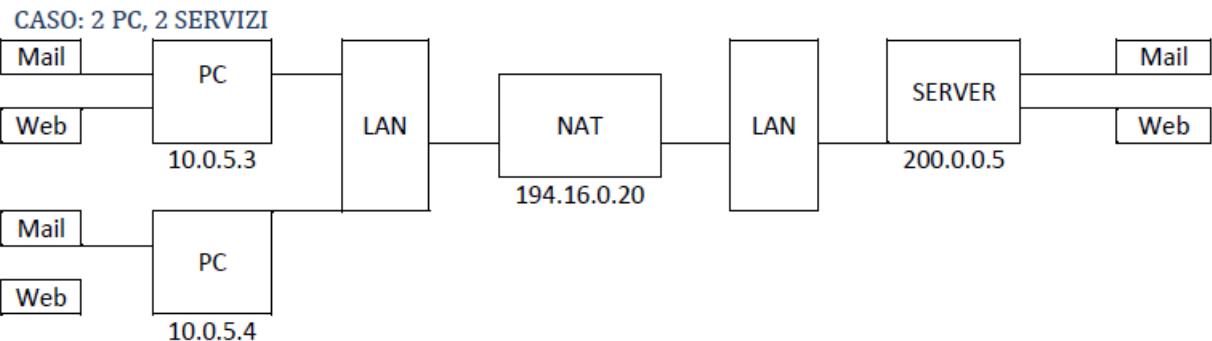
Ora sia il PC che il server svolgono servizio sia web che mail. Quando il PC manda un pacchetto mail o web, a seconda del servizio richiesto, viene imposto dal livello 4 di mandarlo a una determinata porta del server. Nei server le porte sono standard (a meno che le si cambi di proposito): ad esempio, il servizio mail risponde sempre alla porta 125 e il servizio web alla porta 80.

Quando il mail server risponde e il messaggio arriva al NAT, questo deve sapere a quale porta del PC deve essere indirizzato il messaggio.

Notare che, il NAT, è dunque uno strumento di livello 3 con funzioni di livello 4.

# porta In	IP interno	IP esterno	IP Destinazione	# porta out
1	10.0.5.3	194.16.0.20	200.0.0.5	Mail
2	10.0.5.3	194.16.0.20	200.0.0.5	Web

Esempio

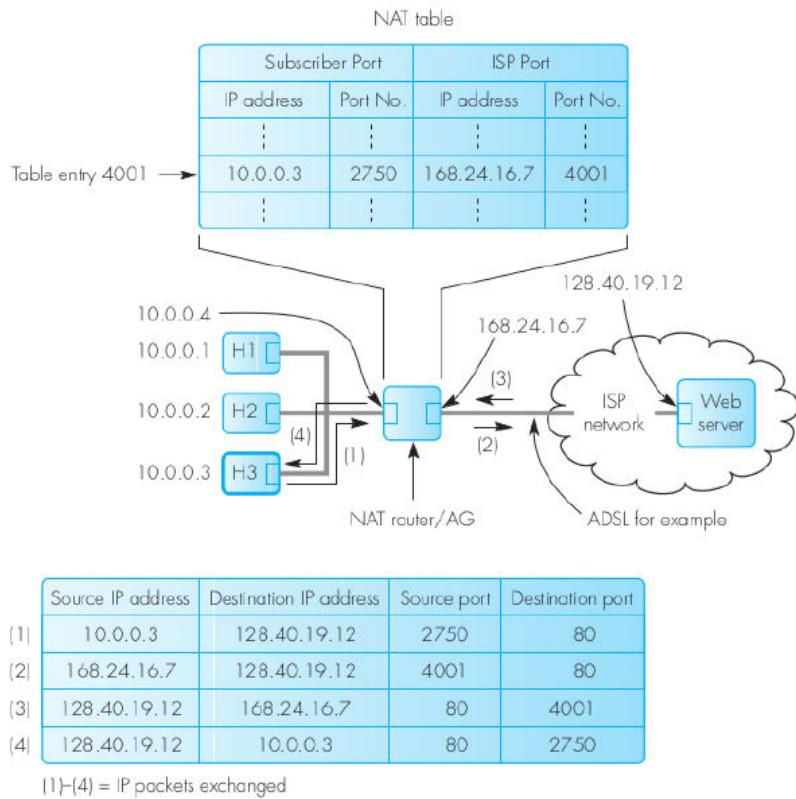


Se ora 2 PC chiedessero di comunicare col server mail, la risposta non potrebbe più essere come nel caso precedente perché il NAT non riuscirebbe a capire a quale dei due PC deve andare il messaggio arrivato. Viene così salvato nella tabella, per ogni messaggio proveniente dai PC, una porta destinazione diversa, anche se sono entrambi messaggi dello stesso servizio server.

# porta In	IP interno	IP esterno	IP Destinazione	# porta out
1	10.0.5.3	194.16.0.20	200.0.0.5	3600
2	10.0.5.3	194.16.0.20	200.0.0.5	3601
1	10.0.5.4	194.16.0.20	200.0.0.5	3602
2	10.0.5.4	194.16.0.20	200.0.0.5	3603

Se volessi creare un server all'interno della NAT, avrei 2 possibilità affinché la NAT non nasconde il server e gli utenti esterni possano comunicare con esso:

- assegno una porta e comunico ad ogni utente la porta
- collego il server al DMZ (un collegamento che possiedono le NAT più evolute, dove non avviene il mascheramento). Verrà dunque assegnato al server un indirizzo IP proprio.



Con le porte, NAT è robusto:

- Ogni host definisce una porta su cui comunicare, il NAT le registra e può così indirizzare tutti i pacchetti verso la giusta destinazione. Anche se due host scelgono la stessa porta, il NAT le cambia a loro insaputa per evitare conflitti.
- Una tabella NAT o PAT conterrà quindi i campi:
 - **#PortaIn**, porta dell'host local (SourcePort)
 - **IP interno**, indirizzo IP privato dell'host
 - **IP esterno**, IP pubblico di riconoscimento del NAT per instradare i pacchetti dell'host local verso la rete Internet
 - **NAT-Port**, porta assegnata dal NAT (sostituisce SourcePort se quest'ultima è già occupata, è necessaria univocità)
 - **IP destinazione**, indirizzo IP pubblico esterno (es. server)
 - **#PortaOut**, porta dell'host global (dalla well-known viene fatta fork server, necessaria univocità della porta anche in out)

NAT quindi aiuta a risparmiare indirizzi IP.

CIDR – Classless Inter Domain Routing

CIDR propone un altro metodo di indirizzamento, rispetto a quello a classi. “*Sarebbe meglio, invece di assegnare ad un'azienda tutta una porzione di classe, assegnare circa la quantità di indirizzi IP che servono.*”

Verrà assegnato un blocco di indirizzi contigui ad ogni richiesta, considerando quindi anche il caso di possibile espansione (il totale non sarà necessariamente contiguo). Quando devo capire di che sottorete è quel pacchetto, faccio and logico tra la maschera e il dest address per ogni associazione che ha richiesto indirizzi CIDR; quando ricavo una base di un'associazione conosciuta, inoltro il pacchetto. La notazione CIDR è nella forma a.b.c.d/x dove x è il numero di 1 nella subnet mask. I rimanenti $y=(32-x)$ bit permettono di calcolare il numero di host nella sottorete, pari a 2^{y-2} . Il -2 è dovuto al fatto che due indirizzi particolari, il primo e l'ultimo, sono riservati rispettivamente all'indirizzo di rete e l'indirizzo di broadcast.

Esercizio

In un sistema che utilizza CIDR, le tabelle di routing hanno – come entry relativa alla ditta Acme Inc. – i dati: 148.125.16.0, 255.255.240.0.

Dire qual è il numero massimo di host che possono essere connessi alla rete di Acme, e quale tra le alternative seguenti indica l'indirizzo di un host nella rete:

- 10010100.01111101.00010110.10011100
- 10010101.01101001.00101001.01011011
- 11001011.00111100.00011011.10000111
- 10111001.01000111.01001100.10001001

SOL: I bit riservati per host identifier sono l'ultimo ottetto, e gli ultimi 4 bit del terzo ottetto (da 240 = 11110000 in binario). Con $8+4=12$ bit si possono indirizzare $2^{12} = 4096$ host. La rappresentazione in binario dell'indirizzo base è: 148=10010100 ; 125=01111101 ; 16=00010000. Quindi l'unico possibile indirizzo valido per un host è il primo mostrato: 10010100.01111101.00010110.10011100

Esercizio

L'Università di Bologna si fa assegnare le reti da 194.76.16._ a 194.76.23._ Calcolare la maschera usata per fare CIDR.

SOL:

$$194 = 128 + 64 + 2 ;$$

$$76 = 64 + 8 + 4 ;$$

$$16 = 16$$

Quindi $\Rightarrow 11000010 . 01001100 . 00010000 . _$ [ultimo ottetto non ci interessa]

Per calcolare la maschera devo sapere quante reti ci sono tra 16 e 23.

Sono 7 reti

$$255 - 7 = 248;$$

$$248 = 128 + 64 + 32 + 16 + 8 \Rightarrow 11111000 ;$$

mask: 11111111.11111111.1111000.00000000

Ineffettuando l'ultimo ottetto di ogni rete del gruppo, si nota che differiscono per gli ultimi 3 bit

23 2	r=1	↑
11 2	r=1	
5 2	r=1	
2 2	r=0	
1 2	r=1	

$$23 = 00010111$$

$$16 = 00010000$$

Esercizio

[Ex.6.3] Ad una rete è stato allocato un blocco di 1024 indirizzi di rete da 200.30.0.0 a 200.30.3.255. Assumendo che sia usato CIDR, rappresentare questi indirizzi in forma binaria e derivare la netmask da usare – in dotted notation – e il netID di questa rete.

SOL: $200.30.0.0 \Rightarrow 200 = 128 + 64 + 8 \Rightarrow 11001000.00011110.[0]^8.[0]^8$

$200.30.3.255 \Rightarrow \Rightarrow 11001000.00011110.00000011.11111111$

Netmask deve annullare la differenza tra le due, quindi è : $[1]^8.[1]^8.11111100.[0]^8$

Ovvero: 255.255.[128+64+32+16+8+4]=252.0

(252 è il risultato di $2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2$, mancano 2^1 e 2^0 che sono lasciati per l'host_id , si poteva calcolare anche facendo $[2^8]=256 - [2^1+2^0=]3 - 1 = 252$) perché i numeri vanno da 0 a 255!

Mentre **netID** è l'indirizzo base 200.30.0.0

Esercizio

Supponiamo di voler convertire nel sistema binario il numero decimale 47:

$$\begin{array}{r}
 47 : \quad \text{con Resto} = 1 \\
 2 = \\
 \hline
 23 : \quad \text{con R} = 1 \\
 2 = \\
 \hline
 11 : \quad \text{con R} = 1 \\
 2 = \\
 \hline
 5 : \quad \text{con R} = 1 \\
 2 = \\
 \hline
 2 : \quad \text{con R} = 0 \\
 2 = \\
 \hline
 1 \quad \quad \quad \text{R} = 1
 \end{array}$$

verso di lettura $\implies (47)_{10} = (101111)_2$

ARP – Address Resolution Protocol

Per poter comporre pacchetti di livello 2 e quindi poter parlare ad un host della stessa sottorete, devo conoscere due indirizzi: l'indirizzo IP privato e l'indirizzo MAC.

Il protocollo ARP, dato un indirizzo IP, fornisce il corrispondente indirizzo MAC.

Sono previsti due tipi di messaggi dal protocollo ARP:

- ARP Request
- ARP Reply

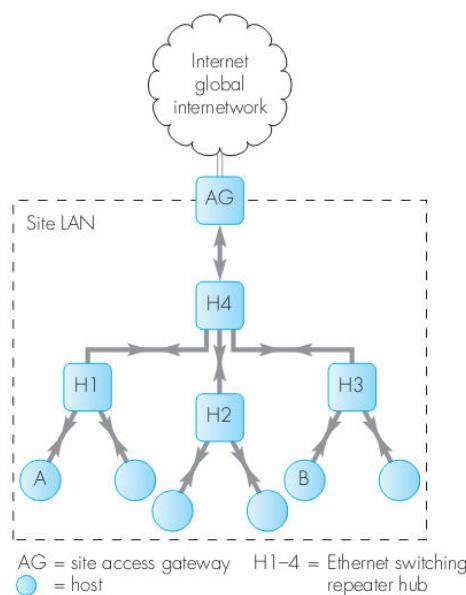
Un ipotetico host 192.168.1.1 che vuole comunicare con l'host 192.168.1.2 manderà una ARP request (contenente il proprio MAC, il proprio indirizzo IP e l'indirizzo IP di destinazione) a tutti gli host della rete; quando 192.168.1.2 riceverà l'ARP request, risponderà con un ARP reply destinato al MAC sorgente e contenente il proprio MAC.

Nota: a livello 3 si ragiona attraverso gli IP address, eppure mi servono gli indirizzi MAC per trovare fisicamente la macchina.

Il protocollo ARP svolge un *matching* tra IP e indirizzi MAC e attraverso una tabella crea la corrispondenza IP – MAC.

Per ottimizzare le prestazioni e limitare il traffico, le associazioni IP/indirizzo MAC di ogni host con cui si viene in contatto tramite *request*, vengono memorizzate nella tabella ARP (ARP cache) di ciascun host cosicché non sia necessario effettuare continue richieste per eventuali successivi indirizzamenti verso terminali già noti.

Sempre per questo motivo il gateway prende nota di tutte le coppie che passano sulla rete, anche se non sono messaggi ARP indirizzati a lui.

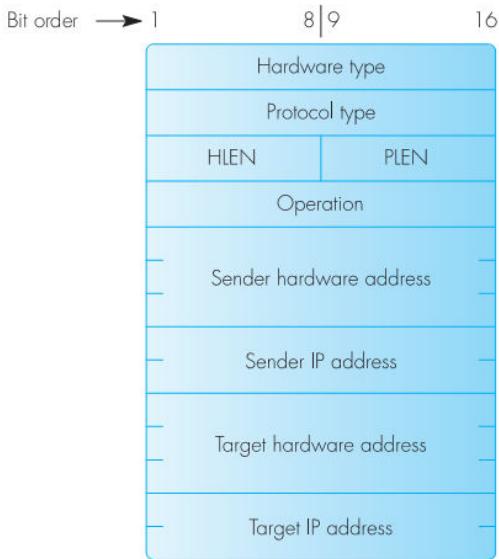


Nel caso A voglia parlare con un nodo presente in un'altra rete, il gateway è dotato di un proxy ARP che si sostituisce a B (A crederà di parlare con B). Il messaggio arriverà a B che poi si occuperà di inviarlo in modo corretto.

AG capisce che A vuole parlare con un nodo esterno semplicemente facendo un matching del net_id dell'indirizzo IP del nodo destinazione.

Formato protocollo ARP

Dimensione = 14 x 2 B = 28 B



- **Hardware type**, specifica il tipo di interfaccia hw su cui l'utente cerca una risposta (in Ethernet si setta il campo ad 1).
- **Protocol type**, indica il tipo di indirizzo ad alto livello che il mittente ha fornito (per IP si setta a 0x0800)
- **Hardware Len e Protocol Len**, consentono di usare ARP su reti arbitrarie perché specificano la lunghezza dell'indirizzo hardware (MAC) e dell'indirizzo del protocollo di alto livello (IP)
- **ARP Operation**, specifica se si tratta di una richiesta ARP (valore 1), risposta ARP (valore 2), richiesta RARP (valore 3) oppure una risposta RARP (valore 4)

Nell'ARP request il campo **target IP address** non è utilizzato (rimane vuoto) in quanto è proprio quello che si vuole ottenere con l'ARP reply.

ARP nei protocolli di livello 2



Type = frame type = 0800 (hex) = IPv4 datagram
 0806 = ARP request/reply message
 8035 = RARP request/reply message

Il pacchetto di livello 3 viaggia nel campo dati del frame ethernet IEEE802.3

Riservati per il dato erano 46 B:

- 6 per LLC
- 2 per Type, serve per specificare quale protocollo utilizza questo messaggio (come protocol selector in Ipv4)
- 28 per messaggio ARP
- 10 padding

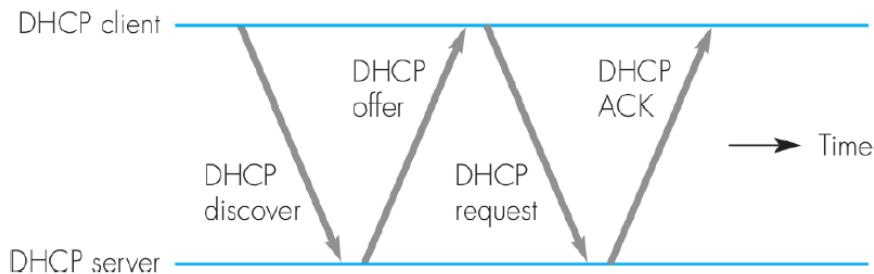
RARP – Reverse ARP

Normalmente ogni stazione conosce la sua coppia IP/MAC. Se però pensiamo a host diskless, può esistere il caso nel quale un host conosce il suo MAC, ma non il suo indirizzo IP. Ecco dove il protocollo RARP entra in azione: viene predisposto un server RARP, che possiede una tabella che contiene tutte le coppie MAC/IP degli host diskless. A questo punto, un host potrà effettuare una RARP request che verrà intercettata dal server, quest'ultimo potrà rispondere con una RARP reply contenente l'indirizzo IP dell'host diskless, oltre alla sua coppia IP/MAC utile per l'utilizzo del normale protocollo ARP.

DHCP – Dynamic Host Configuration Protocol

È un protocollo utilizzato per richiedere a un server DHCP un indirizzo IP dinamico.

Come sappiamo, per poter comunicare, un host deve possedere un indirizzo IP. Con DHCP la ricerca e l'acquisizione di questo indirizzo ricade proprio sull'host che lo richiede. Come definito dall'RFC 2131,2132, DHCP è un semplice protocollo client-server, basato su quattro messaggi:



Nel dettaglio rappresentano:

• DHCP Discover

Il client invia a tutti i server una richiesta per ottenere un IP dinamico.

Il client non possiede un indirizzo IP, quindi invia come indirizzo sorgente 0.0.0.0 (che identifica la propria macchina), mentre l'indirizzo destinazione è 255.255.255.255 (genera messaggio broadcast, viene inviato a tutti i server);

Questo messaggio contiene un **transaction ID**, che ha il compito di identificare univocamente la richiesta (è generato dal valore del clock e dall'indirizzo MAC dell'host source) cosicché il server possa associarne le risposte inviate al client

• DHCP Offer

È la risposta del server. Contiene:

- lo stesso *transaction ID* della discover del client
- l'*indirizzo IP proposto* dal server per il client,
- la *Subnet Mask*
- il tempo durante il quale il client potrà utilizzare questo indirizzo, chiamato *tempo di lease*.

ICMP Check: ogni server sceglie un indirizzo IP da offrire ma deve verificare che non sia già in uso, questa verifica viene fatta dal server inviando un messaggio a quell'indirizzo, se si riceve una risposta, l'indirizzo dovrà essere cambiato.

• DHCP Request

Il Client sceglie uno tra gli indirizzi proposti dai server e manda un messaggio a tutti i server.

La risposta all'offerta del server, contiene gli stessi parametri e serve ad accettare/rifiutare l'indirizzo e i parametri proposti

Gli indirizzi IP offerti e scartati possono essere dati a qualche altro host.

L'indirizzo IP che viene accettato non potrà più essere assegnato a nessun altro host per tutto il tempo di lease.

• DHCP Ack

Il server "scelto" manda un ACK di conferma.

Contenente sempre gli stessi parametri di Offer e Request, valida il client.

Alcune casistiche:

Se dopo un certo tempo che ho mandato *discover* non ricevo nessuna offerta, lo rimando perché il messaggio potrebbe perdersi.

Se dopo il *request* il server ha già dato l'indirizzo IP a un altro host, e quindi non può convalidarlo, al posto dell'ACK, viene mandato un segnale di NACK. L'host, a quel punto, o sceglie altre offerte da altri server, oppure rimanda *discover* a tutti.

Se all'host che ha fatto *request* non arriva ne ACK ne NACK, dopo un certo tempo rimanda *request* per un po di volte. Se non ha ancora ricevuto risposta, o cambia offerta oppure ricomincia dal *discover*.

ICMP – Internet Control Message Protocol

Protocollo di livello 3 con la funzionalità puramente di controllo.

ICMP, definito nell'RFC 1256, è usato da hosts, routers e gateways specialmente per la gestione della rete, ma fornisce un'ampia varietà di funzioni. Le principali sono:

- Rilevare gli errori
- Rilevare congestione dei nodi
- Verificare raggiungibilità dei nodi (ping IP è best effort)
- Notifica di cambio percorso (se un nodo è congestionato, manda *source quence* chiedendo di rallentare il traffico, o direttamente alla sorgente o hop per hop)
- Indirizzo della sottorete
- Misurare prestazioni dei link

Function	ICMP message	Use
Error reporting	Destination unreachable	A datagram has been discarded for the reason specified in the message
	Time exceeded	Time-to-live parameter in a datagram expired and hence datagram discarded
	Parameter error	A parameter in the header of a datagram is unrecognizable
Reachability testing	Echo request/reply	Checks the reachability of a specified host or gateway
Congestion control	Source quench	Requests a host to reduce the rate at which datagrams are sent
Route exchange	Redirect	Used by a gateway to inform a host attached to one of its networks to use an alternative gateway on the same network for forwarding datagrams to a specific destination
Performance measuring	Time-stamp request/reply	Determines the transit delay between two hosts
Subnet addressing	Address mask request/reply	Used by a host to determine the address mask associated with a subnet

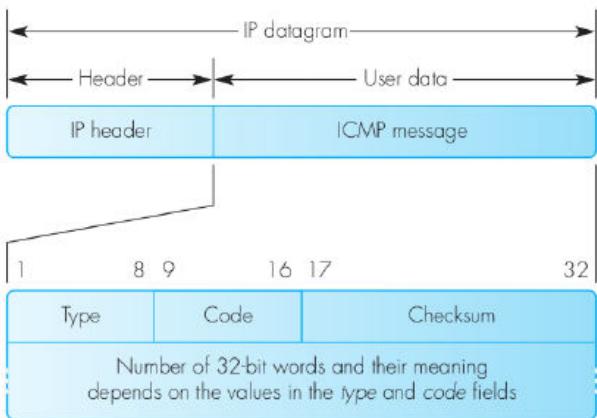
Questo protocollo, in base a ciò che rileva, può modificare le tabelle di routing.

Se tramite ICMP viene rilevata una congestione del nodo, posso:

- avvisare i nodi che mi precedono di bufferizzare in uscita per rallentare il mio lavoro (occorrerebbe fare attenzione che i nodi a loro volta non congestionino la rete)
- avvisare il nodo sorgente di non mandarmi troppi messaggi

Struttura ICMP

ICMP è contenuto nell'IP datagram e possiede l'header IP:



- *Type*, indica il tipo di controllo che si vuole effettuare (specificati nella tabella sopra)
- *Code*, indica le opzionali informazioni aggiuntive, per esempio il motivo per il quale una stazione risulta irraggiungibile

Instradamento

Le funzionalità principali dei Router sono:

- controllo (ICMP)
- Forwarding (tabella di instradamento)

Per ogni IP address indirizzabile e raggiungibile in rete è indicata la porta d'uscita con la quale raggiungo il nodo con quell'indirizzo, attraversando il cammino migliore.

In realtà, si tiene nota degli IP dei nodi vicini; gli IP non noti vengono gestiti da altri router candidati
La strada migliore può cambiare nel tempo

Def (Routing): funzionalità che riempie la tabella di instradamento e la tiene aggiornata

Def (Forwarding): processo che fa passare un pacchetto da una coda di ingresso ad una coda di uscita.

Forwarding e Routing sono asincroni.

Abbiamo due modi (algoritmi) per fare routing:

- Distance Vector
- Link-State Shortest Path First

E per la realizzazione di questi algoritmi è stato implementato un preciso protocollo:

- RIP
- OSPF

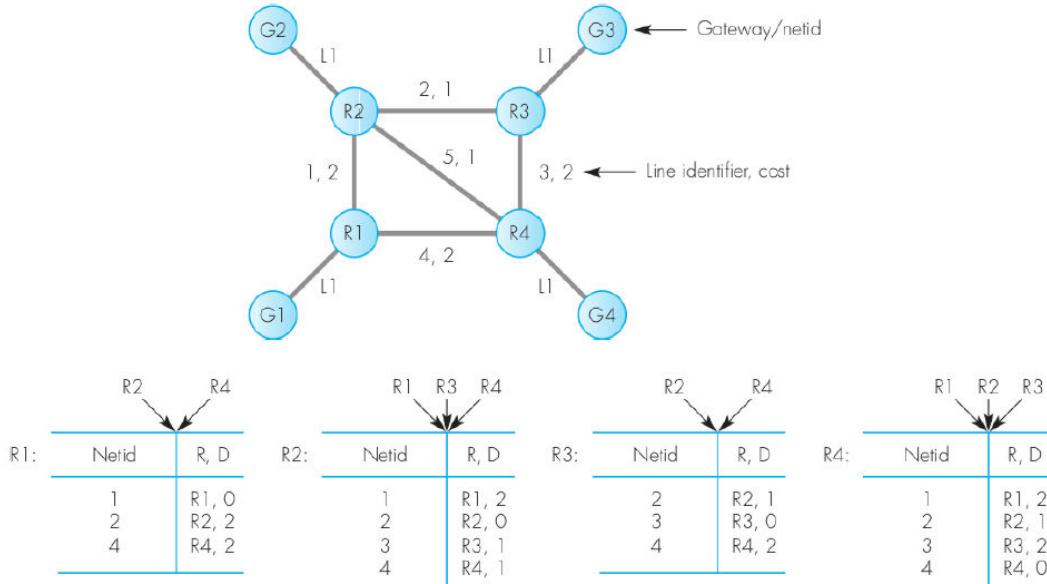
Distance Vector (p.316)

L'algoritmo Distance Vector è un algoritmo distribuito che consente ad ogni router di costruire una routing table che contiene il costo di ogni percorso (Distance) per raggiungere tutti gli altri.

Inizialmente, ogni router si costruisce la propria tabella valutando il costo dei link che lo connettono ai propri vicini.

Il costo di un cammino è la somma dei costi associati ai tratti che formano il cammino. Quando si hanno più scelte, si sceglie il cammino di costo minimo.

Esempio 1:



Passo1

Ogni nodo all'inizio conosce solo se stesso e i nodi a lui adiacenti. Quindi..

Il nodo A conosce: se stesso, il nodo B, il nodo E.

Il nodo A salva le informazioni sui costi dei link in una tabella detta **tabella delle adiacenze**.

	A	
A	0	/
B	3	1
E	4	6

	E	
E	0	/
D	3	5
A	2	6

	B	
B	0	/
A	3	1
C	4	2
D	2	3

	D	
D	0	/
B	3	3
E	2	5
C	3	4

	C	
C	0	/
B	4	2
D	4	4

Passo2

Le informazioni riguardo la distanza da un nodo all'altro viaggiano da nodo a nodo attraverso pacchetti chiamati "vettori" delle distanze.

Il DV(A) dice ai soli nodi adiacenti B, E:

- B,3
- E,4

Ora B sa che esiste un nodo di nome E e che ci arriva con un costo di 4 (*costo da A verso E*) + 3 (*costo da B verso A*) = 7. Queste informazioni vengono memorizzate nella **tabella di routing** di B.

Analogamente le informazioni memorizzate nella tabella di routing di A saranno aggiornate dopo l'arrivo dei distance vector di B e di E.

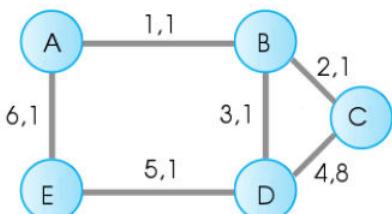
Tabella di Routing di A		
	A	0 /
Arriva DV _B	B	3 1
	E	4 6
Arriva DV _E	C	7 1
	D	5 1
	D	7 6

Scartato perchè
c'è già una strada
più corta!

Criticità dell'algoritmo Distance Vector

- altissimo overhead su reti ampie, dovuto allo scambio dei vettori (viene quindi usato solo su reti piccole)
- *bouncing effect*, vediamolo con un esempio...

Esempio Bouncing Effect ("effetto rimbalzo")



Le entry per raggiungere C sono:

Stazione	Costo	Link
A	4	1
B	3	1
C	0	—
D	4	3
E	3	6

Supponiamo ora che il link 2 si guasti, e non sia più percorribile.

Il nodo B è il primo ad accorgersi, ed aggiorna la sua tabella inserendo ∞ come costo di raggiungimento di C sul link 2. Quando però A invia la propria tabella a B, invia l'informazione che può raggiungere C con costo 2, e quindi B aggiorna la sua tabella di routing inserendo l'informazione di passare da A per raggiungere C. Dopo questo aggiornamento, B invia la propria tabella agli altri nodi e quindi la situazione diventa:

Stazione	Costo	Link
<i>A</i>	2	1
<i>B</i>	1	2
<i>C</i>	0	—
<i>D</i>	2	3
<i>E</i>	3	5

A e B, per questo sfortunato scambio di tabelle, continuano a scambiarsi il pacchetto senza mai farlo procedere verso la destinazione. Questo fino a quando non diventa più conveniente una strada diversa.

Perché è successo questo?

- perché nell'algoritmo non viene mai specificato attraverso quale nodo il pacchetto arriva a destinazione
 - la correttezza dell'algoritmo dipende dall'*interleaving* negli scambi di vettori.

Un modo per risolvere questo problema è il Trigger Update

Trigger Update, appena rilevato un imprevisto, la stazione che ha rilevato il problema spedisce subito il suo vettore senza aspettare lo scadere del timer (inizio dello scambio dei vettori). Ciò nonostante, questa soluzione riduce la possibilità che avvenga l'errore ma non la elimina. Infatti abbiamo ancora il problema che il messaggio di update possa perdere oppure corrompersi.

Count-To-Infinity Criticity In alcuni casi il Bouncing Effect può degenerare in una seconda criticità: count to infinity.

Esempio



Le tabelle verso A sono:

Stazione	Costo	Link
<i>A</i>	0	—
<i>B</i>	1	1
<i>C</i>	2	2
<i>D</i>	3	3

Supponiamo ora che si guasti il link 1.

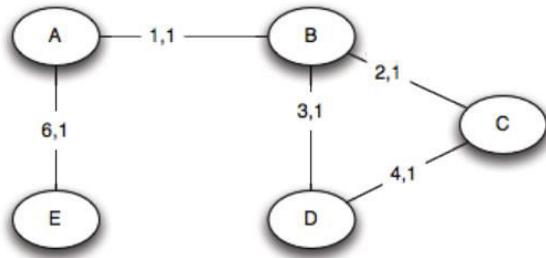
B aggiorna il suo vettore, ma se non è lui il primo ad inviare ma riceve invece il vettore di C o D, allora B aggiornerà la sua tabella e quindi la situazione continuerà erroneamente ad aggiornarsi, aumentando i costi.

Split Horizon

Questa tecnica è candidata a risolvere il problema di bouncing-effect e della criticità count-to-infinity. Ogni nodo propaga il Distance Vector, ma invia sempre ∞ come costo di raggiungimento di stazioni indirette (che raggiungerebbe tramite la stazione a cui sta inviando il vettore, cioè il vicino).

Problemi con Split Horizon

Abbiamo detto che ogni nodo, quando propaga il suo DV, lo spedisce correttamente su ogni link tranne quello che vuole utilizzare (su cui mette infinito), ma se uno di questi DV non arriva a destinazione?



B rileva guasto sul link 1

tabelle verso A:

costo link

da B:	inf	1
da C:	2	2
da D:	2	3
da E:	1	6

Nel caso migliore, B propaga subito il suo DV_B

Cosa succede se C perdesse il pacchetto di B?

costo link

da B:	inf	1
da C:	2	2
da D:	inf	3
da E:	1	6

C propaga DV_C

costo link

da B:	inf	1
da C:	2	2
da D:	3	4
da E:	1	6

D propaga DV_D

costo link

da B:	4	3
da C:	2	2
da D:	3	4
da E:	1	6

In alcuni casi, Distance Vector con Split Horizon porta alla non convergenza delle tabelle.

Possibili soluzioni:

- utilizzare sempre la funzionalità di trigger update vista prima ma, per ogni entry, dopo 6 mancati update si aggiorna a ∞

RIP: Soluzione alle criticità di DV

È il protocollo di Internet che risolve i problemi di routing con tecnica Distance Vector.

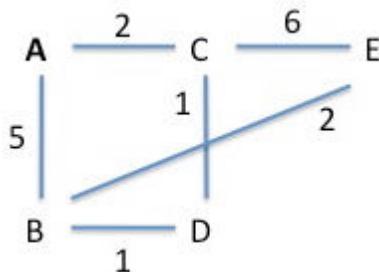
RIP è stato il primo protocollo di instradamento di Internet.

Proprietà:

- 1) Per ogni entry esiste un timer. Se per 6 tempi di update (circa 30 secondi l'uno) non ho risposta, allora viene messa la entry a infinity
 - 2) Trigger Update: esiste un timer di propagazione del DV per ogni nodo. Se un nodo su un suo link rileva un fallimento, spedisce subito il DV ai vicini, anche se i timer non sono ancora scaduti (attenzione: si può sempre perdere il pacchetto, si è sempre esposti al count-to-infinity, ma è raro)
 - 3) Il costo di un link si indica con il numero di hop: 0 – 15, 16 = infinity
 - 4) Update Storm (tempesta di Update): può capitare che i timer scadano tutti insieme, quindi viene generato tantissimo traffico in rete.
- Soluzione: ogni nodo genera il proprio update con un ritardo random (0 – 5 secondi)

Esercizio

Rappresentare le tabelle di routing costruite dopo il primo scambio di Distance Vector nella rete mostrata



t_0	A	B	C	D	E
A		5,B	2,C	non esiste	non esiste
B	5,A		non esiste	1,D	2,E
C	2,A	non esiste		1,D	6,E
D	non esiste	1,B	1,C		non esiste
E	non esiste	2,B	6,C	non esiste	

t_1	A	B	C	D	E
A		5,B	2,C	3,C	7,B
B	5,A		2,D	1,D	2,E
C	2,A	2,D		1,D	6,E
D	3,C	1,B	1,C		3,B
E	7,B	2,B	6,C	3,B	

Esercizio

Si consideri la rete in figura. Usando Distance Vector, ad un certo tempo il nodo C riceve i seguenti vettori (si assume che i costi alle destinazioni siano riportati in ordine alfabetico): da B (5, 0, 8, 12, 6, 2); da D (16, 12, 6, 0, 9, 10); da E (7, 6, 3, 9, 0, 4). I costi stimati da C verso B, D ed E sono rispettivamente 6, 3, 5. Qual è la nuova tabella di instradamento di C?

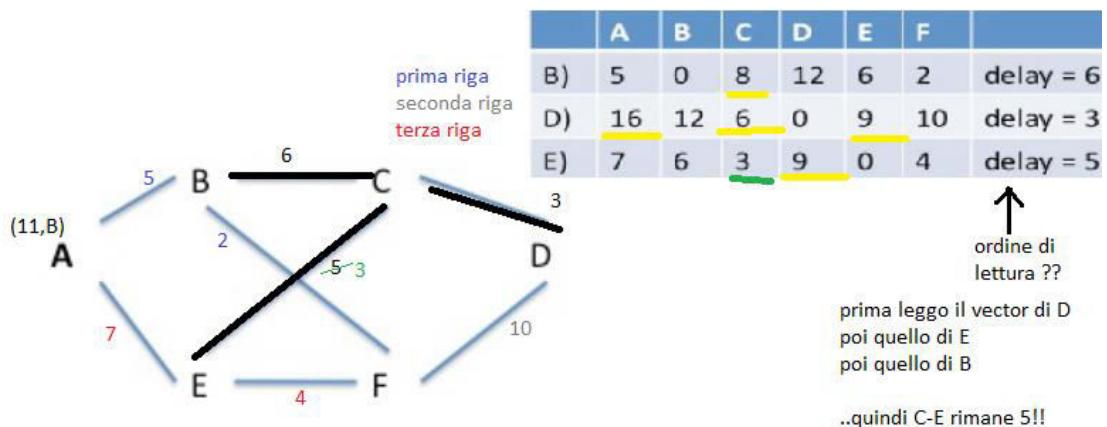


tabella di C :	A	B	C	D	E	F
hop	11	6	0	3	5	8
da	B	B	C	D	E	B

sarebbe come scrivere :
(11,B)
difianco al nodo

Link State Routing

Invece di scambiare i vettori delle distanze, si scambiano i vettori di stato. Cosa significa?

Tutti i nodi si trasmettono i costi dei link a loro connessi (link state).

Questi "vettori di stato" vengono inviati non solo ai vicini, ma in *flooding* a tutti gli altri.

Un nodo diventa quindi indipendente dagli altri perché ha tutte le informazioni che gli servono per avere una visione completa della rete.

Un vettore di stato conterrà il costo di raggiungimento di un nodo ai suoi vicini, e sarà sempre così (diversamente dai distance vector). Questo dato viene mandato a tutti gli altri nodi consentendo a tutti i nodi di mappare tutta la sottorete.

Si utilizzano dei contatori per limitare la permanenza in rete dei pacchetti e di tecniche selettive per forzare la direzione. Ineffetti questo metodo genera un numero enorme di pacchetti (bisogna contare anche tutti gli ACK di risposta) ma seleziona anche il percorso più veloce.

Caratteristiche del pacchetto Link State:

- **numero di sequenza**, In una topologia magliata, lo stesso link state può giungere da nodi diversi. Ho bisogno quindi di un meccanismo per essere in grado di droppeare i link state già ricevuti (il seq_number appunto).
- **fattore di aging**, usato per eliminare pacchetti che continuano a girare in rete (e accorgersi di eventuali loop). Può capitare che un pacchetto permanga troppo nella rete (time-to-live).

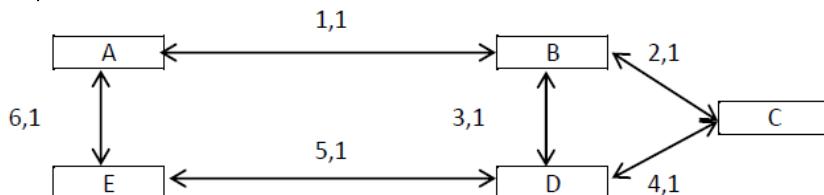
Il protocollo che implementa il Link State è OSPF: una volta determinata la tipologia della rete, ogni nodo calcola i cammini minimi verso ogni possibile destinazione, con l'algoritmo di Dijkstra.

Se un flooding fallisce, verrà aggiornato al prossimo.

Ogni router, al boot, eseguirà:

1. Determinazione address nodi vicini
2. Determinazione dei costi dei nodi vicini
3. Costruzione tabella locale
4. Comunicazione della tabella in flooding a tutti gli altri nodi
5. Calcolo dello Shortest Path per tutti i nodi (Viene applicato l'algoritmo di Dijkstra - SPF)
6. scaduto un timer, ritorna al punto 2

Esempio



All'inizio, tramite operazioni di *discovery* e *request*, ogni nodo si informa della presenza dei soli vicini:

A	B	C	D	E
A /	B /	C /	D /	E /
B 1	A 1	B 1	D 1	A 1
E 1	C 1	D 1	E 1	D 1
D	D	C	C	E

Poi queste informazioni verranno inviate tramite flooding a tutti i nodi della rete.

Ogni volta che un nodo riceve un pacchetto di informazioni lo segnala al nodo mittente tramite un messaggio di ACK.

Per evitare loop dei pacchetti, vengono mandati anche un numero di sequenza del pacchetto e un codice di aging.

Vediamo ora il formato del LS (link state) di A:

A	
Sequenza-Aging	
B	1
E	1

Il link state è dunque più piccolo di un DV perché contiene solo le informazioni dei suoi vicini.

LS di B = 4 (A B C D)

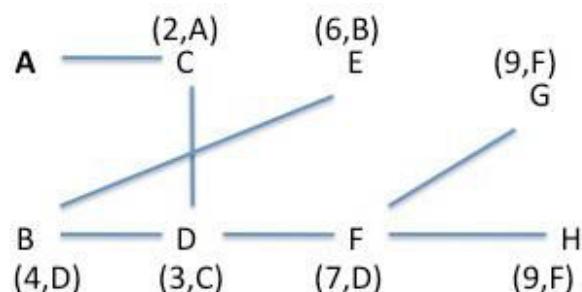
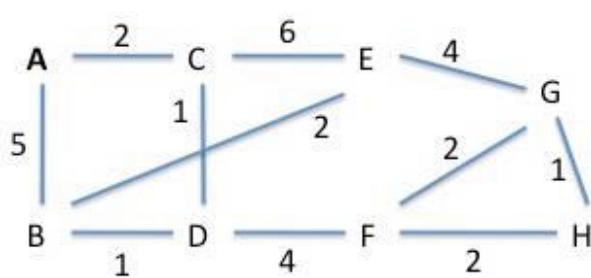
DV di B = 5 (A B C D E)

Link state con flooding risolve le criticità di Distance Vector ma inserisce $O(n^2)$ messaggi nella rete per fare routing.

Come soluzione si designa un router predefinito (*designated router*): tutti gli stati vengono inviati a questo router ed è proprio quest'ultimo a calcolare i cammini minimi e a mandarli in flooding a tutti i "suoi" nodi ogni periodo di tempo (in genere solo se sono cambiati).

Esercizio

Nel grafo seguente, trovare lo shortest path da A ad ogni altra destinazione utilizzando l'algoritmo di Dijkstra



7 msg scambiati per fare broadcast tra 8 nodi

Partendo da A si calcola il cammino minimo (in base al peso dei link) per andare in ciascun nodo:

- 1- Si ridisegnano tutti i nodi
- 2- Si scrive accanto a ciascun nodo scelto come destinazione, qual è il nodo più vicino del percorso minimo trovato e il relativo peso ottenuto.
- 3- Si disegna il link che collega il nodo più vicino al nodo destinazione del percorso minimo trovato

Il **nodo destinazione** cambia di volta in volta, l'operazione si ripete per ciascuno (7 link).

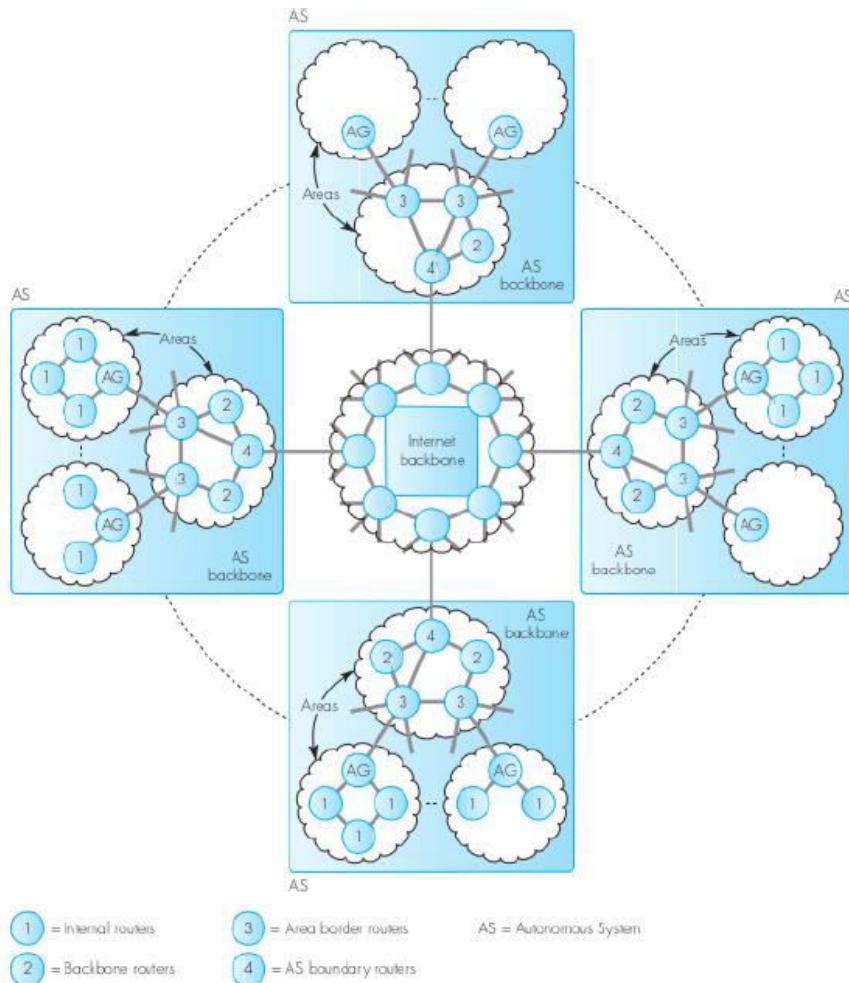
OSPF – Open Shortest Path First (p.340)

OSPF è il protocollo di Internet definito nell'RFC 2328 che gestisce la gran parte dei problemi di routing con tecnica Link State.

La strada più corta (Short Path First) viene calcolata attraverso l'algoritmo di Dijkstra.

OSPF opera :

- negli **autonomous system**, (sistemi autonomi) composti da diverse sottoreti collegate a un'area zero,
- tra **area backbone**, (aree zero) area più alta a livello gerarchico. Un'area zero (in figura *AS backbone*) può essere composta sia da router che da sottoreti. All'area zero sono collegati speciali routers, chiamati **AS boundary routers**, che fungono da gateway tra la sottorete degli host e la Internet Backbone (dorsali internet).



Ogni Autonomous System è composto da una Backbone Area (area 0) che connette le varie sottoreti e gli altri AS.

I router che permettono questi collegamenti sono i border router i quali utilizzano il protocollo BGP che analizzeremo a breve.

L'area 0 è solitamente composta solo da router (comunque non end-devices).

Dato che la Backbone Area funziona con OSPF, per evitare i problemi relativi al flooding, si è adottata una strategia più centralizzata detta **designated router**: nell'Area 0 viene scelto un router che concentra su di sé tutte le update fatte su ogni nodo. Le informazioni raccolte da ogni nodo (le adiacenze) vengono spedite al designated router. Solo esso costruirà le tabelle di routing e le invierà a tutti i nodi connessi, periodicamente. In questo modo evito traffico inutile ma ho maggior vulnerabilità.

I tipi di messaggi che vengono scambiati durante OSPF sono:

- *Hello*, usato da un router per scoprire nuove reti/router adiacenti, scoprendo così il costo di un link
- *Link State Update*, viene inviato a intervalli periodici dal designated router mediante flooding e ha lo scopo di istruire un router sullo stato e il costo corrente di ogni link della topologia. È provvisto di un numero di sequenza con il quale il router ricevente può capire se il messaggio rappresenta un nuovo aggiornamento o una vecchia copia. Questo tipo di messaggio viene inoltre inviato da un router quando il costo o lo stato di un link cambia.
- *Link State Ack*, ogni router valida un Link State Update con questo messaggio

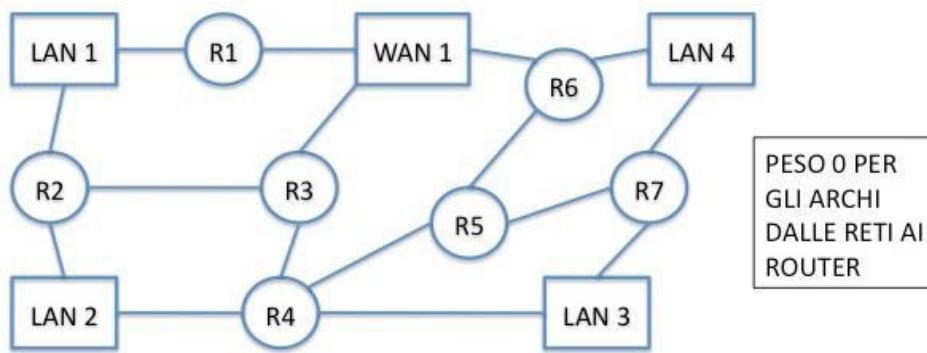
- *Database Description*, viene usato per informare il ricevente del messaggio se è disponibile o meno un aggiornamento
- *Link State Request*, viene usato per richiedere un Link State Update da ogni router adiacente

Ogni messaggio viaggia in un pacchetto IP indipendente, inoltre OSPF offre anche:

- *Multiple Paths to Dest*, mantiene cioè in memoria tutti i possibili percorsi con uguale costo
- *Source Routing Possible*, uno specifico percorso può essere configurato a mano (tipico caso per raggiungere il designated router). Nell'header del pacchetto verranno indicati tutti gli hop da percorrere (*waypoint*), in modo statico. In IPv4 viene indicato nel campo Options
- *Multiple Metrics*, i link possono essere considerati mediante diversi parametri, come bit-rate o distanza fisica ecc.

Esercizio

Allo scopo di applicare OSPF, si rappresenti la rete seguente come un grafo orientato, usando linee tratteggiate per gli archi che hanno sicuramente peso 0.



BGP – Border Gateway Protocol

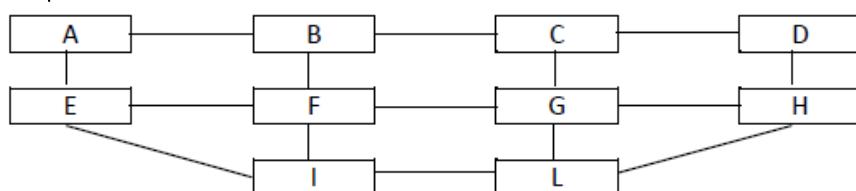
BGP viene usato da ogni AS boundary routers, e quindi permette la comunicazione tra i router che compongono l'area zero.

I messaggi sono inviati tramite TCP, e quindi questo è più un protocollo di livello 7 che di livello 3.

Usa un approccio Distance Vector, ma opportunamente modificato. Chiamato **Path Vector**, indica esplicitamente tutto il percorso da fare per raggiungere un AS boundary router. Al posto di propagare le distanze, si propagano cammini.

Partendo dai Distance Vector, il Path Vector viene costruito usando le tabelle dei propri vicini e scegliendo il cammino minimo che porta alla destinazione; in questo caso il termine "vicino" può non indicare solo la distanza o bit-rate, ma anche distanza dovuta a scelte politiche: questo perché gli AS boundary routers possono anche trovarsi in diversi stati.

Esempio



Vediamo quale può essere il percorso migliore da F a D: il percorso migliore è FCGD

Ora supponiamo che si rompe il link FG

Il nodo F cerca quindi altri possibili link per raggiungere D, e lo fa attraverso i messaggi provenienti dai nodi vicini

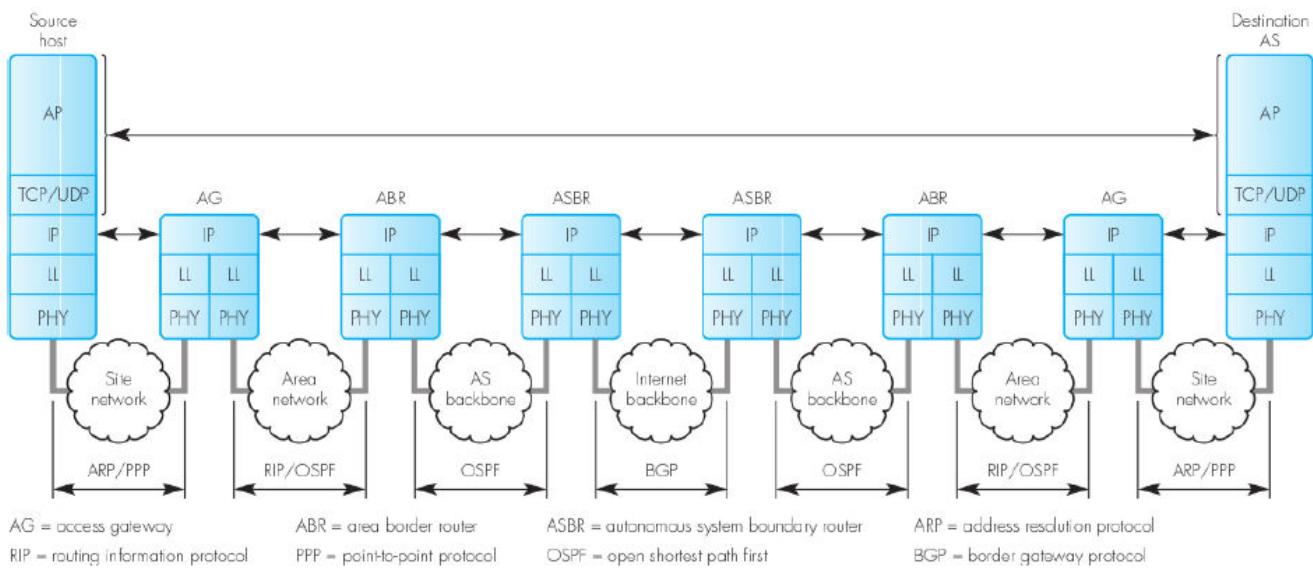
Al nodo F arrivano i PV :

- Da B arriva BCD con costo 8 (link buono, tengo questo)
- Da E arriva EFGCD con costo 3 (ma F si accorge che è presente il link che si è rotto, dunque lo scarta)
- Da I arriva IFGCD con costo 10 (ma F si accorge che è presente il link che si è rotto, dunque lo scarta)

In BGP vengono usati 4 tipi di messaggi:

- *Open*, usato per aprire una relazione con un AS boundary router adiacente
- *Update*, usato sia per trasmettere informazioni sull'instradamento per un singolo percorso che per comunicare una lista di percorsi che devono essere rimossi
- *Keep Alive*, viene utilizzato per validare una Open e per mantenerla attiva, confermandola periodicamente

- *Notification*, usato per informare che si è verificata una condizione di errore

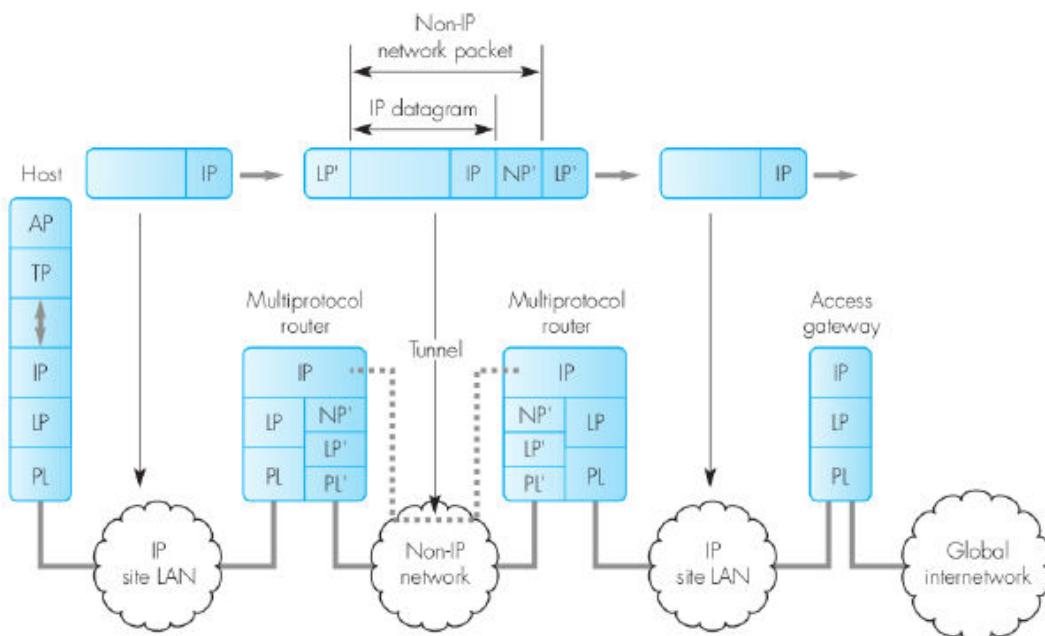


- gli **host** mantengono informazioni sufficienti per poter instradare direttamente pacchetti a host della stessa sottorete, a un subnet router e al default gateway (AG)
- I **subnet router** contengono informazioni sufficienti per poter instradare pacchetti agli altri subnet router e al default gateway (AG)
- I **gateway** mantengono informazioni sufficienti per poter instradare pacchetti alla loro sottorete, cioè direttamente a un host o a un subnet router
- i **router di bordo area** (Area Border Router) contengono informazioni sufficienti per poter instradare pacchetti ad altri router di bordo area nello stesso AS (o differente)
- i **router di bordo AS** possono instradare pacchetti ad altri AS boundary, attraverso la Internet backbone.

Border Gateway Tunneling

Quando un pacchetto IP viaggia attraverso sottoreti, non è detto che tutte usino il protocollo IP.

I router di frontiera delle reti che non usano IP provvedono quindi a incapsulare il pacchetto IP in un altro pacchetto conforme al protocollo usato nella sottorete da attraversare, che verrà gestito nella sottorete non-IP e rimosso dall'ultimo router quando si ritorna sul protocollo IP.



La soluzione adottata è quella di incapsulare il pacchetto IP in un pacchetto di livello 3a (quindi la sotto-rete non IP nemmeno si accorge che sta passando un pacchetto IP).

MPLS – Multi Protocol Label Switching (p.370)

Finora non abbiamo mai considerato i vincoli di tempo. Questo fattore diventa molto importante quando si parla dei sempre più diffusi sistemi real-time in cui lo scambio di messaggi deve essere rapido (si pensi ad esempio alla trasmissione in streaming)

Abbiamo visto nelle lezioni precedenti che le aree 0 funzionavano con un protocollo OSPF, in realtà al giorno d'oggi esso è stato sostituito dal protocollo MPLS. Già dal nome possiamo intuire che si tratta di un protocollo di livello 2 più che del livello 3: lo switch è uno strumento di livello 2 che non fa routing ma forwarding (come MPLS). Questo protocollo aggiunge ai pacchetti di livello 3 una sorta di etichetta (label) che identifica il tipo di priorità di velocità che il pacchetto deve avere.

In base a quanto detto, bisogna anche prevedere una diversa politica di scheduling nelle code di uscita: i pacchetti prioritari andranno in code corte e veloci, gli altri su code lente.

Funzionamento Etichettamento

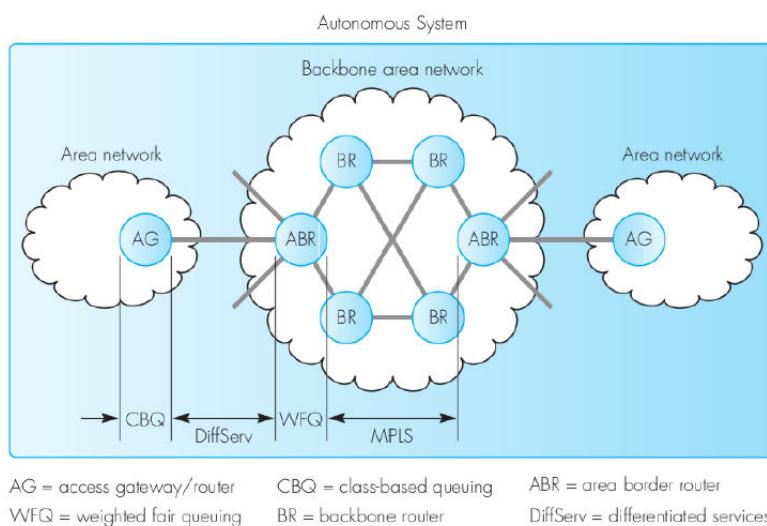
Alla ricezione di un pacchetto da una delle interfacce di ingresso, sappiamo che un router controlla che il pacchetto sia destinato alla sua sottorete controllando l'indirizzo IP in base alla sua address mask.

Oltre a questo valore, il pacchetto contiene un campo ToS (Type of Service) che viene gestito da un **packet classifier**.

Quest'ultimo ha il compito di scegliere le regole di scheduling che devono essere applicate al pacchetto:

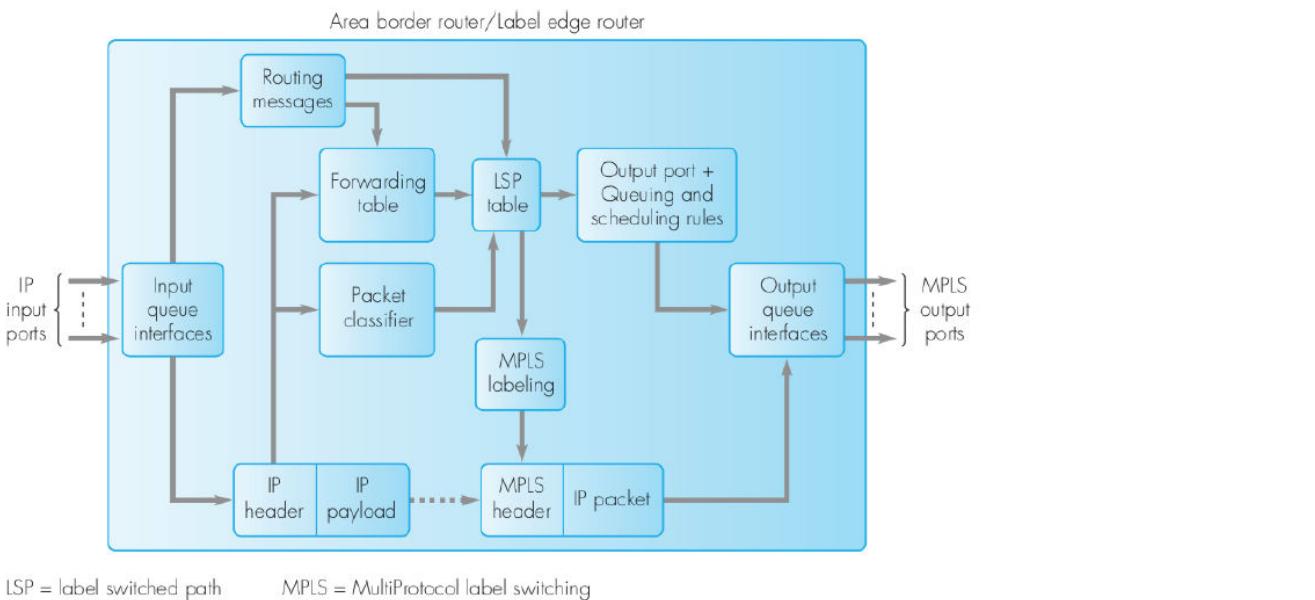
- controlla nell'header del pacchetto IP il suo ToS
- assegna una etichetta al pacchetto
- aggiunge un nuovo header
- inserisce il pacchetto nella coda di uscita più appropriata

In genere il traffico viene diviso in base al tipo, e non in base al singolo flusso di dati (**DiffServ**). L'idea di MPLS invece è di aggregare il traffico di un flusso ed effettuare un bilanciamento tra questi ultimi; questo comportamento viene applicato nell'area di backbone a cui ogni AS è collegato:

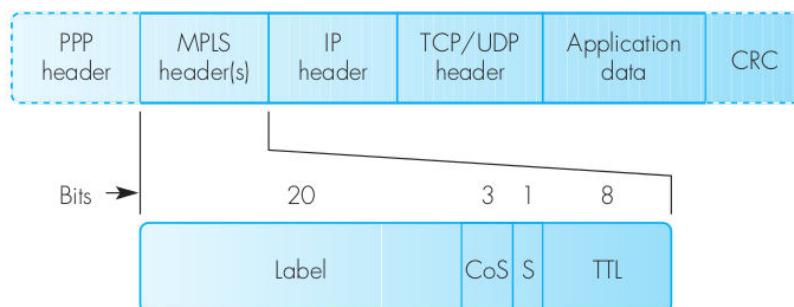


In dettaglio, possiamo così dividere i compiti:

- **Access gateway**, effettua lo scheduling in base alla classe, perché deve distinguere un largo numero di applicazioni e utenti. Quando il traffico è stato classificato e scheduled, procede con la separazione di ogni flusso di dati (**shaping**), necessario per permettere l'interpretazione corretta dei pacchetti di ogni flusso da parte degli area border router (ABR), che separano i pacchetti in code separate (Es. real time, non real time, best effort..) (**grooming**).
- **Area border router**, in questo contesto è anche noto come *LER* (*Label Edge Router*), perché provvede ad aggiungere/rimuovere l'header MPLS dalla testa del pacchetto IP. L'header MPLS ha senso solo ad ogni hop, non come l'indirizzo IP di destinazione che ha significato per tutto il percorso. Questo perché quando vengono usati sempre i percorsi minimi, alcuni area border router potrebbero sovraccaricarsi formando un **hotspot**. Per evitare questi hotspots, viene cambiato l'header MPLS e instradato il pacchetto verso un altro percorso. Viene effettuato quindi un **load balancing** attraverso **label switching**. L'obiettivo primario di MPLS è la gestione del traffico. Per raggiungerlo, su ogni flusso viene determinato un *LSP* (*Label Switched Path*) attraverso l'area zero. Quindi alla ricezione di un pacchetto IP, l'indirizzo di destinazione viene usato per determinare la porta di uscita, mentre il valore DiffServ contenuto in DS (primi 6 bit del campo ToS) viene analizzato dal modulo LSP. Con queste informazioni viene creato l'header MPLS per l'hop e quindi scelte le regole di scheduling del pacchetto.



Sono sufficienti 20 byte per l'header MPLS, che contiene:



PPP = point-to-point protocol

S = 1 single label in header field

CoS = class of service

TTL = time to live

= 0 multiple labels present

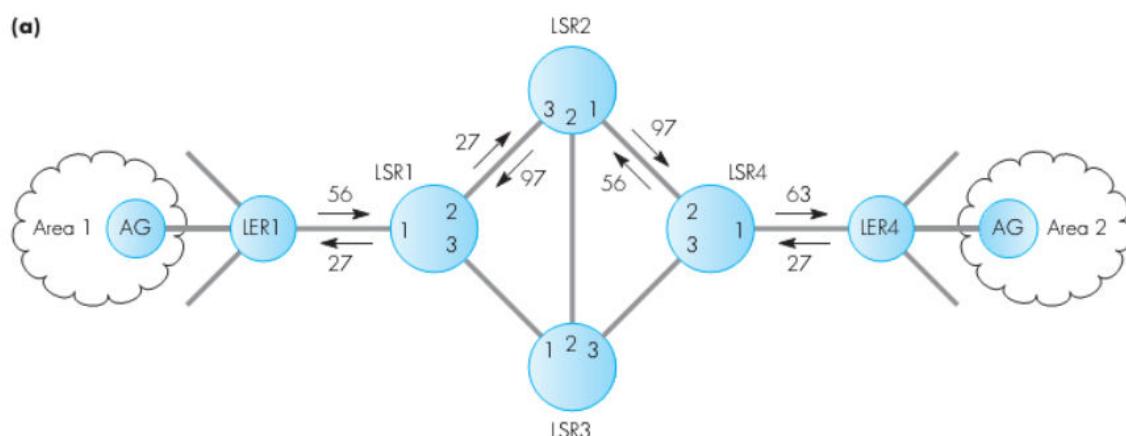
Dove:

- Label, identifica la politica per questo hop e contiene l'identificatore per la porta d'uscita
- CoS, stabilisce le regole di scheduling

Esempio Forwarding

La tabella della porta 1 dell'LSR1 va interpretata così: "se ti arriva sulla porta 1 un pacchetto con etichetta 56, mandalo in uscita con etichetta 27 e usa una politica di accodamento di tipo 3".

Ovviamente, all'inizio, tutte le tabelle sono vuote, devo quindi saltare la parte di etichettatura e guardare l'indirizzo IP presente nell'header. Successivamente costruirò la corrispondenza tra IP e label.



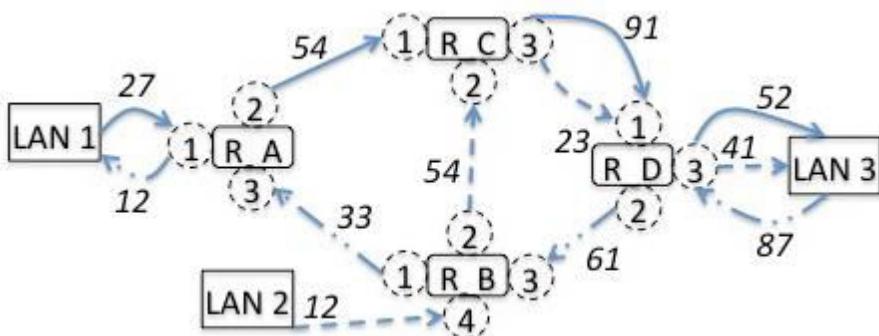
(b)

LSR1 Port1				LSR2 Port1				LSR4 Port1			
Output port	Label	Q + S rule		Output port	Label	Q + S rule		Output port	Label	Q + S rule	
1,56 →	2	27	3	1,56 →	3	97	5	1,27 →	2	56	5
Port2				Port2				Port2			
Output port	Label	Q + S rule		Output port	Label	Q + S rule		Output port	Label	Q + S rule	
2,97 →	1	27	5								
Port3				Port3				Port3			
Output port	Label	Q + S rule		Output port	Label	Q + S rule		Output port	Label	Q + S rule	
3,27 →	1	97	3								

Note: two different flows [i] from Area 1 to Area 2 and [ii] from Area 2 to Area 1

Esercizio

Dallo schema di rete seguente, ricavare le tabelle di instradamento dei router in accordo a MPLS. /* e viceversa: date le tabelle, ricavare lo schema del sistema */



ROUTER A		ROUTER B		ROUTER C							
iif	label	oif	label'	iif	label	oif	label'	iif	label	oif	label'
1	27	2	54	3	61	1	33	1	54	3	91
3	33	1	12	4	12	2	54	2	54	3	23

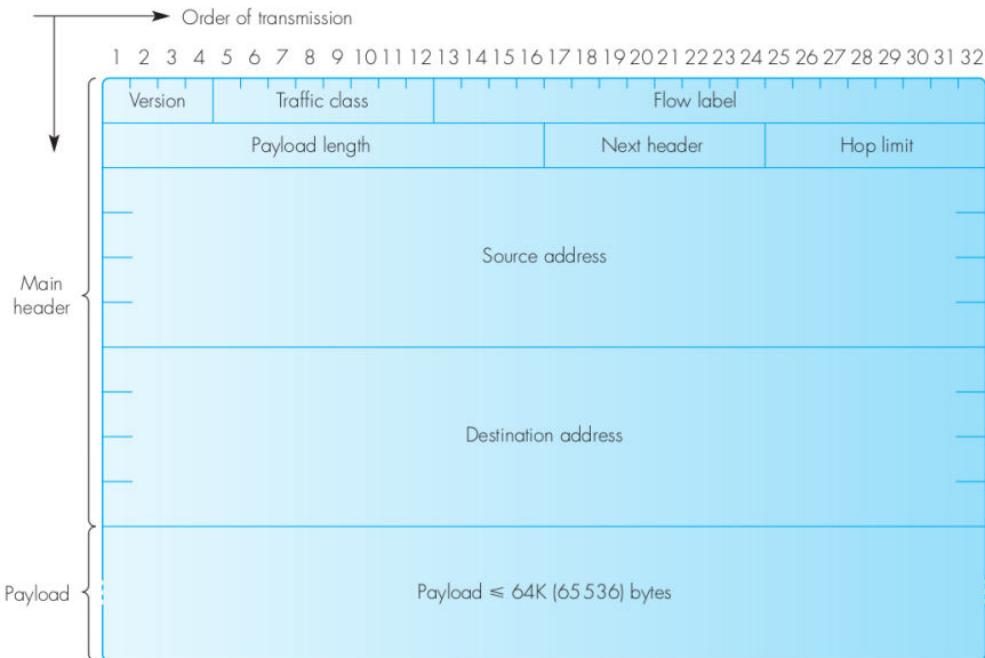
ROUTER D			
iif	label	oif	label'
1	91	3	52
1	23	3	41
3	87	2	61

IPv6

Nato per sopperire alla mancanza di indirizzi IPv4, aggiunge nuove funzionalità, le cui principali sono:

- spazio degli indirizzi aumentato da 32 bit a 128 bit (da 2^{32} a 2^{128}) divisi in 8 word da 2 byte
- indirizzi gerarchici, per ridurre le dimensioni delle routing table associate ai router della Internet backbone
- un header semplificato per aumentare la velocità di processing dei router (la fase di routing rimane invariata)
- introduzione di sistemi di sicurezza come autenticazione e crittazione
- una funzione di autoconfigurazione per consentire ad un host di ottenere un indirizzo IP tramite la rete senza l'intervento umano.
- compatibilità con i protocolli IPv4

Un header IPv6 basic è di 40 byte (contro i 20 B di IPv4):



8 B per parametri

{

- **Version**, (4 bit) indica la versione del datagramma IP: per IPv6 è settato a 6
- **Traffic class** (8 bit), si traduce come “classe di traffico”, successore di ToS, classifica il pacchetto favorendo le regole di routing e lo smistamento assegnando una classe di priorità – “by priority” – rispetto ad altri pacchetti provenienti dalla stessa sorgente (valori alti corrispondono a priorità alta). Viene usata nel controllo della congestione.
- **Flow label** (20 bit), usata dal mittente per etichettare una sequenza di pacchetti come se fossero nello stesso flusso. Supporta la gestione del QoS (Quality of Service), consentendo ad esempio di specificare quali etichette abbiano via libera rispetto ad altre. Al momento questo campo è ancora in fase sperimentale. (settato a 0, indica un pacchetto best-effort. È usato per gestire il controllo di flusso e assieme a Traffic Class garantisce i requisiti QoS)
- **Payload length** (16 bit), è la dimensione del payload, ovvero il numero di byte di tutto ciò che viene dopo l'header. Da notare che eventuali estensioni dell'header sono considerate payload e quindi conteggiate nella lunghezza del carico. Max 64 KB (kilobyte, 65535 byte detto *Jumbogram*) e min 536 byte (perché 576 è il payload minimo per essere certi che il pacchetto non venga frammentato – 536 + 40 B header IPv6 = 576)
- **Next header** (8 bit), indica quale tipo di intestazione segue l'header di base di IPv6. Molto simile al campo protocol dell'header IPv4, del quale usa gli stessi valori:

Header IP	Extension H 1	Extension H 2	Extension H 3	Extension H 4	Header TCP/UDP
Header IPV6					

Innovazione di IPv6 che consente di usare più spazio per l'header (necessario per frammentare e opzioni). Se non c'è estensione di header, allora Next Header conterrà il puntatore all'header del livello superiore (cioè TCP). Se c'è estensione, Next header punta ad un HE (Extension Header) che contiene anch'esso un campo Next Header e così via.. fino ad arrivare a Next Header che punta a TCP.

- **Hop limit** (8 bit), è il limite di salti consentito, praticamente è il TTL per IPv6, non è il tempo ma il numero di hop che si contano (default 256 hop). Il suo valore viene decrementato di 1 ogni volta che il pacchetto passa da un router: quando arriva a zero viene scartato.

}

32 B per indirizzi

{

- **Source Address** 16 B per indirizzo sorgente
- **Destination Address** 16 B per indirizzo destinazione

}

Extension Headers

Se il pacchetto IPv6 è un pacchetto base, allora il campo NextHeader punterà all'header TCP; altrimenti conterrà il valore del tipo di pacchetto che deve estendere (esempio: hop per hop: 0)

Osservazione: in IPv6 solo la sorgente può frammentare. Occorre quindi conoscere la MTU (maximum transfer unit) di ogni sottorete che il pacchetto dovrà attraversare. Per fare ciò, prima di inviare un pacchetto, viene inviato un ICMP che riporterà indietro alla sorgente la taglia minima del pacchetto trasferibile su tutto il percorso.

Analizziamo i vari tipi di Extension Headers:

Next Header

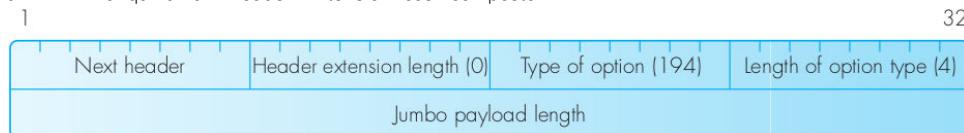
In IPv6 le opzioni vengono messe nel Next Header

L'header IPv6 può diventare più grande per contenere informazioni aggiuntive:

- opzione hop per hop:** informazioni per i router visitati lungo un percorso
- intradamento:** elenco dei router coinvolti nell'intradamento dalla sorgente (*source routing*)
- frammentazione:** informazioni che consentono alla destinazione di ricomporre un messaggio frammentato
- autenticazione:** informazioni che consentono alla destinazione di verificare l'identità della sorgente
- sicurezza:** informazioni che consentono alla destinazione di decifrare il contenuto del payload
- opzioni per la destinazione:** facoltative.

Opzione Hop By Hop

Se c'è 00 nei bit meno significativi del Next Header del pacchetto IPv6, allora si usa Hop By Hop. Serve per gestire pacchetti più grandi di 64 K. Avrò quindi un Header Extension così composto:



Dove:

- *Next Header*, rappresenta il puntatore all'header successivo
- *Header Extension Length*, esclude i primi 8 byte, quindi in questo caso è 0
- *Type of option*, 194 per hop by hop
- *Length of option type*, lunghezza del Jumbo payload length, in questo caso 4
- *Jumbo payload length*, lunghezza del payload

Routing

Se c'è 43 nei bit meno significativi del Next Header, allora si usa Routing. Serve per implementare il source routing. Avrò quindi un Header Extension così composto:

Next header	Header extension length	Routing type (0)	Segments left
Reserved	Strict/loose bit map (24 bits)		
Address [1]			
Address [2]			
⋮			
Address [N]			

Bit map: 1 = strict source routing $N \leq 24$
 0 = loose source routing

Next header: hop-by-hop options = 0
 routing = 43

Dove:

- *Header Extension Length*, al massimo 24 indirizzi da cui si vuole passare
- *Routing Type*, non è ancora ben definito..
- *Segments Left*, dovrebbe essere inizializzato al numero di step intermedi che partecipano. Viene decrementato ad ogni hop prefissato
- *Reserved*
- *Strict/loose bit map*, tanti bit quanti i possibili router fissati. Se c'è 1 voglio un collegamento diretto verso quel router (routing esatto), se c'è 0 non mi interessa come arrivo a quel router ma è sufficiente che ci si passi (routing approssimato). Questo campo mi dice quindi se il prossimo router è indicizzato dal precedente in modo diretto o meno

Fragment

Next header	Res(erved)	Fragment offset	Res	M
Identification				

Come IPv4, abbiamo l'identificazione del pacchetto, il fragment offset e il More Fragment bit. Come differenze notiamo che l'unico a frammentare è il router sorgente, non come in IPv4 dove frammenta ogni gateway per la sua sottorete.
 Quindi la sorgente deve sapere dove il pacchetto andrà e come frammentarlo, e lo fa attraverso speciali comandi ICMP. Vuol dire anche che le sessioni IPv6 usano source routing per i pacchetti frammentati, perché devono far passare i pacchetti sempre dalla stessa via

Formato indirizzi IPv6

Un indirizzo IPv6 è rappresentato comunemente da 8 blocchi di 16 bit divisi da ":", che per comodità vengono scritti in esadecimale; inoltre, vista la alta probabilità di blocchi composti da soli 0, è possibile omettere il blocco o i blocchi di 0 contigui e, raddoppiando il separatore ":"; passando al blocco successivo.

Esempio

16bit: 16bit : 16bit : 16bit : 16bit : 16bit
 FDEC : BA98 : 7654 : 3210 : 0000 : 0000 : 0089 *in notazione completa*
 FDEC : BA98 : 7654 : 3210 :: 0089 *in notazione contratta*
 (:: = tutti 0 in mezzo)

Per facilitare il processo di transizione dal vecchio al nuovo protocollo sono state tuttavia introdotte tre tipologie di indirizzi unicast IPv6, pensate per poter contenere al loro interno indirizzi IPv4. In particolare gli indirizzi:

- *IPv4-Compatible IPv6*, impiegati in alcune situazioni di protocol tunneling, sono formati da 96 bit posti a 0 seguiti dall'indirizzo IPv4 unicast da cui derivano (es. ::193.206.71.151).
- *IPv4-Mapped IPv6* sono invece formati da 80 bit a 0 seguiti a una porzione di 16 bit a 1 e infine dall'indirizzo IPv4 da cui derivano (es. ::FFFF:193.206.71.151).
- *IPv4-Translated IPv6* sono invece formati da 64 bit a 1, seguiti da 16 bit a posti 1 e quindi altri 16 a 0 e quindi dai bit dell'indirizzo IPv4 da cui derivano (es. ::FFFF:0:193.206.71.151).

Queste ultime due tipologie di indirizzi vengono impiegate in quelle situazioni in cui è necessario far colloquiare nodi IPv4-only con nodi IPv6-only.

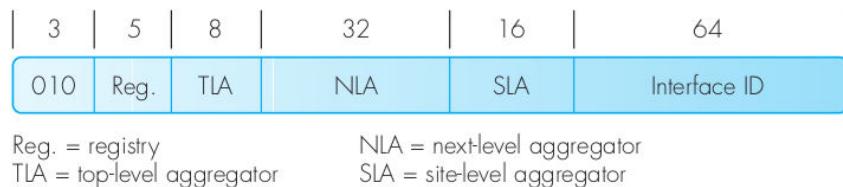
IPv6 attua una organizzazione gerarchia basata sulla grandezza del servizio:

- Tier 1, organizzazioni a livello continentale. Con Registry e TLA si identifica una singola organizzazione di livello 1
- Tier 2, organizzazioni a livello nazionale. Campo NLA
- Tier 3, organizzazione a livello locale. Campo SLA

Questa organizzazione ha un grande impatto sulla dimensione delle tabelle di routing, visto che si trasforma in una divisione dei 128 bit tramite PF (**Prefix Format**) e quindi tutti i pacchetti con stesso prefisso verranno instradati nella stessa direzione.

Si effettua quindi una aggregazione degli indirizzi (**address aggregation**).

I primi bit dell'indirizzo si chiamano prefisso. Un prefisso del tipo 010 indica una struttura gerarchica così composta:



- 010 indica la comunicazione punto-punto
- registry indica il continente
- TLA indica la nazione
- NLA indica una zona intra-nazionale
- SLA indica una zona locale
- Interface ID è così composta:
 - 16 bit per la sottorete (NetID, sono usati per il subnetting)
 - 48 bit corrispondente all'indirizzo MAC dell'host (HostID)

Infine, per consentire interoperabilità tra IPv4 e IPv6, è anche possibile inserire un indirizzo IPv4 in un indirizzo IPv6:



Interoperabilità IPv6/IPv4

La transizione IPv4/IPv6 riguarda il passaggio nella rete globale Internet del protocollo di livello di rete IP dalla versione 4 alla versione 6. Il passaggio porta ad ingenti problemi di compatibilità poiché la versione 4 del protocollo IP è molto consolidata e rappresenta il protocollo più usato nelle reti per l'indirizzamento e il successivo instradamento. Il piano di transizione è riportato nella RFC 2893.

Difficoltà della transizione

Il motivo principale che ha portato alla definizione del protocollo [IPv6](#) è il graduale esaurimento degli indirizzi passati su protocollo [IPv4](#). Ma l'eterogeneità e vasta dimensione della rete porta ad ingenti problemi di compatibilità poiché la versione 4 del protocollo IP è molto consolidata e rappresenta il protocollo più usato nelle reti per l'indirizzamento e il successivo instradamento. Il piano di transizione è riportato nella RFC 2893.

IPv4 non può venire aggiornato tutto in una volta, quindi ci troviamo ad avere macchine IPv4 che parlano con macchine IPv6. Come permettere questa possibilità?

La politica naturalmente adottata per la transizione ad IPv6 consiste in un graduale passaggio da un protocollo all'altro, cercando di far coesistere le due versioni di IP in un'unica rete. Per far ciò la strada seguita fino a questo momento consiste nel costruire [router](#) e [switch](#) di livello 2 e 3 in grado di interpretare entrambi i protocolli, inoltre, da qualche anno i nuovi [sistemi operativi](#) sono in grado di generare indirizzi IPv6 e di interpretarli. In questo modo ogni host nella rete è individuabile da almeno due indirizzi, uno dato da IPv4 ed uno da IPv6. Ma, come detto prima, la sostituzione di tutti i router nel mondo risulta un lavoro piuttosto arduo e allora si è proceduto ad operare via [software](#) cercando in qualche modo di aggirare la non interpretabilità di IPv6.

Tutte le soluzioni finora create possono essere individuate tre categorie principali:

- *dual-stack*
- *NAT-PT*
- *tunneling*

1- Dual stacks/protocols

La tecnica del **dual stack** prevede l'utilizzo del doppio stack IP, nella pila protocollare. Questo doppio stack permette di poter interpretare entrambe le versioni del protocollo e, quindi, smistare ai livelli superiori il contenuto del pacchetto senza che questi sappiano da quale protocollo IP derivì. Un aspetto interessante è che a ciascun pacchetto IPv4 e IPv6 è associato un diverso *EtherType*, e ciò semplifica lo smistamento del pacchetto che può essere fatto al secondo livello della pila protocollare, senza quindi dover accedere al campo *version* del pacchetto di livello 3.

Il dual stack è senza dubbio una delle tecniche più semplici da implementare ma presenta molte controindicazioni. Innanzitutto aumenta la complessità della rete: [router](#) e [switch](#) vengono dotati della proprietà del multiprotocollo ma il lavoro di questi viene ulteriormente aggravato poiché devono interpretare più istanze dello stesso protocollo. Inoltre non risolve il problema della diminuzione degli indirizzi IPv4 poiché secondo la tecnica del dual stack un'interfaccia dev'essere sempre e comunque dotata di due indirizzi, IPv4 ed IPv6. I due indirizzi, inoltre, debbono essere entrambi annunciati in internet e ciò non semplifica la situazione del routing, anzi, tendenzialmente la complica.

Successivi miglioramenti al dual stack hanno portato alla nascita di nuove tecniche quali il DSTM e l'ALG.

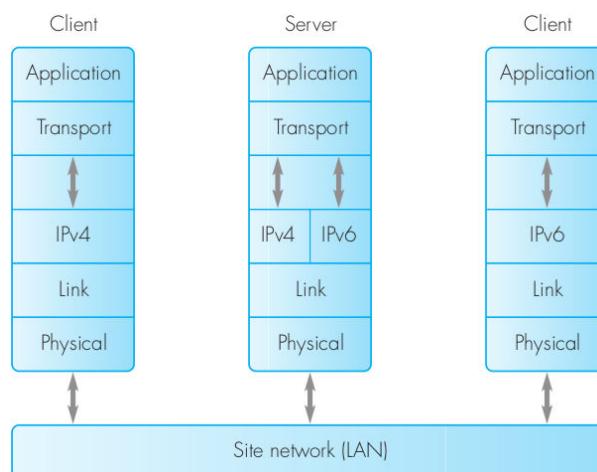
L'**ALG (Application Level Gateway)** è una soluzione che permette ad una rete totalmente IPv6 di poter dialogare con postazioni IPv4 al di fuori della rete stessa. All'interno di questa vi sono solo host che comunicano mediante il protocollo IPv6 tra loro e l'IPv4 non è previsto, solo il [gateway](#) è dotato di dual stack. In questo modo se gli host interni alla rete vogliono dialogare con server esterni (o il contrario), ciò è reso possibile grazie alla capacità del gateway di interpretare entrambi i protocolli.

Il vantaggio della tecnologia ALG consiste nel dotare solo un apparato in tutta la rete del dual stack, non appesantendola ulteriormente.

Però ad oggi i router ALG possono operare solamente su alcuni dei servizi di rete (per esempio [HTTP](#)), se invece si vuole dotare la rete di altre funzionalità, lo si fa aggiungendo altri router ALG dotate di tali funzionalità.

Esempio

Come intuibile dalla figura, una LAN può avere al suo interno sia macchine IPv4 che macchine IPv6: allora si provvede ad equipaggiare le macchine che potrebbero trovarsi a dover comunicare con entrambe le versioni (in questo caso il server) con entrambi i protocolli, che possono essere discriminati in base al campo Version dell'header IP.

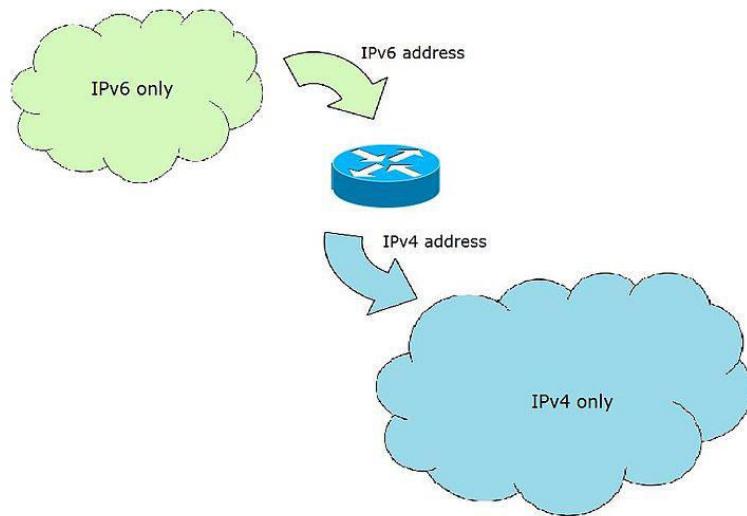


2- Network Address Translator – Protocol Translator (NAT-PT)

Il **NAT-PT** è un sistema che sfrutta i concetti introdotti dalla tecnologia dei [NAT](#). Infatti esso opera una conversione dell'indirizzo IPv6 in indirizzo IPv4 e viceversa secondo le tecniche di un NAT IPv4, permettendo in questo modo a due reti con protocolli IP diversi di poter comunicare tra di loro.

Naturalmente questa tecnica (come nel caso di NAT IPv4) introduce molti limiti, infatti alcuni servizi non funzionano, a meno che non si introducano nella rete specifici ALG. Inoltre può introdurre molti limiti nelle prestazioni e nella complessità della rete.

A fronte di limiti e pregi, questa tecnica risulta essere piuttosto efficiente per le reti IPv6 poiché si tratta di soluzione temporanea che prevede la rimozione del NAT-PT nel momento in cui anche le reti confinanti si saranno adeguate al protocollo IPv6.

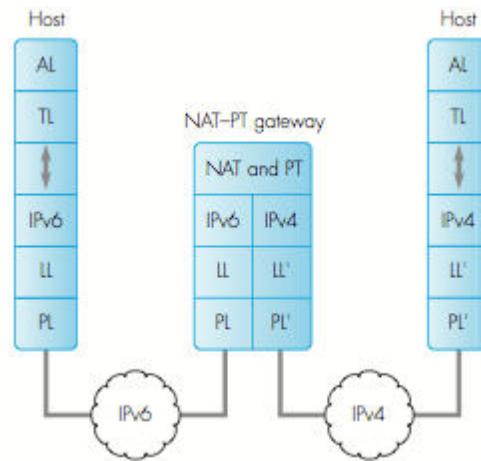


Esempio

Questo tipo di interoperabilità riguarda un host collegato alla rete IPv6 che intende comunicare con un host collegato ad una rete IPv4. In questo caso l'indirizzo ed il formato del pacchetto è differente. Pertanto deve essere effettuata un'operazione di traduzione tra i gateway intermedi, effettuata dal NAT del gateway.

Quando la macchina IPv6 vuole comunicare con quella IPv4, deve per forza passare tramite il NAT per rendere l'indirizzo di destinazione (IPv4) un indirizzo IPv6. La conversione avviene in questo modo: dato un indirizzo IPv4, il corrispettivo IPv6 diventa ::FF:IPv4.

L'host IPv4 per comunicare con l'host IPv6 deve avere indirizzi compatibili. Il NAT non esegue più una traduzione degli indirizzi ma presta per tutto il tempo della sessione (scandito da un timer) un indirizzo IPv4 disponibile in un pool all'host IPv6. Il NAT si salverà in una sua tabella il matching tra gli indirizzi IPv4 e IPv6.



3- Tunneling

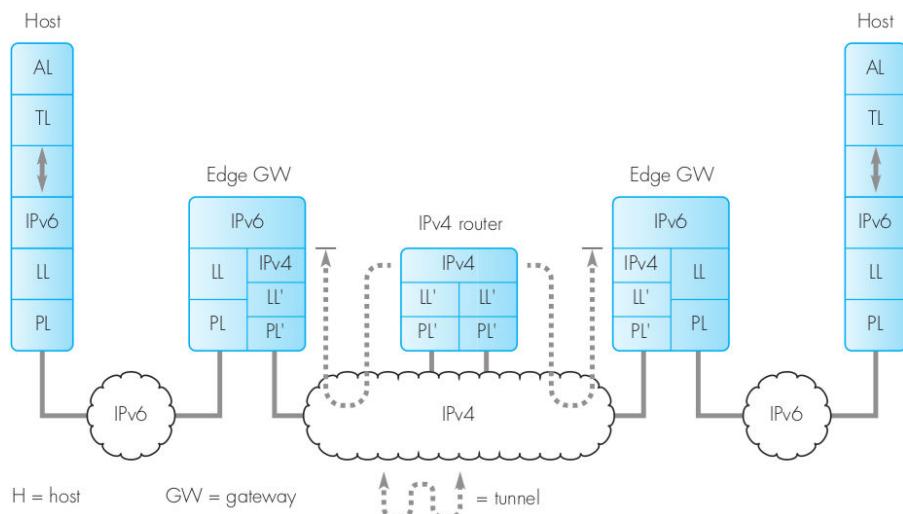
Ad ora la tecnica del tunneling è quella più utilizzata per far fronte ai problemi di incompatibilità tra le reti [IPv4](#) ed [IPv6](#). Questa tecnica usa il principio del [tunneling](#), per cui si stabilisce un collegamento point to point tra due host. I pacchetti IPv6 vengono, così, incapsulati dall'host sorgente in pacchetto IPv4, inviati nel tunnel e, una volta giunti a destinazione, l'host li decapsula e li tratta come se fossero comuni pacchetti IP.

Il tunneling IPv6 su IPv4 ha una difficile realizzabilità per le reti globali e quindi il suo utilizzo è limitato ad applicazioni e comunicazioni in reti locali più o meno grosse.

Per superare questi limiti sono stati creati numerosi protocolli basati sempre su questa tecnica.

Esempio

Un requisito fondamentale per la interoperabilità è la possibilità di far parlare isole IPv6 attraverso una sottorete IPv4. Per ottenere questo risultato, i gateway che connettono le isole alla rete intermedia devono avere due stack di protocolli: uno per supportare il traffico proveniente dall'isola e un altro per il normale traffico IPv4 proveniente dalla rete intermedia.



Funzionamento:

- encapsulamento del pacchetto IPv6 in un pacchetto IPv4 e come indirizzo di destinazione viene inserito l'indirizzo IPv4 del gateway remoto IPv4/IPv6
- a questo punto il pacchetto (diventato un normale pacchetto IPv4) può raggiungere il gateway remoto attraverso la rete intermedia
- all'arrivo al gateway remoto, questi rileva che è un pacchetto encapsulato dall'indirizzo di destinazione e quindi l'header IPv4 viene rimosso e il pacchetto IPv6 viene passato allo stack IPv6, che può trattarlo nella maniera corretta e instradarlo all'interno dell'isola di destinazione.

Livello Transport (p.405)

Da questo momento in poi non si parlerà più di nodi ma di host (dal livello 4 al 7).

Siamo all'interno dell'host, ora dobbiamo gestire la comunicazione tra processi all'interno della stessa macchina.

È il primo livello end-to-end, in cui le operazioni vengono svolte sulle macchine host, visto che i problemi di instradamento sono risolti ai livelli inferiori. I servizi offerti in questo livello possono essere affidabili (TCP – Transmission Control Protocol) o non affidabili (UDP – User Datagram Protocol).

Naming

Il problema di naming consiste nel riuscire a determinare univocamente un'entità, consentendone così la comunicazione.

Ogni livello ha un protocollo destinato alla soluzione di questo problema:

- tra il livello 2 e il livello 3 c'è ARP/RARP
- siccome IP viene usato da una vasta quantità di protocolli, all'arrivo di un pacchetto IP esiste un **Protocol Selector** che, leggendo il campo Protocol dell'header IP, capisce a quale protocollo il payload è destinato (se UDP o TCP)

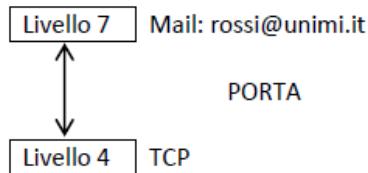
Una volta che il pacchetto è arrivato al livello 4, il protocollo (UDP o TCP) deve mandare il pacchetto al servizio corretto il quale dovrà leggerne i dati.

Si ha quindi la necessità di:

- associare un indirizzo IP ad nome simbolico di un servizio (indirizzo di posta elettronica, indirizzo web, ecc)
- legare una singola applicazione e quindi uno specifico processo di un servizio, ad un'indirizzo IP

Per il primo problema, abbiamo il protocollo DNS, che per ogni nome, restituisce l'indirizzo IP corrispondente e che verrà trattato nel dettaglio nella parte dedicata al livello 7.

Per il secondo problema, invece, si introduce un nuovo tipo di astrazione: la **porta**. Ogni applicazione che comunica in rete viene associata a una porta, cioè a un valore numerico.



Porta: descrittore di sessione di livello 4 che lega in modo univoco l'istanza di livello 4 con il processo di livello 7.

A livello 4, la coppia <IP,porta>, chiamata **socket**, determina in modo univoco un processo in Internet.

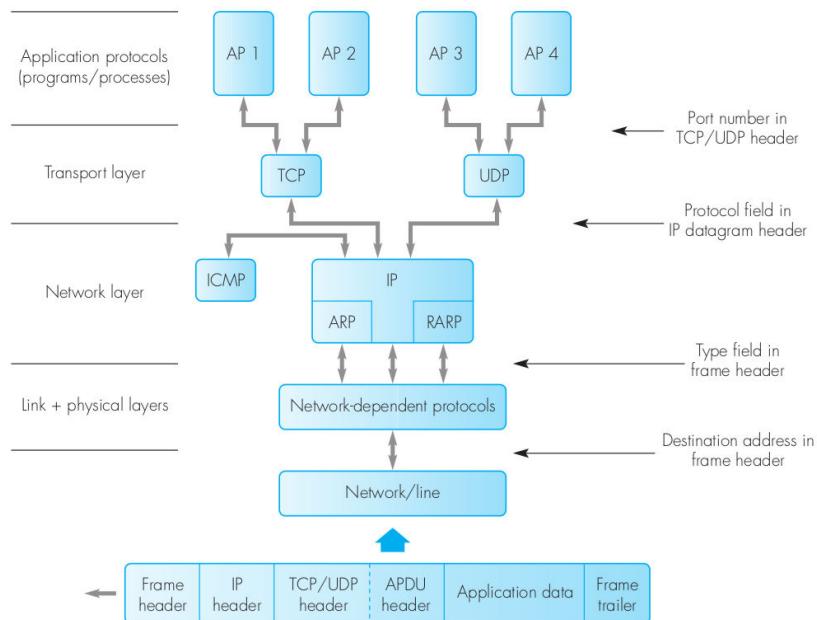
Troveremo quindi in ogni segmento TCP/UDP la porta sorgente e la porta destinazione; in generale, la porta sorgente viene detta **ephemeral port** (porta di vita breve), perché ha importanza solo localmente, cioè solo per determinare il processo che ha generato il pacchetto. Viene scelta una nuova porta ogni volta che un processo vuole comunicare in rete.

Invece la porta di destinazione deve essere nota, cosicché si possa generare il segmento in modo corretto: queste porte, dalla 1 alla 1023, vengono dette **well-known ports**, e ogni applicazione degna di nota ne ha una fissata (per esempio http risiede sulla porta 80, SSH sulla 22, FTP sulla 21).

In totale si hanno 2^{16} possibili porte:

- Dalla 0 alla 1023 sono associate a servizi standard (mail, web..)
- Dalla 1024 alla 49151 sono associate a “programmi famosi” (aziende private)
- Le porte successive sono lasciate libere

Vediamo in figura un riepilogo dei protocolli visti e le loro interazioni:



Architettura client/server

Tutti i servizi internet seguono il paradigma client/server

Client: sempre attivi (ovvero sono essi ad effettuare le richieste)

Server: sempre passivi (rispondono alle richieste)

I servizi offerti da un server stanno in ascolto su delle porte predefinite (comprese tra 0 e 1024), mentre i client hanno un numero di porta dinamico (da 1025 in poi).

Esempio

Due client web sulla stessa macchina vogliono comunicare con uno stesso server, con IP2 sulla porta 80.

CL1: porta x, IP1

CL2: porta y, IP1

Server: www.unimi.it, porta 80, IP2

CL1, tramite DNS, scopre che l'indirizzo www.unimi.it è associato all'indirizzo IP: IP2.

CL1 comunica al protocollo TCP queste informazioni:

Src1: port x, IP1

Dest1: port 80, IP2

<Porta_Sorgente, IP_Sorgente, Porta_Destinazione, IP_Destinazione, Protocol_Selector> sono i parametri di una socket.

Esempio:

<x, IP1, 80, IP2, TCP>

Lo stesso accade per CL2:

Src2: port y, IP1

Dest2: port 80, IP2

L'unica cosa che cambia tra le due richieste è la porta di origine.

TCP, lato server, riesce a discriminare le due richieste e nella risposta verranno settati:

Risposta a CL1:

Src: port 80, IP2

Dest: port x, IP1

Risposta a CL2:

Src: port 80, IP2

Dest: port y, IP1

Osservazione: Un server, per gestire connessioni multiple, crea un figlio (`fork()`) per ogni connessione attiva. Fintanto che il padre non riceve una richiesta, rimane in wait aspettando la prima connessione. Il numero di figli che un server può creare è predefinito.

TCP – Transmission Control Protocol

Fornisce un servizio che garantisce affidabilità alla comunicazione best-effort di IP.

La comunicazione TCP è generalmente una comunicazione client/server.

È bidirezionale, cioè apre due connessioni, una client-server, l'altra server-client.

TCP svolge tre diverse operazioni:

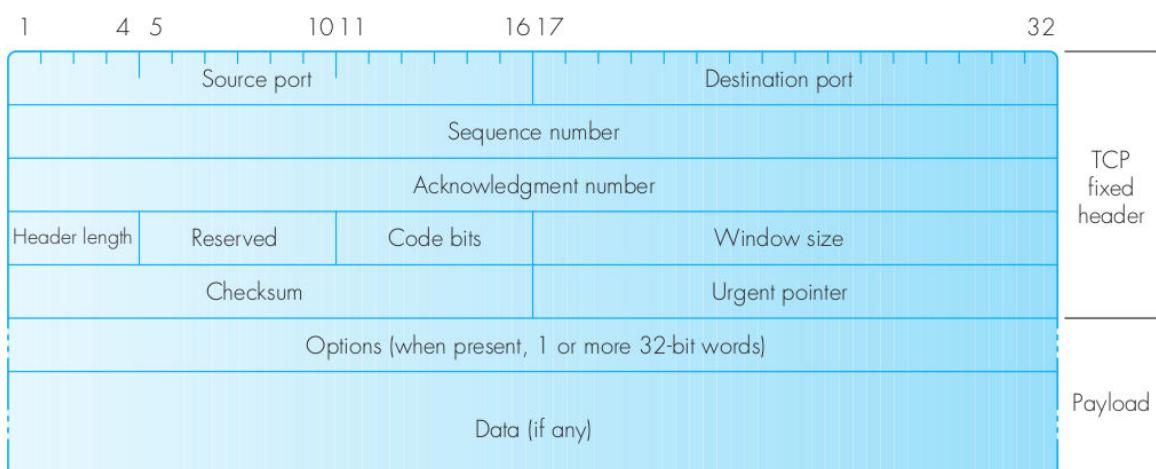
1- configura una connessione logica tra due socket precedentemente create

2-trasferisce in modo affidabile i blocchi di dati su questa connessione

3-chiude la connessione logica

È un protocollo a finestra. Le unità dati si chiamano **segmenti**.

Un header TCP è formato da minimo 20 byte:

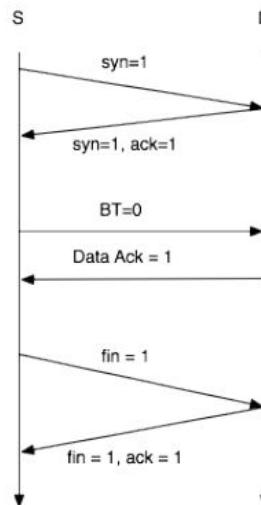


Dove:

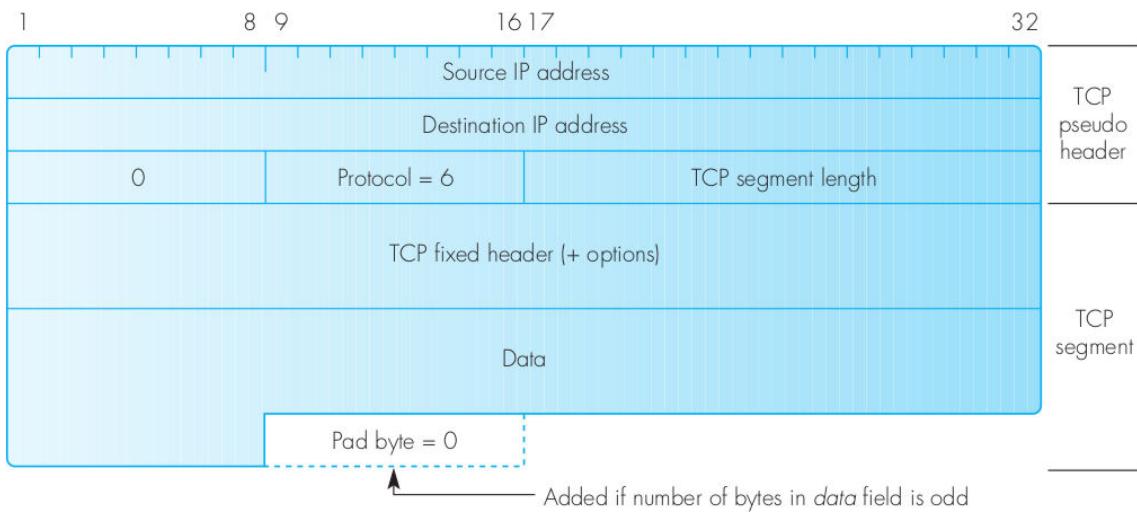
- **Source port** (16 bit), identifica il numero di porta sull'host mittente associato alla connessione TCP
- **Destination port** (16 bit), identifica il numero di porta sull'host destinatario associato alla connessione TCP
- **Sequence Number** (32 bit), numero del segmento del dato da trasferire. Numera i byte, perché TCP considera payload come stringhe di byte. È puntatore al primo byte nel campo dati (TCP vede la connessione come un flusso dati)
- **Acknowledgement Number**, (32 bit) numero del segmento che ho correttamente ricevuto; per convenzione TCP lato mittente valida il byte successivo a quello che è stato ricevuto correttamente. Questo campo ha significato solo se il campo ACK è impostato a 1. È usato solo dalla destinazione.

[Sequence Number e ACK Number garantiscono l'affidabilità del protocollo]

- **Header length**, (data offset) numero di parole da 32 bit presenti nel campo Options (l'header ha una lunghezza variabile)
- **Reserved** (4 bit), bit non utilizzati e predisposti per sviluppi futuri del protocollo
- **Code bits** (6 bit), maschera di 6 bit che identifica il tipo di segmento. Più bit possono essere settati a 1, aggiungendo più informazioni in un singolo segmento. Possibili valori:
 - SYN: stabilisce una connessione
 - ACK: conferma della validità del campo
 - FIN: chiude la connessione
 - PHS: invio dei dati alla ricezione di questo segmento
 - URG: urgente
 - RST: rifiuta una richiesta di connessione o ripristina i valori iniziali del valore di sequenza.A seconda del tipo di segmento che sto mandando, sono messi a 1 i bit corrispondenti.



- **Window Size** (16 bit), anche Advertise Window, è la larghezza della finestra per spedire i frammenti. Viene usata per negoziare lo spazio di memoria disponibile tra mittente e destinatario. La sua dimensione è decisa a priori prima di iniziare la comunicazione, è scelta in base alla dimensione massima di trasferimento del canale in uso e del buffer lato destinazione. Si usa per evitare la frammentazione.
- **Checksum**, contrariamente alla checksum di IP che riguardava solo l'header, questa riguarda tutto il segmento, cioè header + payload. Inoltre, per aggiungere un livello di controllo in più, alcuni campi del pacchetto IP (mostrati in figura) sono inclusi nella generazione di questa checksum, e formano il cosiddetto **TCP pseudo header**



Entrambi gli pseudo header (sorgente e destinazione, uno costruito localmente e l'altro con i parametri che giungono nel primo segmento) vengono mantenuti localmente per la composizione dei pacchetti successivi e, inoltre, per sicurezza, evitando attacchi MITM (Man In The Middle).

Lo pseudo header non viene spedito, ma viene costruito localmente da ogni macchina.

Osservazione: se non viene definita a priori la massima dimensione del segmento, di default è 536 B

- **Urgent pointer** (16 bit), denota il numero di byte nel campo dati che deve essere subito consegnato all'applicazione che sta comunicando. Questi byte sono noti come **urgent data**, o **expedited data**. Ha significato solo se il flag URG è settato a 1 ed indica lo scostamento in byte a partire dal Sequence Number del byte di dati urgenti all'interno del flusso.
- **Options**, usato per decidere, per esempio, la dimensione di un segmento (MSS – Maximum Segment Size): infatti visto che la comunicazione è end-to-end, la dimensione massima può essere scelta liberamente. Se la dimensione non viene specificata, viene scelto 536 byte: comando i due header, TCP e IP, **ottengo i famosi 576 byte**
- **Payload**, rappresenta il carico utile da trasmettere

Comunicazione TCP

Il client fa praticamente solo 2 cose:

- Crea socket
- Crea connessione dove indica IP, porta sorgente e destinazione

Il server invece

- Crea socket
- Bind: unisce l'indirizzo IP del processo alla porta interessata
- Listen: dichiara il numero di *fork* massime che potrà fare
- Accept: diventa passivo in attesa di una comunicazione client
- ..
- Fork: crea un figlio con una nuova porta (random) e fa gestire a lui la richiesta del client.

La comunicazione TCP è distribuita su tre fasi:

- apertura della connessione (o **connect - three way handshake**)
- trasferimento dati (o **Data Transfer**)
- chiusura connessione (o **Close**)

TCP nasce dall'esigenza di identificare in modo univoco una sessione tra una coppia di interlocutori. Per ogni coppia di interlocutori esiste un canale virtuale (detta **connessione**: sessione identificata univocamente su tutta la rete).

La connessione viene creata con uno scambio esplicito di informazioni di inizializzazione.

Apertura della connessione

È caratterizzata da uno scambio di segmenti prefissati, cioè:

1- Il client manda un segmento con bit SYN attivo

Quando si apre una connessione (primitiva `connect()`), ogni entità TCP deve scegliere un numero di sequenza iniziale *ISN* (*Initial Sequence Number*): essi sono entrambi diversi da 0 e cambiano in ogni connessione, garantendo così che un segmento relativo a una specifica connessione non interferisca con un'altra (es. quella successiva). Ogni volta che viene inizializzata una nuova connessione viene “consumato” un diverso numero di sequenza.

Il numero di sequenza viene scelto in base ai bit meno significativi del clock.

Al momento della connessione, gli interlocutori si scambiano informazioni di inizializzazione, per esempio la taglia massima del TDU (Transfer Data Unit).

Per instaurare una connessione, il client invia un primo segmento contenente nel campo Code Bits il bit SYN=1 (tutti gli altri a 0) ed il numero di sequenza scelto.

2- Il server risponde con SYN+ACK, cioè dice al mittente che ha accettato il suo SYN.

Il server va inizializzato con la primitiva `listen()`, che specifica quante connessioni attive può gestire contemporaneamente e successivamente `accept()`, che mette il server in attesa della prima connessione.

Nel caso in cui il server non sia in `listen`, al client viene inviato un segmento con bit RST attivo. Alla ricezione, la procedura di connessione verrà interrotta.

Lato server viene fatta una `fork()` (se non si è superato il numero massimo di connessioni da gestire) che genera un processo figlio che gestisce la connessione con quel client. Il “server figlio” deve trovare il suo numero di sequenza e spedirlo al client insieme all’ACK del segmento successivo (inizializzando il proprio segmento con SYN=1 e ACK=1).

3- Il client manda un segmento con bit ACK attivo, per validare il SYN+ACK del server.

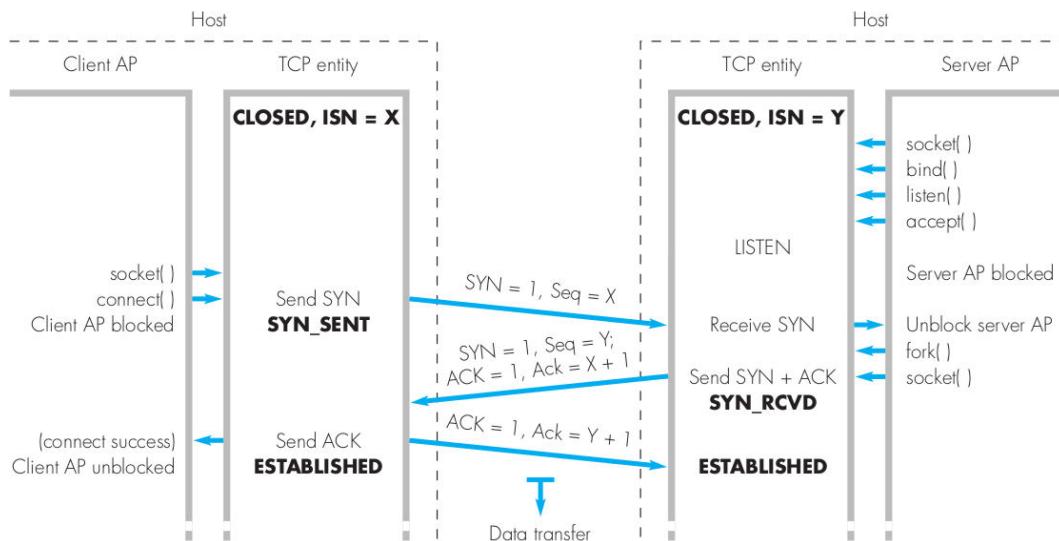
Il client risponde convalidando il segmento del server e a questo punto la connessione è stabilita.

Osservazioni:

Gli ISN vengono validati attraverso i due ACK.

I numeri di sequenza disponibili sono 2^{32}

L’apertura consuma un Byte, anche se non viene spedito nessun byte dati, da ciò si deduce che la conferma di ciascun segmento equivale al trasferimento di un byte di dati in ciascuna direzione. Questa apertura è detta Three Way Handshake (bisogna scambiare almeno 3 segmenti).



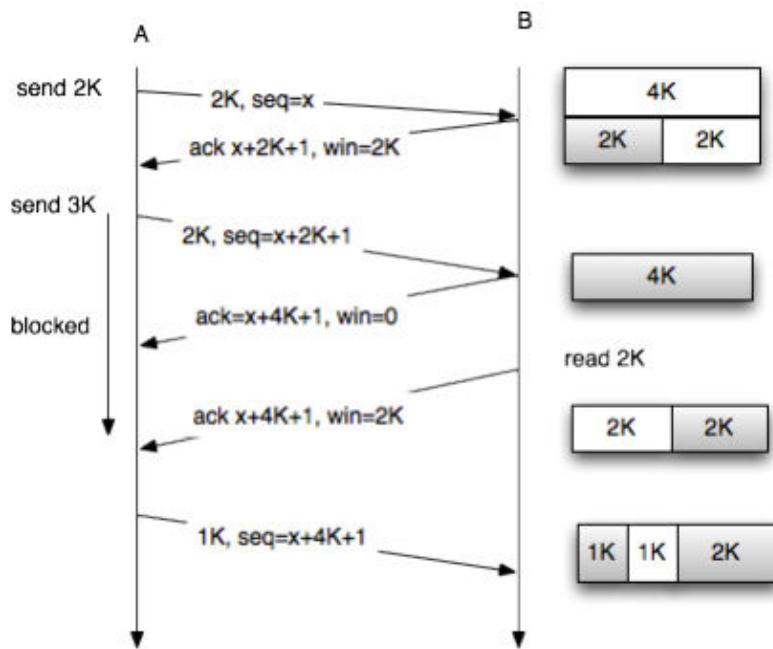
Controllo di flusso

Consideriamo il caso in cui il sender TCP invii segmenti più velocemente rispetto ai tempi di lettura e di processing del ricevente.

Per fare in modo che il mittente non invii dati se il destinatario non è in grado di ricevere, si utilizza il campo **Window Size** nel segmento TCP, il quale indica il numero di byte che si è in grado di ricevere (W_R), ed è determinato dallo spazio disponibile nel buffer di ricezione.

In particolare lo schema adottato è il seguente:

- lato mittente viene mantenuta una variabile, W_S , contenente il valore della dimensione del buffer lato ricezione W_R , cioè pari al numero di byte che è possibile inviare;
- lato ricezione viene mantenuta W_R , inizializzata alla dimensione del buffer, e indica il numero di byte in grado di ricevere;
- finché $W_S \geq 0$ e ci sono dati da inviare, il mittente può spedire segmenti decrementando di volta in volta il valore di W_S sottraendo il numero di byte inviati;
- man mano che il buffer lato ricezione si riempie, W_R viene decrementato;
- man mano che invece i dati vengono letti dal buffer del ricevitore (dal livello superiore), il valore della variabile W_R viene incrementato;
- il destinatario, dopo aver inviato un ACK indicante la dimensione della sua finestra pari a zero, legge i dati dal buffer decrementandone il valore. In questo modo il ricevente è di nuovo in grado di ricevere i segmenti. Per comunicare che è di nuovo disponibile, viene inviato al sender un nuovo ACK contenente il valore di W_R aggiornato, questo segmento è detto **window update**



In figura, lato ricevente, i blocchi evidenziati in grigio identificano la parte di memoria che viene occupata nel buffer di ricezione. La massima dimensione di trasferimento è di 4K. Quindi all'inizio la $W_R=4K$.

Dopo il primo messaggio inviato dal client (send 2K), il buffer del ricevente diventa 2K. Il sever comunica al client che la nuova dimensione del suo buffer è 2K. Questo messaggio di ACK contiene l'informazione $WIN = 2K$.

Il client ha ora da inviare 3K (nella figura, send 3K). La massima dimensione trasferibile è però di 2K, quindi si inviano 2K.

Quindi dopo aver ricevuto il secondo messaggio, il buffer del ricevente si riempie ma manca ancora 1 K da inviare.

Il ricevente manda un WIN_UPDATE al client informandolo che non ha più memoria in ricezione.

Non appena il buffer di ricezione diventa nuovamente libero, il ricevente invia un window update aggiornando il mittente.

Nell'esempio il secondo ACK viene inviato non appena la finestra di ricezione ha a disposizione o la metà del buffer oppure uno spazio libero pari alla MSS (Maximum Segment Size, che è in questo caso pari a 2K). Ineffetti in questo esempio i due valori coincidono.

Il mittente una volta ricevuta l'informazione che il ricevente è disponibile a ricevere, invierà l'ultimo messaggio (1K).

Quando la capacità del canale è illimitata, l'unico limite è la capacità del buffer di ricezione.

A ogni segmento scambiato, il server comunica al client la disponibilità di memoria nel buffer tramite il campo Window Size.

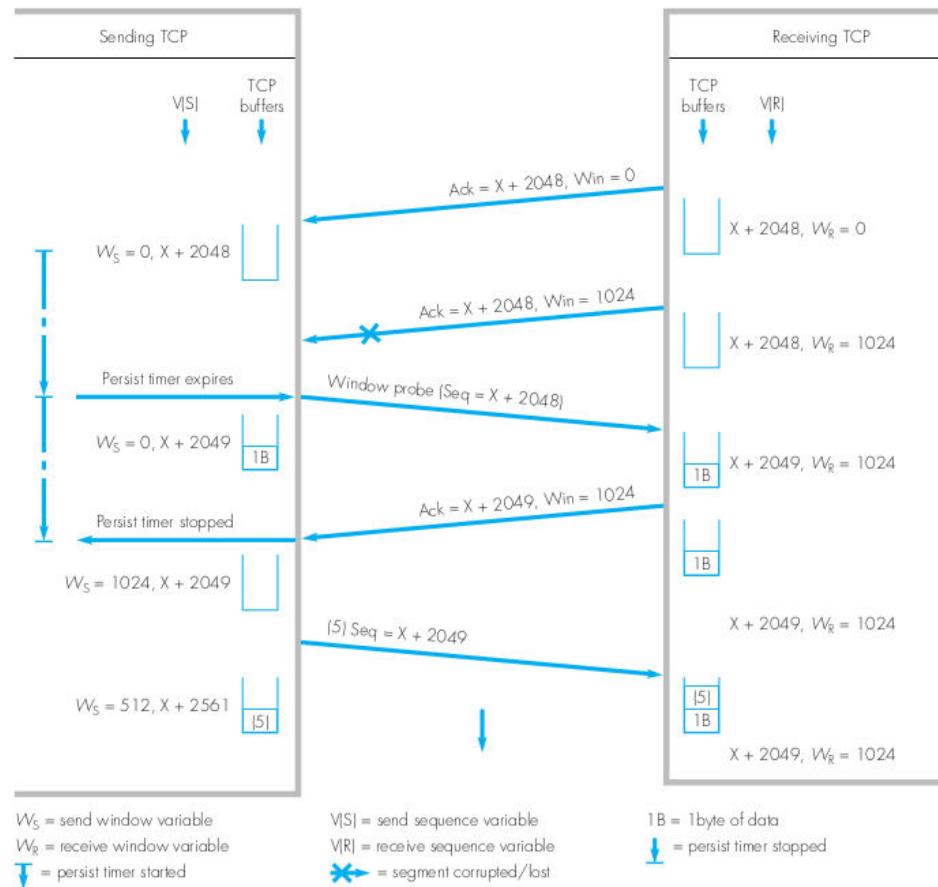
Quando il buffer è pieno, il sender si blocca finché l'applicazione lato ricevente non legge dal buffer. A quel punto verrà avvisato che può spedire.

In caso di perdita dell'ACK che avvisa il sender che il buffer è diventato libero, esistono due strategie di recovery:

1-Persistent Timer (salvaguarda il client)

Ogni volta che viene ricevuto un segmento contenente $win_size = 0$, il sender viene bloccato e viene fatto partire un timer.

Se un segmento contenente un aggiornamento della finestra non viene ricevuto prima della scadenza del timer (parliamo quindi del windows update), TCP client, tramite il segmento *window probe*, richiede esplicitamente al server l'ACK perduto. Dopo un numero predefinito di tentativi, in caso di non risposta dal server, il client chiude la connessione.



Dato che il window update è stato perso (vedi figura sopra) ci troviamo in una situazione di *deadlock* in quanto entrambi i lati stanno aspettando l'altra parte. Per evitare che questa situazione si presenti, **ogni volta che il mittente setta la sua variabile W_S a zero, viene fatto partire un timer, allo scadere del quale, se non è stato ricevuto un window update, viene inviato il segmento detto window probe**. A seguito della ricezione di un segmento probe, viene inviato un ACK che indicherà la dimensione della finestra, se essa sarà ancora pari a zero il timer ripartirà, altrimenti ricomincerà il flusso di dati. Questa procedura viene ripetuta ogni 60 s fino a che non viene ricevuto un valore della finestra diverso da zero o la connessione viene chiusa.

2-Keep Alive Timer (salvaguarda il server)

Durante la connessione, allo scadere di un timer, il server periodicamente spedisce un *keep alive probe*, che controlla che il client sia ancora attivo. Dopo un numero predefinito di non risposte, il server chiude la connessione, evitando quindi di tenere aperte connessioni inesistenti.

Per ogni connessione aperta il timer è settato di default a 2 ore, e se durante questo tempo non vengono ricevuti dati dal client, il server invia un segmento di probe e setta il timer a 75 s. Se il client è ancora attivo risponderà con un ACK, altrimenti il timer del server scadrà ed esso reinvierà il segmento di probe con timer a 75 s. Questa procedura viene ripetuta 10 volte, dopo di che, in mancanza di ACK, la connessione viene chiusa.

Esercizio

Un host TCP sta inviando finestre di 65535 B su un canale da 1 Gbps che ha ritardo one-way di 10 msec. Qual è il max throughput ottenibile? Qual è l'efficienza della linea?

SOL:

Si invia una finestra ogni 20 msec,

perciò il MAX throughput è di 50 finestre al sec [$20 \text{ msec} = 0.020 \text{ s}, 1 \text{ s} / 0.02 = 50$]

Quindi: $65.535 \times 50 = 3.276.750 \text{ Bps}$ (= 3,3 MBps) [è massimo n° di bit al secondo che possono essere trasmessi]

$3.276.750 \text{ Bps} \rightarrow 26,214 \text{ Mbps}$ [= $(3.276.750 \cdot 8)$ conversione da byte a bit per sec].

L'efficienza della linea è : $26,214 \text{ Mbps} / 1000 \text{ Mbps} = 2.621\%$

[Efficienza = MAX bit rate o throughput / bwth]

Ottimizzazioni TCP Data Transfer

Ottimizzazioni sui segmenti piccoli

Consideriamo ora il caso in cui un server trasferisca una grande quantità di caratteri ad un client il quale è però in grado di leggere solo un carattere alla volta. Dopo aver ricevuto il primo blocco di caratteri, il client, risponderà con un ACK indicando la dimensione della finestra pari a zero, **successivamente ogni volta che un carattere sarà letto dal buffer, invierà al server un window update per indicargli che può inviare un nuovo carattere.**

Ogni volta che il carattere sarà ricevuto, il buffer si riempirà nuovamente e sarà inviato un ACK indicante la dimensione della finestra pari a zero. Dopo aver letto un nuovo carattere dal buffer, sarà inviato un window update informando il server che il client è disponibile.

In questo modo **un solo carattere alla volta può essere inviato**. Questa procedura è detta *silly window syndrome* e viene risolta utilizzando l'*algoritmo di Clark*. Vediamo un esempio.

Esempio TELNET

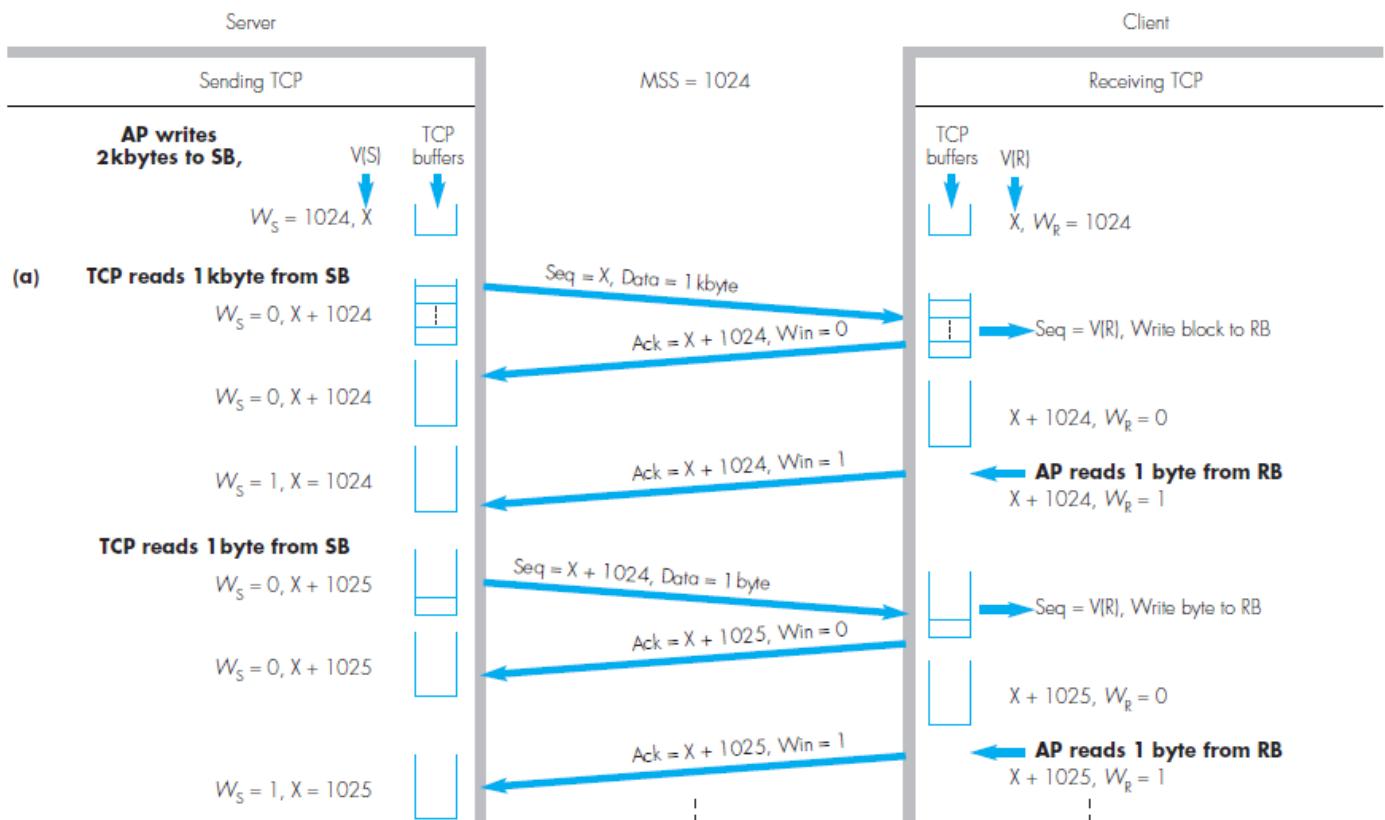
Telnet è un'applicazione che permette di usare un editor di testo remoto. Il client invia un carattere alla volta e il server conferma ogni singolo carattere rispedendolo indietro (*echo*).

La comunicazione, in una sessione Telnet, è composta solamente da segmenti con lunghezza minore di MSS. Quindi, possono essere attuate ottimizzazioni per abbassare la quantità di segmenti ed evitare il traffico di segmenti con bassissima percentuale di byte utili.

In una normale sessione Telnet, quando l'utente digita un carattere, questo viene subito inviato al server, che appena lo riceve lo rispedisce al mittente (*echo*). Il mittente una volta ricevuto il carattere, lo mostra sul terminale (quindi ogni segmento Telnet ha *code bit PSH* attivo).

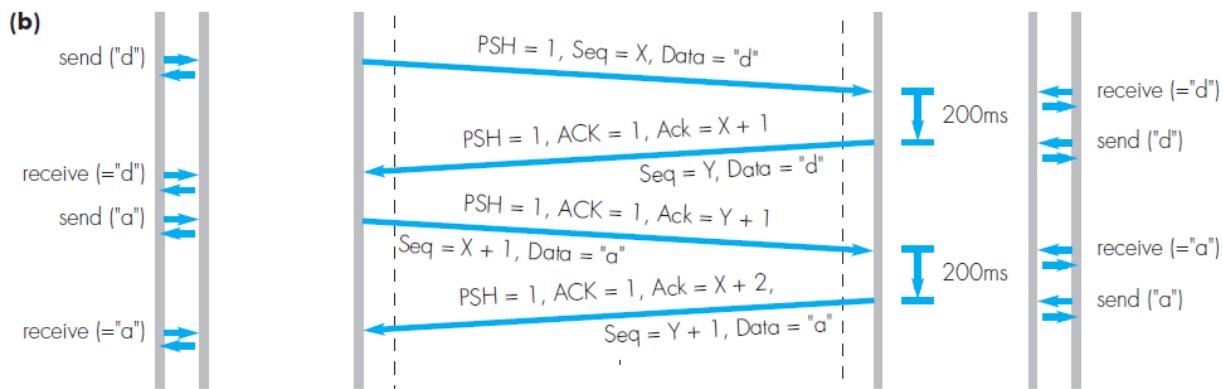
Oltre a questi due segmenti (uno di invio e l'altro di ricezione del carattere), altri due segmenti ACK vengono inviati (vedi figura). Un ACK per informare il server che la finestra è piena e il successivo ACK per informare il server che la finestra è di nuovo libera.

In termini di performance, per un singolo carattere (1 byte) vengono spediti 4 segmenti che contengono ognuno 40 byte (se considerati sia header TCP che header IP).



Delayed acknowledgements

Un primo miglioramento, noto come *delayed acknowledgements*, consiste nel far attendere il server un tempo prefissato (200ms) dopo la ricezione del carattere; in questo modo, gli eventuali dati che in questo tempo arrivano sul buffer di invio del mittente potranno essere raggruppati e inviati assieme all'ACK del carattere precedente correttamente ricevuto. Vengono così ridotti significativamente il numero di segmenti necessari per la comunicazione.



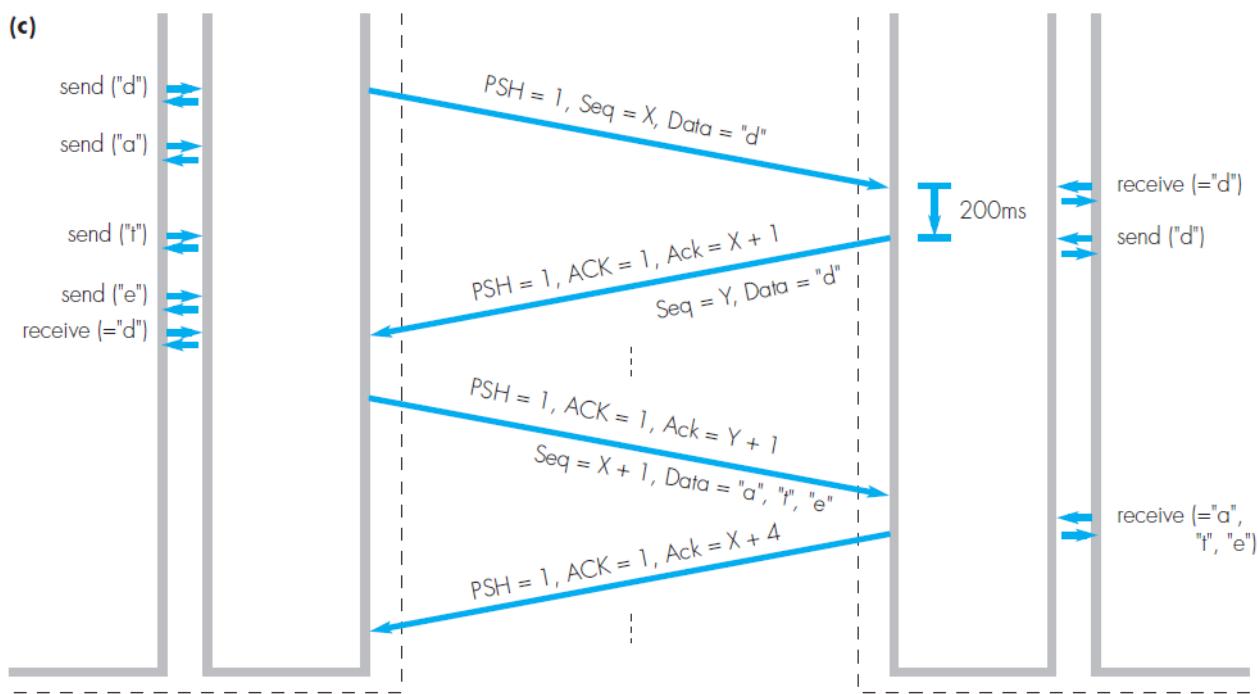
Questo metodo è però accettabile con un basso RTT, come quello presente all'interno di una LAN, altrimenti il ritardo tra l'invio e la comparsa del carattere sullo schermo del client diventerebbe troppo grande e potrebbe risultare oneroso per l'utente.

Algoritmo di Nagle

Viene quindi usata una versione modificata del delayed acknowledgement, chiamato *algoritmo di Nagle* (RFC 896).

Secondo questo algoritmo, ogni entità TCP può avere solo un “segmento piccolo” alla volta, in attesa di ACK.

In questo modo, mentre l'entità TCP lato server aspetta la ricezione dell'ACK, il buffer d'invio si riempie così che, quando sarà nuovamente possibile inviare (cioè alla ricezione dell'ACK spedito dal client), più dati utili potranno essere inviati in un solo segmento.



Esercizio

Si consideri una connessione TCP con applicazione interattiva a lato sender che produce 1B di dati ogni 20 msec e con link avente banda di 100 Kbps e tempo di propagazione di 100 msec. Dire quali sono le dimensioni dei primi 3 segmenti inviati sulla connessione, se TCP adotta l' algoritmo di Nagle.

SOL: Il **primo segmento** include 20B header TCP e **1B dati**.

Viene trasmesso in $Tx = (21 \times 8) \text{ bit} / 100000 \text{ bps} = 1.68 \text{ msec}$. [omessa semplificazione zero]

L'ack è ricevuto a sorgente dopo $Tx + 2 \text{ Tp} = 1.68 + 2 \times 100 = 201.68 \text{ msec}$.

-----In questo tempo la sorgente ha accumulato $[201.68/20] = 10$ caratteri.

Il **secondo segmento** comprende 20B header TCP + **10B dati**,

che vengono trasmessi in $[Tx=(30 \times 8)/100000] = 2.4 \text{ msec}$.

Il relativo ack è ricevuto a sorgente dopo $Tx + 2 \text{ Tp} = 2.4 + 2 \times 100 = 202.4 \text{ msec}$.

-----In questo tempo la sorgente ha accumulato $[202.4/20] = 10$ caratteri.

Quindi il **terzo segmento (e tutti i successivi)** comprenderanno **10B dati**.

Algoritmo di Clark

Un'altra soluzione (al problema *silly window syndrome*) è l'algoritmo di Clark: consiste nel compattare i window update lato server, cioè ritardare la spedizione del windows update finché il buffer del server ha raggiunto la MSS oppure se raggiunge la metà della memoria disponibile. In questo modo il ricevente non informerà mai il mittente se la dimensione della finestra è piccola, e quest'ultimo non invierà mai segmenti contenenti una quantità di dati ridotta.

Esercizio

Una connessione TCP che adotta l'algoritmo di Clark ha dimensione di receiver window 3800B e MSS 1200B. Se a lato receiver c'è un'applicazione interattiva che consuma 1B ogni 40 msec., e a lato sender c'è sempre disponibilità di dati da spedire, dire ogni quanto il receiver invia un window update, assumendo che i ritardi di propagazione siano trascurabili.

SOL

TCP che usa Clark su lato receiver.

È data un'applicazione che legge poco, vogliamo ridurre il numero di WIN_UPDATE

Abbiamo come capacità del buffer di ricezione pari a 3800 B e MSS = 1200 B

Qual è il minimo fra la metà del buffer e la MSS?

$\text{Min}(\text{buffer}/2 ; \text{MSS}) = \text{MSS}$

$3800/2 = 1900 > 1200$

A buffer pieno, 1200B sono consumati in:

$40 \text{ msec} \times 1200 = 48 \text{ sec}$ che è la frequenza con cui è generato un window update;

Controllo degli errori

Il controllo degli errori di TCP è molto simile a quello effettuato da HDLC. La differenza più grande è che in HDLC i numeri di sequenza in invio e in ricezione si riferivano a blocchi di dati, mentre in TCP si riferiscono a byte dello stream totale.

Inoltre, dato che il Round Trip Time è molto più grande in Internet che su un singolo link (dove opera HDLC), la dimensione della finestra non è derivabile dai numeri di sequenza.

Esiste invece un campo specifico nell'header TCP, chiamato *window size advertisement*, che indica ad ogni passo il massimo numero di byte che possono essere ricevuti.

TCP usa ACK cumulativi e non usa NACK. Ogni segmento è validato da un ACK e per ciascuno di essi esiste un timer che viene resettato ogni volta che riceve un ACK.

Fast Retransmit

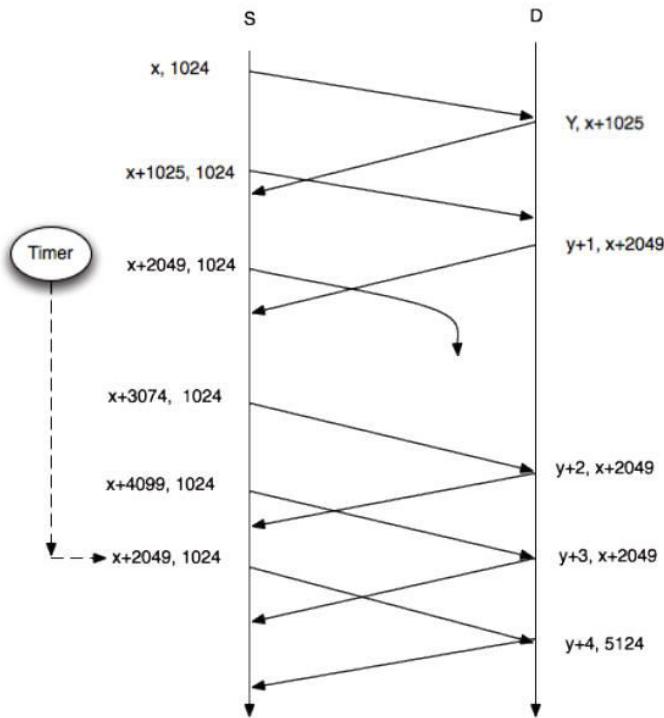
Dato che i segmenti inviati relativi ad una comunicazione TCP viaggiano sulla rete Internet i percorsi seguiti possono essere diversi: conseguentemente i segmenti potrebbero non arrivare nell'ordine nel quale sono stati inviati.

Quando al ricevente arriva un segmento fuori sequenza, esso si limita a memorizzarlo temporaneamente e ad inviare un ACK indicando il segmento che si aspetta. Normalmente il segmento mancante arriva entro un breve intervallo di tempo, e a quel punto il ricevente risponderà con un ACK che validerà tutti i segmenti ricevuti fino ad esso. L'approccio scelto è quindi quello di ACK cumulativo.

Questa situazione lato sender si presenta come la ricezione di un ACK che indica la perdita di un segmento seguito dopo un piccolo intervallo di tempo da un secondo ACK che valida tutti i segmenti inviati. Sarebbe quindi desiderato che la ritrasmissione del pacchetto non avvenga alla ricezione del primo ACK fuori sequenza. Per questo si è scelto di aspettare la ricezione di tre ACK fuori sequenza prima di reinviare il segmento mancante. Il numero 3 è dato da un'euristica.

Questa tecnica della ricezione dei tre ACK è detta *Fast Retransmit* e comprende anche l'utilizzo di un timer di ritrasmissione (RTO) associato ad ogni segmento inviato lato sender.

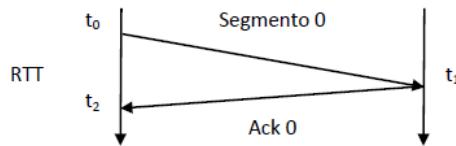
Al termine dell'RTO, se non si è ancora ricevuto l'ACK, il segmento mancante viene ritrasmesso. Il timer è necessario anche nel caso in cui il mittente non ha più segmenti da inviare, poiché non si accorgerebbe del segmento mancante perché non riceverebbe più ACK.



Osservazione: se per ritrasmettere un segmento andato perso, aspettassi lo scadere del timer RTO, potrei ricoverare troppo tardi l'errore.

RTO - Il Timer TCP

A livello 4 non posso conoscere in modo esatto i tempi di percorrenza della rete. Devo quindi trovare un modo per stimarli.



RTT cambia infatti in base a:

- Malfunzionamenti della trasmissione
- Stato di congestione della sottorete

Quanto tempo aspetto prima di rimandare il segmento nel caso non ho ricevuto l'ACK? Ovvero, come dimensiono l'RTO (*Retransmission Time-Out*)?

$$RTO = RTT + \text{offset di tempo}$$

Quanto dimensiono l'*offset*? ... se troppo grande spreco tempo!

Una corretta scelta è indispensabile per una buona efficienza delle procedure di controllo degli errori, ma calcolarlo non è semplice a causa del gran numero di variabili in gioco.

Sappiamo che RTO dipende direttamente dal RTT e che quest'ultimo è molto variabile quando si tratta di Internet. Allora, RTO deve essere calcolato dinamicamente e variare durante la connessione.

Inizialmente si parte con un valore di RTO relativamente elevato

Come decido il primo e il secondo RTO di una connessione visto che non possiedo tutte le misure RTT precedenti?

- Il primo RTO di default è circa 3 secondi
- Il secondo RTO si attiva alla ricezione del primo ACK, vedi la formula:

$$RTO = RTT + K(D) \quad , K = 4 \quad e \quad D = \frac{RTT}{2}$$

D è l'approssimazione della deviazione standard sull'RTT

Se RTT = M equivale a RTO = 3M

Vediamo ora come si calcolano i prossimi RTO.

Una volta a regime (superati i primi 2 ACK della trasmissione), ogni volta che un ACK viene ricevuto prima che il timer scada, verrà effettuata una nuova misura del RTT. Questo valore viene utilizzato per aggiornare la stima usando la seguente formula:

$$RTT_{new} = \alpha \cdot RTT_{old} + (1 - \alpha)M$$

M indica la misura di RTT appena effettuata, equivale al tempo in cui ricevo ACK per il segmento inviato (il terzo ACK seguendo l'esempio). α è un valore di "levigamento" che determina l'influenza di M sul calcolo del nuovo RTT. Il valore di α consigliato è 7/8 (valore ottenuto sperimentalmente: ~ 0.9).

Il nuovo valore di RTO viene calcolato utilizzando sia RTT che la sua deviazione standard:

$$D = \beta \cdot D + (1 - \beta)|RTT - M|$$

Anche il valore consigliato di β è 7/8.

Il nuovo valore di RTO allora diventa:

$$RTO = RTT + 4D$$

Osservazione: Per ogni RTO minore di un secondo, RTO viene fissato ad 1 secondo (quindi RTO non può mai essere minore di 1 sec)

Nb. Sono da calcolarsi in ordine:

$$D_{new} = \beta \cdot D_{old} + (1 - \beta)|RTT_{old} - M|$$

$$RTT_{new} = \alpha \cdot RTT_{old} + (1 - \alpha)M$$

$$RTO = RTT_{new} + 4D_{new}$$

Alpha (o Beta) = 7/8, è detto fattore di smorzamento

Osservazione: questa tecnica funziona bene solo se non ci sono errori.

Esercizio

Se il RTT di TCP è correntemente 30 msec RTO è 38 msec e i successivi ack arrivano dopo 26 msec e 32 msec dalle rispettive trasmissioni, quali sono le nuove stime di RTT e RTO usando $\alpha=0.9$?

SOL

Abbiamo:

RTT_old = 30 sec

RTO_old = 38 sec

L'ACK per segmento 1 (S1) arriva dopo 26 secondi

L'ACK per segmento 2 (S2) arriva dopo 32 secondi

Ci viene chiesto di calcolare RTT e RTO dopo che è arrivato il segmento 1 e dopo il segmento 2.

Notiamo che la connessione TCP non è allo stato iniziale, quindi non dobbiamo partire con un RTT e RTO approssimato ma sono valori calcolati già in base alle stime che abbiamo studiato.

In questo esercizio è fornito l'RTT e RTO, manca però il valore D che è un dato che dobbiamo calcolare per poter effettuare le stime dei nuovi valori. Dai dati che abbiamo ricaviamo D dalla formula:

$$RTO = RTT + K \left(\frac{RTT}{2} \right) \quad , K = 4$$

E presupponendo RTT/2 = D, abbiamo:

$$RTO = RTT + 4D$$

$$RTO - RTT = 4D$$

$$D = \frac{38 - 30}{4} = 2$$

Ora effettuiamo le stime con le formule canoniche:

$$D = \beta \cdot D + (1 - \beta)|RTT - M|$$

$$RTT_{new} = \alpha \cdot RTT_{old} + (1 - \alpha)M$$

$$RTO = RTT + 4D$$

Quindi ricordando che si ha ricevuto l'ACK del primo segmento dopo M = 26 msec, abbiamo:

S1

$$D = 0.9 * 2 \text{ sec} + 0.1 * |30 - 26| = 1.8 + 0.4 = 2.2$$

$$RTT = 0.9 * 30 + 0.1 * 26 = 27 + 2.6 = 29.6$$

$$RTO = 29.6 + 4 * 2.2 = 38.4$$

Notiamo che il nuovo valore di RTO è aumentato (38.4 msec appena calcolati, mentre prima era 38 msec).

Si continua con la stima del successivo RTO dopo aver inviato il secondo segmento, e quindi dopo averne ricevuto l'ACK (M=32 msec)

S2

$$D = 0.9 * 2.2 + 0.1 * |29.6 - 32| = 1.98 + 0.24 = 2.22$$

$$RTT = 0.9 * 29.6 + 0.1 * 32 = 26.64 + 3.2 = 29.84$$

$$RTO = 29.84 + 4 * 2.22 = 38.72$$

Notiamo che il nuovo valore di RTO è aumentato ancora.

Esercizio

Una connessione TCP **trasmette i primi segmenti S1 e S2**. Indicare quali sono i valori di RTT e RTO:

- i. usati per la trasmissione con successo di S1 che genera una misura di RTT pari a 41 msec.
- ii. usati per la trasmissione con successo di S2

SOL: La soluzione è determinata considerando quanto stabilito dallo RFC 2988, secondo il quale:

1. inizialmente RTO è posto uguale a 3 sec.
2. dopo la prima misura M si pone RTT = M; D = M/2; RTO = RTT + 4D = 3M
3. per le misure successive l'aggiornamento è eseguito con le formule canoniche di stima dell'RTO.

Per la trasmissione (i), l'RTT non è disponibile mentre RTO = 3 sec.

Per la trasmissione (ii) alla ricezione dell'ack di S1, l'aggiornamento è effettuato come descritto. Dunque:

$$D = 41/2 = 20.5, \text{ RTT} = 41, \text{ e } RTO = 41 + 4 \times 20.5 = 123 \text{ msec.}$$

Esercizio

Sapendo che l'RTT del segmento 0 era di 20 ms e che la Deviazione standard D era 2, trovare RTT, D e RTO del segmento 1 sapendo che la misura del tempo tra l'invio del segmento e la ricezione è stata di 30.

SOL

Usiamo le formule:

$$D = \beta \cdot D + (1 - \beta)|RTT - M|$$

$$RTT_{new} = \alpha \cdot RTT_{old} + (1 - \alpha)M$$

$$RTO = RTT + 4D$$

$$D = 0.9 \times 2 + 0.1 \times |20 - 30| = 1.8 + 1 = 2.8$$

$$RTT = 0.9 \times 20 + 0.1 \times 30 = 21$$

$$RTO = 21 + 4 \times 2.8 = 32.2$$

nb. In questo esempio, viene fornita la misura della deviazione standard D, la stima di RTT (cioè M) del segmento 0. Quindi possiamo risolvere i nuovi valori del segmento 1 utilizzando subito le formule della stima di RTO.

Esercizio

Se il RTT di TCP e' correntemente 30 msec e i seguenti ack arrivano dopo 26 msec, 32 msec e 24 msec dalle rispettive trasmissioni, qual e' la nuova stima di RTT usando $\alpha=0.9$?

SOL: in questo esercizio non sono forniti e tanto meno richiesti i parametri RTO e D, si vuole sapere che valori di RTT si avranno nelle trasmissioni successive. Ci basta la formula:

$$RTT_{new} = \alpha \cdot RTT_{old} + (1 - \alpha)M$$

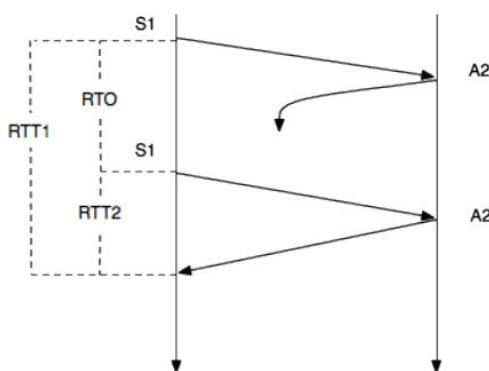
$$RTT_1 = 0.9 \times 30 + 0.1 \times 26 = 29.6$$

$$RTT_2 = 0.9 \times 29.6 + 0.1 \times 32 = 29.84$$

$$RTT_3 = 0.9 \times 29.84 + 0.1 \times 24 = 29.256$$

Retransmission ambiguity problem

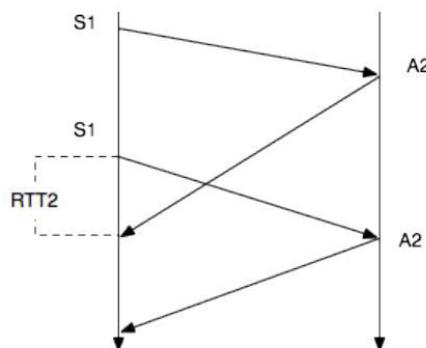
I calcoli dell'RTO visti precedentemente non sono più attendibili se un segmento viene perso, o comunque in caso di errore. Per esempio, se il mittente non riceve ACK su un segmento, dopo aver aspettato RTO lo rimanda. Ma l'RTT del segmento qual è? E quello totale? E il tempo della seconda trasmissione?



Ho due possibilità: se scelgo RTT1 stimo per eccesso RTT.
Se scegliersi RTT2 avrei lo stesso dei problemi

Se scelgo di considerare solo il tempo della ritrasmissione, ci sono altri problemi: se la rete è in stato di congestione, può capitare che RTO scada prima che l'ACK mi arrivi.

Se considero come nuovo RTT il tempo della sola ritrasmissione finisco per prendere il tempo in cui mi arriva l'ACK del segmento precedente, non di quello corrente, ed è quindi un dato completamente sbagliato.



Se ho la rete congestionata non perdo nessun segmento ma ritrasmetto più volte lo stesso perché scade RTO, quindi stimo un RTT troppo piccolo (e quindi sbagliato).

Nota: Ci siamo trovati in questa situazione perché si è scelto un RTO troppo breve.

Per non avere questi problemi (*retransmission ambiguity problem*), è adottata la politica di Karn.

Timer di Karn

Se si verifica un errore, il nuovo RTO è il doppio di quello vecchio.

Dopo riparto come se avessi ricevuto il primo ACK.

Quindi con la formula:

Esercizio

Una connessione TCP ha RTT=27 msec e RTO = 35 msec quando si verifica un retransmission timeout sulla trasmissione del segmento S1.

Indicare quali sono i valori di RTT e RTO:

- usati per la ritrasmissione con successo di S1 che supera la condizione di errore.

SOL: Per la ritrasmissione (i), in accordo a Karn, il RTT non viene aggiornato mentre RTO è raddoppiato, ovvero RTO=70 msec.

Poi, per la seconda trasmissione si usa la formula:

$$RTO = RTT + K \cdot (RTT/2)$$

Opzione TimeStamp

Abbiamo assunto che la misurazione effettuata per il nuovo calcolo dell'RTT venga effettuata per ogni segmento inviato; questo è vero solo per alcune implementazioni. Nella realtà per finestre piccole (quindi connessioni brevi), TCP fa una sola stima iniziale e basta; per finestre grandi invece tale stima viene fatta ogni 2 o 3 segmenti trasmessi. Se voglio avere stime più frequenti, va dichiarato in fase di apertura della connessione specificando l'opzione **time stamp** nell'header TCP.

La richiesta di utilizzo dell'opzione viene effettuata dall'entità TCP che esegue un'apertura attiva della connessione, includendo l'opzione time-stamp nel segmento SYN. La richiesta viene accettata dal destinatario includendo la stessa opzione nel segmento SYN restituito.

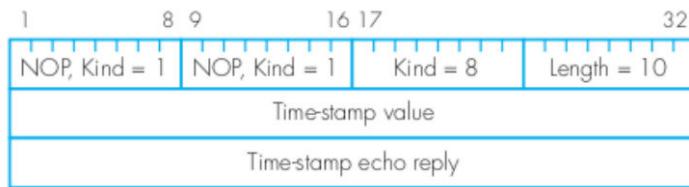
Ogni entità TCP possiede un time-out a 32 bit incrementato ad intervalli compresi tra 1 ms ed 1 s. Per ogni segmento inviato il mittente legge il valore del timer e lo inserisce nel segmento in corrispondenza del campo *time-stamp value*.

Quando il destinatario riceve il messaggio, risponderà con il rispettivo ACK e includerà nel campo *time-stamp echo reply* il valore letto nel campo *time-stamp value* del segmento ricevuto così che il mittente possa misurare il valore dell'RTT sottraendo al valore corrente il valore ricevuto (ottenendo così il tempo effettivo con cui è stato inviato il segmento).

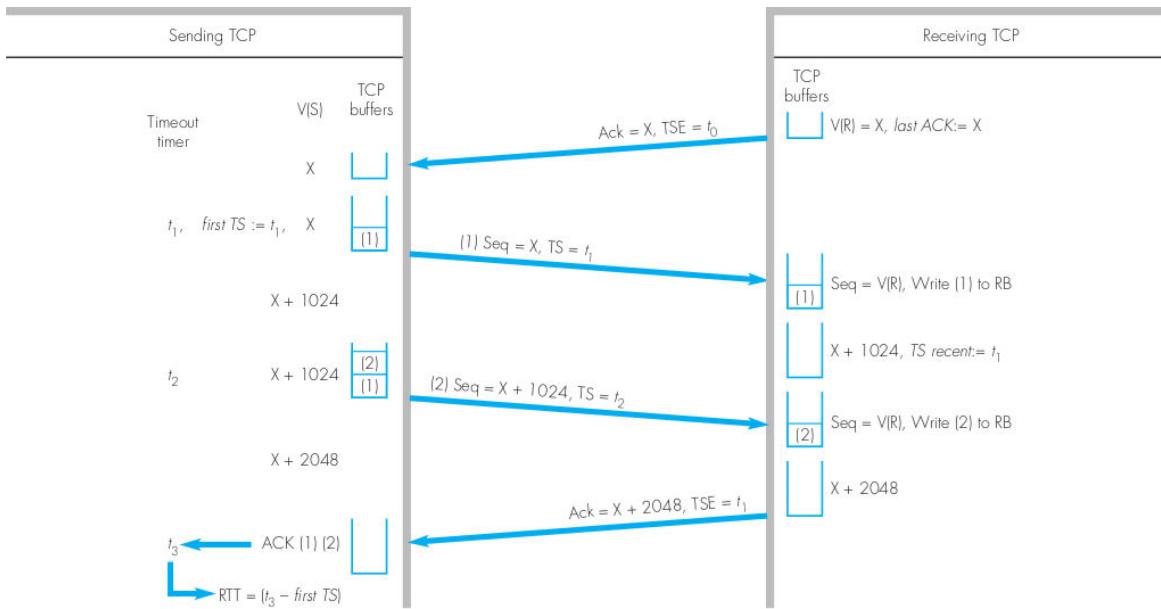
La misura del tempo M sarà dunque una semplice differenza tra il valore attuale del contatore e il valore presente nel campo *Time-stamp echo reply* dell'ack.

Un ottimizzazione prevede la possibilità di usare ack cumulativi, dove nel timestamp si trova il valore solo del primo segmento della catena.

Le informazioni relative ai tempi viaggiano in un header opzionale dalla seguente struttura:



E la figura seguente mostra il suo utilizzo.



Esempio

TCP Timestamps Option (TSopt):

Kind: 8

Length: 10 Byte

Kind=8	10	TS Value (TSval)	TS Echo Reply (TSecr)
1	1	4	4

Ogni host ha un timer di 32 bit che ogni 500 ms viene incrementato.

Il client all'istante t_1 invia un segmento al server settando nel timer stamp $\text{TSval} = t_1$

Il server, riceve il segmento al tempo t_2 , setta $\text{TSecr} = t_2$ e spedisce ACK al client

Il client riceve al tempo t_3 l'ACK e calcola t_3-t_2 e t_2-t_1 per stimare l'RTT.

TCP valida una coppia di tempi. Essendo cautelativo, per stimare RTT, usa la coppia di istanti più lontana.

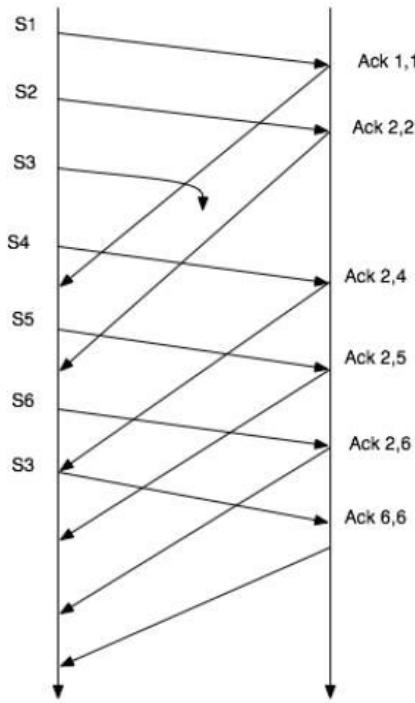
S-ACK

ACK selettivo invece che ACK cumulativo (RFC 2018).

Il selective ACK è un acknowledge contenente l'ultimo pacchetto ricevuto in sequenza e l'ultimo pacchetto che potrebbe essere fuori sequenza. Appena ricevuto un ACK fuori sequenza, viene spedito il pacchetto corretto. Funziona anche per *multiple loss*.

È molto simile alle politiche di recovery del livello DataLink. S-ACK viene settato come attivo nel primo segmento SYN che il mittente manda al destinatario, ovviamente come campo Opzione.

Vediamo un esempio di come funziona:



Controllo della congestione

Siccome TCP può essere operativo su più connessioni simultaneamente, è inclusa anche una procedura di controllo della congestione.

A differenza del livello 2, TCP non conosce la capacità del canale, pur conoscendo invece la dimensione del buffer lato ricezione (che sarà la dimensione massima della finestra disponibile per questa connessione) e MSS (stabilito in fase di connessione).

Per dimensionare la finestra in modo corretto, TCP adotta un approccio cauto.

Abbiamo visto che il protocollo TCP deve conoscere la misura del buffer del ricevente per configurare al meglio la grandezza della finestra. La dimensione della finestra però è condizionata anche dallo stato di congestione della rete sottostante: è stupido "stressare" una rete già congestionata (satura) scegliendo una dimensione della finestra elevata.

Per fare queste valutazioni devo conoscere lo stato della sottorete.

Si può avere una stima della congestione attraverso lo studio dei ritardi e degli errori di trasmissione avvenuti nella sottorete.

TCP, quando si avvia, parte con una finestra piccola e man mano l'aumenta fino al limite imposto dal ricevitore. Quando però avviene un errore si azzera tutto e si parte ancora con la più piccola dimensione possibile.

Nel caso in cui non si verifichino errori (o perdita ACK), TCP divide il trasferimento dati in 3 fasi:

1- **Slow Start** TCP trasmette segmenti piccoli partendo dalla dimensione minima dei segmenti stabilita all'inizio della connessione cioè MSS, maximum segment size, e imposta dimensione della finestra di congestione a 1. Ricevuto l'ack, raddoppia tale dimensione ($\#$ finestra= $2^{\#}$ trasmiss. X MSS). Questa fase viene iterata fino a quando non si raggiunge la soglia SST (pari alla metà della dimensione massima della finestra). Per ogni segmento inviato (in figura è di 1K), il sender si aspetta un ack.

2- **Congestion Avoidance** All'inizio di questa fase la finestra di congestione ha come dimensione la metà della dimensione massima della finestra decisa in fase di apertura della connessione (MSS). Da questo punto in poi (sempre se non si verificano errori) TCP trasmette finestre aumentandone la dimensione linearmente (ovvero aggiungendo 1 segmento alla finestra). Il sender si aspetta un ack per ogni segmento della finestra inviato (in modo da convalidare tutta la finestra). Raggiunta la soglia massima trasferibile, si entra nella fase Constant.

3- **Constant**, durante questa fase si spediscono finestre di congestione della stessa dimensione pari al massimo possibile, stabilito in fase di apertura della connessione.

NB: Le connessioni TCP con vita breve non sono molto efficienti, perché partono con lo slow start (vedi prime versioni di HTTP che era di tipo non persistente).

Durante lo slow start, TCP raddoppia la grandezza della finestra per ogni segmento convalidato.

Durante la congestion avoidance, TCP aumenta la grandezza della finestra di 1 segmento per ogni finestra convalidata.

Slow Start & Congestion Avoidance

L'uso elevato dei cavi in fibra ottica riduce notevolmente il numero di errori di trasmissione, per questo la causa della maggior parte della perdita dei pacchetti è la congestione della rete. Per questo TCP include una procedura di controllo della congestione che utilizza la velocità di ricezione degli ACK per regolare la velocità con cui vengono inviati i segmenti.

Essa funziona nel seguente modo:

- ogni entità TCP mantiene una variabile W_c detta **congestion window variable**
- all'apertura della connessione, il mittente non conosce la capacità e setta quindi W_c al valore di MSS concordato
- il mittente invierà quindi un singolo segmento e farà partire un timer di ritrasmissione. Se l'ACK non viene ricevuto prima che il timer scada il segmento verrà reinviato; altrimenti se l'ACK sarà ricevuto in tempo, allora W_c verrà incrementato a due volte MSS (come previsto dalla fase slow start). Il mittente può inviare due segmenti ed ogni volta che riceve in tempo l'ACK incrementa W_c

Come cresce? La variabile W_c viene incrementata esponenzialmente:

$$W_c = 2^n \cdot \text{MSS} \quad \text{Esempio: } (2^0 \cdot \text{MSS}) \rightarrow (2^1 \cdot \text{MSS}) \rightarrow (2^2 \cdot \text{MSS}) \rightarrow (2^3 \cdot \text{MSS}) \rightarrow (2^4 \cdot \text{MSS}) \rightarrow \dots$$

Ciò nonostante, questa fase viene detta **slow start** e continua finché:

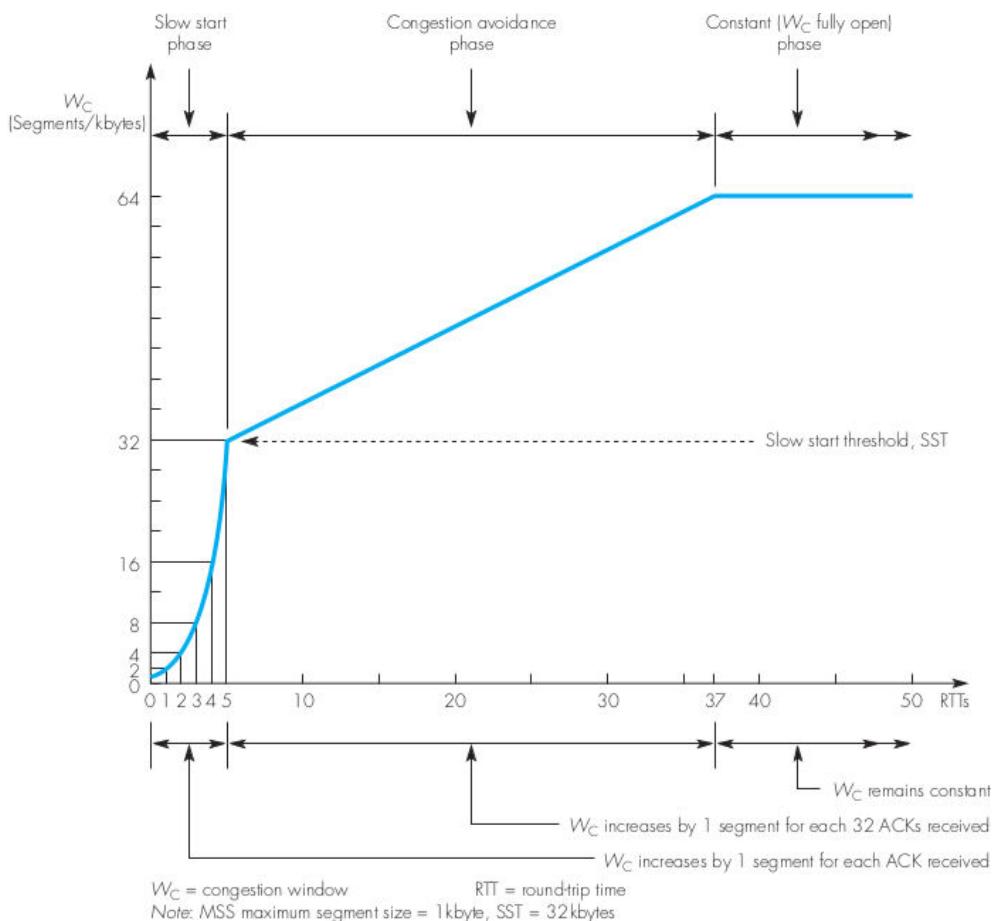
- non viene ricevuto un ACK duplicato
- non scade un timer di ritrasmissione
- non si raggiunge una certa soglia detta **SST (slow start threshold)**

se nessun errore si è verificato, una volta che la SST è stata raggiunta, si entra in una seconda fase detta **Congestion Avoidance**.

Durante questa fase la variabile W_c viene incrementata linearmente aggiungendo MSS per ogni ACK ricevuto.

$$W_c = W_c + \text{MSS}$$

Questa fase continua fino a che non viene raggiunta la dimensione max prevista in fase di setup della connessione (di 64 KB nell'esempio) oppure se raggiungo la dimensione del Buffer Receiver (cioè $W_c = W_r$, saturazione) e, da quel punto in poi W_c resta costante.



Esercizio

Si considerino gli effetti dell'uso di slow start su una linea con RTT di 10 msec e nessuna congestione. La finestra di ricezione è 24 KB e la massima dimensione di segmento è 2 KB. Quanto tempo è necessario prima che possa essere inviata una finestra intera, se la slow start threshold è 32 KB?

SOL:

$$RTT = 10 \text{ msec}$$

$$Wr = 24 \text{ KB}$$

$$MSS = 2 \text{ KB}$$

$$SST = 32 \text{ KB}$$

A t=0 invio 2KB. Poiché non c'è congestione, ricevo correttamente l'ACK. ($2^0 \times 2\text{KB}$)

A t=10 msec invio 4 KB; ($2^1 \times 2\text{KB}$)

A t=20 msec invio 8 KB; ($2^2 \times 2\text{KB}$)

A t=30 msec invio 16 KB. ($2^3 \times 2\text{KB}$) [Wc cresce in modo esponenziale = "partenza lenta", $Wc = 2^n \cdot MSS$]

A t=40 msec la congestion window diventa 32 KB ($2^4 \times 2\text{KB}$)

[Soglia di partenza lenta ("slow start threshold") viene raggiunta]

$\min(Wc, Wr) \rightarrow \min(32, 24) = 24$, perciò a t=40 msec invio l'intera finestra di ricezione.

L'esercizio non chiede quanto tempo ci impiega a raggiungere la slow start, ma chiede quanto tempo ci impiega a riempire la finestra Wc (cioè la dimensione concordata al setup della connessione in base alle capacità del buffer di ricezione Wr)

Fast Recovery / Fast Retransmit

Come abbiamo visto nel controllo del flusso di trasferimento dati di TCP, il destinatario manda 3 ACK duplicati (e nel caso peggiore scade il RTO).

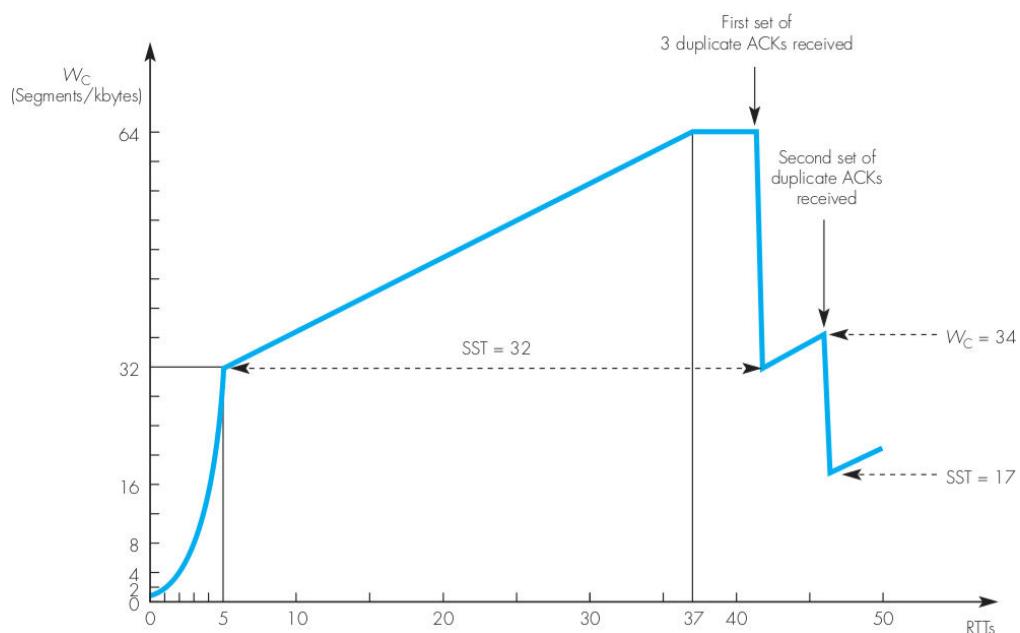
Per il controllo della congestione, questo non è considerato un errore grave in quanto il receiver ha ricevuto comunque quasi tutti i pacchetti, quindi posso escludere che sia un problema di congestione della rete.

Il livello di congestione viene considerato leggero e quindi, alla ricezione del terzo ACK duplicato:

- 1 - SST viene settata al valore di W_c dimezzato, spesso indicato come $FlightSize/2$ ($FlightSize$ = byte di *outstanding*, quelli già trasmessi)
- 2 - W_c è il valore minimo tra il nuovo valore di SST appena calcolato e il valore della soglia SST iniziale,

$$SST = \frac{(curr\ w_c)}{2} \quad e \quad w_c = new\ SST$$

Si procede quindi con la fase di congestion avoidance.



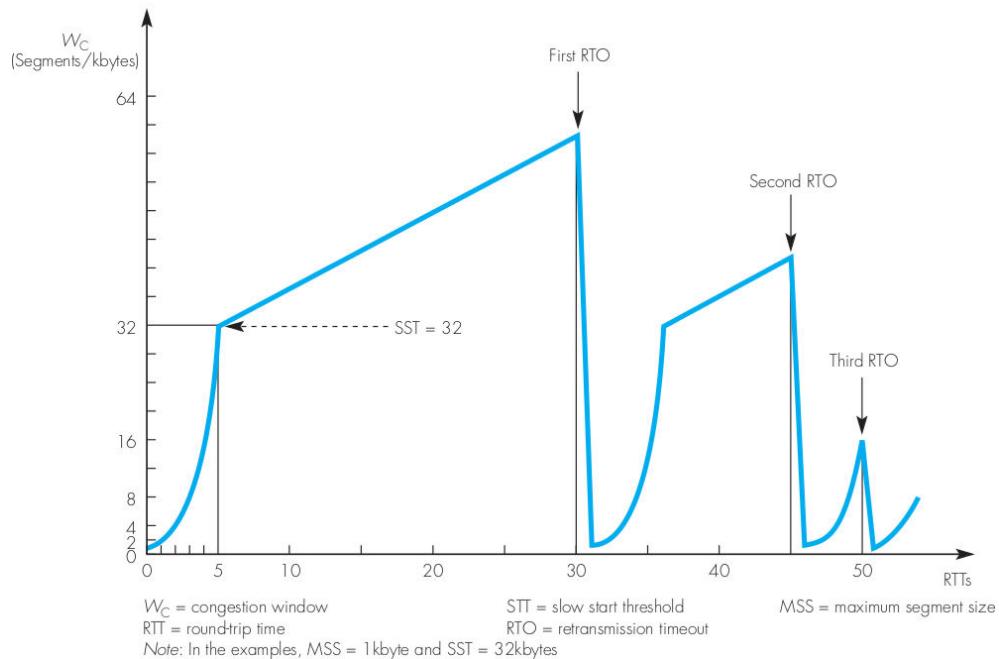
RTO recovery

Si verifica quando scade il timer della trasmissione.

È un errore considerato grave perché si ipotizza che la rete sia congestionata.

Il valore di W_c viene resettato a 1, si ricomincia con la procedura slow start e SST è data dalla metà della dimensione di W_c che ha causato l'errore (flight size/2).

$$SST = \frac{(curr\ w_c)}{2} \quad e \quad w_c = 1 \text{ MSS}$$



Esercizio

Si supponga che la finestra di congestione di TCP sia di 18 KB nel momento in cui si verifica un retransmission timeout. Quanto sarà grande la finestra se le successive 4 trasmissioni avvengono con successo? Si assuma che la max dimensione di segmento sia 1 KB e la slow start threshold sia di 16KB.

SOL:

$W_c = 18 \text{ KB}$
 $\text{MSS} = 1 \text{ KB}$
 $\text{SST} = 16 \text{ KB}$

La soluzione è determinata considerando quanto stabilito dallo RFC 5681, secondo il quale quando si verifica un retransmission timeout, la slow start threshold SST è portata al valore $SST = \text{FlightSize}/2$, dove FlightSize è il valore della dimensione della finestra W_c quando si è verificato RTO. La congestion window viene riportata a 1 MSS; la sua crescita è esponenziale fino al raggiungimento della SST e lineare successivamente.

Nell'esercizio dato, la nuova SST viene posta a $18/2 = 9 \text{ MSS/KB}$.

$$SST = \frac{\text{FlightSize}}{2} = \frac{18}{2} = 9 \text{ KB} (\approx 8, \text{ perché } 9 \text{ non è multiplo di } 2!)$$

La nuova fase di Slow Start viaggia ora da 0 a 9 KB

Riparto da Max TPDU=1 KB; alla seconda trasmissione invio 2 KB, alla terza 4 KB e infine alla quarta invio 8 KB.
($2^n \cdot \text{MSS} = 2^3 \cdot 1 \text{ KB} = 8 \text{ KB}$)

NB: alla prima ritrasmissione ho $n=0$, poi $n=1\dots$. Alla quarta ritrasmissione ho $n=3$.

Le successive trasmissioni corrette avranno quindi come valori di W_c 1, 2, 4, 8. Quindi dopo la quarta ritrasmissione $W_c = 8 \text{ KB}$

Se invece di RTO si fosse verificato un Fast Retransmit causato da 3 ACK?

$$SST = \frac{FlightSize(=Ex\ Wc)}{2}$$

SST_new = 9 KB

Poiché l'errore non è grave, si ricalcola SST come sopra e $Wc = SST$. Poi si procede con la fase di congestion avoidance, incrementando linearmente Wc per ogni ritrasmissione corretta:

$$Wc = 9, 10, 11, \underline{12} \quad [Wc = Wc + 1]$$

dopo 4 ritrasmissioni corrette ho :

$$\underline{Wc = 12 \text{ KB}}$$

Esercizio

Si supponga che la finestra di congestione di TCP sia di 18KB quando si verifica l'arrivo di 3 duplicate ack. Si assuma che la MSS sia 1 KB e la SST sia 16KB. Quali sono i due nuovi valori di SST e Congestion Window? Quanto sarà grande la finestra se alle successive 2 trasmissioni (oltre alla ritrasmissione) corrisponde l'arrivo di duplicate ack? Cosa succede se dopo la 3° trasmissione si riceve un altro duplicate ack?

SOL: La soluzione è determinata considerando quanto stabilito dallo RFC 5681, secondo il quale quando si verifica la ricezione di 3 ack duplicati viene eseguita la procedura di Fast Retransmit / Fast Recovery: la SST viene ridimensionata come nel caso di retransmission timeout mentre la congestion window viene posta uguale alla nuova SST. Da qui la crescita della finestra riprende in modo lineare. Se dovesse verificarsi la ricezione di altri 3 ack duplicati, la procedura viene ripetuta.

Dopo la prima terna di duplicate ack, $SST = 18/2 = 9 \text{ KB}$ e $Wc = 9 \text{ KB}$.

Dopo 2 trasmissioni, sarà $SST = 9$ e $CW = 11$.

Se dopo la 3° trasmissione si verifica nuovamente 3 ack duplicati, $SST = 11/2 = 5$ e $Wc = 5$.

Esercizio

Su una connessione TCP con segment size massima di 2 KB, congestion window=16 KB, soglia=32 KB, si verifica un fallimento trasferendo un segmento di 16 KB. In base all'algoritmo slow start, a quale valore è posta la nuova soglia e quale è la dimensione del prossimo segmento?

SOL

MSS = 2KB
 Cw = 16 KB
 SST = 32 KB

Fast Recovery: Il prossimo segmento avrà sempre dimensione 2 KB, $Wc = 8 \text{ KB}$ la soglia SST verrà fissata a 8 KB.

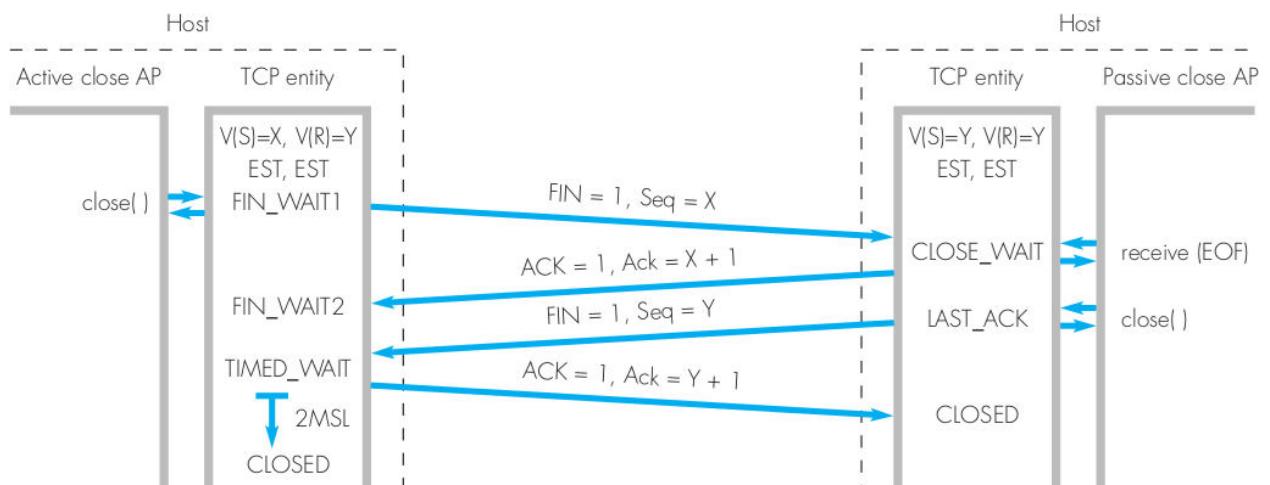
Chiusura normale

Il client effettua una chiusura attiva, il server una chiusura passiva. Essendo TCP una connessione bidirezionale, è possibile chiudere solo un lato della connessione, mentre l'altro rimane funzionante e deve essere chiuso anch'esso.

Quando il client esegue la `close()`:

- Manda un messaggio con bit FIN attivo al server
- Il processo server riceve il codice EOF ed effettua anch'esso la `close()` validando il client
- Anche il server chiude, quindi manda al client un segmento con FIN attivo
- il client valida con ACK il FIN del server

CLIENT	SERVER
Richiedo chiusura	Comunica al client la possibile chiusura
	Ack al client
Aspetta	Server chiude connessione
	Invia ack (FIN)
Ack di risposta alla chiusura server	
Aspetto per accettarmi che il server abbia ricevuto il mio ultimo ack	
Chiudo connessione	



Il client, dopo la FIN ricevuta dal server, aspetta un tempo regolato da un timer pari a 2MSL; entro questo tempo è ragionevole pensare che, se l'ACK mandato dal client che conferma la chiusura del server è andato perso, un altro FIN arrivi al client dal server.

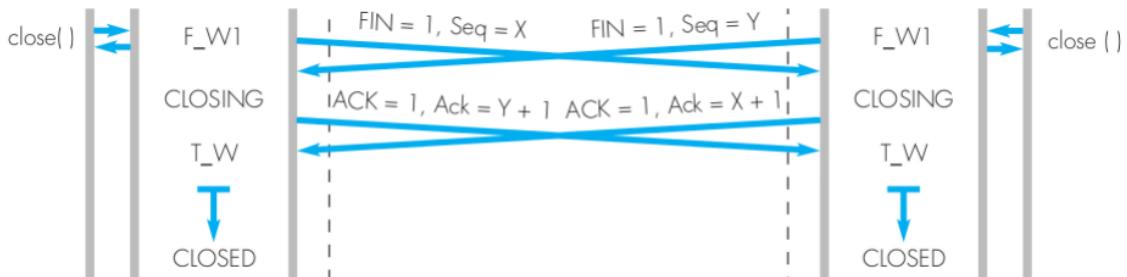
Il client allora potrà così rispondere nuovamente al server prima di chiudere.

MSL è il Maximum Segment Lifetime, cioè il massimo intervallo nel quale un segmento può viaggiare in Internet prima di essere droppeato, quindi relativo al campo TTL dell'header IP.

Simultaneous close

Se siamo in una situazione peer-to-peer (detta *simultaneous close*):

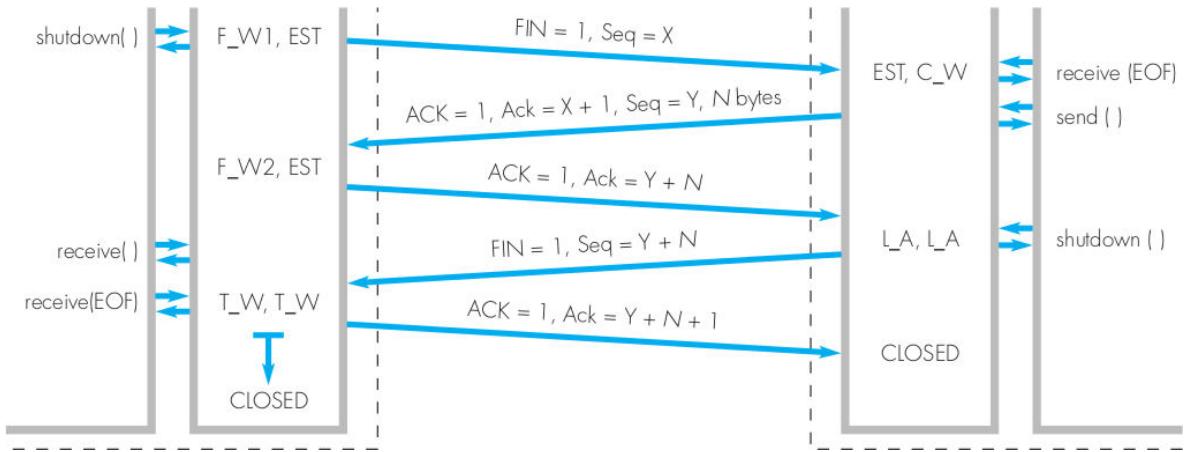
- Entrambi gli attori mandano la FIN
- Entrambi mandano l'ACK per il FIN ricevuto
- Vanno in "timed_wait" (T_W) per essere sicuri che l'ACK arrivi



Half-close

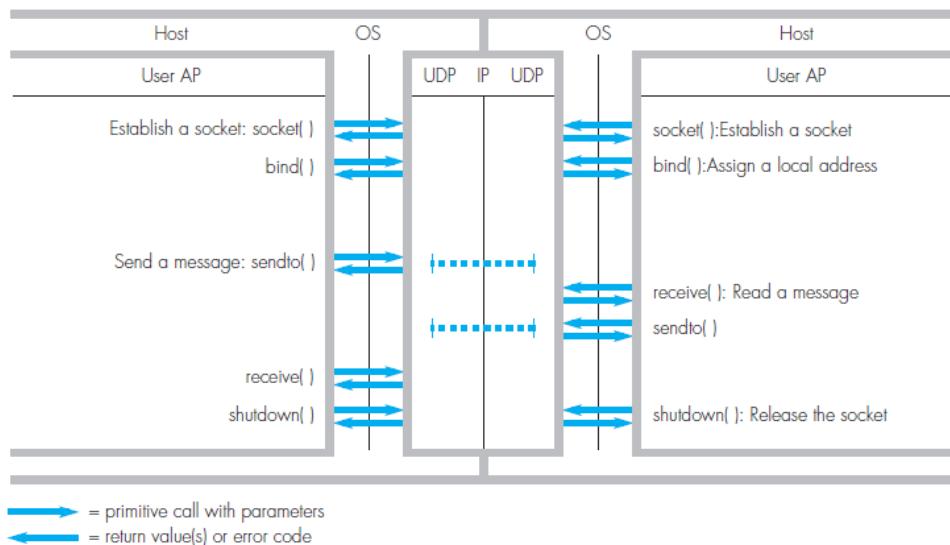
È anche possibile una situazione nella quale solo una parte abbia finito l'invio dei dati, mentre l'altra ha ancora segmenti pendenti (detta *half-close*):

- Dopo l'invio della FIN (chiusura attiva dell'attore a sinistra), essa viene validata dalla parte destra;
- Prima di chiudere però, la parte destra deve aspettare che tutti i pacchetti pendenti siano inviati
- Al termine, viene effettuata la chiusura attiva della parte destra (FIN) e ci si comporta come client-server



UDP – User Datagram Protocol (p.445)

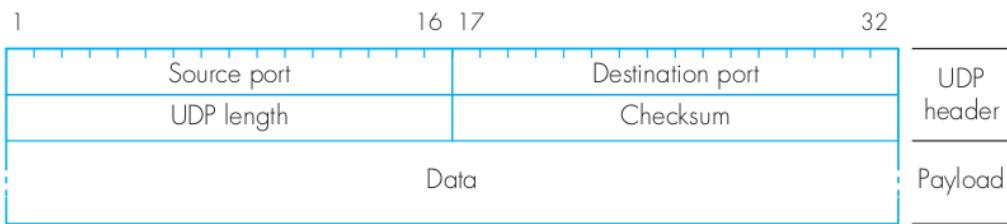
E' un servizio puramente datagram, non aggiunge nulla ad un servizio best effort del livello IP. È abbastanza usato in tutte le applicazioni dove vengono scambiate delle singole unità dati o pacchetti. Con un canale datagram ho dei vantaggi da un punto di vista prestazionale.



Quella sopra è l'interfaccia socket con le primitive per un servizio non affidabile. Socket e bind sono analoghe all'interfaccia TCP, shutdown serve per chiudere la socket, però chiude un solo lato perché è locale. "send-to" e "receive" sono primitive datagram su una rete socket.

UDP non fornisce affidabilità, ma è più veloce di TCP. Inoltre non esiste il concetto di finestra: appena un segmento è pronto viene spedito (non vengono aperte e chiuse connessioni). Per la sua natura, si tende ad usare UDP per informazioni di piccole dimensioni, in modo da evitare frammentazione. In UDP quindi se i segmenti vengono persi o risultano corrotti, sarà il protocollo a livello applicazione che si occuperà di ritrasmetterli. Un esempio di utilizzo di UDP è il protocollo BGP il quale scambia i suoi path vector utilizzando UDP appunto.

Header UDP



Dimensione 8 B, dove:

- *UDP Length*, in teoria la dimensione massima è di 65535 byte, ma poiché non affidabile si spediscono sempre segmenti con lunghezza massima 8192 byte

$$2^{16} - 20 - 8 \\ \text{dove } 20 \text{ header IP e } 8 \text{ header UDP}$$

- *Cheksum*, è opzionale e formato come quello di TCP, quindi basato sull'intero segmento più uno pseudo-header ricavato da alcuni campi dell'header IP. Se non viene utilizzato il cheksum è di tutti zeri.

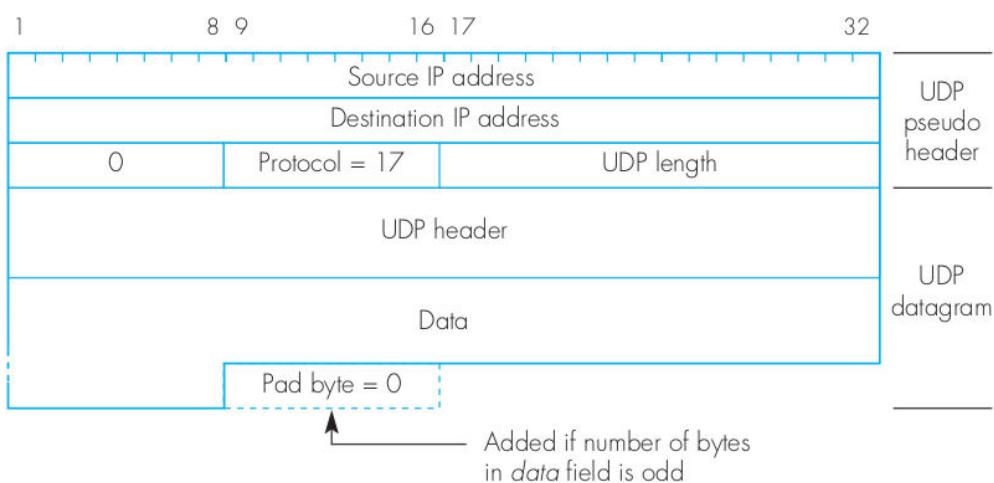
Non c'è Acknowledgement

UDP non viaggia frammentato su IP, viaggia su un'unica entità

Se è necessaria la frammentazione, questa viene svolta a livello applicazione

Per convenzione Internet è obbligata a mandare frame di 512 B.

Ricordiamo che solo creando una connessione esplicita tra sorgente e destinazione abbiamo il controllo sul flusso.



Performance e ottimizzazioni (vedi anche p.61)

I vari livelli della scala ISO-OSI, visti fino ad ora, comunicano tramite interfacce: ogni livello prepara il suo pacchetto e lo passa al livello inferiore. A livello architettonico e ingegneristico la struttura è solida ma computazionalmente implica molti *context switch* (tra livello 3 e 4). La vera implementazione infatti è diversa dal disegno architettonico.

1) I livelli 3,4,7 si scambiano solo puntatori alla struttura dati, non l'intero dato.

Una prima ottimizzazione riguarda gli header di ciascun livello. Nel passare dal livello n al livello n+1 (e viceversa) si utilizzano dei puntatori (non viene fatta nessuna copia degli header). Ogni livello prepara solo il proprio header e passa al livello inferiore un puntatore alla locazione di memoria dei dati. Per esempio: il livello datalink, nella preparazione del frame, scorrerà tali puntatori per comporre il frame finale da mandare al livello fisico.

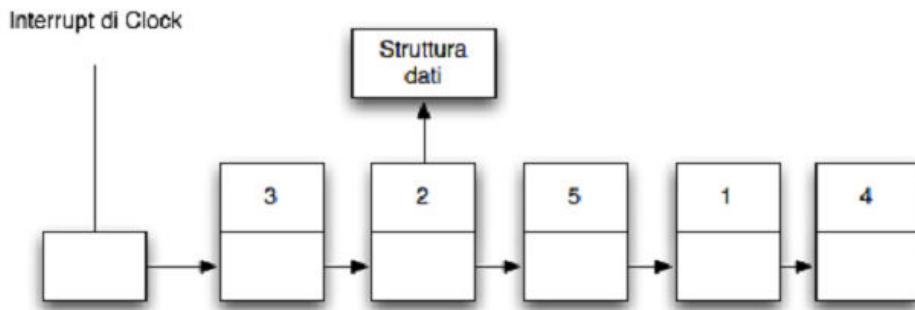
2) Gli header sono formati attraverso la modifica di header "preconfezionati" (Template Header)

La preparazione degli header in ogni livello, viene effettuata di fatto su campi sempre uguali (stessa dimensione e spesso stessi valori) per cui solitamente si usano i "prototype" header. Quindi, un ulteriore ottimizzazione consiste nell'uso di **template**: strutture statiche in cui ogni livello compila solo i suoi campi. Tale struttura è mantenuta fino alla fine della connessione.

3) Timer virtuale (o logico) - vedi pag. 63

Tecnica per gestire la coda dei timer, ogni segmento ha un timer associato e per ciascuno di essi si ha un RTO.

Poiché ogni finestra dovrebbe possedere un proprio timer, nel caso la dimensione fosse di 32 sarebbe difficile tenere 32 timer contemporaneamente. Si ricorre dunque a liste puntate contenenti il valore di tempo da aggiungere al valore del timer precedente. Si ricorre a un descrittore il quale indica per differenza l'elapsed time (timer di scadenza).



Ogni descrittore di segmento TCP ha associato un numero che indica dopo quanti tick di clock scadrà il timer associato. È incrementale, ovvero nell'esempio, il primo scadrà tra 3 tick, il secondo tra 5, il terzo tra 10..... a ogni tick di clock viene decrementato solo il primo numero del primo descrittore. Quando arriva a 0, viene tolto.

Un descrittore viene tolto anche quando arriva l'ack corrispondente. Ogni elemento nuovo viene aggiunto in coda, quindi ho 2 puntatori, uno in testa ed uno in coda.

Un altro metodo utilizzato è la **Ruota del tempo** (utilizzato non tanto nella realtà quanto per simulatori di protocolli di rete):

Ho un array circolare di N posizioni (dove N è la dimensione massima del timer), inizializzato a [0,1,2,3,..., X, ..., N] e un puntatore che punta alla prima posizione. A ogni tick di clock il puntatore si sposta verso destra di 1 mod N. Ogni posizione punta ad una lista di segmenti che scadono dopo X tick. Quando il puntatore si sposta, vengono gestiti tutti i segmenti in attesa in quella posizione. Se arriva un ack di un segmento qualsiasi, viene tolto dalla lista solo quell'elemento.

Livello Application

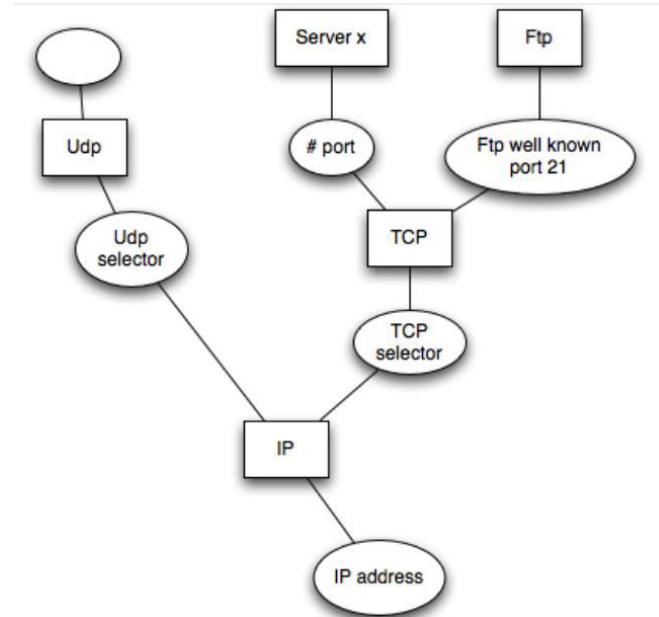
Tutti i livelli dall'uno al quattro, hanno una funzionalità ben precisa. Il livello 7 è anomalo da questo punto di vista, in quanto fonde in sé funzionalità diverse, tante quante sono i servizi offerti all'utente finale.

I servizi di livello 7 che vedremo sono:

- SMTP: posta elettronica
- FTP : file transfer

- DNS: risolutore di nomi logici in indirizzi IP
- HTTP: web protocol
- DHCP: è implementato a livello 7 ma offre funzionalità di livello 3.

Nella suite di protocolli TCP/IP, dato un IP address e una porta, i servizi forniti da TCP e UDP permettono a due AP di comunicare in modo trasparente, e infatti non importa che i due AP si trovino sulla stessa macchina, nella stessa sottorete o in due punti diversi del globo. TCP/UDP non si interessa del proprio payload, ma si limita solo a trasferirlo da un interlocutore all'altro: potrebbe quindi essere necessario un metodo per verificare che la sintassi e la semantica dei messaggi tra le due AP sia la stessa.



Naming & Addressing in rete

Abbiamo visto tecniche e algoritmi per identificare un partner in una comunicazione di rete:

- MAC – Ethernet (livello 2)
- IP-Address (livello 3)
- Socket (livello 4)
- DNS (livello 7)

Le socket servono per legare i servizi di livello 7 a servizi di livello 4

DNS trasferisce da IDN (Internet Domani Name) a NDN (Network Domain Name).

Internet domain name → DNS → IP Address → ARP → Local network address

DNS – Domain Name System

Un AP che vuole comunicare con un corrispondente AP deve conoscere l'indirizzo IP e la porta di quest'ultima. Come sappiamo, la porta sarà ricavabile facilmente, essendo una porta nota associata a quel particolare AP; ma l'indirizzo IP?

Per facilitare la ricerca di un indirizzo IP, il DNS permette ad ogni entità presente sulla Rete di possedere, oltre all'indirizzo IP, un **nome simbolico**. DNS (RFC 1034, 1035) si occupa quindi dell'associazione nome-indirizzo IP, e alla mappatura da uno all'altro prima dell'apertura della connessione.

Esempio

rossi@dico.unimi.it

rossi = nome dell'utente

@ = identificatore del servizio (mail)

dico.unimi.it = nome del dominio

Esempio

www.dico.unimi.it

www = identificatore del servizio

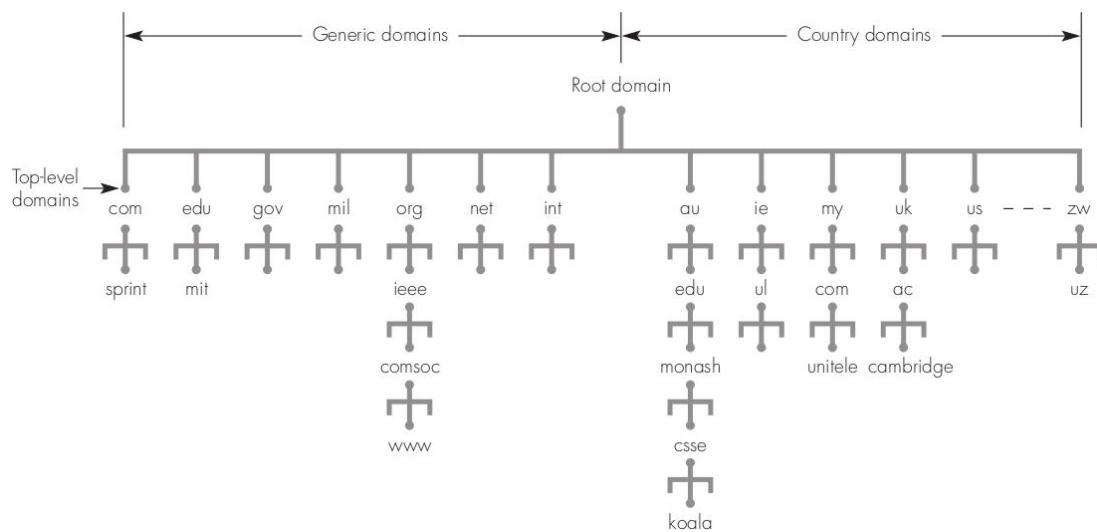
dico.unimi.it = nome del dominio

Struttura di DNS

Tutti i dati che vengono usati dal DNS sono chiamati *domain space name* (spazio dei nomi di dominio) e sono indicizzati per nome. La struttura è gerarchica, in quanto:

- È amministrabile in modo distribuito
- Si sceglie di assegnare i nomi in relazione alla locazione geografica, in modo che molte delle richieste di risoluzione di un nome possano essere risolte più velocemente e senza introdurre traffico in zone non interessate (vale il principio di località spaziale).

Abbiamo quindi una situazione di questo tipo:

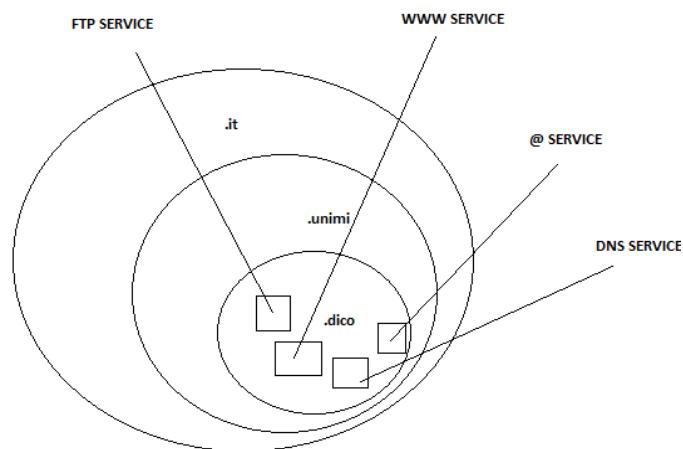


Dove:

- **Generic Domains**, sono quei domini che esistevano quando Internet era ancora relativamente piccola e quindi essi bastavano per identificare una particolare organizzazione (nati in america)
- **Country Domains**, nati quando Internet è cresciuta, introducono domini specifici per ogni nazione (ISO 3166)

Per ogni dominio esiste un DNS che è un server che gestisce lo spazio dei nomi.

L'organizzazione è autoritativa ed è l'unica responsabile dell'associazione nome-IPaddress per ogni dominio.



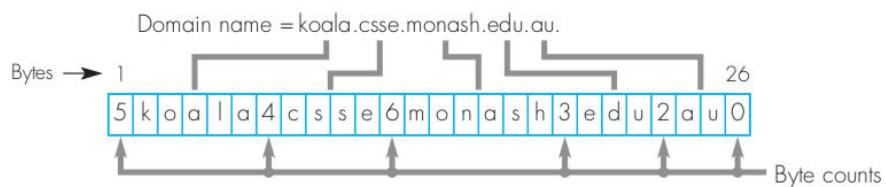
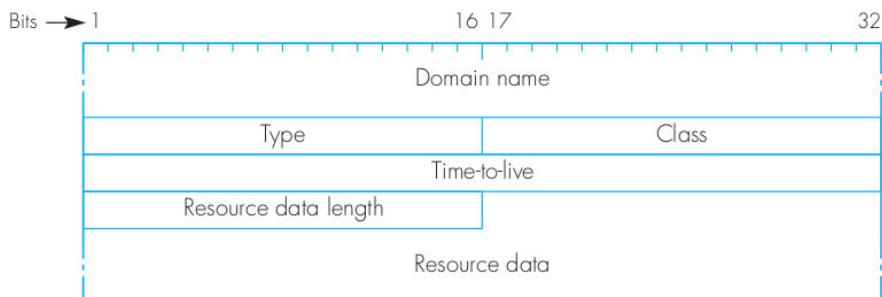
Esempio di Rappresentazione topologica

Dominio = è uno spazio di nomi organizzato in modo gerarchico

Servizio = può essere di diversi tipi (WEB, FTP, MAIL...)

Resource Records

Ogni dominio può avere informazioni aggiuntive al solo indirizzo IP; esse sono memorizzate in uno o più resource records, tutti indicizzati dal nome di dominio relativo. Un resource record è così formato:



Type	Value	Meaning
A	1	IP address
NS	2	Name server
PTR	12	Pointer record
HINFO	13	Host information
MX	15	Mail exchange

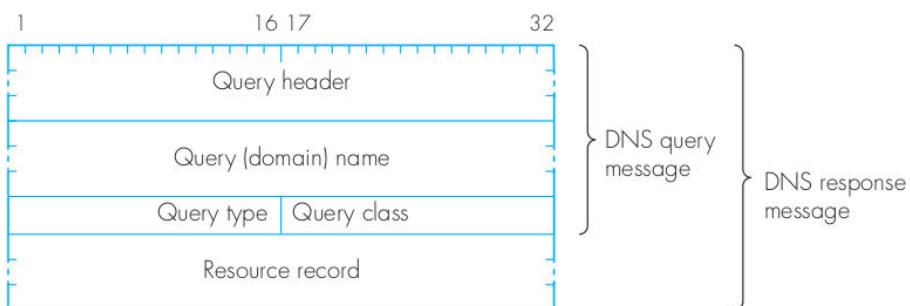
Dove:

- **Domain name**, è il nome di dominio a cui il record è correlato. Come specificato in figura, il formato prevede l'utilizzo di un counter di un byte prima di ogni etichetta, che ne determina la lunghezza. L'ultimo carattere è sempre lo 0, che rappresenta la root.
- **Type**, indica il tipo di record, che può essere uno di quelli specificati in tabella. Ad esempio A indica che il record contiene un indirizzo IP in forma binaria, mentre PTR un indirizzo IP in notazione decimale puntata; HINFO indica la presenza di informazioni sul tipo dell'host e sul sistema operativo, mentre MX contiene un'e-mail gateway che si può utilizzare per instradare messaggi di posta.
- **Time To Live**, indica il tempo in secondi entro i quali l'informazione contenuta nel record è valida (necessaria per motivi di caching, il valore tipico è 2 giorni).

DNS query messages

Il DNS opera su database di indirizzi, nomi logici che stanno nei vari server, di conseguenza i messaggi DNS sono delle query. Sono di piccola dimensione e quindi viaggiano attraverso UDP (se il messaggio si perde lo rimando). Se viaggiassero su TCP il tempo effettivo di trasmissione potrebbe essere inferiore al tempo necessario per aprire/chiudere la connessione TCP.

Il formato è il seguente:



Dove:

- **Query header**, è una stringa standard da 12 byte e contiene 16 bit di identificazione e 16 bit flags. Il primo è assegnato dal client che invia la richiesta, e viene lasciato inalterato dal server nella risposta, perché usato dal client per accoppiare la risposta alla relativa richiesta. Il secondo, invece, consiste in un numero di sottocampi; per esempio, un campo da 1 bit è usato per determinare se si tratta di una richiesta o una risposta, mentre un campo da 4 bit per identificare il tipo della ricerca
- **Query domain name**, contiene il nome di dominio a cui la richiesta è associata, nello stesso formato visto all'interno dei resource records

Per tutti i servizi di livello 7 e in tutti i comandi di controllo si usano caratteri ASCII (anche come valori dei campi)

Servers dei nomi

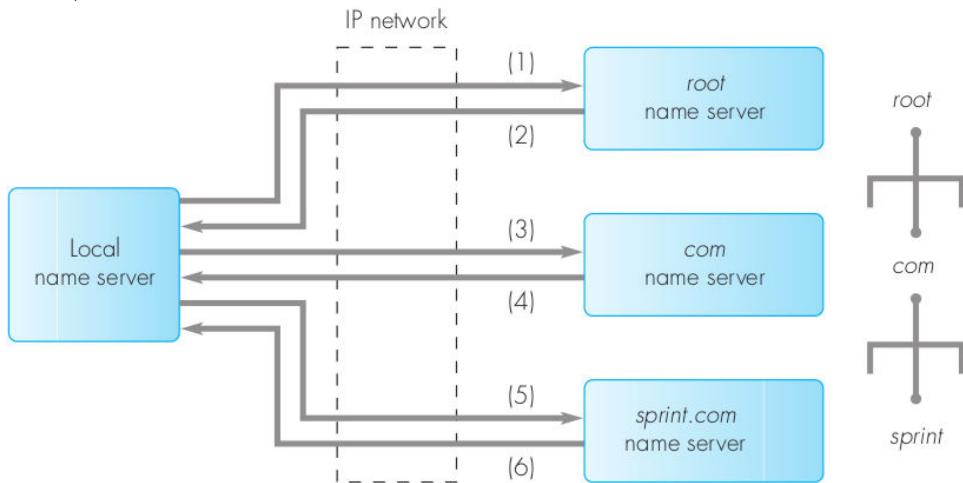
Come abbiamo già precisato, l'adozione di una struttura gerarchica facilita la risoluzione dei nomi, che nella maggior parte dei casi vengono risolti localmente. Per ottenere questa efficacia, lo spazio del dominio dei nomi è stato diviso in **zone**, ognuna delle quali mantiene e gestisce una parte dei domini. Ogni zona è poi amministrata da un'autorità separata, che è anche responsabile di fornire uno o più server dei nomi (**name server**) per la zona. In base al livello di gerarchia in cui ci si trova, un name server può avere autorità su una singola zona o su zone multiple.

Risoluzione dei nomi

Associati a ogni zona troviamo quindi un primary (name) server, ma probabilmente anche uno o più secondary (name) server: il primo contiene effettivamente i records relativi alla sua porzione del domain name space, mentre i secondi mantengono una cache dei dati contenuti nel primo. Il **caching** avviene quando, verificando che il record richiesto non è presente nel proprio database, i secondary o primary server indirizzano la richiesta al server del livello superiore e, alla ricezione dell'indirizzo IP richiesto, il server che ha iniziato la richiesta mantiene una copia volatile della risposta per il TTL previsto (come già detto, di default 2 giorni).

Fin'ora abbiamo visto la struttura del DNS, ma come funziona il servizio di naming?

Se ogni primary server potesse parlare con tutti gli altri primary server, l'overhead sarebbe altissimo; si è quindi deciso di non far conoscere ai primary server l'indirizzo IP dei suoi omonimi, ma invece di permettere il contatto solo con un insieme non esteso di top-level root name servers, che mantengono le coppie nome-indirizzoIP di tutti i primary server e, a ogni richiesta, comunicano quale primary server interpellare. Questo metodo di risoluzione del nome prende il nome di **recursive name resolution**, e quello che segue è un esempio del suo uso, nel quale si richiede la risoluzione del nome *sprint.com*. (questo nome simbolico, che termina con un punto per indicare la root, è un esempio di nome di dominio assoluto, anche detto **FQDN – Fully Qualified Domain Names**, ogni nome che non termina con un punto è quindi un nome incompleto o relativo):



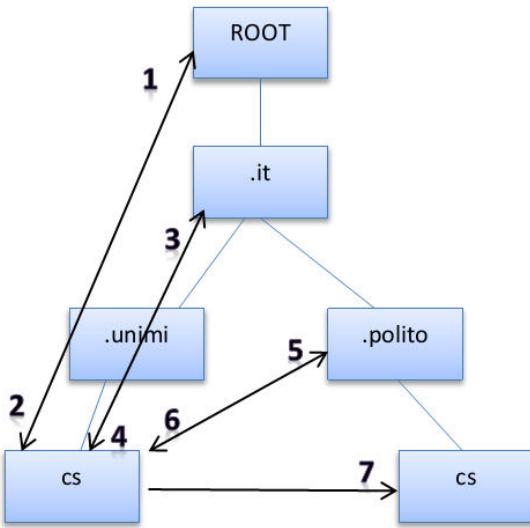
- (1) = local name server sends a recursive query message containing name of the destination host – for example, the *sprint.com*. gateway – to the *root* name server
- (2) = the *root* server returns the IP address of the *com* server
- (3) = local server sends a recursive query to *com* name server
- (4) = the *com* server returns the IP address of the *sprint.com* server
- (5) = local server sends a recursive query to *sprint.com* server
- (6) = the *sprint.com* server sends IP address of *sprint.com*. gateway (host)

Ancora un esempio di recursive name resolution

`cs.unimi.it` vuole parlare con `cs.polito.it`

- 1- `.cs` chiede di risolvere il nome `cs.polito.it` a `root`
- 2- `root` risponde con l'indirizzo del dominio `.it` perché non ha tutti i sottodomini
- 3- `.cs` chiede a `.it` di risolvere il nome

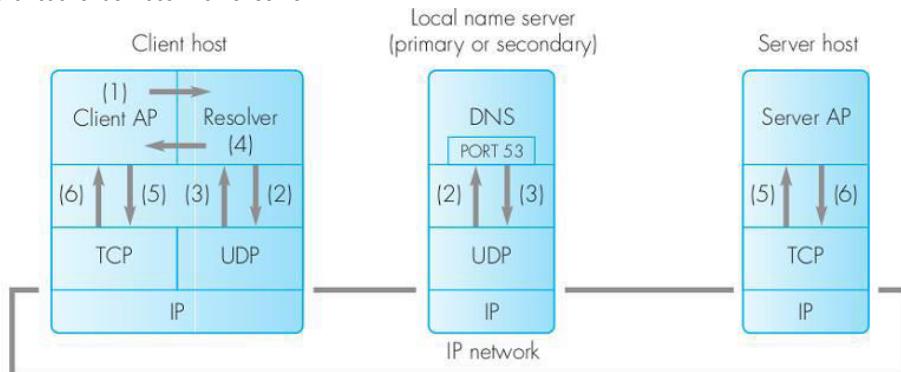
- 4- .it risponde con il dominio sottostante .polito
- 5- .cs chiede a .polito
- 6- .polito risponde con l'indirizzo di .cs
- 7- .cs.unimi.it comunica con cs.polito.it
- 8- .cs.unimi.it riceve la risuzione completa di cs.polito.it



Tutti gli host si riferiscono al DNS locale per risolvere un indirizzo

Il resolver è installato fisicamente nell'host che vuole effettuare una richiesta e usa UDP

Tutto inizia da una richiesta da parte di un host. In quest'altro esempio, la richiesta può essere subito soddisfatta perché il record è presente nella cache del local name server:

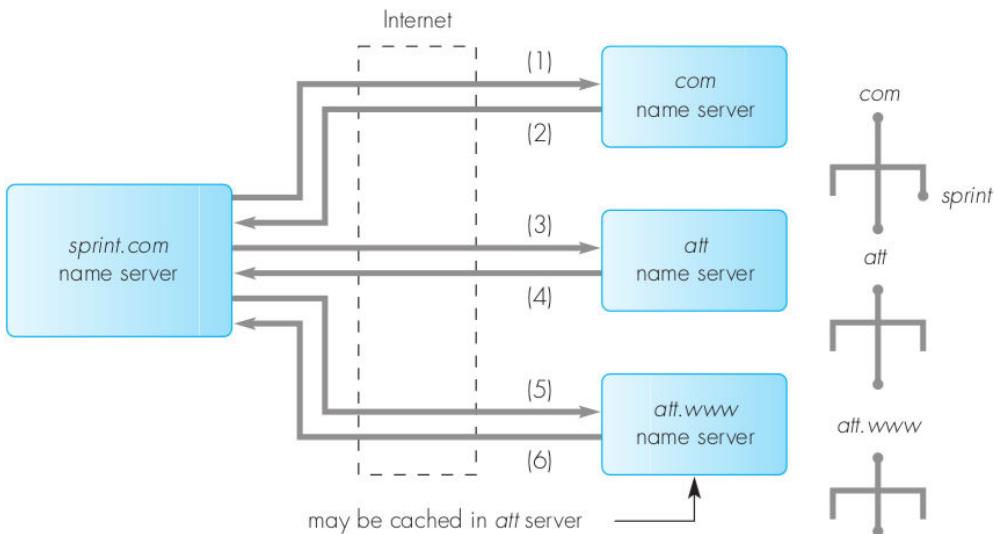


- (1) = resolver invoked by client AP with the name of the server host
(2) = resolver sends a type-A query containing the name of the server host to its local DNS
(3) = local DNS returns a type-A resource record containing the IP addresss of the server
(4) = resolver returns IP address of the server to the client AP
(5)/(6) = client and server APs carry out networked application/transaction

Il local DNS guarda nella cache se ha già una copia di quella risoluzione del nome.

Per evitare che venga sempre fatta una richiesta alla root, **iterative name resolution** propone di iterpellare sempre il server di livello superiore più vicino. Questo implica che i primary server debbano conoscere non gli indirizzi IP dei top-level root servers, ma del server del livello superiore.

Esempio di risoluzione iterativa

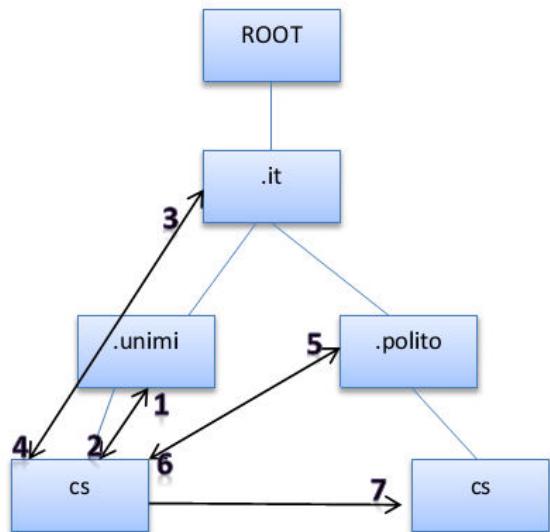


- (1) = local name server sends an iterative query message containing the name of the destination host – att.www – to the next higher-level server – com
- (2) = the com server replies with the IP address of the att server
- (3) = local server sends an iterative query to the att server
- (4) = the att server returns the IP address of the att.www server
- (5) = local server sends an iterative query to the att.www server
- (6) = the att.www server returns the IP address of the requested Web host

Ancora un esempio di iterative name resolution

cs.unimi.it vuole parlare con cs.polito.it

- 1- .cs chiede di risolvere il nome cs.polito.it al suo dominio soprastante (.unimi)
- 2- .unimi risponde con l'indirizzo del dominio .it
- 3- .cs chiede a .it di risolvere il nome
- 4- .it risponde con il dominio sottostante .polito
- 5- .cs chiede a .polito
- 6- .polito risponde con l'indirizzo di .cs
- 7- .cs comunica con .polito.it
- 8- cs.unimi.it riceve la risoluzione completa di cs.polito.it



Abbiamo visto due tecniche: ricorsiva e iterativa

Nella ricorsiva, la prima query va alla root. Ogni host è configurato per conoscere l'IP address di almeno una root. Abbiamo in totale 4 richieste e 4 risposte (migliore se troviamo qualcosa in cache)

Nella procedura iterativa non risalgo sino alla root ma risalgo l'alberatura dal basso. Il primo che incontro è il mio DNS padre.

Nel caso peggiore, cioè quello in cui non riesco a trovare nessuna copia di risoluzione nella cache dei vari DNS, ho impiegato sempre 8 step. Non molto differente dalla ricorsiva.

È preferibile l'approccio iterativo semplicemente perché è più facile che ci siano valori in cache nel dominio superiore più vicino alla macchina che sta richiedendo la risoluzione.

Esercizio

Un resolver di nome sulla macchina *mercurio.dsi.unimi.it* deve ottenere l'indirizzo IP della macchina *mars.cs.ucla.edu*. Quanti passi occorrono per risolvere il nome se l'informazione non è contenuta in cache al name server locale?

SQL 1-**mercurio** interroga il DNS locale, unimi.dsi

- 2- DNS locale, **unimi.dsi** interroga il name server per il dominio .edu
- 3- il name server .edu inoltra la richiesta al name server di ucla.edu il quale conosce l'indirizzo di cs.ucla.edu
- 4- il name server .edu chiede al name server di ucla.edu l'IP address di mars.cs.ucla.edu

In altrettanti passi l'informazione viene propagata all'indietro fino alla macchina richiedente. Nel caso considerato servono in totale 8 passi.

Esercizio

Il server www del dipartimento di informatica si trova sulla macchina *ghost.dsi.unimi.it*. Come è possibile indirizzare *ghost* usando l'URL www.dsi.unimi.it?

SQL Nel DNS per dsi.unimi.it è sufficiente inserire una entry per il record CNAME che specifichi l'alias:

www.dsi.unimi.it 86400 IN CNAME ghost.dsi.unimi.it

Esercizio

L'indirizzo www.dsi.unimi.it denota, all'interno del DNS del dipartimento di informatica, il server di WWW. Come è possibile rendere questo indirizzo indipendente da una macchina particolare?

SQL Inserendo la entry nel DNS in cui viene specificato il tipo CNAME; ad esempio:

www.dsi.unimi.it 86400 IN CNAME mercurio.dsi.unimi.it

Esercizio

In base a quale informazione nel DNS una mail indirizzata a rossi@dsi.unimi.it viene spedita al mail server del Dipartimento *mercurio.dsi.unimi.it* ?

SQL Il DNS mappa un domain name nel corrispondente resource record, una 5-tupla il cui campo type = MX (Mail Exchange) consente di specificare quale dominio è incaricato di ricevere email per un dato dominio. Una possibile entry del DNS per dsi.unimi.it sarà quindi:

dsi.unimi.it 86400 IN MX mercurio.dsi.unimi.it

Esercizio

Si consideri la rete del DICO, in cui il mail server del dominio *dico.unimi.it* è sulla macchina *hostsrv1.usr.dico.unimi.it*, che ha come alias *tic.usr.dico.unimi.it*. Il web server DICO, www.dico.unimi.it, è sulla macchina *hostsrv2.usr.dico.unimi.it*, che è un server Dell con sistema operativo Linux Debian. Gli indirizzi IP dei 2 server sono rispettivamente 173.149.83.24 e 173.149.83.25. Quali DNS si possono ricavare da questa descrizione?

<i>dico.unimi.it</i>	MX	IN	172800	<i>hostsrv1.usr.dico.unimi.it</i>
<i>hostsrv1.usr.dico.unimi.it</i>	CNAME	IN	172800	<i>tic.usr.dico.unimi.it</i>
www.dico.unimi.it	CNAME	IN	172800	<i>hostsrv2.usr.dico.unimi.it</i>
<i>hostsrv2.usr.dico.unimi.it</i>	HINFO	IN	172800	Dell server, Linux Debian OS
<i>tic.usr.dico.unimi.it</i>	A	IN	172800	173.149.83.24
<i>hostsrv2.usr.dico.unimi.it</i>	A	IN	172800	173.149.83.25

Esercizio

Si supponga che l'utente *pagani@dico.unimi.it* debba mandare una mail al destinatario *halsall@cs.wales.ac.uk*. Dire quanti accessi a DNS sono

necessari nel caso il DNS della University of Wales contenga i seguenti record:

cs.wales.ac.uk	TXT	IN	172800	"Comp. Science Dept., Wales Univ."
cs.wales.ac.uk	MX	IN	172800	mail.cs.wales.ac.uk
mail.cs.wales.ac.uk	CNAME	IN	172800	srv.usr.cs.wales.ac.uk
www.cs.wales.ac.uk	CNAME	IN	172800	srv.usr.cs.wales.ac.uk
srv.usr.cs.wales.ac.uk	HINFO	IN	172800	HP server, Linux OS
mail.cs.wales.ac.uk	HINFO	IN	172800	HP server, Linux OS
srv.usr.cs.wales.ac.uk	A	IN	172800	139.118.27.142

SOL: Servono 3 accessi:

il primo permette di scoprire che il mail server per il dominio cs.wales.ac.uk è l' host mail.cs.wales.ac.uk (riga 2);
il secondo accesso permette di scoprire che "mail.cs.wales.ac.uk" è un alias per l' host srv.usr.cs.wales.ac.uk (riga 3);
l'ultimo accesso consente di trovare l' IP address di srv.usr.cs.wales.ac.uk (riga 7)

Posta elettronica

Nel servizio di posta elettronica la comunicazione avviene tra due apparati o macchine client che si scambiano messaggi.

Entrambe le macchine client si appoggiano al loro server di posta elettronica per trasferire il messaggio.

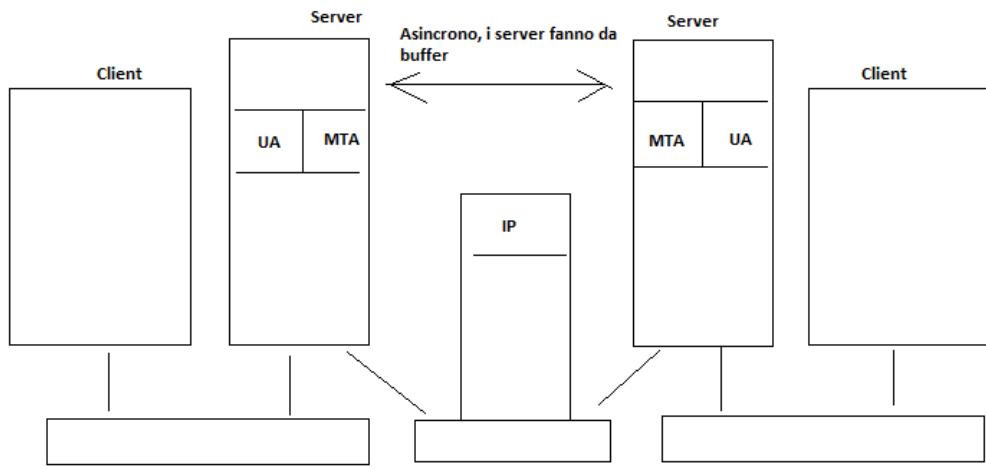
Il client è l'utente che vuole inviare la mail e il server di posta del client è la macchina che offre il servizio vero e proprio di invio del messaggio. Il server di posta è inoltre incaricato di ricevere i messaggi e inoltrarli alla macchina client appropriata.

Consideriamo ora quali sono le componenti funzionali per un mittente di posta elettronica e come avviene un invio di un messaggio.

Abbiamo:

- 2 **User agent**, risiedono a livello 7 rispettivamente una nella macchina client e l'altra nel server di posta.
 - UA client gestisce la scrittura/lettura per la gestione delle mail, consentendone la trasmissione in rete attraverso TCP (software di posta come Thunderbird hanno il loro UA)
 - UA server, risiede fisicamente nel server di posta e permette di interagire con gli UA dei client
- 2 **Message Transfer Agent**, risiedono entrambi nel livello 7 del server di posta.
 - MTA client si occupa di sincronizzare i dati con i client, mantenendo per ognuno di essi una IN mailbox nota come *message store* (usa port x e protocollo SMTP di livello 7)
 - MTA server si occupa di cercare il server di posta elettronica del destinatario, interrogando il DNS locale (usa well-known port 25 – usa TCP)

Analogamente per il destinatario avremo altri 2 MTA e 2 UA.



Esempio schematico per l'invio di una mail dall'utente A all'utente B

1-Il mittente prepara una mail compilando i campi necessari e poi invia il messaggio.

Mail To: gerla@cs.ucla.edu

gerla = utente
@ = tipo di servizio
cs.ucla.edu = dominio

Per l'invio si utilizza lo User Agent del client (che risiede sulla mia macchina locale).

L'UA è parte del client, è composta dall'interfaccia per scrivere la mail e dalla parte di comunicazione POP3 o IMAP (POP3 in disuso, IMAP nel caso deve essere fatta copia del messaggio inviato sia su server che su client).

2-Quando invio, riceve prima il mio server di posta elettronica

UA passa il messaggio all'UA del server locale (quello del mio dominio di posta).

UA Server funziona in modo analogo alla fork lato server di vista in TCP e si prepara a ricevere il messaggio e per spedire (FIFO)

Il messaggio viene quindi gestito dall'MTA (port x) del server il quale passa l'indirizzo mail logico (gerla@cs.ucla.edu) ad una specifica struttura (il resolver) che attraverso il DNS trova l'indirizzo IP del destinatario. Poi spedisce il messaggio al server di posta del dominio del destinatario (MTA a dest, si usa well-known port 25).

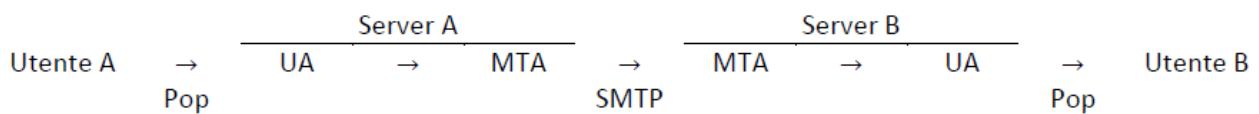
3-I server di posta elettronica comunicano fra di loro.

Lo standard RFC 1939 si occupa di stabilire le regole di comunicazione tra client e server di posta (POP) e delle regole di comunicazione tra server del mittente e server del client (SMTP).

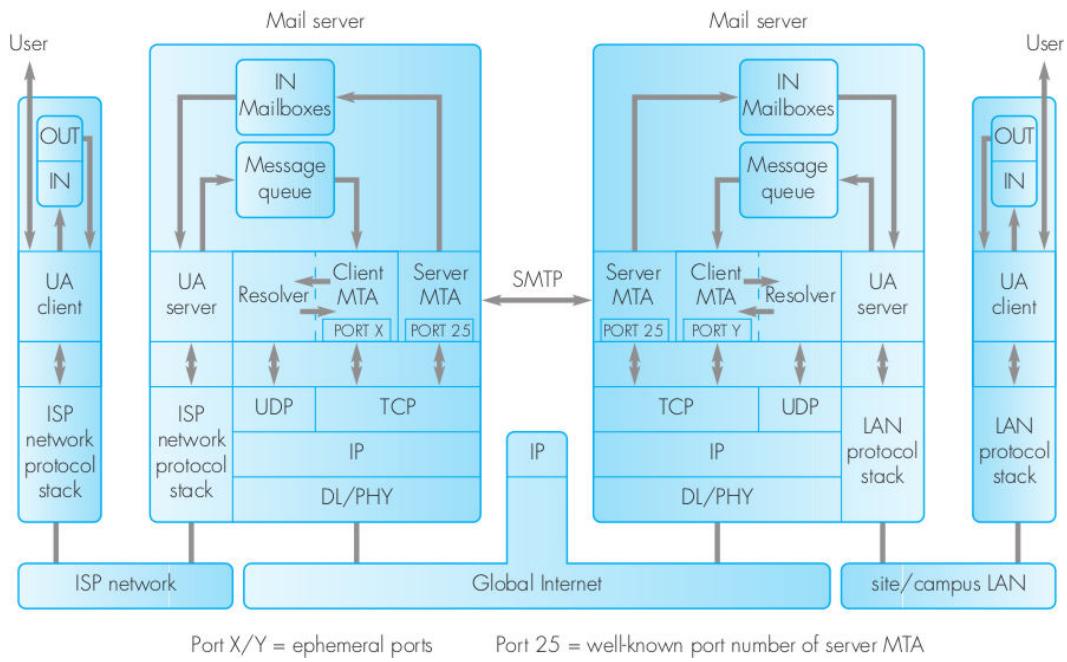
Il destinatario non deve necessariamente essere presente al momento dell'invio.

La comunicazione è asincrona: il messaggio una volta arrivato, rimarrà sul server del destinatario finché questi non lo cancellerà.

Per questo motivo è definita client-server, ma concettualmente, effettua operazione prettamente peer-to-peer.

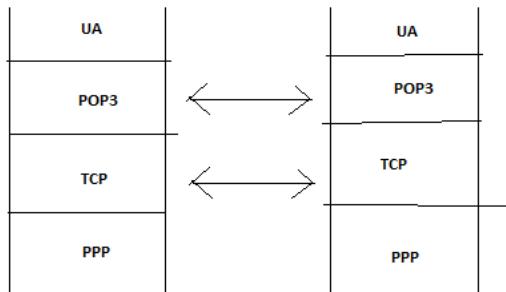


Una procedura analoga è prevista anche dall'altro lato (client/destinatario invia allo stesso modo).



Si vede bene in questa architettura come sono assemblati tutti i servizi e livelli che abbiamo studiato fin'ora.

Esempio Resolver



Rossi@dico.unimi.it (sorg)

gerla@cs.ucla.edu (dest)

Rossi ha mailbox sul server dico.unimi.it

Questo sever non conosce cs.ucla.edu, tantomeno "gerla"

Il server del mittente vuole risolvere cs.ucla.edu

Il server di posta elettronica ha anche lui un resolver quindi prima si cerca di risolvere localmente

Se "gerla" non è riconosciuto, torna indietro una mail di notification fail (il DNS non è riuscito a risolvere)

Cosa viaggia in termini di formato fra le diverse macchine? ASCII (inviamo dei caratteri)

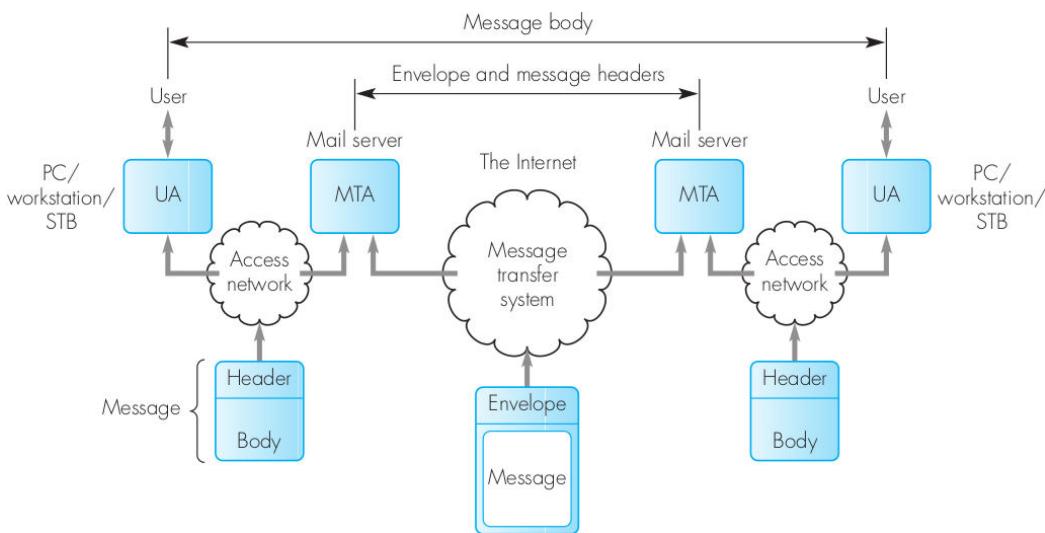
Struttura dei messaggi e-mail

- **Header**, tutti i campi devono avere un formato standard

Used also by the message transfer system	From: To: Cc: Received: Return-Path:	E-mail address of person who created the message E-mail address of primary recipient List of e-mail addresses of other (secondary) recipients Route followed through message transfer system Name of last MTA
Used by UA/user	Sender: Date: Message-Id: Reply-To: Subject:	E-mail address of the sender of the message Date and time message was sent by UA Unique identifier assigned to the message by the UA E-mail address to which a reply should be sent Single-line title of the message
User-defined	X-PhoneNumber: X-FaxNumber:	Sender's phone number Sender's fax number

- **Body**, è la mail vera e propria, cioè tutto ciò che è scritto dall'utente attraverso il proprio UA

Tutti i dati che viaggiano via mail sono nella codifica ASCII (128 caratteri US → 2⁷ bit, vedi tabella più avanti quando si parla di codifica Base64)



In genere, se i dati sono stringhe di caratteri, tutti i campi dell'header e del body sono convertiti in NVT (Network Virtual Terminal) ASCII, codice nel quale:

- ogni carattere è composto da 8 bit: il più significativo è 0 (NUL), i rimanenti 7 sono il codice ASCII
- Esempio**
- 100 0001= A
Diventa: 0100 0001
- Il contenuto del messaggio può essere composto solo da linee con massimo 1000 caratteri NVT ASCII.
 - ogni riga termina con la sequenza "\r\n", cioè un Carriage Return (CR) seguito da un Line Feed (LF)
 - L'ultima riga del messaggio del body finisce con il carattere speciale punto "."

MIME – Multipurpose Internet Mail Extensions (RFC 1341,2045)

Permette l'invio di tipi di dati diversi attraverso la posta elettronica, come dati binari, audio, video ma mantenendo lo stesso sistema di trasferimento dei messaggi.

Con MIME, vengono introdotti opportuni campi aggiuntivi all'header della mail:

- *MIME-Version*, defines the version of MIME that is being used
- *Content-Description*: Short textual description of the message content
- *Content-Id*: Unique identifier assigned by the UA
- *Content-Transfer-Encoding*: The transfer syntax being used
- *Content-Length*: The number of bytes in the message body

Dove:

- **MIME-Version**, quando presente, informa l'UA destinatario che il contenuto della mail non è normale testo NVT ASCII. Se non è presente quindi, si tratta di una normale mail di solo testo. Inoltre include la versione MIME che è stata usata.
- **Content-Type**, descrive il tipo di informazione contenuta nel messaggio, in base a come definito nello standard RFC 1521. Alcuni valori possibili: Text, Image, Audio, Video, Application, Message.

Codifica Base64

Per essere trasferiti, tutti i tipi non ASCII, vengono convertiti in base64, una codifica nella quale:

- ogni dato viene diviso a blocchi di 24 bit
- ogni blocco viene diviso in 3 sottoblocchi da 6 bit ciascuno
- i 3 sottoblocchi sono interpretati come ASCII

Esempio

Un file codificato come 1001 1110 0010 1100 non ha lo 0 come prima cifra, quindi non è ASCII.

Raggruppo in blocchi di 24 bit: se ne ho di meno (ad esempio 16, come in questo caso) allora aggiungo alla fine tante codifiche del carattere "=" equivalente a 0011 1101 in ASCII, per arrivare a 24.

Quindi diventa:

1001 1110 0010 1100 → 1001 1110 0010 1100 0011 1101

Divido in 4 blocchi di 6 bit:

100111 : 100010 : 110000 : 111101

Codifico ogni blocco di 6 bit usando la tabella base64

Binary	ASCII	Binary	ASCII	Binary	ASCII	Binary	ASCII
000000	A	010000	Q	100000	g	110000	w
000001	B	010001	R	100001	h	110001	x
000010	C	010010	S	100010	ı	110010	y
000011	D	010011	T	100011	j	110011	z
000100	E	010100	U	100100	k	110100	0
000101	F	010101	V	100101	l	110101	1
000110	G	010110	W	100110	m	110110	2
000111	H	010111	X	100111	n	110111	3
001000	I	011000	Y	101000	o	111000	4
001001	J	011001	Z	101001	p	111001	5
001010	K	011010	a	101010	q	111010	6
001011	L	011011	b	101011	r	111011	7
001100	M	011100	c	101100	s	111100	8
001101	N	011101	d	101101	t	111101	9
001110	O	011110	e	101110	u	111110	+
001111	P	011111	f	101111	v	111111	/

Quindi:

100111→n

100010→i

110000→w

111101→9

Adesso converto la stringa ottenuta (niw9) di caratteri ASCII in binario:

n → 110 1110

i → 110 1001

w → 111 0111

9 → 011 1001

Bit positions	7	0	0	0	0	1	1	1	1
	6	0	0	1	1	0	0	1	1
	5	0	1	0	1	0	1	0	1
4 3 2 1									
0 0 0 0	NUL	DLE	SP	0	@	P	\	p	
0 0 0 1	SOH	DC1	!	1	A	Q	a	q	
0 0 1 0	STX	DC2	"	2	B	R	b	r	
0 0 1 1	ETX	DC3	#	3	C	S	c	s	
0 1 0 0	EOT	DC4	\$	4	D	T	d	t	
0 1 0 1	ENQ	NAK	%	5	E	U	e	u	
0 1 1 0	ACK	SYN	&	6	F	V	f	v	
0 1 1 1	BEL	ETB	'	7	G	W	g	w	
1 0 0 0	BS	CAN	(8	H	X	h	x	
1 0 0 1	HT	EM)	9	I	Y	i	y	
1 0 1 0	LF	SUB	*	:	J	Z	j	z	
1 0 1 1	VT	ESC	+	;	K	[k	{	
1 1 0 0	FF	FS	,	<	L	\	l		
1 1 0 1	CR	GS	-	=	M]	m	}	
1 1 1 0	SO	RS	.	>	N	^	n	~	
1 1 1 1	SI	US	/	?	O	—	o	DEL	

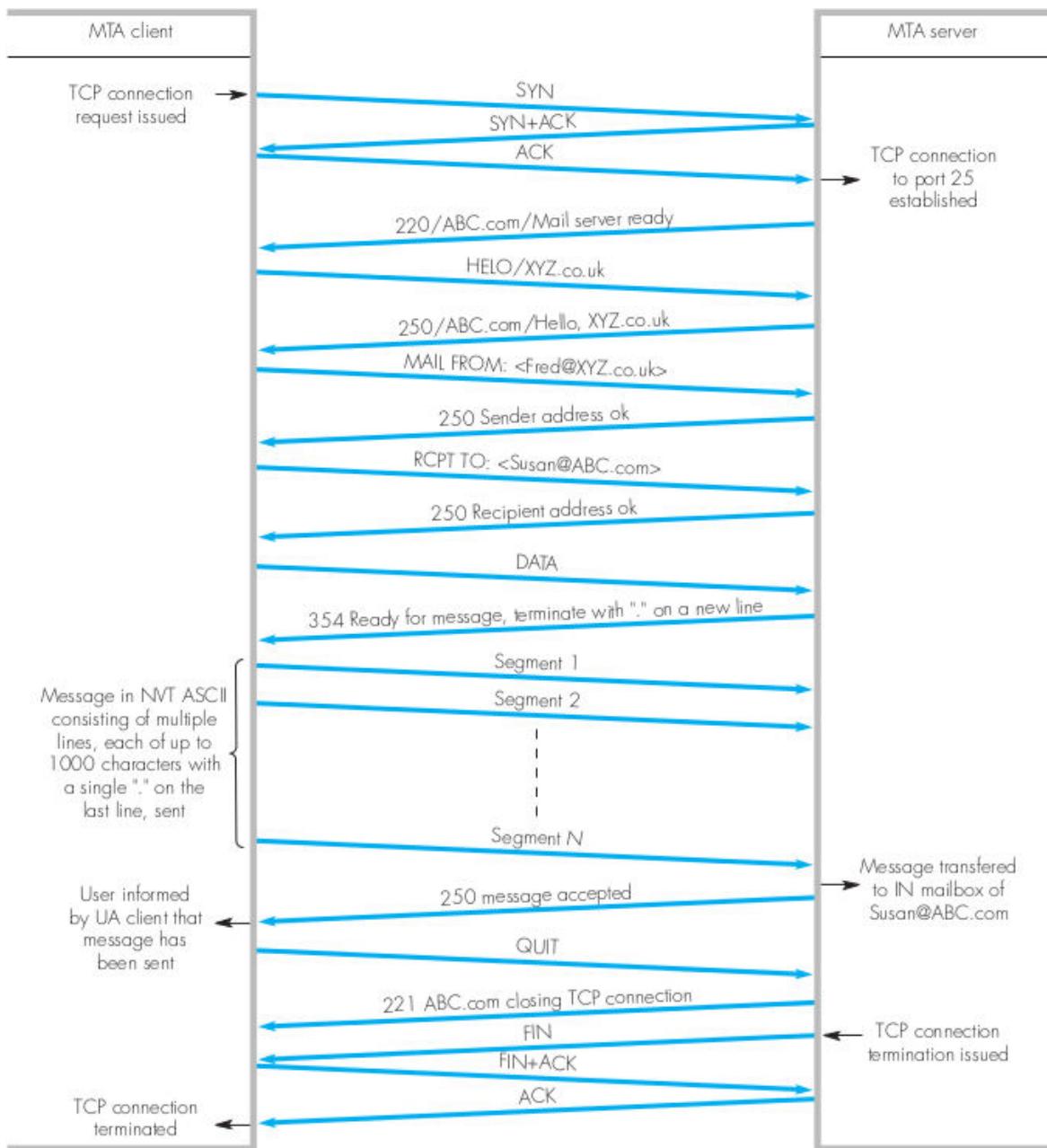
Comunicazione e-mail

Questo è un esempio di comunicazione tra MTA tra il client XYZ.co.uk e il server ABC.com (protocollo SMTP – RFC 821):

Commands (Client MTA → Server MTA)	Descriptions
HELO Mailserver-name	Sends DNS name of the client mail server
MAIL FROM: <e-mail address of sender>	e-mail address of sender
RCPT TO: <e-mail address of recipient>	e-mail address of recipient
DATA	Request to send the message
QUIT	Requests recipient server to close TCP connection
RSET	Abort current mail transfer

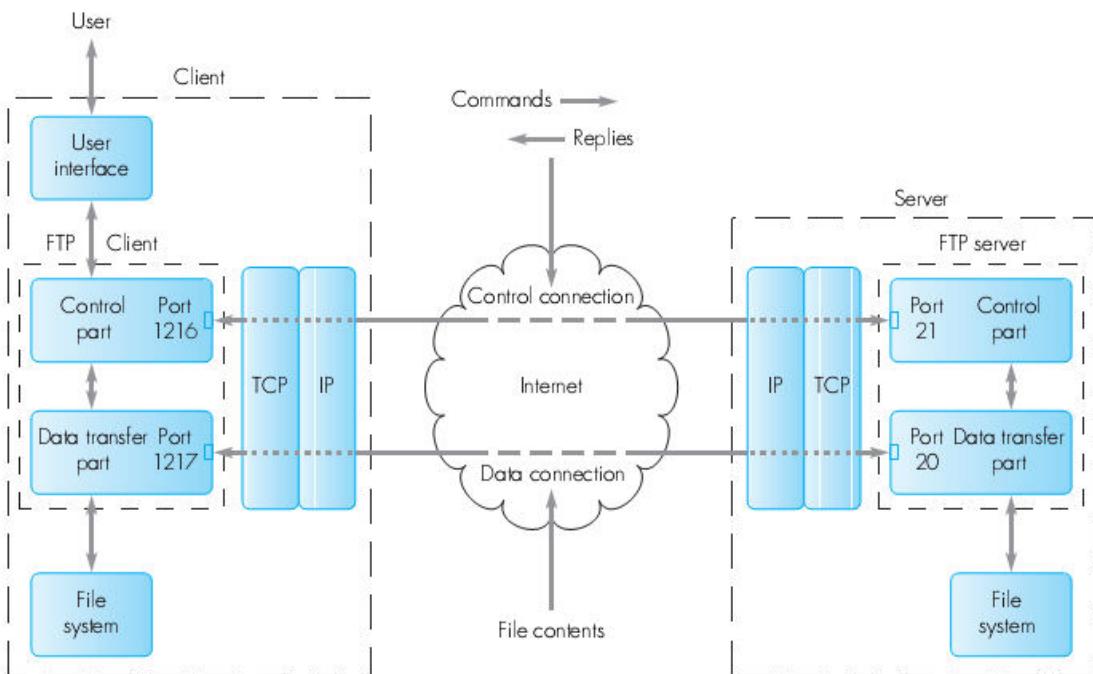
Responses (Server MTA → Client MTA)	Descriptions
220	Recipient server is ready
221	Recipient server is closing TCP connection
250	Command carried out successfully (ACK)
354	Indicates the recipient server is ready to receive message
421	Service request declined
450	Mailbox unavailable
⋮	⋮
551	Addressed user is not here

} Error responses



Tutto questo è svolto attraverso comunicazione TCP.

FTP – File Transfer Protocol



Ports 1216/1217 = ephemeral ports

Port 21 = well-known port of control connection

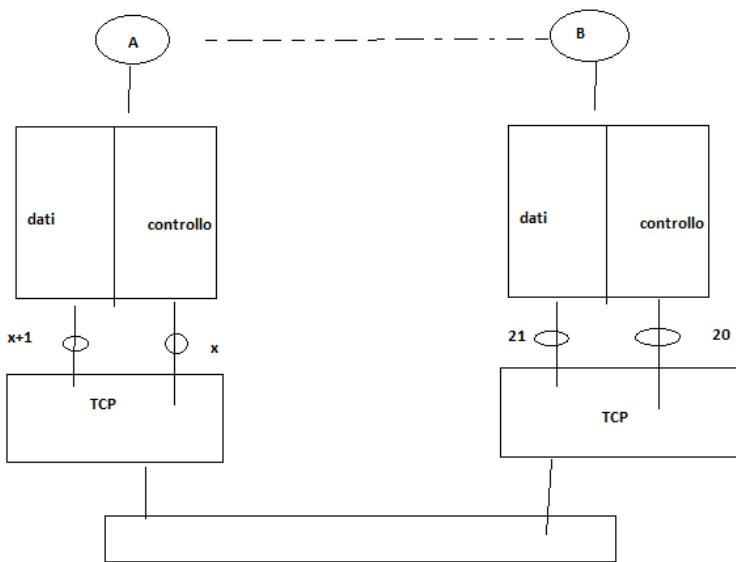
Port 20 = well-known port of data connection

http è un protocollo di file transfert ma FTP è diverso rispetto http, perché è out-of-band!

I dati viaggiano separati dal controllo, viaggiano in due canali (connessioni TCP) distinte.

Porta 21 controllo

Porta 20 dati



La porta 20 viene “stranamente” aperta lato server.

Come fa il server a conoscere la porta $x+1$ del client? Il server deve sapere $x+1$

Connessione di controllo c’è comando “port” in cui esplicita qual è la porta

Alcuni comandi tipici : USER, PASS, TYPE → i parametri sono 6

Esempio

PORT <parameter> → <N1, N2, N3, N4, N5, N6>

I parametri N1, N2, N3, N4 contengono l’IP address:

N1 = 192
N2 = 168
N3 = 1
N4 = 1

I parametri N5 e N6 contengono il numero di porta (16 bit)
Come per subnet abbiamo 8 bit più significativi dicono quanti slot di 255 sto usando, quindi..

N5 = 2 → (numero di slot relativi a N6)
N6 = 110

Come faccio a conoscere la porta?

2 x 256 = 512

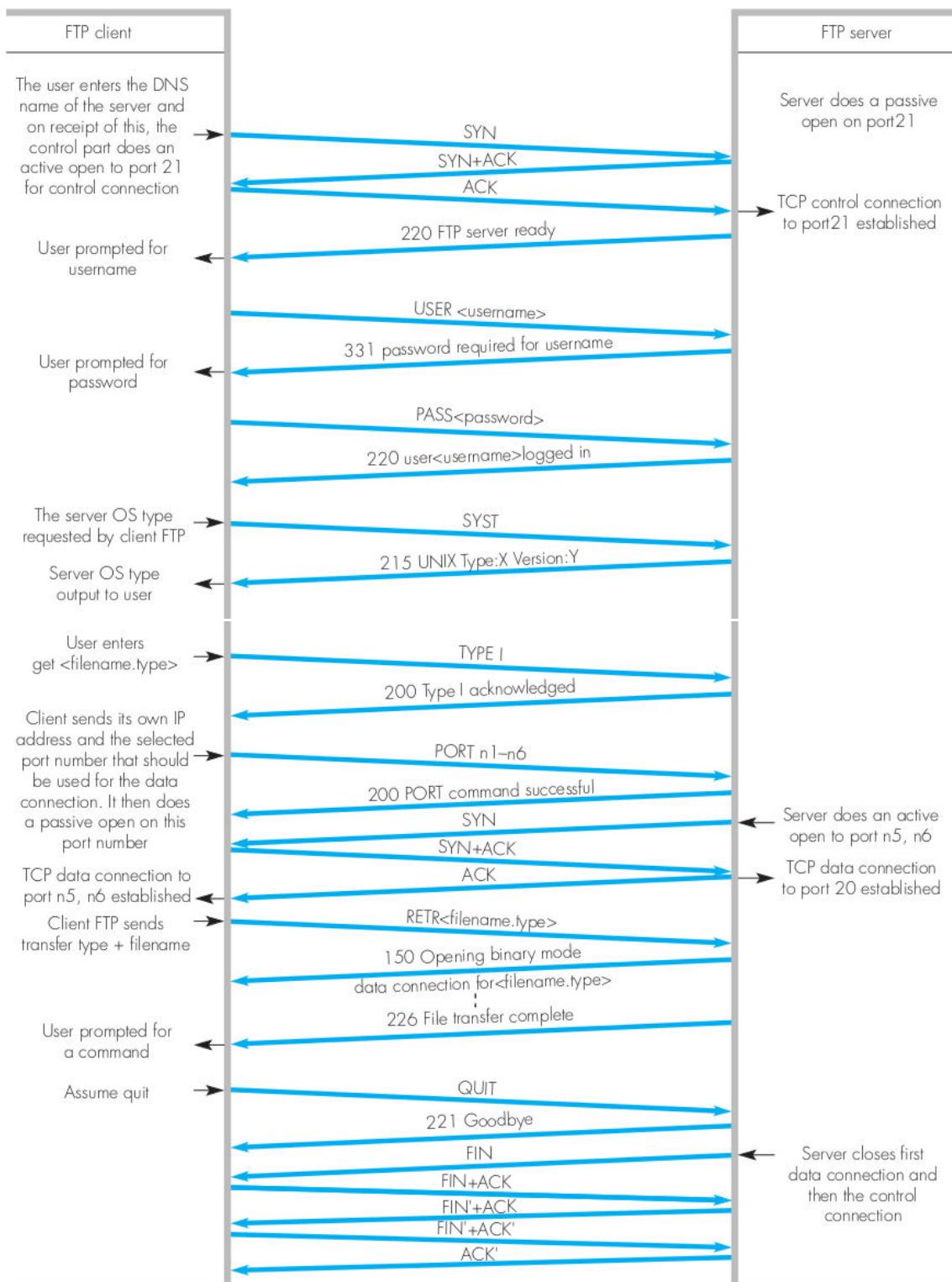
512 + 110 = 622 → porta a cui il server si collega

Comunicazione FTP

220 FTP server ready
331 Password required for <username>
230 User <username> logged in
215 Server OS Name Type: Version
200 File type acknowledged
200 PORT command successful
150 Opening ASCII/Binary mode data connection for <file name>
226 File transfer complete
221 Goodbye
425 Data connection cannot be opened
500 Unrecognized command
501 Invalid arguments
530 User access denied

(a) Command	Description
USER username	User name on the (FTP) server
PASS password	User's password on the server
SYST	Type of operating system requested
TYPE type	File type to be transferred: A (ASCII), I (Image/Binary)
PORT n1, n2, n3, n4, n5, n6	Client IP address (n1–n4) and port number (n5, n6)
RETR filename.type	Retrieve (get) a file
STOR filename.type	Store (put) a file
LIST filelist	List files or directories
QUIT	Log off from server

Questo è un esempio di comunicazione:



Come si vede, il canale dati sulla porta 20 viene aperto dal server dopo che ha richiesto al client il suo indirizzo IP e la porta che vuole usare. Quando la coppia è stata validata, il client effettua un'apertura passiva e il server un'apertura attiva.

HTTP

Protocollo in-band (usa sola comunicazione TCP per dati e controllo)

Come ftp è un file transfer → mascherato perché :

<http://www.dsi.unimi.it/home.html>

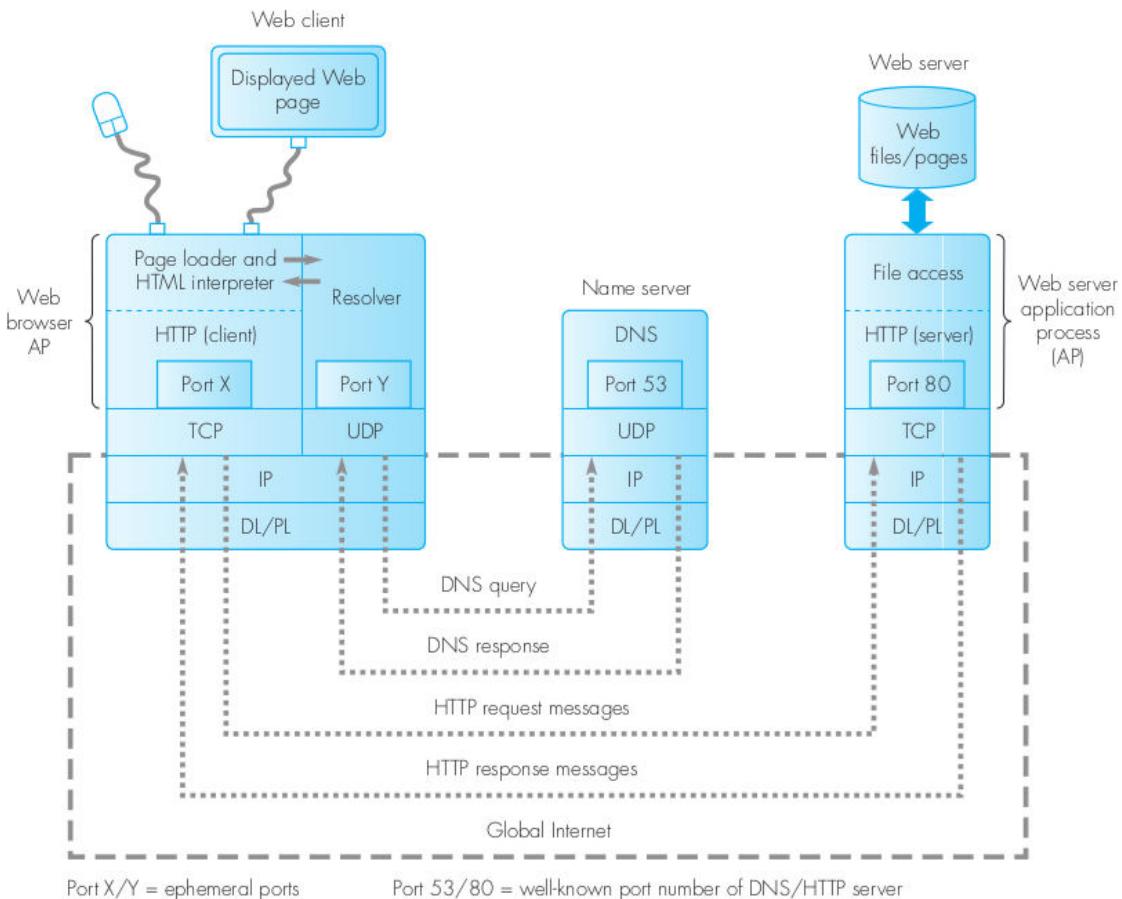
(protocollo://servizio.**dominio**/percorso del file)

Porta per lato client

Porta 80 lato server

Modalità di connessione non persistente agli inizi si apre una nuova connessione ogni volta

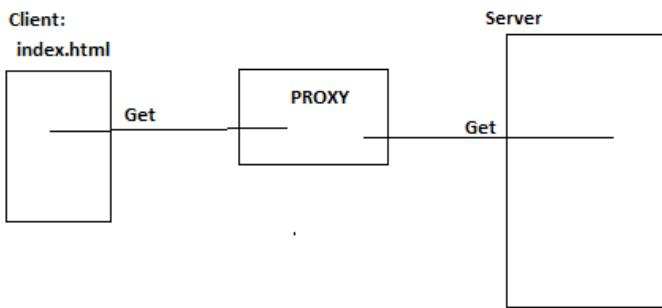
Le connessioni oggi sono “persistent”



Si tratta sempre di file transfert. Passi principali:

1. determinazione URL
2. interrogazione DNS per risoluzione nome
3. attesa risposta DNS
4. connessione TCP sulla porta well-known 80 dell'IP risultato della query DNS
5. GET della pagina richiesta
6. Attesa risposta server
7. Rilascio della comunicazione TCP
8. Visualizzazione immagini
9. Fetch e visualizzazione di specifiche parti delle pagine (immagini, video..)

Inizialmente, ad ogni pagina richiesta dallo stesso server, veniva aperta e chiusa una connessione TCP per ogni risorsa richiesta, causando così continui slow start e apertura di moltissime connessioni uguali. Da http 1.1, le connessioni vengono invece tenute aperte per un quanto di tempo, consentendo l'uso di una sola connessione. Come in SMTP, anche in HTTP i dati vengono codificati in NVT ASCII.



Se c'è un proxy di mezzo, questo verifica se ci sono stati cambiamenti e se ce ne sono stati chiede al server
Il web è molto cachato

Anche in questo caso si usa la codifica NVT

Il browser ha 2 interfacce

Prima di andare diretto dal server, devo andare dal DNS per risolvere gli indirizzi

Esercizio

Supponete che un server web abbia le seguenti caratteristiche:

- tempo di apertura di una connessione TCP: 2 ms,
- tempo di apertura e lettura di un file di dimensioni minori di 8 KB: 1 ms,
- la velocità della linea di connessione Internet: 32 Mbps (si tratta di una grandissima azienda).

Se la richiesta tipica è di una pagina Web di circa 800 parole, **quante richieste al secondo riuscirà a smaltire il server?**

SOL Assumiamo che 800 parole corrispondano a circa 4000 Byte [una parola sarebbe composta da 5 caratteri da 1 B ciascuno, quindi $800 \times 5 = 4000 B$].

Assumiamo inoltre che l'overhead dovuto ai tag HTML e agli header http sia di 1000 B. [il file che viene spedito ha dim tot di $(4 KB + 1 KB = 5KB) < 8 KB$]

Ogni richiesta occuperà il server per circa $2 + 1 = 3 \text{ ms}$, [cioè il tempo di apertura conn TCP + tempo apertura file]

$$[1 \text{ s} / 3 \cdot 10^{-3} \text{ s} = 0,333 \cdot 10^3]$$

Non potranno essere smaltite più di 333 richieste al secondo.

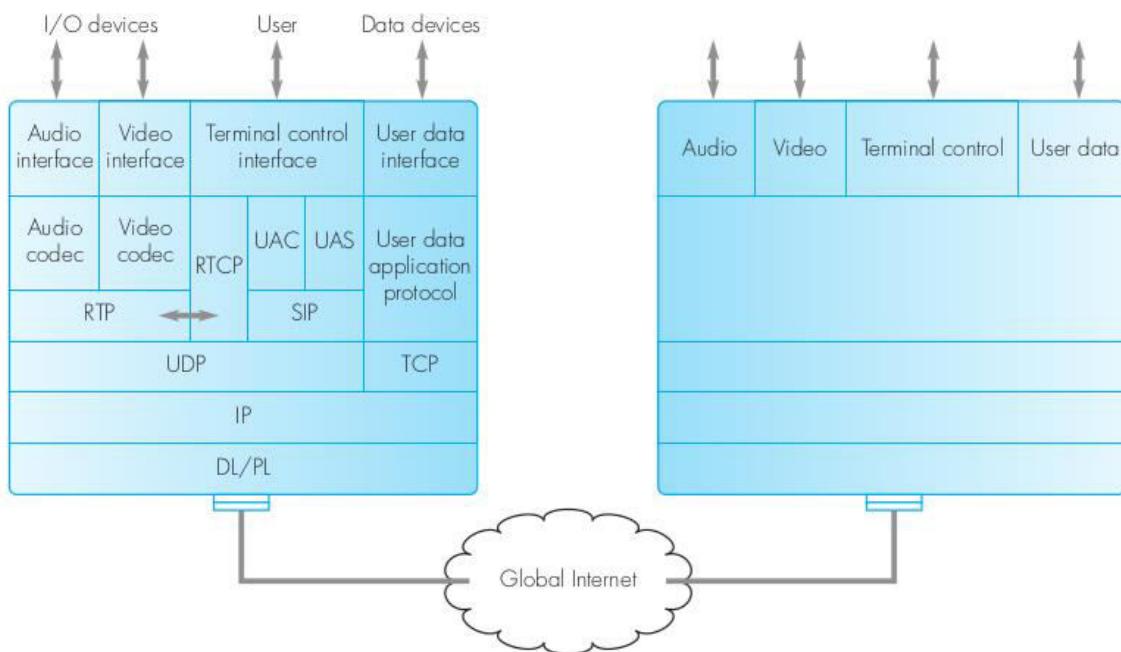
Se ogni risposta richiede la trasmissione di 5000 byte (40000 bit) su una linea a 32 Mbps, **il server potrà trasmettere circa 800 risposte al secondo.** [per ottenere questa risposta, prima trovo il $Tx = f/bwth = 40 \cdot 10^3 \text{ b} / 32 \cdot 10^6 \text{ bps} = 1,25 \cdot 10^{-3} \text{ s}$ Ora vediamo quanti ce ne stanno in un secondo: $1 \text{ s} / 1,25 \cdot 10^{-3} \text{ s} = 0,8 \cdot 10^3$]

Le prestazioni del server sono quindi limitate in questo caso dai tempi di apertura della connessione e di lettura file e non dai tempi di trasferimento.

SIP – Voice Over IP

In telecomunicazioni il protocollo **SIP** (Session Initiation Protocol) è un protocollo di rete basato su IP, definito dalla RFC 3261, e impiegato principalmente per applicazioni di telefonia su IP o VoIP come protocollo di *segnalazione* prima e dopo aver instaurato una comunicazione vocale tra due o più utenti.

SIP gestisce in modo generale una sessione di comunicazione tra due o più entità, ovvero fornisce meccanismi per instaurare, modificare e terminare (rilasciare) una sessione. Attraverso il protocollo SIP possono essere trasferiti dati di diverso tipo (audio, video, messaggistica testuale, ecc). Inoltre, il SIP favorisce un'architettura modulare e scalabile, ovvero capace di crescere con il numero degli utilizzatori del servizio. Queste potenzialità hanno fatto sì che il SIP sia, oggi, il protocollo VoIP più diffuso nel mercato residenziale e business, sorpassando di molto altri protocolli



RTP = real-time transport protocol

RTCP = real-time transport control protocol

UAC = user agent client

UAS = user agent server

SIP = session initiation protocol

SIP message	Usage
INVITE	Invites a user to join a call/session
ACK	Used to acknowledge receipt of an INVITE response message
REGISTER	Used to inform a SIP redirect server of the current location of a user
OPTIONS	Used to request the capabilities of a host device
CANCEL	Terminates a search for a user
BYE	Inform the other user(s) that the user is leaving a call/session

SIP esegue un servizio analogo a quello che si fa con skype.

SIP è un protocollo che serve a fare la segnalazione vocale fra due utenti remoti. È uno standard.

È formato da una pila di I/O che serve per incanalare la voce in una rete digitale attraverso l'utilizzo di un codec audio che si appoggia sul protocollo RTP che si appoggia a sua volta a UDP perché non servono i servizi di affidabilità garantiti da TCP.

Non voglio riinviare un segmento perché non voglio sentire due volte la stessa cosa.

Si tollera la perdita di qualche secondo del collegamento audio. Audio e video viaggiano su UDP.

RTP è il responsabile di chi trasmette e di chi riceve, è colui che decide la dimensione del buffer di ricezione del file.

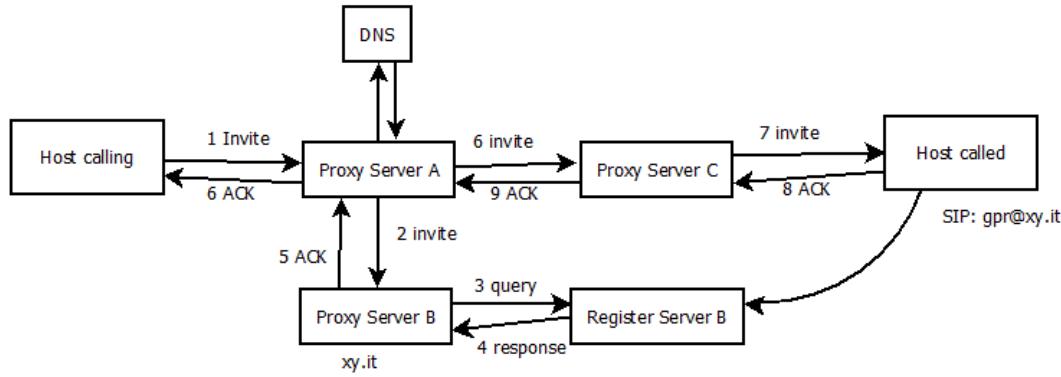
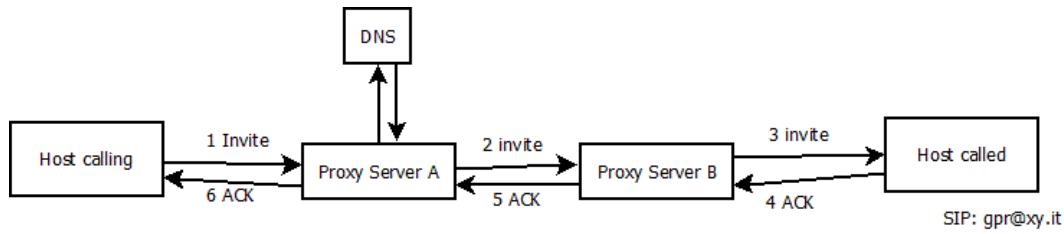
RTCP è il canale di controllo del flusso dati.

Usa UDP perché è più veloce di TCP e perché la perdita di piccoli segmenti è preferibile alla lentezza nello scaricamento dello stream.

Il protocollo SIP gestisce il controllo delle impostazioni standard (chi sono, su quale pc lavora, ...) e si basa su UDP.

I comandi SIP sono: invite, register,... che si impatta su un server SIP a cui mi leggo e specifico le sedi in cui mi posso connettere, ACK, OPTIONS, CANCEL, BYE..

Ogni dominio SIP ha un proxy server e un register server. Con questo schema apro una connessione. Questo è un esempio di traffico real time che è presente ora in internet.



1) Considerare uno schema Go-Back-N con una finestra di 3 frame. Mostrare con un esempio perché 3 soli numeri di sequenza non sono sufficienti.

$$K = 2^n - 1 = 2^2 - 1 = 3$$

Abbiamo scelto $n=2$ bit perché ci bastano per poter rappresentare 3 frame.

00=0

01=1

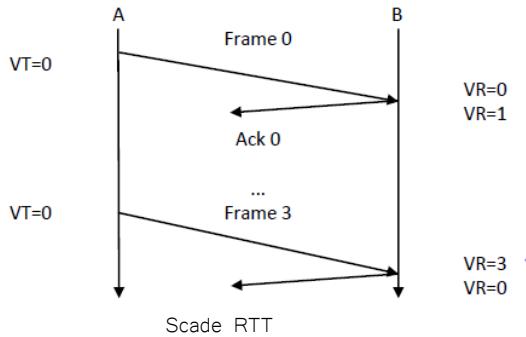
10=2

Ora i numeri di sequenza che ci servono sono

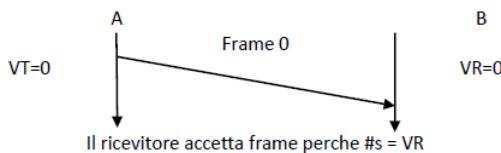
$$K+1 = 4$$

Se si perdono tutti e K gli ACK, mi basta un solo numero in più in modo tale da disaccoppiare la nuova finestra da quella vecchia

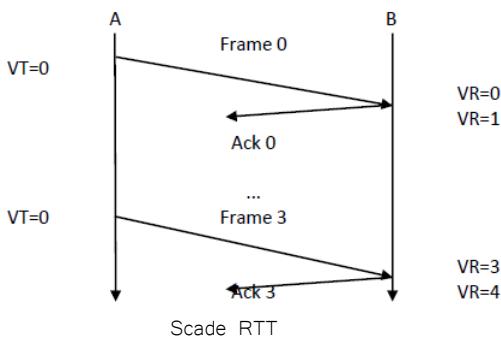
Prendiamo il caso di $\#S = K$



Il timeout scade e poiché non sono arrivati gli ACK, il trasmettitore riinvia i pacchetti. Il ricevitore (che non sa che i suoi ACK sono andati persi) vedrebbe ora i nuovi pacchetti come nuovi frame validi!



Prendiamo ora il caso di $\#S = K + 1$



Lato ricevente $\#S \neq VR$, infatti è $VR(4)$ anziché $VR(0)$.

Il ricevitore dopo aver ricevuto il frame 3 si aspetta il frame 4 che è il frame del nuovo ciclo di finestra.

Quando il mittente riinvierà il frame 0 il ricevente sarà in grado di distinguere il frame nuovo della seconda finestra.

2) Si consideri uno schema Selective Repeat con finestra 8 in cui il ricevitore spedisce un ACK ad ogni frame ricevuta in finestra. Ipotizzando che l'ACK della frame 2 non raggiunga il sender, mostrare il comportamento del protocollo nei due casi di ACK cumulativo e selettivo.

Per avere una finestra di dimensione 8 vuol dire che ho usato 3 bit e quindi $K=2^3$

Lato trasmissione si usano $K/2$ numeri di sequenza (stessa cosa lato destinazione)

Caso di ACK cumulativo:

il mittente nonostante non abbia ricevuto l'ACK per il frame 2, continua a trasmettere i successivi frame e si aspetta di ricevere un ACK che confermi la ricezione di tutti i frame trasmessi fino al numero di sequenza indicato dall'ACK dell'ultimo frame inviato.

Quando viene ricevuto il pacchetto corretto che era stato scartato, per ogni altro pacchetto ricevuto in questo lasso di tempo, viene inviato un solo ack che convaliderà tutti i frame precedentemente spediti.

Se invece il ricevente va in timeout, il mittente sarà costretto a dover ritrasmettere il frame 2, e tutti gli altri frame successivi già trasmessi e ricevuti.

Caso di ACK selettivo:

quando viene ricevuto un pacchetto corrotto, il nodo ricevente entra in retransmission mode. Viene scartato il pacchetto sbagliato ed inviato un NAK (in questo caso indicherà che il frame 2 non è arrivato). I pacchetti successivi al frame 2 sono comunque bufferizzati perché corretti. Quando viene ricevuto il pacchetto corretto che era stato scartato, per ogni altro pacchetto ricevuto in questo lasso di tempo, viene inviata una raffica di ack che li convalida tutti.

3) Descrivere le 4 fasi del protocollo DHCP

4) La tecnica di routing Link State non soffre di count-to-infinity. Perché?

5) Descrivere il comportamento di TCP nella fase di slow start considerando sia il trasferimento corretto di TPDU sia il caso di errore nella trasmissione.

6) Descrivere la codifica di trasferimento Base64

Per essere trasferiti, tutti i tipi non ASCII, vengono convertiti in base64, una codifica nella quale:

- ogni dato viene diviso a blocchi di 24 bit
- ogni blocco viene diviso in 3 sottoblocchi da 6 bit ciascuno
- i 3 sottoblocchi sono interpretati come ASCII

Esempio

Un file codificato come 1001 1110 0010 1100 non ha lo 0 come prima cifra, quindi non è ASCII.

Raggruppo in blocchi di 24 bit: se ne ho di meno (ad esempio 16, come in questo caso) allora aggiungo alla fine tante codifiche del carattere "=" equivalente a 0011 1101 in ASCII, per arrivare a 24.

Quindi diventa:

1001 1110 0010 1100 → 1001 1110 0010 1100 0011 1101

Divido in 4 blocchi di 6 bit:

100111 : 100010 : 110000 : 111101

Codifico ogni blocco di 6 bit usando la tabella base64

Binary	ASCII
000000	A
000001	B
000010	C
000011	D
000100	E
000101	F
000110	G
000111	H
001000	I
001001	J
001010	K
001011	L
001100	M
001101	N
001110	O
001111	P

Binary	ASCII
010000	Q
010001	R
010010	S
010011	T
010100	U
010101	V
010110	W
010111	X
011000	Y
011001	Z
011010	a
011011	b
011100	c
011101	d
011110	e
011111	f

Binary	ASCII
100000	g
100001	h
100010	j
100011	j
100100	k
100101	l
100110	m
100111	n
101000	o
101001	p
101010	q
101011	r
101100	s
101101	t
101110	u
101111	v

Binary	ASCII
110000	w
110001	x
110010	y
110011	z
110100	0
110101	1
110110	2
110111	3
111000	4
111001	5
111010	6
111011	7
111100	8
111101	9
111110	+
111111	/

Quindi:

100111→n

100010→i

110000→w

Adesso converto la stringa ottenuta di caratteri ASCII (niw9), in binario (usando la tabella ASCII):

n → 110 1110
i → 110 1001
w → 111 0111
9 → 011 1001

Bit positions	7	0	0	0	0	1	1	1	1
	6	0	0	1	1	0	0	1	1
	5	0	1	0	1	0	1	0	1
4 3 2 1									
0 0 0 0	NUL	DLE	SP	0	@	P	\	p	
0 0 0 1	SOH	DC1	!	1	A	Q	a	q	
0 0 1 0	STX	DC2	"	2	B	R	b	r	
0 0 1 1	ETX	DC3	#	3	C	S	c	s	
0 1 0 0	EOT	DC4	\$	4	D	T	d	t	
0 1 0 1	ENQ	NAK	%	5	E	U	e	u	
0 1 1 0	ACK	SYN	&	6	F	V	f	v	
0 1 1 1	BEL	ETB	'	7	G	W	g	w	
1 0 0 0	BS	CAN	(8	H	X	h	x	
1 0 0 1	HT	EM)	9	I	Y	i	y	
1 0 1 0	LF	SUB	*	:	J	Z	j	z	
1 0 1 1	VT	ESC	+	;	K	[k	{	
1 1 0 0	FF	FS	,	<	L	\	l	l	
1 1 0 1	CR	GS	-	=	M]	m	}	
1 1 1 0	SO	RS	.	>	N	^	n	~	
1 1 1 1	SI	US	/	?	O	—	o	DEL	

7) Descrivere la funzione della primitiva PORT (con relativi parametri) usata da FTP

Esercizi – dare risposta riportando i passi seguiti per ottenerla

8) Volendo progettare una rete CSMA/CD che opera a 40 Mbps, indicare quale vincolo imposto dallo standard IEEE 802.3 andrebbe modificato per conservare compatibilità di frame?

9) In quanti passi il DNS del dominio *unimi.it* risolve il nome *di.polimi.it* se adotta una politica iterativa (non si ipotizzi uso di cache negli step intermedi)?

10) Sia data una connessione TCP con valori attuali di RTT e RTO rispettivamente uguali a 20 sec e 30 sec. Se il prossimo segmento S1 viene validato da un ACK dopo 40 secondi calcolare i nuovi valori di RTT e RTO ipotizzando parametri alfa e beta uguali a 0.9.

11) Una frame di 2 Kbit deve essere trasmessa su canale di comunicazione in rame da 40 Mbps e lungo 50 Km. Calcolare l'utilizzo del canale se il livello data-link usa un protocollo Selective Repeat con numeri di sequenza rappresentati in un campo di 3 bit.

12) Su una connessione TCP con applicazione interattiva a lato sender sono generati 2 Byte di dati ogni 10 msec. ipotizzando un canale da 80 Kbps e tempo di propagazione di 50 msec , dire quali sono le dimensioni dei primi 3 segmenti inviati sulla connessione se TCP adotta l'algoritmo di Nagle.

[Gli studenti iscritti F1X – NON devono rispondere alla domanda 7 e agli esercizi 8 e 9]

1) In un protocollo a finestra che adotta una politica di ACK cumulativo è utile disporre di un NACK?

La risposta è SI in quanto ce lo dice il formato del frame HDLC. In particolare ricordiamo che all'interno del frame vi è un campo controllo in cui, nel caso di frame di supervisione, i valori possono essere:

1. RR : è un Ack in Go Back N. Ogni frame RR contiene un numero di sequenza che conferma la ricezione corretta.

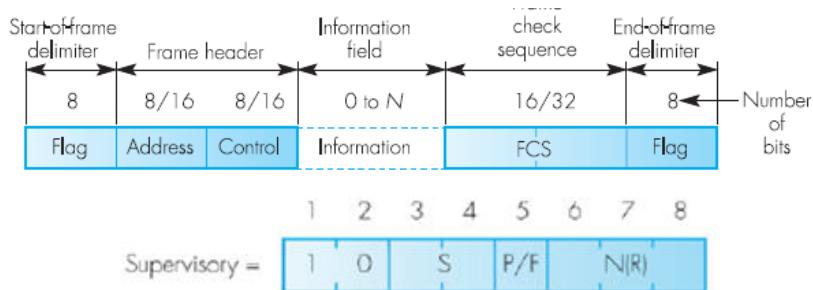
2. RIJ : è un Nack in Go Back N

3. SREJ: è un Nack nel Selective Repeat

4. RNR: funzione che può essere usata dal secondario per comunicare al primario di interrompere la trasmissione di nuovi frame.

Come si vede, i tipi (2) e (3) sono dei NACK che possono essere utilizzati uno per Go Back N e l'altro per Selective Repeat.

Inoltre la procedura di segnalazione di mancata conferma di ricezione (o errore della correttezza) del pacchetto, è una ottimizzazione che serve ad anticipare il mittente per avvertirlo del problema. Non è quindi necessario dover aspettare la scadenza del timer per riinviare il frame perso.



2) Si consideri una rete IEEE 802.3 con bridge e switch. A) Descrivere come un bridge gestisce la propria tabella di forwarding, B) descrivere la differenza fra bridge e switch e C) indicare almeno 1 ragione per preferire un dispositivo all'altro nel progetto di una LAN.

Descrizione di come un bridge gestisce la propria tabella di forwarding:

Per inoltrare le frame verso i domini giusti, il bridge mantiene una tabella (chiamata tabella di forwarding) di indirizzi MAC per ciascuna porta, e in base al suo contenuto è in grado di capire verso quale porta, e quindi quale dominio, inoltrare la frame. La tabella può essere creata manualmente dall'amministratore di rete attraverso apposito software residente, oppure può essere creata automaticamente tramite un meccanismo di auto-apprendimento (*auto-learning*) tramite il traffico progressivo di pacchetti sul bridge associando porta di provenienza con l'indirizzo del mittente. Tale apprendimento può essere reso ancora più sofisticato ed efficiente prevedendo la cancellazione da parte del bridge stesso dell'indirizzo MAC dopo un certo periodo di tempo in cui non viene usato (*tempo di invecchiamento*) evitando così l'aggiornamento manuale e problemi di scalabilità all'aumentare del numero di host nella rete.

All'accensione del Bridge le tabelle degli indirizzi (forwarding) sono vuote perciò al passaggio di una frame questa viene inoltrata su tutte le linee del Bridge (ad eccezione di quella di arrivo) eseguendo quello che viene chiamato Flooding.

In pratica i bridge sono sempre più dispositivi plug-and-play per cui si parla di *bridge trasparenti*.

La differenza principale tra bridge e switch è che il primo lavora a livello 2 mentre il secondo lavora a livello 3 (i router e i bridge sono molto simili però dal punto di vista strutturale).

Bridge	Switch
Differenze	
In fase di apprendimento, lo switch, quando non conosce la porta di uscita, propaga su tutte le porte TRANNE quella da cui ha ricevuto il frame (tipo broadcast).	In fase di apprendimento, lo switch, quando non conosce la porta di uscita, propaga su tutte le porte COMPRESA quella da cui ha ricevuto il frame (tipo broadcast).
Opera a livello 2	Opera a livello 3
	Opera in modalità full-duplex
Fa CarrierSense	Ha collegamenti punto-punto, non ci sono collisioni
Fa CollisionDetection	Tutti i vincoli di lunghezza massima del cavo e velocità di trasmissione non ci sono più, rimangono vincoli per il formato del frame.
In comune	
Usa MAC	
Commuta la linea d'ingresso con una d'uscita, è trasparente e crea la tabella di forwarding	

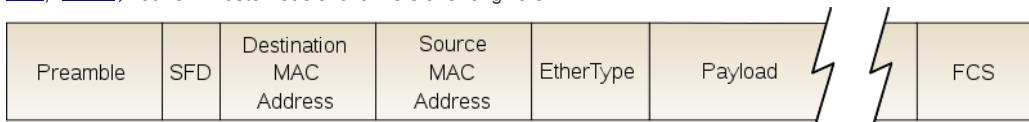
Indicare almeno 1 ragione per preferire un bridge rispetto allo switch:

3) Descrivere le differenze tra una frame IEEE 802.3 e una IEEE 802.1Q

Nella pila di protocolli di rete del modello di riferimento ISO/OSI, 802.3 occupa il livello fisico e la parte inferiore del livello datalink. IEEE ha infatti ritenuto opportuno suddividere questo livello in due parti: LLC, *Logical Link Control* e MAC, *Media Access Control*. Il sottolivello LLC è comune a tutti gli standard della famiglia IEEE 802, mentre il sottolivello MAC è più strettamente legato al livello fisico, e le sue diverse implementazioni hanno il compito di offrire un'interfaccia comune al livello LLC. Fra queste implementazioni vanno ricordate in particolare 802.4, token bus e 802.5, token ring e 802.1Q per VLAN.

Differenze nei formati:

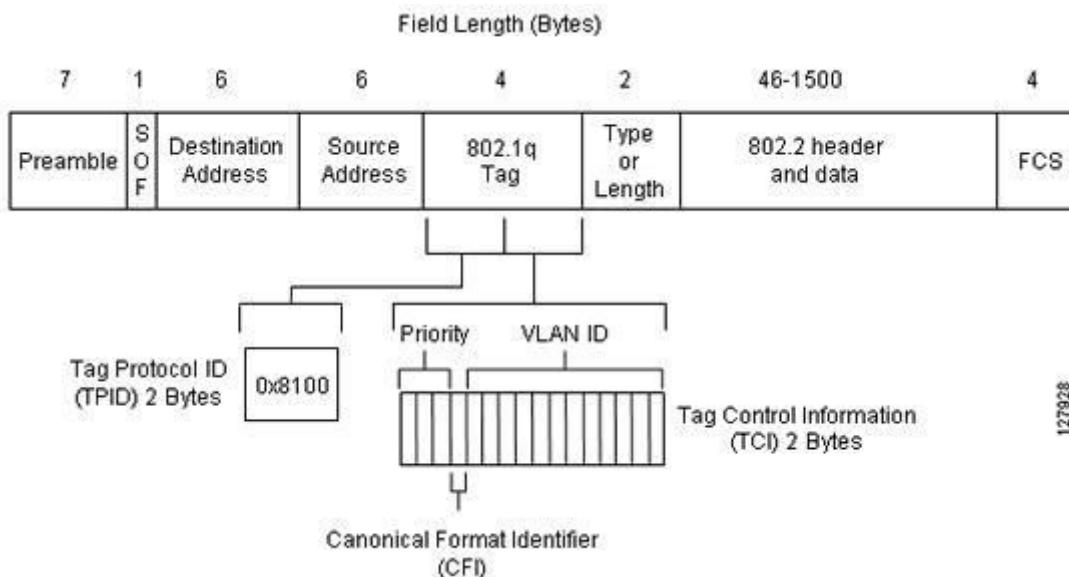
Nonostante Ethernet abbia diverse tipologie, l'elemento comune è nella struttura del [pacchetto](#) Ethernet detto *frame*, che viene definito DIX ([DEC](#), [Intel](#), [Xerox](#)) ed è rimasto fedele alla versione originale.



Questo è il *frame* ovvero il pacchetto dati ricevuto dallo strato di datalink nella pila di protocolli. Gli elementi sono:

- *Preamble* (preambolo), di 7 byte: questi primi byte hanno valore 10101010 e servono a "svegliare" gli adattatori del ricevente e a sincronizzare gli oscillatori con quelli del mittente.
 - *Start Frame Delimiter* (SFD), di 1 byte: questo byte ha valore 10101011 e la serie dei due bit a 1 indica al destinatario che sta arrivando del contenuto importante; è protetto mediante la violazione del codice Manchester; svolge la stessa funzione del campo *flag* della trama HDLC;
 - *Destination MAC address* (indirizzo di destinazione), di 6 byte: questo campo contiene l'indirizzo LAN dell'adattatore di destinazione; se l'indirizzo non corrisponde, il livello fisico del protocollo lo scarta e non lo invia agli strati successivi;
 - *Source MAC address* (indirizzo sorgente), di 6 byte;
 - *EtherType* (campo tipo), di 2 byte: questo campo indica il tipo di protocollo del livello di rete in uso durante la trasmissione, oppure — nel caso di frame IEEE 802.3 — la lunghezza del campo dati;
 - *Payload* (campo dati), da 46 a 1500 byte: contiene i dati reali, che possono essere di lunghezza variabile in base alla MTU della Ethernet; se i dati superano la capacità massima, vengono suddivisi in più pacchetti, mentre se i dati non raggiungono la lunghezza minima di 46 byte, viene aggiunto del *padding* (riempitivo) della lunghezza opportuna;
 - *Frame Check Sequence* (FCS), controllo a ridondanza ciclica (CRC), di 4 byte: permette di rilevare se sono presenti errori di trasmissione; in pratica, il ricevente calcola il CRC mediante un algoritmo e lo confronta con quello ricevuto in questo campo.

Il frame IEEE 802.3 è simile al frame 802.1 ma ad eccezione del campo tipo che diventa *Tipo o Lunghezza* e il preambolo ridotto a 7 byte con 1 byte trasformato in *Start of Frame*.



802.1Q non incapsula il frame originale, ma aggiunge 4 byte all'header.

- I primi 2 byte modificati riguardano il *tag protocol identifier* **TPID**. Questo contiene il tag **EtherType** che viene cambiato in 0x8100, numero che evidenzia il nuovo formato del frame.
 - I successivi 2 byte riguardano il *tag control information* **TCI** (detto anche **VLAN Tag**) così suddiviso:

- **user_priority**: Questo campo a 3 bit può essere utilizzato per indicare un livello di priorità per il frame. L'utilizzo di questo campo è definito in: [IEEE 802.1p](#).
- **CFI**: Campo di 1 bit che indica se i [MAC address](#) nel frame sono in forma canonica.
- **VID**: campo di 12 bit che indica l'ID delle VLAN, che possono così essere fino a 4096 (ossia 2^{12}). Di queste, la prima (VLAN 0) e l'ultima (VLAN 4095) sono riservate, quindi gli ID realmente usabili sono 4094.

4) Descrivere come vengono propagati i pacchetti di stato in un protocollo di routing link state

In informatica e telecomunicazioni 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 ai router di un certo Autonomous System.

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*.

Quando un nodo riceve un *Pacchetto Link State* confronta il numero di sequenza del pacchetto con quello dell'ultimo pacchetto ricevuto da quel nodo:

- se il numero di sequenza indica che il pacchetto è più recente di quello memorizzato, il nuovo pacchetto viene memorizzato e inoltrato a tutti i nodi collegati, eccetto quello da cui è stato ricevuto;
- se il numero di sequenza è invariato il pacchetto viene scartato;
- se il numero di sequenza indica che il pacchetto ricevuto è meno recente di quello memorizzato, quest'ultimo viene trasmesso al nodo mittente.

Ogni nodo memorizza i pacchetti ricevuti e costruisce una mappa completa e aggiornata della rete: il *Link State Database*, ottenendo così gli stessi risultati.

Ogni nodo esegue in maniera indipendente un algoritmo, generalmente una variante dell'Algoritmo di Dijkstra, per determinare il cammino minimo per raggiungere ogni nodo della rete ponendosi come radice dell'albero dei cammini minimi.

Al termine della elaborazione per ogni nodo di destinazione abbiamo il suo predecessore lungo il cammino a costo minimo dal nodo radice.

È possibile costruire la tabella di routing di un nodo memorizzando per ciascuna destinazione il nodo successivo sul cammino a costo minimo. Vantaggi e svantaggi:

L'utilizzo di un algoritmo di routing Link State, presenta diversi vantaggi:

- può gestire reti composte da un gran numero di nodi;
- converge rapidamente al cammino minimo;
- difficilmente genera cammini ciclici;
- è facile da comprendere poiché ogni nodo ha la mappa della rete;

Il principale svantaggio di un algoritmo Link State è la complessità di realizzazione, anche dovuta alla notevole capacità di memoria ed elaborazione richiesti dai router stessi.

5) Elenicare i parametri che vengono concordati dalle entità TCP all'atto dell'apertura della connessione.

Quando si apre una connessione (primitiva *connect()*), ogni entità TCP deve scegliere un numero di sequenza iniziale *ISN* (*Initial Sequence Number*): essi sono entrambi diversi da 0 e cambiano in ogni connessione, garantendo così che un segmento relativo a una specifica connessione non interferisca con un'altra (es. quella successiva). Ogni volta che viene inizializzata una nuova connessione viene "consumato" un diverso numero di sequenza.

Il numero di sequenza viene scelto in base ai bit meno significativi del clock.

Al momento della connessione, gli interlocutori si scambiano informazioni di inizializzazione, per esempio la taglia massima del TDU (Transfer Data Unit).

Per instaurare una connessione, il client invia un primo segmento contenente nel campo Code Bits il bit SYN=1 (tutti gli altri a 0) ed il numero di sequenza scelto.

6) Nella comunicazione fra un host sorgente IPv6 e un host destinazione IPv4, il gateway fra le due reti deve avere la funzione di NAT e Protocol Translator. Descrivere tali funzionalità.

Un tipo di interoperabilità riguarda un host collegato a una rete IPv6 – che quindi ha un indirizzo IPv6 e usa un protocollo IPv6 – che intende comunicare con un host collegato a una rete IPv4 – che quindi ha un indirizzo IPv4 e usa un protocollo IPv4.

In questo caso, indirizzi e formati dei pacchetti sono differenti, pertanto deve essere effettuata un'operazione di traduzione dai router/gateway intermedi. Le traduzioni possono essere effettuate sia al livello di rete sia al livello di applicazione.

Utilizzando il primo metodo i pacchetti IPv6/IPv4 ricevuti vengono convertiti in pacchetti IPv4/IPv6 semanticamente equivalenti. Ciò richiede un traduttore degli indirizzi di rete (NAT, network address translator) e un traduttore dei protocolli (PT, protocol translator). Per questo il gateway intermedio si chiama NAT-PT. Abbiamo già visto come un indirizzo IPv4 può essere incorporato in un indirizzo IPv6 attraverso il tunneling.

Per inviare un datagramma/pacchetto da un host con indirizzo IPv6 a un host con indirizzo IPv4, il NAT nel gateway può rapidamente ottenere l'indirizzo IPv4 di destinazione dall'indirizzo di destinazione nell'intestazione di base del pacchetto IPv6. Il problema è quale indirizzo di origine bisogna indicare nell'intestazione del pacchetto IPv4.

Una soluzione potrebbe essere quella di riservare al NAT un blocco di hostid per la rete IPv4 di destinazione. Gli hostid insieme al netid della rete di destinazione, formano un blocco di indirizzi IPv4 unici. Per ogni nuova chiamata/sessione, il NAT alloca un indirizzo IPv4 non utilizzato di questo blocco per la durata della chiamata/sessione.

Poi crea una voce in una tabella che contiene l'indirizzo IPv6 dell'host V6 e il suo equivalente (temporaneo) indirizzo IPv4.

Il NAT traduce un indirizzo nell'altro quando il pacchetto viene trasmesso. A questi indirizzi è associato un timeout; quindi, se nessun pacchetto viene ricevuto entro l'intervallo di timeout, l'indirizzo viene restituito al blocco degli indirizzi liberi.

Un gateway NAT-PT funziona purché il payload del pacchetto non contenga indirizzi di rete. Questo è il caso di molti protocolli di applicazione. Per esempio, il protocollo dell'applicazione FTP spesso ha gli indirizzi IP incorporati all'interno dei suoi messaggi. In tali casi, quindi, la traduzione deve essere effettuata al livello applicazione. Per questo, il gateway associato viene detto *gateway del livello applicazione* (ALG, *application level gateway*). Ciò richiede un programma di traduzione separato per ogni protocollo di applicazione.

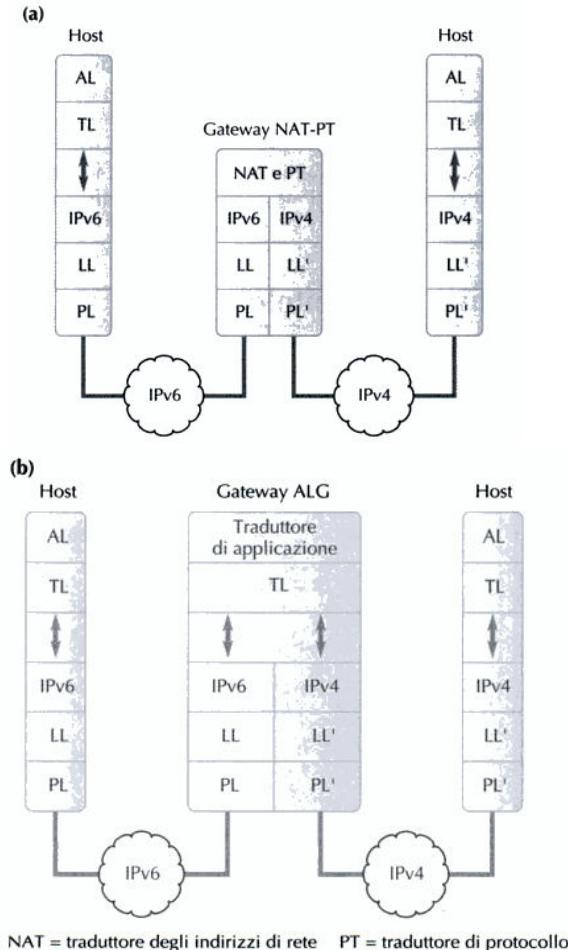


Figura 6.49 Interoperabilità IPv6/IPv4 tramite i traduttori: (a) livello di rete; (b) livello di applicazione.

7) In un router di una backbone area, in base a quali informazioni viene definita l'etichettatura assegnata ad un pacchetto MPLS e indicare se tale etichettatura ha validità locale o per tutto il percorso sorgente-destinazione.

Alla ricezione di un pacchetto da una delle interfacce di ingresso, sappiamo che un router controlla che il pacchetto sia destinato alla sua sottorete controllando l'indirizzo IP in base alla sua address mask.

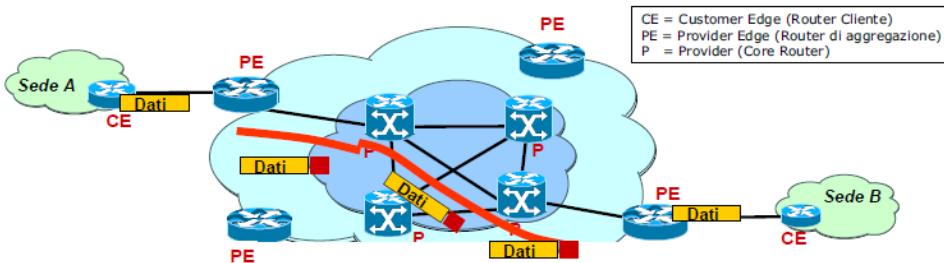
Oltre a questo valore, il pacchetto contiene un campo ToS (Type of Service) che viene gestito da un **packet classifier**.

Quest'ultimo ha il compito di scegliere le regole di scheduling che devono essere applicate al pacchetto:

- controlla nell'header del pacchetto IP il suo ToS
- assegna una etichetta al pacchetto
- aggiunge un nuovo header
- inserisce il pacchetto nella coda di uscita più appropriata

In genere il traffico viene diviso in base al tipo, e non in base al singolo flusso di dati (**DiffServ**). L'idea di MPLS invece è di aggregare il traffico di un flusso ed effettuare un bilanciamento tra questi ultimi; questo comportamento viene applicato nell'area di backbone a cui ogni AS è collegato:

- 1 - ricezione del pacchetto
- 2 - classificazione, etichettatura ed inoltro del pacchetto ricevuto
- 3 - lettura label, applicazione servizi, inoltro all'hop successivo
- 4 - rimozione label e consegna del pacchetto



8) Descrivere il significato del comando PORT 193, 15, 0, 0, 5, 237.

Port è un comando FTP:

PORT <parameter> → <N1, N2, N3, N4, N5, N6>

I parametri N1, N2, N3, N4 contengono l'IP address:

N1 = 193

N2 = 15

N3 = 0

N4 = 0

I parametri N5 e N6 contengono il numero di porta (16 bit)

Come per subnet abbiamo 8 bit più significativi dicono quanti slot di 255 sto usando, quindi..

N5 = 5 → (numero di slot relativi a N6)

N6 = 237

Come faccio a conoscere la porta?

N5 × 256 =

5 × 256 = 1280

1280 + N6 =

1280 + 237 = 1517 → porta a cui il server si collega

La formula completa è:

PORT <parameter> → <N1, N2, N3, N4, N5, N6>

IP = N1.N2.N3.N4

PORT = (N5 × 256) + N6

Esercizi – dare risposta riportando i passi seguiti per ottenerla

9) una stazione connessa ad una rete Ethernet subisce 5 collisioni consecutive nel tentativo di spedire un frame. Dopo quanto tempo al massimo potrà eseguire la successiva ritrasmissione?

SOL

$$2^n - 1 = 31$$

$$31 \times 51.2 = 1587.2$$

Si potrà spedire frame dopo un periodo random tra (0;1587.2 ms)

10) un canale di 300 Km in fibra ha bit rate di 1.2 Mbps. Per quale range di frame size si può ottenere un'efficienza almeno del 60% su tale canale se si adotta idleRQ?

SOL

Dati:

$$b = 1.2 \text{ Mbps} = 1.2 \times 10^6 \text{ bps}$$

$$d = 300 \text{ Km} = 3 \times 10^5 \text{ m}$$

$$u = 0.6$$

$$\text{idleRQ U} = \text{Tx}/(\text{Tx} + 2\text{Tp})$$

$$\text{Tp} = d/v = (3 \times 10^5 \text{ m}) / (3 \times 10^8 \text{ m/s}) = 10^{-3} \text{ s} = 0.001 \text{ s} = 1 \text{ ms}$$

$$0.6 = k \frac{\frac{f}{b}}{\left(\frac{f}{b} + 2\text{Tp} \right)}$$

$$0.6 = 1 \frac{f}{(f + 2b\text{Tp})}$$

$$0.6(f + 2b\text{Tp}) = f$$

$$0.6f + 1.2b\text{Tp} = f$$

$$1.2b\text{Tp} = f - 0.6f$$

$$1.2b\text{Tp} = 0.4f$$

$$\frac{1.2b\text{Tp}}{0.4} = f$$

$$f = 3 \times (1.2 \times 10^6 \text{ b/s}) \times (10^{-3} \text{ s}) = 3.6 \times 10^3 \text{ b} = 3600 \text{ bit}$$

3600/8 = 450 B è la max frame size per un utilizzo del 60%

11) una connessione TCP ha correntemente RTT di 25 msec e RTO pari a 33 msec. Se i segmenti S1 e S2 sono trasmessi con successo e i rispettivi ACK arrivano dopo 28 msec e 31 msec, qual è la nuova stima di RTT usando alfa e beta pari a 0.9?

SOL

$$\text{RTT} = 25$$

$$\text{RTO} = 33$$

$$M_1 = 28$$

$$M_2 = 31$$

RTT_new ?

$$D_{new} = \beta \cdot D_{old} + (1 - \beta) |RTT_{old} - M|$$

$$RTT_{new} = \alpha \cdot RTT_{old} + (1 - \alpha)M$$

$$RTO = RTT_{new} + 4D_{new}$$

Poiche viene chiesta solo la stima di RTT mi serve solo la formula:

$$RTT_{new} = \alpha \cdot RTT_{old} + (1 - \alpha)M$$

Quindi:

$$RTT_{s1} = 0.9 \times 25 + 0.1 \times 28 = 22.5 + 2.8 = 25.3$$

$$RTT_{s2} = 0.9 \times 25.3 + 0.1 \times 31 = 22.77 + 3.1 = 25.87$$

(verificare che l'esercizio è corretto)

12) Una connessione TCP che adotta l'algoritmo di Clark ha dimensione di receiver window 4000B e MSS 1400 B. Se a lato receiver c'è un'applicazione interattiva che consuma 1 B ogni 20 msec e a lato sender c'è sempre disponibilità di dati da spedire, dire ogni quanto il receiver invia window update, assumendo che i ritardi di propagazione siano trascurabili.

SOL

Wr = 4000 B

MSS = 1400 B

$\min\{1400, 4000/2\} = \min\{1400, 2000\} = 1400$ B. [vedi p. 437]

A buffer pieno, 1400B sono consumati in 20 msec.

20 msec x 1400 = 28000 ,cioè 28 sec., che è la frequenza con cui è generato un window update; dopo questo tempo (28 sec), il sender invia un segmento di taglia MSS tornando a riempire il buffer.

[NB. Gli studenti iscritti F1X – NON devono rispondere alle domande 7 e 8 e all'esercizio 9]

Esame 31-01-2011 (Tema A)

1) Descrivere la procedura di apprendimento di un bridge trasparente.

Il bridge trasparente è un bridge che apprende col tempo la dislocazione delle macchine nelle varie LAN.

Per apprendere la dislocazione delle macchine esso costruisce una tabella di forwarding.

Inizialmente la tabella è vuota poi quando arriva un messaggio proveniente da A, il bridge salva nella tabella la porta di A.

Legge quindi che la destinazione è B e poiché la porta della destinazione non è nota, il frame viene inoltrato a tutte le porte tranne quella di provenienza, cioè di A (inoltro in flooding). Questa procedura è giustificata dal fatto che il bridge non sa ancora a quale dominio di collisione appartiene il nodo B (quindi a quale porta è associato).

Se la porta di B è nota ed è la stessa di A, il frame non viene inoltrato, ma raggiunge la destinazione attraverso il bus che collega i nodi terminali dello stesso dominio di collisione.

Se la porta di B è nota ma appartiene a un altro dominio di collisione rispetto a quello di A, il frame viene inoltrato solo sulla porta di B. Una volta raggiunta la destinazione, il nodo B risponde inviando il suo frame di risposta nuovamente al bridge il quale inoltrerà come descritto prima al nodo A.

Periodicamente la tabella di forwarding viene resettata e ricostruita per tenerla aggiornata.

I periodi di refreshing variano in base alla frequenza dei cambiamenti dei nodi collegati al bridge.

2) Alle stazioni di una LAN sono stati assegnati gli indirizzi IP da 10.24.0.0 a 10.24.0.8. Il router che fornisce accesso a Internet svolge funzioni di NAT. Riportare la struttura della tabella NAT nel caso in cui le stazioni 10.24.0.2 e 10.24.0.7 hanno ciascuna una connessione TCP aperta verso il server 194.64.20.0.

Porta In	IP Interno	IP Nat (o esterno)	Nat Port	IP Dest	Port Dest
X	10.24.0.2	10.24.0.8	Y1	194.64.20.0	?
X	10.24.0.7	10.24.0.8	Y2	194.64.20.0	?

3) Descrivere come TCP rileva la condizione di Fast Retransmit e come si comporta per gestirla.

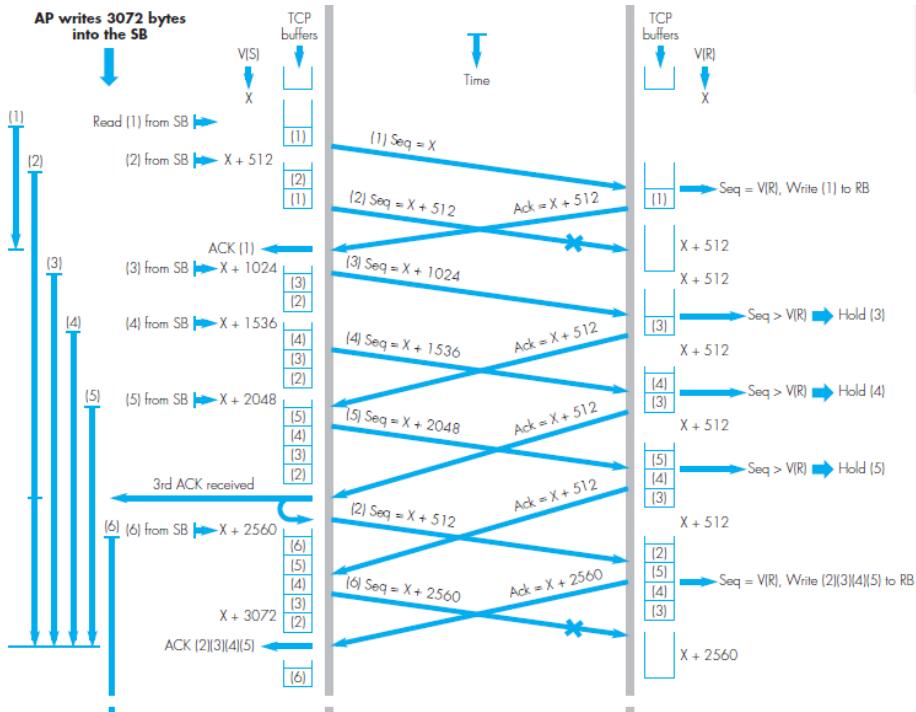
Vediamo come TCP rileva la condizione di Fast Retransmit.

Dato che i segmenti inviati relativi ad una comunicazione TCP viaggiano sulla rete Internet i percorsi seguiti possono essere diversi: conseguentemente i segmenti potrebbero non arrivare nell'ordine nel quale sono stati inviati. Quando al ricevente arriva un segmento fuori sequenza, esso si limita a memorizzarlo temporaneamente e ad inviare un ACK indicando il segmento che si aspetta.

La ritrasmissione del pacchetto non avviene però alla ricezione del primo ACK fuori sequenza.

Si è scelto di aspettare la ricezione di tre ACK fuori sequenza prima di reinviare il segmento mancante. Il numero 3 è dato da un'euristica.

Questa tecnica della ricezione dei tre ACK è detta *Fast Retransmit* e comprende anche l'utilizzo di un timer di ritrasmissione (RTO) associato ad ogni segmento inviato lato sender.



Vediamo ora come si comporta TCP per gestire questo tipo di errore.

La formula che viene utilizzata è la seguente:

$$SST_{new} = \frac{W_c}{2}$$

$$W_c = SST_{old}$$

Vediamolo con un esempio:

Si suppone che la finestra di congestione di TCP sia 18 KB nel momento in cui si verifica un retransmission timeout.

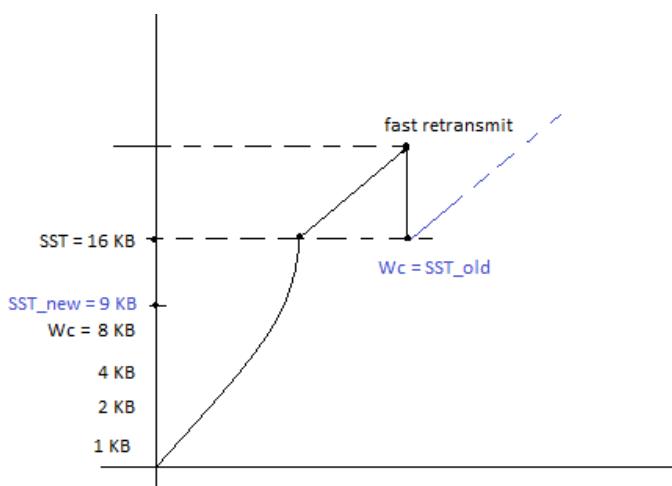
Si assume che la max dimensione di un segmento sia 1 KB e la SST = 16 KB.

Quali saranno i valori della nuova SST e W_c ?

SOL

$$SST_{new} = 18/2 = 9 \text{ KB}$$

$$W_c = 16 \text{ KB.}$$

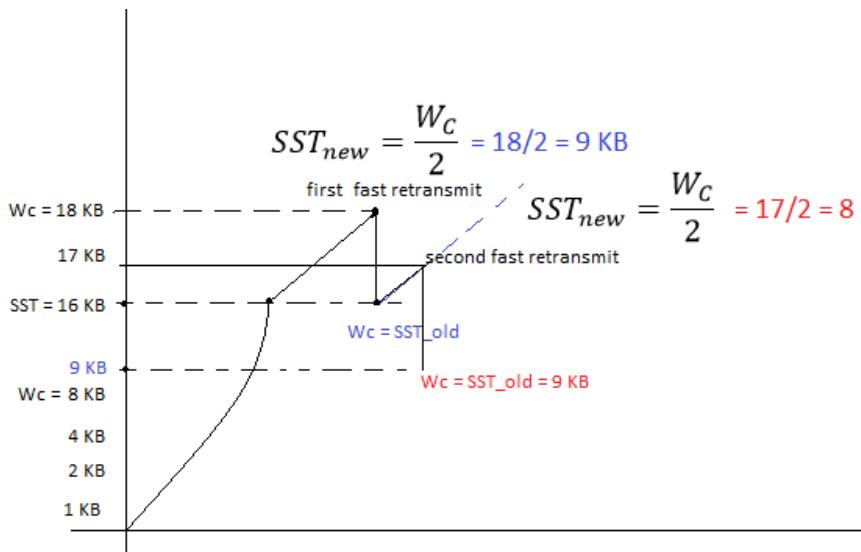


Se adesso trasmettessi correttamente 2 segmenti avrei:

$$W_c = 17 \text{ KB}$$

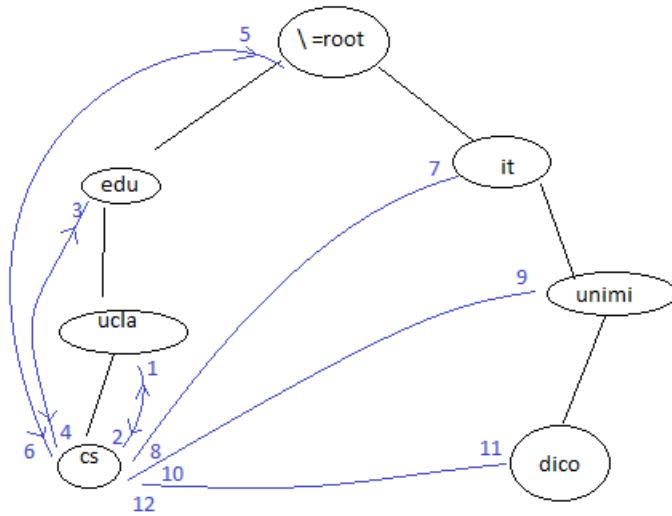
$$W_c = 18 \text{ KB}$$

Se adesso si verificasse nuovamente un retransmission timeout avrei:



4) In quanti passi il DNS del dominio cs.ucla.edu risolve il nome dico.unimi.it se adotta una politica iterativa (non si ipotizzi uso di cache negli step intermedi)?

12 passi



--> = contatta

Giusto per ripasso, se fosse stata ricorsiva sarebbe stato:

1 - Il name-server cs.ucla.edu richiede all'indirizzo root l'indirizzo di "it"
2 - root risponde fornendo l'indirizzo di it

3- Il ns cs.ucla.edu richiede al ns di it l'indirizzo di unimi

4- it risponde fornendo l'indirizzo di unimi

5- Il ns cs.ucla.edu richiede al ns di unimi l'indirizzo di dico

6- unimi risponde fornendo l'indirizzo di dico

7- Il ns cs.ucla.edu chiede di risolvere la richiesta a dico

8- dico risponde risolvendo la richiesta

In totale 8 passi.

5) Considerate la politica di ritrasmissione adottata da TCP e le due tecniche Go-Back-N e Selective Repeat. Indicare le eventuali analogie che rilevate nel confronto fra TCP e le due tecniche citate.

6) Il routing basato su Link State non soffre del problema del count to infinity. Spiegare perché e indicare quale sostanziale differenza fra i due protocolli consente di ottenere il risultato.

Ospf ha pacchetti con aging e num seq

Quiz 1)

Un canale ha bit rate 1 Mbps e ritardo di propagazione di 2 msec. Le frame usano un campo di sequenza di 2 bit. Per quale dimensione di frame GoBackN ha efficienza pari al 100%?

- a) 2480 bit
- b) nessuno di questi valori
- c) 2000 bit
- d) 2500 bit

SOL

$$b=1\text{Mbps} = 1 \times 10^6 \text{ bps}$$

$$Tp = 2 \text{ msec} = 2 \times 10^{-3} \text{ sec}$$

$$K_{gbn} = (2^n)-1 = (2^2)-1 = 4-1 = 3$$

$$U = k \frac{T_x}{T_x + 2Tp}$$

$$1 = k \frac{T_x}{T_x + 2Tp}$$

$$k \frac{\frac{f}{b}}{\left(\frac{f}{b} + 2Tp\right)} = 1 \rightarrow k \frac{f}{(f + 2bTp)} = 1$$

$$\frac{1}{(f + 2bTp)} = \frac{1}{fk} \rightarrow \frac{f + 2bTp}{fk} = \frac{1}{f}$$

$$f + 2bTp = fk$$

$$2bTp = f * k - f$$

$$2bTp = f(k - 1)$$

$$\frac{2bTp}{k - 1} = f$$

$$2bTp = 2 \times (1 \times 10^6 \text{ b/s}) \times (2 \times 10^{-3} \text{ sec}) = 4 \times 10^3 \text{ b}$$

$$k-1 = 2$$

$$f = 4000 \text{ b / 2} = 2000 \text{ bit}$$

Quiz 2)

I campi Source (S) e Destination (D) Address di un pacchetto DHCP Discover contengono:

- a) S: 0, D:broadcast
- b) S: IP sorgente, D: IP Server DHCP
- c) S: IP sorgente, D: IP Destinazione
- d) S: MAC Sorgente, D: IP Destinazione

Quiz 3)

Una connessione TCP ha RTT=24 msec e RTO=32 msec e l'ultimo segmento trasmesso S_n genera una misura $M=30$ msec. Indicare (i) i nuovi valori di RTT e RTO usati per S_{n+1} e (ii) quelli usati per la ritrasmissione di S_{n+1} nella ipotesi che la prima trasmissione generi un retransmission timeout.

- a) (i) RTT=21.6 msec e RTO=33.14 msec; (ii) RTT=20 msec e RTO=64 msec
- b) (i) RTT=24.6 msec e RTO=33.96 msec; (ii) RTT=24.6 msec e RTO=67.92 msec

- c) (i) RTT=20.8 msec e RTO=34.8 msec; (ii) RTT=20.8 msec e RTO=69.6 msec
d) (i) RTT=20.8 msec e RTO=35.68 msec; (ii) RTT=20 msec e RTO=64 msec

SOL

RTT = 24

RTO = 32

M = 30

$$D_{new} = \beta \cdot D_{old} + (1 - \beta) |RTT_{old} - M|$$

$$RTT_{new} = \alpha \cdot RTT_{old} + (1 - \alpha)M$$

$$RTO = RTT_{new} + 4 D_{new}$$

Calcolo RTT e RTO per la trasmissione di S_n+1

$$RTO = RTT + 4D$$

$$D = (RTO - RTT)/4$$

$$D = (32 - 24)/4 = 2$$

$$D = 0.9 \cdot 2 + 0.1 \cdot |24 - 30| = 1.8 + 0.6 = 2.4$$

$$RTT = 0.9 \cdot 24 + 0.1 \cdot 30 = 21.6 + 3 = 24.6$$

$$RTO = 24.6 + 4 \cdot 2.4 = 34.2 \text{ (nelle soluzioni proposte è } 34.8!?)$$

Calcolo RTT e RTO per la ritrasmissione di S_n+1 nel caso di RTO

Usando Karn, raddoppio RTO e RTT rimane invariato:

$$RTO = 2 \cdot RTO_{old} = 2 \cdot 34.2 = 68.4$$

$$RTT = 24.6$$

Quiz 4)

Si consideri una LAN Gigabit Ethernet. Indicare la lunghezza massima prevista dallo standard fra stazione e hub e la dimensione minima della frame.

- a) 400 m, 40 bit
- b) 800 m, 512 bit
- c) 200 m, 512 bit
- d) 2500 m, 500 byte

Quiz 5)

Si supponga che un IP access gateway debba instradare su una LAN Ethernet un pacchetto di 4200 B di dati. Indicare il contenuto del campo fragment offset nei frammenti IP generati.

- a) 1500, 3000, 1200
- b) 0, 1500
- c) 0, 185, 370
- d) 185, 370, 1240

SOL

MTU Ethernet = 1500 B

Header di 20 B

1500 - 20 = 1480 B per payload

Data = 1480, FragmentOffset = 0 B, TotalLength = 4200 B, M = 1

Data = 1480, FragmentOffset = 1480/8 = 185 B, TotalLength = 4200 B, M=1

Data = 1240, FragmentOffset = (1480+1480=) 2960/8 = 370 B, TotalLength = 4200 B, M=0

Si supponga che un IP access gateway debba instradare su una LAN Ethernet un pacchetto di 3000 B di dati. Indicare il contenuto del campo fragment offset nei frammenti IP generati.

- a) 1500, 3000, 1200
- b) 0, 1500

- c) 0, 185, 370
d) 185, 370, 1240

SOL

MTU Ethernet = 1500 B
Header = 20B
Payload = 1500-20 = 1480 B

Data = 1480, FragmentOffset = 0 B, TotalLength = 3000 B, M=1
Data = 1480, FragmentOffset = 1480/8 = 185 B, TotalLength = 3000 B, M=1
Data = 40, FragmentOffset = (1480+1480=) 2960/8= 370 B, TotalLength = 3000 B, M=0

Quiz 6)

FTP è un protocollo out of band fra un client (che apre la connessione di controllo) ed un server (che l'accetta). Quale dei due end-point si occupa di aprire la connessione dati fra quali porte?

- a) Il server fra la propria porta 20 e la porta che il client segnala in precedenza sul canale di controllo
- b) Il client che apre sia la connessione di controllo che quella dati contemporaneamente
- c) Il server fra la porta 20 del client e la sua porta 21
- d) Il client fra la sua porta 20 e la porta 20 del server

Quiz n.6 Tema A

Un canale ha bit rate 1 Mbps e ritardo di propagazione di 5 msec. Le frame usano un campo di sequenza di 3 bit. Per quale dimensione di frame Selective Repeat ha efficienza almeno 80%?

SOL

$T_x = f/b$
 $T_p = 5 \text{ msec} = 5 \cdot 10^{-3} \text{ s}$
 $U = (k) \cdot T_x / (T_x + 2T_p)$
 $K = (2^n)/2 = (2^3)/2 = 8/2 = 4$

$$k \frac{\frac{f}{b}}{\left(\frac{f}{b} + 2T_p\right)} = 0.8 \rightarrow k \frac{f}{(f + 2bT_p)} = 0.8$$

$$\frac{1}{(f + 2bT_p)} = \frac{0.8}{fk} \rightarrow \frac{f + 2bT_p}{fk} = \frac{1}{0.8}$$

$$f + 2bT_p = \frac{fk}{0.8}$$

$$0.8(f + 2bT_p) = fk$$

$$0.8f + 1.6bT_p = fk$$

$$1.6bT_p = fk - 0.8f$$

$$1.6bT_p = f(k - 0.8)$$

$$f = \frac{1.6bT_p}{(k - 0.8)}$$

$$1.6bT_p = 1.6 \cdot (1 \cdot 10^6 \text{ bps}) \cdot (5 \cdot 10^{-3} \text{ s}) = 8 \cdot 10^3 \text{ b} = 8000 \text{ bit}$$

$$k - 0.8 = 4 - 0.8 = 3.2$$

$$f = 8000/3.2 = 2500 \text{ bit}$$

1) Su una connessione TCP la sorgente genera un carattere ogni 3 msec, per un totale di 30 caratteri. Quanti segmenti invia TCP per trasferire i 30 caratteri se il round trip time è stabilmente fissato in 15 msec?

- a) 1
- b) 6
- c) 30
- d) 7

SOL

$3 \cdot 30 = 90$ ms (tempo necessario per inviare 30 caratteri)

Dato che TCP utilizza Nagle abbiamo che mentre l'entità TCP lato client aspetta la ricezione dell'ACK, il buffer d'invio si riempie così che, quando sarà nuovamente possibile inviare (cioè alla ricezione dell'ACK spedito dal server), più dati utili potranno essere inviati in un solo segmento.

Quindi in 90 ms invio:

90ms/15ms = **6 segmenti errata!**

In questo esercizio c'è chi dice che la soluzione è:

Io avrei risposto la A

In quanto TCP aspetta fino a 200ms per vedere se qualche dato viene posto nel buffer di invio dal processo locale.

Però non ne sono convinto di tale ragionamento.

(sembra ineleggibile giusto se considerassi anziché l'algoritmo di Nagle, il semplice *delayed acknowledgement*, in cui il tempo per accumulare dati nella finestra è 200 ms che è maggiore dei 90 ms necessario per inviare 30 caratteri)

Avrebbe avuto senso ma cmq si sarebbe spedito 2 segmenti.

Il primo lo invio subito poi aspetto i 200 ms e invio gli altri 29.

Non c'è una risposta con 2 segmenti... Quindi....

E c'è chi dice che la soluzione è:

1° invio RTT = **15**; D = unk; RTO = 3sec:

[1 carattere inviato] ACK_1 dopo 15

2° invio RTT = 15; D = 7,5 RTO = 45

[6 caratteri inviati] ACK_2 dopo 15

3° invio RTT = 15; D = 6,75 RTO = 42

[11 caratteri inviati] ACK_3 dopo 15

4° invioRTT = 15 D = 6,075 RTO = 39,3

[16 caratteri inviati] ACK_3 dopo 15

5° invioRTT = 15 D = 5,4575 RTO = 36,83

[21 caratteri inviati] ACK_3 dopo 15

6° invioRTT = 15 D = 4,92075 RTO = 34,6828

[26 caratteri inviati] ACK_3 dopo 15

7° invioRTT = 15 D = 4,428675 RTO = 32,7147

[30 caratteri inviati] ACK_3 dopo 15

La risposta D

Calcoli non necessari.. Meglio....

La risposta giusta è la D:

1 invio : 1 segmento in 15.

2 invio: 5 segmenti in 15

3 invio: 5 segmenti in 15

4 invio: 5 segmenti in 15

5 invio: 5 segmenti in 15

6 invio: 5 segmenti in 15

7 invio: 4 segmenti in 15

RTT è costante ed è 15

Il primo segmento lo invio subito, in quanto non è specificato che viene utilizzato un algoritmo particolare (tipo .

2) Un canale in rame lungo 2 Km ha banda 6 Mbps. Per quale range di frame size Idle RQ ha efficienza almeno 50%?

- a) 610 bit
- b) 540 bit
- c) 120 bit
- d) 312 bit

SOL

I dati:

$$d = 2 \text{ Km} = 2 \cdot 10^3 \text{ m}$$

$$T_p = 2 \cdot 10^3 \text{ m} / 2 \cdot 10^8 \text{ m/s} = 10^{-5} \text{ s} = 10 \text{ micros}$$

$$bwth = 6 \cdot 10^6 \text{ bps}$$

$$U\text{-idle} = Tx/(Tx + 2Tp)$$

Vogliamo che l'efficienza sia del 50% e ci chiediamo che valore ha f della formula $Tx=f/bwth$

Quindi:

$$50\% = 0.5$$

$$0.5 = (f/b)/[(f/b) + 2Tp]$$

$$0.5 = \left(\frac{\frac{f}{b}}{\frac{f}{b} + 2Tp} \right)$$

Moltiplico per b sia a num che a denom (parte destra):

$$0.5 = \frac{f}{f + 2bTp}$$

Divido da entrambe le parti per f

$$\frac{0.5}{f} = \frac{1}{f + 2bTp}$$

Ora, moltiplico da entrambe le parti per il denominatore del secondo membro

$$\frac{0.5(f + 2bTp)}{f} = 1$$

$$\frac{0.5f + bTp}{f} = 1$$

$$0.5f + bTp = f$$

$$2bTp = f$$

$$f = 2 \cdot 6 \cdot 10^6 \text{ bps} \cdot 10^{-5} \text{ s} = 120 \text{ bit}$$

3) Lo header di IEEE 802.3 prevede un campo PAD di dimensione 0-46. A che cosa serve?

- a) A contenere i bit di ridondanza per la rilevazione degli errori
- b) A contenere il Payload Data del frame
- c) A contenere l'estensione dello header in caso di VLAN
- d) A garantire che il tempo di trasmissione dei frame sia superiore ad un minimo stabilito dallo standard

4) A cosa serve il campo "intestazione successiva" dello header IPv6?

- a) Equivale al campo "fragment offset" di IPv4, che indica la posizione del prossimo byte nello stream a cui appartiene il pacchetto
- b) In realtà IPv6 non ha un campo "intestazione successiva", come IPv4
- c) Come link allo header TCP che segue quello IPv6 oppure uno o più header estesi previsti dallo standard
- d) In IPv6 "intestazione successiva" ha la stessa funzione del bit *urgent* di IPv4

5) Una coppia sender-receiver a livello data-link adotta un protocollo a finestra P con tecnica Go-Back-N. Se lo header per P prevede al massimo 7 bit per il campo sequence number, qual è la dimensione massima utilizzabile per la finestra di trasmissione?

- a) 128
- b) 127
- c) 255
- d) 64

SOL

Abbiamo 7 bit per #seq e siamo in GoBackN, quindi si ha:

$$K = (2^n) - 1 = (2^7) - 1 = 127 \text{ lato trasmissione}$$

Ora devo considerare il lato ricezione che ha K+1

Infatti per Go-Back-N è #SEQ = K+1

Quindi dovrebbe essere #SEQ = 128 nooooooo! È solo lato trasmissione!

6) Una connessione TCP ha RTT=72 msec, D=3 e RTO=84 msec quando si verifica un retransmission timeout. Indicare quali sono i valori di RTT e RTO: (i) usati per la ritrasmissione con successo che produce misura M=104 msec; (ii) usati per la trasmissione con successo del segmento successivo.

- a) (i) RTT=60 msec e RTO=128 msec; (ii) RTT=60 msec e RTO=128 msec
- b) (i) RTT=72 msec e RTO=128 msec; (ii) RTT=63.10 msec e RTO=77.86 msec
- c) (i) RTT=72 msec e RTO=168 msec; (ii) RTT=72 msec e RTO=168 msec
- d) (i) RTT=72 msec e RTO=168 msec; (ii) RTT=75.2 msec e RTO=97.52 msec

SOL

$$\text{RTT} = 72$$

$$D = 3$$

$$\text{RTO} = 84$$

Infatti:

$$\text{RTO} = \text{RTT} + 4D$$

$$\text{RTO} = 72 + 4 \cdot 3 = 84$$

Per la prima ritrasmissione, in accordo a Karn, il RTT non viene aggiornato mentre RTO è raddoppiato, ovvero $\text{RTO} = 84 \cdot 2 = 168$ msec.

Quindi :

$$\text{RTT} = 72 \text{ msec}$$

$$\text{RTO} = 168 \text{ msec}$$

(Tutte le volte che nell'esercizio delle stime di TCP vi è un retransmission timeout, si parte con Karn. Se non fosse avvenuto alcun retransmission timeout, allora avrei potuto sin da subito utilizzare la formula con le stime..)

Calcolo i valori successivi, quelli del secondo segmento, per stima:

(ora abbiamo anche il valore di M=104)

$$D_{\text{new}} = \beta \cdot D_{\text{old}} + (1 - \beta) |RTT_{\text{old}} - M|$$

$$RTT_{\text{new}} = \alpha \cdot RTT_{\text{old}} + (1 - \alpha)M$$

$$RTO = RTT_{\text{new}} + 4 D_{\text{new}}$$

Segmento 2)

$$D = 0.9 \cdot 3 + 0.1 \cdot |72 - 104| = 2.7 + 3.2 = 5.9$$

$$RTT = 0.9 \cdot 72 + 0.1 \cdot 104 = 64.8 + 10.4 = 75.2$$

$$RTO = 75.2 + 4 \cdot 5.9 = 98.8 \quad (\neq 97.52 \text{ mi viene diverso!})$$

7) Quale conoscenza della topologia della rete si costruisce un router che utilizza l'approccio Distance Vector per l'instradamento?

- a) L'intera topologia della rete
- b) Il next hop per raggiungere ogni nodo della rete e una stima della propria distanza da esso
- c) Le identità dei nodi vicini e le destinazioni che ognuno di essi conosce
- d) I path completi per raggiungere ogni destinazione in rete

8) Nella libreria delle socket, quale funzione svolge la primitiva accept()?

- a) Accetta la richiesta di connessione ed esegue la fork() del processo Server per attivare un processo figlio associato ad una specifica richiesta di connessione

- b) Crea le strutture dati lato Server necessarie a gestire il numero specificato di richieste di connessione al Server
c) Pone il Server in stato di attesa fino alla ricezione di una richiesta di connessione
d) Pone il Client in stato di attesa fino alla ricezione di una primitiva di Confirm da parte del Server

Esame 24-02-2010 (Tema A)

- 1) Come viene calcolato il valore dello slot time considerato da Ethernet?

O-----N-----N-----N-----O 500 m da nodo a nodo

$$Tp = d/v = 500 \text{ m} / 2 \times 10^8 = 250 \times 10^{-8} = 2.50 \times 10^{-6} \text{ s} = 2.5 \text{ microsec}$$

$$2.5 \times 5 = 12.5$$

Infatti se considero il Tp per 2.5 Km ho:

$$Tp = d/v = 2500 \text{ m} / 2 \times 10^8 = 1250 \times 10^{-8} = 12.5 \times 10^{-6} \text{ sec} = 12.5 \text{ microsec}$$

$$Tx + 2Tp = RTT$$

Tp non varia.....

Mentre Tx? Viene solo elaborato dal host mittente:

$$Tx = f/bwth$$

$$Tx = (64 \times 8) \text{ bit} / 10^7 \text{ bps} = 512 / 10^7 \text{ bps} = 51.2 \times 10^{-6} \text{ s} = 51.2 \text{ microsec} = \text{slot time}$$

$$Tx = 2Tp + k = 2 \times 12.5 + k = 25$$

Mancano ancora 25 microsec... dove sono?

Se ho 4 ripetitori all'andata e 4 al ritorno, facendo 25 (parte rimanente) / 8 = circa 3.125

Questo è il ritardo introdotto da ciascun ripetitore per attraversarlo.

Tornando allo slot time...

Se considerassi la ethernet da 100 Mbps

Devo scegliere cosa cambiare:

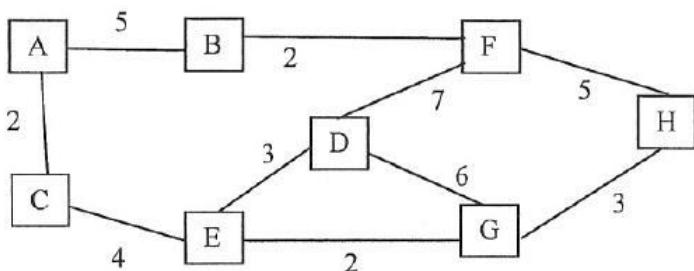
-posso mantenere lo stesso slot ma dovrei cambiare la frame size

-posso mantenere la stessa frame size minima di 64 B ma in questo caso cambia lo slot time.

$$Tx = f/b$$

$$F=Tx \times b = 51.2 \times 10^{-6} \times (10^8 \text{ bps}) = 5120 \text{ bit} = 5120 / 8 = 640 \text{ B}$$

- 2) Usando l'algoritmo di Dijkstra, trovate il cammino di costo minimo tra i nodi A e H nel grafo seguente, *mostrando tutti i passaggi della computazione.*



- 3) Illustrare la procedura per la chiusura di una connessione TCP, dando motivazione delle informazioni scambiate e dei passaggi effettuati.

- 4) Descrivere come, nella comunicazione sincrona, il clock del ricevitore si sincronizza con quello del trasmettitore all'atto della ricezione di una frame.

Gli studenti di informatica per le telecomunicazioni NON rispondono alle seguenti domande

5) Il server svrA.dsi.unimi.it è il web server del Dipartimento, ed è un host con sistema operativo Ubuntu che ha come alias il nome www.dsi.unimi.it. L'indirizzo IP di svrA.dsi.unimi.it è 1234::5678:E5A3. Da questa descrizione della rete ricavare i DNS resource record opportuni.

6) Le versioni recenti di http adottano una politica persistente di gestione delle connessioni TCP. Motivare le ragioni di tale scelta rispetto alle versioni con connessioni non persistenti.

Esame 24-02-2010 (Tema B)

1) Per quale ragione la dimensione minima di una frame Ethernet è fissata a 64B?

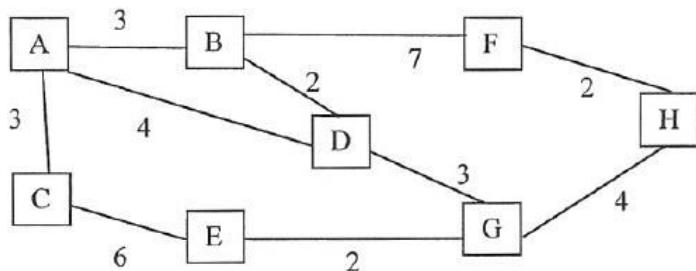
Lo standard IEEE 802.3 impone che t_x sia maggiore o uguale a 51.2 us, chiamato **slot-time**, che include, oltre a $2 t_p$, il ritardo dei repeater e una maggiorazione "per stare tranquilli". È calcolato pensando alla massima lunghezza di una rete Ethernet, cioè 2.5 Km. In 51.2 us riesco a mandare (minimo) 64 Byte mantenendo la rilevazione della collisione, quindi tutti i frame di Ethernet devono avere almeno questa lunghezza.

$$Tx = 2Tp = f/bwth$$

$$F = 2Tp \times bwth = (51.2 \times 10^{-6} \text{ s}) \times (10 \times 10^6 \text{ bps}) = 512 \text{ bit}$$

(64 Byte corrispondono a circa 512 bit che è la max frame size in una Ethernet che ha banda di 10 Mbps e $2Tp = 51.2$ microsec.)

2) Usando l'algoritmo di Dijkstra, trovare il cammino di costo minimo tra i nodi A e H nel grafo seguente, mostrando tutti i passaggi della computazione.



Queste sono tutte le strade per arrivare ad H... ma non è ciò che viene richiesto

A-3-B

$$B - (3+2=5) - D$$

$$D - (5+3=8) - G$$

$$G - (8+4=12) - H$$

$$B - (3+7=10) - F$$

$$F - (10+2=12) - H$$

A-3-C

$$C - (3+6=9) - E$$

$$E - (9+2=11) - G$$

$$G - (11+4=15) - H$$

A-4-D

$$D - (4+2=6) - B$$

$$B - (6+7=13) - F$$

$$F - (13+2=15) - H$$

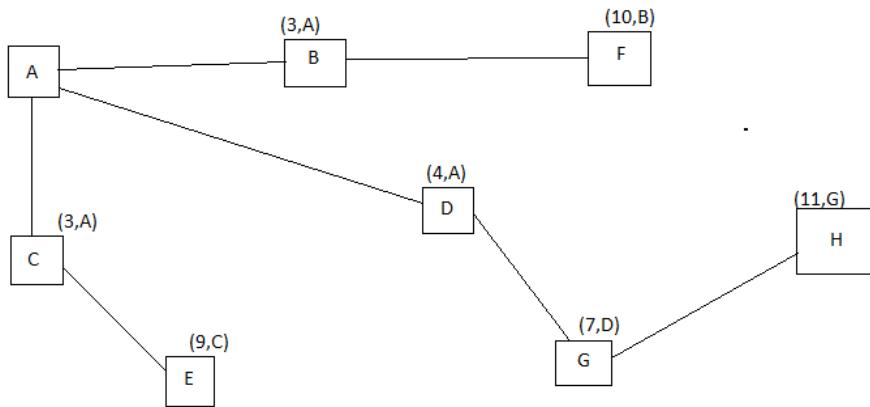
$$D - (4+3=7) - G$$

$$G - (7+4=11) - H$$

Il cammino di costo minimo è A-D-G-H con costo 11.

Se calcolassi il percorso minimo per tutte le destinazioni, troverei in modo più semplice il percorso minimo per arrivare ad H

Vedi disegno.



3) In che cosa consiste il problema della silly window che può presentarsi in TCP, e come si risolve?

La silly window syndrome (sindrome da finestra sciocca, abbreviata con SWS) è un problema legato alla cattiva implementazione del [controllo di flusso](#) a livello [TCP](#). Un processo di scrittura molto lento da parte del mittente nel buffer di trasmissione (o di lettura da parte del ricevente) porta infatti all'invio di segmenti di dati molto piccoli, aumentando così il rapporto tra header e dati con un conseguente uso inefficiente del canale.

Sindrome causata dal mittente

Nel caso in cui il processo di scrittura dei dati nel [buffer TCP](#) del mittente sia molto lento, il protocollo spedirà una serie di pacchetti contenenti una quantità di dati molto bassa, con un uso inefficiente del canale (è infatti molto meglio spedire un solo pacchetto con n byte di dati, per il quale bisogna pagare il peso di un solo header, che spedire n pacchetti contenenti solo un byte di dati, per ognuno dei quali bisogna pagare il peso di un header, un rapporto di $1/n$ contro $n/n=1$).

La soluzione a questo problema consiste nel trattenere i dati nel [buffer](#) allo scopo di spedirli in un unico segmento. Tuttavia un'attesa troppo lunga potrebbe causare dei ritardi troppo grandi nella trasmissione.

Un'ottima soluzione a questo problema è fornita dall'[algoritmo di Nagle](#), secondo il quale i dati devono essere accumulati nel buffer per poi venire spediti in un unico blocco alla ricezione dell'[ACK](#) dell'ultimo pacchetto trasmesso o quando si raggiunge la massima dimensione fissata per un segmento ([MSS](#)). Questo semplicissimo algoritmo riesce a risolvere il problema tenendo anche conto della velocità di trasmissione dei pacchetti: se questa è più lenta della scrittura dei messaggi (il mittente riesce ad accumulare una notevole quantità di dati nel buffer prima dell'arrivo del [riscontro](#)) vengono creati pacchetti con il massimo rapporto dati/header, sfruttando al meglio le risorse del canale. Se invece la rete è più veloce, i pacchetti risulteranno più piccoli, assicureranno una certa continuità nella trasmissione e verrà garantito comunque un utilizzo più efficiente delle risorse del canale che nel caso in cui l'algoritmo non venga utilizzato.

4) Descrivere come, nella comunicazione asincrona, il clock del ricevitore si sincronizza con quello del trasmettitore all'atto della ricezione di un carattere.

Le trasmissioni ethernet utilizzano una famosa codifica per le trasmissioni in banda base, detta *codifica Manchester*. In questo caso è previsto che la velocità del clock sia doppia rispetto ai bit trasmessi. Manchester infatti garantisce una transizione di stato a metà di ogni bit.

Gli studenti di informatica per le Telecomunicazioni NON rispondono alle seguenti domande

5) Il webserver del Dipartimento risiede sul server www.dico.unimi.it, che ha come alias il nome `servweb.dico.unimi.it`, che è una macchina HP con sistema operativo Debian. L'indirizzo di `servweb.dico.unimi.it` è 171.24.132.87. Da questa descrizione della rete ricavare i DNS resource record opportuni.

<code>www.dico.unimi.it</code>	CNAME	IN	172800	<code>servweb.dico.unimi.it</code>
<code>servweb.dico.unimi.it</code>	HINFO	IN	172800	HP server, Debian OS
<code>servweb.dico.unimi.it</code>	A	IN	172800	171.24.132.87

6) Le prime versioni di HTTP adottano una politica non persistente di gestione delle connessioni TCP. Argomentare gli svantaggi di tale approccio.

L'approccio non persistente veniva adottato nelle prime versioni di HTTP. La transazione HTTP è composta da uno scambio di richiesta e risposta e il client apre una connessione TCP separata per ogni risorsa richiesta:

- il client apre la connessione TCP con il server
- il client invia il messaggio di richiesta HTTP sulla connessione
- il server invia il messaggio di risposta HTTP sulla connessione

- la connessione TCP viene chiusa

Svantaggi:

Si ha overhead per l'instaurazione e l'abbattimento della connessione TCP: per ogni trasferimento di risorsa, 2 round trip time (RTT) in più

Si ha overhead per il meccanismo slow start del TCP: la finestra ha dimensione pari a 1 all'inizio di ogni nuova connessione TCP

Esame 29-01-2008

1) Si calcoli l'utilizzo di rete fornito da un protocollo di livello Data-Link che adotta l'approccio Selective Repeat su un canale in rame di 100 Km, con banda di 40 Mbps, frame size di 600 bit e numeri di sequenza rappresentati con 4 bit.

- a) circa 11.8%
- b) circa 23.6%
- c) quasi 5%
- d) più del 8.6%

SOL

$$d = 100 \text{ Km} = 100 \cdot 10^3 \text{ m} = 1 \cdot 10^5 \text{ m}$$

$$\text{bwth} = 40 \text{ Mbps} = 40 \cdot 10^6 \text{ bps}$$

$$f = 600 \text{ b}$$

$$\#\text{seq} = 4 \text{ bit}$$

$$U = K \cdot [Tx/Tx+2Tp]$$

Quindi..

Con selective repeat ho:

$$K = 2^n/2 = 16/2 = 8 \text{ (che è la dimensione della finestra)}$$

$$Tx = f/bwth = 600 \text{ b} / 40 \cdot 10^6 \text{ bps} = 15 \cdot 10^{-6} \text{ sec}$$

$$Tp = d/v = 10 \cdot 10^4 \text{ m} / 2 \cdot 10^8 \text{ m/s} = 5 \cdot 10^{-4} \text{ s}$$

$$U = K \cdot [Tx/Tx+2Tp]$$

$$U = 8 \cdot [15 \cdot 10^{-6} \text{ sec} / (15 \cdot 10^{-6} \text{ sec} + 2 \cdot 5 \cdot 10^{-4} \text{ s})]$$

$$U = 8 \cdot [15 \cdot 10^{-6} \text{ sec} / 15 \cdot 10^{-6} \text{ sec} + 10 \cdot 10^{-4} \text{ s}]$$

$$U = 8 \cdot [15 \cdot 10^{-6} \text{ sec} / 10^{-4} \cdot (0.15 + 10)] =$$

$$U = 8 \cdot [15 \cdot 10^{-6} \text{ sec} / 10^{-4} \cdot 10.15] = 8 \cdot [1.48 \cdot 10^{-2}] = 11.84 \cdot 10^{-2} = 0.1184$$

$$U = \underline{\underline{11.84 \%}}$$

2) Un amministratore di rete deve gestire una LAN di campus a cui è assegnato l'indirizzo di rete di classe B 146.73.0.0. Assumendo che la rete debba essere suddivisa in sottoreti, ognuna delle quali deve poter connettere fino a massimo 60 host, e connesse tra loro attraverso router, indicare: (i) quante sottoreti al massimo si possono indirizzare, e (ii) una appropriata subnet mask per le sottoreti.

a) si possono indirizzare 1023 reti; netmask = 11111111 11111111 11111111 11000000

b) si possono indirizzare 1023 reti; netmask = 11111111 11111111 11111111 11100000

c) si possono indirizzare 2047 reti; netmask = 11111111 11111111 11111111 10000000

d) si possono indirizzare 2047 reti; netmask = 11111111 11111111 11111111 11000000

SOL

I bit riservati per l'host identifier sono gli ultimi 6 bit dell'ultimo ottetto.

Con 6 bit infatti si possono indirizzare $2^6 = 64$ host.

La rappresentazione in binario dell'indirizzo base è:

```
146 | 2   r=0
 73 | 2   r=1
 36 | 2   r=0
 18 | 2   r=0
  9 | 2   r=1
   4 | 2   r=0
   2 | 2   r=0
```

1 | 2 r=1

146 = 1001 0010

73 | 2 r=1
36 | 2 r=0
18 | 2 r=0
9 | 2 r=1
4 | 2 r=0
2 | 2 r=0
1 | 2 r=1

73 = 01001001

Quindi ho:

10010010.01001001.00000000.00000000

Allora una appropriata **subnet-mask** sarà:

11111111.11111111.11111111.11000000

Poiché la classe B prevede 16 bit per la rete, 16 per gli host

Le sottoreti disponibili saranno:

16 -
6 (bit per gli host) =
10 bit per le sottoreti

Quindi : $2^{10} = 1024$ (-1 perché è l'indirizzo base) = 1023 sottoreti in totale.

Nb. Se nella conversione da decimale a binario vedo che ottengo un numero binario di 7 cifre, posso aggiungere davanti uno 0 per arrivare a 8 cifre

76 | 2 r=0
38 | 2 r=0
19 | 2 r=1
9 | 2 r=1
4 | 2 r=0
2 | 2 r=0
1 | 2 r=1

76 = 11001100

3) Una stazione connessa ad una rete Ethernet subisce 6 collisioni consecutive nel tentativo di spedire un frame. In quale range temporale avverrà la successiva ritrasmissione?

- a) [0, 3225.6]
- b) [0, 63]
- c) [0, 64]
- d) [0, 1638.4]

SOL

Uno slot di BEB è 51.2 us (microsec.)

BEB: $I=[0; (2^N) - 1]$

Quindi:

BEB: $I=[0; (2^6) - 1]$

$I=[0; 63]$

$63 \cdot 51.2 = 3225.6$

BEB: $I=[0; 3225.6]$

4) Quale problema potrebbe verificarsi se il protocollo DHCP includesse lo scambio solo dei primi due messaggi previsti?

- a) potrebbero essere assegnati più indirizzi IP diversi allo stesso host

- b) potrebbe essere assegnato uno stesso indirizzo IP a host diversi
- c) potrebbe essere ricordato come libero un indirizzo IP assegnato ad un host
- d) un host potrebbe non essere certo di poter usare l'indirizzo IP ricevuto

5) A cosa servono i pacchetti di Link State Request in OSPF?

- a) a richiedere ai router adiacenti informazioni aggiornate di link state
- b) a richiedere periodicamente a tutti i nodi della rete informazioni aggiornate di link state
- c) ad inviare in flooding il proprio link state
- d) ad informare i vicini del proprio link state quando viene adottata la politica di source routing

6) Una connessione TCP ha correntemente RTT=16 msec., RTO=24 msec. e D=2. Si determini il valore di RTT e RTO se le due successive trasmissioni avvengono con successo e gli ack sono ricevuti rispettivamente (1) dopo 23 msec. e (2) dopo 28 msec. dalle rispettive trasmissioni. Si applichi l'abituale valore di a.

- a) RTT = 17.83 ; RTO = 30.65
- b) RTT = 16.27 ; RTO = 24.83
- c) RTT = 20.34 ; RTO = 29.41
- d) RTT = 21.33 ; RTO = 26.87

SOL

Il segmento1 (S1) arriva dopo 23 ms
Il segmento2 (S2) arriva dopo 28 ms

$$D_{new} = \beta \cdot D + (1 - \beta) |RTT_{old} - M|$$

$$RTT_{new} = \alpha \cdot RTT_{old} + (1 - \alpha)M$$

$$RTO = RTT_{new} + 4 D_{new}$$

S1)

$$D = 0.9 \cdot 2 + (0.1) |16 - 23| = 1.8 + 0.7 = 2.5$$

$$RTT = 0.9 \cdot 16 + 0.1 \cdot 23 = 14.4 + 2.3 = 16.7$$

$$RTO = 16.7 + 4 \cdot 2.5 = 26.7$$

S2)

$$D = 0.9 \cdot 2.5 + 0.1 \cdot |16.7 - 28| = 2.25 + 1.13 = 3.38$$

$$RTT = 0.9 \cdot 16.7 + 0.1 \cdot 28 = 15.03 + 2.8 = 17.83$$

$$RTO = 17.83 + 4 \cdot 3.38 = 31.35 \sim 30.65 \text{ ms (probabile arrotondamenti nel calcolo)}$$

7) Una connessione TCP si trova ad operare nelle seguenti condizioni: finestra di ricezione 64KB, finestra di congestione 26 KB, soglia SST 16 KB. Se a questo punto il sender riceve 3 ACK duplicati, quale sarà la nuova finestra di congestione?

- a) 16 KB
- b) 13 KB
- c) 8 KB
- d) 27 KB

SOL

Dai dati abbiamo:

$$Wr = 64 \text{ KB}$$

$$Wc = 26 \text{ KB}$$

$$SST = 16 \text{ KB}$$

Dopo tre ACK duplicati siamo nel caso di congestione lieve quindi inizia la fase di Fast Recovery:

$$SST = Wc/2 = 26/2 = 13$$

$$Wc = 13$$

Si riprende con la fase di congestion avoidance, quindi la Wc viene incrementata linearmente:

8) L'algoritmo proposto da Clark opera:

- a) su connessione TCP lato receiver
- b) in TCP per dimensionare il timer di keepalive
- c) su connessione TCP lato sender
- d) in TCP e UDP per la stima del RTO

- 9) In un router interno ad una rete MPLS, quale delle seguenti funzioni NON viene svolta?
- routing in base all'indirizzo destinazione del pacchetto
 - routing in base alla label del pacchetto
 - aggiornamento delle tabelle di instradamento
 - accodamento del pacchetto nelle code di output in base ai requisiti di scheduling del pacchetto

10) Una connessione TCP ha RTT = 20 msec. e RTO = 26 msec. quando si verifica un timeout sulla trasmissione del segmento con # sequenza = 10. Quali sono i valori di RTT, D e di RTO usati per la ritrasmissione con successo del segmento 10?

- RTT = 20, D = 1.5, RTO = 52
- RTT = 10, D = 4, RTO = 52
- RTT = 40, D = 2, RTO = 26
- RTT = 36, D = 4, RTO = 40

SOL

RTT = 20 ms
 RTO = 26 ms
 #seq = 10

$$\begin{aligned} \text{RTO} &= \text{RTT} + 4D \\ D &= (\text{RTO} - \text{RTT})/4 = (26-20)/4 = 1.5 \text{ ms} \end{aligned}$$

Per trovare il valore di RTO si fa riferimento alla politica di Karn secondo il quale, quando si verifica un retransmission timeout, RTT rimane invariato e raddoppio l'RTO:

$$\begin{aligned} \text{RTO} &= \text{RTO}_{\text{old}} \cdot 2 = 26 \cdot 2 = 52 \text{ ms} \\ \text{RTT} &= 20 \text{ ms} \end{aligned}$$

Manca calcolo del RTO

Esame 22-07-2007

1) Da cosa è composto un canale T1?

- Da frame di 24 canali ognuno dei quali trasporta un segnale campionato 8000 volte al secondo
- Da frame di 890 canali ognuno dei quali trasporta un segnale campionato 125 volte al secondo
- Da frame di 125 canali ognuno dei quali trasporta un segnale campionato 4000 volte al secondo
- Da frame di 193 canali ognuno dei quali trasporta un segnale campionato 8000 volte al secondo

2) Si consideri un protocollo di livello Data Link che usa un approccio Selective Repeat per inviare frame su un canale in rame della lunghezza di 8 Km, avente banda di 16 Mbps. Quale dimensione di frame produce un maggiore utilizzo di rete se la finestra è di 5 frame?

SOL

Dai dati ho:

K=5
 d=8 km
 b=16 Mbps
 f?

$$\begin{aligned} \text{Tx} &= f/b \rightarrow f = \text{Tx} \times b \\ U &= k \times (\text{Tx}/(\text{Tx}+2\text{Tp})) \\ \text{Quindi:} \\ \text{Tp} &= d/v = (8 \cdot 10^3 \text{ m})/(2 \cdot 10^8 \text{ m/s}) = 4 \cdot 10^{-5} \text{ s} = 40 \text{ us} \\ \text{Bwth} &= 16 \text{ Mbps} = 16 \cdot 10^6 \text{ bps} \\ K &= (2^n)/2 = (2^5)/2 = 32/2 = 16 \end{aligned}$$

$$U = K \cdot (\text{Tx}/(\text{Tx} + 2\text{Tp}))$$

Si pone U=1 e risolvo l'equazione:

U = 1 se

$$k \frac{f}{\left(\frac{f}{b} + 2Tp\right)} \geq 1 \rightarrow k \frac{f}{(f + 2bTp)} \geq 1$$

$$\frac{1}{(f + 2bTp)} \geq \frac{1}{fk} \rightarrow \frac{f + 2bTp}{fk} \geq 1$$

$$\frac{f + 2bTp}{fk} \geq 1$$

$$2bTp \geq f * k - f$$

$$2bTp \geq f(k - 1)$$

$$\frac{2bTp}{k-1} \geq f$$

$$f \geq (2 \cdot b \cdot Tp) / (k-1) = (2 \cdot 16 \cdot 10^6 \text{ bps} \cdot 4 \cdot 10^{-5}) / 4 \text{ s} = 320b$$

???? riproviamo..

$$k \frac{f}{\left(\frac{f}{b} + 2Tp\right)} \geq 1$$

$$\frac{f}{b} = \frac{1}{k} \left(\frac{f}{b} + 2Tp \right)$$

$$\frac{f}{b} = \left(\frac{f}{bk} + \frac{2Tp}{k} \right)$$

$$\frac{f}{b} = b \left(\frac{f}{bk} + \frac{2Tp}{k} \right) = \left(\frac{f + 2bTp}{k} \right) = \frac{f}{k} + \frac{2bTp}{k}$$

$$\frac{f}{b} - \frac{f}{k} = \frac{2bTp}{k}$$

$$fk - f = 2bTp \rightarrow f(k-1) = 2bTp \rightarrow$$

$$\frac{2bTp}{k-1} = f$$

???? riproviamo se al posto di avere U=1 ci accontentassimo di un utilizzo del 50%, cioè U=0.5..

$$k \frac{\frac{f}{b}}{\left(\frac{f}{b} + 2Tp\right)} \geq 0.5$$

Moltiplico per b il primo membro

$$k \frac{f}{(f + 2bTp)} = 0.5$$

Porto a destra f e k

$$\frac{1}{(f + 2bTp)} = \frac{0.5}{fk}$$

Porto a numeratore il denominatore del primo membro

$$\frac{0.5(f + 2bTp)}{fk} = 1$$

$$\begin{aligned} bTp &= fk - 0.5f \\ bTp &= f(k - 0.5) \\ \frac{bTp}{k - 0.5} &= f \end{aligned}$$

3) A cosa serve il persist timer?

- A) evitare la chiusura di un solo lato di una connessione TCP
- B) evitare condizioni di deadlock su una connessione TCP generate da errori nei trasferimenti di windows
- C) controllare la ritrasmissione nel caso di perdita di 3 segmenti consecutivi
- D) controllare la ritrasmissione di frame in un protocollo CSMA/CD 1-persistent

4) Si consideri una connessione TCP avente SST=20 KB. Se vengono ricevuti tre ACK duplicati nel momento in cui la congestion window ha dimensione 34 KB, quale sarà la dimensione della congestion window se le successive 3 trasmissioni avvengono con successo e se la minimum segment size è 1KB?

SOL

I dati che abbiamo sono:

SST=20 KB

Wc=34 KB

La congestione dopo tre ack duplicati è lieve, quindi:

$$SST_{new} = \frac{W_c}{2} = \frac{34}{2} = 17$$

$$W_c = SST_{new} = 17$$

Si procede con la fase di congestione avoidance:

18,19,20

il valore di Wc dopo tre trasmissioni è 20

5) Se il RTT di TCP è correntemente 30 msec, RTO è 38 msec, D=2 e i successivi ack arrivano dopo 26msec, 32msec e 24 msec dalle rispettive trasmissioni, quali sono le stime finali di RTT e RTO usando ALFA=0.9?

SOL

RTT=30 ms

RTO=38 ms

D=2

Il segmento 1 (S1) arriva dopo 26 ms

Il segmento 2 (S2) arriva dopo 32 ms

Il segmento 3 (S3) arriva dopo 24 ms

Si usa la formula per le stime

$$D_{new} = \beta \cdot D + (1 - \beta)|RTT_{old} - M|$$

$$RTT_{new} = \alpha \cdot RTT_{old} + (1 - \alpha)M$$

$$RTO = RTT_{new} + 4 D_{new}$$

S1)

$$RTT = 0.9 \cdot 30 + 0.1 \cdot 26 = 27 + 2.6 = 29.6 \text{ ms}$$

$$D = 0.9 \cdot 2 + 0.1 \cdot |30 - 26| = 1.8 + 0.1 \cdot 4 = 2.2 \text{ ms}$$

$$RTO = 29.6 + 4 \cdot 2.2 = 38.4 \text{ ms}$$

S2)

$$RTT = 0.9 \cdot 29.6 + 0.1 \cdot 32 = 29.84 \text{ ms}$$

$$D = 0.9 \cdot 2.2 + 0.1 \cdot |29.6 - 32| = 1.98 + 0.24 = 2.22 \text{ ms}$$

$$RTO = 29.84 + 4 \cdot 2.22 = 38.72$$

S3)

$$RTT = 0.9 \cdot 29.84 + 0.1 \cdot 24 = 29.256 \text{ ms} \sim \underline{29.26 \text{ ms}}$$

$$D = 0.9 \cdot 2.22 + 0.1 \cdot 129.84 - 241 = 1.998 + 0.584 = 2.582 \text{ ms}$$

$$RTO = 29.256 + 4 \cdot 2.582 = \underline{39.584 \text{ ms}}$$

6) Un router R adotta la tecnica link state per determinare il cammino da S a D. Può R conoscere l'intera sequenza di router da attraversare sul cammino da S a D?

- A) no, è possibile conoscere solo il cammino minimo per raggiungere i router adiacenti a R
- B) si, e può garantirsi che venga rispettato quel cammino specificando il campo source routing nelle opzioni dell'header di IP
- C) si, infatti questa è la tecnica usata da BGP
- D) no, non è possibile conoscere l'intero cammino con la tecnica link state

7) Come fa BGP a superare il problema di count-to-infinity?

- A) scambia i path completi da sorgente a destinazione
- B) utilizza un approccio link state
- C) Indica nei pacchetti il path completo che devono seguire
- D) Adotta la tecnica dello split horizon

8) L'algoritmo Binary Exponential Backoff:

- A) Consente di risolvere le collisioni in un numero esponenziale di tentativi
- B) adatta il tempo di ritrasmissione di una frame allo scadere del timeout di trasmissione
- C) garantisce di risolvere le collisioni in un tempo indipendente al numero di collisioni subite da una stazione
- D) si adatta al numero di stazioni che tentano l'accesso al canale risolvendo rapidamente le collisioni con poche stazioni e garantendo tempi finiti in caso contrario

9) Il campo fragment offset dell'header IP contiene:

- A) la posizione del primo byte utente nel frammento, in multipli di 8 byte, riferito allo stream di byte fornito dal livello 4
- B) la posizione del primo byte utente nel frammento, riferito allo stream di byte fornito al livello 4
- C) la dimensione minima negoziata per il frammento IP
- D) il numero di byte residui da trasferire in frammenti successivi per completare lo stream di byte fornito dal livello 4

10) Si consideri una rete magliata in cui la distanza minima in hop fra il nodo A ed il nodo U è di 10 hop. Il tempo medio per trasferire un pacchetto su un solo hop è di 1 msec, ogni nodo scambia i propri distance vector ogni 30 sec e tale scambio si ipotizza sincrono su tutti i nodi. Dopo quanto tempo il nodo Y riceve la distance vector di A, supponendo che al tempo T0 i nodi iniziano ad eseguire il protocollo?

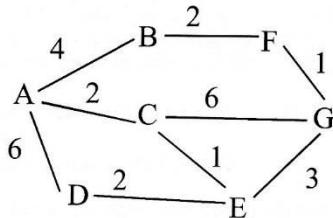
- A) T0+30 sec
- B) T0+30sec e 10msec
- C) T0+10msec
- D) T0+5min e 1 msec

Esame di Reti di Calcolatori (12 cfu)

1 Ottobre 2002

(L'esame vale anche come Architetture II vecchio ordinamento, 12 cfu)

- 1) Che valore hanno i ritardi di trasmissione e di propagazione di un pacchetto di 8 Kb inviato su un canale avente banda 2 Mbps e lunghezza 3 Km?
- 2) Descrivere la struttura di un frame nel caso si utilizzi un canale T1.
- 3) Quali informazioni contengono i contatori RC e CD usati nelle reti DQDB?
- 4) Per la rete mostrata, si computi lo shortest path da A ad ogni altro nodo della rete evidenziando i passaggi di etichettatura secondo l'algoritmo di Dijkstra.



- 5) Illustrare una situazione in cui la politica di rilascio simmetrico di una connessione di livello trasporto fallisce.
- 6) Quali sono le funzioni svolte da un protocollo a finestra e a quale livello della gerarchia di protocolli opera ?
- 7) Quali sono le variabili considerate da TCP per svolgere il controllo di flusso ?
- 8) Su un host A sono installati sia TCP che UDP a livello trasporto. All'atto della ricezione di un pacchetto, il protocollo IP deve poter capire se contiene un payload TCP o UDP e consegnarlo al destinatario corretto. Come puo' svolgere questa funzione ?
- 9) Il livello Applicazione e' definito come 7° livello della architettura ISO/OSI. Questo livello e' presente anche in uno stack TCP/IP ?
- 10) In quali condizioni di traffico e' preferibile una politica non-persistent CSMA ad una persistente ?

Non rispondere alle seguenti domande per l'esame di 1 UD (6 CFU)

- 11) Quali informazioni contiene un record NS nel database di un Domain Name System? Quali record NS sono normalmente presenti in ogni database?
- 12) Descrivi come un ftp server comunica con il proprio client attraverso TCP.

