

1 Livello applicazione

due tipi di architetture

- client server
- peer to peer (condivisione di file)

1.1 Comunicazione Dei Processi

un processo è un'istanza di esecuzione di un programma.

La comunicazione dei processi **all'interno della singola** macchina è governata dal **sistema operativo** (interprocess communication).

In rete, i processi comunicano tramite uno scambio di messaggi. In questo contesto li etichettiamo nel seguente modo:

- client (avvia la comunicazione)
- server (attende di essere contattato per avviare la comunicazione)

1.1.1 socket

Si suppone la presenza di un'infrastruttura esterna che permetta la comunicazione tra i processi in rete, e si fa uso dei **socket**, interfaccia software che si frappone tra un processo applicativo e il protocollo di trasporto.

Il programmatore potrà scegliere il protocollo di trasporto ed in alcuni casi anche alcuni parametri (dimensione buffer / segmento)

Per identificare un processo in rete si usa:

- **l'indirizzo IP**: della macchina che lo ospita
- **il socket**: attraverso l'uso di un numero porta. (web = 80 | posta = 25 | ecc...)

1.2 Protocolli di trasporto

Sono disponibili diversi protocolli, che possono essere classificati secondo i seguenti parametri:

- **trasferimento dati affidabile**

trasporto senza perdita di pacchetti, utile ad esempio: posta elettronica, finanza.

nel caso contrario si parla di **loss-tolerant application**, ad esempio multimediali

- **throughput**

quantità effettiva di dati trasmessi in un dato periodo di tempo.

variabile nel tempo a causa della condivisione della banda. il protocollo può fornire uno stabile. Ad esempio: app telefonia internet, necessita di 32 kbps

bandwidth-sensitive application: multimediali

elastic application: posta elettronica

- **temporizzazione**

garanzia sul "timing", vincoli sul ritardo di ricezione.

utilie per applicazioni in real-time.

- **sicurezza**

servizi di cifratura.

1.2.1 protocollo TCP

1.2.1.1 caratteristiche

- **orientato alla connessione**

prevede uno scambio informazioni di controllo a livello di trasporto prima della comunicazione a liv. applicazione (handshaking)

- **trasporto affidabile dei dati**

- **controllo congestione**

1.2.1.2 fasi della comunicazione

- handshake
- connessione tra i socket dei processi in modalità full-duplex
- scambio messaggio

- chiusura connessione

1.2.1.3 sicurezza

di default non viene fornito un layer di sicurezza. per questo è stato sviluppato il **Transport layer security (TLS)**, un modulo software che permette: la cifratura, l'autenticazione end-to-end, ecc...

1.2.2 protocollo UDP

1.2.2.1 caratteristiche

- leggero
- non orientato connessione
- trasferimento dati non affidabile
- non controlla congestione

1.3 Protocollli livello applicazione

definiscono la comunicazione tra i processi di un'applicazione presenti in sistemi periferici diversi.

in particolare:

- tipo messaggio: richiesta o risposta
- sintassi e semantica dei campi

ex: HTTP descritto dalla RFC 7230

di seguito i principali protocoli a livello di applicazione, in ordine cronologico.

1.3.1 telnet

servizio che permette di collegarsi ad un terminale remoto.

client: prende input tastiera, impacchetta, invia al server.

server: riceve pacchetto da client e li passa alla tastiera virtuale.

feedback visivo

1. pressione tasto
2. invio al server di un pacchetto contenente la pressione avvenuta
3. scrittura sul terminale telnet
4. invio pacchetto al client per feedback
5. output client

nel caso di problematiche non viene mostrato nel client un input non inserito nel server

1.3.1.1 problemi

1. spreco risorse: si impiegano sempre 20byte per la comunicazione
2. login: la psw viene inviata in chiaro

1.3.2 file transfer protocol (FTP)

mette a disposizione un servizio client-server, per accedere a directory o cartelle di un computer remoto.

permette l'uso di sistemi eterogenei (diversi: sistemi operativi, strutture, set caratteri, ecc...)

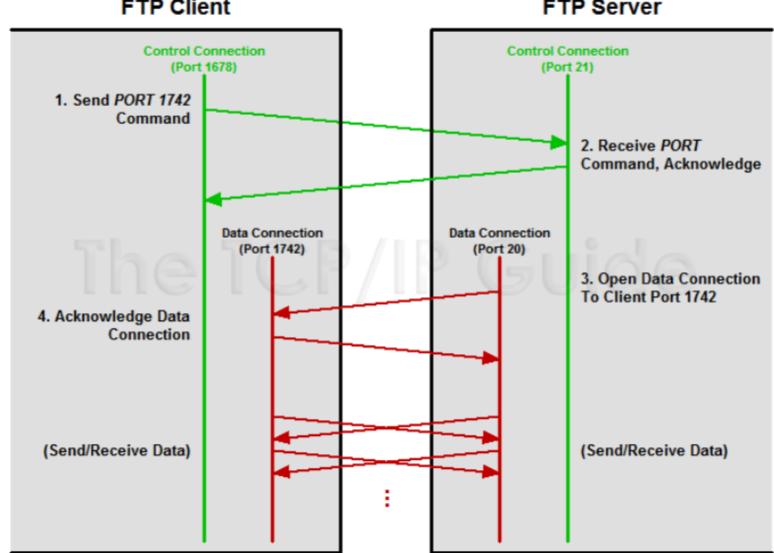
si possono stabilire due tipi di connessioni:

- **attiva**

il client stabilisce una connessione per lo scambio dei comandi ed il server ne stabilisce una per lo scambio di dati.
non è la connessione default in quanto potrebbe causare problemi in presenza di firewall.

- **passiva**

il client stabilisce due connessioni (comandi, dati).



1.3.3 hyper text transfer protocol (http)

protocollo che definisce il modo in cui i client richiedono le pagine ai web server e come quest'ultimi le trasferiscono ai client.

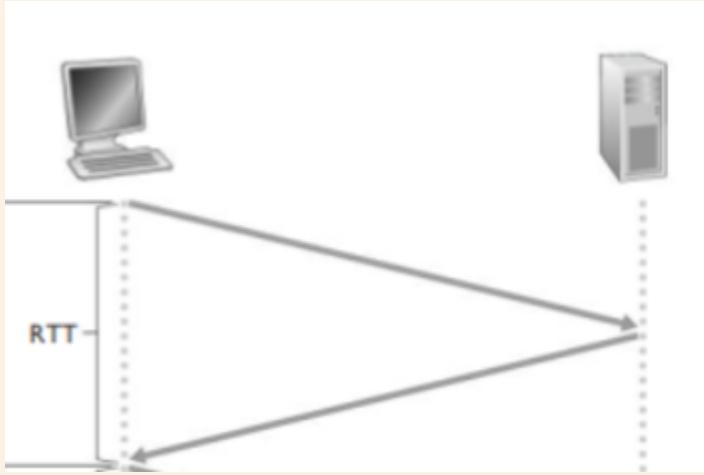
oggetto: file presente nella pagina (img, file di vari tipi, ecc...)

caratteristiche:

- **TCP**
quindi ci sarà un preventivo handshake tra client e server, dopo il quale potranno comunicare.
- **Stateless**
è un protocollo "senza stato", ovvero non mantiene tracce di richieste precedenti.

⚠ Round Trip Time (RTT)

tempo impiegato da un piccolo pacchetto per fare il viaggio: client - server - client



di seguito le varie versioni.

1.3.3.1 Http 1

connessione non persistente:

una connessione per ogni oggetto scaricato.

fasi

- richiesta TCP del client, tramite la propria socket, verso la porta 80 (default HTTP) del server
- server riceve, incapsula info richieste ed invia al client.
- server comunica al client di interrompere la comunicazione. verrà interrotta quando il client avrà ricevuto il messaggio integro.
- client riceve, la connessione termina
- verrà stabilità una connessione per ogni oggetto richiesto

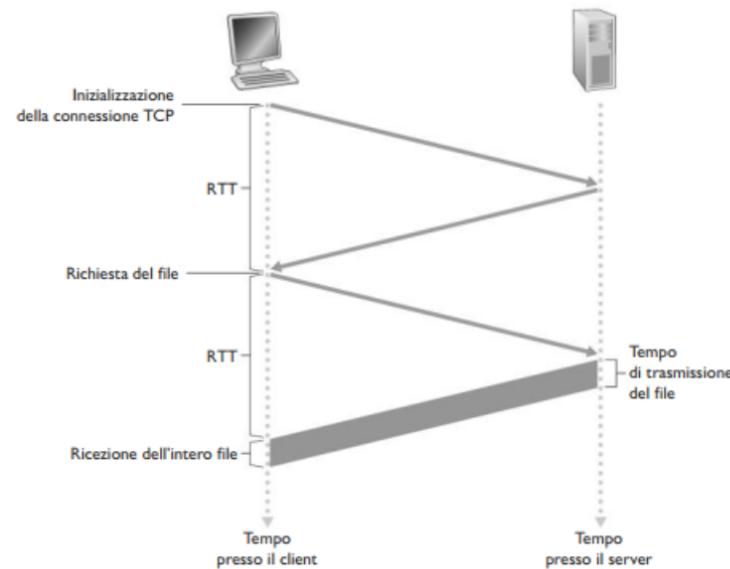
come si nota, coppie di richieste devono essere inviate in connessioni separate. si crea il cosiddetto:

three-way handshake

formato in 3 fasi:

1. client invia segmento TCP al server

2. server conferma con uno ulteriore
3. il client conferma nuovamente tramite una richiesta HTTP



avremo quindi un tempo di trasmissione che si approssima a 2 RTT.

1.3.3.2 Http 1.1

problema: Head of line

si è notato che con l'invio di tutti gli oggetti su una singola connessione (HTTP 1.1), sorgeva il problema di **Head of line (HOL)**. Nel momento in cui nella pagina HTML è presente un oggetto di grandi dimensioni (video) nella parte alta, seguito da oggetti di piccole dimensioni, si creerà un ritardo degli oggetti sottostanti il video.

persistente.

Vengono aperte più connessioni parallele, su cui si effettuano richieste sequenziali
il numero di connessioni attive è vincolato dai browser.

la connessione verrà chiusa dopo un determinata inattività temporale.

1.3.3.3 Http 2

permette la richiesta e la risposta sulla stessa connessione (multiplexing) TCP, riducendo la latenza.

Il server, tramite una singola richiesta, effettuerà un invio/push al client di oggetti aggiuntivi. (sapendo la struttura della pagina). Questo è possibile in quanto le pagina di base HTML identifica di suo gli oggetti necessari per il render completo.

In HTTP 2 si cerca di minimizzare le connessioni (in 1.1 i browser massimizzano). si suddividono i messaggio in frame e si alternando messaggi di richiesta e risposta sulla stessa connessione. questa operazione (demultiplexing) è a carico di un sottolivello di framing di HTTP che esegue inoltre una codifica binaria delle info (più efficiente da operare e meno soggetta ad errori).

1.3.3.4 Http 3

lavora con il protocollo di trasporto QUIC.

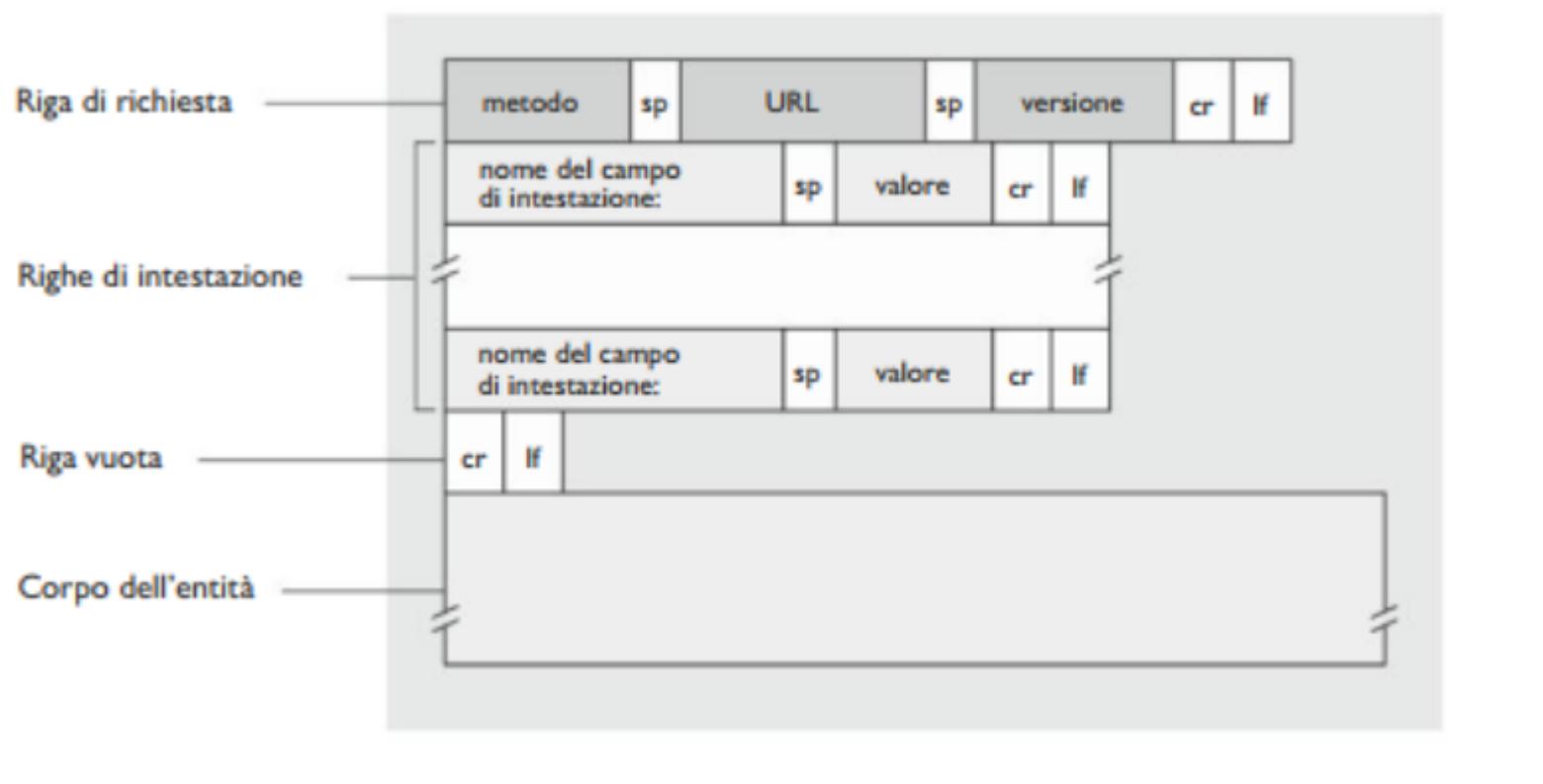
1.3.3.5 formato richiesta

request line: prima riga della richiesta

header lines: successive

```
GET somedir/file.html HTTP/[version]
Host: www.prova.it
Connection: close // connessione non persistenti
User-agent: [browser]/[version]
Accept-language: [lang]
```

 generalizzazione



- cr: carriage return, ritorno a capo
- lf: line feed, nuova linea
- sp: due punti

il corpo viene riempito nel caso di richieste "POST" con i dati inseriti

1.3.3.6 formato risposta

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

- 200 OK: la richiesta ha avuto successo e in risposta si invia l'informazione.
- 301 Moved Permanently: l'oggetto richiesto è stato trasferito in modo permanente; il nuovo URL è specificato nell'intestazione Location: del messaggio di risposta. Il client recupererà automaticamente il nuovo URL.
- 400 Bad Request: si tratta di un codice di errore generico che indica che la richiesta non è stata compresa dal server.
- 404 Not Found: il documento richiesto non esiste sul server.
- 505 HTTP Version Not Supported: il server non dispone della versione di protocollo HTTP richiesta.

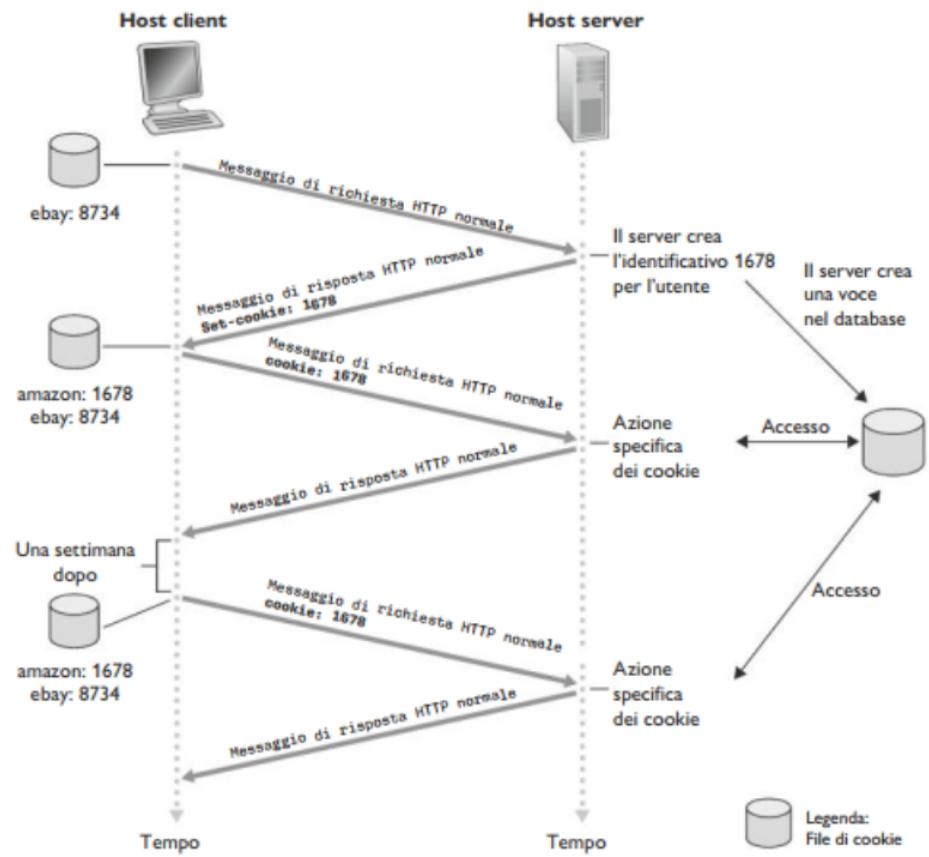
1.3.3.7 cookie

nonostante http sia stateless (non mantenga uno stato della connessione) in certi casi è necessario autenticare l'utente per questo vengono usati i **cookie**.

composti da 4 componenti:

1. riga di intestazione risposta HTTP
2. riga di intestazione nella richiesta HTTP
3. file mantenuto nel browser dell'utente

4. un database sul server che tiene traccia delle info collegate ai cookie



1.3.3.8 web caching - proxy server

entità di rete che permette di soddisfare richieste HTTP di un altro server. salva all'interno della sua memoria copie di oggetti richiesti recentemente.

vantaggi:

- riduzione tempi risposta
 - minore larghezza di banda necessaria
- il collo di bottiglia che spesso si crea tra client e server è molto inferiore a quello tra client e proxy.

fasi:

- browser stabilisce connessione TCP con il proxy server ed invia richiesta HTTP
- se proxy non ha l'oggetto, fa una richiesta al server effettivo
- nel momento in cui può soddisfare la richiesta risponderà al client, mantenendo al suo interno una copia

si crea il problema che una copia presente nel proxy potrebbe essere scaduta, viene risolto con il meccanismo di **GET condizionale**

1.3.3.9 GET condizionale

in quanto all'interno del proxy viene memorizzata la data di ultima modifica.

nel momento in cui potrebbe essere modificato il file si invia al server originale la seguente richiesta get.

si aggiunge l'header `if-modified-since`.

```
GET ...
Host: ...
If-modified-since: [timestamp] //sarà uguale alla data salvata di ultima modifica
```

se non è stato modificato:

```
HTTP/1.1 304 Not modified
Date: ...
Server:
[empty]
```

1.3.4 Posta elettronica

3 componenti principali:

- user agent (applicazione client che permette l'invio di messaggi)
- mail server
- protocollo SMTP

1.3.4.1 Simple mail transfer protocol (SMTP)

protocollo orientato alla connessione che permette la comunicazione tra mail server ed user agent, tramite lo scambio di stringhe in formato ASCII. instaura connessioni TCP per garantire un trasferimento dati affidabile; inoltre si fa uso di connessioni persistenti.

il formato del corpo dei messaggi è 7bit, una forte restrizione per i file moderni.

Per garantire l'invio di caratteri al di fuori di questo range (Ex: multimediali) viene codificato in **UUENCODE**.

- l'oggetto viene suddiviso in gruppi di 3bytes (24bit). Se non si raggiungono 3byte si aggiunge 0 alla fine
- si suddividono ulteriormente in gruppi di 6bit (si creano gruppi di bit nel range: 0-63)
- si somma il valore 32 in modo tale da creare un range: 32 - 95

fasi della comunicazione:

- user agent mittente comunica con il proprio mail server (tramite protocollo TCP) i messaggi da inviare (adesso il mail server farà da client SMTP)
- il client SMTP stabilisce connessione TCP con server SMTP destinatario, sulla porta 25 (default)
- si identificano (handshake) indicando email
- scambio messaggi ASCII sulla stessa connessione
- chiusura connessione

il formato dei messaggi è definito in RFC 5322.

Prevede un intestazione identificativa "MAIL FROM, RCP TO, SUBJECT, ecc...".

il client termina la comunicazione con un "QUIT"

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr ... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Ti piace il ketchup?
C: Che cosa ne pensi dei cetrioli?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

1.3.4.2 POP3 e IMAP

in quanto SMTP è un protocollo push; per effettuare la lettura delle mail (pull) sarà necessaria un protocollo di accesso alla posta.

POP3

E' un protocollo di accesso alla posta semplice. Essendo un protocollo semplice le sue funzionalità sono limitate. **POP3 entra in azione quando lo user agent apre una connessione TCP verso il mail server sulla porta 110.** Quando la connessione TCP è stabilita allora il POP3 procede in tre fasi:

- **Autorizzazione:** lo user agent invia nome utente e password per autenticare l'utente.
- **Transazione:** lo user agent recupera i messaggi, può anche marcare i messaggi per la cancellazione, rimuovere i marcatori e ottenere statistiche sulla posta.
- **Aggiornamento:** ha luogo dopo che il client ha inviato il comando quit che conclude la sessione POP3. In questo istante il server mail rimuove tutti i messaggi marcati per la cancellazione.

In una transazione POP3 lo user agent invia comandi e il server reagisce a ogni comandi con una tra due possibili risposte **+OK e -ERR.**

La fase di autorizzazione ha due principali comandi: user <username> e pass <password>.

Altri comandi sono:

- **list:** user agent chiede al server di elencare la dimensione dei messaggi memorizzati
- **retr:** recupera il messaggio
- **del:** cancella il messaggio

IMAP

Con l'accesso tramite POP3 dopo aver scaricato i messaggi sulla macchina, con questi messaggi scaricati possiamo lavorarci. Come per esempio poter creare cartelle dove inserire i messaggi. Con il POP3 questa cosa non è possibile perché non fornisce all'utente alcuna procedura per creare cartelle remote e assegnare loro i messaggi.

Per risolvere questi problemi è stato realizzato il protocollo IMAP. **Anche IMAP è un protocollo di accesso alla posta ma presenta maggiori potenzialità rispetto al POP3 ed è più complesso.**

Un server IMAP associa ogni messaggio arrivato al server a una cartella. I messaggi in arrivo sono associati alla cartella **INBOX** del destinatario. L'utente può poi spostare il messaggio in altre cartelle, leggerlo, cancellarlo. Il protocollo IMAP fornisce comandi per consentire agli utenti di creare cartelle e spostare i messaggi da una cartella all'altra.

Fornisce comandi anche per fare ricerche nelle cartelle sulla base di criteri specifici.

I server IMAP conservano informazioni di stato sull'utente da una sessione all'altra. Ci sono anche comandi che permettono agli user agent di ottenere singole parti dei messaggi, caratteristica utile quando si dispone di una connessione con limitata ampiezza di banda tra lo user agent e il mail server.

1.3.5 DHTML

Dynamic html.

L'utente tramite il suo browser avvia una ricerca ma è una richiesta dinamica, cioè il server non troverà subito la pagina pronta, ma interagendo con degli script e/o database prepara la pagina html e risponde al browser. Ormai quasi tutti i server web hanno pagine dinamiche e non più pagine statiche. La pagina web viene costruita online quando arriva la richiesta.

1.3.6 DNS

protocollo usato per tradurre gli "hostname" in indirizzi IP, semplificando l'identificazione di una particolare macchina in rete.

1.3.6.1 servizi offerti

- **host aliasing**

assegnazione di sinonimi a particolari host. L'hostname originale verrà detto "hostname canonico". Il DNS server oltre a fornire IP quindi potrà fornire anche l'hostname canonico.

- **Mail server aliasing**

servizio di alias anche per mail server.

- **load distribution**

Servizio che permette di distribuire il carico tra i server replicati. Quest'ultimi vengono usati spesso nei servizi web che subiscono un traffico intenso; in pratica si va a replicare il contenuto di una pagina su diverse macchine (IP diversi).

Quando un client effettua una richiesta DNS, il server risponde con uno dei vari indirizzi dei server, cambiandolo in base al traffico corrente.

1.3.6.2 implementazione

le fasi di una comunicazione DNS sono le seguenti:

- client DNS interroga server DNS tramite l'invio di datagrammi UDP alla porta 53
- DNS in seguito ad un ritardo che si può aggirare tra i "ms" ed i "s" risponderà fornendo le info necessarie.

l'implementazione può essere:

- centralizzata
- distribuita a livelli

1.3.6.2.1 DNS centralizzato

Prevede un unico DNS server contenente tutte le corrispondenze.

Questo approccio fa sorgere le seguenti problematiche (comuni a tutti gli schemi centralizzati):

- **Un solo point of failure**: guasti
- **Volume di traffico**: gestione di richieste
- **Posizione fisica distante rispetto al client**

Questo approccio non è di certo in grado di adattarsi ad una crescita esponenziale degli host.

1.3.6.2.2 DNS distribuito

risolve il problema della scalabilità.

Si fa uso di DNS server distribuiti nel mondo, organizzati in maniera gerarchica.

la richiesta DNS viene effettuata partendo dall'alto della gerarchia, ognuno fornirà gli indirizzi dei server sottostanti.

Classificazione:

- **root server**

Per internet sono presenti 13 root server. Di questi viene effettuata la copia in circa 1000 server.

- **top level Domain (TLD)**

Gestiscono domini di primo livello (com, org, net, gov, ecc...) ed anche quelli relativi ai paesi (it, uk, fr, ecc...) .

- **server autoritativi**

gestiscono tutti i record DNS (associazion hostname - IP)

- **locale / default name server**

non appartiene strettamente a questa gerarchia.

usati in locale per: servizi di cache, inoltro query, sicurezza.

Le richieste effettuate possono essere di 2 tipi:

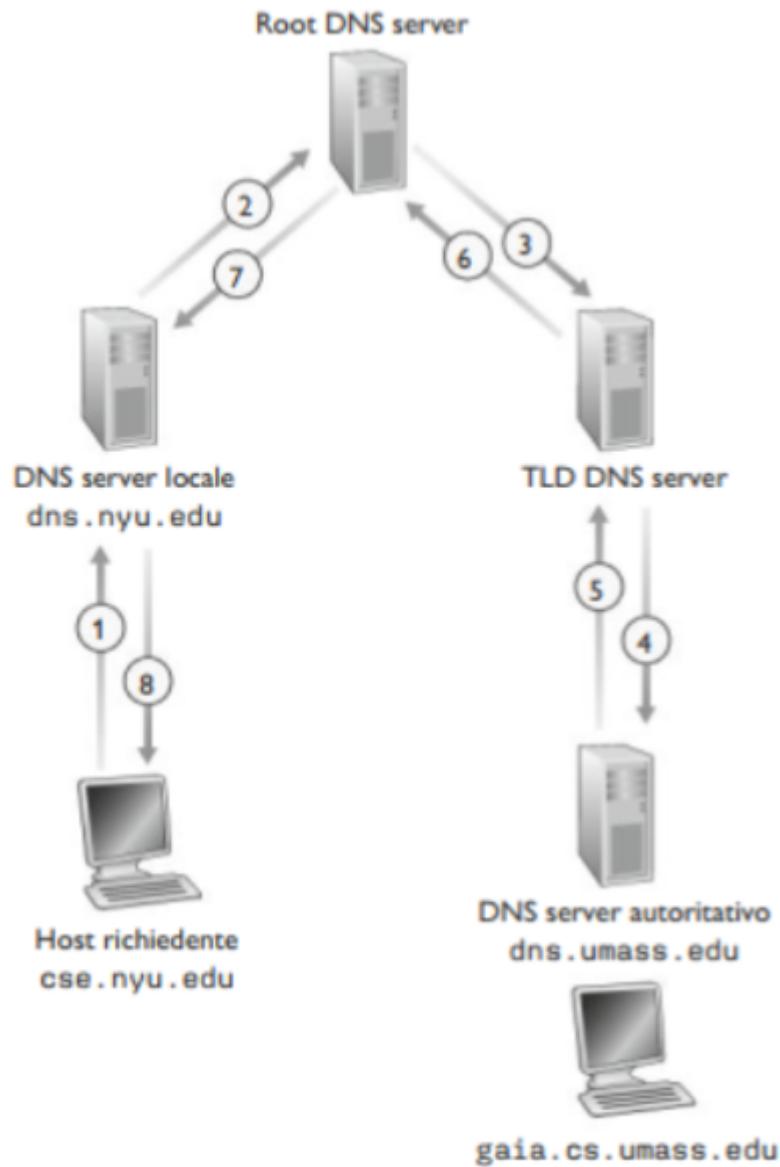
- **ricorsive**

effettuata dai client, richiedono una risposta finale alla domanda.

Il server che riceverà questo tipo di richiesta andrà a fare una serie di richieste **iterative** ad altri DNS server.

- **iterative**

verrà fornito al "client ricorsivo" un riferimento al server autoritativo oppure ad un altro server DNS che potrebbe avere maggiori info.



⚠ DNS caching

al fine di ottimizzare il traffico in ogni DNS server locale viene effettuata una copia di una recente risoluzione.

1.3.6.3 Record di risorsa

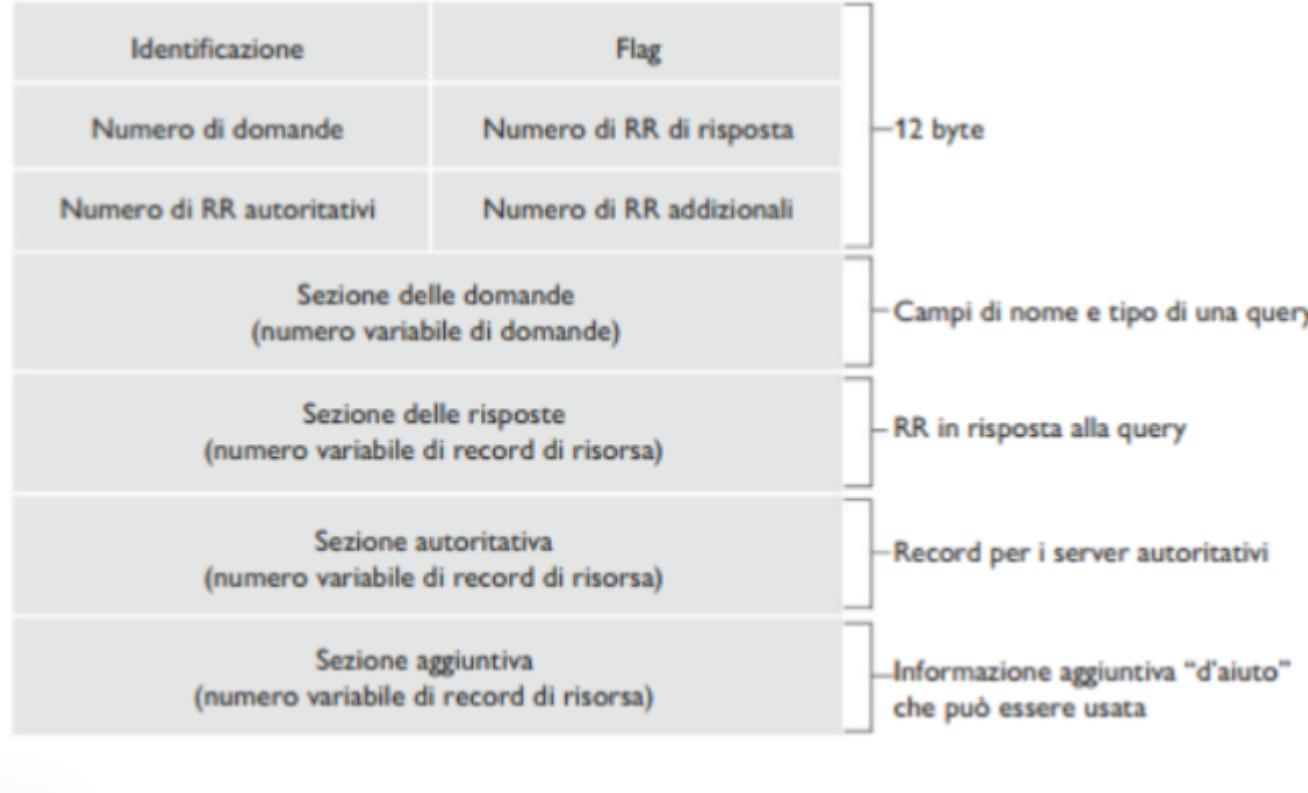
i server che implementano l'architettura distribuita memorizzano i cosiddetti **resource record** (RR)

Contiene i seguenti campi (Name, Value, Type, TTL) :

- **Name, Value:** dipendono da "type"
- **Type:**
 - **A:** corrispondenza hostname e IP. Anche se il server non è autoritativo rispetto a questo hostname, potrebbe contenere questo tipo di record nella proprio cache
 - Ex: (foo.com, [ip], A, [TTL])
 - **NS:** name è un dominio e value è hostname del server autoritativo associato
 - Ex: (foo.com, dns.foo.com, NS)
 - **CNAME:** fornisce il nome canonico
 - Ex: (foo.fooo.com, foo.com, CNAME, [TTL])
 - **MX:** value è il nome canonico del mail server (name)
 - Ex: (foo.com, mail.foo.com, MX, [TTL])
- **TTL:** tempo residuo del record all'interno della cache del server

1.3.6.4 Messaggi DNS

Di seguito è riportata la struttura della query DNS:



1.3.6.4.1 Header section

Contiene al suo interno una serie di campi:

- **identificazione**: id richiesta (16bit)
 - **flag**: formato a sua volta dai seguenti bit (i nomi attribuiti sono personali)
 - **tipo_richiesta**: se è una query (0) o reply (1)
 - **is_autoritativo**: viene impostato se il server in questione è autoritativo per l'hostname richiesto
 - **recursion_desired_flag**: se si desidera una query ricorsiva.
 - **recursion_available**: impostato solo nelle risposte. indica se il server supporta la ricorsione
- [tipo_richiesta][is_autoritativo][recursion_desired_flag][recursion_available]
- **Numero di ...**: campi che identificano le occorrenze dei seguenti campi

1.3.6.4.2 Sezione delle domande

Contiene info sulla richiesta:

- nome: hostname richiesto
- tipo: si usano i type di "[Record di risorsa](#)"

1.3.6.4.3 Sezione delle risposte

Contiene RR relativi all'hostname richiesto. Una risposta potrebbe contenere vari RR, a causa dei server replicati.

1.3.6.4.4 Sezione autoritativa

Contiene RR di altri server autoritativi (cache?).

contiene i record DNS che provengono da un server DNS autoritativo.

include informazioni che il server autoritativo conferma come accurate per il dominio in questione.

I tipi di record che si trovano comunemente nella sezione autoritativa includono:

- **Record A (Address):** Associa un nome di dominio a un indirizzo IPv4.
- **Record AAAA (IPv6 Address):** Associa un nome di dominio a un indirizzo IPv6.
- **Record MX (Mail Exchange):** Specifica i server di posta elettronica per il dominio.
- **Record NS (Name Server):** Elenca i server dei nomi autoritativi per il dominio.

1.3.6.4.5 Sezione aggiuntiva

Le informazioni supplementari che potrebbero essere utili per risolvere la query originale o per ulteriori richieste successive.

Questi record non sono strettamente necessari per rispondere alla query originale, ma sono inclusi per migliorare l'efficienza della risoluzione DNS.

La sezione aggiuntiva spesso contiene:

- **Record A o AAAA per i server dei nomi:** Se nella risposta autoritativa sono elencati dei server dei nomi (NS), la sezione aggiuntiva può includere i record A o AAAA che forniscono gli indirizzi IP di questi server dei nomi.
- **Record per servizi specifici:** Se nella query sono coinvolti record come MX o SRV (Service), la sezione aggiuntiva può contenere gli indirizzi IP di questi server per evitare la necessità di ulteriori ricerche.

1.4 Nomi dei dati

TCP/IP Model Layers

Name of PDU (Protocol Data Unit)

Application Layer

Message

Transport Layer

TCP Segment
UDP Datagram

Network Layer

IP Datagram

Datalink Layer

Frame

Physical Layer

Bits

1 Livello trasporto

Il livello di trasporto fornisce una connessione logica tra i processi comunicativi, ovvero crea un astrazione che simuli una connessione diretta.

Lato mittente, si occupa spezzettare (se necessario) i messaggi provenienti dal livello applicativo, aggiugendo un intestazione di trasporto, per poi infine passarlo al livello di rete; nel quale viene incapsulato in un pacchetto (datagramma ed inviato a destinazione).

in questo livello, i pacchetti prendono il nome di "segmenti" (transport-layer segment).

Lato destinatario effettua l'operazione opposta rispetto ai "datagrammi" provenienti dal livello di rete.

I principali protocolli usati sono **User Datagram Protocol** (UDP) e **Transmission Control Protocol** (TCP).

Occorre notare che il livello di rete non garantisce nessuna garanzia rispetto alla consegna effettiva e soprattutto sull'ordine di arrivo. Fa solo del suo meglio, si dice che sia un protocollo **best-effort**.

Un ulteriore compito del livello di trasporto sarà il passaggio dalla comunicazione "tra sistemi periferici" a quello tra "processi in esecuzione su sistemi periferici" (multiplexing e demultiplexing).

2 Multiplexing e Demultiplexing

Servizio offerto dal livello di trasporto che permette il passaggio dalla comunicazione "host-host" (fornita dal livello di rete) alla comunicazione tra i processi in esecuzione sugli host.

Ricorda che i dati non vengono trasferiti direttamente al processo ma sempre tramite la socket associata.

Quindi definiamo:

- multiplexing: processo che segmenta i dati e li incapsula con intestazioni di trasporto
- demultiplexing: processo che permette trasportare i segmenti verso la socket corretta

Ex: impacchettare una lettera (multiplexing) ed inviarla ad un cugino. lo zio spacchetta la lettera e la consegna al cugino (demultiplexing).

3 UDP

protocollo di trasporto minimale, non orientato alla connessione/connectionless (non effettua handshaking).

Si limita a: multiplexing/demultiplexing, piccoli controlli di integrità

3.1 perché si usa?

3.1.1 velocità trasferimento

Data la mancanza di handshake, non appena il processo applicativo trasmette i dati, UDP impacchetta il tutto in un segmento e invia al livello di rete.

Le applicazioni in real-time spesso dispongono di un "minimum sending rate", quindi non sopportano ritardi, che possono scaturire da eventuali handshake.

Questo però implica la problematica della congestione di rete. In quanto UDP non fornisce strumenti per evitarla, se la banda si satura nessun pacchetto arriverà a destinazione.

La problematica potrebbe riguardare in certi casi anche le sessioni TCP presenti sulla stessa banda saturata, i quali verrebbero "soffocati" a loro volta.

3.1.2 Mancanza di stati

Non mantiene gli stati della connessione e non tiene traccia di vari parametri (congestione, numeri di sequenza, ack) nei sistemi periferici.

Questo permette un minore peso

3.1.3 Leggerezza pacchetti

L'intestazione di un pacchetto è di 8 byte.

Più avanti vedremo che TCP ne occupa 20

3.2 Struttura segmenti

Definito dallo standard RFC 768.



4 campi di 2 byte ognuno.

- **Numero di porta di origine/destinazione:** permette il demultiplexing, ovvero permettono all'host destinatario di trasferire i dati al processo corretto
- **Lunghezza:** Lunghezza in byte del segmento UDP, utile in quanto ogni segmento ha una grandezza variabile in base al campo "dati"
- **Checksum:** permettono il controllo dell'integrità

3.2.1 checksum

Permette la rilevazione degli errori, ovvero verificare se durante il trasferimento alcuni bit sono stati alterati.

Lato mittente si effettua la somma di tutte le word da 16bit presenti nel segmento, per poi applicare il complemento ad 1 (si invertono i valori binari).

Lato destinatario si sommano le 3 word con checksum arrivato. Se il risultato è una word formata solo da 1 non ci sono stati errori.

[MITTENTE]

Date queste 3 word:

- 1) 0110011001100000
- 2) 0101010101010101
- 3) 1000111100001100

Si sommano tutte:

1) 0110011001100000+

2) 0101010101010101=

1011101110110101

1011101110110101+

3) 1000111100001100=

0100101011000010

N.B. il riporto dell'ultimo bit è stato sommato sul primo

Si effettua il complemento ad 1:

1011010100111101

[DESTINATARIO]

0100101011000010+

1011010100111101=

1111111111111111

Perché implementare un meccanismo di controllo degli errori, nonostante siano già presenti protocolli implementati a livello collegamento che ne occupano?

1. non c'è garanzia che tutti i collegamenti verifichino gli errori
2. se verificano, lo fanno a livello collegamento. Quindi se si verifica un errore nella memoria del router non verrebbe verificato.

Comunque UDP non implementa particolari soluzioni a questi errori di trasmissione, si limita ad uno delle seguenti:

- scartare segmento
- trasmettono all'applicazione con un avvertimento

4 Trasferimento dati affidabile

Protocollo di trasporto affidabile (reliable data transfer). Quindi con nessuna alterazione o perdita delle informazioni ed inoltre con un ordine di consegna uguale all'invio.

si considera al momento il trasferimento di dati unidirezionale

4.1 la costruzione del protocollo

4.1.1 metodo a stati finiti

implementato tramite macchina a stati finiti.

L'evento che causa la transizione è posto sopra la linea, le azioni intraprese dopo l'evento sono sotto.

Quando non succede niente si scrive "Λ"

4.1.1.1 rdt 1.0

si assume che il canale sottostante sia completamente affidabile.

sender:



a. rdt1.0: lato mittente

- aspetta la richiesta di invio pacchetto
- quando viene invocata "rdt_send(data)" prepara il pacchetto e lo invia

receiver:



b. rdt1.0: lato ricevente

- processo analogo

4.1.1.2 rdt 2.0

Si tiene conto di possibili errori, Ex: bit corrotti. avvenuti nei canali fisici quando il pacchetto viene trasmesso, propagato o inserito nel buffer.

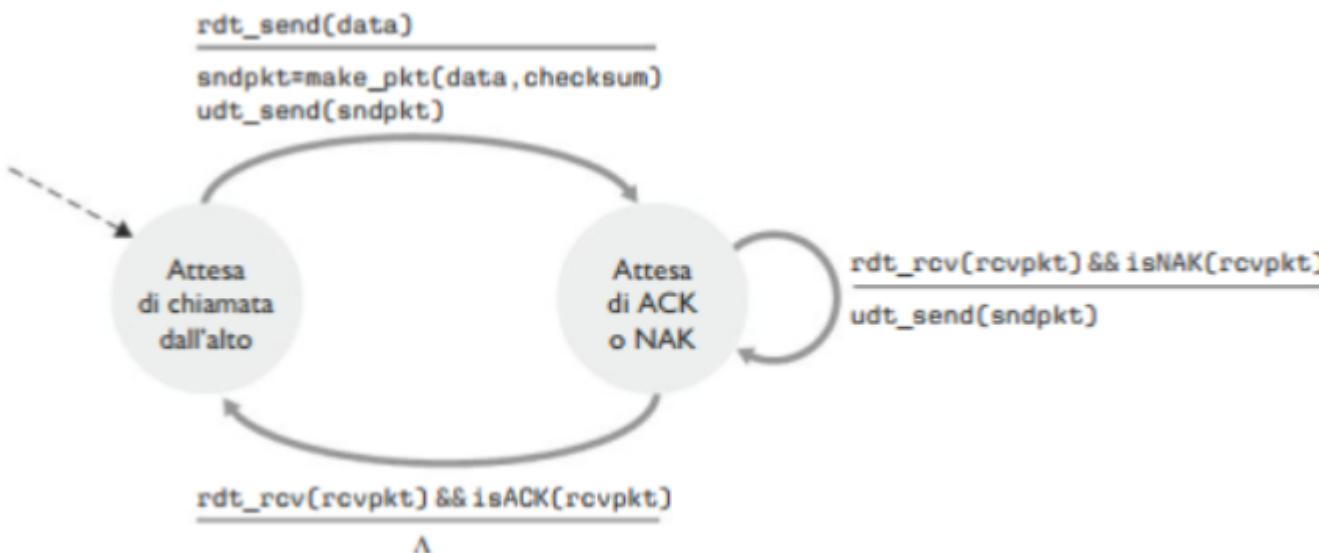
same alla [v1](#) ma si aggiunge un messaggio di conferma (ack) che sarà presente nel sender e receiver. Se la conferma è negativa, si ritrasmette.

⚠ Protocolli ARQ

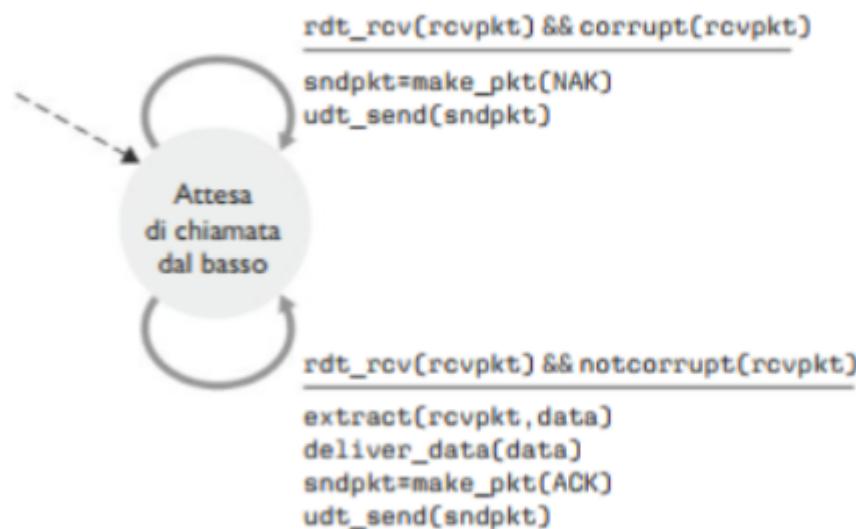
I protocollo affidabili che adoperano le ritramissioni sono detti "**Automatic Repeat Query/Request (ARQ)**".

Quest tipi di protocolli devono essere dotati di:

- meccanismi di error detection: Ex il checksum
- receiver feedback: invio di pacchetti particolari che indicano l'esito della comunicazione (ACK, NACK)
- ritransmission: un pacchetto ricevuto corrotto sarà ritrasmesso



a. rdt2.0: lato mittente



b. rdt2.0: lato ricevente

problema:

- quando il mittente è in stato di attesa è bloccato e non può ricevere altre `rdt_send()`. Questo è un problema comune tra i protocolli stop and wait
- si crea il problema di ricezione ack corrotti.

possibili soluzioni:

- ritrasmette ack all'infinito (non conviene).
Ex:"Cosa hai detto?. No cosa hai detto tu?. No cosa cosa hai detto tu..."
- aggiunta bit checksum
- riinvio del pacchetto.

4.1.1.3 rdt 2.1

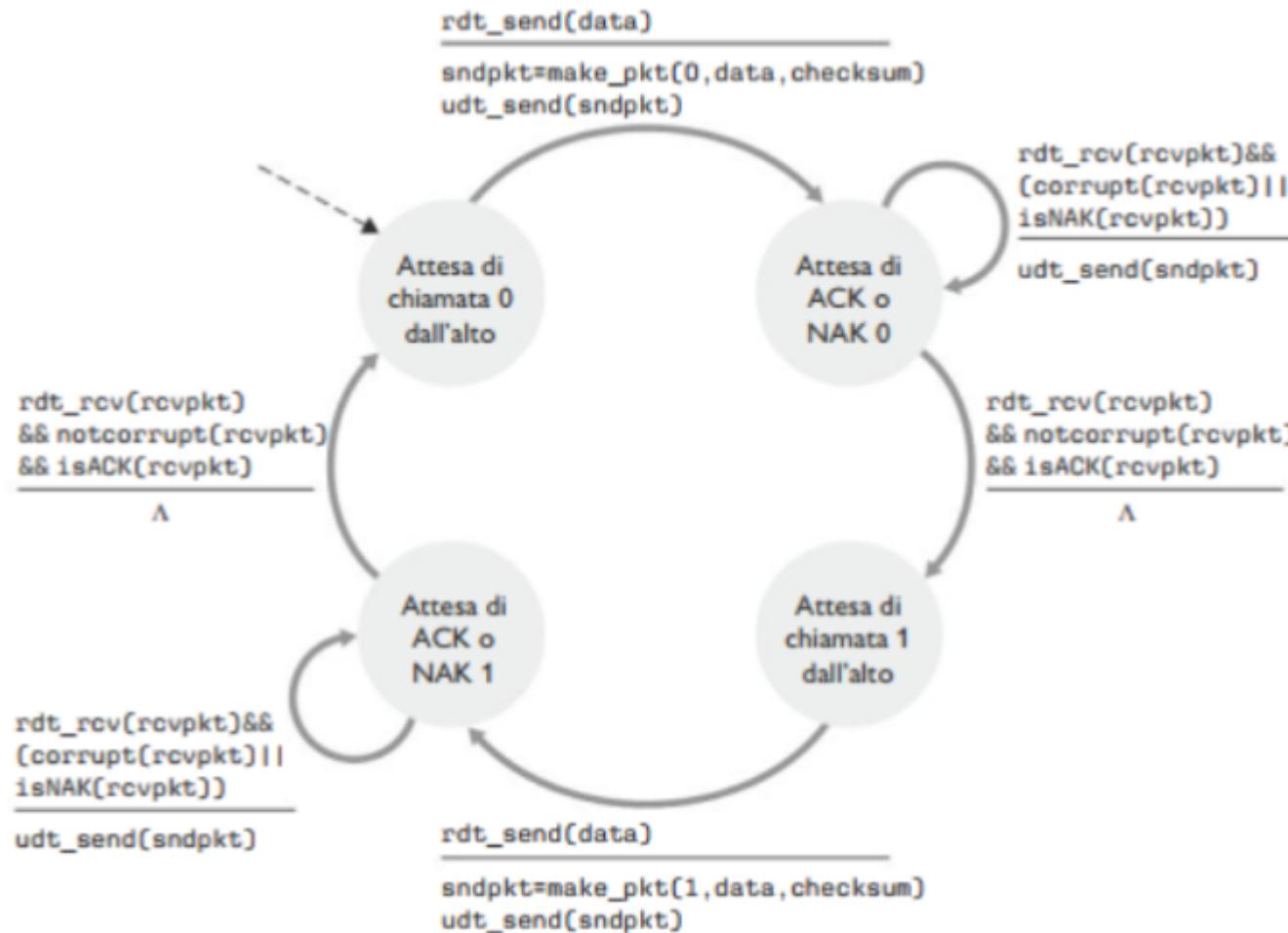
rispedendo i segmenti viene scaturito il problema dei "pacchetti duplicati".

Come faccio a sapere se il pacchetto appena arrivato è stato ritrasmesso o contiene dati nuovi?

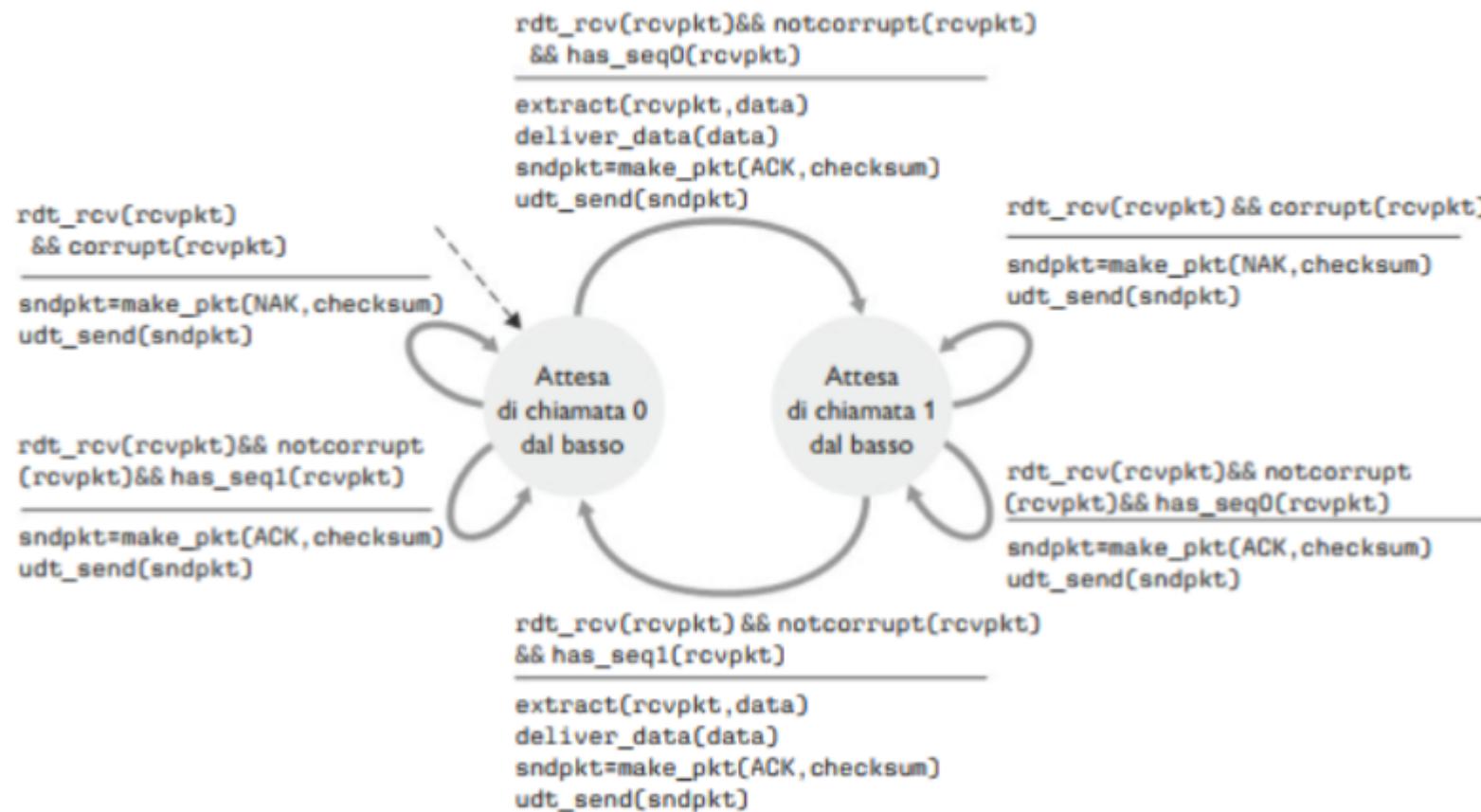
Una possibile soluzione è introdurre all'interno del pacchetto dati il "numero di sequenza" (sequence number). Ad esempio questo caso si implementa con un singolo bit

- Ex: se mando un pacchetto con id=0, mi aspetto di ricevere un `sequence_number = 0`. se ricevo altro, rimando.

mittente:



destinatario:

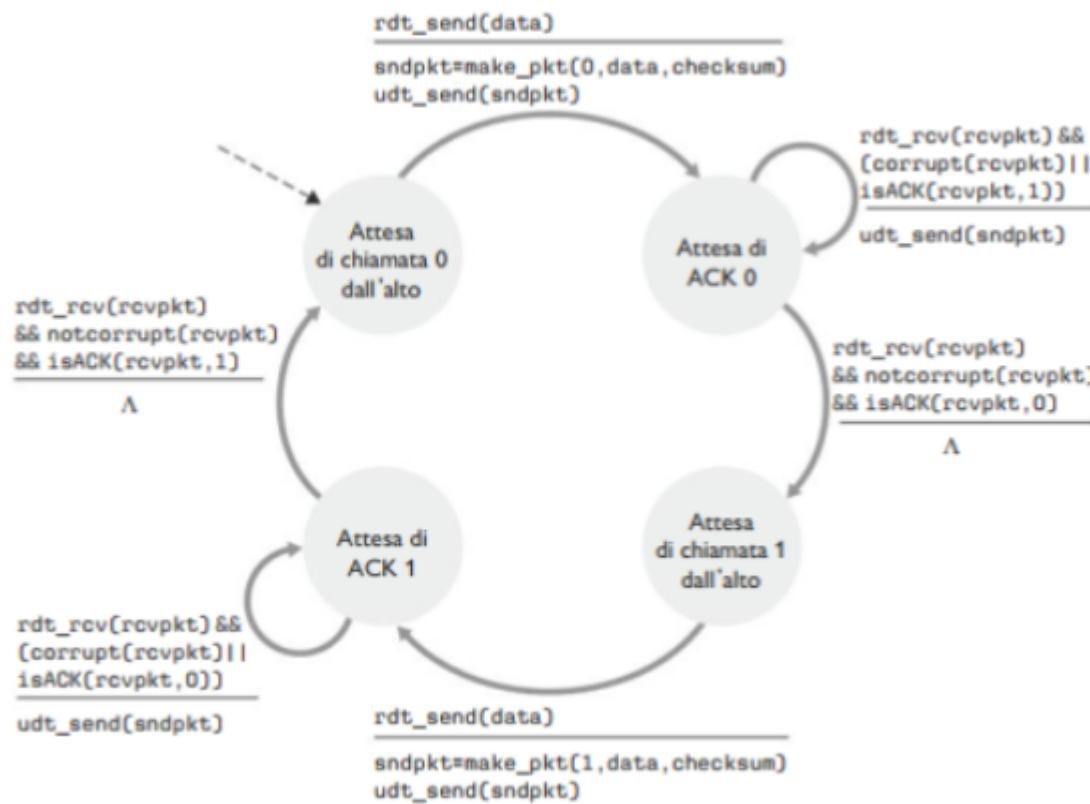


ura 3.12 Ricevente rdt2.1.

si potrebbe simulare il comportamento di un NACK inviando ACK duplicati in modo da indicare al mittente la ricezione non integra di un certo pacchetto. Implementato in [rdt 2.2](#).

4.1.1.4 rdt 2.2

Figura 3.13 Mittente
rdt2.2.



4.1.1.5 rdt 3.0

adesso si suppone che il canale possa danneggiare bit e smarrire pacchetti. Quindi il protocollo dovrà occuparsi della rilevazione e risoluzione di questi errori.

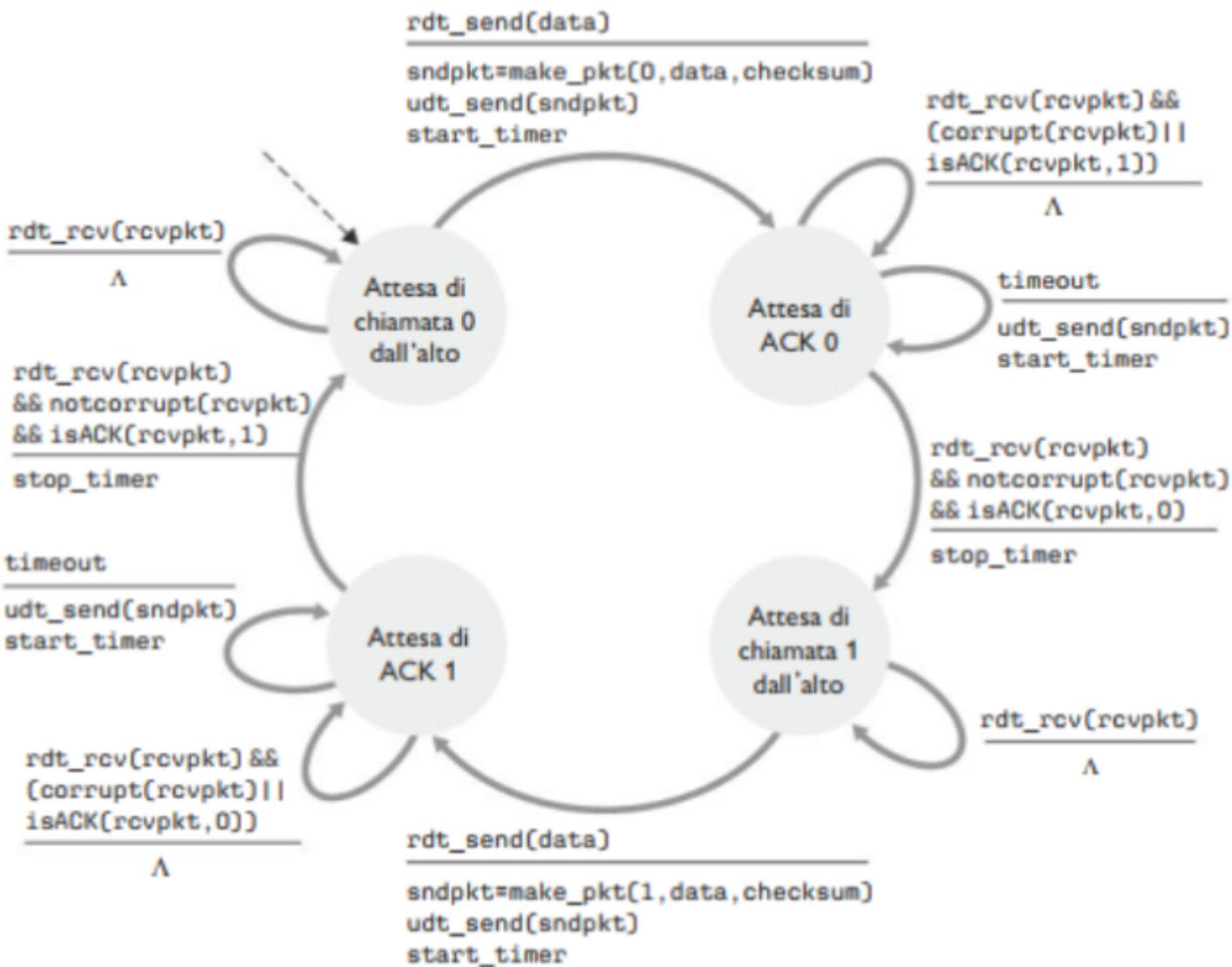
La **risoluzione** è già stata trattata dalla precedenti versioni con soluzioni come: checksum, sequence_number, ACK, ritrasmissioni.

La rilevazione necessita di un meccanismo in più. Tra i vari approcci ne usiamo uno che incentra sul mittente questo onere.

Si assume che esiste un **intervallo di tempo** dopo il quale si da per certo che il pacchetto sia smarrito, dopo il quale viene ritrasmesso. Il problema è quindi determinarlo, di sicuro sarà almeno il tempo che impiega andata e ritorno in aggiunta al tempo di elaborazione.

Con questo approccio si ricrea il problema dei pacchetti di dati duplicati che però può essere colmato con l'uso del [sequence_number](#)

Mittente

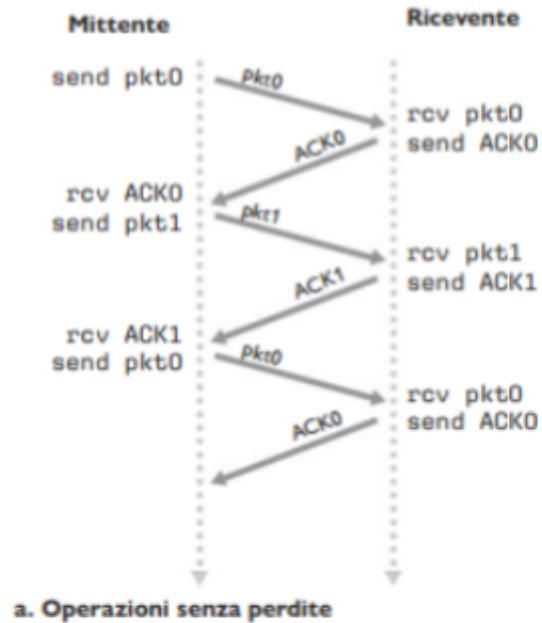


si dovrà quindi implementare lato mittente:

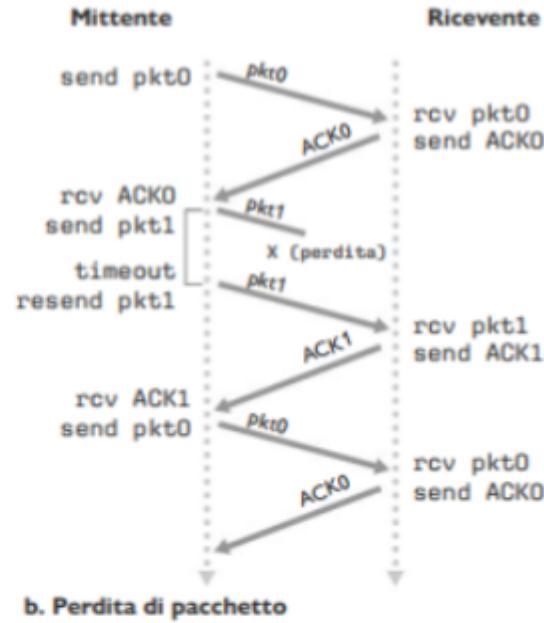
- inizializzazione count down timer al momento dell'invio di un pacchetto
- meccanismo di interrupt dopo scadenza contatore
- blocco contatore

⚠ problem

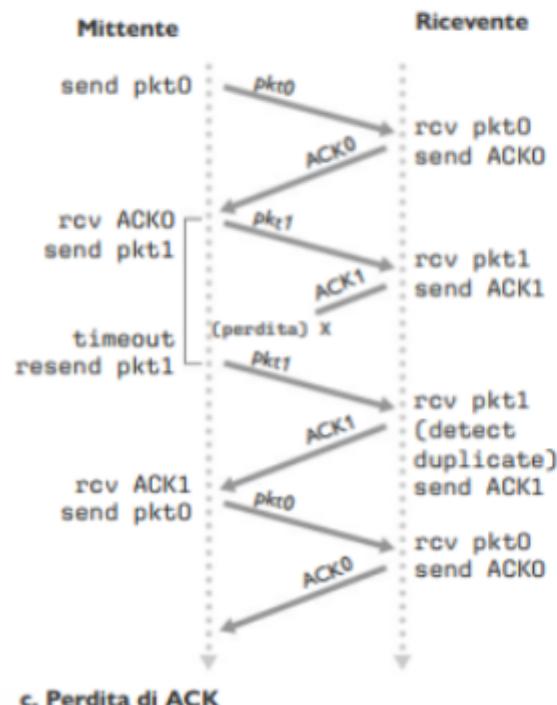
- si sta supponendo che prima o poi arrivi un pacchetto giusto.
- non si tiene conto di possibili ritardi dei pacchetti, che verrebbero interpretati come persi
- stiamo parlando di un protocollo "stop and wait", quindi non è funzionale alle reti odierne



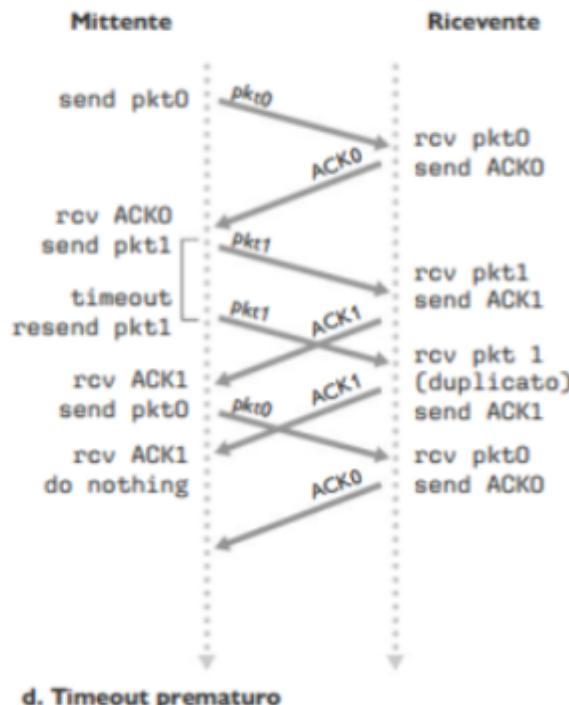
a. Operazioni senza perdite



b. Perdita di pacchetto



c. Perdita di ACK



d. Timeout prematuro

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

nel caso si creino ritardi, in quanto il sequence_number è di 1 solo bit (infatti prende il nome di "protocollo ad alternanza di bit") può accadere che viene spedito un ack che viene interpretato dal sender come conferma. una possibile soluzione è aggiungere bit.

4.1.1.6 ricapitolazzo macchine rdt

rdt 1.0: canale affidabile, invio e ricevo senza controllare

rdt 2.0: canale con errori, aggiungo ack/nack checksum

problem: ack corrotti

rdt 2.1: sequence_number

rdt 2.2: ack/nack => ack duplicati

rdt 3.0: timeout

4.1.1.7 problemi stop and wait: il throughput

un ulteriore problema dei protocolli stop and wait è: non ottimizzare il canale.

EX:

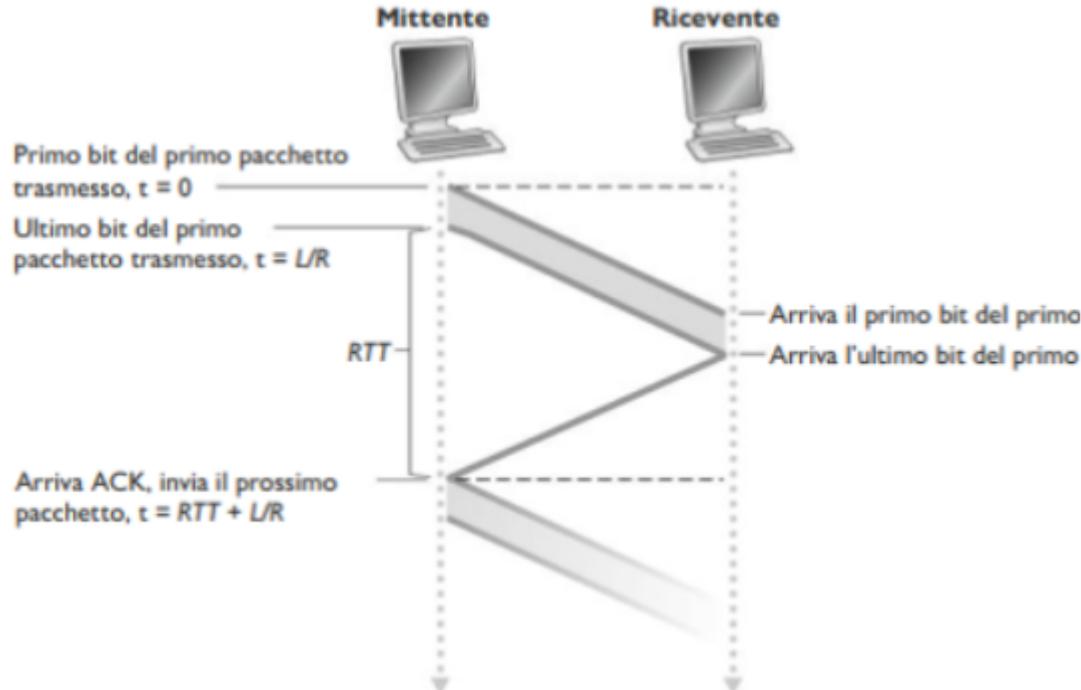
Supponendo di avere:

- R: un canale di 1Gbps (10^9 bit al secondo).
- L: trasmettere pacchetti dimensione 1000 byte

La propagazione sul canale avviene alla velocità della luce (200 km/s). Quindi $RTT = 30$ millisecondi.

tempo per trasmettere L :

$$d_t = \frac{L}{R} = \frac{8000}{10^9 \text{bit/s}} = 8\mu\text{s}$$



a. Funzionamento con stop-and-wait

Il mittente invia il primo bit a:

$$t = 0$$

L'ultimo verrà trasmesso a

$$t = \frac{L}{R} = 8\mu s$$

Quindi l'interno pacchetto raggiunge la destinazione a:

$$t = \frac{RTT}{2} + \frac{L}{R} = 15.008ms$$

Per semplificità si assume che l'ack non impieghi un tempo rilevante. Per questo possiamo dire che arriva a destinazione al tempo:

$$t = RTT + \frac{L}{R} = 30,008ms$$

A questo punto possiamo notare l'effettivo utilizzo del mittente come

$$U_{\text{mittente}} = \frac{L/R}{RTT + L/R} = \frac{0,008}{30,008} = 0,00027$$

Avremo quindi:

$$\text{throughput} = \frac{1000 \cdot 8}{30,008} = 267 kbps$$

rispetto ad un canale che permette un trasferimento ad 1 Gb/s.

pag 201

4.1.2 metodo a pipeline

una possibile soluzione al [problema throughput](#) è permettere al mittente di inviare pacchetti nonostante non siano arrivati ack precedenti.

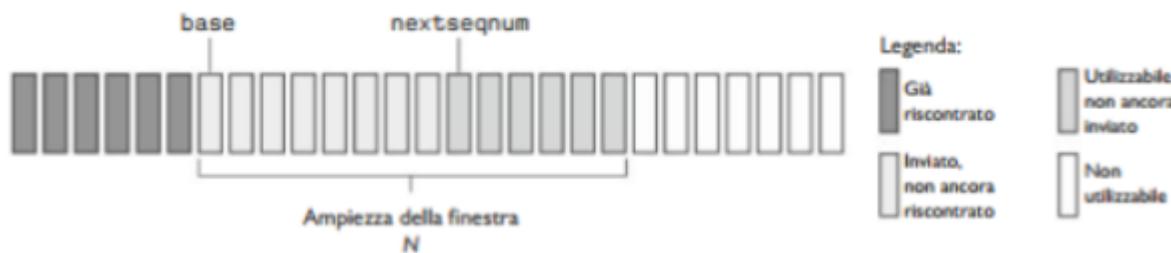
possibili conseguenze:

- ogni sequence_number dovrà essere univoco
- memorizzazione pacchetti non ancora confermati (lato mittente) o ricevuti correttamente (destinatario)

4.1.2.1 Go-Back-N (GBN)

il mittente può trasmettere più pacchetti senza aspettare i rispettivi ACK, il numero è limitato ad un certo N .

La seguente immagine rappresenta la visione del mittente rispetto all'intervallo dei numeri di sequenza:



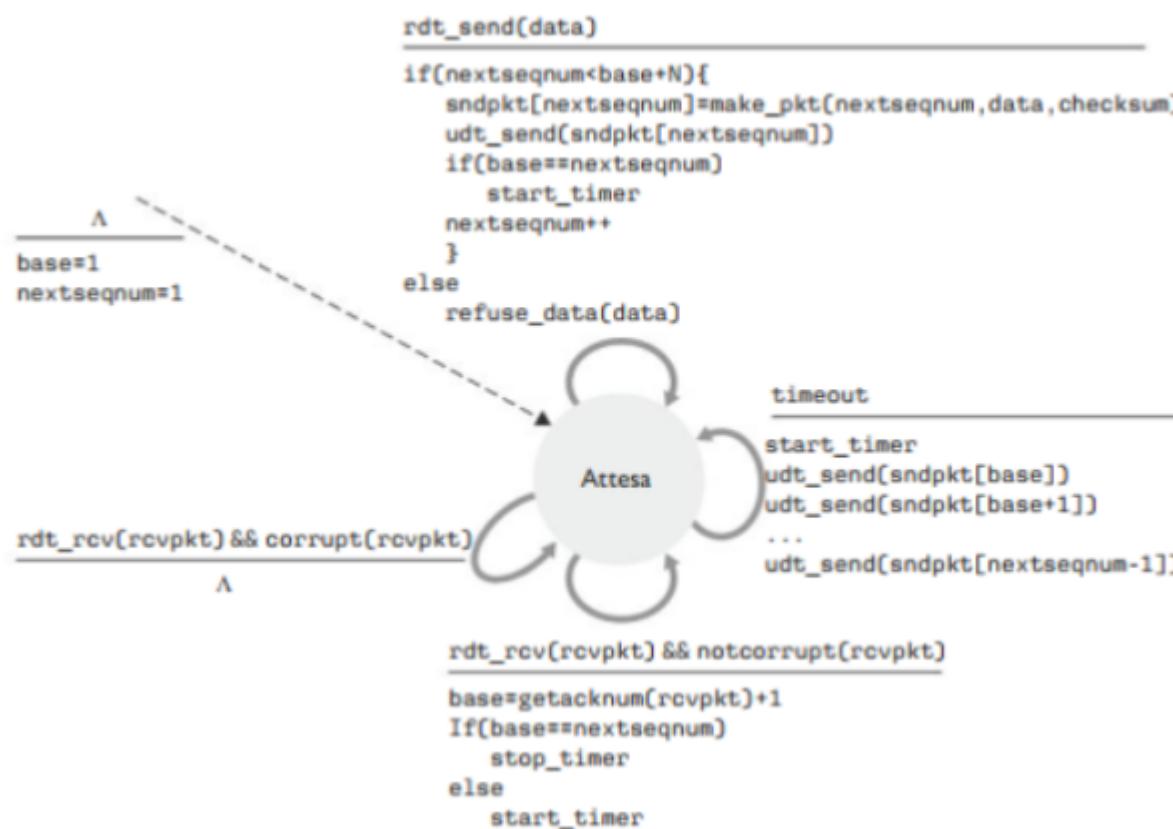
si possono distinguere 4 intervalli

- $[0, \text{base}-1]$: pacchetti trasmessi che hanno ricevuto gli ack
- $[\text{base}, \text{nextseqnum} - 1]$: inviati ma che non hanno ricevuto ack.
- $[\text{nextseqnum}, \text{base} + N - 1]$: possono essere usati per mandare pacchetti (nel caso arrivino dal liv. applicazione)
- $\{\text{seq_num} > \text{base} + N\}$: non possono essere ancora inviati

può essere visto come una fineta che trasla in base all'arrivo degli ack.

Per questo GBN viene detto "protocollo a finestra scorrevole".

automa stati finiti per mittente che implementa GBN:



Destinatario:

```
rdt_recv(rcvpkt)
  && notcorrupt(rcvpkt)
  && hasseqnum(rcvpkt,expectedseqnum)



---


extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(expectedseqnum,ACK,checksum)
udt_send(sndpkt)
expectedseqnum++

-----> A


---

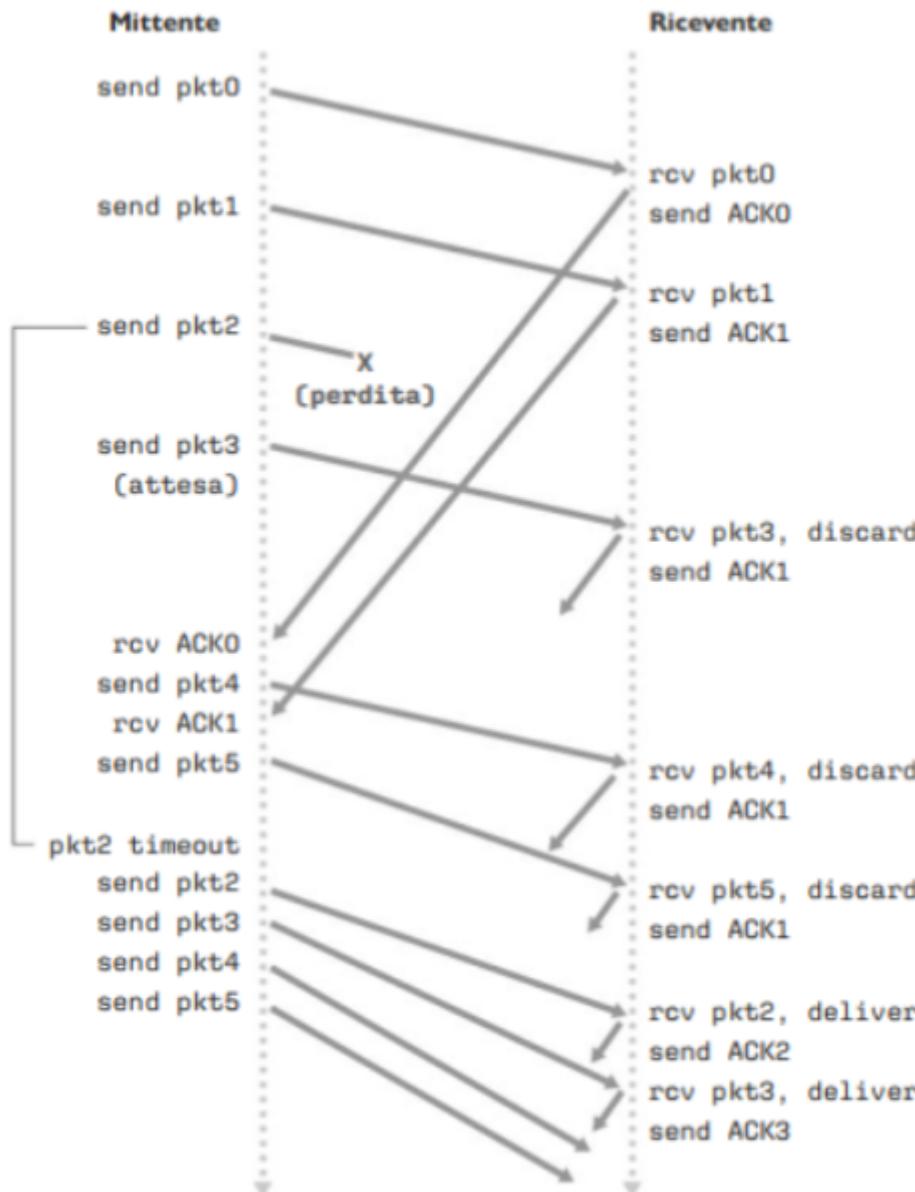

expectedseqnum=1
sndpkt=make_pkt(0,ACK,checksum)
```



lato destinatario sarà presente il controllo del numero di sequenza; se ricevuto in ordine (se l'ultimo numero di sequenza ricevuto è $n-1$) manda un ack per quel pacchetto. Si ricorda che gli ack sono "cumulativi", quindi inviando un $ack=n$ tutti i precedenti saranno confermati.

Nel contario (non arrivino in ordine) i pacchetti vengono ignorati. Si potrebbe pensare di salvare quelli arrivati in un buffer e poi consegnarli al livello superiore al momento opportuno. Però nel caso arrivi integro il pacchetto $n+1$ ed il pacchetto n si sia perso, da definizione GBN provvede a reinviarli entrambi.

Esempio GBN in funzione:



un problema legato a questo metodo è la probabile ritrasmissione di numerosi pacchetti in seguito ad un pacchetto corrotto.

Il problema si accentua nel caso di una finestra e larghezza di banda di grandi dimensioni, che provocano la presenza di tanti pacchetti all'interno

della pipeline.

Questa ritrasmissioni potrebbero provocare delle saturazioni.

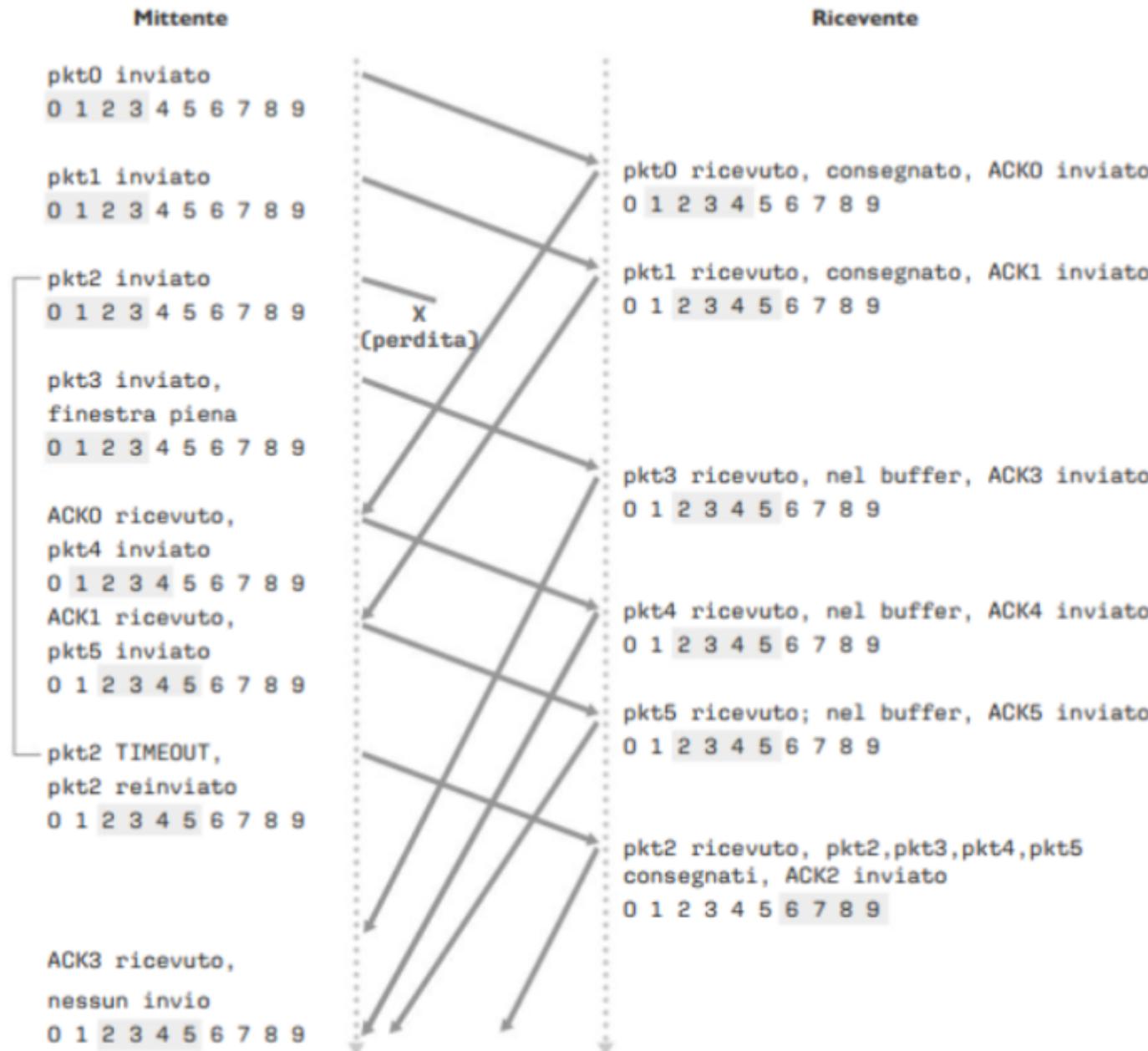
Per questo problema si ricorre ai [protocolli a ripetizione selettiva](#) (SR, selective repeat protocol), che evitano le ritrasmissioni non necessarie.

#attentionplis

la finestra ha dimensioni finite specificate nell'intestazione del pacchetto. dati quindi k bit avremo una finestra che comprende il seguente intervallo di numeri di sequenza

$$[0, 2^{k-1}]$$

4.1.2.2 Ripetizione selettiva



operazioni del mittente:

- **invio dati**

controlla il successivo numero di sequenza disponibile per inviare il pacchetto. se è all'interno della finestra viene inviato, caso contrario viene

salvato nei buffer o restituiti al livello superiore.

- **timeout**

fa uso di timeout temporali che cautelano contro la perdita dei pacchetti. per ogni pacchetto è presente un timeout che fa scattare un trigger al suo termine per l'invio di quel singolo pacchetto

- **ACK ricevuto**

controllo ack ricevuti. se riceve un `ack = send_base` (numero più piccolo della finestra) la base della finestra si muove verso il numero di sequenza più piccolo che non ha ricevuto ancora ack. se durante lo spostamento sono presenti pacchetti in precedenza bloccati dalla dimensione della finestra, vengono inviati.

destinatario:

- **ricezione pacchetti**

il pacchetto con numero di sequenza nell'intervallo $[rcv_base, \ rcv_base + N - 1]$ è stato ricevuto correttamente, quindi viene restituito un ack selettivo.

Se era già stato ricevuto viene inserito nel buffer.

Nel caso `numero_sequenza = rcv_base` allora tutti i pacchetti consecutivi nel buffer vengono passati al livello superiore.

- **ACK fuori intervallo**

per ritardi di propagazione potrebbe accadere che i pacchetti sono stati passati al livello superiore. Si vengono a creare però i seguenti problemi:

Ad esempio avendo a disposizione 2 bit (numeri di sequenza: 0, 1, 2, 3). vengono ricevuti correttamente i pacchetti $\{0, \ 1, \ 2\}$; il destinatario quindi muove la finestra "in avanti" comprendendo i pacchetti $\{3, \ 0, \ 1\}$.

Si potrebbe creare il problema che:

non riconosco nuovo segmento da ritrasmissione.

in caso di `window_size` troppo piccola, se viene perso un ack particolare, viene rispedito il relativo pacchetto che corrisponde ad un numero di sequenza compreso nella nuova finestra. quindi riconosciuto come nuovo.

Questo problema è legato anche al canale di comunicazione che potrebbe consegnare vecchie copie di un pacchetto. Nelle reti TCP moderne si ipotizza un tempo dopo il quale il pacchetto (numero di sequenza) non si più presente in rete, ed è circa 3 minuti.

Finestra del mittente
(dopo la ricezione)

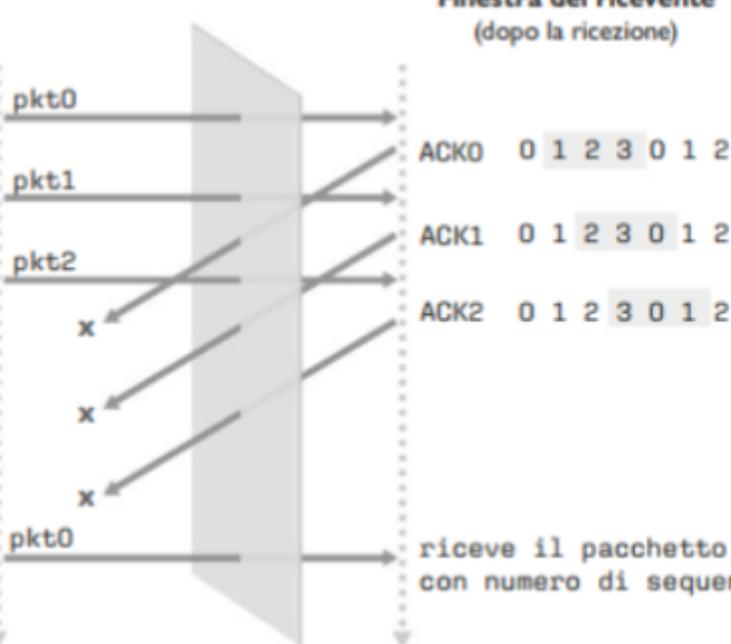
0 1 2 3 0 1 2

0 1 2 3 0 1 2

0 1 2 3 0 1 2

timeout
ritrasmette pkt0

0 1 2 3 0 1 2



a.

Finestra del mittente
(dopo la ricezione)

0 1 2 3 0 1 2

0 1 2 3 0 1 2

0 1 2 3 0 1 2

0 1 2 3 0 1 2

0 1 2 3 0 1 2

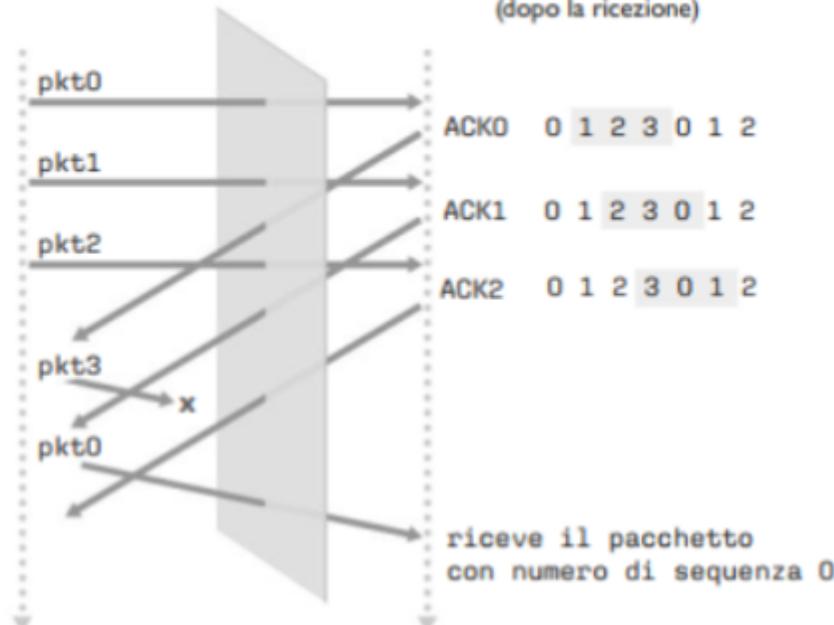
Finestra del ricevente
(dopo la ricezione)

ACK0 0 1 2 3 0 1 2

ACK1 0 1 2 3 0 1 2

ACK2 0 1 2 3 0 1 2

riceve il pacchetto
con numero di sequenza 0



b.

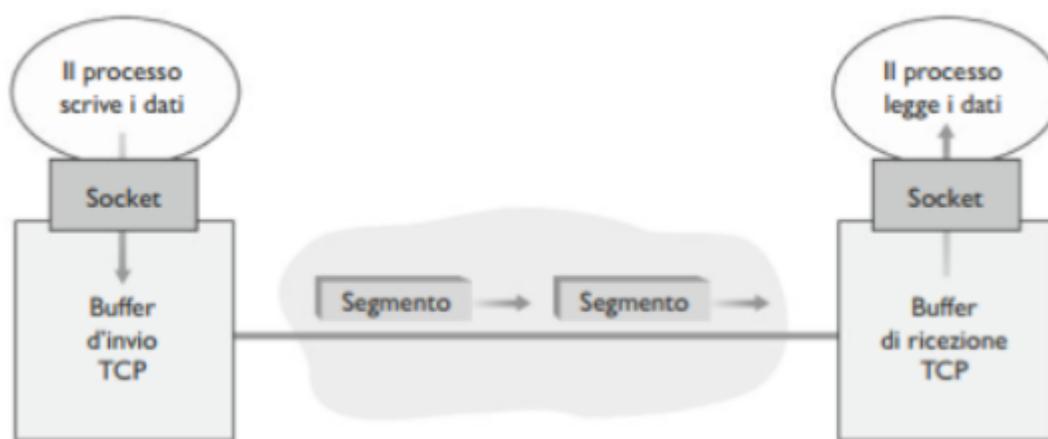
non so quindi distinguere una ritrasmissione da un invio di un nuovo pacchetto. "è come se ci fosse una cortina....".

si deve quindi capire come impostare in maniera ottima le dimensioni della finestra (fine capitolo).

quindi occorre avere un valore di window size \leq metà spazio numeri di sequenza (2^k , con k =bit destinati al numero di sequenza)

4.2 TCP

protocollo di trasporto usato da Internet. orientato alla connessione (handshake a tre vie).
offre un servizio full-duplex ed anche point to point.



Tra i vari compiti si occupa di dirigere i dati al buffer di invio della connessione (uno è riservato per l'handshake). All'interno del TCP non è specificato **quando** inviare i dati ma semplicemente quando è più conveniente.

La massima quantità dei dati prelevabili e posizionati nel segmento viene limitata dalla "dimensione massima di segmento" (MSS, Maximum segment size), impostata determinando:

1. **unità trasmissiva massima** (MTU, Maximum Transmission Unit): lunghezza del frame più grande che supporta il canale
2. scegliendo di conseguenza un MSS tale che il segmento TCP stia all'interno di un singolo frame a livello collegamento, aggiungendo le intestazioni TCP/IP (40byte).

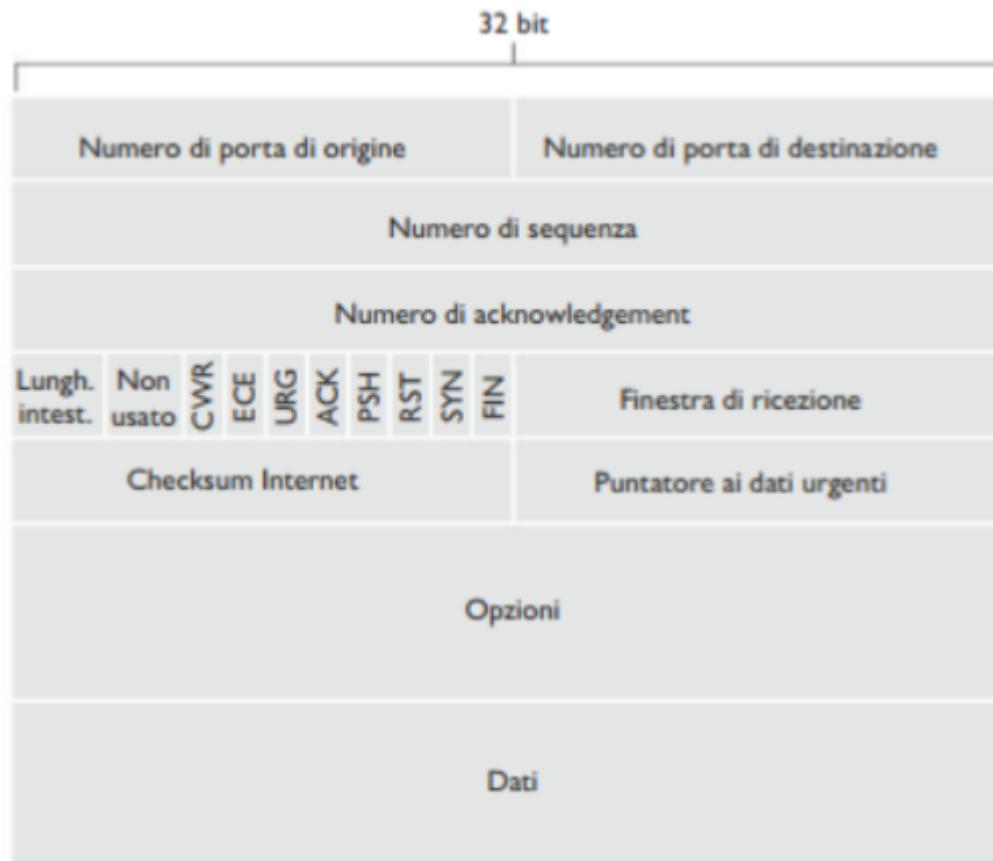
In genere i protocolli ethernet hanno MTU di circa 1500 byte.

$$MSS = MTU - [\text{dimensione_header_TCP/IP}]$$

In casi di sforamento del MSS si procede a frammentare in più segmenti

Ricapitolando TCP si occupa di accoppiare i dati del client ad un intestazione TCP, andando a creare i cosiddetti "segmenti TCP". Questi verranno poi passati al livello di rete che li incapsulerà nei datagrammi IP per poi essere immessi in rete.

4.2.1 struttura segmenti

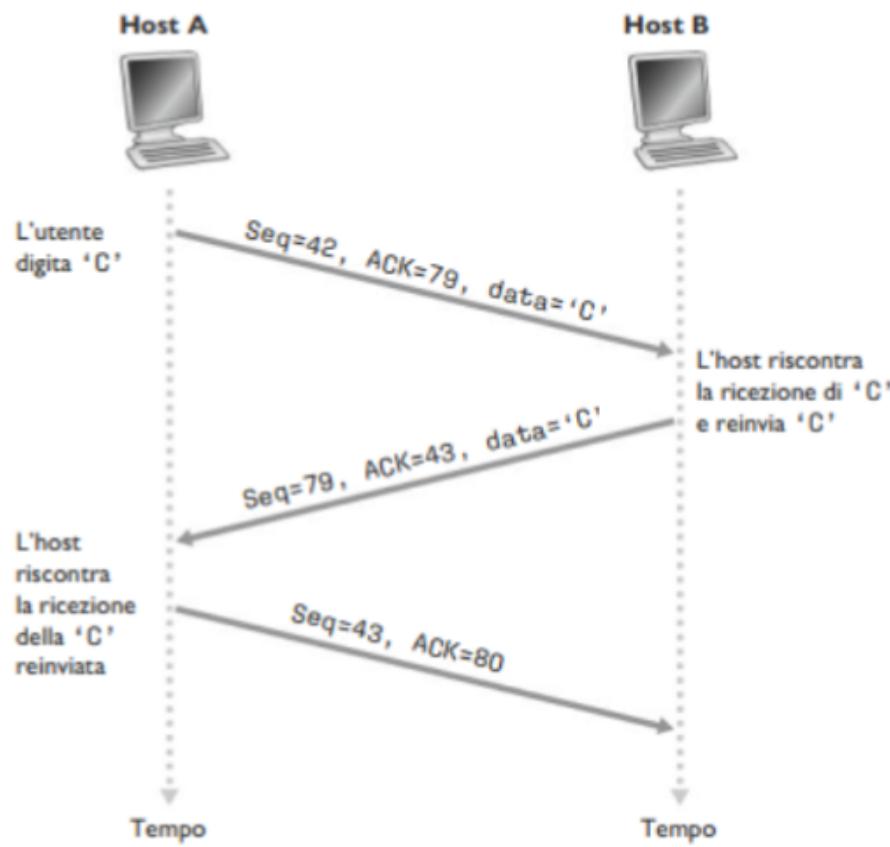


come in UDP sono presenti: porta origine e destinazione (multiplexing/demultiplexing) ed il checksum.

Inoltre sono compresi:

- **Numero di sequenza e Numero di acknowledgment**

32 bit, per implementare il trasporto affidabile. Il numero di sequenza indica il numero di sequenza del primo byte del campo dati; ACK invece è il numero del successivo byte che l'host sta aspettando. [qui un in introduzione](#). viene fatto uso di [ack cumulativi](#) (attenzione: TCP non è proprio implementato come un GBN ma ci sono [alcune differenze](#)). Un caso d'uso particolare di questo campo è il [feedback visivo di telnet](#):



in questo caso particolare il trasporto degli ack è detto piggybacked o piggyback, in quanto nel segmento di ritorno la lettera "C". L'ultimo segmento inviato è solo una conferma.

- **Finestra di ricezione (receiver window field):**

16 bit, per il controllo del flusso.

- **Lunghezza intestazione:**

4 bit, indica la lunghezza dell'intestazione TCP in multipli di 32 bit. la lunghezza è variabile a causa delle opzioni, ma generalmente è vuoto quindi di norma la lunghezza è 20 byte

- **Reserved**

4 bit, Bit non utilizzati e predisposti per sviluppi futuri del protocollo; dovrebbero essere impostati a zero.

- **Flag:** 6 bit
 - ACK: 1 bit, indica se il segmento contiene un ACK per un segmento consegnato con successo
 - RST, SYN, FIN: 3 bit usati per impostare e chiudere la connessione (reset, inizio, fine)
 - CWR, ECE: 2 bit per il controllo congestione.
 - cwr = 1 indica che il mittente ha ricevuto un segmento con ece = 1.
 - ece = 1 indica che supporta [ecn](#) durante il 3 way handshake. router "marchia" il datagramma [notificando congestione](#)
 - PSH: 1 bit, se il valore è 1 i dati devono essere inviati al livello superiore (non bufferizzati)
 - URG: indica la presenza di dati nel segmento con priorità di consegna alta (urgenti), puntati dal puntatore dei dati urgenti.
- **Puntatore dati urgenti:**
16 bit, offset rispetto al sequence_number. Usato per comunicare al livello app del destinatario quali sono i dati urgenti.

4.2.2 timeout e stime RTT

TCP fa uso di un timeout come in [rdt 3.0](#). Il problema è capire come stipulare il timeout.

Dovrà essere sicuramente più grande del RTT (Round trip time), ma di quanto? come deve essere stimato RTT?

4.2.2.1 Calcolo RTT

SampleRTT = intervallo di tempo tra invio del segmento e ricezione del suo ACK.

In generale ci si sofferma sul SampleRTT di un singolo segmento alla volta. Inoltre il calcolo viene effettuato solo sui segmenti inviati una singola volta e non sulle copie.

Data la fluttuazione dei valori di SampleRTT (congestione, carico di lavoro sui sistemi periferici) si effettua una media che TCP chiama

EstimatedRTT

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

con $\alpha = \frac{1}{8}$

Si tratta di una semplice media pesata che dà maggiore importanza ai valori precedenti, rispetto a quelli vecchi (come lo scheduling di sistemi).

Questo aspetto è fondamentale in quanto riflettono meglio la congestione attuale. Questa media è detta **media mobile esponenziale ponderata** (EWMA, Exponential weighted moving average), esponenziale in quanto il peso dei valori decresce esponenzialmente.

Si definisce anche una stima di quanto SampleRTT si discosti da EstimatedRTT

$$DevRTT = (1 - \beta) \times DevRTT + \beta \cdot | SampleRTT - EstimatedRTT |$$

4.2.2.2 Timeout

Non dovrà essere minore (ritrasmissioni inutili) di EstimatedRTT, ma neanche troppo più grande (ritardi).

Quindi si imposta:

$$\text{Timeout} = \text{RT0} \text{ (Retrasmission timeout)} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$

Sarà quindi grande quando ci sarà molta fluttuazione nei valori DevRTT e viceversa.

Il valore iniziale del timeout è generalmente 1 secondo.

Quando si verifica un timeout, la variabile viene raddoppiata al fine di evitare un timeout prematura di un probabile ACK in arrivo. ^{^92ee04n}
timeout è tipo un interrupt che raddoppia il valore dell'intervallo.

4.2.3 trasferimento dati affidabile

Come già detto il livello di rete non garantisce né la consegna ordinata né l'integrità dei dati, per questo TCP crea un servizio di trasporto dati affidabile.

Per quanto riguarda l'uso dei timer in TCP si usa un solo timer anche per più segmenti. Prima si era parlato di associare un timer ad ogni segmento, ma è efficace solo teoricamente, concretamente richiede la gestione di considerevoli risorse.

Si ricorda che il timer in uso è li [TimeoutInterval](#).

RTO è sovrastimato btw no problema perché ci troveremo in fast retrasmitt.

4.2.3.1 Compiti Mittente

```

/* Ipotizziamo che il mittente non subisca imposizioni dal
controllo di flusso o di congestione TCP, che i dati dall'alto
abbiano dimensione inferiore a MSS e che il trasferimento dati
avvenga in un'unica direzione */

NextSeqNum = InitialSeqNumber
SendBase = InitialSeqNumber

loop (per sempre) {
    switch (evento)

        evento: dati ricevuti dall'applicazione a livello
        superiore crea il segmento TCP con numero di sequenza
        NextSeqNum
            if (il timer attualmente non è in funzione)
                avvia il timer
            passa il segmento a IP
            NextSeqNum = NextSeqNum + lunghezza(dati)
            break;

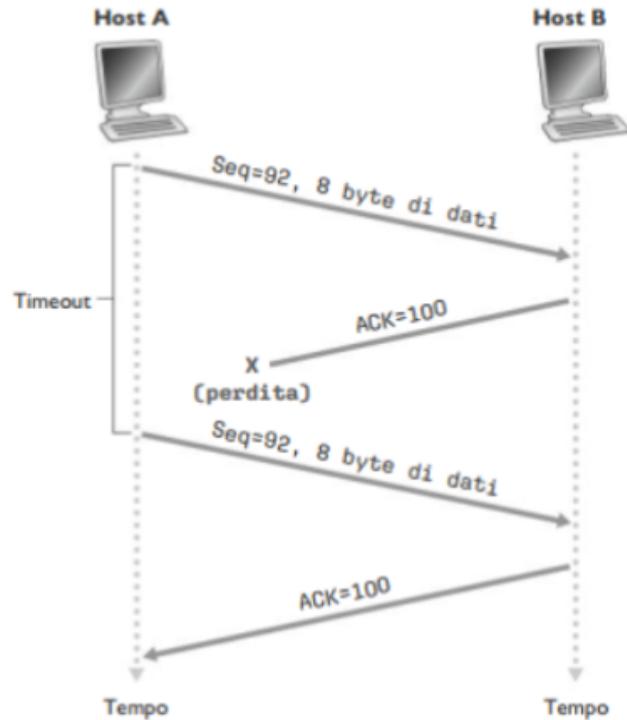
        evento: timeout del timer
            ritrasmetti il segmento che non ha ricevuto ACK con il
            più piccolo numero di sequenza
            avvia il timer
            break;

        evento: ACK ricevuto, con valore del campo ACK pari a y
            if (y > SendBase) {
                SendBase = y
                if (esistono attualmente segmenti senza ACK)
                    avvia il timer
            }
            break;
    } /* fine del loop */
}

```

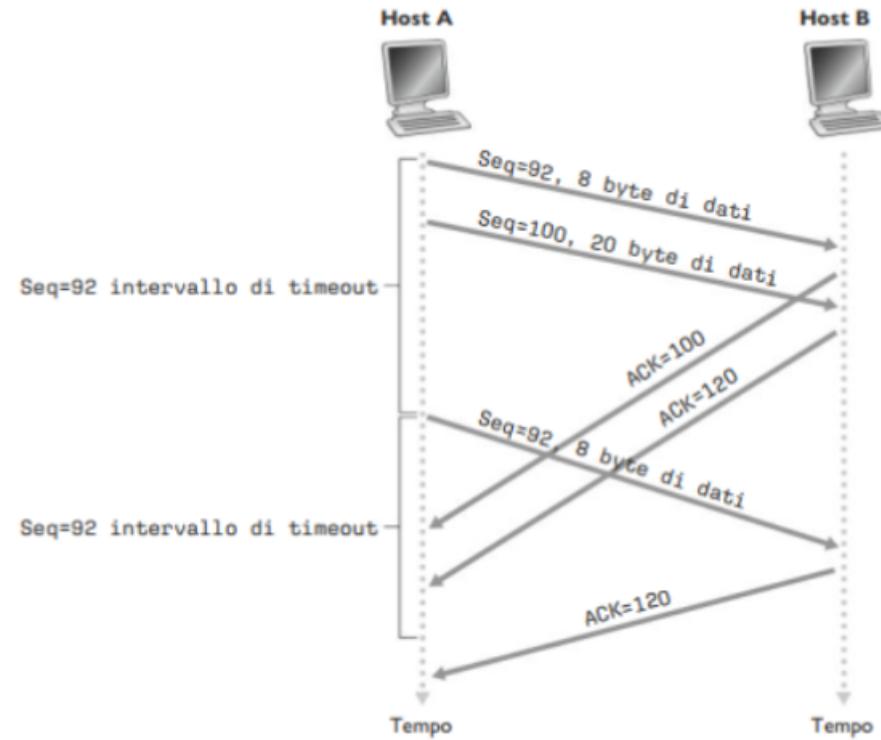
4.2.3.2 scenari possibili

perdita ACK:



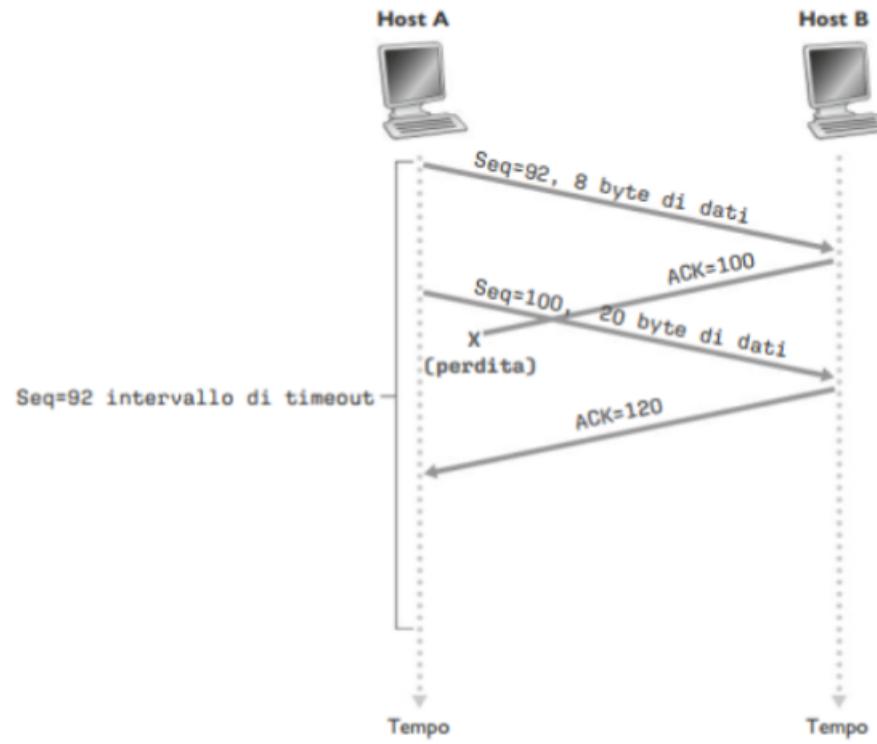
L'ack viene perduto e si trigger il timeout. Rispedisce il segmento e riavvia (raddoppio) il timer

Segmento non ritrasmesso:



Scatta il timeout per $\text{Seq}=92$, il segmento viene ritrasmesso. fino a quando non arriva il relativo ack, il $\text{Seq}=100$ non viene ritrasmesso.

Ack cumulativo:



Il primo ack viene perso, ma prima del timeout arriva l'ack cumulativo.

4.2.3.3 Raddoppio intervallo di timeout

Anziché derivarlo da EstimatedRTT e DevRTT, la maggior parte delle implementazioni TCP procedono a raddoppiare il valore precedente.

La scadenza del timer può essere causata da una congestione della rete, se durante questi periodi una sorgente continua a mandare pacchetti può aggravare la situazione.

4.2.3.4 Ritrasmissione rapida

Il lato negativo è che il periodo del timer può rivelarsi lungo, causando ritardi.

La ritrasmissione non avviene per forza solo alla fine di un timer, ma anche grazie all'arrivo di ACK duplicati.

Un elenco dei vari scenari di invio ACK:

Evento	Azione del ricevente TCP
Arrivo ordinato di segmento con numero di sequenza atteso. Tutti i dati fino al numero di sequenza atteso sono già stati riscontrati.	ACK ritardato. Attende fino a 500 millisecondi per l'arrivo ordinato di un altro segmento. Se in questo intervallo non arriva il successivo segmento, invia un ACK.
Arrivo ordinato di segmento con numero di sequenza atteso. Un altro segmento ordinato è in attesa di trasmissione dell'ACK.	Invia immediatamente un singolo ACK cumulativo, riscontrando entrambi i segmenti ordinati.
Arrivo non ordinato di segmento con numero di sequenza superiore a quello atteso. Viene rilevato un buco.	Invia immediatamente un ACK duplicato, indicando il numero di sequenza del prossimo byte atteso (che è l'estremità inferiore del buco).
Arrivo di segmento che colma parzialmente o completamente il buco nei dati ricevuti.	Invia immediatamente un ACK, ammesso che il segmento cominci all'estremità inferiore del buco.

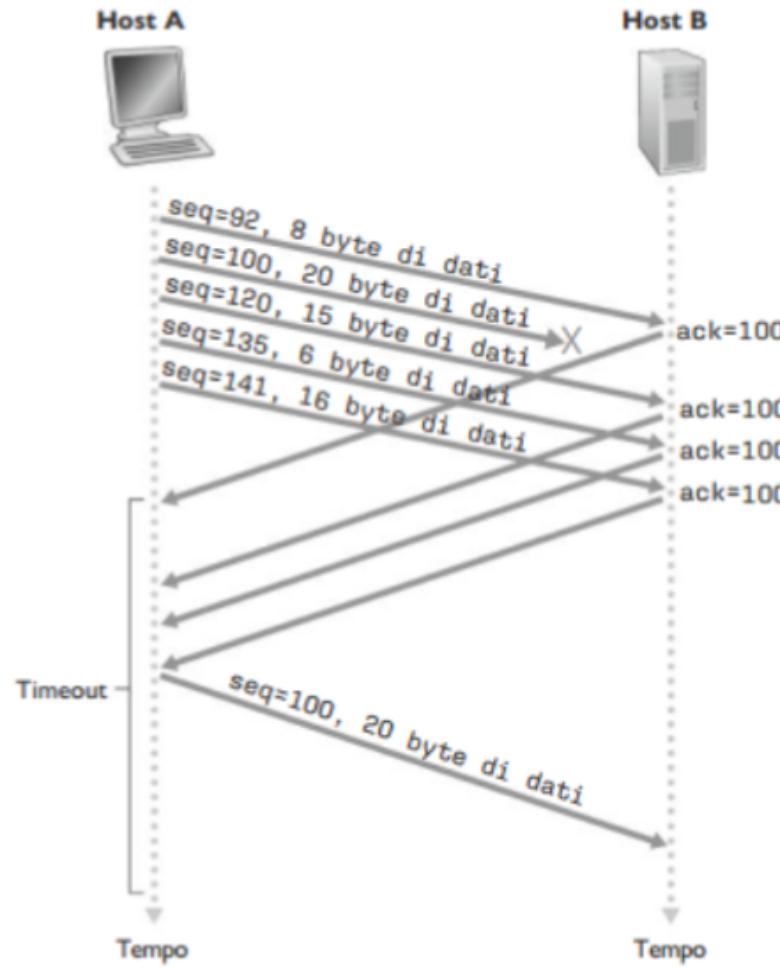
Nel caso di 3 ACK duplicati TCP effettua una ritrasmissione rapida (più avanti verrà spiegato perché proprio 3).

Il [codice del mittente](#) cambia di conseguenza:

```
/* Ipotizziamo che il mittente non subisca imposizioni dal  
controllo di flusso o di congestione TCP, che i dati dall'alto  
abbiano dimensione inferiore a MSS e che il trasferimento dati  
avvenga in un'unica direzione */
```

```
NextSeqNum = InitialSeqNumber  
SendBase = InitialSeqNumber  
  
loop (per sempre) {  
    switch (evento)  
  
        evento: dati ricevuti dall'applicazione a livello  
        superiore crea il segmento TCP con numero di sequenza  
        NextSeqNum  
            if (il timer attualmente non è in funzione)  
                avvia il timer  
                passa il segmento a IP  
                NextSeqNum = NextSeqNum + lunghezza(dati)  
                break;  
  
        evento: ACK ricevuto, con valore del campo ACK pari a y  
            if (y > SendBase) {  
                SendBase = y  
                if (esistono attualmente segmenti che non hanno  
                    ricevuto un ACK)  
                    avvia il timer  
                }  
            else { /* un ACK duplicato per un segmento */  
                incrementa il numero di ACK duplicati ricevuti per y  
                if (numero di ACK duplicati ricevuti per y == 3) {  
                    /* ritrasmissione rapida */  
                    rispedisci il segmento con numero di sequenza y  
                }  
            }  
            break;  
    } /* fine del loop */
```

Ex:



4.2.3.5 TCP è GBN oppure SR

sembra che TCP sia implementato secondo il metodo [Go-back-N](#), però sorgono alcune differenze.

Si ricorda che gli ACK sono cumulativi perciò il mittente deve memorizzare solo il numero di sequenza più basso che non ha ricevuto ack (SendBas) ed il numero di sequenza del successivo byte da inviare (NextSeqNum).

Moltre implementazioni TCP dispongono un buffer (non ordinato) dei segmenti ricevuti correttamente.

Ad esempio:

Si inviano N segmenti, che vengono ricevuti senza errori. Ipotizziamo che il destinatario invii i rispettivi ack, ma che quello relativo ad un certo pacchetto n si perda.

- GBN si comporterebbe riinviaendo tutti i pacchetti da $n, n + 1, \dots$ fino ad N
- TCP si limita a ritrasmettere al massimo il pacchetto n , ma in certi casi nemmeno questo in quanto potrebbe arrivare un ack cumulativo prima della scadenza del timer di n

Buttato lì a caso: Btw con RFC 2018 è stata proposta una versione ibrida di TCP che implementa gli [ack selettivi](#) per l'ultimo segmento ricevuto senza errori ed in ordine.

4.2.4 controlli di flusso

Si ricorda che gli host agli estremi della connessione TCP riservano dei buffer che contengono i byte corretti ed in sequenza, tramite i quali i processi applicativi possono leggere (quando ritiene opportuno, non per forza all'istante successivo all'arrivo). Per questo se l'app è lenta il buffer può andare in overflow.

Per questo TCP offre un **flow-control service** per evitare la saturazione del buffer di destinazione.

Si basa sul paragonare la velocità di invio del mittente con la velocità di lettura dell'applicazione ricevente.

Si suppone che i segmenti non in ordine siano scartati.

4.2.4.1 implementazione

viene disposta una finestra di ricezione, detta anche "**rwnd**", (approccio simile alle [implementazioni pipeline](#)), ovvero una variabile che fornisce al mittente un'indicazione dello spazio libero disponibile nel buffer destinatario.

In quanto TCP è full duplex, sia mittente che destinatario mantengono questa finestra.

Ogni qual volta un mittente invia dati, il destinatario dovrà riservare spazio nel suo buffer, il cosiddetto "**RcvBuffer**".

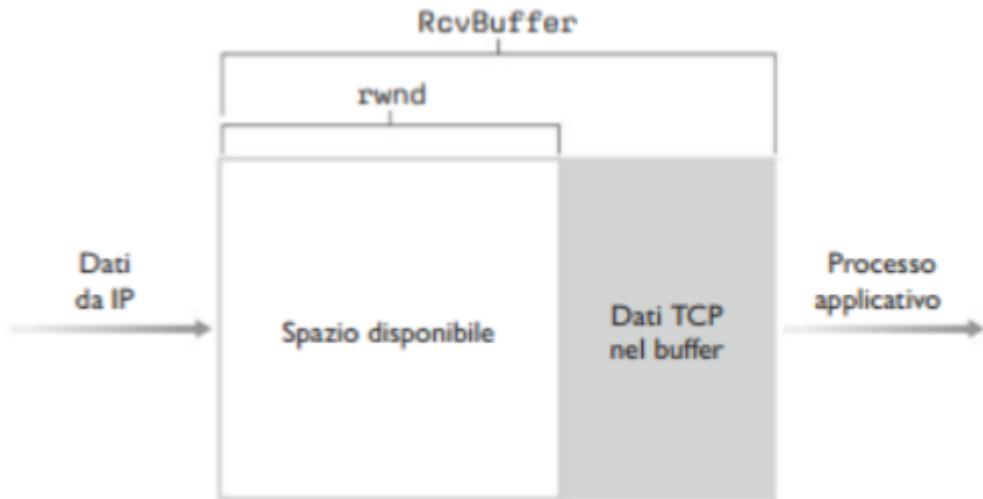


Figura 3.38 Finestra di ricezione (**rwnd**) e buffer di ricezione (**RcvBuffer**).

Occorre definire le seguenti variabili:

- **LastByteRead**: numero dell'ultimo byte letto dal buffer da parte del processo applicativo
- **LastByteRcv**: Numero dell'ultimo byte che proviene dalla rete ed è stato copiato nel buffer

Per non avere overflow dovremmo per forza avere:

```
LastByteRead - LastByteRcv <= RcvBuffer
```

La finestra di ricezione viene quindi impostata in base alla quantità di spazio disponibile nel buffer

```
rwnd = RcvBuffer - (LastByteRead - LastByteRcv)
```

Il mittente tiene traccia delle variabili:

- **LastByteSent**: ultimo byte inviato
 - **LastByteAcked**: ultimo byte confermato da ACK
- dovrà quindi controllare che la quantità di dati non confermati da ACK sia al di sotto del valore di rwnd

LastByteSent - LastByteAcked <= rwnd

Sorge però il seguente problema: se blocco la trasmissione di segmenti nel momento in cui arriva un segmento con `rwnd=0`, nel momento in cui il buffer comincia a svuotarsi, il mittente starà fermo. Per questo si usa mandare 1 byte al destinatario, il quale risponderà con ACK, con all'interno del segmento il valore rwnd.

4.2.5 gestione connessioni

Come viene stabilita e rilasciata la connessione TCP?

Attraverso la procedura denominata "Handshake a tre vie", che si caratterizza nello scambio di segmenti in 3 fasi.

4.2.5.1 Apertura connessione

avviene secondo 3 passaggi:

1. Segmento SYN

Client invia uno speciale segmento TCP non contenente dati a livello applicativo, ma il bit "SYN" nell'intestazione è posto ad 1. Per questo prendo il nome di "segmento SYN".

Inoltre sceglie un numero di sequenza casuale (`client_isn`) e lo pone nel campo numero di sequenza.

Il tutto viene incapsulato in un datagramma IP ed inviato al server

2. Segmento SYNACK

Se il pacchetto arriva all'host server, questo lo estraе, alloca i buffer e le variabili TCP alla connessione; per poi inviare la risposta al mittente con il cosiddetto "Segmento SYNACK".

Le caratteristiche sono le seguenti:

- non contiene dati applicativi
- bit SYN = 1
- campo ACK = `client_isn + 1`
- numero di sequenza è detto `server_isn`

3. Ricezione SYNACK

Alla ricezione anche il client alloca buffer e variabili alla connessione, per poi rispondere al server con un ulteriore segmento. Pone il bit ACK = `server_isn + 1`, bit SYN = 0 dato che la connessione è stata stabilita, inoltre può inserire i dati applicativi da inviare al server.

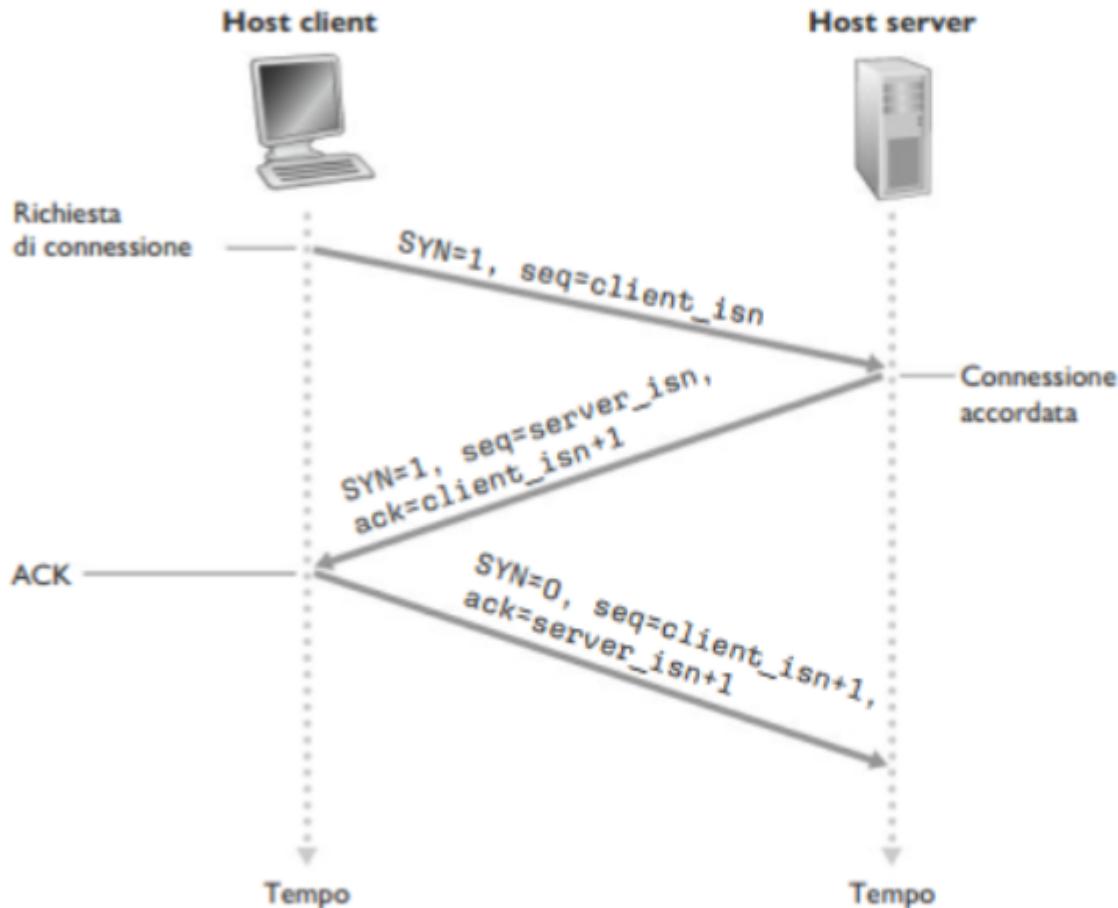


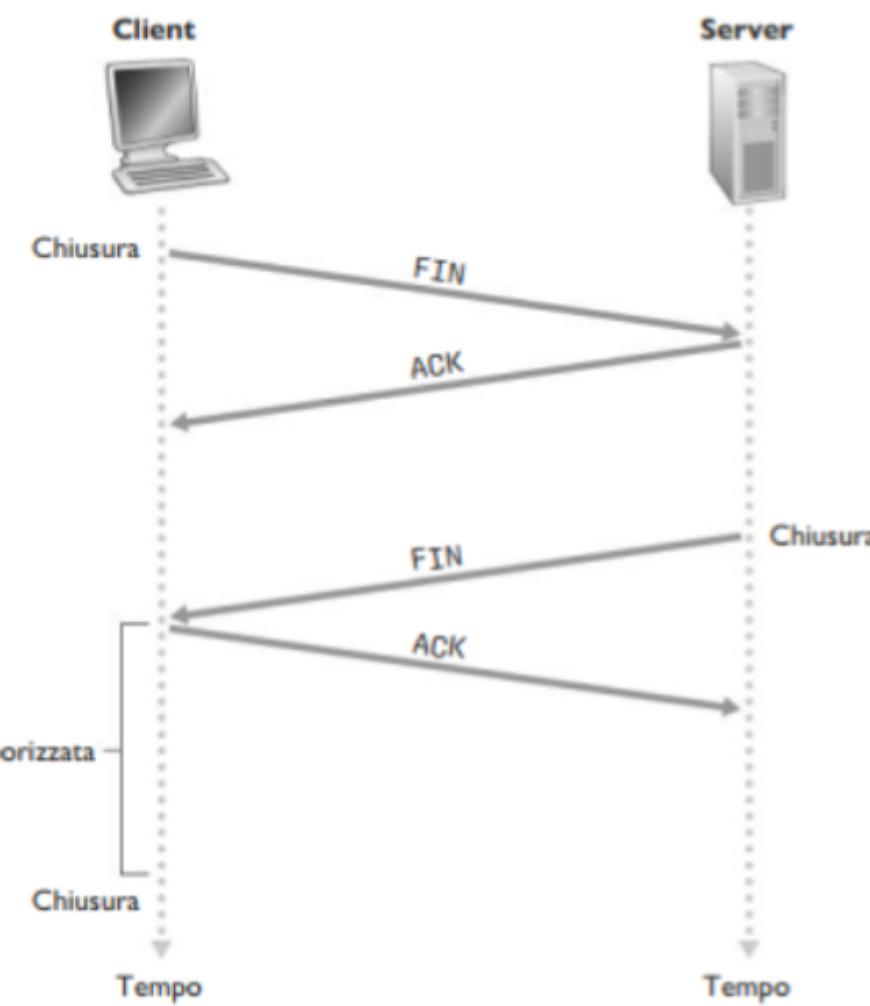
Figura 3.39 Handshake a tre vie di TCP: scambio di segmenti.

4.2.5.2 Chiusura connessione

La chiusura di una connessione implica la dealocazione di tutte le relative risorse.

È analoga all'apertura solo che viene impostato il bit `FIN = 1` durante la comunicazione per definire questo tipo di operazione.

Figura 3.40 Chiusura di una connessione TCP.



4.2.5.3 Stati della connessione

Durante le connessioni si possono identificare varie fasi in cui il client ed il server si trovano.

Client:

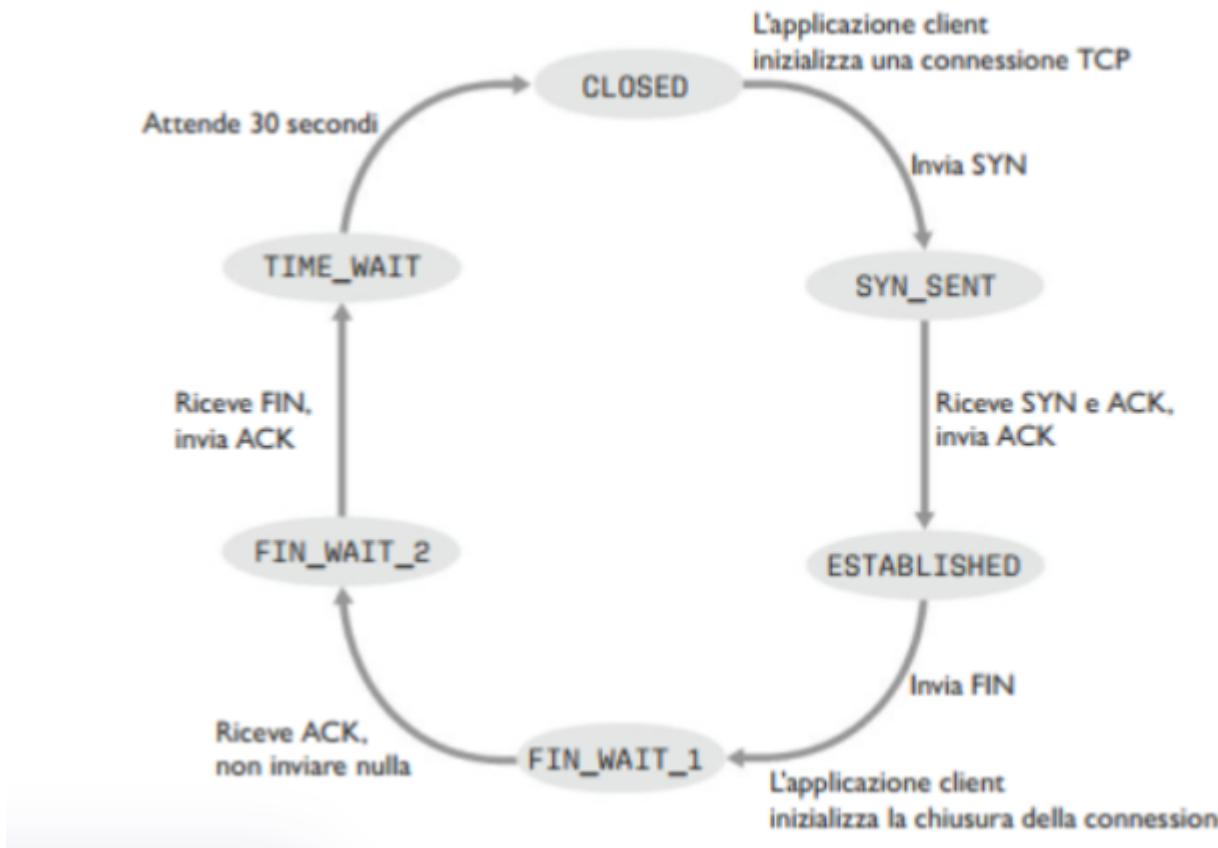
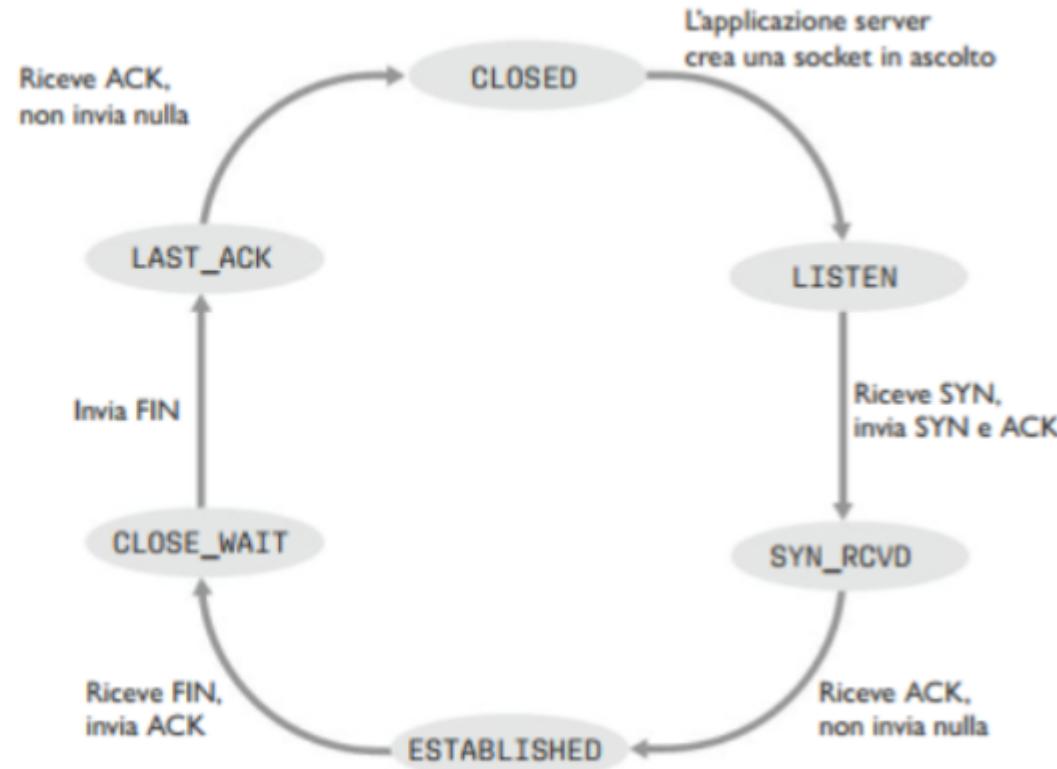


Figura 3.41 Tipica sequenza di stati visitati da un client TCP.

Server:

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



Se si richiede una connessione su una porta non attiva del server, questo risponderà con un segmento contenente il bit RST impostato ad 1.

4.2.5.4 caso d'uso: nmap

nmap è uno strumento che permette di analizzare le porte di un host.

Ad esempio voglio esaminare la porta TCP 6789 su un host bersaglio.

nmap manderà un TCP SYN con porta destinazione 6789, i risultati possibili sono i seguenti:

- **TCP SYNACK**

host sorgente riceve un TCP SYNACK dall'host bersaglio. Fa capire che è in esecuzione un applicazione su quella porta.

- **TCP RST**

host sorgente riceve un TCP RST, significa che il SYN ha raggiunto il bersaglio, ma su quest'ultimo non è in esecuzione alcuna applicazione.

Questo però in ambito di sicurezza ci informa che il segmento non è stato bloccato da alcun firewall

- **non riceve nulla**
possibile blocco da firewall

4.2.5.5 problemi di sicurezza: SYN flood

la tecnica di handshake prima citata, che prevede l'allocazione delle risorse in seguito alla ricezione del SYN, pone le basi un classico attacco Dos (Denial of service); in questo caso particolare prende il nome di SYN flood.

Se i server dopo un certo intervallo di tempo non ricevono ack relativi al terzo passo dell'handshake deallocano le risorse.

Questo attacco prevede che l'aggressore invii un gran numero di segmenti TCP SYN al server, il quale alloca risorse sino a satursi; non permettendo agli utenti legittimi di usufruire dei servizi.

Una possibile soluzione è l'uso dei SYN cookie.

- Quando al server arriva un TCP SYN, non sa se è legittimo. Crea quindi un numero di sequenza TCP, eseguendo una funzione hash sugli indirizzi ip e porta di sorgente e destinazione, ed una chiave segreta (può essere usata per più connessioni) . Questo numero è il cosiddetto "**cookie**". Il server quindi risponde con un SYNACK; l'aspetto importante è che il server non ha allocato nulla fino ad ora, neanche il cookie (non può causare danni).
- Quando arriva la risposta del client, il server applica la funzione hash sui campi del segmento arrivato. Se come nella norma il numero di sequenza (hash) + 1 corrisponde con l'ACK, allora il server crea una connessione aperta ed una socket.

4.2.6 controllo congestione

La congestione si basa sulla prevalenza nel canale di traffico duplicato rispetto ai dati effettivi.

4.2.7 controllo flusso vs congestione

a differenza della congestione, il **controllo di flusso**, si incentra nel ricevente, in particolare nel suo buffer.

si evita di far viaggiare eccessivi pacchetti se a destinazione non c'è lo spazio (nei buffer).

in TCP si usa la finestra [rwnd](#).

il problema quindi non risiede né nel mittente né nel client, in quanto per notificare questa quantità i pacchetti viaggiano (la rete non è collassata). inoltre si aggrava in casi di canali di comunicazione lunghi.

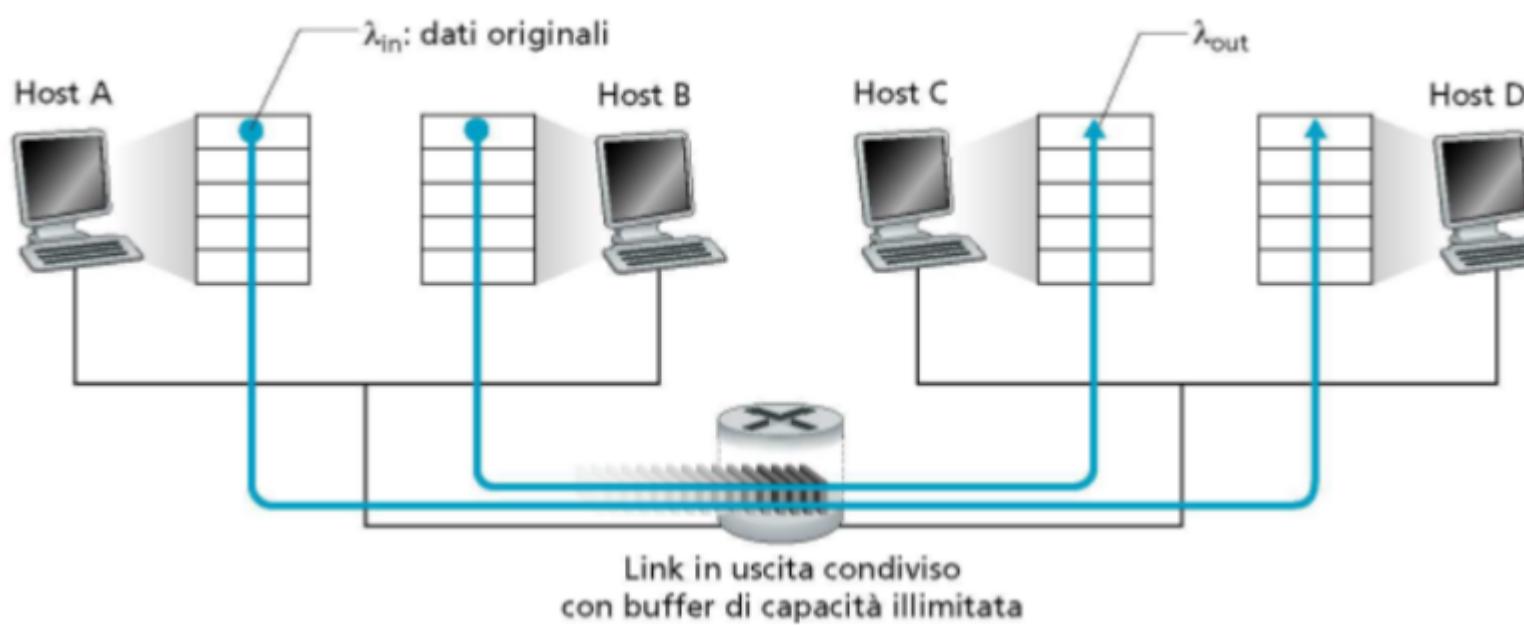
4.2.7.1 Scenari possibili

si possono verificare 3 scenari possibili

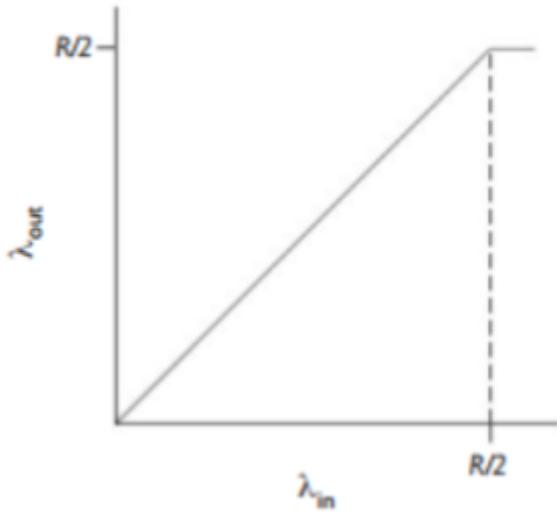
$$\lambda_{in} \text{ bytes} = \text{traffico generato}$$
$$\lambda_{out} \text{ bytes} = \text{traffico che arriva a destinazione}$$

4.2.7.1.1 due mittenti e un router con buffer limitati

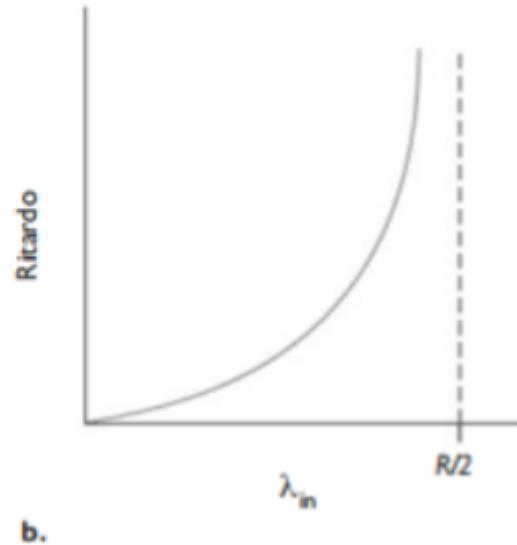
Due host (A e B) con una connessione che condivide un singolo router intermedio.



Data una capacità R del canale condiviso, si assume che il router intermedia abbia un buffer infinito che permette di non perdere pacchetti.



a.

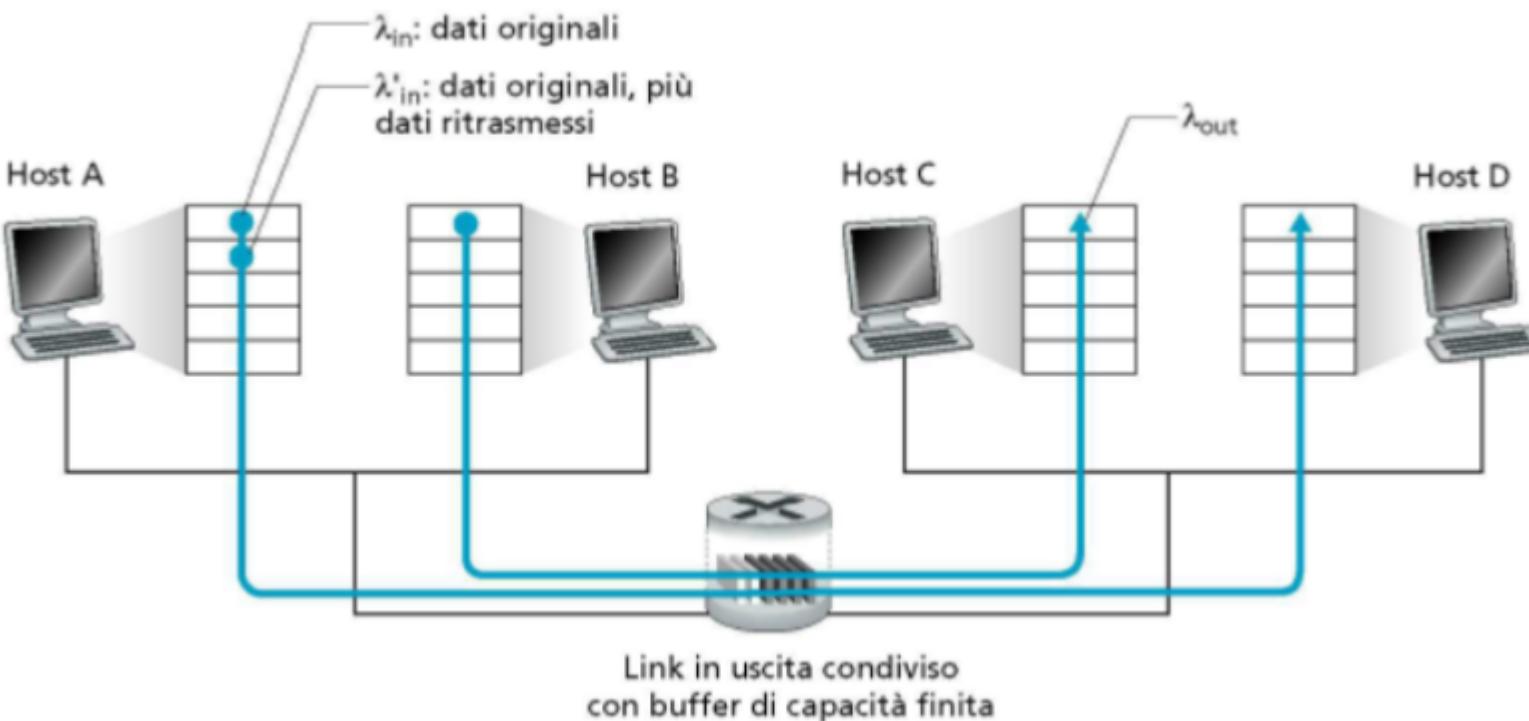


b.

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

Il grafico a sx mette in relazione in throughput per connessione () con il tasso di invio, si nota che il limite superiore sia $R/2$ dato che sta venendo condiviso per inviare e rispondere. Avere questo valore sembra un ottimo risultato, però guardando il grafico a destra si nota che aumentando il tasso di invio, aumenta esponenzialmente il ritardo causando un arrivo dei pacchetti dopo tempo infinito (non si perdono per il buffer illimitato)

4.2.7.1.2 due mittenti e un router con buffer limitato



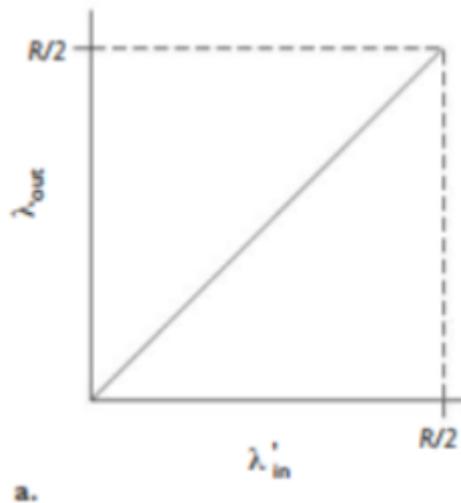
Nel pratico non può esistere un buffer illimitato, è necessario quindi tenere conto delle perdite e delle relative ritrasmissioni.

Innanzitutto bisogna prestare attenzione al termine "tasso di trasmissione". Quest'ultimo indicherà il tasso di trasmissione verso la socket (λ_{in} bytes); **carico offerto** indicherà invece il tasso di trasmissione dei segmenti, che quindi potranno essere ritrasmessi per fornire il servizio di trasferimento affidabile (λ'_{in} bytes).

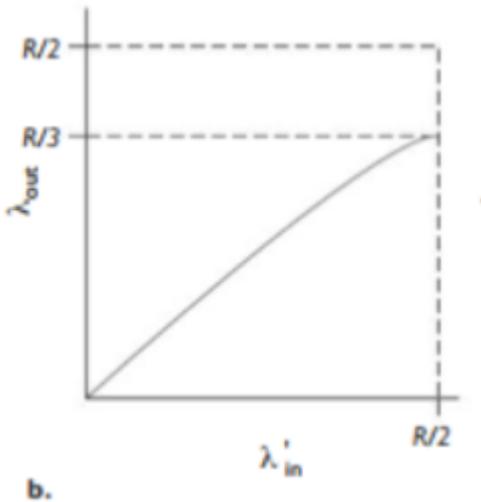
Le prestazioni dipendono da come si effettua la ritrasmissione, possiamo trovarci in 3 casi:

- 1. Host A è in grado di determinare la capacità del buffer:** verrà trasmesso un pacchetto solo quando il buffer è libero.

$$\lambda_{in} = \lambda'_{in}$$



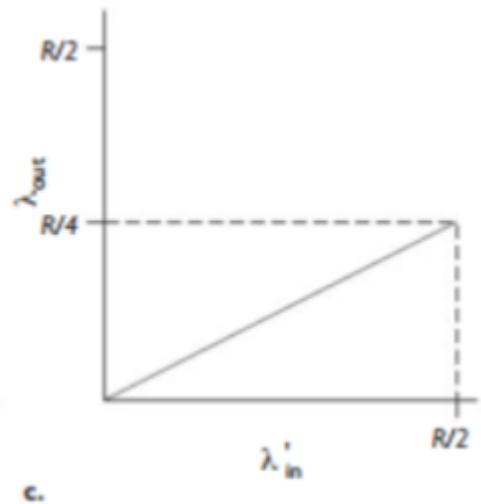
2. **Ritrasmissione solo con certezza di pacchetto perduto:** un po' più realista ma un po' forzata, si potrebbe implementare con un timeout alto.



Si nota del grafico che avendo un carico offerto (λ'_in) che tende a $R/2$ si avrà un λ_{out} di $R/3$. Ovvero su 0,5 unità di dati trasmessi, in media solo 0,333 sono originali ed il resto ritrasmissioni; causate appunto dall'overflow dei buffer limitati.

3. **Ritrasmissione non necessaria:** mittente va in timeout prematuramente e ritrasmette un pacchetto non perduto (ex: subito ritardi in coda).

Sarà quindi lavoro inutile per il router che trasmetterà un pacchetto che verrà scartato, utilizzando di conseguenza banda utile.

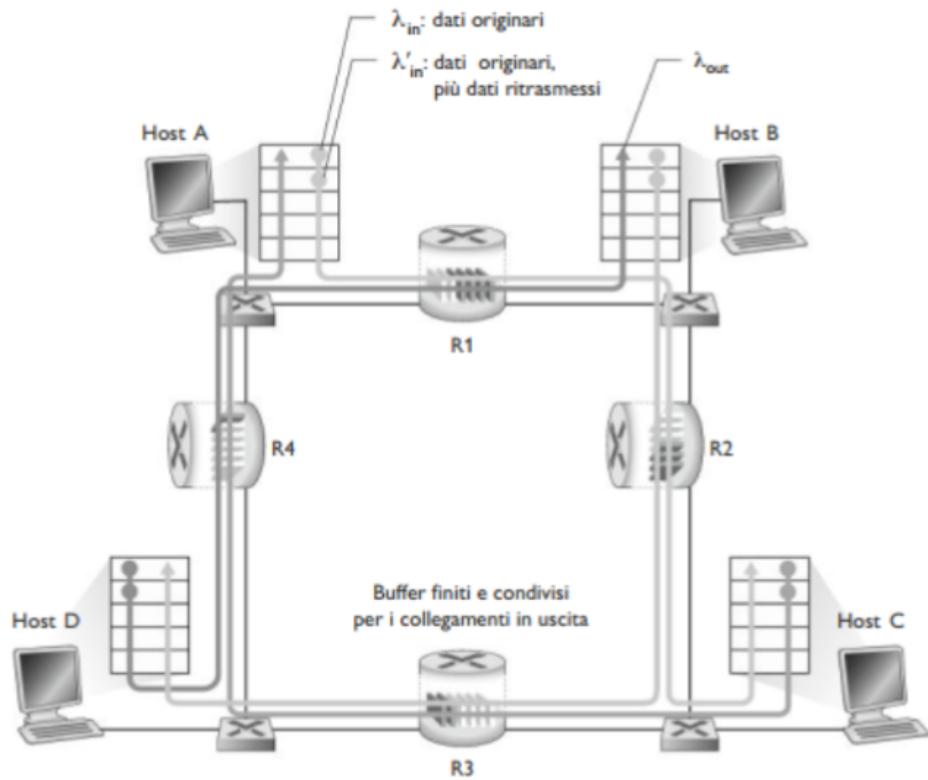


in questo caso di intradamento duplice del pacchetto, possiamo notare come λ_{out} tenda a $R/4$.

4.2.7.1.3 Quattro mittenti, router buffer finiti e percorsi con più collegamenti

il peggiore.

4 Host trasmettono pacchetti su collegamenti sovrapposti tra loro; implementano inoltre meccanismi di timeout e ritrasmissione per il trasferimento dati affidabile.



In questo caso per arrivare ad host opposti (ex: A-C) dovrò inevitabilmente passare per un router intermedio.

Per **bassi valori** di λ_{in} gli overflow sono rari, quindi con un incremento di λ_{in} avremo un incremento di λ_{out} .

La situazione cambia per **valori estremamente grandi** di λ_{in} (e quindi λ'_{in}); potrebbe capitare infatti che il throughput D-B sia estremamente più grande di quello A-C, causando quindi nel tempo una saturazione del buffer che predilige il traffico B-D (annullando A-C).

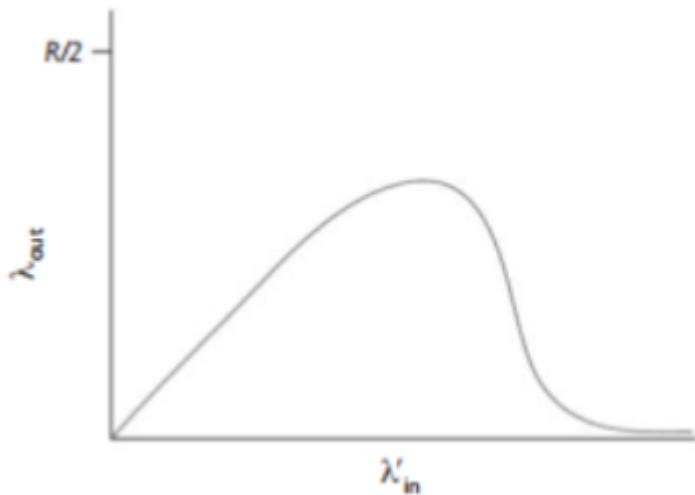


Figura 3.48 Prestazioni dello Scenario 3 (buffer di dimensione finita e percorsi multihop).

Il fulcro del problema si basa sulla pessima gestione dei router rispetto all'instradamento dei pacchetti. una possibile soluzione potrebbe essere che il router dia più priorità ad un pacchetto che ha già superato un certo numero di router ("hop").

4.2.7.2 approcci pratici

- **end-to-end**

il livello di rete non fornisce supporto al livello di trasporto. il controllo viene effettuato sui sistemi periferici osservando il comportamento della rete. La perdita di segmenti (timeout, ack duplicati) TCP ad esempio viene considerata come congestione, TCP opera di conseguenza diminuendo la dimensione della finestra.

- **assistito dalla rete**

i componenti a livello di rete forniscono un feedback al mittente sullo stato della rete. Ad esempio inviando un bit che indica la presenza di traffico. È possibile implementare un feedback più sofisticato come il controllo di **congestione ATM ABR** (available bit rate) che consente ad un router di informare sulla propria frequenza trasmisiva che può supportare sul canale uscente. TCP lo usa in certi casi associato ad end-to-end.

ci sono due modi per inviare feedback assistiti dalla rete:

1. **chockepacket**: router invia al mittente un pacchetto "di strozzatura", informando che è congestionato
2. imposta un campo all'interno del pacchetto che sta fluendo verso il destinatario, questo alla ricezione informerà il mittente. in questo caso è necessario almeno un RTT (cwr, ece). si fa impostando il campo TOS nel datagramma IP

4.2.7.3 congestione secondo TCP

Si impone al mittente un limite al tasso di invio sulla propria connessione in funzione della congestione di rete percepita.

Se c'è scarso traffico incrementa il tasso e viceversa.

Oltre alle variabili "LastByteRead" e "rwnd" si tiene traccia agli estremi di un var aggiuntiva "finestra di congestione" (cwnd).

Si impone che la quantità di dati non confermati di ack non sia maggiore tra il minimo dei valori di cwnd e rwnd

```
LastByteSent - LastByteAcked <= min{cwnd, rwnd}
```

TCP, in base alla frequenza di arrivo degli ACK (sintomo che non ci sono problemi in rete), incrementa / diminuisce la grandezza delle finestre.

Proprio per questo si dice **auto-temporizzato**.

Determinato questo meccanismo, sorge però il problema che se tutti gli host nello stesso momento trasmettono troppo velocemente viene causato un collasso; nel caso opposto (lentamente) si causerebbe un sottoutilizzo.

Come fa quindi TCP a capire la velocità di trasmissione ma allo stesso tempo utilizzare in maniera ottima la banda?

Ci si basa sui seguenti criteri:

- **segmento perso implica congestione**: quando un segmento del mittente viene perso, i tassi di trasmissione di questo vengono decrementati
- **aumentare tasso di trasmissione in seguito ad ACK**: un ACK (non duplicato) viene interpretato come un segnale che in rete non ci siano problemi, i segmenti vengono consegnati con successo, quindi la finestra di congestione può essere incrementata.

4.2.7.4 l'algoritmo di congestione TCP

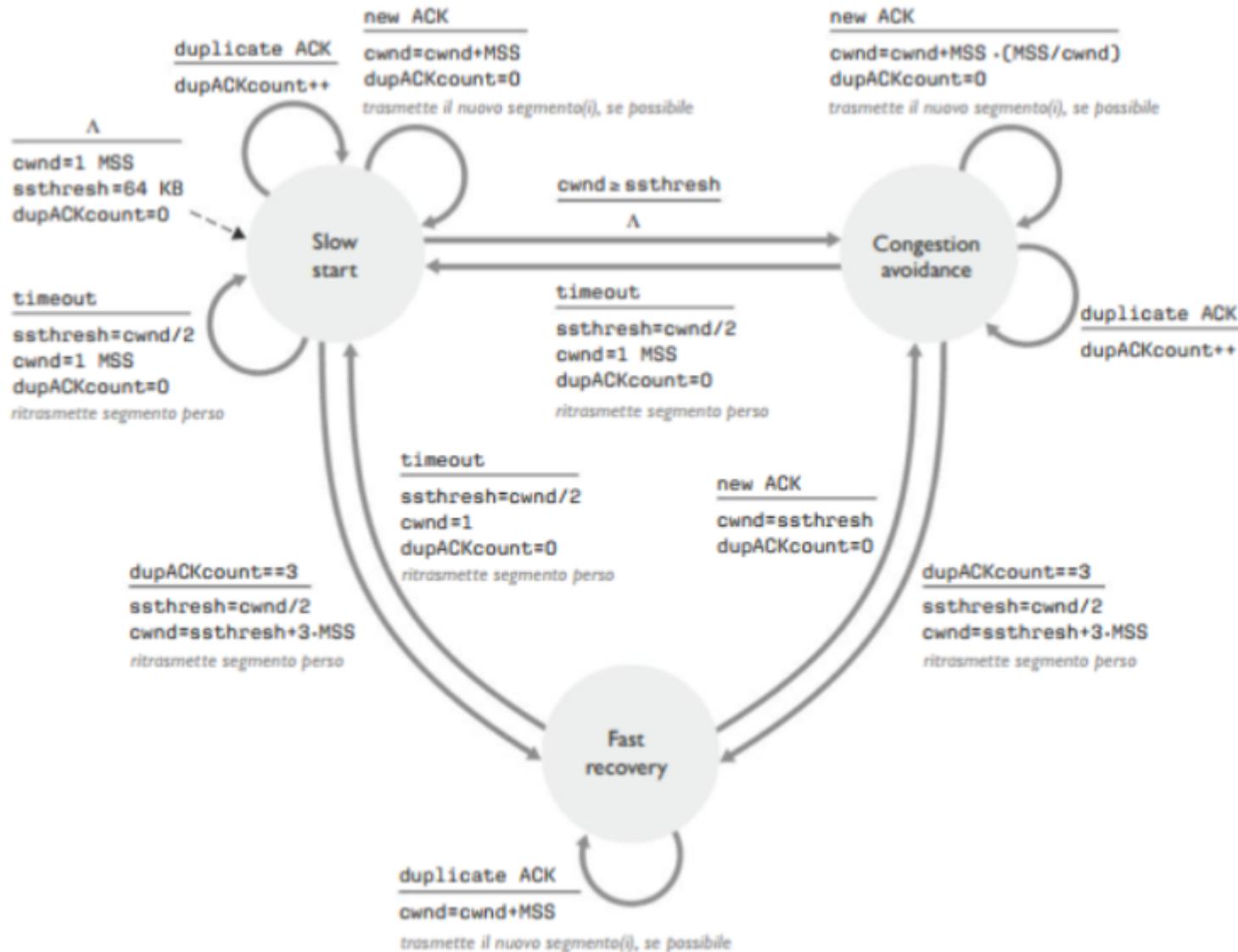


Figura 3.51 Descrizione tramite automa a stati finiti del controllo di congestione TCP.

si basa su 3 fasi principali:

1. Slow start

Si imposta il cwnd ad 1 MSS, avremo di conseguenza una velocità iniziale di MSS/RTT.

MSS = 500byte

RTT = 200ms

velocità iniziale = 20kbps

Ad ogni arrivo di ACK (prima che si verifichi timeout, duplicati) il valore di cwnd aumenta di 1 MSS ed invia il doppio dei segmenti mandati nella volta precedente.

Questo provoca un raddoppiamento della velocità di trasmissione ad ogni RTT, che parte lentamente ma cresce esponenzialmente.

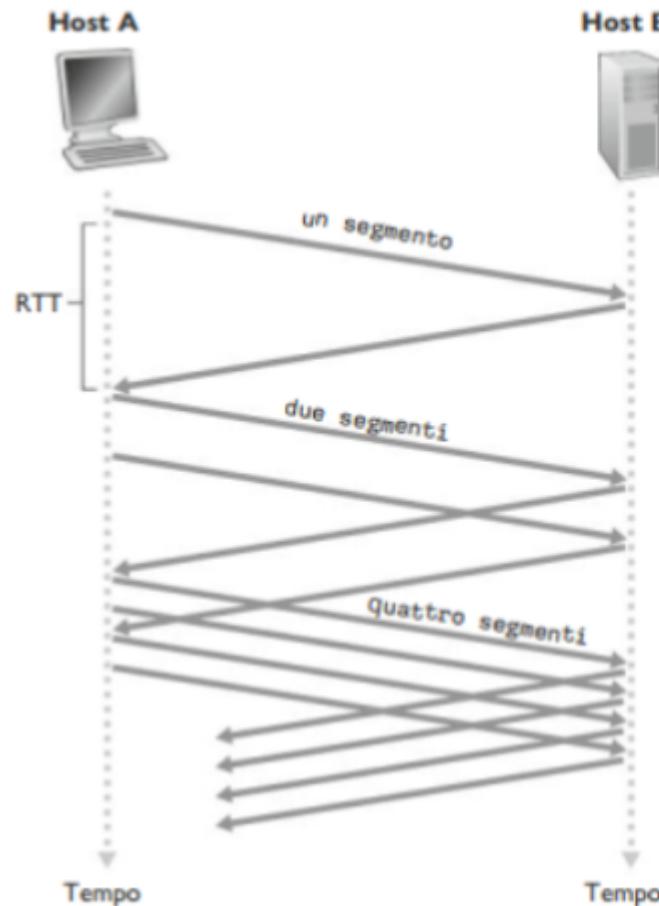


Figura 3.50 Slow start
di TCP.

Questa fase può terminare in 3 modi:

1. si verifica un fenomeno di **perdita**: cwnd torna ad 1 MSS e riparte il processo di incremento. Inoltre in questo momento viene impostata la variabile $ssthresh = cwnd/2$ (slow start trashhold) (con cwnd di quando è stata rilevata la perdita).

2. Quando cwnd ritorna al valore sstresh (dopo che si è verificata la prima perdita), entra in modalità **congestion avoidance**.
3. Quando vengono rilevati 3 ack duplicati, opera una ritrasmissione rapida ed entra nello stato di **fast recovery**.

2. Congestion avoidance

Quando siamo in questa fase si cerca di avere un approccio più conservativo.

Invece di raddoppiare cwnd, si aumenta di 1MSS ogni RTT. Un approccio usato è incrementare cwnd come segue ad ogni ack ricevuto:

ex: vengono inviati cwnd/mss segmenti quindi 10 segmenti. gli ack relativi incrementeranno di 1/10 cwnd, quindi solo quando arriveranno tutti e 10 avremo un incremento di 1 MSS.

Quando si verifica un timeout (perdita) imposta le variabili come fa slow start.

Nel caso di perdita rilevata da ack duplicati (la risposta è meno drastica in quanto la rete comunque non è collassata ed i pacchetti arrivano) imposta così le variabili:

```
-- il 3 comprende gli ack duplicati  
cwnd = (cwnd / 2) + 3  
  
-- con cwnd al tempo dell'arrivo dei 3 ack duplicati  
sstresh = cwnd / 2
```

ed infine entra nello stato di fast recovery.

3. Fast recovery

In questa fase per ogni ack duplicato il valore di cwnd viene incrementato di 1 MSS.

Se si verifica un timeout passa allo stato di slow start.

Se arriva un ack passa a congestion avoidance.

il suo uso è opzionale.

4.2.7.5 TCP tahoe e reno

Sono due versioni di TCP

- Tahoe: implementa slow start e congestion avoidance. andando in fase di congestione ad ogni evento di perdita.
- Reno: slow start, congestion e fast retrasmitt

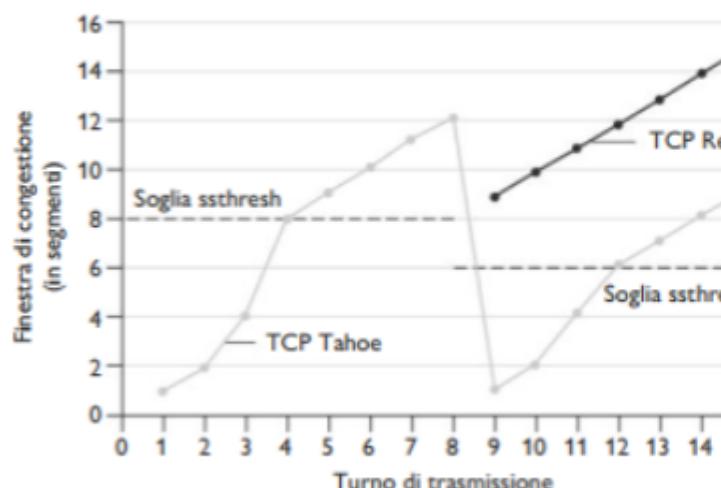


Figura 3.52 Evoluzione della finestra di congestione TCP (Tahoe e Reno).

4.2.7.6 riepilogo

Il controllo sulla congestione si basa quindi su un incremento lineare pari ad 1 MSS per RTT (slow start) e di un decremento moltiplicativo (congestione avoidance).

Per questa ragione il controllo di congestione è detto **incremento additivo, decremento moltiplicativo** (AIMD), produce quindi il seguente effetto a

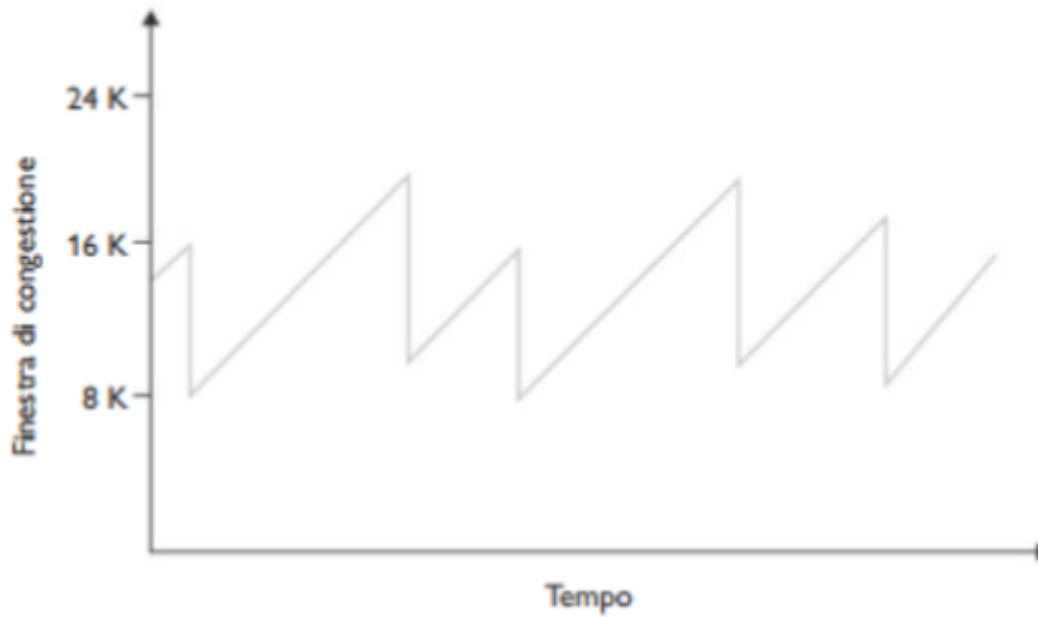
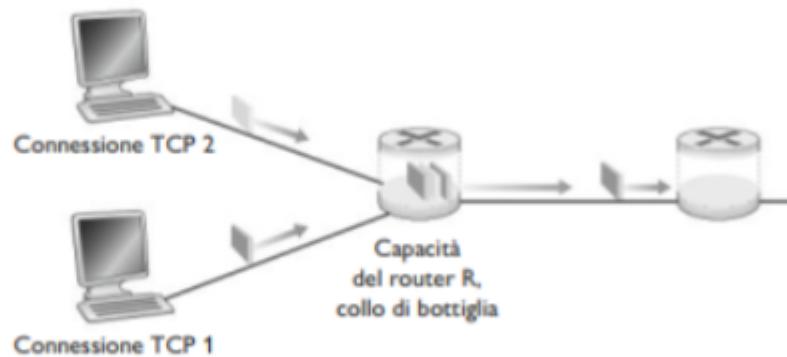


Figura 3.53 Controllo di congestione a incremento additivo e decremento moltiplicativo.

4.2.8 fairness

Consideriamo il caso semplice di 2 connessione che condividono un collegamento con capacità trasmissiva R

Figura 3.56 Due connessioni TCP che condividono un singolo collegamento che fa da collo di bottiglia.



TCP è un algoritmo fair?

è importante da mantenere per evitare che un singolo client sia capace di saturare la banda.

Disponendo sugli assi 2 throughput.

se la banda è divisa equamente, ci aspettiamo che il throughput cada essattamente sulla bisettrice del primo quadrante. Risultando in una somma uguale ad R.

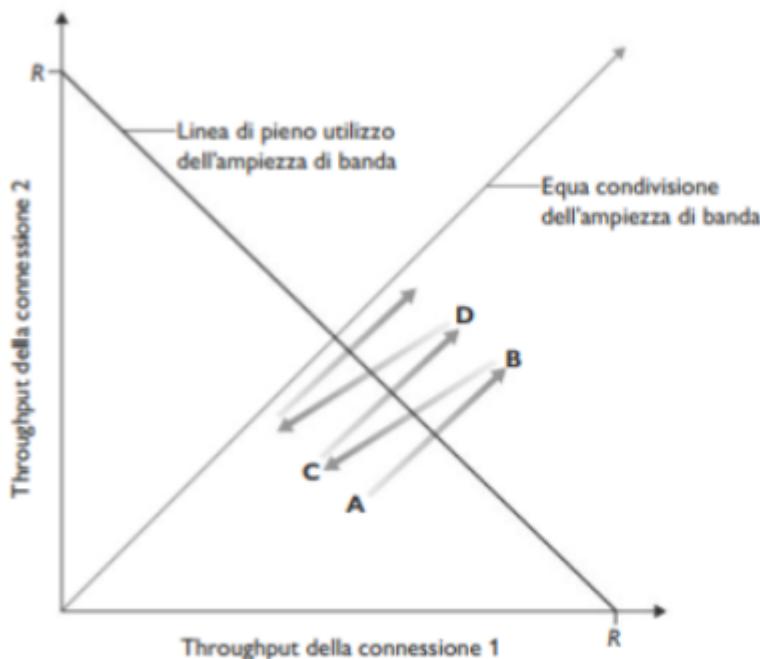
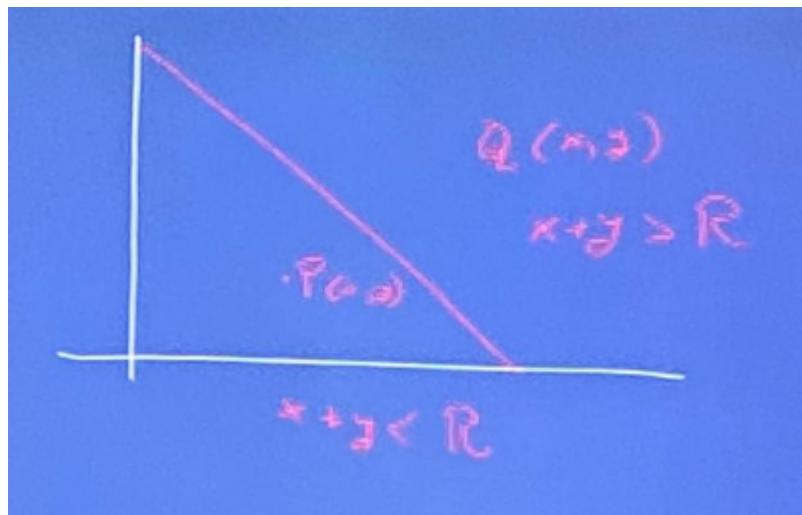


Figura 3.57
Throughput delle
connessioni TCP 1 e 2.

possiamo identificare $(R, 0)$ come il punto in cui la banda è satura solo dalla connessione 1.

$(0, R)$ viceversa. la linea sottesa tra questi due punti identifica la capacità del canale



notiamo quindi:

- tutti i punti al di sotto la linea occuperanno una banda minore di R
- al di sopra che la saturano, popolando di conseguenza i buffer

Partendo dal punto $Q(x'', y'')$, seguendo AIMD il flusso verrà dimezzato, arrivando al punto $P(x', y')$. avremo quindi:

$$x' = \frac{x''}{2} \quad y' = \frac{y''}{2}$$

studiano la retta passante tra questi due punti:

$$\begin{cases} y' = mx' + n \\ y'' = mx'' + n \end{cases}$$

scrivo tutto sotto forma di y''

$$\begin{cases} \frac{y''}{2} = m \frac{x''}{2} + n \\ y'' = mx'' + n \end{cases}$$

moltiplico per due la equazione

$$\begin{cases} y'' = mx'' + 2n \\ y'' = mx'' + n \end{cases}$$

semplifico le equazioni

$$0 = 0 + 2n - n \implies n = 0$$

se $n = 0$ vuol dire che la retta passa per l'origine.

seguendo l'andamento a zigzag di aimd tenderò alla bisettrice, ovvero dove si verifica il fairness.

questa dimostra è limitata al fatto che suppongo

- abbiano stesso rtt
- mss uguali
- perdite nello stesso istante

btw funziona.

4.2.8.1 in parola povere

Supponendo che ad un certo punto le dimensioni della finestra TCP siano tali che il throughput corrisponda al punto A.

Dato che la porzione di banda è minore di R, non si verificano perdite e l'algoritmo procede ad aumentare la finestra di 1MSS per ogni RTT.

Di conseguenza il throughput congiunto procede lungo la semiretta a 45° uscente dal punto, portandosi nel punto B.

Se in questo punto si è superato R, si procede con il decremento (fattore 2) che porta nel punto C.

Si effettua quindi l'incremento analogamente rispetto al punto A e così via.

Si può notare che andando avanti si può notare che la banda utilizzata fluttuerà sulla equa condivisione.

Si è notato infatti che gli host con RTT più bassi tendono ad accaparrarsi più banda, portando ad uno scompenso.

4.2.9 QUIC (Quick UDP Internet Connection)

Ci possono essere situazioni in cui i servizi di UDP o TCP non sono completamente adatti ad una particolare app.

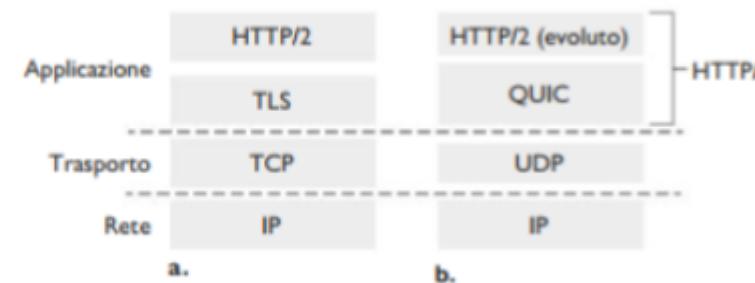
Ad esempio si può pensare di prendere i servizi offerti da UDP ed eliminare tutte le limitazioni sul flusso di TCP.

Questo approccio è usato da QUIC.

Protocollo a livello applicazione usato per migliorare le prestazioni a livello di trasporto per http sicuro.

Circa il 7% del traffico internet odieno si basa su QUIC.

Figura 3.58 Stack di protocolli di (a) tradizionale HTTP sicuro e (b) QUIC sicuro basato su HTTP/3.



Dotato delle seguenti caratteristiche:

- **orientato connessione e sicuro**

handshake, cifratura pacchetti. più rapido di un approccio standard in quanto normalmente si dovrebbe stabilire una connessione TCP e poi TSL (sopra TCP)

- **flussi (streams)**

astrazione di consegna bidirezionale dei dati, affidabile. QUIC consente il multiplexing di diversi flussi a livello applicazione all'interno di una singola connessione. in http/3 ad esempio per ogni oggetto di una pagina è destinato un flusso. I dati di più flusso possono essere contenuti all'interno di un singolo segmento (porta ad un HOL solo rispetto ai flussi inglobati in questo segmento)

- **trasferimento affidabile**

fornita rispetto ad ogni flusso

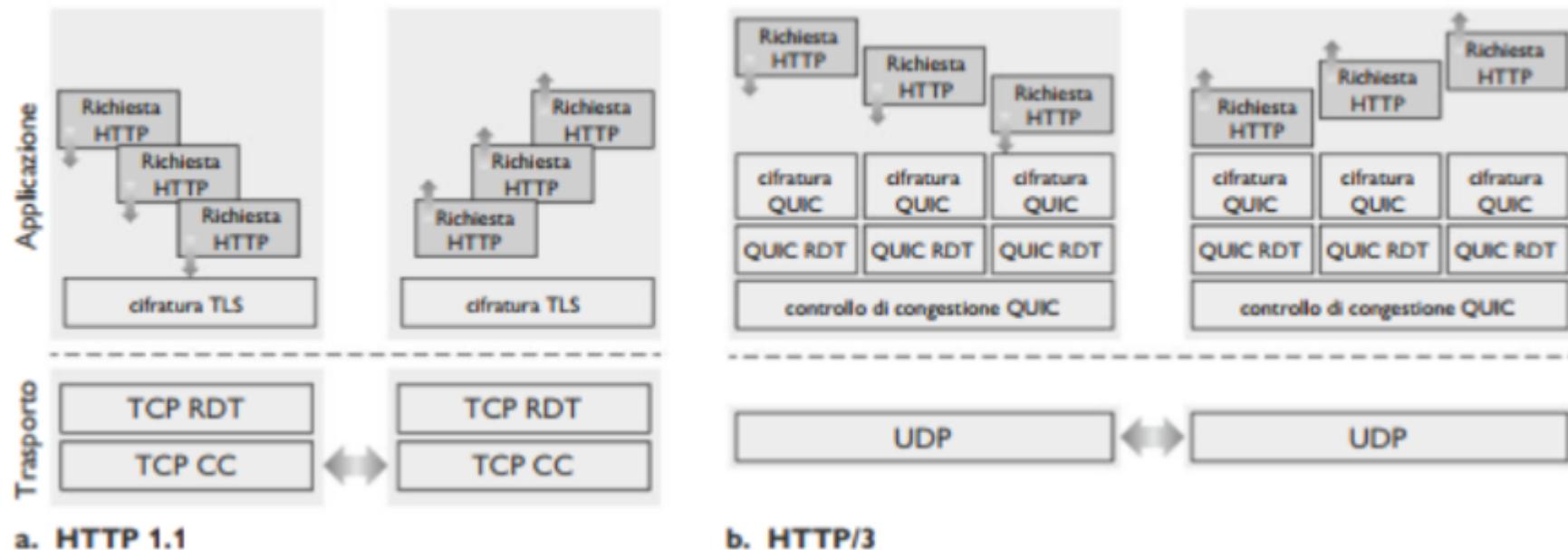


Figura 3.59 (a) singola connessione client-server che utilizza la cifratura TLS sul trasferimento dati affidabile (RDT) e il controllo di congestione TCP. (b) connessione client-server con flussi multipli che utilizza la cifratura, il trasferimento dati affidabile e il controllo di congestione di QUIC sopra il servizio non affidabile di UDP.

- **controllo congestione**

si basa su TCP NewReno

1 Livello Rete

I compiti principali del livello di rete sono:

- **forwarding** (inoltro)
trasferimento del pacchetto sull'appropriato canale di uscita. quando arriva all'interno del router si determina quale interfaccia di uscita usare.
generalmente implementata via hardware, in quanto avviene in una scala temporale nell'ordine di nanosecondi.
- **routing** (instradamento)
determinare i percorsi che i pacchetti devono eseguire per arrivare a destinazione. implementato via software tramite particolari algoritmi.

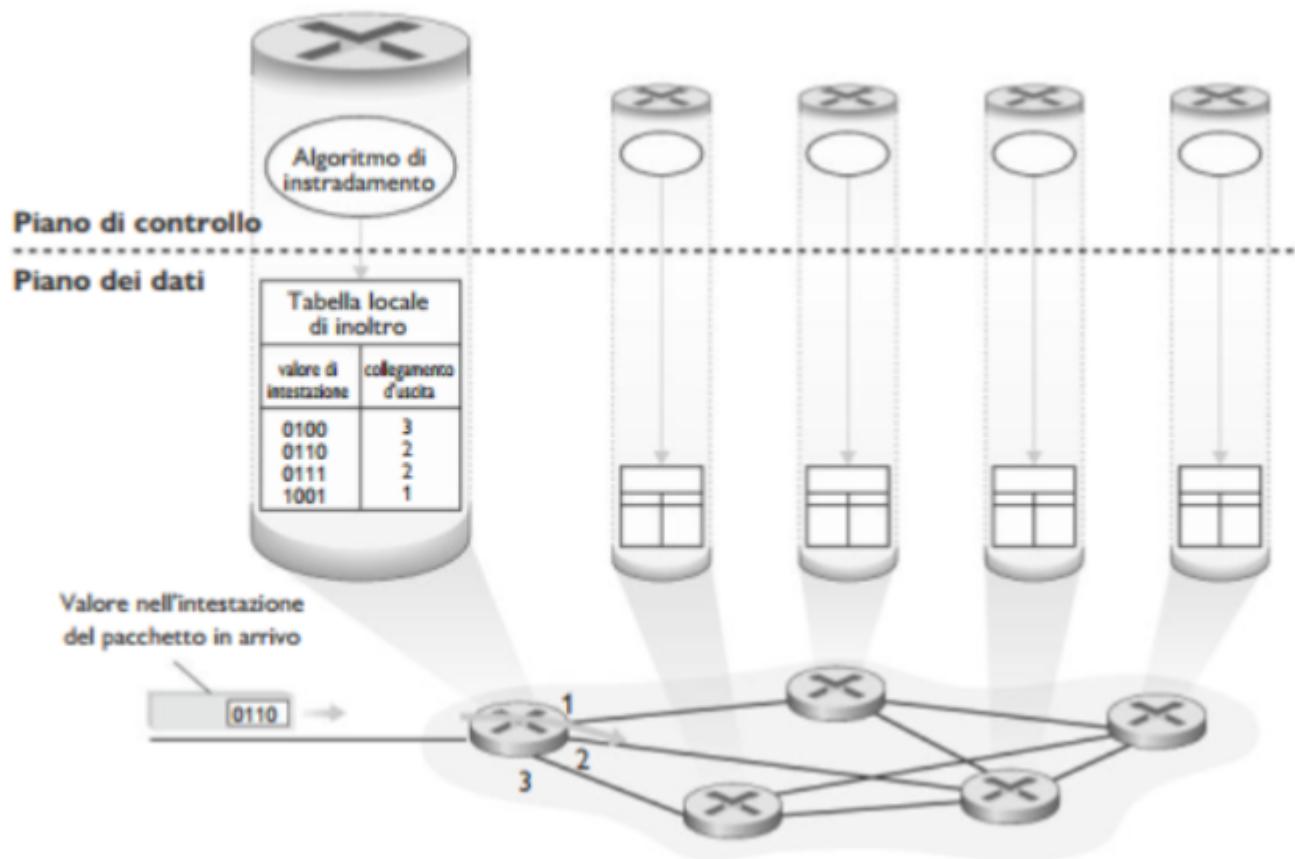
```
ip route
```

Per effettuare il forwarding viene fatto uso di particolari tabelle dette "**tabella di forwarding**".

Ci sono due moarpeggiodi in cui possono essere implementate:

1. **ogni router determina la propria**: i router comunicheranno tra loro per determinarle. svolgono quindi anche la funzione di routing.

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



2. **Software defined networking (SDN)**: un controller remoto (data center con alta affidabilità, ex: ISP o terzi) che calcole e distribuisce le tabelle ai vari router. il dispositivo quindi effettuerà solo il forwarding.

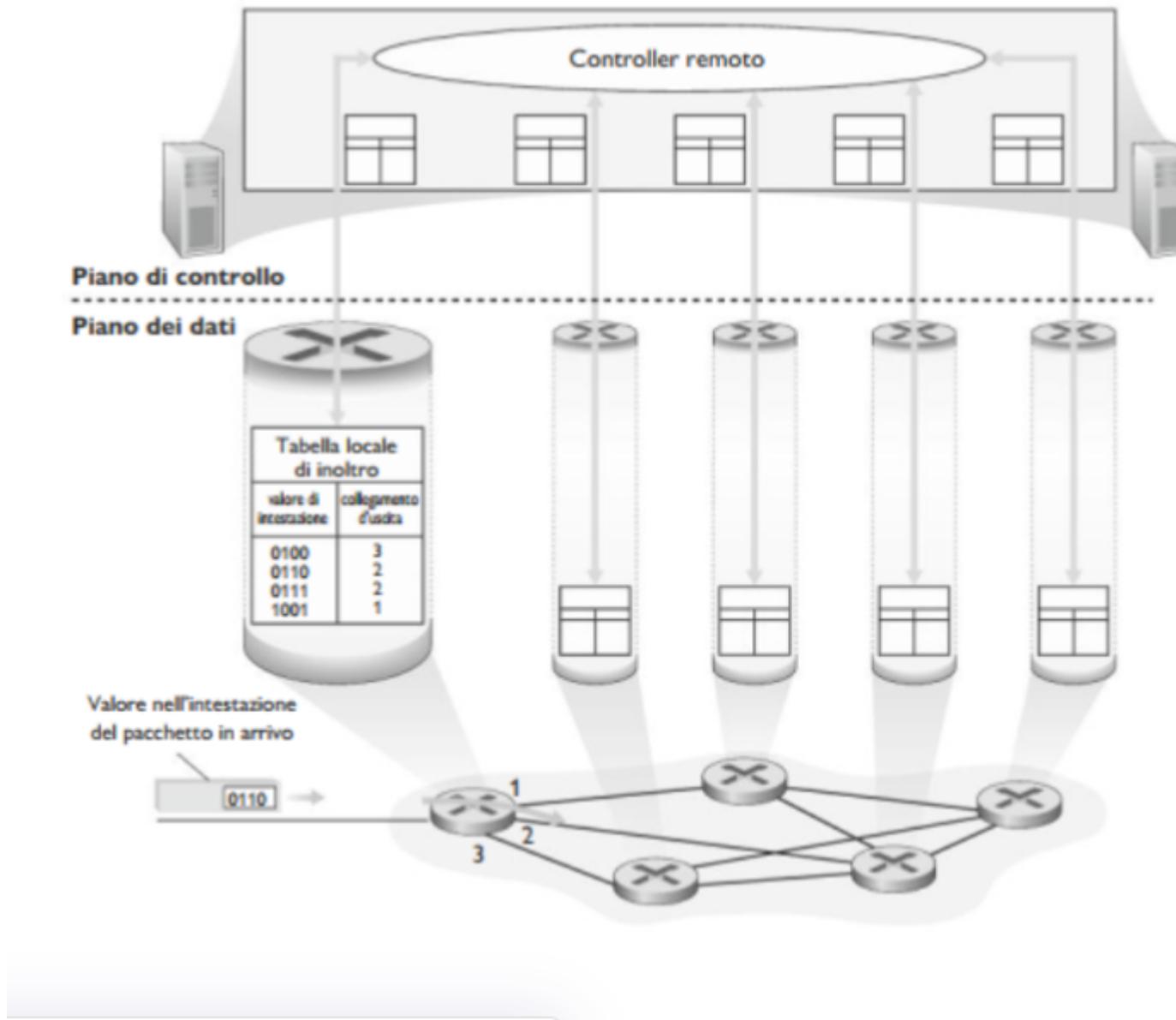


Figura 4.3
Un controller remoto determina e distribuisce i valori delle tabelle di inoltro.

1.1 servizi offerti

Adesso sorgono queste domande sul livello di rete:

- il trasferimento è affidabile?
- arrivano in ordine?

- controlla la congestione?

Tutto dipende dal modello di servizio del livello di rete. Potrebbe offrire:

- consegna garantita: assicura che prima o poi arriva
- consegna garantita con ritardo limitato
- consegna ordinata: arrivo con ordine di invio
- **banda minima garantita**: simula il comportamento di 1 collegamento trasmissivo con un bit rate specifico, nonostante il percorso effettivo può attraversarne molteplici
- sicurezza: cifratura sorgente e viceversa.

Internet offre un servizio best-effort. Non garantisce nulla dei servizi sopra citati.

Ad esempio esiste l'architettura ATM che offre servizi: ordine, ritardo limitato, banda garantita.

1.2 Termini usati nel capitolo

- forwarding = switching
- **commutatore di pacchetto / packet switch**: dispositivo generico che si occupa di trasferire un pacchetto da interfaccia in entrata ad uscita in base alla sua intestazione
- commutatori a livello di collegamento: stabiliscono in base al valore del campo del frame del frame a live. collegamento (si vede più avanti)
- router: commutatori in base al campo di rete

1.3 datagram vs circuiti virtuali

Io sai.

2 Chi c'è dentro il router?

si possono individuare 4 componenti principali:

- **Porte di ingresso**: svolgono diverse funzioni:

1. collegamento fisico. fornisce il punto di connessione fisica dove un cavo di rete si collega al router
2. livello di collegamento. sono in grado di interpretare e gestire le informazioni che arrivano attraverso il collegamento di rete in modo che possano essere trasferite correttamente attraverso il router e inviate al loro destino finale.
3. ricerca. in modo che il pacchetto che arriva alla struttura di commutazione arrivi nell'uscita corretta

- **struttura di commutazione:**

switching fabric.

collega le porte di ingresso a quelle in uscita.

- **porte di uscita:**

memorizzano i pacchetti provenienti dalla struttura di commutazione e li trasmettono sul canale in uscita (operazioni di liv. collegamento e fisico).

nei collegamenti bidirezionali in genere la porta di uscita è collegata a quella di ingresso del collegamento (line card, effettivamente la porta fisica - il pezzo di metallo a cui attacchi il cavo santo d'dio).

- **processore di controllo:**

router tradizionali esegue i protocolli e gestisce le tabelle, in quelli SDN è responsabile della comunicazione con i controller remoti

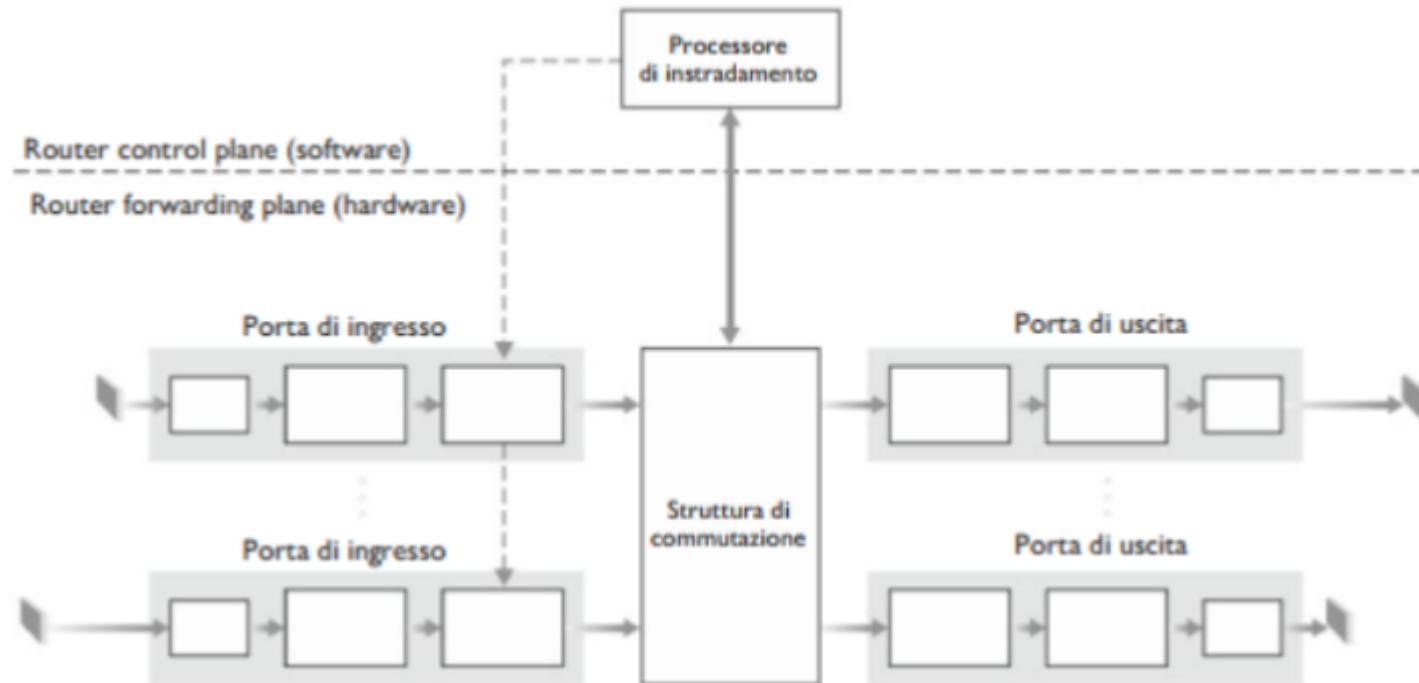


Figura 4.4 Architettura di un router.

Come si nota dalla figura, il cosiddetto "piano dei dati" è implementato via hardware, in quanto generalmente la velocità dei collegamenti è elevata. Si lavora in genere con ordini di ns, cosa impossibile per un sw su un hw tradizionale.

Il "piano di controllo" (processore di instradamento) è invece sw dato che si parla di millisecondi o secondi.

l'inoltro si può basare su diversi fattori:

- in base sulla destinazione
- generalizzato: ex: in base al mittente (gruppi di pacchetti indirizzati verso una particolare uscita), ecc... (si vede più avanti)

2.1 elaborazione porte di ingresso

come detto le porte di ingresso svolgono un ruolo centrale, è qui che viene determinata la porta di uscita verso cui dirigere il pacchetto tramite la struttura di commutazione.

La determinazione avviene con la tabella di inoltro, la quale viene copiata in ogni porta di ingresso; in particolare viene copiata sulla line card dal processore di instradamento tramite un bus separato.

Si considera ora il forwarding basato sulla destinazione (con indirizzi IPV4).

Un tabella di inoltro classica è basata sul disporre varie righe, per ogni riga si associa un indirizzo alla porta di uscita. Avere 32mln di righe è impossibile quindi si prediligono range di indirizzi.

Ad esempio, supponendo di avere 3 uscite, vogliamo avere questi inoltri:

Intervallo degli indirizzi di destinazione					Interfaccia
da	11001000	00010111	00010000	00000000	0
a	11001000	00010111	00010111	11111111	
da	11001000	00010111	00011000	00000000	1
a	11001000	00010111	00011000	11111111	
da	11001000	00010111	00011001	00000000	2
a	11001000	00010111	00011111	11111111	
altrimenti					3

si può semplificare come segue:

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

Il router provvede ad associare un prefisso dell'indirizzo di destinazione con una riga della tabella.

Se ci sono corrispondenze multiple si adotta la regola di corrispondenza a prefisso più lungo.

Concettualmente è semplice, però a tassi di trasmissione Gbps deve essere effettuata in ns. Non solo deve essere fatta hw ma si deve fare uso di particolari alg. che vadano oltre la ricerca lineare.

Inoltre è necessario porre attenzione ai tempi di accesso a queste memorie, vengono infatti spesso usate le **Ternary content address memories** (TCAM) che restituiscono in tempo costante la corrispondenza di un IP.

Una volta determinata la porta di uscita verrà passato alla struttura di commutazione. In alcune strutture può essere fermato prima di entrarci in quanto è utilizzato da altre porte (stile sistemi operativi), entra così in una coda per poi essere schedulato...

2.2 struttura di commutazione (switching)

Legenda:



Porta
di ingresso



Porta
di uscita

il cuore del router, attraverso la quale i pacchetti vengono commutati (inoltrati).

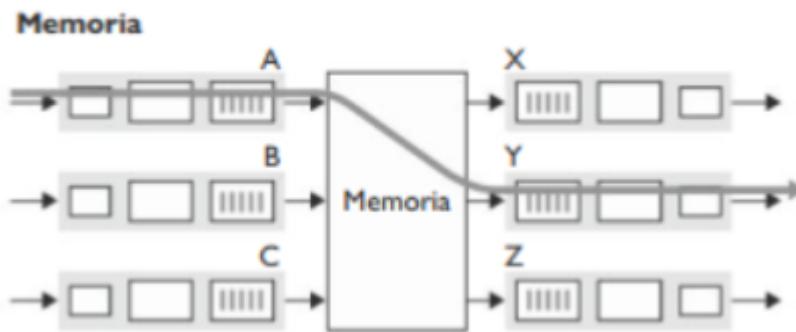
può essere implementata in vari modi:

- **commutazione in memoria**

i primi router erano calcolatori tradizionali che vedevano le porte di ingresso e uscita come dispositivi I/O.

Quando arriva un pacchetto, la porta lo segnala tramite interrupt e quindi lo copia nella memoria del processore di instradamento. Questo estra dall'intestazione indirizzo di destinazione, verifica la tabella di inoltro, copia nel buffer della porta di uscita corrispondente.

Notiamo però che due pacchetti con la stessa destinazione non partiranno mai simultaneamente.

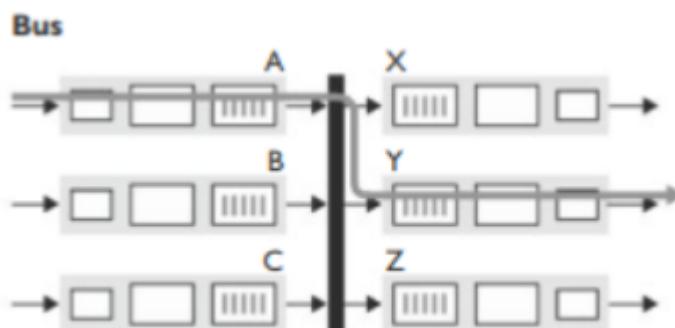


- **commutazione tramite bus**

le porte di ingresso trasferiscono il pacchetto direttamente alle porte di uscita tramite un bus.

ogni pacchetto viene "marchiato" con un identificativo (intestazione) di porta di uscita, per poi essere inviato a tutte le porte, solo quella con l'id relativo lo accetterà. Quest'ultima poi lo rimuoverà in quanto è al solo fine di uso interno.

l'invio è sequenziale, uno alla volta. Inoltre la larghezza di banda della commutazione è limitata alla grandezza del bus.



- **commutazione attraverso rete di interconnessione**

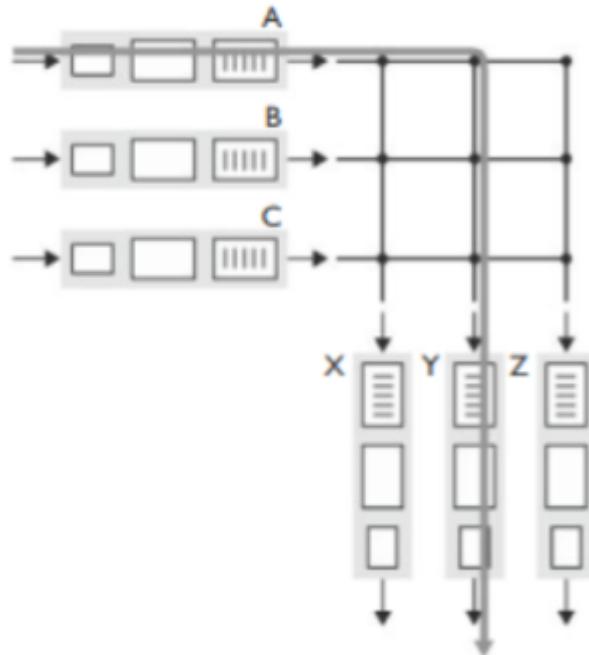
superà la limitazione di banda del bus singolo.

Si implementa una matrice di commutazione (crossbar switch), consiste in $2n$ bus che collegano n porte. Ogni bus verticale si interseca con quelli orizzontali, ogni punto di incrocio può essere aperto o chiuso a seconda dell'esigenza.

Ad esempio se un pacchetto deve andare da A->Y, il controller chiude l'incrocio e la porta ed invia il pacchetto. Allo stesso tempo il collegamento B->X rimane aperto.

Quindi con questo metodo possono inoltrare più pacchetti in parallelo. Si dice infatti che una matrice sia **non-blocking** se un pacchetto in uscita verso una porta non venga bloccato, a meno che non esiste un altro pacchetto diretto verso la stessa (porta).

Rete di interconnessione



2.3 elaborazione porte di uscita

permette di prendere i pacchetti dalla memoria della porta in uscita e li trasmette sul collegamento in uscita.

2.3.1 accodamenti

si possono formare quindi code in corrispondenza delle porte che possono causare la famosa perdita dei pacchetti (saturazione memoria del router).

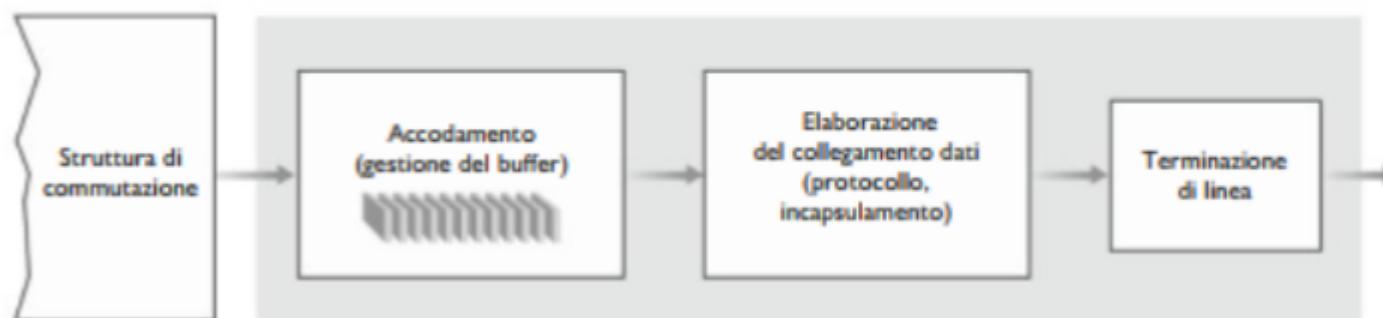


Figura 4.7 Elaborazione alle porte di uscita.

in base al tipo di porta possono avvenire diversi fenomeni.

Si suppone innanzitutto:

- R_{line} : tassi di trasmissione delle porte, al secondo.
- N porte
- pacchetti abbiano tutti la stessa lunghezza
- R_{switch} : tasso di trasferimento della struttura di commutazione. N volte più veloce di R_{line} in modo da soddisfare tutte le richieste parallele.

2.3.1.1 in ingresso

Si innesca il fenomeno di ****Head of line (HOL, blocco in testa)**.

Supponendo che:

- velocità trasmissive di tutti i collegamenti siano uguali
- i pacchetti siano trasferiti alle code di uscita con metodo FCFS (first come first serve)

sarà possibile trasferire più pacchetti in parallelo, ma su **porte differenti**. Se quindi due pacchetti in testa alle code di ingresso sono destinati alle stesse code di uscita, bloccheranno di conseguenza i pacchetti precedenti (nonostante siano diretti verso qualche altra porta).

Contesa della porta di uscita all'istante t
un solo pacchetto grigio scuro può essere trasferito

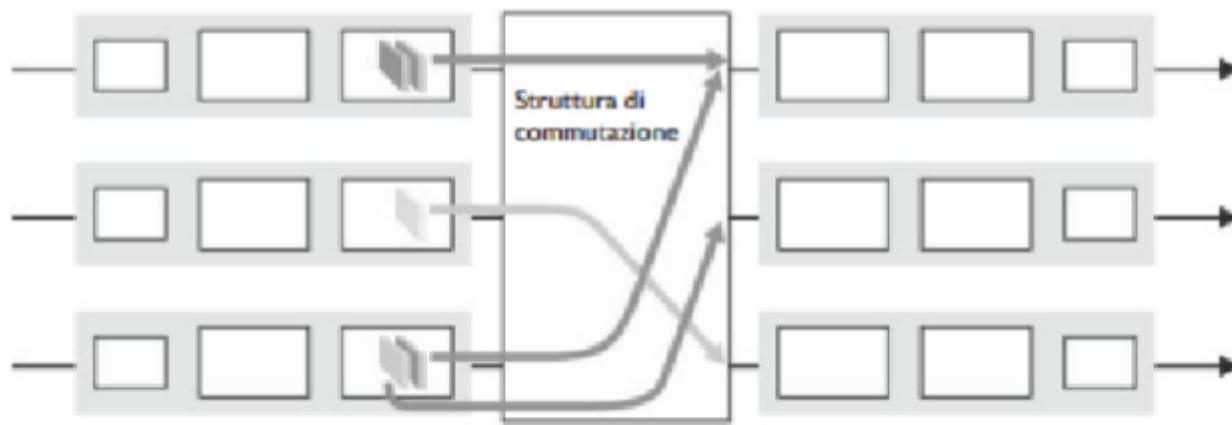
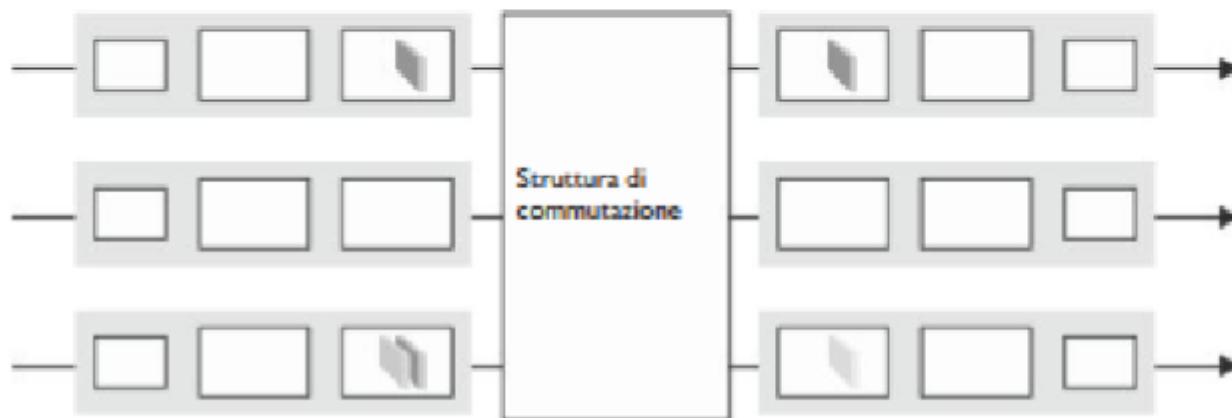


Figura 4.8 Blocco in testa (HOL) alla coda di input di uno switch.

Il pacchetto grigio chiaro subisce blocco HOL



Legenda:

destinato alla porta di uscita superiore

destinato alla porta di uscita centrale

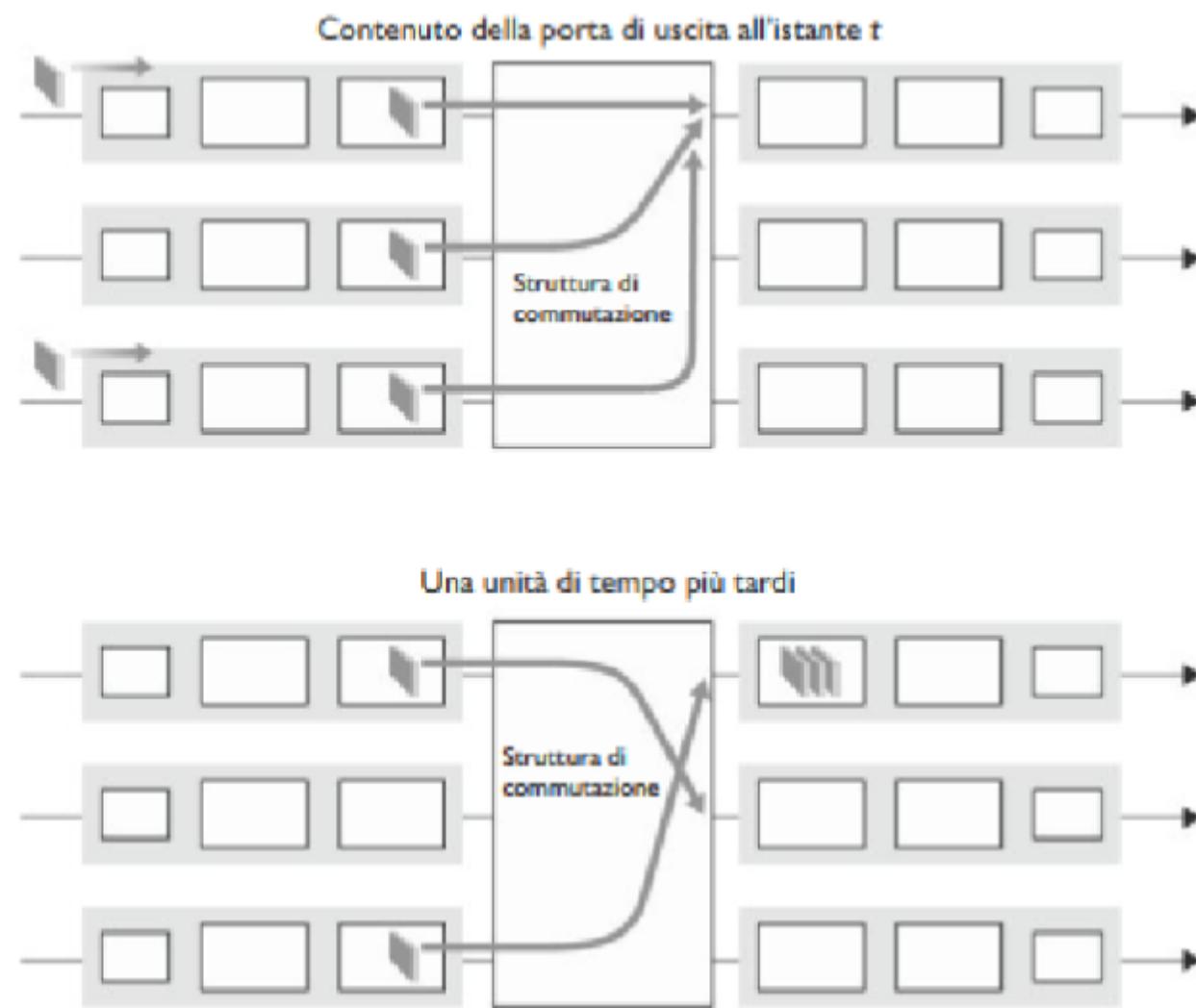
destinato alla porta di uscita inferiore

2.3.1.2 in uscita

Dalla velocità supposte all'inizio di R_{line} ed R_{switch} , e dalla possibilità della porta di uscita di inviare solo un pacchetto alla volta, possiamo notare come si creino situazione di sovraccarico del buffer. Infatti nel tempo d'invio di un pacchetto è probabile che arrivino altri n pacchetti che si dovranno mettere in coda.

Figura 4.9

Accodamento alle porte di uscita.



Si adottano quindi politiche di "drop-tail" (eliminazione coda) per scartare alcuni pacchetti e quindi inviare al mittente segnali di congestione. uno degli alg. di eliminazione più noti è RED (random early detection).

Inoltre in presenza di queste code uno "schedulatore di pacchetti" deve stabilire in quale ordine trasmetterli.

2.3.2 quanta memoria di buffer usare?

è di fondamentale importanza capire la grandezza ottimale del buffer, in quanto più si allunga la coda e più peggiora le situazione sopra citata.

Per molti anni la regola è stata la seguente:

$$\text{Buff_Size} = C \cdot RTT$$

con C la capacità del link ed RTT il round trip medio.

Ex:

$$C = 10\text{Gbps}$$

$$RTT = 250\text{ms}$$

$$B = 2,5\text{Gbit}$$

In seguito a diversi studi, in presenza di un gran numero di flussi TCP (N), viene suggerita invece la seguente:

$$\text{Buff_Size} = RTT \cdot \frac{C}{\sqrt{N}}$$

Si potrebbe pensare di aumentare a dismisura i buffer, però questo porta a dei problemi di ritardi end-to-end. Non conviene.

La grandezza del buffer fa sorgere anche il problema di **bufferbloat**.

2.4 schedulazione dei pacchetti

come si determina l'ordine di uscita (dei pacchetti) dalle code.

2.4.1 FCFS

coda FIFO. vengono inviati rispetto all'ordine di arrivo.

in caso di insufficienza di spazio la **packet discarding policy** decide se eliminare o meno il pacchetto (perderlo).

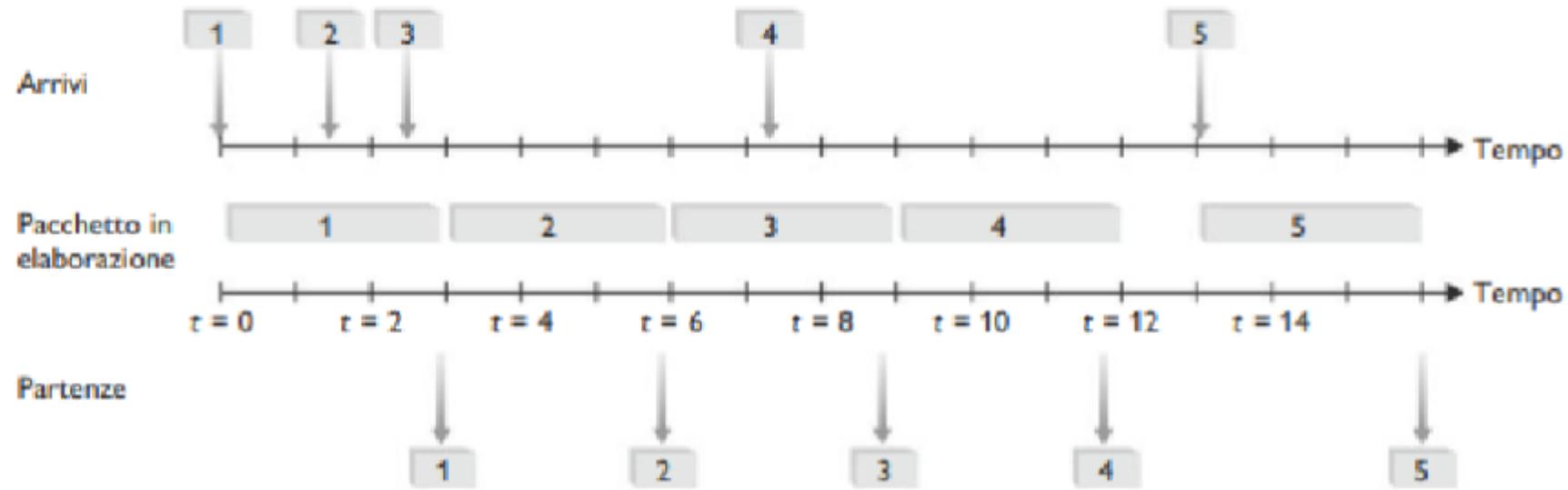


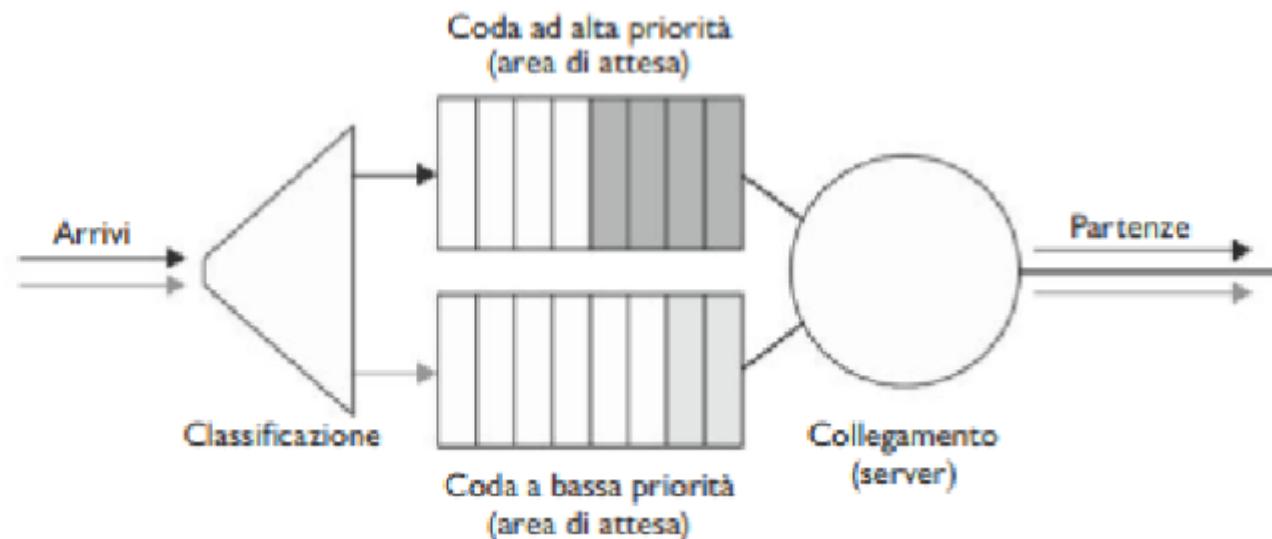
Figura 4.12 Coda FIFO in azione.

2.4.2 Code con priorità

si assegna una priorità (classe) ad ogni pacchetto, verrà inviato quello con priorità più alta presente nelle code. ad esempio un pacchetto voice-over-ip avrà più priorità rispetto ad uno SMTP.

Ciascuna classe ha la propria coda.

Figura 4.13 Coda con priorità.



In questo esempio abbiamo: 1,3,4 (alta), 2,5 (bassa)

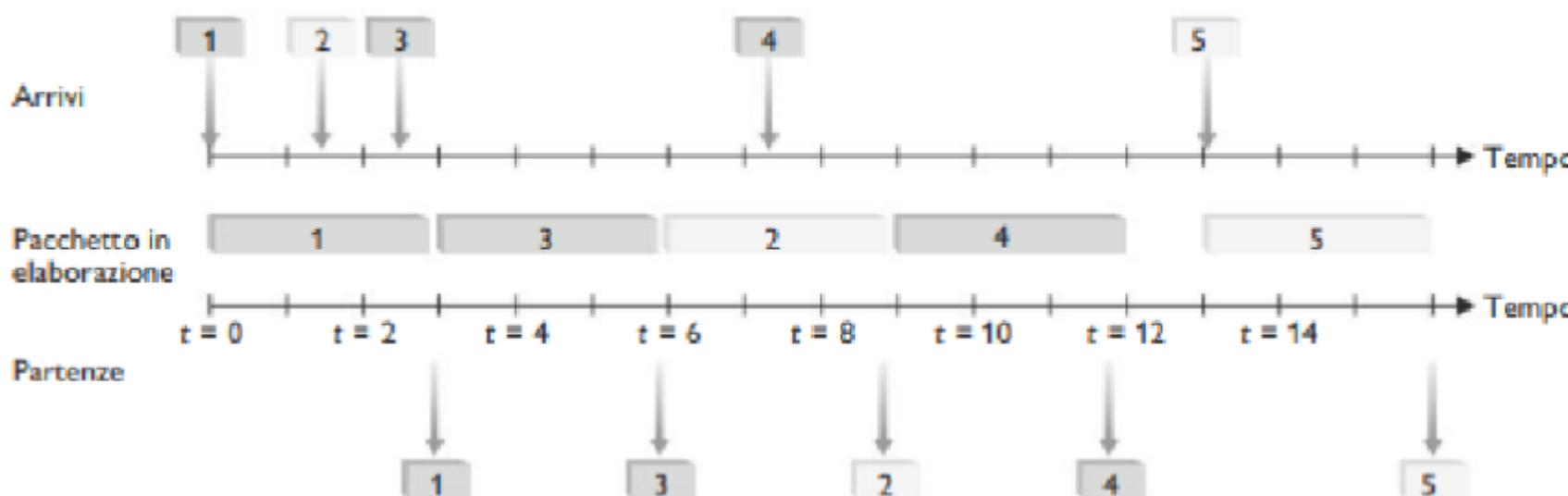


Figura 4.14 Operazioni di una coda con priorità.

nella modalità di accodamento a priorità senza prelazione la trasmissione dei pacchetti non può essere interrotta. In questo caso il 4 viene inviato dopo il 2.

2.4.3 Round Robin e Weighted fair queuing (WFQ)

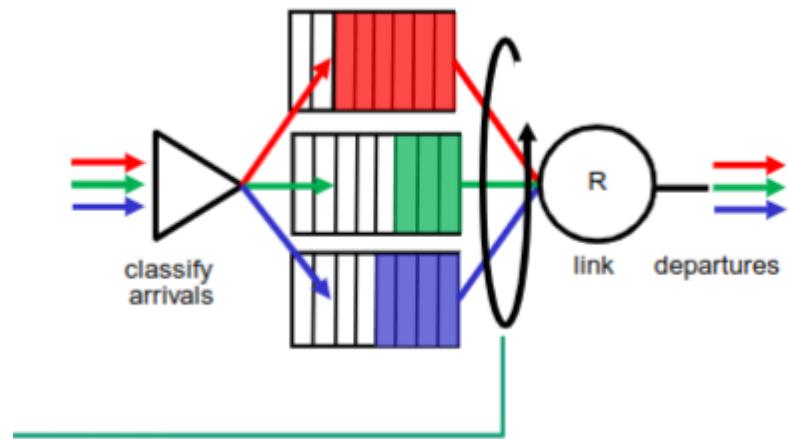
round robin classico: si invia in alternanza pacchetti di classi differenti.

la versione conservativa **work-conserving round robin** si invia sempre un pacchetto (fino a quando sono presenti nelle code), se trova una coda vuota controlla la successiva.

una versione generalizzata è **Weighted fair queuing (WFQ)** che implementa una round robin classico conservativo, ma aggiunge un servizio differenziato per ogni classe.

Si associa un peso w_i ad ogni classe. e si avrà un rendimento pari a:

$$\text{rendimento}_i = \frac{w_i}{\sum w}$$



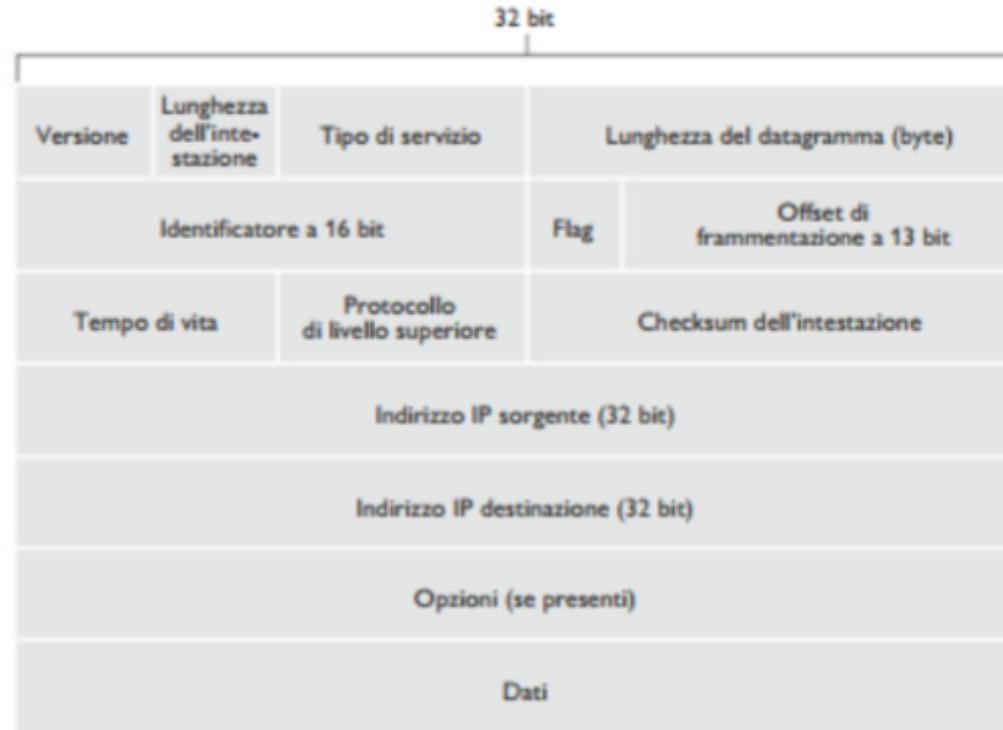
3 Protocollo internet (IP)

il pacchetto a livello di rete è detto **datagramma**

3.1 IPv4

3.1.1 formato datagrammi

Figura 4.17 Formato dei datagrammi IPv4.



- **Numero di versione:** 4bit
- **lunghezza intestazione** (header length): 4bit. dati i vari campi opzioni, le intestazioni ip sono variabili. questi bit indicano dove iniziano i dati del datagramma
- **tipi di servizio** (type of service, tos): 2bit, specifica il tipo di datagramma. ad esempio se richiede basso ritardo, alto throughput. serve anche a [notificare la congestione](#)
- **lunghezza del datagramma:** 16bit.
- **identificatore:** identifica i pacchetti e i suoi frammenti
- **flag:** per la frammentazione. può essere "dont fragment" (DF) o "more fragment" (MF)
- **offset di frammentazione:** 13bit, indica l'offset rispetto al primo byte del primo frammento.
- **time to live (TTL):** implementato per avere la sicurezza che un datagramma non resti per sempre in rete. viene decrementato al passaggio (hop) da un router. se è 0 deve essere scartato
- **protocollo:** tipo di protocollo usato a livello trasporto. TCP = 6 invece UDP=16

- **checksum intestazione**: checksum come quello al [livello trasporto](#), applicato a tutte le intestazioni.

#attenzione! perché si fa checksum anche al liv. di rete?

- liv. trasporto applica il checksum su tutto il segmento, qui solo sulle intestazioni
- TCP/UDP non per forza appartengono alla stessa pila di protocolli di IP. quindi può lavorare anche con altri

- **indirizzi ip sorgente e destinazione**: 32 bit ognuno
- **opzioni**: non vengono inseriti in tutti i datagrammi.
- **payload** (dati): contiene il segmento al liv. trasporto. può anche trasportare altri tipi di dati (messaggi ICMP).

3.1.2 frammentazione

il router interagisce con diversi canali, in quanto ogni canale ha uno specifico [MTU](#) i pacchetti possono dover essere frammentati / divisi, per poi essere riassemblati a destinazione.

esempio di frammentazione:



si nota come l'ultimo datagramma abbia **MF = 0** perché è l'ultimo.

esempio di non frammentazione, impostando **DF = 1**



3.1.3 indirizzamento

gli indirizzi sono scritti in notazione decimale puntata.

in internet la strategia è detta **Classless interdomain routing** (CIDR, pronunciata "cider").

ad esempio:

a.b.c.d/x

dove x è il "**prefisso**" (numeri di bit bloccati, sottorete).

Quindi per ogni rete avremo un numero di host pari a:

$$2^{32-x}$$

in precedenza si usava un **classful addressing**, ovvero le sottoreti erano fisse (8,16,24) ed erano note come reti di classe A,B,C.

Si causavano infatti problemi con il numero di host (pochi o esagerati), che causò un rapido esaurimento degli host.

3.1.3.1 come ottengo un blocco di indirizzi?

se l'azienda pinco pallino vuole richiede un gruppo di indirizzi ha 2 strade:

- **chiedere all'ISP**: il quale ha magari un blocco di indirizzi ad esempio 200.23.16.20/20 che a sua volta li dividerà in blocchi uguali

200.23.16.20/20

-- sceglie di fare 8 gruppi

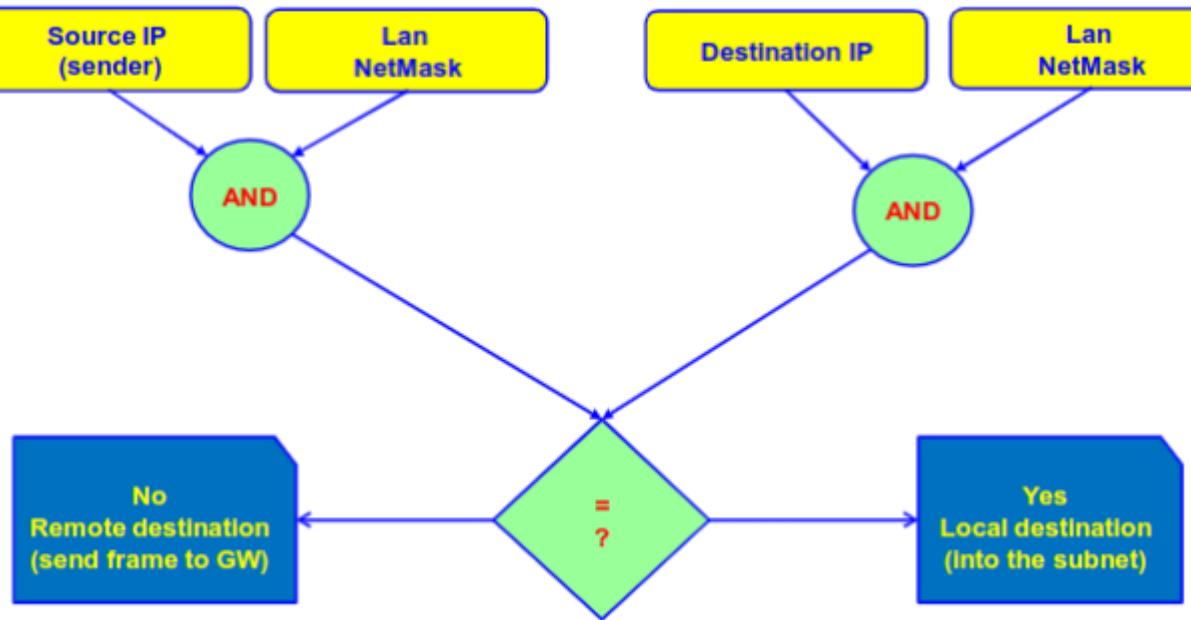
200.23.16.20/23

...

200.23.16.30/23

- **chiedere ICANN** (internet corporation for assigne names and numbers): associazione senza scopo di lucro che alloca IP e gestisce i root DNS.

3.1.3.2 come faccio a capire se la destinazione è in locale?



3.1.3.3 DHCP

una volta ottenuto un blocco di indirizzi occorre assegnarli ai dispositivi, si può fare manualmente oppure in maniera automatica con il **dynamic host configuration protocol** (dhcp). per questo è detto protocollo **plug and play** o **zero-conf**

consente di ottenere: ip, maschera, gateway (porta router per uscire dalla sottorete), indirizzo dns locale.

gli indirizzi offerti possono essere fissi o temporanei (diverso ogni volta che si collega oppure allo scadere del tempo di lease).

è un protocollo client-server e si basa su 4 passi:

- **individuazione server DHCP**

client deve identificare il server DHCP. Invia in broadcast " 255.255.255.255 " un messaggio DHCP-DISCOVER che contiene come sorgente l'indirizzo " 0.0.0.0 " tramite un pacchetto UDP sulla porta 67.

yiaddr = your internet address

- **offerta del server DHCP**

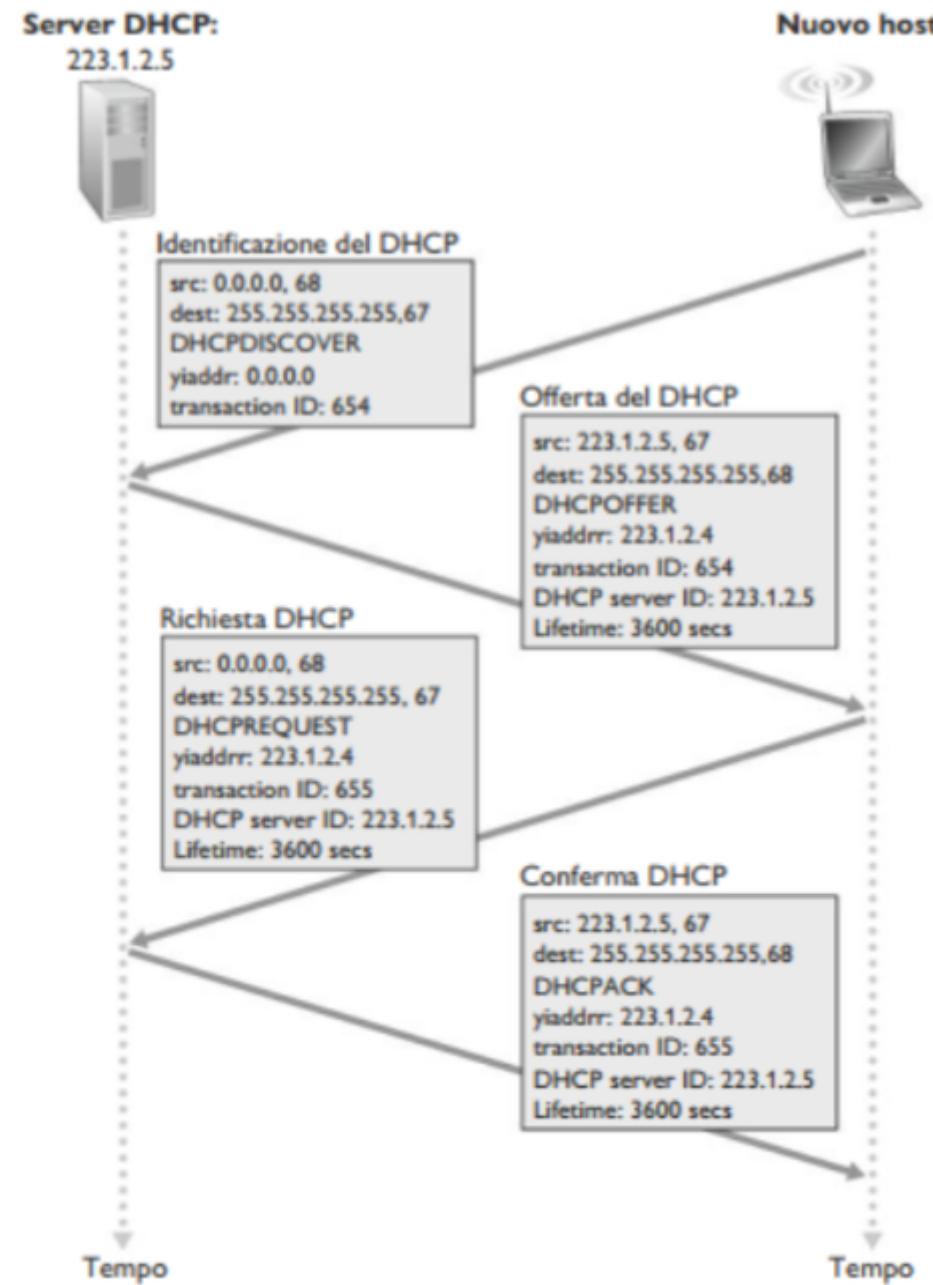
server risponde al client inviando un DHCP-OFFER in broadcast nella sottorete. È possibile che siano presenti più server nelle reti, quindi il

client potrà scegliere fra varie offerte.

l'offerta contiene:

- id transazione
- IP proposto
- maschera
- lease time, durata validità dell'IP (ore/giorni)
- **richiesta DHCP**
client risponde ad una particolare offerta con **DHCP REQUEST**
- **conferma DHCP**
server risponde con **DHCP ACK**

Figura 4.24 Interazione client-server DHCP.



DHCP non è privo di difetti. in ambito di mobilità, quando ci si sposta di sottorete viene assegnato un nuovo indirizzo, perdendo di conseguenza la connessione attiva.

più dettagli su [one note superiori]([DHCP - DNS \(Visualizzazione Web\)](#))

3.1.4 Network address translation (NAT)

protocollo che permette di tradurre gli indirizzi da privati a pubblici.

gli host che si connetteranno ad una particolare porta del router saranno identificati con l'indirizzo associato.

nel momento in cui arriva una risposta, il router usa la **nat translation table**

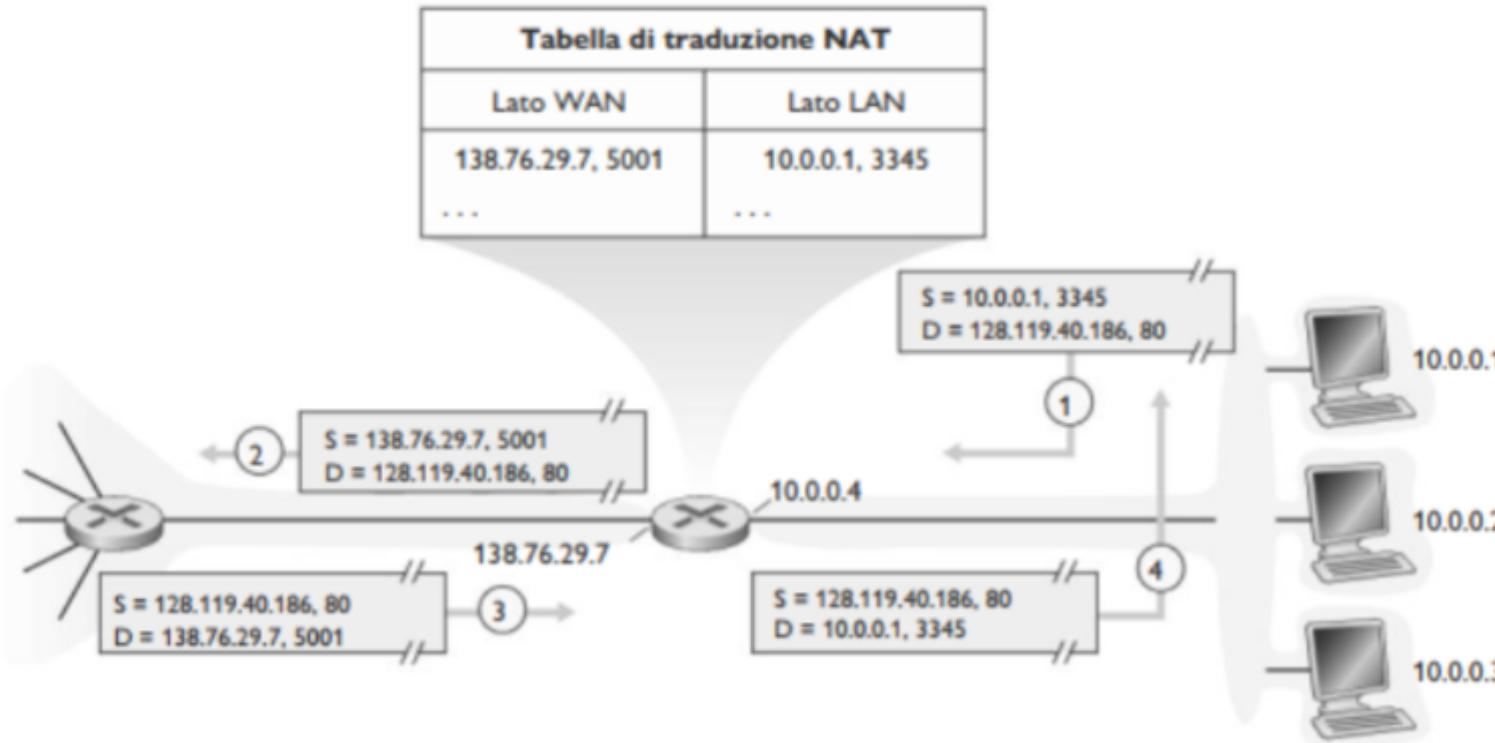


Figura 4.25 NAT.

come si nota dalla figura il router usa un numero di porta diverso per identificare i vari host (generalmente usato per identificare processi, non gli host).

oppure una critica filosofica è che il nat viola il principio end to end, ovvero che gli host devono comunicare tra loro senza intromissione di nodi o modifiche varie (ip, porta) 😊

UPNP: protocollo che permette di creare entry all'interno della tabella NAT (apertura porta) senza che la richiesta parte dalla lan interna.

3.2 IPv6

ipv6 venne introdotto per far fronte a diverse limitazioni di ipv4 come:

- scarsità indirizzi ip (circa nel 2011 sono stati assegnati gli ultimi blocchi)
- scalabilità routing: numero elevati di entry nelle tabelle di routing, data dal fatto che gli indirizzi IPv4 non sono assegnati secondo un ordine geografico ma random.

inoltre furono introdotti nuovi servizi:

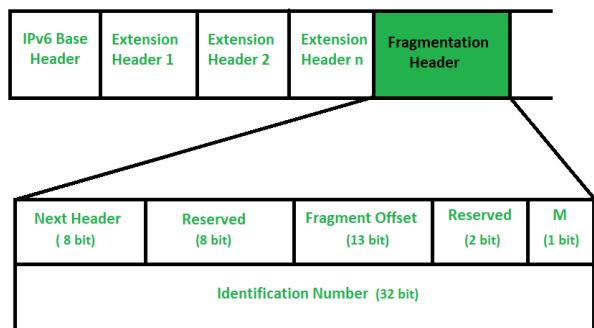
- **sicurezza**: tramite l'implementazione nativa del protocollo Ipsec (internet protocol security) che fornisce meccanismi di autenticazione, integrità e crittografia per proteggere le comunicazioni di rete.
- **auto-configurazione**: noto anche come Stateless Address Autoconfiguration (SLAAC), permette la configurazione autonoma degli host.
- **multicast addressing**: fornisce la trasmissione di dati ad un gruppo di destinatari specifico.

3.2.1 formato datagramma



- **versione**: 4bit, versione protocollo. btw non è immediata la transizione v4 -> v6 (si vedrà in [passaggio da IPv4 a IPv6](#))
- **classi traffico**: 8bit, come il [tos](#) di ipv4 usato per attribuire priorità a determinati datagrammi.

- **etichetta flusso**: 20 bit, identificativo usato per i circuiti virtuali. Questi identificano un particolare percorso (predeterminato) che dovrà seguire il pacchetto.
- **lunghezza payload**: 16bit, intero senza segno. indica il numero di byte che seguono l'intestazione standard di 40byte.
- **intestazione successiva**: contiene il tipo del prossimo header. generalmente contiene il tipo del protocollo di trasporto usato (ex: tcp = 6). nel caso di frammentazione (valore = 44) il payload viene shiftato per fare spazio al fragmentation header (a sua volta questo header avrà un riferimento all'intestazione successiva).



- **limite di hop**: sarebbe il TTL.
- **indirizzi sorgente destinazione**: 128bit
- **dati**

confrontando ipv4 ed ipv6 si notano differenze come:

- **indirizzamento esteso**: indirizzi ip da 128 bit
- **intestazione ottimizzata a 40byte**:
eliminati alcuni campi come:
 - il checksum tanto viene eseguito lo stesso da altri (trasporto o meglio data link) ed inoltre rallentava tutto il processo in quanto a causa sua, veniva ricalcolato ogni volta il TTL (oneroso di risorse).
- campo opzioni

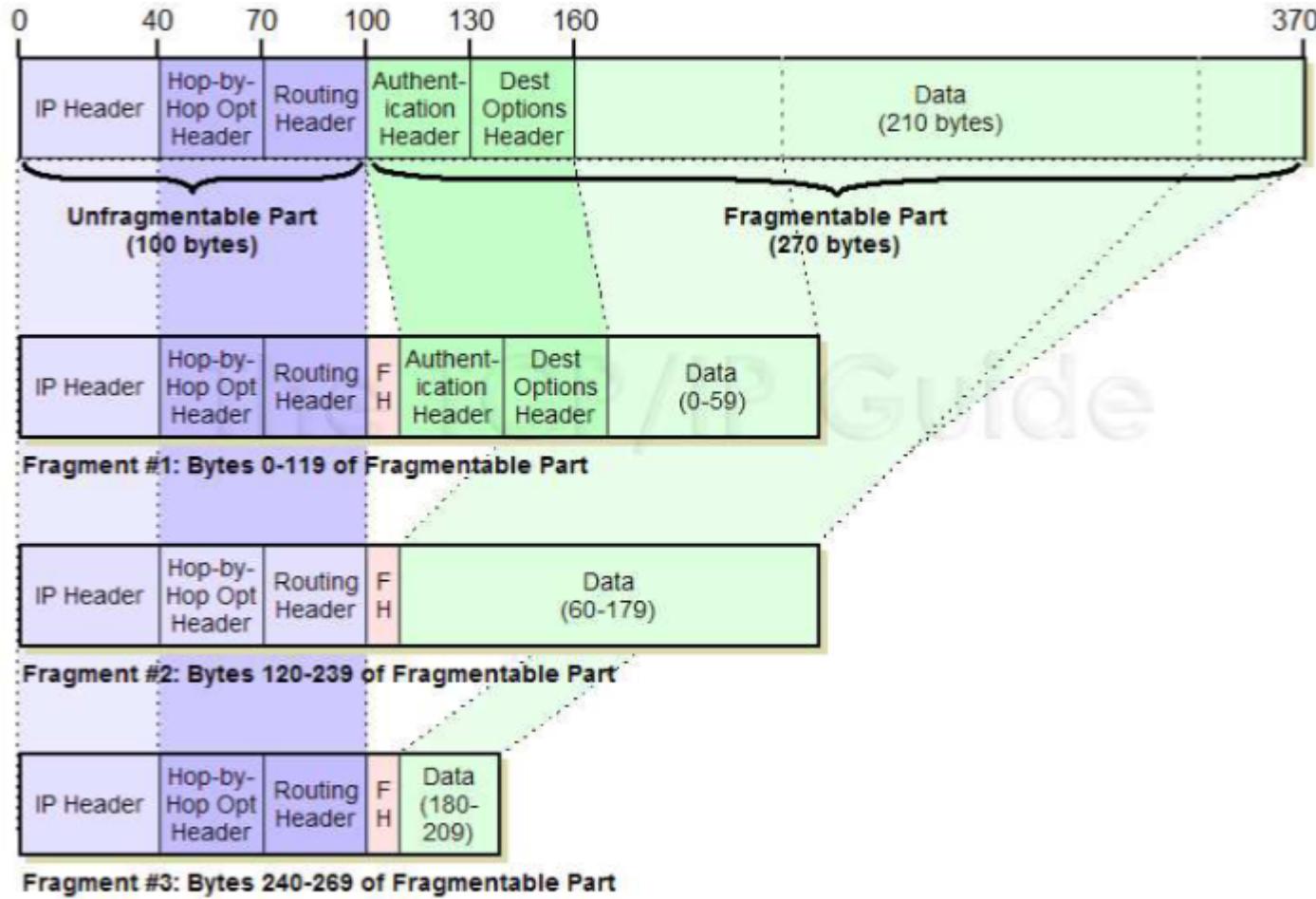
questa ottimizzazione consente un elaborazione più rapida.

ipv6 non permette la frammentazione intermedia nei router:

se un router riceve un datagramma ipv6 troppo grande (MTU), ritorna al mittente un messaggio di errore ICMP.

il mittente poi procederà ad inviarne uno di dimensioni inferiori (minimo 1280bytes). In questo modo si trasferisce questo onere ai sistemi periferici, facilitando l'instradamento.

si userà quindi il fragmentation header (FH), ereditato da [ipv4](#); inoltre **solo** la parte successiva al FH potrà essere frammentata (data)



3.2.2 struttura indirizzo

3.2.2.1 caratteristiche

formato da 8 campi da 16bit, intervallati dai due punti ":" .

sequenza di zeri si possono omettere con " :: ", e gli zero iniziali possono essere omessi.

per facilitare la [transizione](#) ipv4-ipv6 una parte è stata riservata agli indirizzi ipv4, avremo quindi:

::192.168.0.1

con i primi 12byte a zero.

altri indirizzi speciali:

-- loopback

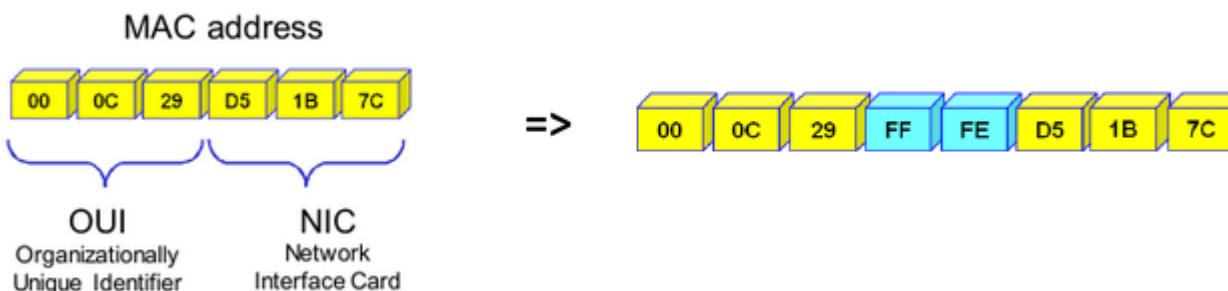
::1

3.2.2.2 indirizzi privati, EUI-64

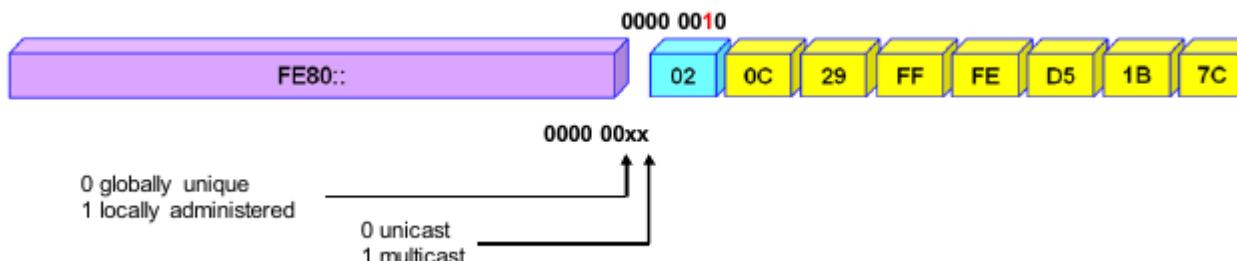
per attribuire un indirizzo ipv6 privato (potrà navigare nella rete domestica) si fa uso di questo standard.

l'indirizzo sarà suddiviso:

- la prima parte dell'indirizzo si imposta di default a `FE80::`
- si estreia il MAC della macchina e gli si aggiungono 2 byte (default, FF ed FE) in mezzo (tra OUI e NIC)



- si setta il penultimo bit dell'indirizzo ottenuto per renderlo locally administrated



windows non usa questo standard. gli indirizzi cominciano con `FE80::` seguiti da numeri random.

3.2.2.3 suddivisione

la suddivisione avviene in base al tipo indirizzamento ed instradamento che viene effettuato:

- **unicast**: l'indirizzo identifica una particolare interfaccia di rete. il datagramma verrà consegnato solo a questa interfaccia.
- **anycast**: identifica un gruppo di interfacce. il datagramma verrà consegnata a solo una delle interfacce di questo gruppo
- **multicast**: come anycast solo che il datagramma arriva a tutte le interfacce.

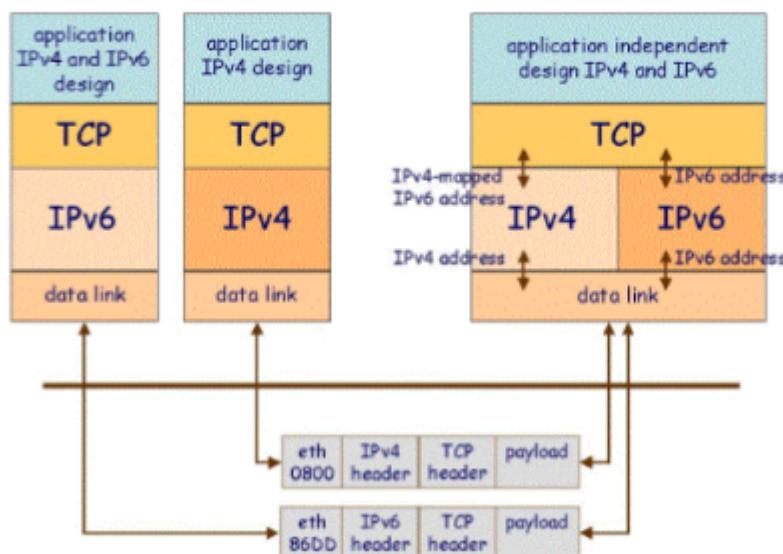
#attenzioneplis ipv6 non ha indirizzi broadcast, viene fatto uso di multicast su tutti i nodi.

3.2.3 passaggio da IPv4 a IPv6

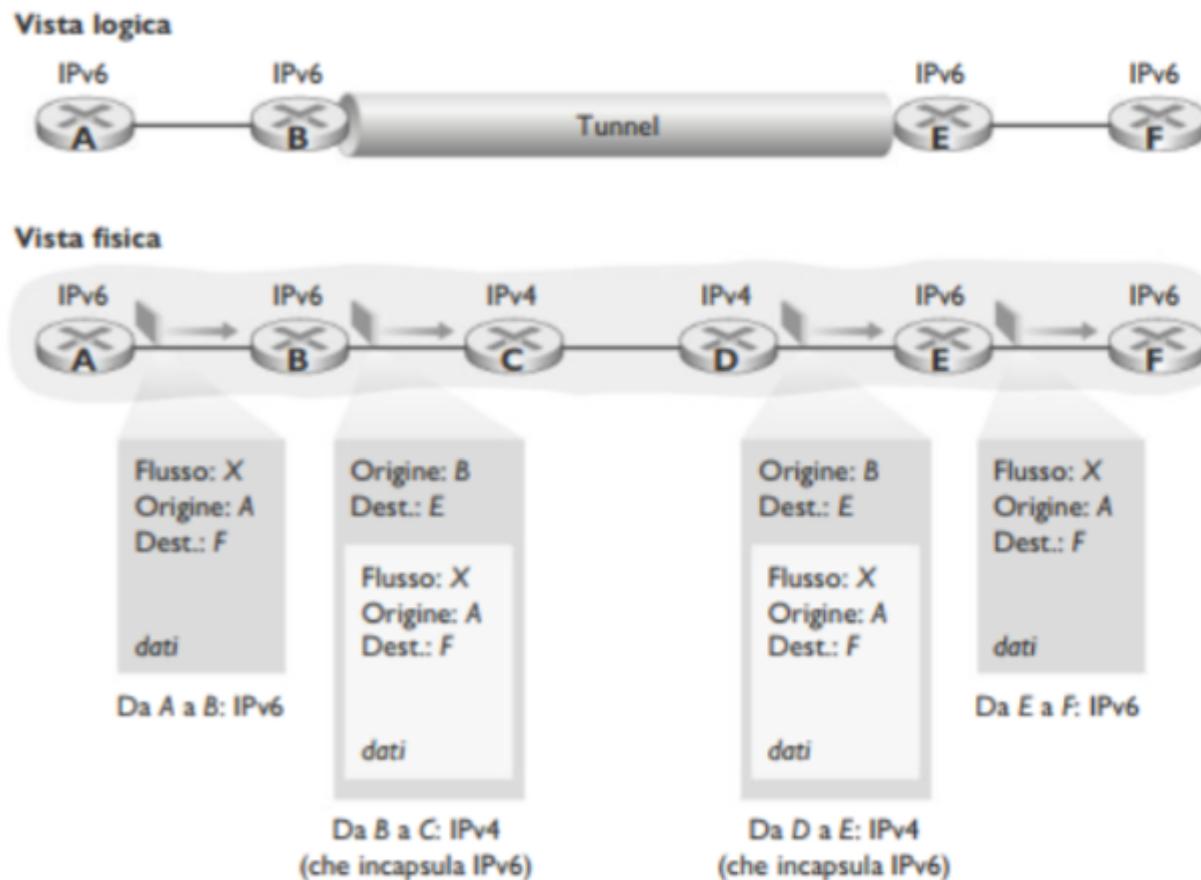
il passaggio non è automatico. i sistemi ipv6 sono retrocompatibili, gli ipv4 no.

si potrebbe agire in due modi:

1. **giornata campale**: giornata in cui tutte le macchine ipv4 spente ed aggiornate ad ipv6. (crazy)
2. **dual stack**: si predisponde una pila protocollare doppia. La macchina capirà se si tratta di un pacchetto IPv4 o IPv6 grazie alla frame Ethernet (di cui parleremo dopo) che dispone di una campo eth che sarà uguale a 0800 o 86DD rispettivamente per IPv4 e IPv6



3. **tunneling**: assumendo che due host ipv6 vogliono comunicare, ma tra di loro sono presenti router intermedi ipv4. si procede incapsulando i pacchetti ipv6 all'interno di pacchetti ipv4, che poi verrano instradati normalmente.



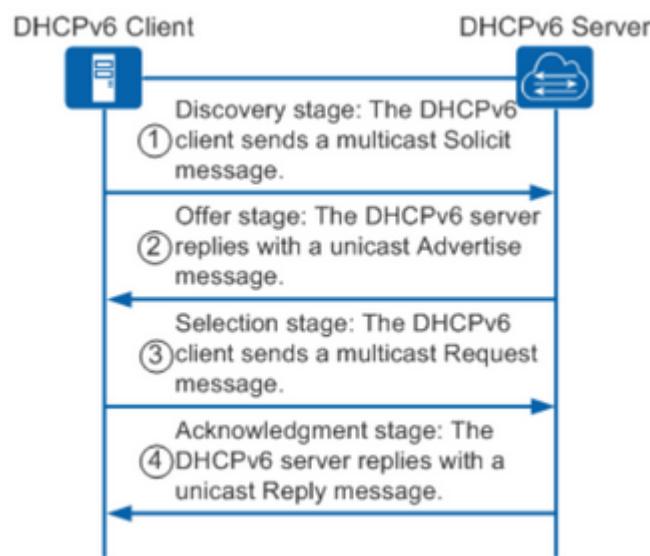
3.2.4 DHCPv6

simile al v4 ma con qualche differenza.

host invierà in questo caso un pacchetto in multicast (invece di broadcast), in questo modo risponderanno solo le macchine che sanno di essere un server dhcp.

l'host inserirà nella request il suo indirizzo locale (local link).

la risposta sarà un messaggio unicast diretto all'host.



3.3 firewall

sono di tipo:

- **software (bad):**

processo applicativo. questa tipologia di firewall si limita a leggere indirizzi e porte e decidere se il pacchetto può andare avanti o essere buttato a seconda dei filtri applicati,tipicamente questi firewall lavorano a livello di rete.

- **hardware (good):**

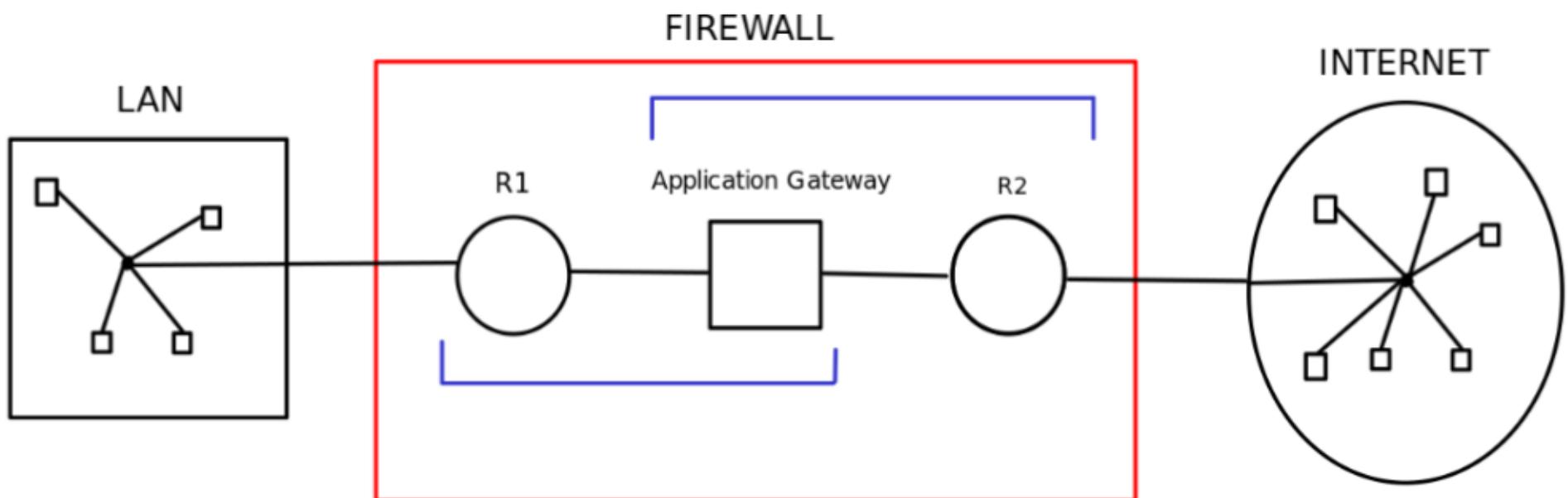
composto da un router di ingresso, un application gateway ed un router di uscita, collegati tramite reti interne non visibili dall'esterno (LAN Private).

I router, saranno dispositivi molto semplici e mancanti di un vero e proprio Sistema operativo (spesso rimpiazzato da un semplice firmware), questo fa sì che essi siano intrinsecamente sicuri, perché hanno molti meno elementi da poter bucare.

Inoltre, non avendo regole di routing, il router più interno (R1) non sarà conosciuto da internet e allo stesso modo, il router più esterno (R2) non sarà a conoscenza della LAN.

quelli hardware sono praticamente dei router che si interfacciano con la zona interna (da proteggere) ed un'altra zona interna che però si interfaccia con l'esterno.

L'application gateway, ci permetterà di effettuare un filtraggio a livello applicativo, ovvero, potremmo controllare il contenuto di ogni singolo pacchetto entrante e decidere secondo dei criteri di scelta programmabili di far passare o meno il pacchetto in questione.



3.4 PROTOCOLLO ARP

Address Resolution Protocol.

associazione ip mac: L'indirizzo IP identifica la posizione geografica della macchina nella rete LAN. Al livello applicativo due programmi comunicano tramite indirizzi IP, ma al livello DataLink la comunicazione avviene utilizzando l'indirizzo fisico della macchina (MAC), ma l'indirizzo MAC è totalmente slegato dalla LAN e dall'applicativo.

- Quindi bisogna trovare un modo per associare l'indirizzo MAC di una macchina al suo indirizzo IP e viceversa.

Il mittente (colui che manda il messaggio) conosce il proprio indirizzo IP e MAC e l'indirizzo IP del destinatario.

Ma non conosce il MAC address del destinatario.

1. **arp request:** Allora quello che fa il mittente è effettuare una richiesta tramite il protocollo ARP che, tramite la generazione di una Frame Ethernet con il campo indirizzo MAC Destinazione impostato a Broadcast e specificando il proprio indirizzo IP, il proprio MAC e l'indirizzo IP del destinatario.

2. **verifica IP:** Essendo una richiesta Broadcast tutte le macchine della LAN ricevono questo frame ethernet e analizzeranno questa richiesta ARP. Tutte le macchine verificheranno se l'IP address della frame ethernet corrisponde con il proprio indirizzo IP e scarteranno la richiesta in caso negativo.
3. **arp reply:** L'unica macchina che accetterà la richiesta sarà la macchina di destinazione, perché l'indirizzo IP coincide. Una volta che la macchina accetta la richiesta questa risponderà al mittente inviando la stessa frame ethernet, impostando di conseguenza il proprio MAC.

tabella arp: Tramite questi messaggi viene riempita la tabella ARP, con la seguente struttura

[Indirizzo IP, Indirizzo MAC, TTL]

- è presente in ogni macchina, è una sorta di cache con dimensione limitata.

problemi: Questo protocollo può essere visto come un autocertificazione, non c'è sicurezza infatti non prevede autenticazione, è un protocollo senza stato un ARP reply può essere mandato senza una precedente ARP request.
di conseguenza:

- tutti i dispositivi che ricevono questa frame devono aggiornare la propria cache.
- E' possibile quindi dirottare il traffico IP. Il MAC address può essere cambiato da CLI in qualsiasi SO, quindi una macchina potrebbe sostituirsi per intero ad un'altra macchina, questo approccio non è pensato con nessun livello di sicurezza e non possiamo fidarci della risposta che ci arriva.

si suppone comunque che salendo la pila protocollare il messaggio verrà autenticato.

#attensionplis gli indirizzi MAC, a differenza degli indirizzi IP non sono gerarchici ma sono di tipo flat infatti, con IP possiamo tracciare una strada conoscendo l'intero indirizzo mentre con i MAC no in quanto ogni indirizzo è indipendente (possiamo paragonare gli IP alla telefonia fissa e i MAC alla telefonia mobile).

#attensionplis Il protocollo ARP è a metà tra il Network Layer e il DLL poiché sfrutta sia indirizzi IP che MAC. Questa osservazione ci fa capire che TCP/IP non è a livelli e la pila è una semplice astrazione. ARP può essere usato solo all'interno della rete locale, per uscire al di fuori di essa è necessario un gateway.

3.5 Protocollo RARP

Non conosco il mio IP ma conosco il mio MAC e mando la richiesta ad un server RARP che conosce l'associazione e la macchina così può autoconfigurarsi.

Questo protocollo di norma non è usato. La richiesta viene mandata in broadcast poiché il client non conosce l'indirizzo del server. Questo protocollo essendo collocato in un server è centralizzato.

3.6 BOOTP

E' un protocollo pensato per effettuare il bootstrap da rete in presenza di macchine senza OS o per le quali si vuole caricare un OS da rete.

fasi:

1. La macchina conosce solamente il proprio MAC, attraverso il server RARP ottiene un indirizzo IP.
2. Tramite questo IP riesce a configurarsi per comunicare e chiedere al server di mandargli tutto il sistema operativo attraverso un protocollo TFTP banalmente un FTP senza autenticazione. E' utilizzato per caricare il firmware di primo utilizzo nei router. TFTP non è un protocollo che si trova nelle macchine essendo molto pericoloso.

RARP + BOOTP serve a caricare l'OS su una macchina senza configurazione iniziale. Questo protocollo non è molto utilizzato, veniva usato in passato quando aggiornare il sistema operativo era oneroso e i costi dei dischi fissi molto elevati.

4 Algoritmi di routing

Lo scopo di un algoritmo di routing è determinare i percorsi di costo minimo tra sorgente e destinazione.

oltre al costo si aggiungono le problematiche delle policy, ovvero che un router di un organizzazione **X** non può inoltrare pacchetti verso router di organizzazioni **Z**.

si può formulare il problema tramite l'uso dei grafi

$$G = (N, E)$$

dove i nodi sono i router e gli archi i collegamenti con associato il relativo peso.

- se un arco non è presente

$c(x, y) = +\text{INF}$

- si considerano grafi non orientati

$c(x, y) = c(y, x)$

- un nodo x è detto **adiacente** ad y se esiste un arco che li collega
- si dice **percorso** una sequenza di nodi

(x_1, x_2, \dots)

tali che ciascuna delle coppie:

$(x_1, x_2), (x_2, x_3), \dots$

sia un arco in E .

4.1 classificazione

gli algoritmi possono essere:

- **flooding**: invio in broadcast su tutti i nodi. sono certo che il pacchetto arrivi a destinazione, ma genero un forte overhead.
- **centralizzato / link state**: riceve in input tutti i collegamenti con i relativi costi e calcola il percorso di costo minimo tra sorgente e destinazione.
- **decentralizzato / distance vector**: il costo minimo viene calcolato in modo distribuito e iterativo. nessun nodo possiede info su tutti i collegamenti di rete (inizialmente), poi attraverso scambio di info con altri nodi adiacenti ricava il percorso minimo.

si differiscono anche in:

- **statici**: i collegamenti cambiano raramente
- **dinamici**: gli instradamenti sono calcolati in base al traffico o topologia rete.

un terzo modo è in base alla sensibilità sul carico della rete (load sensitive o load insensitive). nei sensitive i costi variano dinamicamente in modo da incentivare i collegamenti meno congestionati.

4.2 link-state

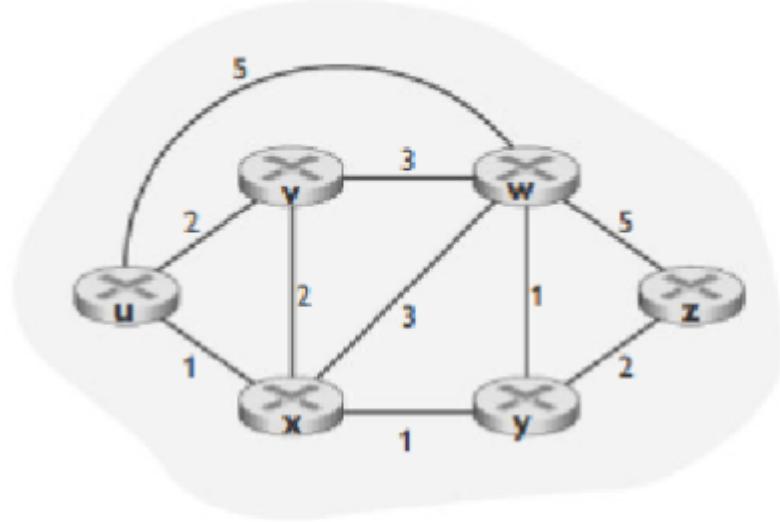
come già detto in un instradamento link state l'algoritmo conosce lo stato della rete. si ottiene facendo inviare ad ogni nodo, lo stato dei propri collegamenti (spesso si usa [link state broadcast](#)).

metrica: capacità dei link, ritardo link, capacità buffer

si usa l'algoritmo di Dijkstra:

```
1  Inizializzazione:
2       $N' = \{u\}$ 
3      per tutti i nodi  $v$ 
4          se  $v$  è adiacente a  $u$ 
5              allora  $D(v) = c(u,v)$ 
6          altrimenti  $D(v) = \infty$ 
7
8  Ciclo
9      determina un  $w$  non in  $N'$  tale che  $D(w)$  sia minimo
10     aggiungi  $w$  a  $N'$ 
11     aggiorna  $D(v)$  per ciascun nodo  $v$  adiacente a  $w$  e non in
12          $N'$ :
13              $D(v) = \min(D(v), D(w) + c(w,v))$ 
14     /* il nuovo costo verso  $v$  è il vecchio costo verso  $v$ 
15        oppure
16        il costo del percorso minimo noto verso  $w$  più il
17        costo da  $w$  a  $v$  */
18     ripeti in ciclo finché non si verifica che  $N' = N$ 
```

quindi avremo ad esempio:

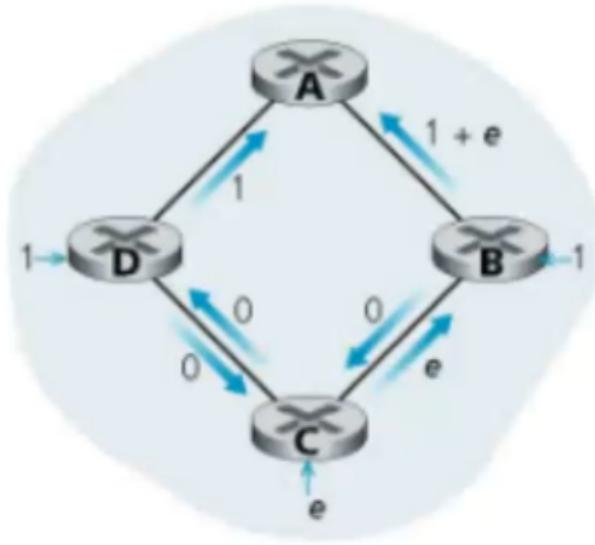


iterazione	N'		D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),d(z)
0	{u}		2,u	5,u	1,u	+	+
1	{u,x}		2,u	4,x		2,x	+
2	{u,x,v}			4,x		2,x	+
3	{u,x,v,y}			3,y			4,y
4	{u,x,v,y,w}						4,y
5	{u,x,v,y,w,z}						

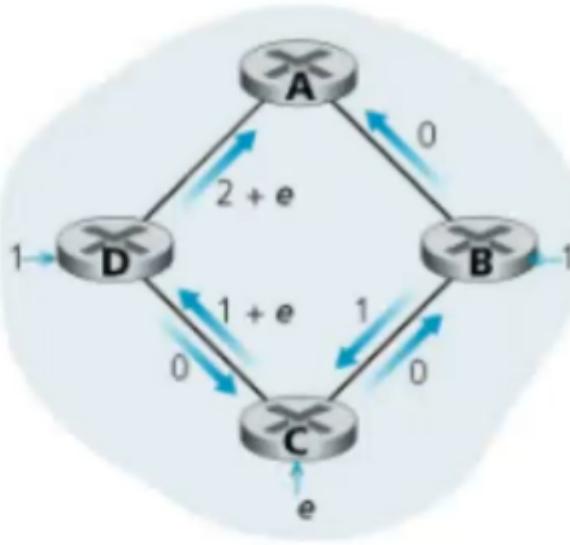
per arrivare ad un determinato nodo avremo a disposizione i predecessori.

complessità $O(n^2)$

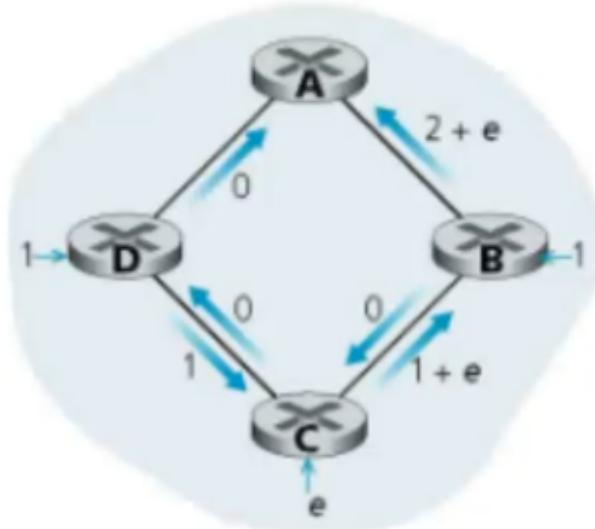
4.2.1 problema oscillazione



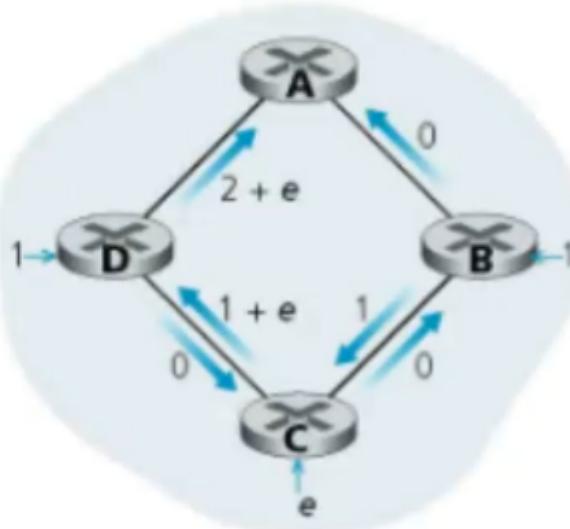
a. Routing iniziale



b. B e C rilevano il miglior percorso verso A, senso orario



c. B, C e D rilevano il miglior percorso verso A, senso antiorario



d. B, C e D, rilevano il miglior percorso verso A, senso orario

il traffico parte da C per arrivare ad A.

assumendo che il peso dei collegamenti è dato dal traffico su di essi. si verificherà un oscillazione dei pesi che provocherà un cambio di rotte continuo.

per evitare ciò si consiglia:

- evitare che i router si sincronizzino, impostando un intervallo randomico per l'invio dell'avviso di collegamento.

4.3 distance-vector

mentre link state usa info globali, distance vector si basa su un approccio:

- iterativo: viene ripetuto fino a quando non avviene ulteriore scambio di info tra vicini
- asincrono: non richiede che tutti i nodi comunicano con altri
- distribuito: ciascun nodo riceve info da un suo vicino, ai quali dopo aver effettuato il calcolo restituisce il risultato.
- auto-terminante: non è necessario un segnale di blocco terminale.

si usa alg. dei bellman-ford.

Algoritmo Bellman-FordA ciascun nodo x :

```

1  Inizializzazione:
2    per tutte le destinazioni  $y$  in  $N$ :
3       $D_x(y) = c(x,y)$  /* se  $y$  non è adiacente, allora  $c(x,y)$ 
   =  $\infty$  */
4    per ciascun vicino  $w$ 
5       $D_w(y) = ?$  per tutte le destinazioni  $y$  in  $N$ 
6    per ciascun vicino  $w$ 
7      invia il vettore delle distanze  $D_x = [D_x(y): y \in N]$ 
      a  $w$ 
8
9  ciclo
10    attendi (finché vedi cambiare il costo di un
        collegamento verso
11      qualche vicino  $w$  o finché ricevi un vettore delle
        distanze
12      da qualche vicino  $w$ )
13    per ogni  $y$  in  $N$ :
14       $D_x(y) = \min_v \{c(x,v) + D_v(y)\}$ 
15
16    se  $D_x(y)$  è cambiato per qualche destinazione  $y$ 
17      invia il vettore delle distanze  $D_x = [D_x(y): y \in N]$ 
      a tutti i vicini
18
19  ripeti il ciclo indefinitamente

```

il seguente è un esempio di 3

router che ricevono in sincronia le tabelle di instradamento:



Figura 5.6
Intradamento distance vector (DV).

Tabella del nodo x

		costo verso		
		x	y	z
		0	2	7
x	*	0	2	7
y	*	*	*	*
z	*	*	*	*

		costo verso		
		x	y	z
		0	2	3
x	*	0	2	3
y	*	2	0	1
z	*	7	1	0

		costo verso		
		x	y	z
		0	2	3
x	*	0	2	3
y	*	2	0	1
z	*	3	1	0

Tabella del nodo y

		costo verso		
		x	y	z
		0	2	1
x	*	*	*	*
y	*	2	0	1
z	*	*	*	*

		costo verso		
		x	y	z
		0	2	7
x	*	0	2	7
y	*	2	0	1
z	*	7	1	0

		costo verso		
		x	y	z
		0	2	3
x	*	0	2	3
y	*	2	0	1
z	*	3	1	0

Tabella del nodo z

		costo verso		
		x	y	z
		0	2	0
x	*	*	*	*
y	*	*	*	*
z	*	7	1	0

		costo verso		
		x	y	z
		0	2	1
x	*	0	2	1
y	*	2	0	1
z	*	3	1	0

		costo verso		
		x	y	z
		0	2	3
x	*	0	2	3
y	*	2	0	1
z	*	3	1	0

Tempo

alg continuerà a lavorare fino a quando le tabelle non si stabilizzano.

4.3.1 count to infinity

problema: se un nodo si guasta (aumenta il costo del collegamento) accade che gli altri impiegherebbero un tempo infinito ("count to infinity") per accorgersene. ovvero si viene a creare un ciclo nel quale il pacchetto fa avanti e indietro tra i due router.

si può risolvere tramite la cosiddetta "inversione avvelenata" (poison reverse). Ovvero, nel momento in cui avviene lo scambio delle tabella tra nodi; il mittente provvede a "mentire" sul costo della destinazione, il cui next hop è il nodo con cui sta comunicando. fornendo come valore +infinito, in modo tale da prevenire un eventuale ritorno al nodo "mittente".

4.4 comparazione LS e DV

LS	DV
centralizzato	decentralizzato
messaggi inviati in broadcast a tutti	messaggi solo per i vicini
ogni nodo possiede la tabella dei propri link	ogni nodo possiede la tabella di tutte le destinazioni
a causa dei ritardi si possono avere delle oscillazioni	problema conteggio infinito (più di 2 nodi)
i messaggi vengono inviati periodicamente (overhead)	messaggio solo se ci sono variazioni

4.5 Routing information protocol

protocollo di routing usati da ipv4.

basato sul distance vector, la metrica usata è l'hop (numero di collegamenti attraversati).

4.5.1 RIPv1

max numero di hop è 15.

le tabelle di routing sono scambiate ogni 30 secondi.

se un percorso verso un router non viene aggiornato entro 180 secondi, la distanza viene impostate ad infinito. se dopo altri 120s ancora nulla (garbage collector timer) il router (interfaccia) viene eliminata dalla tabella di routign.

questa versione usa 2 tipi di messaggi:

- request: chiede info ai router vicini

- response: invia info sul routing

A routing table contains:

- Destination address
- Distance to destination (in hop)
- Next hop: neighbor router to send packets to
- timeout value (180)
- Garbage-collection time value (120)

Command	Version	Must Be Zero
Address Family Identifier		Must Be Zero
IP Address		
	Must Be Zero	
	Must Be Zero	
Metric		

Command : 1 = request, 2= update.

Version : Protocol version.

Address Family Identifier : questo campo indica la versione del protocollo IP e deve essere sempre 2 (IPv4)

IP Address : destination address (network or subnet)

Metric : hop count (value between 1 and 15)

in quanto non viene mandata la maschera possono essere usati solo gli indirizzi classful

4.5.2 RIPv2

versione aggiornata della precedente, introduce i seguenti cambiamenti:

- **indirizzamento CIDR e VLSM:**

CIDR = uso delle maschere di sottorete

VLSM (variable length subnet masking) = il blocco di n bit della maschera che permette la suddivisione ricorsiva delle reti

- **autenticazione messaggi:**
per evitare invi di tabelle di routing malevole che possono portare a congestionare la rete o ad instradamenti. viene effettuata come in [OSPF](#)
- **specifica next hop**
- **split horizon, poison reverse:**

0	7	15	31
Command	Version	0000	
0xFFFF	Autentication Type		
Autentication			
Address Family Ident.	Route Tag		
IP Address			
Subnet Mask			
Next Hop			
Metric			

Command : 1 = request, 2= update.

Version : Protocol version.

Address Family Identifier : always 2 for IP protocol

Route Tag: Identifying External Routes

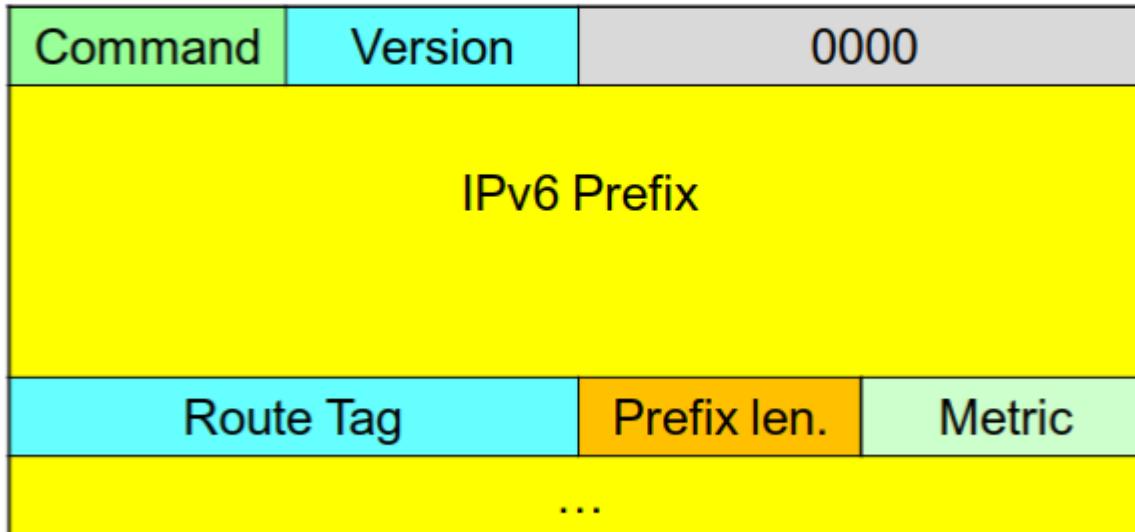
IP Address : destination address (network or subnet)

Metric : hop count (value between 1 and 15)

4.5.3 RIPng

basato su ripv2, ma per uso esclusivo di ipv6.

non ha l'autenticazione.



4.6 AS ed Open Shortest Path First (OSPF)

fino ad ora abbiamo visto la rete come una collezione di router interconnessi. indistinguibili, usavano tutti lo stesso algoritmo. questa visione risulta semplicistica in quanto non tiene conto:

- **scalabilità**: al crescere dei router il tempo richiesto per i vari calcoli diventa proibitivo
- **autonomia amministrativa**: internet è un insieme di ISP che gestiscono autonomamente le proprie reti o le nascondono all'esterno. l'ideale sarebbe la gestione autonoma con la possibilità di connessione esterna

si possono organizzare i router in sistemi autonomi (**Autonomous systems**, AS) gruppi di router posti sotto lo stesso controllo amministrativo che usano lo stesso protocollo (AS routing protocol).

4.6.1 OSPF

creato per rimpiazzare RIP.

generalmente gli ISP usano questo protocollo nei loro AS, un protocollo link state che usa il flooding (broadcast) per inviare info riguardo lo stato e l'alg dijkstra per il percorso minimo.

I costi dei cammini possono essere determinati dall'amministratore oppure in modo inversamente proporzionale rispetto alla banda del collegamento.

4.6.1.1 struttura

Questo protocollo è formato da tre parti differenti:

- **HELLO**: scoperta e verifica dei vicini. Ad ogni cambiamento dei collegamenti oppure periodicamente (ogni 30 minuti, per garantire robustezza) un router manda ai suoi link un pacchetto speciale per sapere chi c'è dall'altra parte e calcolarsi il costo dei link.
- **EXCHANGE**: sincronizzazione iniziale del DB. Utilizzato per l'inizializzazione di un nuovo router. Questo funziona a richiesta.
- **FLOODING**: aggiornamento del DB. Funziona a offerta. Periodicamente il router prepara le informazioni sui suoi link e li manda ad i suoi link. Si basa sempre su Dijisktra.

ogni router ha al suo interno un db che contiene i **link state record**. per aggiornare i campi vengono inviati dei **link state advertisement (LSA)**. i cambiamenti avvengono:

- Quando un router trova un nuovo router adiacente
- Quando un router/link si guasta
- Quando il costo di un link cambia
- Periodicamente, come detto in precedenza, ogni 30 minuti

4.6.1.2 header OSPF

Version	Type	Message Length
Router ID		
Area ID		
Checksum	Autentication Type	
Autentication Data		
Rest of the OSPF Message		

Version: OSPF version number, which is 2 for OSPFv2.

area id: per quanto riguarda [strutturazione a gerarchie](#)

il type può essere:

- **Hello**: periodic transmission for Neighbor Discovery
- **Database Description**: used for the exchange of link-state information of each router
- **LS Request**: requesting specific parts of a neighbor's link-state database
- **Link-State Update**: Transfers link-state ads to neighbors
- **Link-State Acknowledgments**: Send an acknowledgment of receipt of a link-state update

Authentication type: ranging from 0 to 2, corresponding to non-authentication, simple (plaintext) authentication, and MD5 authentication, respectively.

Authentication data: Information determined by authentication type. It is not defined for authentication type 0. It is defined as password information for authentication type 1, and defined as Key ID, MD5 authentication data length, and sequence number for authentication type 2.

[more info](#).

4.6.1.3 sicurezza

in quanto gli scambi di info sono effettuati in broadcast è necessaria questa caratteristica.

gli scambi tra router sono autenticati, per evitare immissioni malevoli nelle tabelle di routing. le autenticazioni disponibili sono:

1. semplice:

si configura una psw (in chiaro) che deve essere condivisa nei pacchetti OSPF inviati

2. autenticazione MD5

si basa su chiavi segrete condivise, configurate in ogni router. in pratica prima di mandare un pacchetto, del suo contenuto viene (prima) calcolato l'hash MD5 e (dopo) inviato. nel pacchetto è presente un intestazione con il risultato.

il destinatario riceverà in chiaro il payload delle nuove info di routing su cui effettuerà l'hashing (con la chiave che hanno in comune tutti i nodi dell'AS), se il risultato è uguale a quello presente nell'intestazione è top.

inoltre sono adoperati anche i numeri di sequenza, ovvero un contatore incrementato ad ogni invio di nuove info. Il ricevente tiene traccia dell'ultimo numero di sequenza accettato e accetta solo gli aggiornamenti con numeri di sequenza successivi. Questo previene l'utilizzo di aggiornamenti OSPF vecchi o duplicati). like [livello di trasporto](#).

4.6.1.4 percorsi con lo stesso costo

quando più percorsi hanno lo stesso costo. OSPF consente di usarli senza dover sceglierne uno in particolare

4.6.1.5 unicast e multicast integrato

viene esteso OSPF con il protocollo MOSPF (multicast ospf).

4.6.1.6 strutturazione a gerarchie

un elemento che favorisce la scalabilità è la possibilità di strutturare i vari sistemi autonomi in aree, che eseguono diversi tipi di algoritmo di instradamento ospf.

i router delle stesse aree si scambiano info ospf.

per ogni area sono predisposti dei router di frontiera (area border router) per [comunicare con le altre aree](#).

Per questo è presente l'area dorsale che si occupa di gestire il traffico tra le varie aree.

4.7 Instradamento tra AS: BGP

border gateway protocol (bgp), rappresenta lo standard de facto...DE FACTO dei protocolli di instradamento tra AS in internet. Ovvero la determinazione delle tabelle di inoltre esterne agli AS.

in bgp le destinazioni non hanno specifici indirizzi, ma prefissi che rappresentano una sottorete.

Quindi le occorrenze saranno del tipo:

(prefisso, sottorete)

ex:

131.17.68.0/22

quindi i ruoli di bgp saranno:

- informare la rete sui prefissi presenti. ogni sottorete dovrà identificarsi in rete ("io esisto") per non rimanere isolata.
- determinare i percorsi ottimi verso le sottoreti.

4.7.1 informazione percorsi

router gateway: router di confine che permette la connessione con altri AS

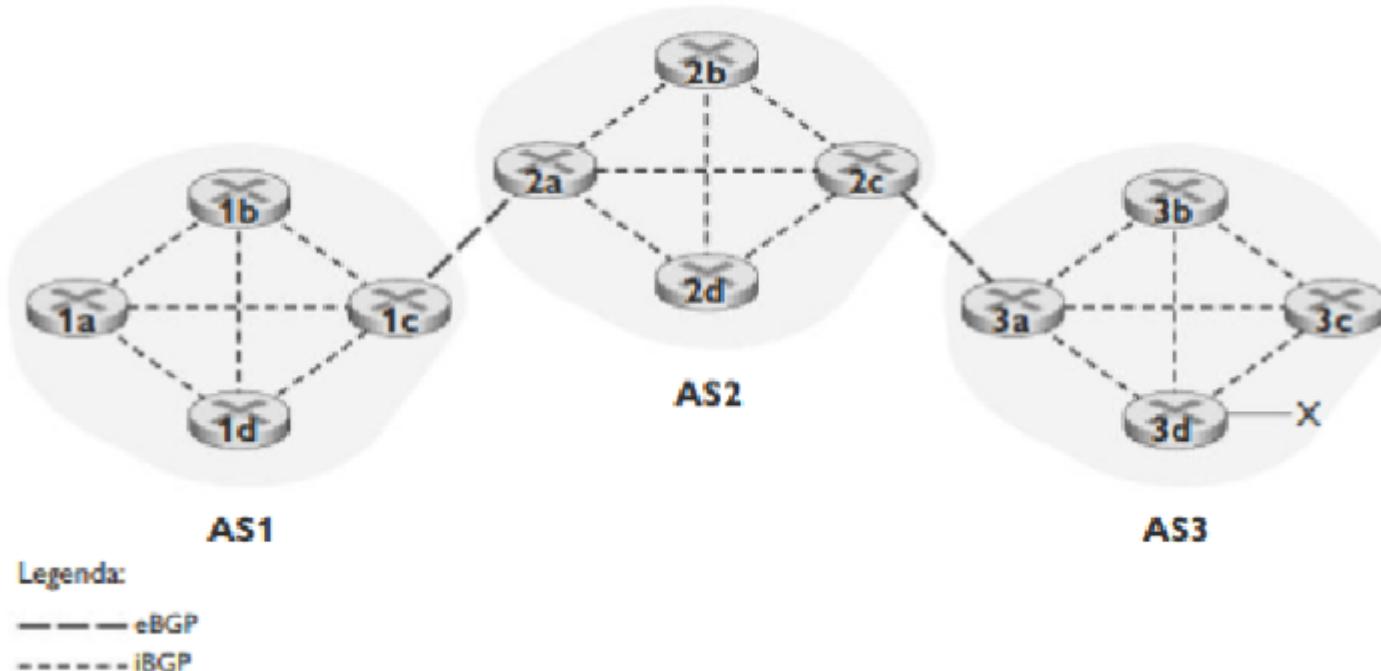
router interno: connessione interna all'AS

ogni router negli AS svolte tutte e due i ruoli.

Lo scambio di info in BGP avviene tramite connessioni TCP semi-permanenti (non viene chiusa dopo ogni trasferimento di dati) sulla porta 179. vengono dette **sessioni BGP**, in particolare:

- **eBGP**, sessione BGP esterna: comunicazione tra due AS
- **iBGP**, sessione BGP interna: all'interno dell'AS

Figura 5.9 Connessioni eBGP e iBGP.



4.7.2 distribuzione percorsi verso le sottoreti

attributi BGP: info aggiuntive inviate con il prefisso

rotta/route: termine usato per indicare l'invio di prefisso + attributi

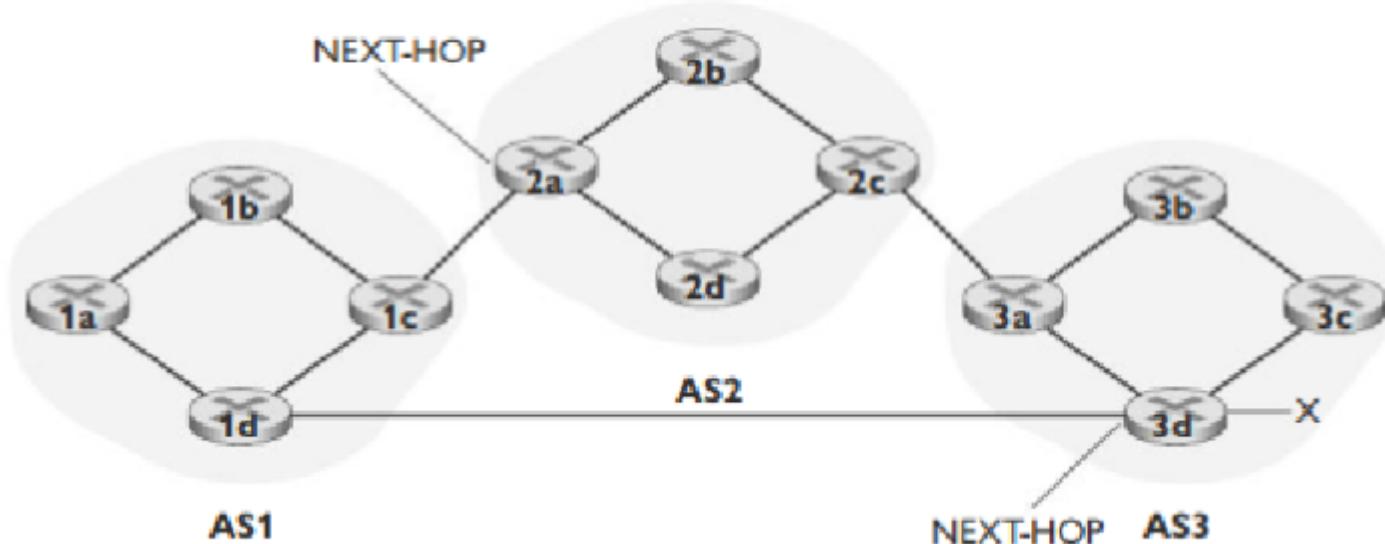


Figura 5.10 Rete con aggiunta del collegamento tra AS1 e AS3.

principali attributi:

- **AS-PATH**: percorso attraverso cui le route BGP hanno attraversato una sequenza di Autonomous System (AS) durante la loro propagazione attraverso Internet. ogni volta che una rotta attraversa un AS, questo aggiunge un suo identificativo. usato principalmente per prevenire loop di routing (se un router vede il proprio as rifiuta il pacchetto).
ex: percorso as1 verso x. avremo due as-path: (1) AS2,AS3; (2) AS3
- **NEXT-HOP**: indirizzo ip interfaccia per raggiungere la sottorete
ex: guardare i next hop sopra.

- **AS Border Router (ASBR):** Router connected to other stand-alone systems
- **BGP speaker:** router that supports the BGP protocol (a BGP speaker does not necessarily coincide with an AS border router)
- **BGP Neighbors:** pair of BGP speakers exchanging inter-AS routing information
 - **interiors:** if they belong to the same AS
 - **exterior:** if they belong to different ASs

4.7.2.1 algoritmo hot potato

si cerca di trovare il percorso di costo minore verso il next-hop. si dice patata bollente in quanto si cerca di inviare preoccupandosi solo della rotta immediata.

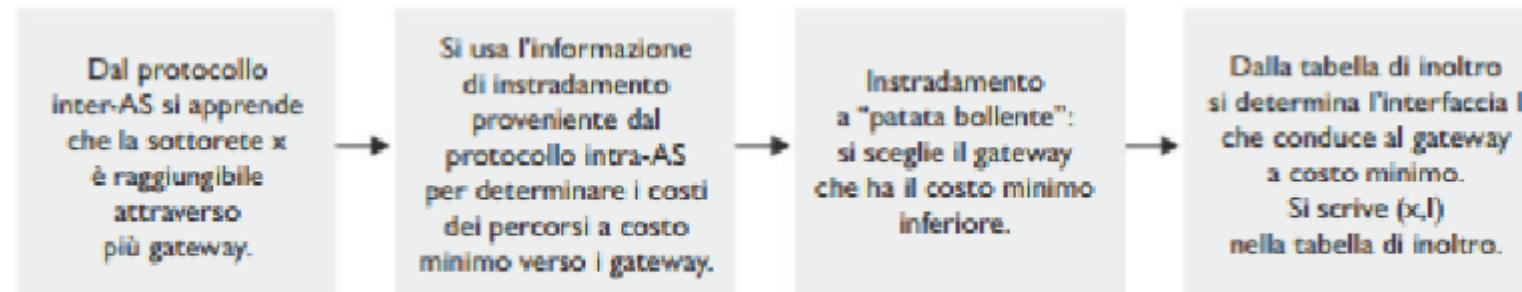


Figura 5.11 Passi per aggiungere una destinazione esterna al sistema autonomo in una tabella di inoltro di un router.

4.7.2.2 algoritmo di selezione delle rotte

in realtà bgp complica hot potato, fornendo come input l'insieme di rotte verso una determinata sottorete per poi andare ad individuarne una (eliminando le altre).

in particolare:

1. alle rotte viene assegnato un valore di **preferenza locale**. impostata dal router o appresa da altri nell'as. si seleziona la preferenza maggiore.
2. tra le preferenze locali si prende quella con AS-PATH più breve (like bellman)
3. si prende la rotta con il next-hop più vicino (costo minore)

4.8 SDN

in più?

4.9 Internet Control Message Protocol (ICMP)

protocollo usato per scambio di info a livello di rete, generalmente per scambio di errori.

ex: ping, destinazione irraggiungibile.

si posiziona al di sopra del livello di rete in quanto viene incapsulato all'interno del datagramma. è il payload del pacchetto (like udp, tcp).

4.9.1 v4

i messaggi sono strutturati in un campo **tipo e codice**:

Tipo ICMP	Codice	Descrizione
0	0	risposta echo (a ping)
3	0	rete destinazione irraggiungibile
3	1	host destinazione irraggiungibile
3	2	protocollo destinazione irraggiungibile
3	3	porta destinazione irraggiungibile
3	6	rete destinazione sconosciuta
3	7	host destinazione sconosciuto
4	0	riduzione (controllo di congestione)
8	0	richiesta echo
9	0	annuncio di un router
10	0	scoperta di un router
11	0	TTL scaduto
12	0	intestazione IP errata

Figura 5.19 Tipi di messaggio ICMP.

4.9.2 v6

Type	Meaning
1	Destination Unreachable
2	Packet Too Big
3	Time Exceeded
4	Parameter Problem
128	Echo Request
129	Echo Reply
130	Group Membership Query
131	Group Membership Report
132	Group Membership Reduction
133	Router Solicitation
134	Router Advertisement
135	Neighbor Solicitation
136	Neighbor Advertisement
137	Redirect
138	Router Renumbering

4 - Data link

388-448

1 Livello Data Link

nodo = generico dispositivo al livello collegamento

collegamenti/link = canali di comunicazione che collegano nodi

su ogni collegamento il data link layer (DLL) si occupa di encapsulare il datagramma in un frame di collegamento (link layer frame) per poi inviarlo sul link.

si può pensare al DLL con un'analogia sui viaggi. se devo fare un viaggio, posso prendere diversi mezzi, ognuno responsabile del mio trasporto.

si può pensare alle persone come il **datagramma**, i tratti di viaggi come **collegamenti** ed il metodo di trasporto come il **protocollo**.

1.1 servizi offerti

- **framing**

incapsulare i datagrammi di rete aggiungendo header speciali utili ad esempio per la validazione dei dati.

- **accesso al collegamento**

gestire l'accesso al mezzo fisico condiviso. tramite MAC (medium access control) vengono specificate le regole con cui immettere i frame nel link.

- **consegna affidabile:**

nonostante sia presente a livello trasporto, viene consigliata nei link soggetti ad un elevato tasso di errore (wireless); in modo da intervenire localmente, evitando una ritrasmissione da parte dei lvl superiori.

viene fatto uso sempre di ack

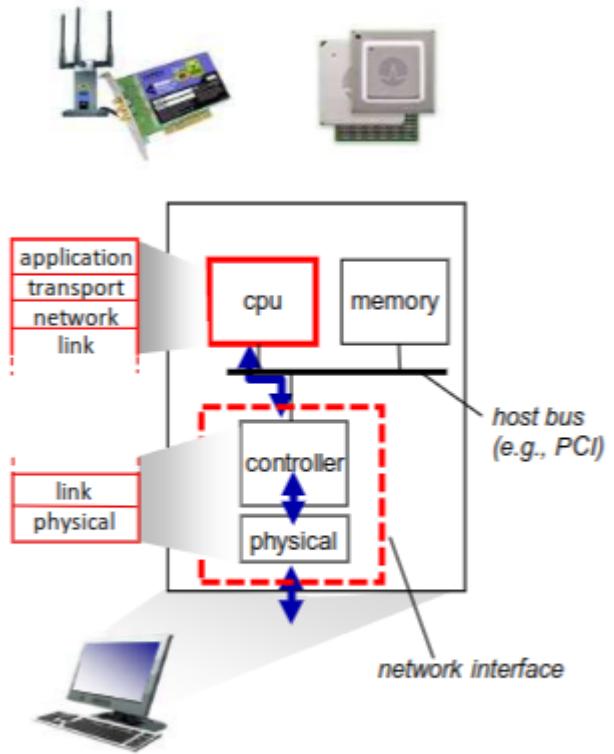
- **rilevazione e correzione degli errori**

più efficace di quelli trattati al lvl di [trasporto](#) o [rete](#) in quanto implementato via hw. tramite l'aggiunta di bit di controllo è consentita la rilevazione e risoluzione degli errori.

1.2 dove viene implementato

nelle [line card](#) dei router tramite un adattatore, la scheda di rete (network interface controller, NIC).

si occupa di molti servizi sopra eletanti (framing, accesso link, errori).

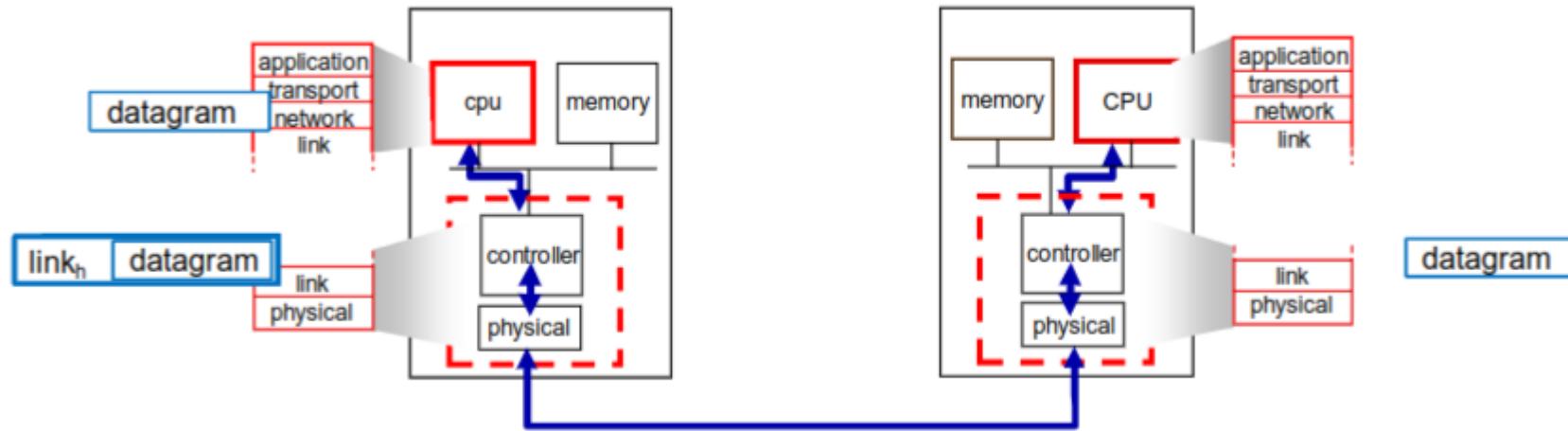


mittente:

- prende un datagramma creato e memorizzato nella memoria host, incapsula in frame w/campi intestazione (bit)
- invia il frame sul link

destinatario:

- riceve frame & controlla errori
- estrae datagramma e lo passa al lvl superiore.



btw il DLL è una combinazione tra hw e sw. il sw lo troviamo:

- nella gestione degli interrupt all'arrivo di una frame ad esempio
- gestione di errori
- passaggio al lvl di rete

1.3 framing

supponiamo di lavorare su un canale che ammette solo i valori di 0 ed 1 (luce).

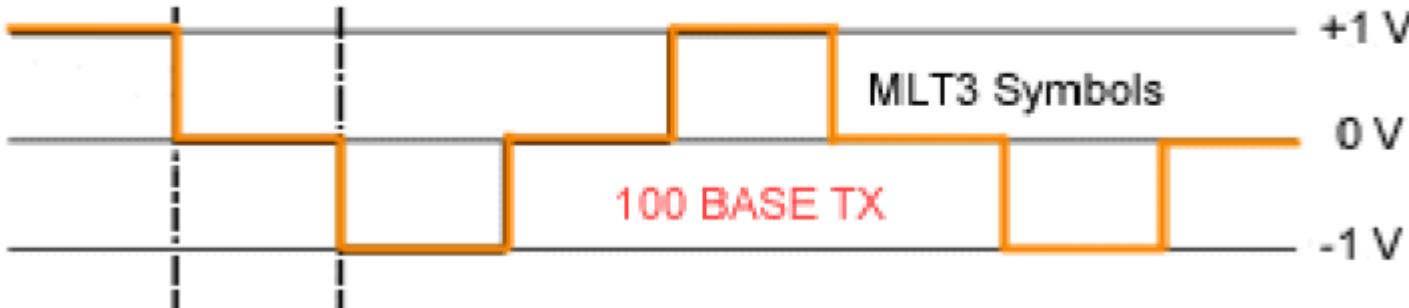
si crea il problema di identificare un assenza di trasmissione dall'invio del valore 0



1.3.1 livelli

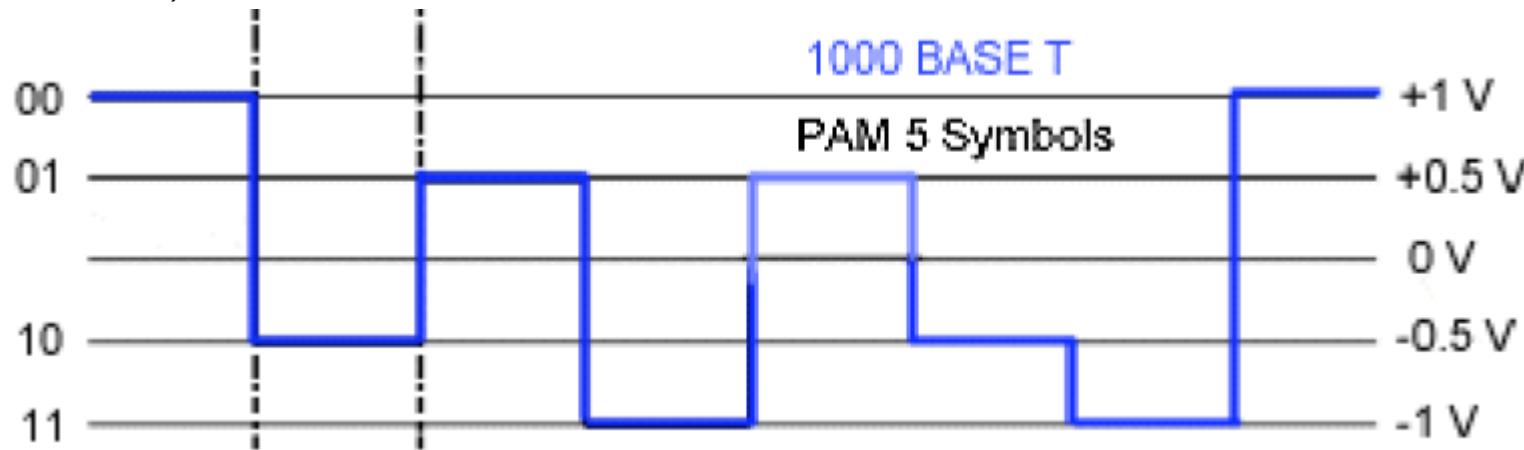
- 3 livelli:
si potrebbe pensare ad un sistema a 3 livelli. se dopo un bit 1 è presente un ulteriore bit 1, indica il cambio di livello.

si inizia a creare una ridondanza che provoca uno spreco di $\frac{1}{3}$ del potenziale.



- 5 livelli:

c'è anche a 5. in questo caso il cosiddetto **burst range**, ovvero le info inviabili in un dato intervallo è uguale a 2 (prima era solo 1, il canale era subito saturo).



queste implementazioni non sono applicabili nella fibra ottica.

1.3.2 4B5B encoding

implementiamo una tabella di conversione da 4 bit a livello logico a 5 bit lvl fisico.

si aggiunge un bit di ridondanza, quindi con uno spreco del 25%.

permette di evitare lunghe catene di bit 0, quindi permette di ovviare al problema dell'identificazione della trasmissione.

vantaggi:

1. nella conversione non ci sono sequenze solo di bit 0 o 1 con lunghezza > 3
 1. se le troviamo => mancanza di comunicazione
2. la conversione viene fatta in modo che statisticamente si generino meno ripetizioni.
3. in questo modo (se non riesco a tradurre) ho fatto una sorta di rilevazione errore.

la codifica consente di ottenere 32 combinazioni differenti:

- 16 combinazioni saranno usate per la traduzione diretta mentre
- delle rimanenti 16
 - solo 6 verranno utilizzate, il resto saranno scartati. Delle 6 combinazioni utilizzate non per la traduzione, 4 saranno utilizzate per indicare l'inizio (SSD) e la fine (ESD) della comunicazione. Un frame non può iniziare dunque per zero in quanto entrambe le combinazioni usate per SSD iniziano con 1.

si usa la seguente tabella di conversione:

Nome	4B	5B	Descrizione
0	0000	11110	hex data 0
1	0001	01001	hex data 1
2	0010	10100	hex data 2
3	0011	10101	hex data 3
4	0100	01010	hex data 4
5	0101	01011	hex data 5
6	0110	01110	hex data 6
7	0111	01111	hex data 7
8	1000	10010	hex data 8
9	1001	10011	hex data 9
A	1010	10110	hex data A
B	1011	10111	hex data B
C	1100	11010	hex data C
D	1101	11011	hex data D
E	1110	11100	hex data E
F	1111	11101	hex data F
I	-NONE-	11111	Idle
J	-NONE-	11000	SSD #1
K	-NONE-	10001	SSD #2
T	-NONE-	01101	ESD #1
R	-NONE-	00111	ESD #2
H	-NONE-	00100	Halt

problema: di questa codifica è che essendo i bit trasmessi a gruppi di 5, quindi non pari, avremo sempre un bit 0 o 1 in più.

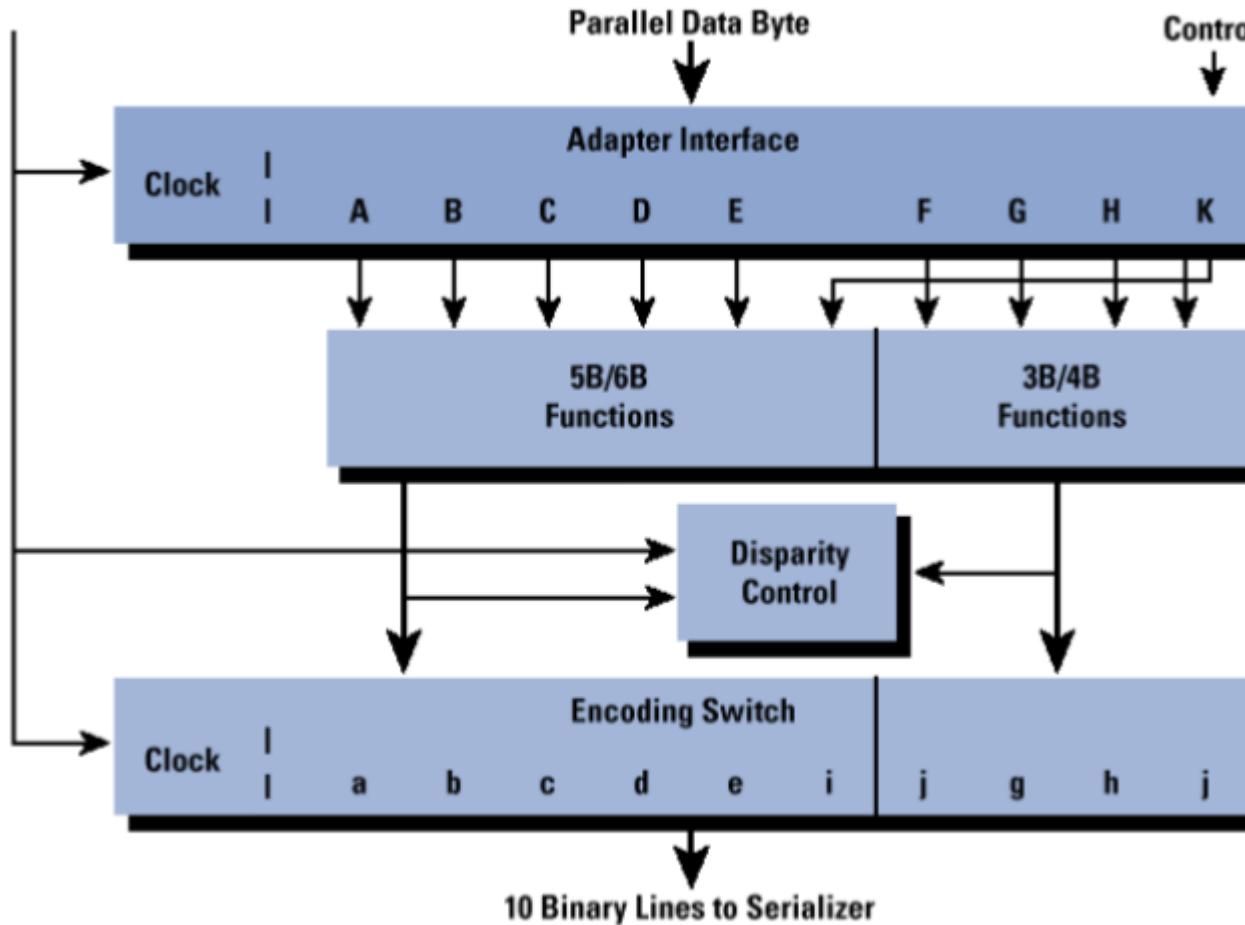
- questo scompenso causerà una polarizzazione elettrica del cavo e ci sarà un trasferimento di energia da un dispositivo all'altro mentre dovrebbero essere trasportati soltanto dati. E' bene evitare di trasferire l'energia ed è ciò che la codifica 8b10b permette di evitare.

1.3.3 8B10B

simile 4b5b, usata in vari standard quali SATA, USB 3.0, HDMI, alcune versioni di Gigabit Ethernet, etc...

vantaggi:

- introdotto per ovviare al problema della polarizzazione di 4b5b. è quindi elettricamente neutro, ovvero si cerca il bilanciamento nel numero di bit.
 - **In un sistema fisico a 2 livelli:** permette di avere una sequenza più ordinata di variazioni tra 1-0 e 0-1, in maniera da sincronizzare meglio il clock e far capire al destinatario con più certezza come interpretare il segnale che arriva.
 - **In un sistema fisico a 3 livelli:** permette di avere un numero uguale di momenti di tensione positiva su cavo con momenti di tensione negativa e quindi avere un bilancio energetico nullo tra i due dispositivi.



fasi:

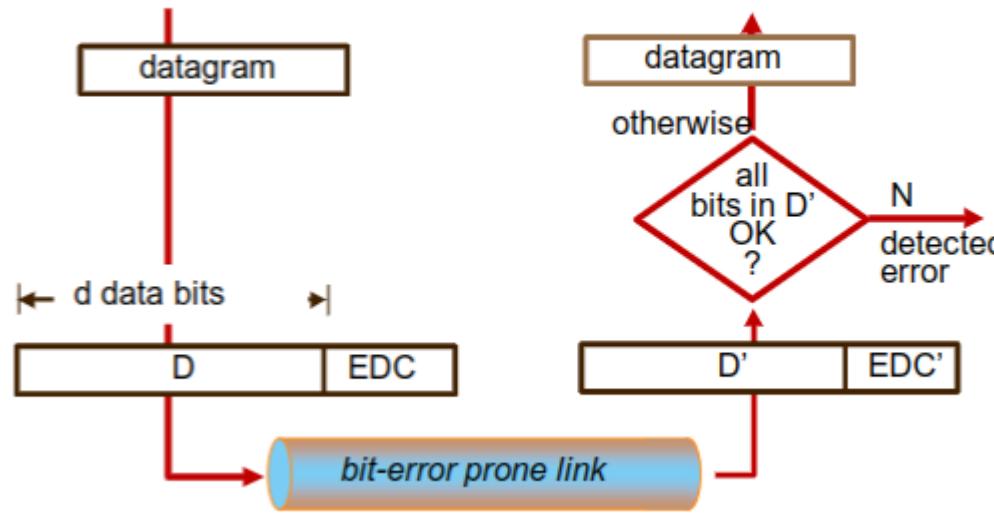
1. Partiamo da 8 bit che vengono suddivisi in un primo gruppo da 5 bit e un secondo gruppo da 3 bit.
2. La prima sequenza di 5 bit viene trasformata in una sequenza di 6 bit
3. La sequenza di 3 bit viene trasformata in 4 bit.

1.4 rilevazione errori

EDC = error detection and correction, bit di controllo

D = dati inviati, protetti da edc

il meccanismo di base sulla rilevazione è il seguente:



1.4.1 parity check

si aggiunge un bit finale ed il suo valore si sceglie in modo da avere un numero di bit 1 pari.

il destinatario dovrà quindi contare le occorrenze, se sono dispari c'è un errore.

btw c'è un problema nel caso di un numero pari di errori.

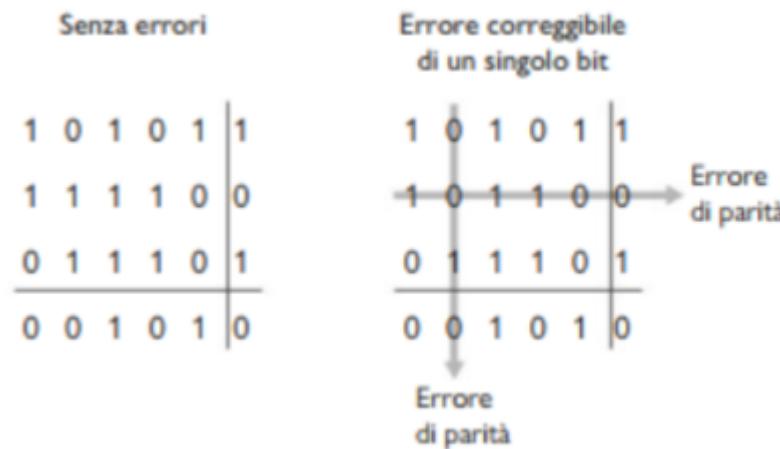
inoltre statisticamente è stato rilevato che gli errori si presentano in **burst** (gruppi di bit consecutivi). la probabilità che un burst non sia rilevato si aggira al 50%.

1.4.1.1 metodo matrice

si dispongono i bit su una matrice e si confrontano i bit di parità orizzontali con quelli verticali (vengono aggiunti dal mittente all'invio).

si suppone che si verifichi un solo errore.

ad esempio:



con questo metodo possiamo anche andare a vedere dove viene corrotto il bit.

questa capacità è detta **forward error correction** (FEC).

questi metodi permettono un notevole risparmio, equivalente all'RTT necessario per far arrivare il **NAK** per poi ritrasmettere il pacchetto.

usata molto spesso in dispositivi di registrazione audio.

1.4.2 checksum

pag 396

1.4.3 cyclic redundancy check (CRC)

codici di controllo a ridondanza ciclica.

si assume come funzione hash la divisione. si assume che il dato da mandare sia un polinomio i cui coefficienti sono i bit, in modo da poter effettuare il controllo sulla divisibilità

dati d bit che costituiscono il dato D da inviare.

ci si accorda, generalmente tramite standard, su una sequenza di $r + 1$ bit detta **generatore** G . è necessario che il bit più a sx sia ad 1.

MITTENTE:

1. si shifta a sx D di r bit (attenzione: il generatore è $r + 1$)

$$D' = D \cdot 2^r$$

2. calcolo R:

$$R = G \text{ } XOR \text{ } D'$$

3. invio la stringa D + R

$$M = 101101 \quad G = 1101 \quad |G| - 1 = 4 - 1 = 3$$

Calcolare il CRC.

101101 000

1101

011001 000

1101

000011 000

11 01

000000 010
resto

invierà quindi la stringa di bit: 101101**10**10

DESTINATARIO:

verifica che la stringa ricevuta sia divisibile per il generatore senza resto

$$\underbrace{((D \cdot 2^r) \text{ } XOR \text{ } R)}_{D'} \text{ mod } G = 0$$

verifico che il resto sia zero

```
101101010|1101  
1101  
----  
01100  
1101  
----  
0001101  
1101  
----  
00000 => [no error]
```

ci sarà sempre una possibilità di collisione, btw più aumento la ridondanza (grandezza generatore) più sarà ridotta.

sono stati definiti dei generatori standard di 8,12,..., 32 bit.

ad esempio ATM ne usa una ad 8bit.

1.4.4 distanza hamming

date 2 codeword, la distanza di hamming è il numero che identifica le occorenze di bit diversi.

10001100

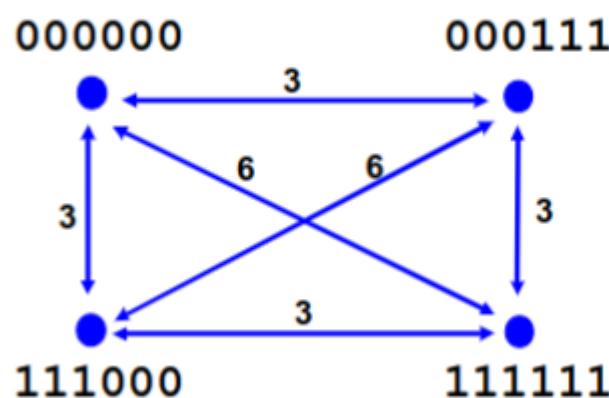
11000100

01001000 d = 2

si dice vocabolario un set di codeword.

si vocabolario completo se contiene 2^n codeword.

la distanza in un vocabolario si calcola prendendo la distanza minima tra due codeword.

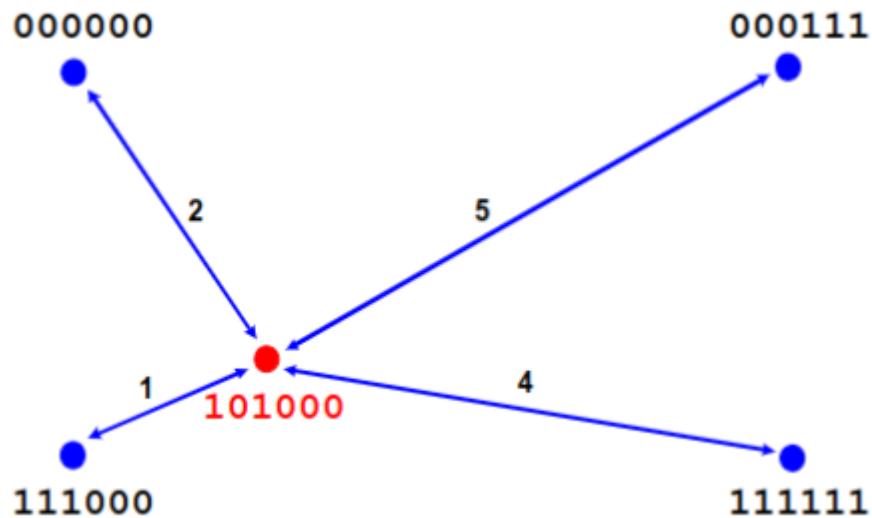


la correzione degli errori si basa sulla seguente assunzione:

$$\Pr(\text{errore} = 1 \text{ bit}) > \Pr(\text{errore} = 2 \text{ bit}) > \dots$$

un primo approccio di correzione potrebbe essere:

data una codeword, calcolo la distanza di hamming con tutto il vocabolario e prendere la word con il valore minore.



questo approccio inizia a funzionare con vocabolari a $d=3$, in quanto con distanza due sorgono delle collisioni (word con stessa distanza dalla word errata).

ex:

vocabolario:

000
011
101

invio la word 001 e non capisco se corrisponde a 011 oppure 101

in general:

- per poter correggere e errori è necessario un vocabolario con distanza $d = 2e + 1$,
- per rilevare gli errori è necessario un vocabolario con distanza $d = e + 1$.

1.4.4.1 formula teorica (il nulla cosmico fritto)

dati m bit, quindi con 2^m combinazioni valide, vorrei creare un codice con correzione singola degli errori con il minor numero di bit ridondanza (r).

quindi:

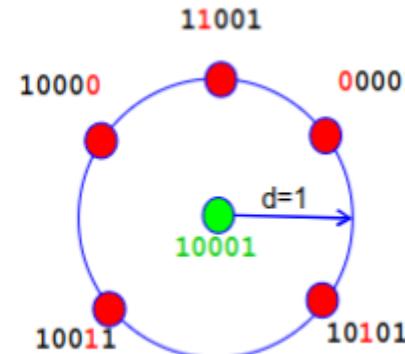
- come calcolo r?
- come costruisco il codice?

sia $n = m + r$, le combinazioni possibili sono 2^n ma quelle valide solo 2^m

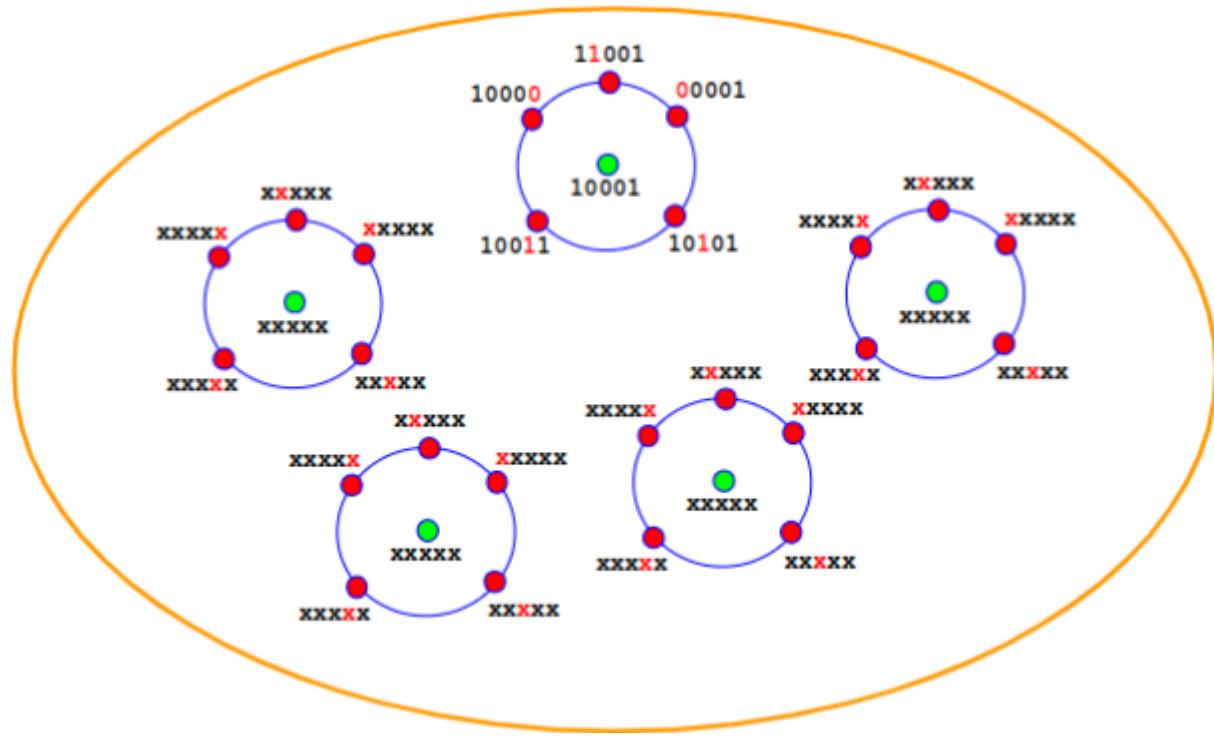
per ogni keyword valida, cambiamo i singoli bit in modo da formare keyword a distanza 1

$n+1$ {
10001
00001
11001
10101
10011
10000}

n invalid codeword at distance 1



quindi la visione di insieme sarà la seguente:



quindi avremo m insiemi formati da $n+1$ elementi, di cui quelli validi saranno 2^m ed il totale 2^m .
per cui otteniamo:

$$(n + 1)2^m \leq 2^n$$

$$(m + r + 1)2^m \leq 2^{m+r}$$

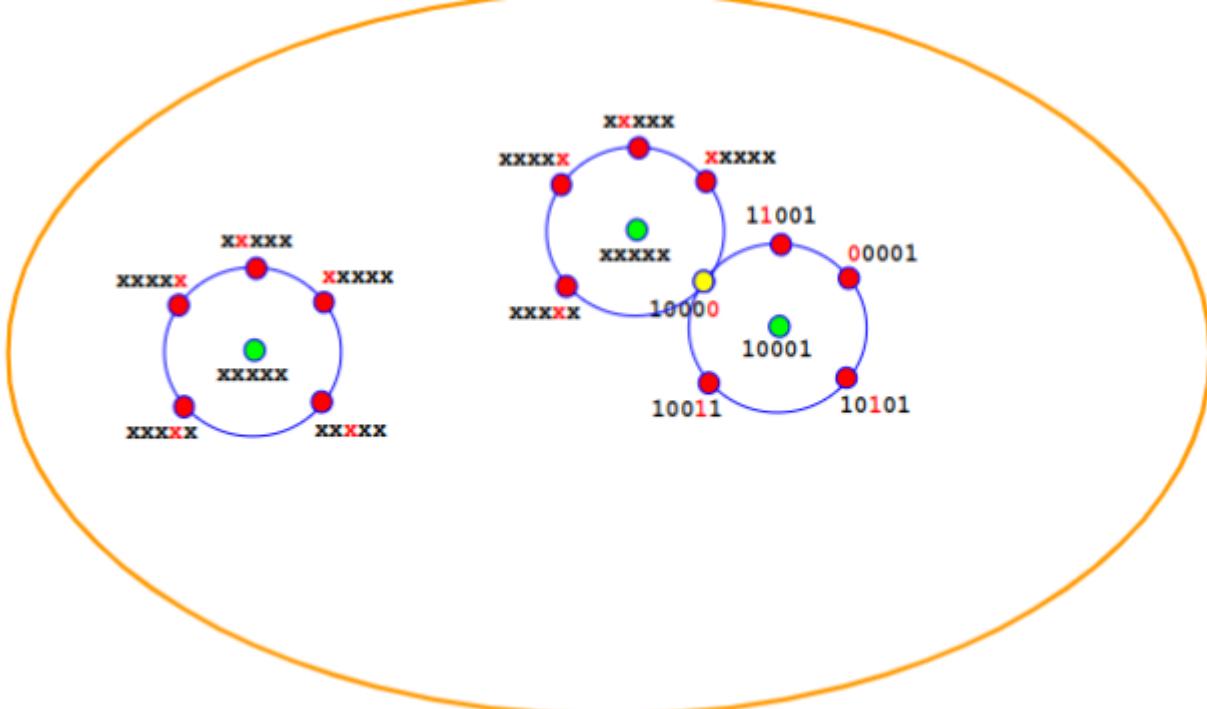
$$(m + r + 1) \leq 2^r$$

$$m + 1 \leq 2^r - r$$

che è vera solo per $d=3$ (assioma), altrimenti si verrebbero a creare una situazione in cui conto due volte e si raggiunge una cardinalità maggiore.

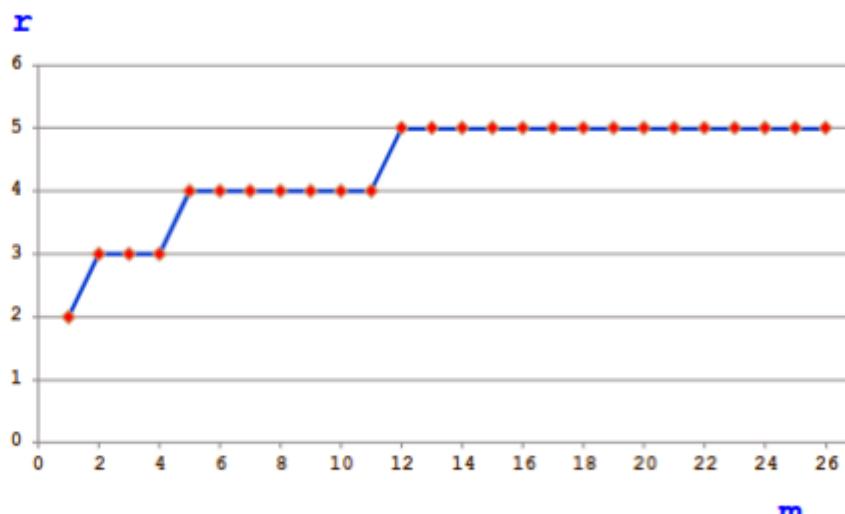
dimostrazione alla viola:

per $d=3$, si suppone per assurdo come in figura che ci sia un keyword a distanza 1. assurdo, sarebbe a dire che vocabolario non è a distanza 3.



in generale possiamo diagrammare i valori di m ed n

m	r	n	m	r	n
1	2	3	14	5	19
2	3	5	15	5	20
3	3	6	16	5	21
4	3	7	17	5	22
5	4	9	18	5	23
6	4	10	19	5	24
7	4	11	20	5	25
8	4	12	21	5	26
9	4	13	22	5	27
10	4	14	23	5	28
11	4	15	24	5	29
12	5	17	25	5	30
13	5	18	26	5	31
			27	6	33

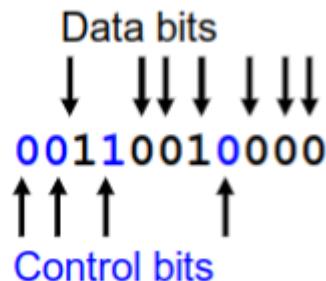


m	r
1	2
2 - 4	3
5 - 11	4
12 - 26	5
27	6

ci vuole il colpo del palazzo.

1.4.4.2 costruzione codice

costruiamo il codice intervallando i bit di dati con i bit di controllo, posizionando questi nelle posizioni a potenza di 2 (0,1,2,4,...) partendo da sx.



il controllo si basa sul vedere i bit di dati come somma di bit di controllo. in seguito tramite operazione di xor sui bit di controllo, si va a controllare la presenza di un eventuale errore.

quindi avremo i bit di dati rappresentati da questa tabella:

3 = 1+2

5 = 1 +4

6 = 2+4

7 = 1+2+4

9 = 1 + 8

10 = 2 +8

11 = 1+2 +8

12 = 4+8

13 = 1 +4+8

14 = 2+4+8

15 = 1+2+4+8

per ogni bit di controllo, si effettua lo xor rispetto al valore contenuto nella posizione in cui sono stati inseriti nella precedente tabella.

$b_1 = 3 \otimes 5 \otimes 7 \otimes 9 \otimes 11 \otimes 13 \otimes 15$

$b_2 = 3 \otimes 6 \otimes 7 \otimes 10 \otimes 11 \otimes 14 \otimes 15$

$b_4 = 5 \otimes 6 \otimes 7 \otimes 12 \otimes 13 \otimes 14 \otimes 15$

$b_8 = 9 \otimes 10 \otimes 11 \otimes 12 \otimes 13 \otimes 14 \otimes 15$

ad esempio il valore 1 è presente nel 3,5,7,....

ad esempio:

```
--stringa di dati
```

```
10010001100
```

```
--posizioniamo i bit di controllo
```

```
xx1x001x0001100
```

```
bit_1 = 1 XOR 0 XOR 1 XOR 0 XOR 0 XOR 1 XOR 0 = 1
```

```
bit_2 = 1 XOR 0 XOR 1 XOR 0 XOR 0 XOR 0 XOR 0 = 0
```

```
bit_4 = 0 XOR 0 XOR 1 XOR 1 XOR 1 XOR 0 XOR 0 = 1
```

```
bit_8 = 0 XOR 0 XOR 0 XOR 1 XOR 1 XOR 0 XOR 0 = 0
```

```
-- costruiamo stringa finale
```

```
101100100001100
```

1.4.4.3 controllo codice

```
-- supponiamo l'arrivo della seguente stringa
```

```
101100100001100
```

```
-- verifichiamo i valori dei bit di controllo
```

```
bit_1 = 1 XOR 0 XOR 1 XOR 0 XOR 0 XOR 1 XOR 0 = 1
```

```
bit_2 = 1 XOR 0 XOR 1 XOR 1 XOR 0 XOR 0 XOR 0 = 1 [ERRORE]
```

```
bit_4 = 0 XOR 0 XOR 1 XOR 1 XOR 1 XOR 0 XOR 0 = 1
```

```
bit_8 = 0 XOR 1 XOR 0 XOR 1 XOR 1 XOR 0 XOR 0 = 1 [ERRORE]
```

notiamo che è presente un errore alla posizione 2 ed 8.

consulto la "tabella delle somme" e noto che queste posizioni di controllo coinvolgono il bit 10.

#attentionplis l'errore può ricadere anche nelle posizioni dei bit di controllo.

#attentionplis stiamo sempre girando intorno all'ipotesi che ci sia solo un bit errato (probabilisticamente), sennò tutto questo non funziona.

per controllare gli errori uso il contatore che incremento (con il valore della posizione, se errore cado a posizione 16 => $c+=16$) ad ogni errore. il valore finale corrisponderà al bit errato (?)

#attentionplis conviene fare prima CRC e poi hamming. nel caso contrario no sense

1.4.4 errori localizzati

spesso capita che gli errori siano concentrati in zone.

possiamo operare come segue, avendo questa stringa:

```
xx1x110xx0x011xx0x110xx1x000xx0x011xx1x100xx0x001xx0x11  
0xx1x011xx1x110xx1x001xx0x011
```

dividiamo le word e trasmettiamo per colonna:

```
xx1x110  
xx0x011  
xx0x110  
xx1x000  
xx0x011  
xx1x100  
xx0x001  
xx0x110  
xx1x011  
xx1x110  
xx1x001  
xx0x011
```

invieremo quindi:

```
xxxxxxxxxxxxxxxxxxxxxx100101001110xxxxxxxxxxxx101001  
010100111010011101010010101011
```

supponiamo ci sia il seguente errore, potremmo operare applicando hamming, in quanto la "raffica" di errori si distribuisce su codeword differenti.

xxxxxxxxxxxxxxxxxxxxx100101001110xxxxxxxxxxxx101001
01010000010111101010010101011

xx1x100
xx0x001
xx0x100
xx1x010
xx0x001
xx1x110
xx0x011
xx0x110
xx1x011
xx1x110
xx1x001
xx0x011

questo approccio è abbastanza costoso in quanto richiede l'uso di sistemi di buffer.

1.4.5 conclusioni

Ricapitolando, per correggere e rilevare è necessaria ridondanza, solo che per correggere è necessaria molta più ridondanza e si perde del tempo; per cui bisogna capire quando è opportuno correggere e quando rilevare soltanto.

conviene sempre? => **NO**

Nella *fibra ottica* ad esempio c'è la possibilità di $1/10^{13}$ bit errati, in questo caso la probabilità è così bassa che non ha neanche senso parlare di correzione d'errore.

potrei ad esempio applicare CRC.

Se al posto della fibra abbiamo il wifi dove il rapporto è 1/1000, se usiamo il metodo di prendo e butto, allora non comunichiamo più; in questo caso ha senso inserire un meccanismo di correzione.

si potrebbe anche pensare ad un approccio più robusto (ridondante):

aggiungere CRC + codifica_hamming .

```
[data] -> CRC --trasmissione--> HAMMING ---
```

```
|  
|
```

```
[data] <- CRC <--trasmissione-- HAMMING <--
```

1.5 accesso al canale

nel momento in cui uso reti broadcast rispetto a point-to-point sorge il problema dell'accesso multiplo al canale.

broadcast può avere più trasmittenti e riceventi allo stesso momento connessi. è necessario definire dei protocolli per ovviare ai vari problemi (collisioni, turnazione degli invii, ecc...).

possiamo suddividere i **protocolli di accesso multiplo** in 3 categorie:

- protocolli suddivisione del canale (channel partitioning)
- protocolli ad accesso casuale (random access)
- protocolli a rotazione (taking-turn)

1.5.1 protocolli a suddivisione del canale

- **tdma** (time division multiple access).
si suddivide il canale in intervalli temporali. durante ogni intervallo può parlare solo un nodo.

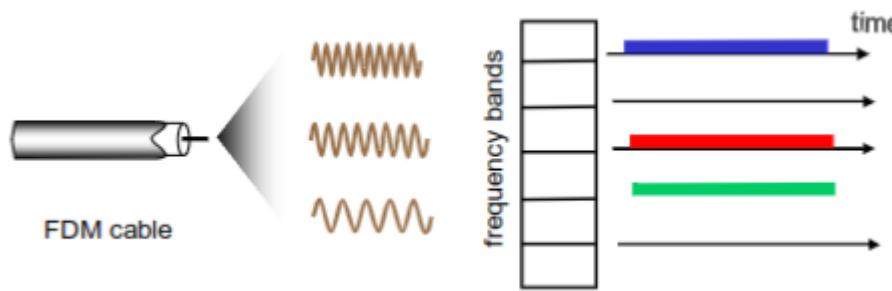


la grandezza di ogni slot è determinata in genere per inviare un singolo pacchetto.

un vantaggio è che sia un protocollo fair in quanto suddivide il tasso trasmissivo esattamente R/N bps . dall'altra è anche uno svantaggio in quanto ogni nodo è limitato a questo tasso, nonostante in certi casi sia l'unico presente.

- **fdma:**

divisione di frequenza. ognuno usa una particolare frequenza per parlare.



come il precedente evita collisioni ed è fair. btw limita la banda a R/N

- **cdma:**

assegna un "codice" (marchio univoco) ad ogni nodo. allo stesso momento vi è una condivisione di tempo e frequenze.

#attenzione! se trasmetto un segnale, potrebbero esserci delle collisioni, ma più che collisioni si dovrebbe parlare di sovrapposizione di segnali. Si parla di collisioni perché se dopo la sovrapposizione non riesco ad ottenere le informazioni originali delle trasmissioni singole allora il segnale è inutile.

1.5.2 protocollo ad accesso casuale

il nodo invierà sempre alla massima velocità consentita dal canale.

quando si verifica una collisione, i nodi ritrasmetteranno più volte il pacchetto. prima di ogni tentativo si attende un timer casuale (random delay).

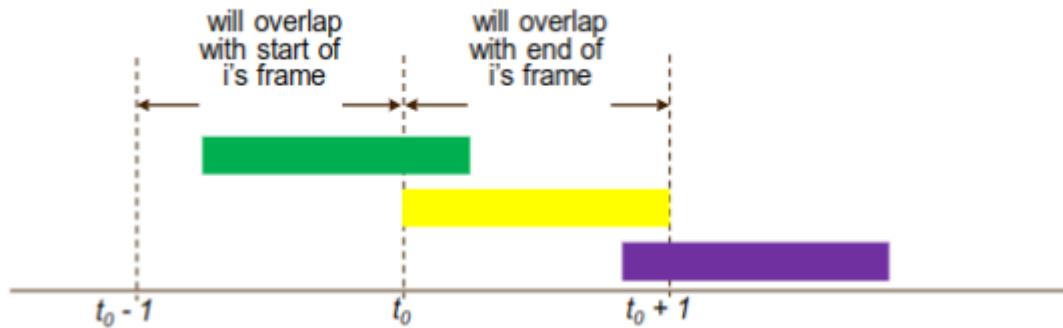
1.5.2.1 ALOHA

sviluppato in origine per far comunicare edifici di uni hawaii.

si posizionava un antenna centrale che opera come ripetitore, questa divide la banda in 2 (upload, download).

quando un pacchetto deve essere trasmesso, viene fatto senza controlli.

avremo quindi:

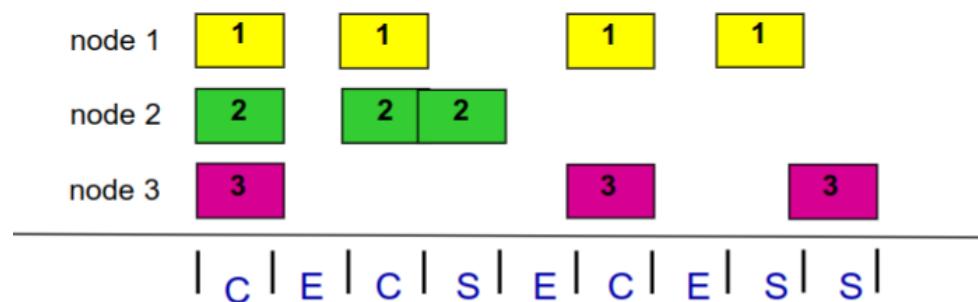


ovvero si creano delle collisioni.

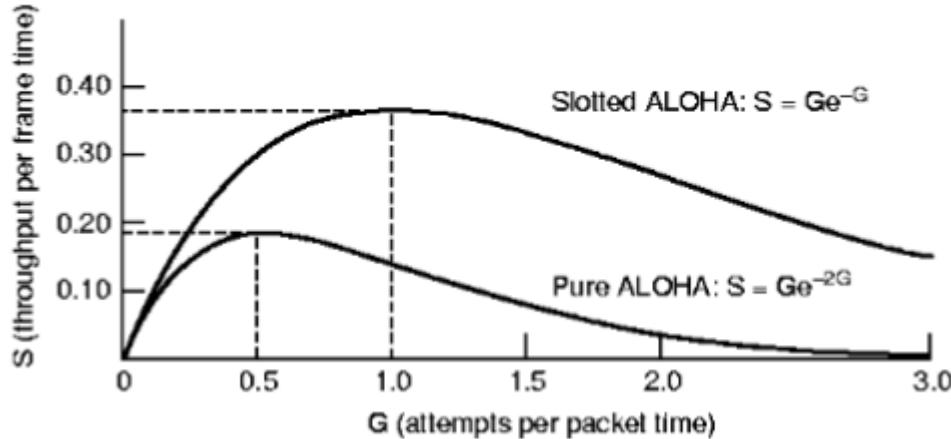
conviene usare questo approccio nel caso di basso tasso di comunicazione.

questa è la versione "unslotted".

il problema delle collisioni può essere risolto dalla suddivisione in slot, da cui viene aggiornato il protocollo in **slotted ALOHA**.



ovvero l'antenna centrale manda dei segnali di temporizzazione, dove ad ogni segnale chi vuole può trasmettere.



uso del canale:

- pure ALOHA: 18,4% delle sue potenzialità
- slotted: 36,8%.

Il problema di base è che ognuno parla quando vuole, un controllo banale è ascoltare il canale per verificare che nessuno stia parlando.

Pros:

- single active node can continuously transmit at full rate of channel
- highly decentralized: only slots in nodes need to be in sync
- simple

Cons:

- collisions, wasting slots
- idle slots
- nodes may be able to detect collision in less than time to transmit packet
- clock synchronization

1.5.3 accesso multiplo con rilevamento della portante (CSMA)

evoluzione di ALOHA.

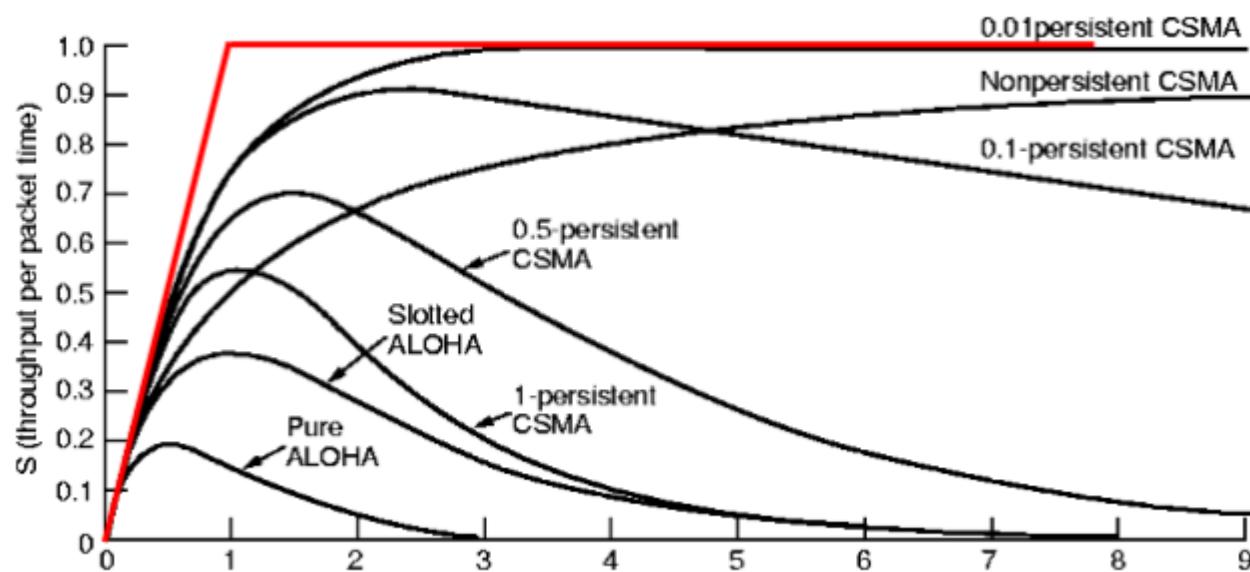
negli approcci precedenti ogni nodo prendeva la decisione di trasmettere indipendentemente dall'attività degli altri nodi.

i protocolli CSMA (carrier sense multiple access) e CSMA/CD (CSMA with error detection) si basano su queste due regole:

- **rilevamento della portante**: nodo ascolta il canale prima di trasmettere. se occupato aspetta finché si libera.
- **rilevamento collisione**: il nodo che trasmette rimane in ascolto, se osserva che qualcun altro sta trasferendo, si arresta.

si possono attuare diversi approcci:

- **CSMA 1-persistent**: appena la precedente comunicazione termina, trasmette immediatamente. il problema è che se ci sono più sistemi che aspettano, comunicheranno nello stesso istante causando collisione.
- **CSMA p-persistent**: ovvero trasmetti con una certa probabilità (p), altrimenti aspetti ancora. La probabilità è random.
- **CSMA non-persistent**: la stazione aspetta un tempo random prima di ricontrillare il canale.



l'ideale sarebbe la linea rossa.

più abbasso la probabilità di trasmettere più aumento le prestazioni, a discapito però delle prestazioni stesse.

come si nota lo **0.01 persistent CSMA** è il migliore, ma equivale ad avere 1% di probabilità di trasmettere.

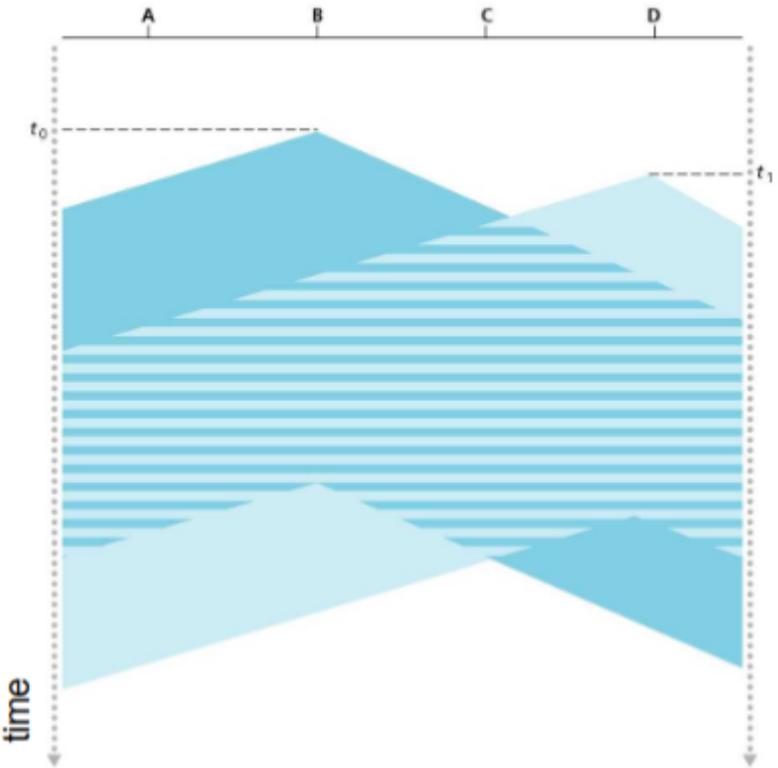
1.5.3.1 perché continuano ad esserci collisioni

nonostante l'ascolto del canale, why? ... because?

si basa tutto sul ritardo di propagazione.

nell'effettivo è necessario un intervallo di tempo non nullo (circa velocità della luce) affinchè un pacchetto arrivi a destinazione.

per cui, un nodo vedendo il canale apparentemente libero inizia a trasmettere (al tempo $t_1 > t_0$), ignaro del fatto che magari un altro nodo (al tempo t_0) ha iniziato una trasmissione (collisione).



maggiore è il ritardo, maggiore sarà la possibilità di collisione (come nella realtà).

1.5.3.2 CSMA/CD

le operazioni dal punto di vista di una scheda di rete sono le seguenti

1. scheda ottiene datagramma di rete, prepara frame a lvl collegamento e lo sistema nel buffer
2. quando il canale è libero (non vi è energia che entra nella scheda del canale), inizia la trasmissione. se il canale è occupato, resta in attesa sino a quando si libera.

3. durante la trasmissione verifica la presenza di altri segnali sul canale. se ne rileva alcuni ferma la trasmissione, aspetta un tempo casuale (**tempo di backoff**) e torna al passo 2

il tempo di attesa, anche detto tempo di backoff è determinato dall'algoritmo "**binary exponential backoff**" (attesa binaria esponenziale) (usato da ethernet).

- in pratica quando il trasmettore rilva l'n-esima collisione, calcola il tempo di backoff prendendo un valore casuale K dall'insieme $\{0, 1, 2, \dots, 2^n\}$.
- detto esponenziale per la cardinalità dell'insieme.

in ethernet il tempo di backoff è pari al tempo necessario per inviare K volte 512 bit (per ovviare al problema della propagazione). con il valore max di n =10.

```
rand_number = rand(0, 2^n)
```

```
tempo_backoff = rand_number * tempo_invio_512bit
```

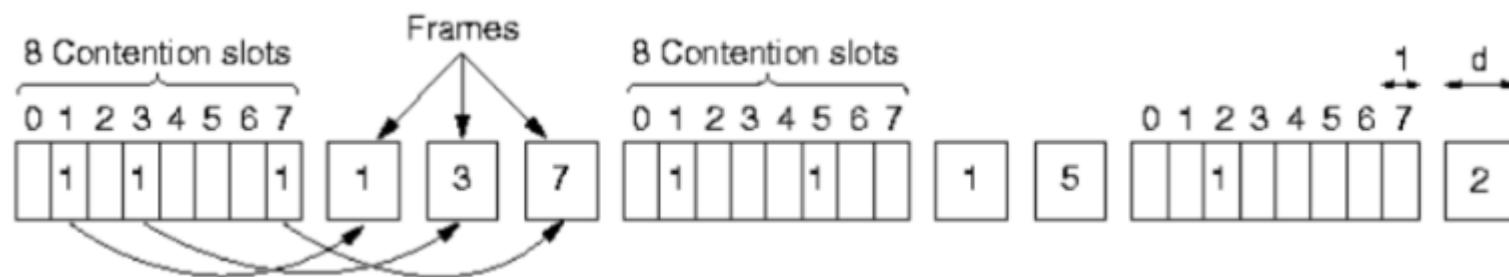
1.5.4 protocolli senza collisione

si potrebbe avere una soluzione senza collisioni?

- si ma con forti svantaggi

1.5.4.1 bit map

si predisponde un buffer, contenente un numero di slot pari numero di stazioni che possono parlare contemporaneamente nel mezzo.



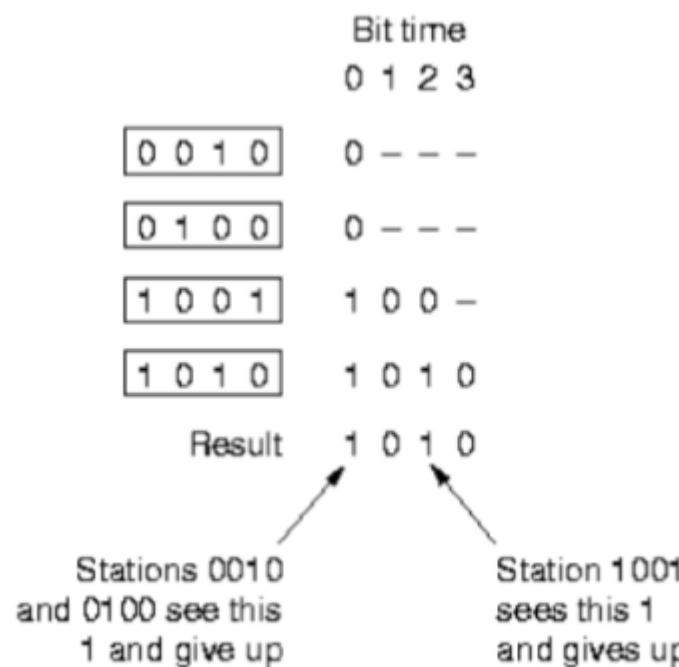
le macchine che vorranno trasmettere metteranno 1bit nel proprio slot. al termine di questo periodo di "contesa", tutte le macchine sapranno chi vorrà parlare.

problemi:

1. Se ci sono tante macchine il periodo di contesa è inutilmente grande, e considerando che le frame hanno la loro ridondanza, quello è un ulteriore ritardo inutile. Inoltre, è uno schema che ha senso se tutte le macchine presenti tendono a parlare. Altrimenti, se abbiamo tante macchine che parlano poco il canale rimane inutilizzato per gran parte del tempo.
2. A priori devo sapere quante macchine ci sono nel mio sistema, non posso aggiungere o togliere macchine a piacimento. Non basta "aggiungere" uno slot, perché devo informare tutte le macchine. se qualcuno non dovesse capire questa informazione (dato che non c'è sincronizzazione) qualche macchina potrebbe interpretare l'ultimo slot come momento dove cominciare a trasmettere.

1.5.4.2 CSMA/BA CanBus

conteggio binario a ritroso.



allo stesso momento le macchine scrivono sul canale il proprio indirizzo, in modo sequenziale partendo dal bit più significativo. se il bit è zero, non scrivono niente.

è una sorta di sfida tra gli indirizzi.

al tempo t_0 scrivono il primo bit, t_1 secondo, ...

il bit 1 prevale su 0. al termine, la macchina rimasta avrà già scritto il suo indirizzo sul canale. basterà adesso aggiungere il pacchetto ed inviare.

vantaggi:

- no ridondanza di slot
- scalabilità macchine

non è fair perché si favoriscono gli indirizzi più alti. si potrebbe pensare ad assegnare delle priorità

1.5.4.3 protocolli a rotazione

dalle proprietà fondamentali dovremmo avere dei protocolli che:

1. dato un nodo, il throughput = R bps
2. dati M nodi, il throughput = R/M

con ALOHA e CSMA si rispetta la 1 ma non 2. vennero quindi introdotti i protocolli a rotazione.

1.5.4.3.1 protocollo di polling

like scheduling cpu.

è presente un nodo principale che comunica ai vari nodi la facoltà di iniziare a comunicare oppure terminare.

btw viene introdotto il **ritardo di polling** ovvero il tempo richiesto per notificare un certo nodo.

inoltre, ad esempio se vuole parlare un solo nodo, non lo farà ad R bps ma ad un tasso inferiore. in quanto ciclicamente il nodo principale contatta gli altri inattivi.

1.5.4.3.2 token bus

si crea un anello logico tra i nodi, i quali si passano un token. chi ha il token occupa il canale.

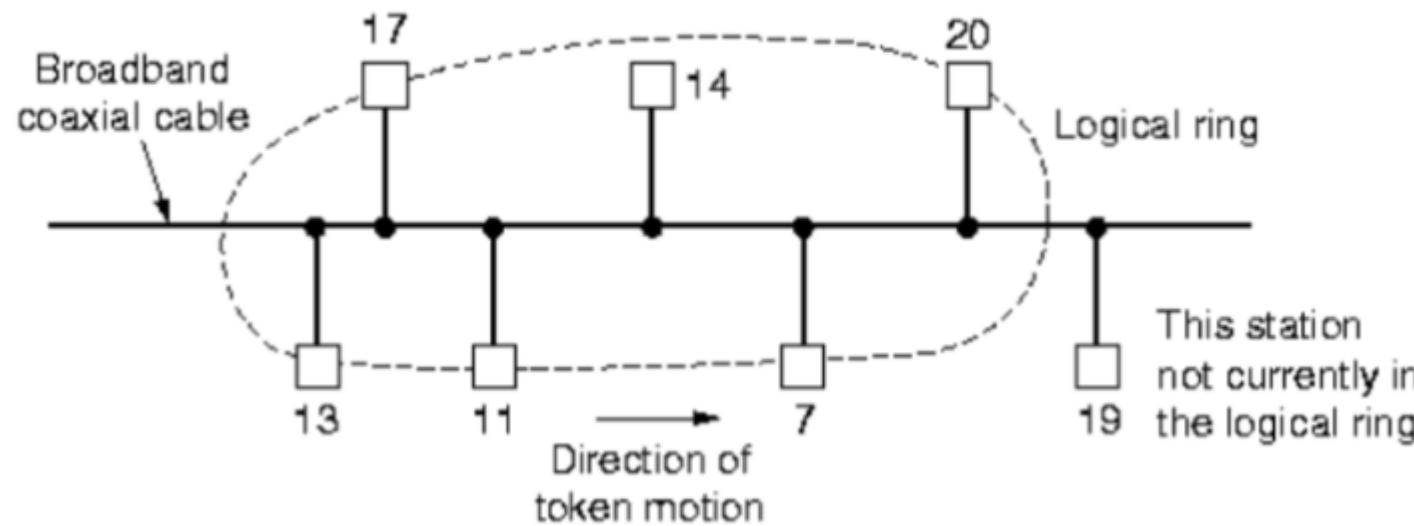
Tutte le macchine sorvegliano il canale per individuare il passaggio di token. Se dopo un certo tempo (più grande del tempo precedente) tutte notano che il token non sta girando allora tutte sono autorizzate a generare un nuovo token, nella speranza che venga generato un solo token alla volta.

quindi ad es. il nodo 1 passa il token al 2, se deve comunicare qualcosa, lo fa, altrimenti lo passa avanti.

svantaggi:

- guasta un nodo => canale fuori servizio

complex => non si usa più



token ring: disposizione ad anello delle macchine

1.6 Indirizzi MAC

indirizzi flat.

dire differenza da indirizzi gerarchici (ip)

sono utili sono nelle situazioni locali, permettono una ricerca in tempo costante (nei casi in cui le tabelle sono popolate, dopo il flooding).

1.7 Ethernet 802.3

implementazione pratica di CSMA/CD.

802.3 è il nome dello standard istituito dalla IEEE.

a differenza di token bus/ring questo standard è best effort, non garantisce la comunicazione.

1.7.1 cablaggi

la prima versione di ethernet era a 10Mbit/s ed erano previsti 4 cablaggi.

Name	Cable	Max. seg.	Nodes/seg.	Advantages
10Base5	Thick coax	500 m	100	Original cable; now obsolete
10Base2	Thin coax	185 m	30	No hub needed
10Base-T	Twisted pair	100 m	1024	Cheapest system
10Base-F	Fiber optics	2000 m	1024	Best between buildings

Nella tabella sopra, sono presenti anche i valori massimi di lunghezza di un segmento di cavo e della quantità di nodi massima che si possono inserire.

questi valori sono quelli entro cui il segnale dovrebbe viaggiare senza problemi, senza causare dispersioni.

possiamo creare dei segmenti più grandi o mettere più nodi, però non è garantito che il segnale viaggi intatto.

- **10Base5**: cavo rigido. per la connessione si usava una presa a “vampiro” che mordeva il cavo e tramite un punteruolo si connetteva allo strato di cavo per la connessione. C'erano dei punti ben precisi dove inserire il connettore, in quanto in quei punti la guaina metallica, che avvolgeva la parte interna era più larga. Bisognava evitare di tagliarlo, in quanto le unioni causano delle resistenze che possono disturbare il segnale.

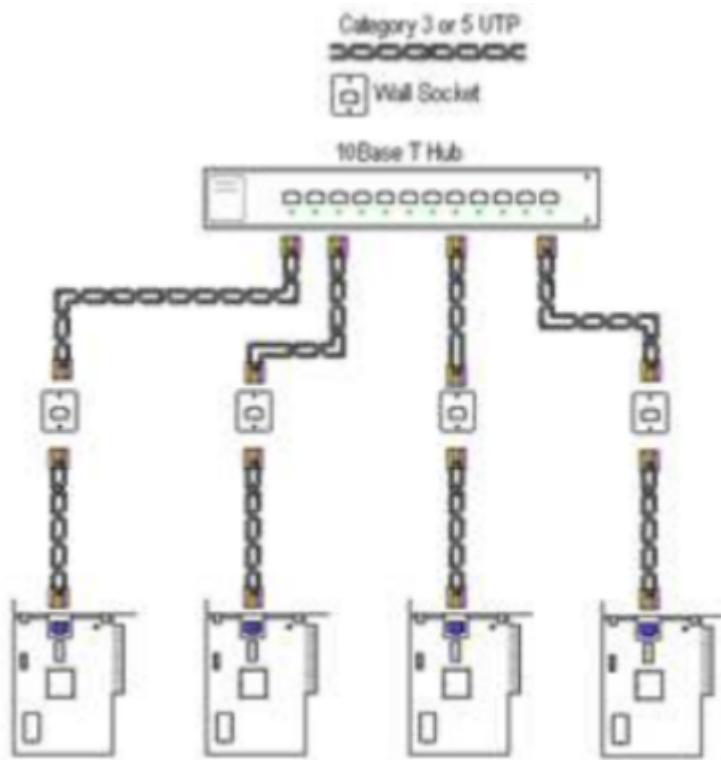
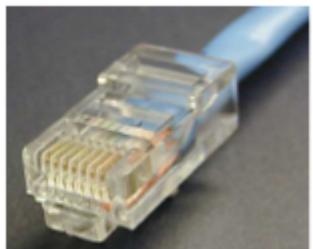


- **10Base2**: più sottile del suo predecessore, permettendo così di poter tagliare il cavo e di innestare le varie macchine con un particolare connettore a “T”.

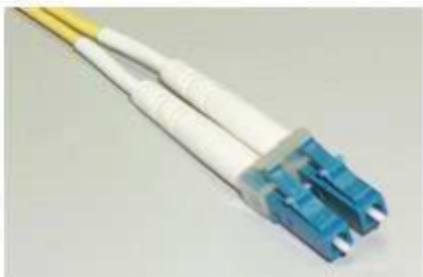


Questi tipi di cavo sono stati abbandonati in quanto:

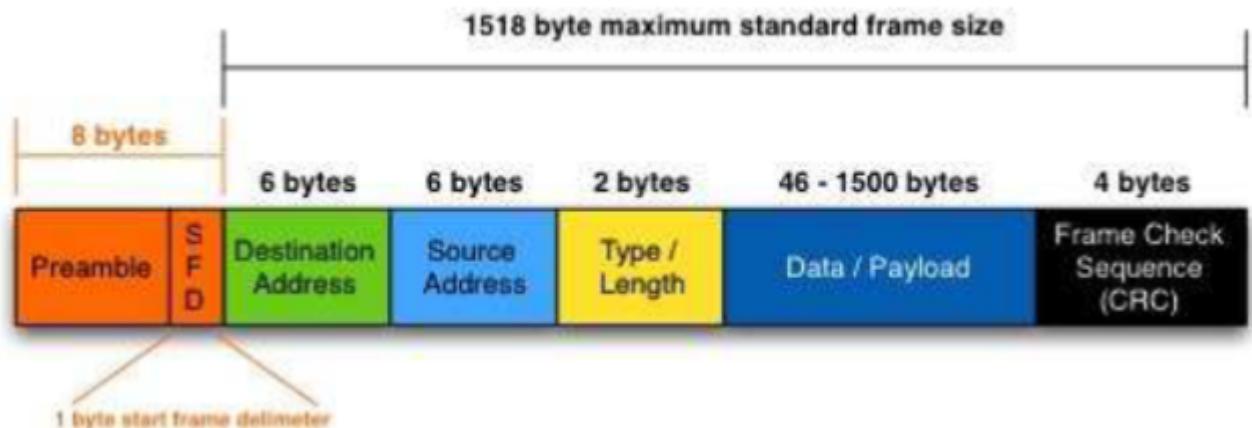
1. alla fine del cavo, era necessario inserire una sorta di tappo. Se una frame si perdeva e colpiva questo tappo rimbalzava, andando a disturbare la comunicazione.
 1. soluzione: inserire una sorta di resistenza che “mangiava” il segnale perso.
 2. guasti, essendo collegate in serie se si guastava una terminazione, essa faceva da tappo disturbando tutto il canale. individuare la terminazione guasta era un grosso problema.
- **10Base-T**: cavo odierno. Twisted Pair, cavo a coppie intrecciate. ha 4 coppie, ogni coppia ha un colore pieno e un colore bordato di bianco. Questi cavi permettono comunicazione massimo 100m con più nodi, questo perché lo schema di collegamento prevedeva un hub centrale, un concentratore, e la connessione era diretta tra scheda di rete e concentratore. Possiamo vederlo anche come un centro a stella.



- **10Base-F:** fibra ottica



1.7.2 frame ethernet



- **preamble:**

8byte. i primi 7 byte conterranno la seguente stringa 10101010 e l'ultimo 10101011. i primi "risvegliano" la scheda di rete ricevente e sincronizzano il [clock](#) con quello del trasmittente (in base al tipo di clock verrà effettuato un tasso di invio specifico). l'ultimo serve per terminare la fase di sincronizzazione (cambia l'ultimo bit)

- **destination/source address:**

6 byte. indirizzo MAC

- **type:**

2 byte. tipo di frame contenuta

0x0800 IPv4

0x0806 ARP

0x0842 Wake-on-Lan

0x8035 RARP

0x809B Ethertalk

0x80F3 AppleTalk Address Resolution Protocol (AARP)

0x8100 VLAN-tagged frame (IEEE 802.1Q)

0x8137 Novell IPX

0x8138 Novell

0x86DD IPv6

- **data:**

da 46 a 1500 byte.

contiene il datagramma IP. se supera [MTU](#) di ethernet (1500byte) l'host dovrà frammentare.

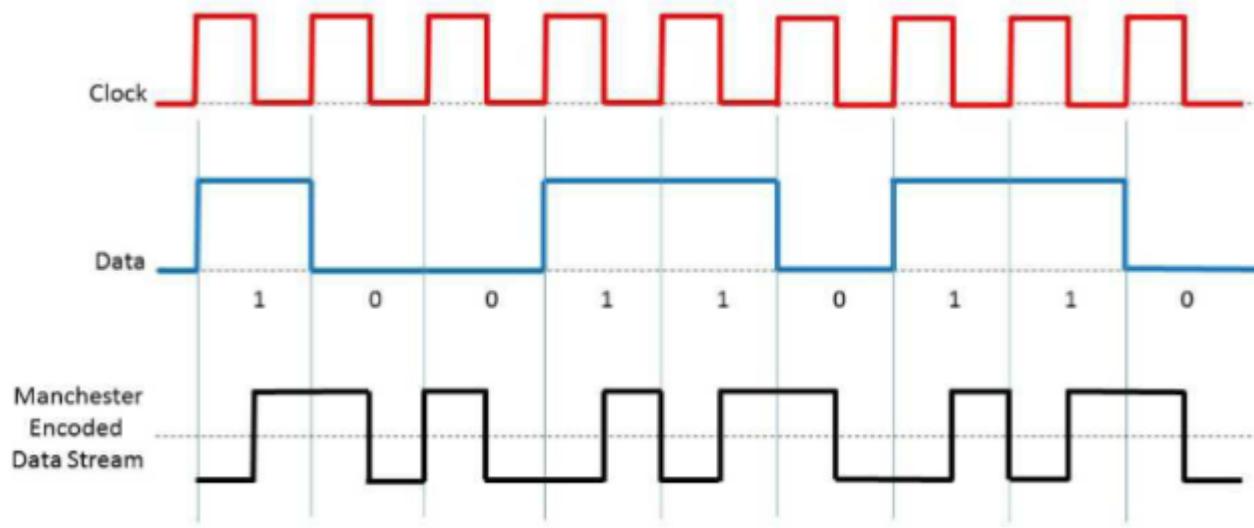
se invece è inferiore al minimo (46 byte) il payload dovrà essere riempito / stuffed per arrivare al minimo. questi poi saranno rimossi usando il [campo lunghezza](#) del datagramma ip.

- [**CRC:**](#)

4 byte. rilevazione errori.

1.7.3 codifica manchester

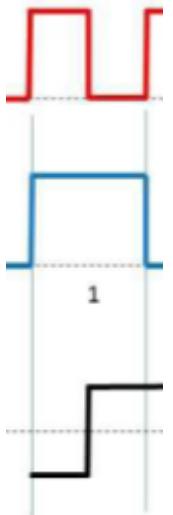
utilizzata per evitare i dubbi sulla [trasmissione o meno del bit zero](#).



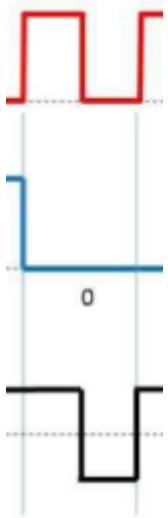
in quanto le variazioni sono identificate maggiormente rispetto ad un segnale costante.

si prevede un segnale di clock, all'interno del quale rappresento i bit da mandare come segue:

- uno con basso/alto



- lo zero con un segnale alto/basso



1.8 Fast ethernet (802.3u)

Se saliamo a 100Mbps la frame minima sale a 640byte, ma non viene fatto (quindi si rimane a 64byte) per avere una sorta di retro compatibilità. Per cui è stata solo aggiornata la velocità.

sono usati i seguenti cablaggi:

Name	Cable	Max. segment	Advantages
100Base-T4	Twisted pair	100 m	Uses category 3 UTP
100Base-TX	Twisted pair	100 m	Full duplex at 100 Mbps
100Base-FX	Fiber optics	2000 m	Full duplex at 100 Mbps; long runs

- FX:
la fibra ottica.
- TX:
è il doppino intrecciato di CAT5 (due coppie in andata e due al ritorno, le ultime due inutilizzate);
- T4:
viene usata per retrocompatibilità con alcuni cablaggi esistenti, il problema è che bisogna utilizzare una codifica particolare chiamata 8B6T (identifica 8 bit con una sestina di 3 valori).

utilizzo il cavo in maniera particolare: una coppia solo in andata, una coppia solo in ritorno e due coppie o in andata o in ritorno, per cui posso avere massimo 3 coppie che vanno a 33Mbps in una sola direzione.

Quindi 33Mbps in full-duplex e 66Mbps in half-duplex, a seconda di quello che serve.

La codifica Manchester non serve più in quanto non devo fare più sincronizzazione e posso utilizzare codifiche tipo la 4B5B, avendo uno spreco molto più basso e quindi sfruttando meglio il mio cavo.

Ovviamente devo cambiare l'hardware in quanto devo generare frequenze maggiori.
inoltre le distanze (tranne la fibra ottica) sono scese però i valori non sono proprio pochi.

1.9 gigabit ethernet

Name	Cable	Max. segment	Advantages
1000Base-SX	Fiber optics	550 m	Multimode fiber (50, 62.5 microns)
1000Base-LX	Fiber optics	5000 m	Single (10 μ) or multimode (50, 62.5 μ)
1000Base-CX	2 Pairs of STP	25 m	Shielded twisted pair
1000Base-T	4 Pairs of UTP	100 m	Standard category 5 UTP

si cerca di usare i cavi esistenti, utilizzando però tutte e 4 le coppie.

Cinque passi verso 1000Base-T (CAT5):

1. Rimuovere codifica 4B5B (100 -> 125 Mbps)

Già solo rimuovendo questa codifica aumentiamo il throughput, ma ritorna il problema del framing.

2. Usare le 4 coppie simultaneamente (125 -> 500 Mbps):

Fast Ethernet ne usa solo 2, una di andata e una di ritorno. Usando tutte e quattro le coppie simultaneamente possiamo quadruplicarlo.

3. Trasmissione full duplex (500 Mbps full duplex):

Conseguenza dell'uso delle 4 coppie simultanee.

4. Usare 5 livelli di codifica invece che 3 (MLT-3) (500 Mbps -> 1Gbps full duplex):

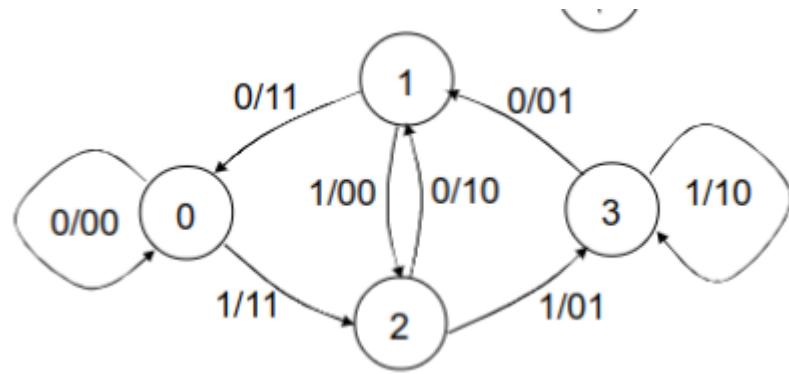
Un livello viene usato per fare framing, gli altri 4 permettono di trasportare 2 bit al livello. Da 500 passiamo a 1 GBps. L'unico problema è che il tasso di errore è così alto che ogni frame sarebbe soggetto a errore.

• Usare Forward Error Correction:

Correzione errori a destinazione per sistemare il problema.

1.9.1 Schemi di Trellis Viterbi.

si usa una macchina a stati finiti.



il cambio di stato è da interpretare come: "devo trasmettere x invio yy"

- 0/00 => devo 0 ... invio 00

è creata in modo simmetrico in modo che un errore provochi dei "bivi" a distanza di hamming differente.

ex: pag 100

- **mittente:**

- devo trasmettere la sequenza 011001.
- Parto dallo stato 0 e ci rimango trasmettendo 00.
- passo nello stato 2 e spedisco la coppia 11.
- Vado nello stato 3 spedendo 01, successivamente passo nello stato 1 spedendo 01.
- passo nello stato 1 spedendo 01 e nello stato 0 spedendo 11.

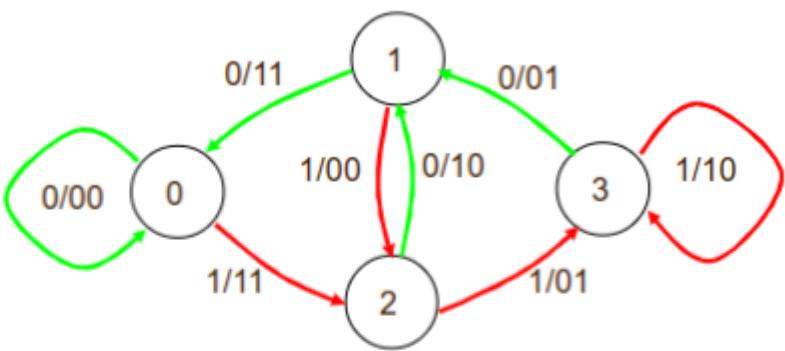
Dopo tutti questi passaggi mi ritrovo con la sequenza " 00 11 01 01 11 11 " che è lunga il doppio dell'originale (ridondanza del 100%).

- **destinatario:** assumo arrivi la stringa corrotta " 01 11 01 11 11 11 "

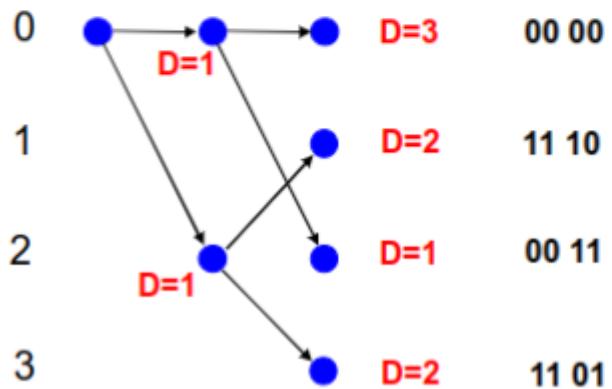
Parto dallo stato 0 e devo utilizzare il cammino 01, e immediatamente mi rendo conto che c'è un errore (la macchina a stati finiti non ha nessun cammino che partendo dallo stato 0 mi porti in un altro stato spedendo 01).

Faccio delle stime probabilistiche in base alla distanza di hamming. ad ogni cambio di stato controllo la distanza di hamming tra la coppia ricevuta e quella che dovrei ottenere dal cambio di stato; infine vado a prendere la coppia con la stringa.

01 11 01 11 11 11

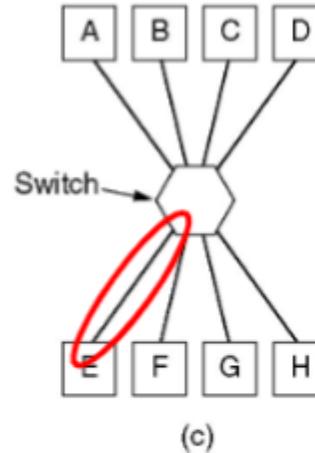
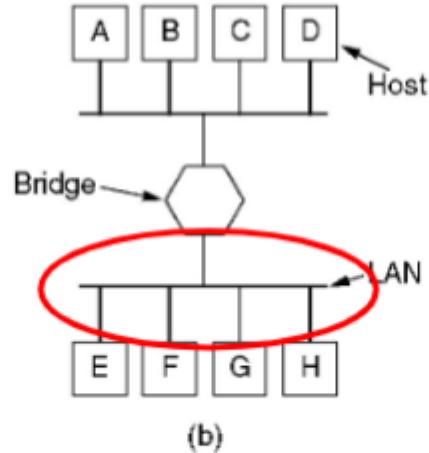
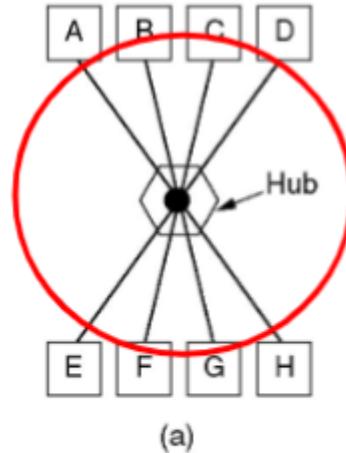


01 11



1.10 hub / bridge ethernet

dominio di collisione = area in cui si propaga l'errore



Collision Domains

hub: dominio di collisione è l'intera rete. non ha memoria interna, procede ad inviare in broadcast.

bridge: permette la connessione di più interfacce con protocolli differenti. il dominio è localizzato alla lan

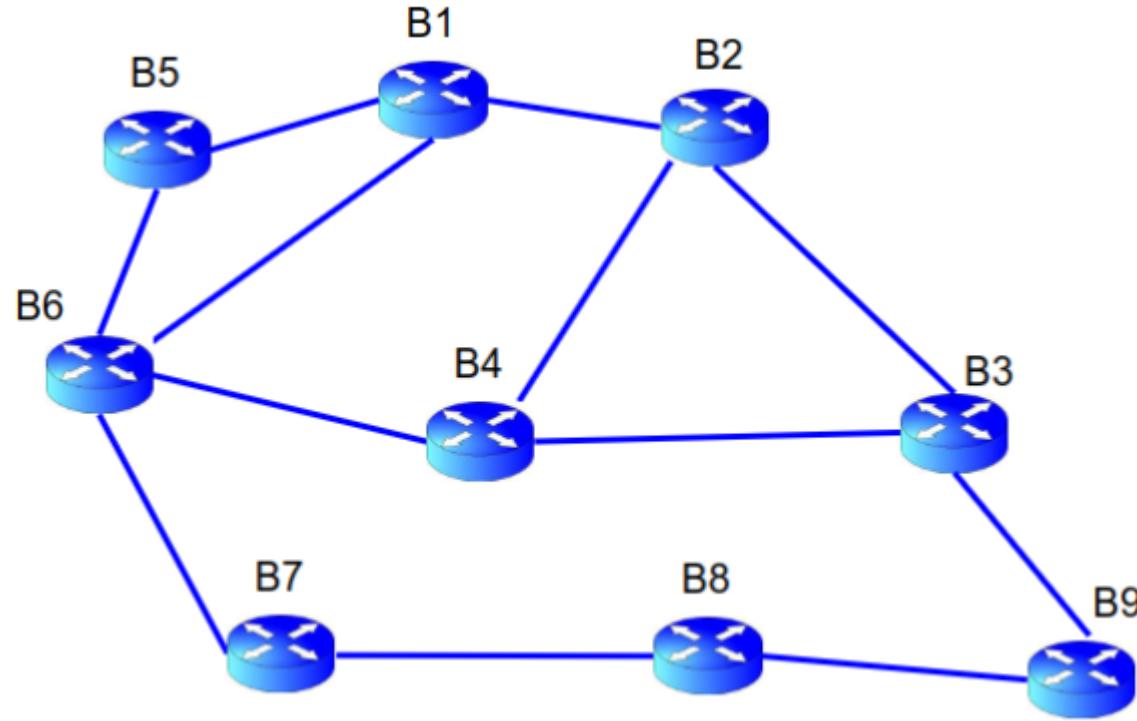
switch: bridge con porte solo ethernet. . è in sostanza un hub che mantiene in memoria i mac address, in quanto in un primo momento, non sapendo le destinazioni invia in broadcast. dominio localizzato alla singola porta
si "utilizzo trasparente"

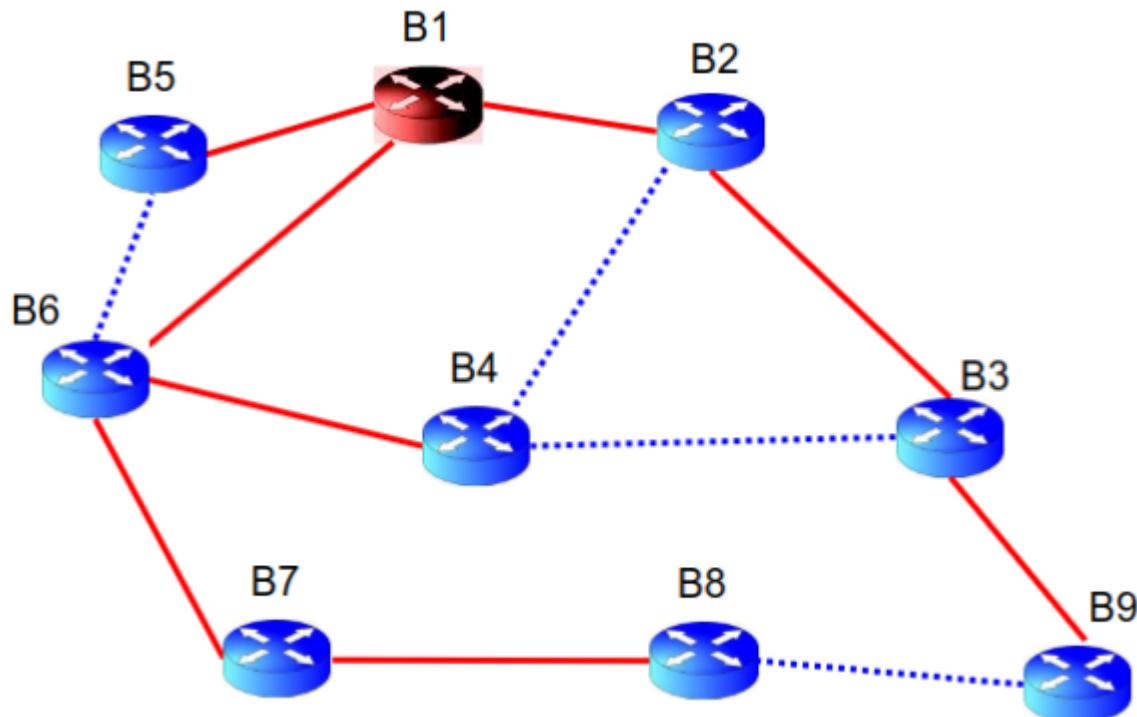
#attenzione! si parla di LAN fino a quando la comunicazione è possibile con i soli MAC (assioma del boss)

ex: pag 117 [slides](#).

1.11 spanning tree protocol

una soluzione al problema dei cicli provocati dal flooding.





1. B1 manda un pacchetto ai suoi vicini e si identifica
2. i suoi vicini rispondono con i loro identificativi e successivamente si fa un confronto per capire qual è l'identificativo più alto e quello più basso
3. una volta stabilito chi vince lo comunica agli altri. Nel nostro esempio B1 ha l'identificativo più alto, per cui diventa radice, B5, B2 e B6 invece diventano nodi di primo livello e se lo comunicano tra di loro, in modo da localizzarsi a vicenda.
4. B6 poi parla con B4 e B7 dicendogli che è sotto B1, inviando l'identificativo di quest'ultimo a B4 e B7; se l'id di B1 è sempre il più alto non cambia nulla, altrimenti viene sostituito da quello che lo batte.

Con un po' di questi passaggi e sapendo quando terminare l'algoritmo, è possibile identificare quali di questi nodi ha l'id più alto, in modo da tirare per prima cosa fuori la radice e in seguito si calcolano gli altri livelli di nodi.

Non sono i percorsi migliori, ma è un albero di copertura che ha come radice il nodo con l'id scelto. Tutto ciò avviene sempre in modalità trasparente.

1.12 virtual lan (VLAN)

Purpose of VLANs:

- savings: reuse of pre-existing lines and equipment;
- flexibility: easy physical movement of users;
- performance increase: broadcast traffic is confined;
- security: Users of different VLANs do not see each other's data frames.

si partiziona logicamente lo switch.

possono essere di due tipi:

- **vlan untagged**

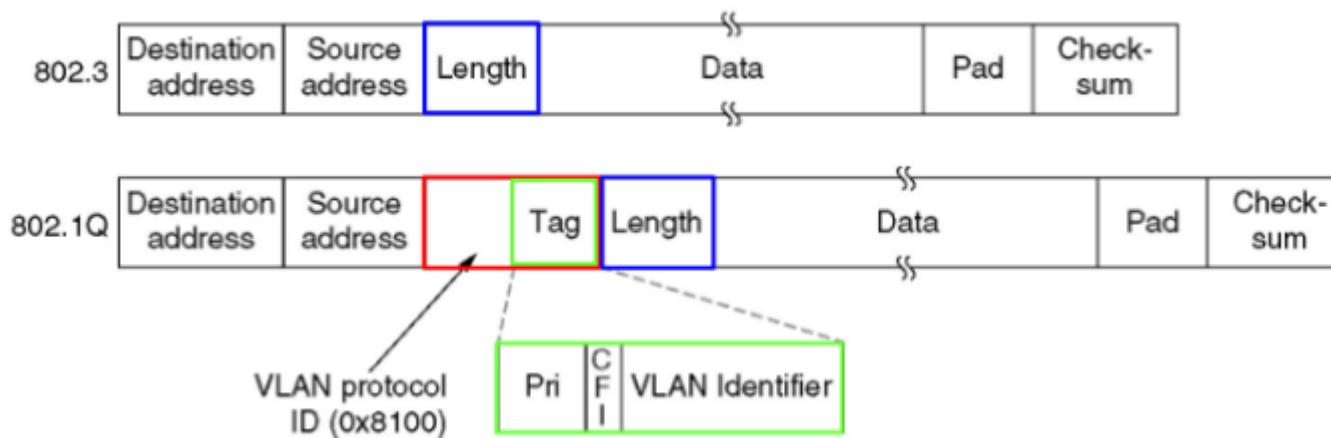
assegnazione di vlan manualmente per ogni porta.

avviene un collegamento diretto tra le vlan. è necessario un supporto di uno switch 802.1q

- **vlan tagged (802.1q)**

si invia una frame particolare, taggandola.

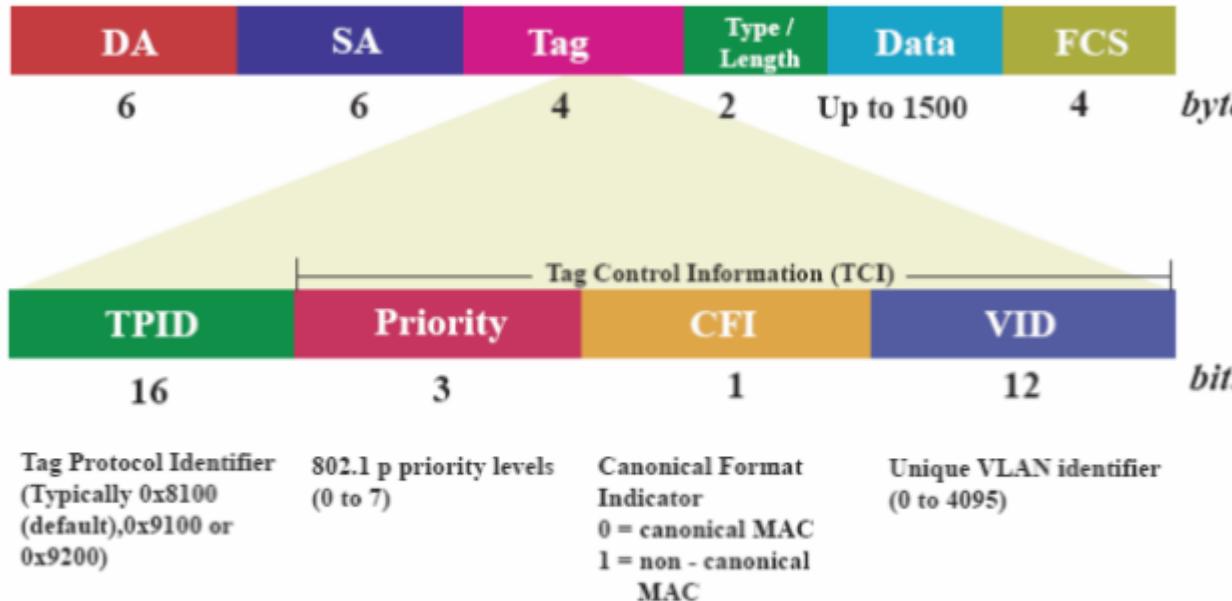
il frame ethernet btw non ha abbastanza spazio, quindi si allegano alla frame 4 byte.



questa aggiunta ha al suo interno:

- TPID (tag protocol identifier): identificativo
- Priority: da 0 a 7
- CFI (Canonical format indicator): identifica se è una frame normale oppure da gestire come vlan. 0 = è usato un mac canonico; 1 = non canonico

- VID (unique vlan identifier):



5 - Esercizi

1 Larghezza di banda

1.1 domanda

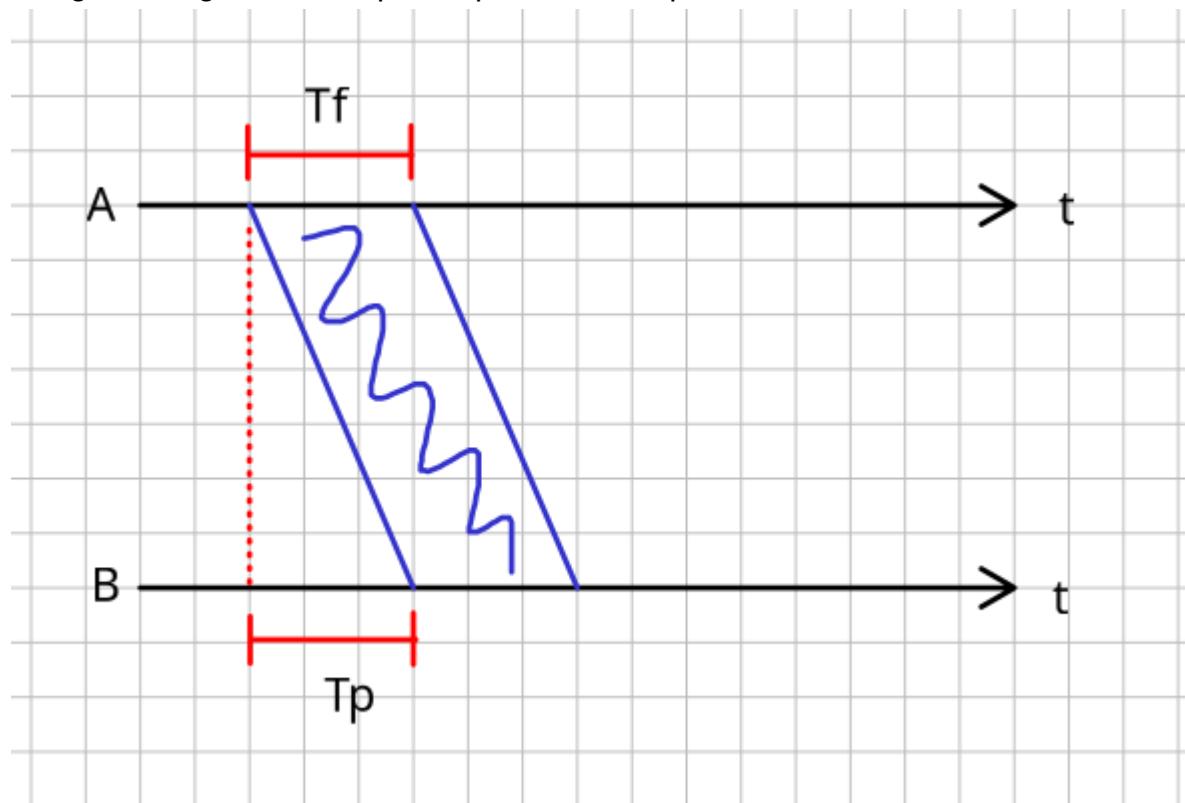
dato un canale di comunicazione tra A e B con un ritardo di 450ms, con larghezza di banda 1Mbps in andata e 100 Kbps al ritorno.

La dimensione del frame da mandare è 11000B, con intestazione di 1000B. La dimensione dell'ack è 150B

Con il protocollo Go back N, qual è la larghezza di finestra minima (W) per avere almeno un throughput di 500 Kbps (Bw)?

1.2 soluzione

1. disegno il diagramma temporale per l'invio del pacchetto



- T_f : intervallo generare la frame e scriverla sul canale.
 - dipende dal bitrate canale, inversamente
 - dimensione frame, direttamente
- T_p : tempo propagazione

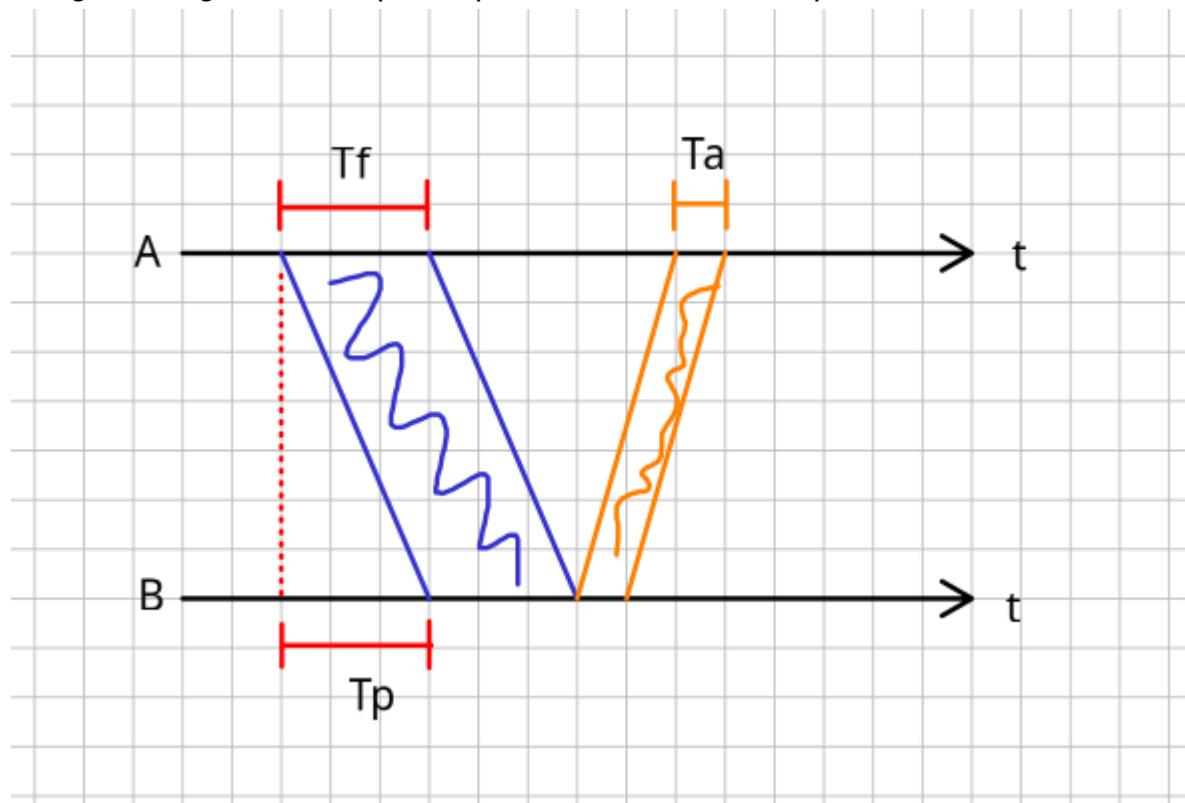
tempo di arrivo a destinazione = $T_p + T_f$

in generale per calcolare il tempo effettivo: $T = \frac{\text{roba da mandare}}{\text{larghezza di banda}}$

2. calcolo il tempo che impiego a scrivere la frame del mittente

$$T_f = \frac{\text{frame}}{bw_{\text{andata}}} = \frac{8 \cdot 110000}{10^6} = 88 \cdot 10^{-3} \text{s}$$

3. disegno il diagramma temporale per l'ack e calcolo il tempo di scrivere l'ack



$$T_a = \frac{ack}{bw_{ritorno}} = \frac{8 \cdot 150}{100 \cdot 10^3} = \frac{1200}{100 \cdot 10^3} = 12 \cdot 10^{-3} s$$

4. calcolo il tempo totale

$$T = T_f + \underbrace{T_{pf} + T_{pa}}_{450ms+450ms} + T_a = 1s$$

5. mi chiedo: "quanti byte ho trasferito in un tempo T "

assioma: "dovrebbe essere sempre un secondo".

calcolo la banda che occupo con un pacchetto, solo payload, rimuovendo header:

$$Bw_{1\text{ pacchetto}} = \frac{8 \cdot (11.000 - 1000)}{1s} = \frac{80000\text{byte}}{1s} = 80 \text{ kbyte/s}$$

In quanto go back n non è un protocollo stop and wait, posso trasferire frame ancora prima di ricevere ack; fino al punto in cui sfioro le dimensioni della finestra:

$$\frac{T}{T_f \cdot W} \geq 1$$

5. ora calcolo la finestra desiderata

$$W_{\minimo} = \left\lceil \frac{Bw_{\text{desiderata}}}{Bw_1 \text{ pacchetto}} \right\rceil = \left\lceil \frac{500 \text{ kbps}}{80 \text{ kbps}} \right\rceil = 7$$

ottenendo una banda:

$$Bw = 80 \text{ kbps} \cdot 7 = 560 \text{ kbps}$$

6. verifico di non sforare la finestra :

$$\frac{1s}{88 \cdot 10^{-3} \cdot 7s} \geq 1$$

$$\frac{1s}{616 \cdot 10^{-3}s} \geq 1$$

2 Spiega il timeout TCP

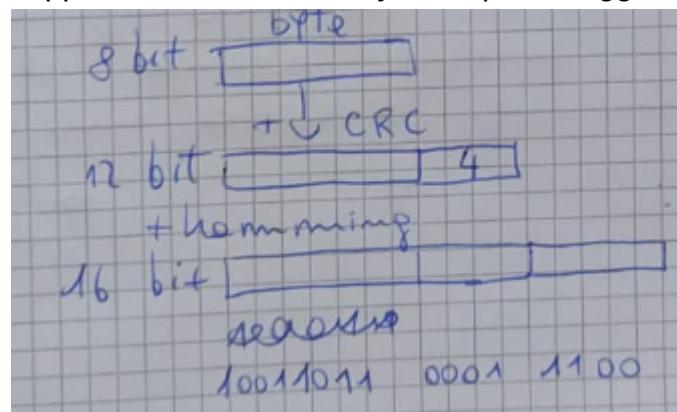
assiome: fare una mappa sulle cose da dire

- perché si usa
- come implemento
 - $SRTT_n$
 - $ERTT_n$
 - DEV
 - RTO
 - se scatta il timer $\Rightarrow 2 * \text{RTO}$
- il timer è sovrastimato, ma viene applicato il fast retrasmitt

3 verifica stringa CRC + Hamming

3.1 domanda

supponendo di inviare 8 byte, ai quali si aggiungono 4 bit di ridondanza CRC e 4 di hamming.



dato il seguente messaggio:

1001101100011100

verificare se è stato trasmesso correttamente

3.2 soluzione

1. inizio a verificare la codifica hamming
mi posiziono sui bit nelle posizioni a potenza di 2

1001101100011100
xx x x x

2. verificare il numero di bit di ridondanza
in questo caso ho $16 - 5 = 11$ bit di dati, quindi è probabile che la stringa sia di 12 bit con 5 bit di ridondanza.

usare il [palazzo della tabella](#).

3. aggiungo il bit mancante

metto un valore random, verrà risolto dall'algoritmo

```
10011011000111001
```

```
xx x x x
```

```
^
```

4. applico l'algoritmo.

per ogni bit di controllo vedo se il valore è uguale. nel caso contrario incremento il contatore `c` che mi indicherà la posizione dell'errore.

l'incremento è pari al valore della posizione del bit di controllo errato.

```
bit_1 = 3 5 7 9 11 13 15 = 0 + 1 + 1 + 0 + 0 + 1 + 0 + 1 = 0
```

il valore è errato

```
c = 1
```

stessa cosa per gli altri bit

```
bit_2 = 3 6 7 10 11 14 15 = 0 [c = 1]
```

```
bit_4 = 5 6 7 12 13 14 15 = 1 [c = 1]
```

```
bit_8 = 9 10 11 12 13 14 15 = 1 [c = 1]
```

```
bit_16 = 17 = 1 [c = c + 16 = 17]
```

5. correggo il bit alla posizione `c`

```
1001101100011100[0]
```

```
xx x x x
```

```
^
```

6. verifico CRC

mantengo la stringa modificata e tolgo i bit di controllo.

avendo il seguente generatore: 10011

```
010100011100 | 10011
```

```
10011
```

```
-----
```

```
001110
```

```
10011
```

```
-----
```

```
11101
```

```
10011
```

```
-----
```

```
011101
```

```
10011
```

```
-----
```

```
011101
```

```
10011
```

```
-----
```

```
011101
```

```
10011
```

```
-----
```

```
011100
```

```
10011
```

```
-----
```

```
011110
```

```
10011
```

```
-----
```

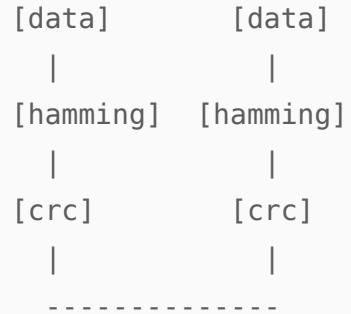
```
01101
```

il resto ottenuto è diverso da zero quindi la stringa è errata.

4 ordine dei controlli

conviene applicare prima hamming e poi crc o viceversa?

applicando hamming-crc.



a destinazione farò il controllo:

- stringa giusta => a cosa mi serve hamming
- stringa sbagliata => hamming potrebbe correggere, ma non se effettivamente la stringa arrivata a destinazione è corretta.

5 Stop & wait con errori

5.1 domanda

canale A - B di lunghezza 370km

- andata = 80Mbps
- ritorno = 8 mbps

protocollo stop and wait

- frame = 100B (instestazione) + 900B (payload) = 1000B
- ack = 200B
- tasso errore andata = $T_{err} = 1,5\%$

- tasso errore ritorno = $x \%$
- RTO = 200 ms

domanda: calcolare tasso errore ritorno per avere una $BW \geq 0.72 Mbps$

5.2 soluzione

1. calcolo i tempi per la generazione, rispettivamente, di frame e ack:

$$T_f = \frac{1000 \cdot 8}{80 \cdot 10^6} = 10^{-4} s$$

$$T_a = \frac{8 \cdot 200}{8 \cdot 10^6} = 2 \cdot 10^{-4}$$

2. calcolo il tempo di propagazione:

$$T_p = \frac{\text{lunghezza canale}}{\text{velocità propagazione nel canale fisico}} = \frac{2 \cdot 370 \text{ km}}{200.000 \text{ km/s}} = 370 \cdot 10^{-5} = 37 \cdot 10^{-4} s$$

calcolo il tempo medio di arrivo:

$$\begin{aligned} T_{\text{medio}} &= \underbrace{(T_f + T_a + T_p)}_{\text{tempo invio}} + \underbrace{(T_{\text{err}} \cdot RTO)}_{\text{tasso di perdite}} \\ &= (1 + 2 + 37) \cdot 10^{-4} s + \underbrace{\frac{1.5 + x}{100}}_{T_{\text{err}}} \cdot \underbrace{200 \cdot 10^{-4}}_{RTO} \\ &= (40) \cdot 10^{-4} + (1.5 + x)20 \cdot 10^{-4} \end{aligned}$$

$$= (70 + 20x) \cdot 10^{-4}$$

$$Bw = \frac{\text{payload}}{T_{\text{medio}}} = \frac{900 \cdot 8 b}{(70 + 20x) \cdot 10^{-4} s} \geq 0.72 Mbps$$

unifico le unità di misura:

$$0.72 Mbps = 72 \cdot 10^4$$

$$\frac{7200b}{(70 + 20x)10^{-4}s} \geq 72 \cdot 10^4 bps$$

$$\frac{7200b}{(70 + 20x)s} \geq 72bps \quad (\text{semplifico i } 10)$$

$$\frac{7200b}{72\frac{b}{s}(70 + 20x)s} \geq 1$$

$$\frac{100}{70 + 20x} \geq 1 \quad \text{semplifico e tolgo unita misura}$$

$$100 \geq 70 + 20x$$

$$\frac{100 - 70}{20} \geq x$$

$$x \leq 1.5$$