NoiStudiamo

APPUNTI

Reti di Calcolatori

Livelli applicazione, trasporto, rete e collegamento,
DNS, TCP, controllo di congestione ed Ethernet

A CURA DI

Paolo Volpini

Novembre 2026

INDICE

Indice degli argomenti

Gli argomenti principali degli appunti e la pagina in cui iniziano.

1. **Introduzione** 3
2. **Application Level** 7
3. **Protocolli** 11
4. **DNS** 15
5. **Transport Layer** 18
6. **Throughput** 27
7. **Protocolli per la pipeline, TCP** 30
8. **Controllo di flusso e di congestione** 43
9. **ECN, Fairness** 54
10. **Network Layer** 56
11. **Network Layer, 2** 63
12. **Network Layer, 3** 69
13. **Data Link Layer** 83
14. **Error detection e correction** 88
15. **CSMA, Ethernet** 93
16. **Gigabit, dispositivi di link** 99

00 - Introduzione

Introduzione alle reti di calcolatori

Struttura di una rete

> *Una rete informatica è una rete che collega più dispositivi.*

Una rete di comunicazione è costituita da due parti:

- una parte **fisica**, che collega fisicamente i dispositivi nel sistema. I dispositivi sono fisicamente connessi tra loro tramite **reti di collegamenti** (*communication link*) e **commutatori di pacchetti** (*packet switch*). Questi collegamenti possono essere di molti tipi e costituiti da varie tipologie di mezzi fisici, come fili di rame, fibre ottiche e onde elettromagnetiche, in grado di trasmettere a velocità di trasmissioni differenti (misurate in *bps*, ossia bit per secondo);
- una parte **logica**. Tale parte permette di trasmettere e ricevere comunicazioni attraverso il mezzo fisico.

Le informazioni trasmesse da un host ad un altro vengono divise in parti. Ad ogni singola parte viene aggiunta una intestazione. L'insieme di queste parti con le relative intestazioni prende il nome di **pacchetto**. I pacchetti sono inviati attraverso la rete verso la destinazione, dove vengono assemblati per ottenere i dati originari. I pacchetti vengono spediti attraverso il commutatore di pacchetto, che prende un pacchetto che arriva da un collegamento in ingresso e lo ritrasmette su uno di quelli in uscita. Principalmente il commutatore di pacchetto è o un **router** o un **commutatore a livello di collegamento** (*link layer switch*). Durante la trasmissione di pacchetti, questi viaggiano attraversando diversi commutatori di pacchetto: la sequenza di collegamenti prende il nome di **route**, ossia di **percorso** attraverso la rete.
I sistemi accedono a Internet tramite gli **Internet Service Provider** (**ISP**), che forniscono accesso a Internet.
Definiti i membri di questa rete, è necessario definire dei **protocolli di comunicazione** per gestire l'invio e la ricezione dei pacchetti.

> *Un **protocollo** definisce il formato e l'ordine dei messaggi scambiati tra due o più entità in comunicazione, così come le azioni intraprese in fase di trasmissione e/o di ricezione di un messaggio o di un altro evento.*

Due principali protocolli Internet sono il **TCP** (**Transmission Control Protocol**) e l'**IP** (**Internet Protocol**).

Struttura a livelli

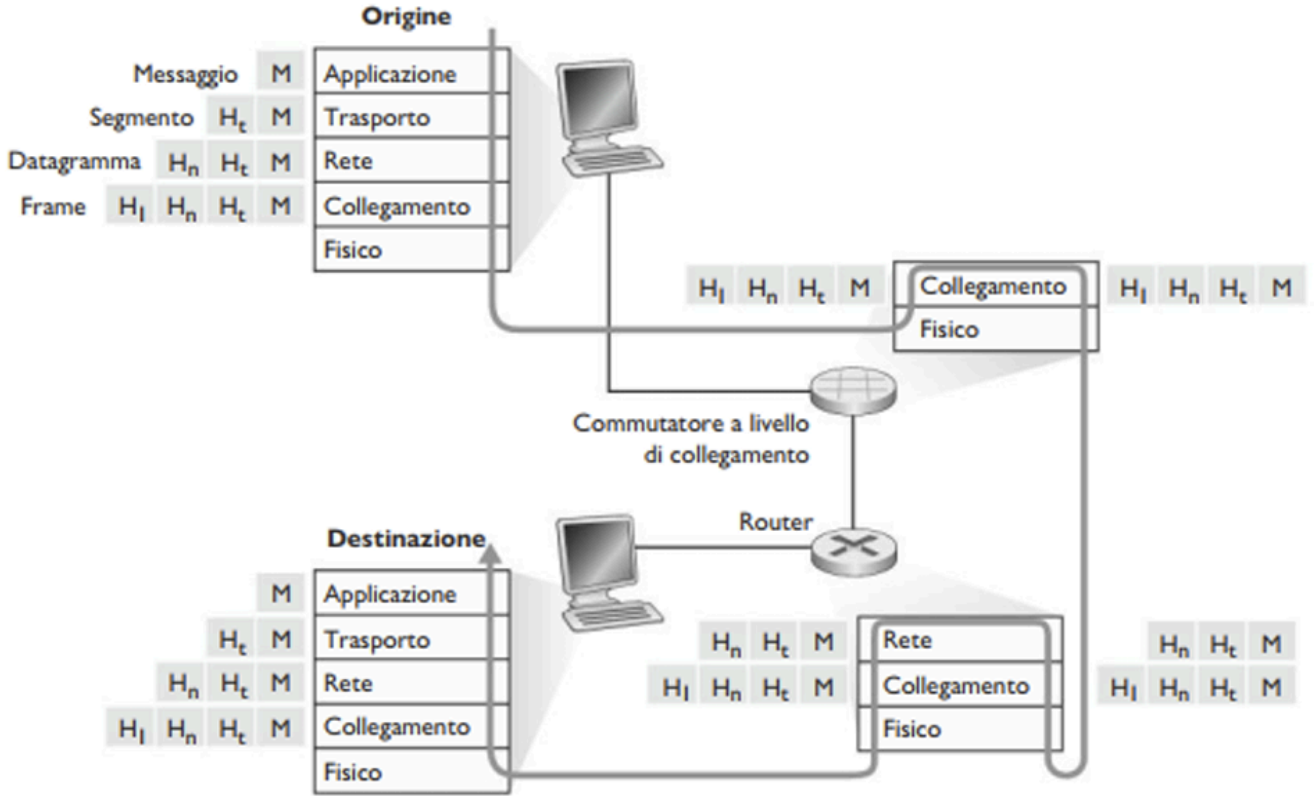
Per dare struttura alla progettazione dell'hardware e dei protocolli, si decise di progettare questi e i software che li implementano in **livelli** (*layer*). Una struttura simile permette di fornire un modo strutturato per trattare i componenti dei sistemi e facilita l'aggiornamento della componentistica. I protocolli dei vari livelli sono detti **pila di protocolli**. Una pila esempio può essere:



La pila consiste di 5 livelli principali:

- il livello di **applicazione**, sede delle applicazioni di rete;
- il livello di **trasporto**, che trasferisce i messaggi del livello di applicazione tra punti gestiti dalle applicazioni;
- il livello di **rete**, che trasferisce i pacchetti a livello di rete da un host a un altro;
- il livello di **collegamento**, che instrada i pacchetti tra la sorgente e la destinazione;
- il livello **fisico**, che trasferisce i singoli bit dei pacchetti da un nodo al successivo.

Incapsulamento



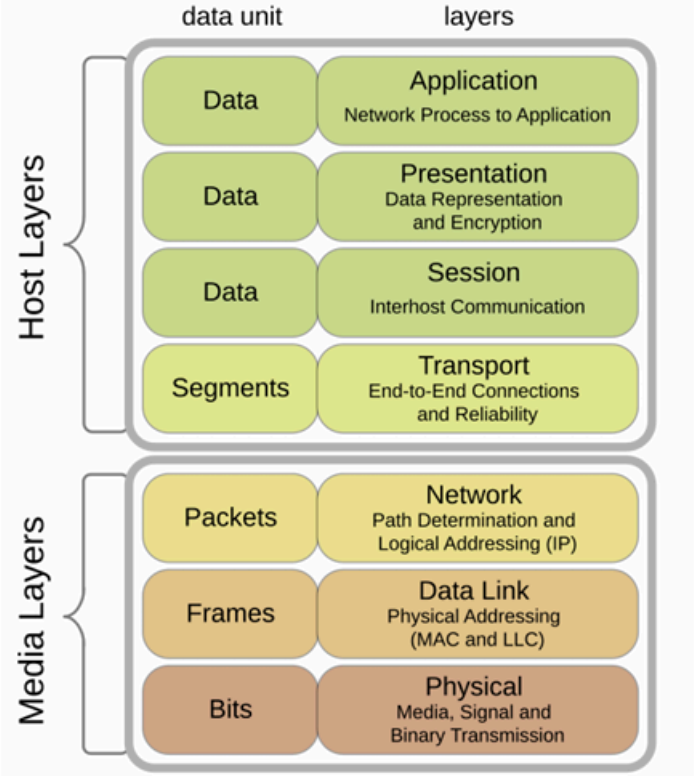
Il modo in cui i messaggi vengono mandati via rete si basa sul concetto di **incapsulamento**. Supponiamo che da un host mittente sia inviato un messaggio a livello di applicazione. Allora il messaggio viene passato al livello di trasposto, **aggiungendo tuttavia un'informazione aggiuntiva** (in figura H_t): il messaggio con l'informazione aggiuntiva costituisce il **segmento a livello di trasporto**. Queste informazioni aggiuntive sono necessarie per consegnare il messaggio all'applicazione desiderata in maniera corretta. Lo stesso segmento viene passato al livello di rete con altre informazioni aggiuntive, creando un **datagramma a livello di rete** (in figura H_n). A questo punto il datagramma viene inoltrato al livello di collegamento con ulteriori informazioni creando un **frame a livello di collegamento**. Ad ogni livello quindi viene aggiunta un'informazione in più.

L'informazione aggiuntiva che viene aggiunta al **payload**, ossia al carico trasportato dal livello precedente, prende il nome di **header**.

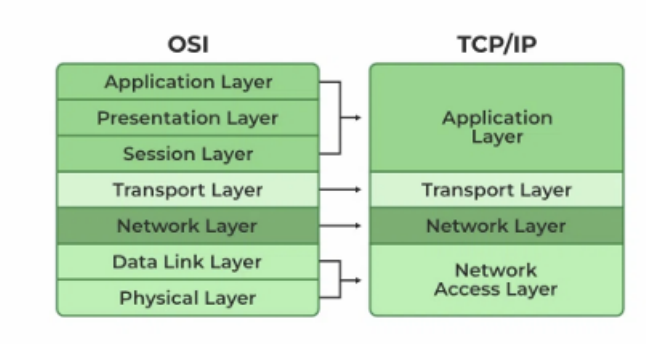
> *Un **header** è un pre-dato di formattazione che serve a preparare colui che riceve l'informazione in relazione a ciò che verrà dopo.*

Modello ISO/OSI, TCP/IP

Il modello teorico utilizzato per descrivere l'attraversamento dei dati nella rete è il **modello ISO/OSI**, costituito da sette livelli:



Nella pratica questo modello **non viene direttamente implementato**, ma serve come riferimento. Infatti nelle applicazioni reali si utilizza il **modello TCP/IP**, più semplice e pratico e basato su 4 livelli:



Comunicazione dei segnali

I dati che vengono trasmessi nella rete vengono processati (limitandoci ai calcolatori) dalle CPU. Ogni processore riesce a interpretare il linguaggio binario. Il segnale viene interpretato tramite la tensione del segnale: con una tensione superiore a $3.3V$, il segnale viene interpretato digitalmente come 1. Se invece i segnali sono al di sotto di $0.8V$, il segnale è interpretato come 0.
Nell'area intermedia tra questi valori di tensione, si ha un'area intermedia che dev'essere evitata.
Durante la propagazione il segnale può essere distorto in base alla distanza di propagazione della distanza. Il segnale inoltre non può cambiare immediatamente tra 0 e 1, in quanto ogni mezzo fisico crea un ritardo dovuto all'induzione. Se la distanza di propagazione aumenta, il segnale viene distorto a causa dei ritardi del dispositivo fisico.

Risulta quindi necessario avere un canale di comunicazione virtuale che sia **affidabile** e **privo di errori**. Tale sistema verrà poi implementato utilizzando un sistema fisico. Questi sistemi quindi devono funzionare **indipendentemente dai mezzi fisici impiegati**.

Tipi di comunicazione

Le comunicazioni possono essere:

- **connectionless**;
- **connection oriented**, in cui sono presenti segnali ACK, ossia segnali **acknowledge**, che permettono di capire se l'informazione è stata inviata correttamente;
- affidabile o non affidabile.

Ad esempio, supponiamo di inviare una lettera ad una persona. Tale tipo di connessione è connectionless, in quanto il "sistema" non prevede appunto nessun segnale ACK. Se invece all'invio della lettera il mittente deve inviare al destinatario una lettera che conferma la ricezione della lettera, questo tipo di trasmissione è connection oriented.
Il vantaggio quindi di un sistema connection oriented è quindi la possibilità di avere idea se la comunicazione è avvenuta con successo.

Il problema delle reti informatiche è che non si sa cosa succede dopo che vengono inviati i dati. Non abbiamo quindi un tracking puntuale di quello succede. Nel momento quindi in cui vengono inviati i dati, non sappiamo se ci possono essere ritardi, problemi di routing ecc. Chi deve ricevere potrebbe anche non sapere che dei dati sono stati inviati. L'affidabilità del canale è un altro problema che dev'essere indirizzato: non sempre infatti è opzionale l'indirizzamento di tali problemi (ad esempio, comunicazioni nel campo militare e medico).
Il meccanismo con cui può essere superato tale problema è il **timing**. Ad esempio, si sa che se entro un periodo finito di tempo non si riceve un segnale ACK, esso viene inviato nuovamente. Il livello relativo all'applicazione che gestisce i segnali poi si occuperà di capire se il segnale inviato è vecchio o meno.

Primitive di servizio

I meccanismi vengono gestiti tramite le **primitive di servizio**.

> *Una primitiva di servizio è una richiesta che viene approvata quando un host vuole comunicare con un altro host.*

Sono quindi le richieste che devono essere approvate durante le comunicazioni tra i nodi di una rete.

Topologie di reti

Le principali tipologie di rete sono **point to point** e **broadcast**. La prima consiste in un invio di dato da un dispositivo ad un altro. Nella seconda invece un dispositivo invia segnali a molteplici dispositivi connessi tra loro. Se i dispositivi sono interconnessi, l'invio di un segnale broadcast richiede la gestione delle **collisioni**.
Principalmente analizzeremo:

- le reti a **bus**, dove i dispositivi sono connessi tra loro tramite un bus di comunicazione;
- le reti ad **anello**, dove ogni dispositivo è connesso a due vicini;
- le reti a **stella**, dove i dispositivi sono connessi tra loro tramite un commutatore di pacchetti.

Un broadcasting nelle reti a bus è molto semplice, ma se un altro sistema di bus decide di farlo si crea una collisione. In questo caso basta introdurre degli **arbitri di bus** che decidono quale dispositivo può utilizzare il bus.
Un broadcasting nelle reti ad anello richiede la definizione di un orientamento per la trasmissione del segnale presenti nella rete. Se ci sono più pacchetti da gestire, basta introdurre una temporizzazione.
Nelle reti a stella ogni dispositivo può connettersi liberamente in quanto basta comunicare tramite il commutatore di pacchetti.
Le reti possono essere classificate in base alla loro estensione geografica (si parla di **tassonomia delle reti**). La più piccola rete è la **PAN** (**Personal Area Network**), con una distanza di 10 metri massimo tra i dispositivi. La **LAN** (**Local Area Network**) ci permettono di occupare un raggio totale di 1 kilometro. La **MAN** (**Metropolitan Area Network**) occupa un raggio di 10 kilometri. La **WAN** (**Wide Area Network**) gestisce anche raggi di 100 e 1000 kilometri. Infine internet occupa un raggio di 10000 kilometri. Internet può essere considerata come una WAN planetaria che connette più WAN.

La trasmissione delle informazioni avviene in modi diversi in virtù del protocollo.

Sicurezza nelle reti informatiche

Alcune problematiche a livello di sicurezza sono:

- **virus**, che sono oggetti dannosi allegati ad un eseguibile;
- **worm**, che si diffondono da un computer ad un altro;
- **botnet**, reti di calcolatori controllate tramite malware.

Attacchi ad una rete informatica sono:

- Denial of Service (DoS);
- Packet sniffing;
- IP spoofing.

Sono alcune problematiche che possono esserci in un'architettura distribuita.

IETF e RFC

Data l'importanza dei protocolli per la creazione di sistemi e prodotti che interagiscano tra loro, risulta fondamentale accordarsi sulle funzioni svolte da ogni singolo protocollo. Gli standard di Internet vengono sviluppati dalla **Internet Engineering Task Force** (**IETF**). Le pubblicazioni sugli standard invece vengono dette **Request For Comment** (**RFC**), che sono richieste tecniche e dettagliate da cui derivano specifiche per dei componenti di rete, in particolare per i collegamenti di rete e per i protocolli.

01 - Application Level

All'interno dell'**application layer** modelliamo le applicazioni e le comunicazioni tra le varie strutture. In una rete di calcolatori risulta necessario la comunicazione tra vari calcolatori (che rappresentano i nodi di tale rete). L'**interprocess communication** è uno strumento che permette la comunicazione tra processi. Abbiamo la necessità allora di dover gestire la comunicazione tra i vari processi.
Le applicazioni di rete richiedono una connessione **affidabile** e che sia **bidirezionale**. La maggior parte delle applicazioni di rete inoltre gestisce l'organizzazione dei vari host utilizzando due principali paradigmi: l'architettura **client-server** e l'architettura **peer-to-peer**.

Architettura Client-Server

Nell'architettura client server vi è un host **sempre attivo** chiamato **server**, che risponde alle richieste di servizio di molti altri host, detti **client**. In questo tipo di architettura **i client non comunicano direttamente tra loro**.

Server

Generalmente il server è l'**host** che riceve richieste e che ha un indirizzo IP statico (ossia **permanente**). Il server necessita di un IP statico per ragioni di praticità: in questo modo non si ha la necessità di comunicare ai nodi che fanno richiesta al server il momento in cui si ha un cambiamento di IP. L'IP statico tuttavia soffre di un problema di sicurezza: se l'IP viene scoperto, si possono utilizzare diversi attacchi diretti al server senza la necessità di dover scoprire nuovamente l'IP. Un server potrebbe non essere obbligatoriamente un solo calcolatore, bensì un data center con infrastruttura scalabile. Infine, il server dev'essere sempre connesso.

Client

Mentre il server dev'essere sempre connesso, il client può connettersi quando ha necessità di comunicare con il server. Inoltre, l'IP del client può essere dinamico.

Servizi nell'application layer che utilizzano il paradigma client-server

Servizi che utilizzano l'application layer sono HTTP, IMAP, FTP.
Il protocollo IMAP è un protocollo per exchange email che permette di vedere la inbox delle mail che virtualizza ciò che giace nei server del servizio mail. Il vantaggio è che il computer non ha localmente le mail salvate, tuttavia lo svantaggio è che i dati sono interamente memorizzati nel server.
Il protocollo FTP è il File Transfer Protocol usato per il trasferimento di file. Oggi non è più usato.

Scalabilità

Il sistema può essere scalabile, ma non senza difficoltà, in quanto nuovi nodi presenti all'interno della rete devono essere sincronizzati con le informazioni del server.

Precisazione sulle Connessioni

All'interno dello stesso host, due processi comunicano tra loro utilizzando una **interprocess communication**. Processi di host diversi comunicano tra loro tramite messaggi. Lo scambio di messaggi permette la **sincronizzazione** tra i vari processi.
Quando diciamo che un client si connette ad un server, in realtà diciamo che **il processo di un client invia messaggi al processo del server**.

Architettura Peer to Peer

In un'architettura **Peer to Peer** (o **P2P**) si sfrutta la comunicazione diretta tra coppie di host, chiamati **peer**. In questo caso il sistema non prevede un **arbitro**.
Lo svantaggio del Peer to Peer risiede nella creazione di una occupazione di banda o di congestione della rete globale e problemi anche nella sicurezza. Mentre il server può rifiutare connessioni di client, in realtà alcuni peer possono essere insicuri.

Scalabilità

Il sistema tuttavia gode di un'instrinseca **scalabilità**, in quanto i nuovi peer, ossia i nuovi nodi, possono inserirsi all'interno della rete senza problemi. Questo permette anche di aumentare la capacità di servizio al sistema senza costi eccessivi.

Servizi che utilizzano Peer to Peer

Servizi che utilizzano il protocollo Peer to Peer sono il P2P file sharing o le connessioni Torrent.

Comunicazione tra processi, Socket

Ogni applicazione di rete è costituita da una coppia di processi che si scambiano messaggi su una rete. Nelle architetture P2P ad esempio il processo di un peer comunica col processo di un altro peer. Questo perché nella realtà non sono gli host che si scambiano messaggi tra loro, bensì i processi che sono eseguiti sugli host.
Per ciascuna coppia di processi comunicanti, se ne distingue uno come **client** e l'altro come **server**: per questo motivo, nell'architettura peer to peer un processo può essere sia client sia server (questo ha anche implicazioni legali nel contesto della pirateria informatica). Lo scambio dei messaggi tra i processi avviene tramite i **socket**.

> *I socket sono interfacce software che permettono di scambiare informazioni.*

Il processo del client apre un socket che identifica un processo chiamante e un processo ricevente. Il client quindi richiede la comunicazione al server stabilendo la comunicazione. Il server poi elabora le informazioni del socket inviato dal client e poi lo invia al client.
Per il socket è necessario il sistema operativo, che detiene le primitive necessarie per gestire il socket.

Indirizzamento dei processi

Per ricevere messaggi, ogni processo deve avere un identificatore. Questo identificatore è l'**indirizzo IP**, un identificatore a 32 bit (IPv4, 128 bit se IPv6). L'identificatore, tuttavia, non identifica il processo, ma l'**host** (calcolatore) su cui viene eseguito il processo. L'indirizzo IP non è un identificatore univoco per il processo.
Oltre all'indirizzo IP viene anche definita la **porta**, che **identifica il servizio a cui si invia la comunicazione**.
Ad esempio, se vogliamo inviare un messaggio HTTP ad un web server, dobbiamo indicare anche la porta per inviare il messaggio HTTP (per HTTP la porta è 80). Per ogni porta vi è una mappatura in base al servizio.
Le porte di internet possono essere divise in 3 gruppi:

- le porte per i servizi di sistema, da 0 a 1023 (ad esempio 443 per HTTPS, 80 per HTTP, 53 per DNS);
- le porte registrate, da 1024 a 49151, assegnate dalla Internet Corporation for Assigned Names and Numbers per specifici usi;
- porte private e/o dinamiche, da 49152 a 65535.

Integrità dei dati

In alcune applicazioni gli scambi di informazioni richiedono che **non si perdano pacchetti** durante la comunicazione. Risulta necessario quindi assicurare l'**integrità dei dati**.
Altre applicazioni richiedono infatti un **throughput** minimo per essere "efficaci", ossia sono **applicazioni sensibili alla banda** (**bandwidth-sensible applications**).

> *Il **throughput** è la quantità di dati scambiati in unità di tempo.*

Per alcune applicazioni inoltre è necessario avere un protocollo che, a livello di trasporto, dia garanzie di temporizzazione, ad esempio che il pacchetto venga ricevuto in un tempo non superiore a più di 100 millisecondi. Queste applicazioni sono **time-sensitive**.
Infine, potrebbe essere necessario fornire ad un'applicazione di rete uno o più servizi di **sicurezza**.
Ad esempio:

- il trasferimento dei file richiede che non ci siano perdite di pacchetti. Il throughput può essere elastico, ma non vi è un'esigenza di avere un minimo throughput. Non è time-sensitive. Un discorso analogo può essere fatto per le mail e i documenti web.
- lo streaming audio e video non richiede obbligatoriamente che non vi siano perdite di pacchetti (si ha una "tolleranza"). Di solito i formati usati per lo streaming di audio e video sono standard della famiglia di formati MPEG. L'MPEG è una traslitterazione con certe sovrapposizioni del JPEG per i video. Molti sistemi di streaming non trasmettono molti frame di differenza quando una scena è prettamente statica. Molti frame invece vengono trasmessi nei cambi di scena. A livello di throughput è necessaria una banda minima di rete per trasmettere l'audio / il video, dipendente dal formato. La trasmissione, inoltre, è time sensitive.

Accenno al Transport Layer e protocolli

Il socket che viene utilizzato per la comunicazione tra processi **funge da interfaccia tra un processo applicativo e il protocollo al livello di trasporto**. I protocolli principalmente utilizzati nel livello di trasporto sono due:

- il **TCP**, una connessione di tipo **reliable** con il **Three Step Handshake**, per cui la connessione tra client e server viene stabilita usando segnali `ACK` come conferma delle connessioni. Nella connessione TCP c'è un controllo di **flusso** e un controllo di

congestione. Non ci sono tuttavia regolamentazioni sul timing e sulla sicurezza: sta all'applicazione introdurre controlli di sicurezza;

- la **UDP**, una connessione **unreliable**. Non vi è la Three Phase Handshake: semplicemente la connessione consiste nell'invio di pacchetti. Non ci sono quindi controlli di flusso e di congestione, né regolamentazioni su timing e sicurezza. Non è affidabile per il mantenimento di sicurezza e di controllo dello stato di invio dei messaggi.
  L'UDP sembra apparentemente inutile, ma in realtà trova applicazioni nella telefonia internet, ossia il **Voice Over IP**. Quando si effettua una chiamata mediante segnali digitali (non la chiamata normale, quindi, che utilizza segnali analogici), si usa il router per instradare i pacchetti. Non è necessario che prima si stabilisca la connessione e poi si scambiano i dati: i dati inviati (in questo caso la voce) stabiliscono già la connessione. Le telefonate di conseguenza sono meno sicure, possono essere intercettate e non c'è controllo di flusso.

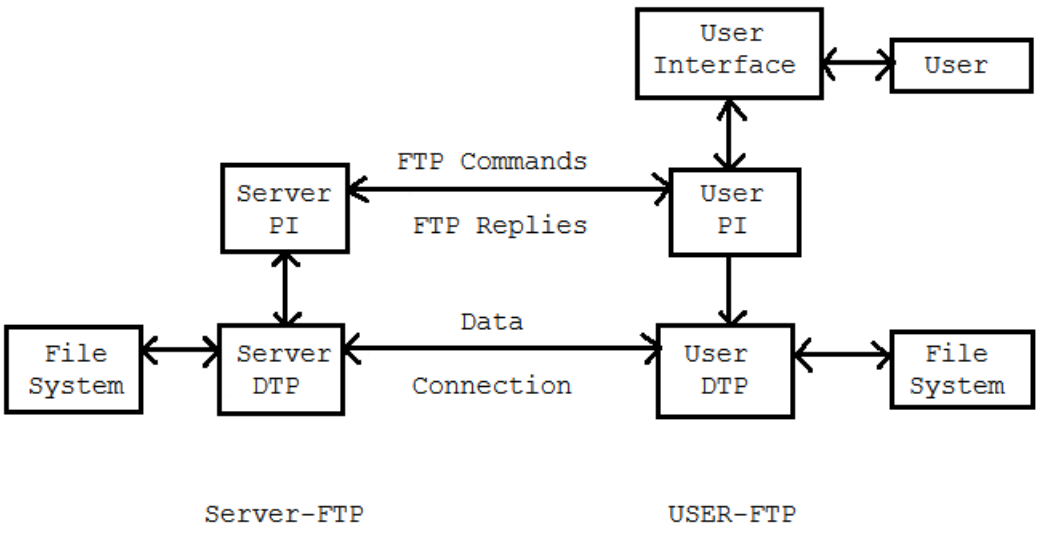
Applicazioni note

Telnet

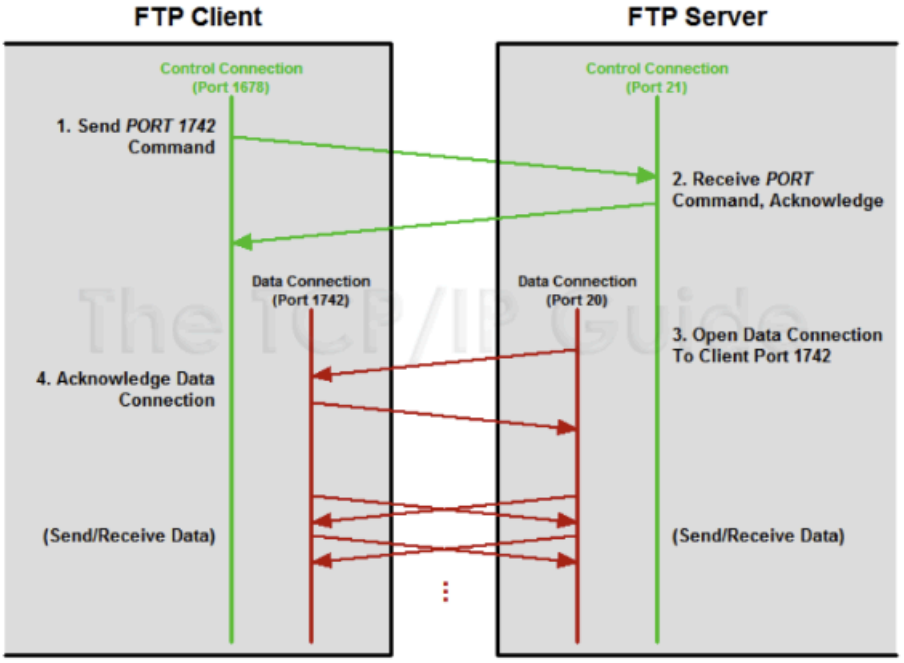
Il **telnet** è un'applicazione che permette di connettersi fisicamente ad un'altra parte. In una shell del sistema operativo, talvolta abbiamo necessità di connetterci ad un altro computer, facendo sì che il terminale che si apre localmente operi sull'altro dispositivo. La shell che si vede localmente è uguale quindi alla shell del server. Lo shell locale diventa un **mirroring** dello shell del server. Tutti i servizi **abilitati** (per questione di sicurezza, nel sistema operativo si può stabilire quali programmi sono accessibili tramite connessione telnet) vengono attivati sul lato server: le operazioni eseguite quindi sono eseguite sul server, e il responso viene inviato.
Il telnet opera sul livello di applicazione del modello OSI. Il server che riceve la connessione telnet e il client che richiede la connessione devono obbligatoriamente avere i driver necessari per stabilire la connessione.
Supponiamo un host A vuole collegarsi ad un host B. Si stabilisce una connessione TCP con il Three Way Handshake. Un primo pacchetto `SYN` viene inviato verso host B. Se la connessione viene accettata, un segnale `SYN` `ACK`, ossia il pacchetto `SYN` a cui viene aggiunto un pacchetto `ACK`, viene inviato da B ad A. Infine, il segnale `ACK` viene inviato da A a B. Dopo che la connessione viene stabilita, l'host A invia un pacchetto a B, ad esempio la lettera "C", inoltrando una richiesta di inserire "C" nello standard input. L'host B invia indietro il carattere "C" nel terminale dell'host A.
Quando la connessione non ha un throughput sufficiente, si vede latenza nella digitazione dei comandi e dei caratteri sul terminale.
Un problema del telnet tuttavia è la sicurezza. Il telnet nelle prime versioni manda i dati in chiaro, ossia **senza crittografare**. Se un agente esterno intercetta l'invio dei pacchetti, allora può ad esempio catturare i pacchetti contenenti le credenziali. Una versione più sicura del telnet che si usa è la connessione **SSH** (**Secure Shell**).
Il problema principale di telnet è proprio la sicurezza, come anticipato precedentemente. In particolare, si pone il problema nel momento in cui bisogna inviare token di autenticazione attraverso una rete pubblica. SSH mitiga il rischio usando un meccanismo di codifica per occultare i contenuti, anche se un agente di terze parti ha accesso all'intero stream di dati.

FTP

Un'altra applicazione nota è **FTP**. Mentre il telnet nasce fondamentalmente per permettere all'utente di eseguire applicazioni in remoto, il protocollo FTP viene utilizzato per il trasferimento dei file. Dal file system del server i file vengono trasferiti in pacchetti. Una volta che abbiamo accesso al file system del server, si può avere accesso tuttavia a tutto quello contenuto nel file system. L'accesso allora dev'essere limitato in maniera opportuna. La sicurezza con l'FTP quindi è fondamentale.
A differenza degli altri protocolli, FTP utilizza due connessioni separate per gestire comandi e dati, utilizzando il seguente modello:



Nel modello la sigla "PI" indica il **Protocol Interpreter**, ossia l'**interprete del protocollo**, utilizzato dal client e dal server per lo scambio di comandi e risposte. "DTP", invece, è il **Data Transfer Process**, ossia il **processo di trasferimento dati**, utilizzando per lo scambio di dati. I due elementi vengono solitamente chiamati "**canale comandi**" e "**canale dati**".
Le connessioni di conseguenza sono due: la prima è la connessione TCP per il trasferimento dei dati sul canale dati, la seconda è una **connessione di controllo** sul canale comandi. Nella connessione di controllo l'FTP client apre una porta solitamente casuale e tramite il canale comandi rende noto il numero della porta al server, inviando un comando `PORT`. Il server quindi attiva la connessione dati al client FTP attivando la porta 20 e invia un segnale `ACK` sulla connessione di controllo. A questo punto il client FTP effettua il binding alla porta 20, quindi il server apre la connessione TCP dati utilizzando sempre un Three Phase Handshake.



Prima della connessione di controllo chiaramente la connessione TCP garantisce l'autenticazione.
L'efficienza e la stabilità della connessione tuttavia diminuisce per garantire che la connessione sia affidabile.

02 - Protocolli

HTTP

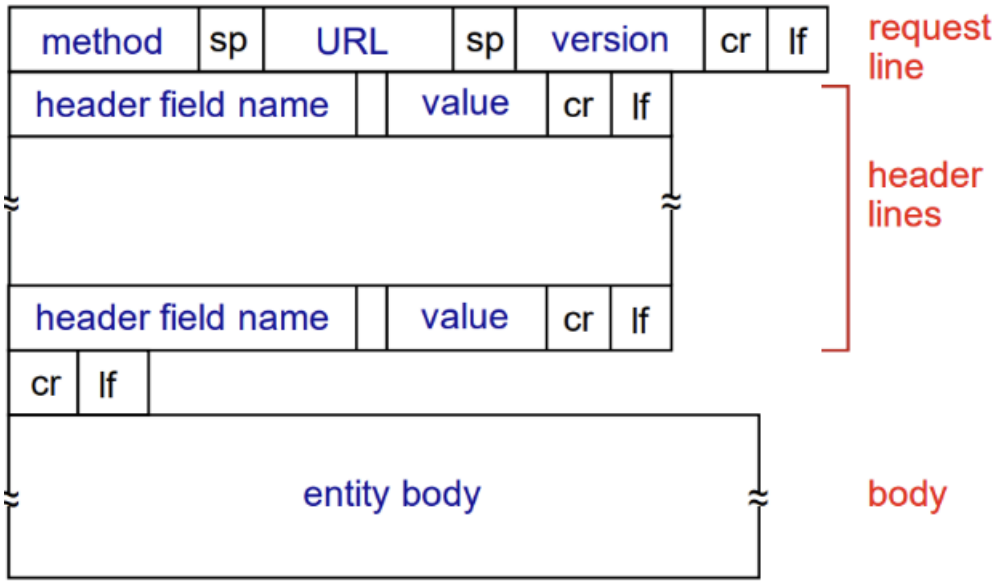
HTTP è un protocollo usato nel livello di applicazione che definisce in che modo i client web richiedono le pagine e gli oggetti referenziati da queste ai web server. Con il termine oggetto si intende genericamente un file multimediale, link o anche una struttura ad albero. Quando ciò accade, utilizziamo l'URL (**Uniform Resource Locator**) che è un alias per rendere user-friendly il contatto con il server: se ciò non esistesse, risulterebbe necessario avere l'indirizzo IP del sito a cui voglio accedere.
La connessione HTTP utilizza il paradigma client-server ed utilizza il protocollo TCP. Il client inizializza la connessione TCP (ossia crea il socket) per il server nella porta 80. Il server accetta la connessione TCP dal client. Infine vengono scambiati i messaggi HTTP e la connessione viene chiusa.
La comunicazione chiaramente richiede del tempo. Il tempo necessario per la connessione prende il nome di **RTT** (**Round Trip Time**). Tale tempo può aumentare in base alla quantità di dati da trasmettere (dipendente dal numero di oggetti presenti nella pagina web da trasmettere).
Se la pagina HTML contiene riferimenti ad altri oggetti, allora gli oggetti possono essere richiesti in un secondo momento usando HTTP. In particolare:

- HTTP 1.0 chiude ogni connessione TCP dopo che l'oggetto viene trasferito;
- HTTP 1.1 usa una connessione TCP persistente, e gli oggetti vengono inviati in pipeline, che verrà chiusa solo alla fine;
- HTTP 2.0 invece usa una compressione degli header.

Supponiamo un client voglia fare una richiesta HTTP ad un server. In un messaggio HTTP di richiesta bisogna avere una **riga di richiesta** (**request line**) e delle **righe di intestazione** (**header lines**). Una riga di richiesta a sua volta è costituita da tre campi:

- un campo **metodo**, che specifica cosa voler fare;
- un campo **URL**, ossia il sito da contattare;
- un campo **versione** del protocollo HTTP che stiamo utilizzando;

Per le righe di intestazione si ha un campo **nome** e un campo **valore**. Infine ci sono i **dati** del messaggio.



Header

> *Un **header** è un pre-dato di formattazione che serve a preparare colui che riceve l'informazione in relazione a ciò che verrà dopo.*

Un esempio di header è l'estensione di un file. Il sistema operativo oltre a verificare l'estensione verifica l'header per capire qual è il tipo di file.
Gli header sono utilizzati in HTTP.

Messaggi HTTP

Tipici messaggi HTTP sono:

- metodo `GET`, usato per chiedere al server di recuperare una risorsa al path (ossia l'URL) specificato;

- metodo `POST`, dove l'input dell'utente viene inviato dal client al server nel corpo dell'entità del messaggio HTTP `POST`. Più precisamente si chiede al server di eseguire un'azione specificata dall'URL fornendo i parametri nel body del messaggio;
- metodo `HEAD`, usato per richiedere gli header che verrebbero restituiti usando una richiesta GET;
- metodo `PUT`, usato per caricare un oggetto nel server. Può anche rimpiazzare un file che esiste in uno specifico URL con il contenuto del corpo del messaggio HTTP `PUT`.

Dopo l'invio di un messaggio HTTP si ha una **risposta** dal server HTTP. Il messaggio di risposta presenta le seguenti informazioni:

- la **versione** HTTP;
- il **codice di stato**;
- il **testo**;
- gli **header** con field name e value.

Codici di stato noti sono:

- 100: Continue;
- **200: OK**;
- 203: Non-Authoritative Information;
- 204: No content;
- 305: Use proxy;
- **400: Bad request**;
- **403: Forbidden**;
- **404: Not Found**;
- 405: Method Not Allowed;
- 500: Internal Server Error;
- **505: HTTP Version not supported**.

Stateless, Stateful

Ogni protocollo o algoritmo può essere definito come **stateless** o **stateful**, ossia protocolli **con o senza memoria di stato**. HTTP ad esempio è **stateless** in quanto vediamo solamente lo stato finale e non si ha la necessità di conservare gli stati intermedi (pertanto un server non ha informazioni riguardanti richieste passate da parte del client). Se, ad esempio, consideriamo l'acquisizione di un ticket in virtù dei dati inseriti, abbiamo bisogno di conservare gli stati intermedi, in quanto l'erogazione del ticket dipende dai dati inseriti.
La conservazione degli stati risulta importante quando è necessario avere le informazioni degli stati intermedi. In tal caso, il protocollo (o algoritmo) si dice **stateful**. I protocolli di tipo stateful, tuttavia, devono essere mantenuti correttamente: se il server o il client va in crash, lo stato dev'essere mantenuto in maniera coerente con l'esito delle operazioni.
In una connessione stateful quindi è possibile ripartire dal punto in cui è avvenuto un crash.

Cookie

I siti web e i browser utilizzano i **cookie** per mantenere alcuni stati durante le transazioni. Permettono di fornire a protocolli stateless funzionalità stateful, quindi di tracciare alcuni dati delle transazioni.
Supponiamo ad esempio di effettuare una richiesta di navigazione ad Amazon, accettando la richiesta di cookie. La prima volta che viene eseguita una richiesta, viene notificata la creazione di un cookie ID, un identificativo unico, che viene aggiunto ad un **database** sul sito. Più precisamente, il server include nella risposta HTTP l'intestazione `Set-cookie: <id>`, contenente l'identificativo. Allora il browser aggiunge una riga al **file dei cookie** che esso gestisce. Una volta che viene creato un cookie ID, **fino a quando i cookie non sono eliminati dal client browser**, Amazon riconoscerà il nostro dispositivo: infatti ogni richiesta inoltrata dal browser al server Amazon includerà l'header `Cookie: <id>`.
I cookie trovano applicazioni ad esempio:

- per le autorizzazioni dei dispositivi;
- memorizzazione dei carrelli (e-commerce);
- consigli su prodotti;
- sessione degli utenti.

Consideriamo il caso studio. Facciamo un HTTP GET per prendere un file HTML. Supponiamo che sulla pagina sono presenti delle pubblicità, che sono dei **sotto-siti incorporati**. Se accettiamo i cookie sulla pagina web da cui abbiamo richiesto la pagina HTML, accettiamo anche i cookie non diretti, che sono **cookie di terze parti**: la pagina web della pubblicità non è stata visitata, tuttavia

abbiamo accettato i cookie anche della pagina.
I siti che non abbiamo visitato quindi hanno un back-end grazie ai cookie di terze parti. Alcuni browser permettono di bloccare quelle di terze parti: ad esempio, Firefox e Safari disattivano di default il tracciamento dai cookie di terze parti.
Il GDPR (**General Data Protection Regulation**, in Europa) sottolinea il fatto che i dati sensibili di una persona non possono essere scambiati liberalmente. In particolare, quando i cookie possono identificare un individuo, i cookie possono essere considerati come **dati personali**, e quindi sono soggetti alle regolamentazioni del GDPR.

DHTML

Possiamo anche creare pagine web che siano dinamiche. In questo caso il form riempito dall'utente e fornito al server viene passato ad un **CGI Script**, che fa una query al database e in base ai dati ritrovati nel database costruisce la pagina HTML.

Versione HTTP

Sia che si utilizzi DHTML che HTML statico, si possono usare diverse versioni di HTTP.
Con HTTP 1.1 è possibile inserire in pipeline gli oggetti da restituire utilizzando un'unica connessione TCP. Il server risponde in modalità FIFO o FCFS (First Come First Served) alle richieste GET. Tuttavia è possibile che con la FCFS si verifichi un **Head Of Line** (**HOL**) **blocking**. Supponiamo che un oggetto pesi molto di più rispetto ad altri: allora tale oggetto blocca la coda.
HTTP 2.0 risolve il problema in maniera intelligente. Le richieste non vengono gestite con un criterio FIFO, ma con un criterio basato su determinate priorità, stabilite dal protocollo. Gli oggetti vengono spezzettati in **slices** (letteralmente "tagli") e vengono inviati al client. In questo modo non verranno bloccati gli altri oggetti: gli oggetti più pesanti quindi verranno inviati a slice senza limitare l'invio di altri elementi. Questo inoltre evita l'apertura di più connessioni TCP parallele, soluzione inizialmente applicata dai browser che supportavano HTTP/1.1.
HTTP 3.0 invece aggiunge più sicurezza, errori per oggetto e controllo della congestione tramite più pipeline tramite UDP.

Proxy

Un **server proxy** è uno strumento che permette di filtrare o mascherare l'accesso a internet da parte di un client. Sono strumenti importantissimi nelle aziende: se un dipendente va su siti non autorizzati, dev'essere obbligatoriamente tracciato.
Il proxy riceve n client in ingresso e inoltra le richieste al server a cui i client si vogliono collegare. Funge quindi da server per i client che si vogliono connettere attraverso il proxy e da client per inoltrare le richieste al posto degli host che sono connessi al proxy.

SMTP

Il protocollo **SMTP** (**Simple Mail Transfer Protocol**) è utilizzato per l'invio delle email. Le tre componenti principali di un sistema postale di internet sono le seguenti:

- **agenti utente** (**user agents**), che consentono agli utenti di leggere, rispondere, inoltrare, salvare e comporre i messaggi (ad esempio Outlook, Gmail, Mozilla Thunderbird);
- **mail server**, dove sono memorizzate le mail salvate per l'utente;
- **protocollo SMTP**, che utilizza il paradigma client-server.

Uno scenario tipico di uno scambio di messaggi in SMTP è il seguente:

1. un utente A invoca il proprio user agent per comporre il messaggio e inviarlo ad un utente B;
2. lo user agent di A invia il messaggio al suo mail server, dove è collocato in una **coda di messaggi**;
3. il lato client di SMTP vede il messaggio nella coda di messaggi e apre una connessione TCP verso un server SMTP in esecuzione sul mail server di B;
4. il client SMTP invia il messaggio di A sulla connessione TCP dopo l'handshake;
5. presso il mail server di B, il lato server di SMTP riceve il messaggio, che viene memorizzato nella casella dell'utente B;
6. quando lo desidera, l'utente B invoca il suo user agent per leggere il messaggio.

Dopo che viene stabilita una connessione TCP, uno scambio di messaggi tra un server (S) ed un client (C) è del tipo:

```
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
```

I comandi utilizzati in SMTP sono vicini all'inglese: `HELO` (abbreviazione di "hello"), `MAIL FROM`, `RCPT TO`, `DATA` e `QUIT` specificano l'avvio di una connessione, il mittente della mail, il destinatario, il messaggio da inviare e la terminazione della connessione. Se ci sono più messaggi da inviare, si utilizza sempre la stessa connessione TCP. Pertanto la connessione TCP è persistente e il comando `QUIT` viene inviato solo dopo aver spedito tutti i messaggi.
Le informazioni che vengono mandate sono in ASCII e la risposta contiene uno status code e una frase. Gli allegati che non sono in ASCII vengono inviati usando il **protocollo MIME** che caratterizza allegati non testuali. Inoltre, i trasferimenti SMTP avvengono direttamente tra server, da un mail server ad un altro senza nessun server intermedio.
Come in HTTP, anche in SMTP vengono utilizzati degli header, separati dal corpo del messaggio mediante una riga senza contenuto e specificati nell'RFC 5322.

POP3 e IMAP

SMTP è un protocollo *push*, ossia un protocollo per l'invio delle mail. Se si vuole invocare il proprio user agent per accedere ai messaggi, si può utilizzare il **protocollo POP3** o il **protocollo IMAP**. Vi sono varie differenze tra i due protocolli, ma la principale è il salvataggio delle mail, che avviene sul computer dell'utente nel caso di POP3 e online nel caso di IMAP. Le letture delle mail, inoltre, sono offline con POP3 e online con IMAP. A livello di backup inoltre, POP3 richiede che sia l'utente ad eseguire il backup, mentre è l'ISP a farlo per IMAP. Infine, POP3 è più semplice da implementare rispetto a IMAP.

03 - DNS

Il **DNS** (**Domain Name System**) è un sistema che permette la traduzione di nomi di dominio leggibili dagli esseri umani in indirizzi IP per identificare i server su Internet. Nel sistema di routing e indicizzazione dei siti web ciò che conta è proprio l'indirizzo IP.
I servizi che offre DNS sono:

- **traduzione hostname-indirizzo IP**;
- l'**host aliasing**. Nella traduzione di un hostname, distinguiamo:
  - **nome canonico**, ossia l'hostname completo che identifica la risorsa a cui accedere;
  - **nomi alternativi o alias**, costruiti sopra il canonico e spesso più semplici da ricordare e digitare;
- **aliasing del mail server**;
- **distribuzione di carico**, quando con la replicazione degli indirizzi IP di un server vogliamo far corrispondere un alias a vari IP.

Risulta necessario avere più DNS nel mondo per vari motivi:

- ricoprire le varie aree del mondo;
- se il DNS va giù, internet sarebbe inaccessibile (senza conoscere gli indirizzi IP);
- il volume di traffico è enorme, con miliardi di richieste di translation di URL al giorno;
- per questioni di distanze fisiche tra i vari dispositivi che si connettono al DNS. Ad esempio, per le applicazioni militari o finanziarie è necessario avere tempi anche molto brevi di connessione;
- la manutenzione del DNS bloccherebbe l'accesso a internet.

Risoluzione

Il meccanismo di risoluzione segue una politica gerarchica ad albero. La prima cosa che viene fatta è utilizzare la **cache locale** del client che vuole connettersi al server; tale cache può essere anche in un host locale. Se la ricerca dà esito positivo, ossia esiste un'associazione tra indirizzo IP e URL inserito, allora l'URL viene risolto immediatamente. Se invece nella cache non viene risolto l'URL, la richiesta viene inoltrata ad un **DNS resolver** o **local DNS server**, un server che verifica se abbia già l'associazione all'URL inserito memorizzata. Se l'esito è negativo, allora il DNS resolver inoltra una richiesta ad uno dei 13 **root DNS server**, che individua il dominio dell'URL (che è la parte finale dell'URL, tipo `.it` o `.com`). Si scende quindi di livello nella gerarchia DNS, accedendo al **Top Level Domain** server corrispondente al dominio dell'URL inserito. Dal Top Level Domain viene contattato il **server DNS autoritativo** contenente l'indirizzo IP dell'URL. Il DNS autoritativo è un DNS dell'organizzazione che memorizza le associazioni nome host - indirizzo IP.
Il DNS resolver non appartiene alla gerarchia di server, ma è comunque centrale nell'architettura DNS. Quando un host si connette ad un ISP, quest'ultimo gli fornisce un indirizzo IP da un suo DNS locale.

Distribuzione dei server

I 13 server root DNS sono replicate in tutte le nazioni. L'associazione che supervisiona i DNS è la DNSSEC.

DNS Primari e secondari

Le possibilità che un DNS primario vada in manutenzione o che ci siano dei problemi di sovraccarico o risposte non corrette è elevato. Si utilizza allora un caching del DNS primario, ossia il **DNS secondario**.
Nel momento in cui arrivano richieste al DNS primario, si verificano controlli di congestione. Si esegue così un bilanciamento di carico, in cui le risposte vengono smistate e si fa routing al DNS secondario. Il vantaggio della cache permette di equilibrare il carico di richieste. Tuttavia il DNS secondario dev'essere **sempre** uguale al DNS primario: se il DNS secondario non viene aggiornato, allora potrebbe associare ad un URL diversi indirizzi IP.
L'aggiornamento avviene quindi a intervalli di tempo costante e ogniqualvolta il DNS primario viene aggiornato. Con questi due meccanismi si ha una maggiore sicurezza che il DNS secondario sia aggiornato. Non siamo totalmente sicuri perché è possibile inviare una richiesta che viene smistata al DNS secondario nel momento in cui il DNS primario si sta aggiornando. Si tratta tuttavia di una situazione al limite, e dal punto di vista statistico la probabilità che tale evento accada è molto bassa.

Indicizzazione (o DNS Query)

Indicizzazione iterativa

La prima opzione per risolvere il nome di dominio è l'**approccio iterativo**. Supponiamo non vi sia l'associazione nel local DNS server. Il DNS resolver allora contatta il root DNS server, che comunica di contattare il Top Level Domain corrispondente. Ad ogni passo quindi il server contattato restituisce il nome del server da contattare.

Il vantaggio è chiaramente quello di **ridurre il carico** sul singolo server DNS. Tuttavia lo svantaggio è quello di dover attendere le risposte a partire dai server.

Indicizzazione ricorsiva

In alternativa è possibile adottare un **approccio ricorsivo**. Si contatta prima il local DNS server: se esso ha l'indirizzo IP, allora l'operazione si conclude. Altrimenti, il local DNS server inoltra la richiesta al root DNS server, che contatta il Top Level Domain DNS, che poi contatta il DNS autoritativo.
Il vantaggio è quello di ridurre il carico sul DNS locale, che non deve gestire le varie risposte ai server DNS diversi. Tuttavia, lo svantaggio risiede nel fatto che i DNS più alti in gerarchia hanno **maggiore responsabilità logica**, in quanto devono restituire le informazioni ricavate dai DNS più in basso nella gerarchia.

Cache locale

Un altro problema da tenere in considerazione è la **cache coherency**, ossia la **coerenza della cache con il server DNS**. Infatti l'host contatta inizialmente la cache, e non direttamente il server DNS.
All'interno del sistema operativo esistono sistemi di **cache refresh**, aggiornando l'informazione dalla memoria centrale alla cache. Similmente funziona per le reti. Per ogni entry del DNS in ogni server viene stabilito un tempo chiamato **TTL** (**Time To Live**), associato al record DNS che indica per quanto tempo il resolver DNS può mantenere nella cache una risposta prima di doverla aggiornare da un server autoritativo. Il resolver quindi contatta il server ad esso sovrastante per capire se il dato è stato aggiornato o meno. Le modalità di aggiornamento sono specificate nello standard RFC 2136.
A livello applicativo un **daemon** gestisce l'aggiornamento del resolver.

Resource Record (RR)

All'interno del server DNS sono memorizzati i **Resource Record** (**RR**). Il formato è del tipo `name, value, type, ttl`. In virtù del type, `name` e `value` assumono diverso significato:

- se `type = A`, `name` è l'**hostname** e `value` è l'**indirizzo IP**. Sono memorizzati nei server autoritativi;
- se `type = NS`, `name` è il dominio e `value` è l'**hostname** del DNS autoritativo di quel dominio. Sono memorizzati nei Top Level Domain server;
- se `type = CNAME`, `name` è l'alias del nome **canonico**, e `value` è il **nome canonico**. Ad esempio, `name = www.ibm.com` e `value = servereast.backup2.ibm.com`;
- se `type = MX`, `value` è il nome del mail-server associato al `name`.

Problemi di sicurezza dei DNS

Uno dei problemi più grossi dei DNS sono gli attacchi denominati **DoS**, ossia **Denial of Service**, che violano il servizio **bombardando il root server con del traffico**. Il root server viene riempito di tantissime richieste fittizie di risoluzioni di IP. Il server potrebbe non riuscire a sopportare tutte le richieste, causando un disservizio. Per tamponare tali richieste, si fa un filtraggio al traffico, ad esempio osservando la semantica dell'URL. Oggi si utilizzano dei motori basati su intelligenza artificiale per analizzare la semantica dell'URL. Si utilizzano anche dei bypass, usando dei proxy per cui se le richieste provengono da determinati IP, allora si sa già che non si devono considerare.
Un altro attacco pericoloso della stessa tipologia è il **bombardamento dei TLD server**, facendo crollare la risoluzione dei siti di un certo dominio. Tuttavia, poiché i server sono riprodotti molto spesso, sono più difficili.

Differenti sono gli attacchi **man-in-the-middle** (**MITM**) che intercettano le query ai server DNS. Il **DNS Poisoning**, una sorta di spoofing, consiste nel porsi tra l'autoritativo e il client richiedente, sostituendo la risposta dell'autoritativo con una risposta fittizia, restituendo quindi un IP differente da quello che il DNS doveva restituire.

Ci sono diverse tecniche per difendersi da questi attacchi. Una tecnica è quella di controllare i **tempi di risposta**, creando profili per tecnologie a intelligenza artificiale per profilare certi utenti e indicarli come utenti affidabili o meno.

Network Management

SNMP

Il **Simple Network Management Protocol** permette la creazione di sistemi per la gestione di una rete. Si tratta di un protocollo per consentire ad amministratori di una rete di dispositivi di raccogliere informazioni riguardante questa.
In un'architettura SNMP vi sono tre principali componenti:

1. **Manager SNMP**: si tratta di un sistema centralizzato per monitorare la rete. Prende anche il nome di **Network Manager Station** (**NMS**). Un host che esegue il client SNMP prende il nome di **manager**.
2. **Agente SNMP**: è un modulo software installato su ogni dispositivo da monitorare all'interno della rete di calcolatori. Il manager accede ai valori memorizzati nel database, mentre l'agente mantiene le informazioni del database.
3. **Management Information Base (MIB)**: è costituita dalle informazioni sulle risorse monitorate. SNMP di per sé non definisce quali variabili devono essere monitorate dal sistema; piuttosto, utilizza un design ampliabile che permette alle applicazioni di definire delle proprie gerarchie: queste gerarchie sono le MIB.

I messaggi SNMP vengono trasmessi tramite il protocollo UDP attraverso la porta 161. Il manager può mandare richieste da qualsiasi porta verso la porta 161 dell'agente. La risposta da parte dell'agente viene restituita nella porta che il manager ha utilizzato per inviare la richiesta. Il manager può anche ricevere delle notifiche (le **traps**) sulla porta 162: queste notifiche vengono generate dagli agenti in caso, ad esempio, di malfunzionamenti in maniera del tutto asincrona. Messaggi tipicamente inviati da un manager verso un agente sono:

- `GetRequest`, ossia la richiesta di recupero di un valore di una variabile;
- `SetRequest`, ossia la richiesta di cambiamento di un valore di una variabile;
- `GetNextRequest`, ossia la richiesta di recupero di possibili variabili e dei loro valori.

Un agente invece invia i seguenti messaggi;

- `GetResponse`, ossia restituisce i valori di una o più variabili;
- `Trap`, ossia notifica in maniera asincrona e **senza necessaria richiesta da parte del manager** situazioni anomale o eventi significativi nella rete.

Esistono tre versioni di SNMP, con la prima versione criticata per la sua scarsa sicurezza, la seconda che migliorò le performance e anche parzialmente la sicurezza, e la terza che modificò radicalmente la sicurezza del protocollo, offrendo una crittografia per i dati e un forte sistema di autenticazione.

04 - Transport Layer

Il transport layer si occupa di comunicazione **puramente a livello logico**. Col transport layer a livello application il sender aggiunge header e manda i pacchetti al receiver.
Vengono distinte due principali entità in questo livello:

- il **mittente**, che divide i messaggi in **segmenti** passati al **network layer**;
- il **destinatario**, che assembla i segmenti ricevuti in un unico messaggio e li passa al livello di applicazione.

Si utilizzano due protocolli: il **TCP** e l'**UDP**.
Ricordiamo che:

- il **Data Link Layer** è il layer riguardante la comunicazione **fisica** tra host connessi fisicamente;
- il **Network Layer** è il layer che crea la **comunicazione logica** attraverso gli IP **tra gli host**;
- il **Transport Layer** gestisce la **comunicazione logica tra i processi**.

Il transport layer seleziona il processo attraverso il meccanismo di socket per la comunicazione interprocess. Altrimenti avremo processi che comunicano in maniera disorganizzata. Risulta necessario quindi avere un'uniformità di comunicazione.

Multiplexing, demultiplexing

Quanto illustrato avviene tramite processi di **multiplexing** e **demultiplexing**.
Supponiamo di avere un processo client P che comunica con un server. Allora invia una richiesta di comunicazione, attraversando application, transport, network e physical layer. Il processo P attiva un **socket** per far comunicare il thread corretto del processo. Questo è tipico ad esempio dei browser, dove ogni tab del browser corrisponde ad un thread diverso. Ogni tab ha un determinato compito, richiedendo precisi dati a dei server. Pertanto ogni thread ha un socket per comunicare correttamente con il server destinatario.
L'host client, quindi, deve radunare frammenti di dati da socket diverse ed incapsulare ogni frammento con header per il livello di trasporto al fine di creare **segmenti** e passarli al layer di rete. Questo processo prende il nome di **multiplexing**.
L'host in ricezione indirizza verso la socket appropriata ogni segmento a livello di trasporto in arrivo. Ogni segmento ha infatti vari campi deputati allo scopo, e dal lato server il livello di trasporto analizza questi campi per identificare la socket di ricezione. Il compito di trasportare i dati dei segmenti a livello di trasporto verso la giusta socket prende il nome di **demultiplexing**.
Per identificare il server si utilizza l'indirizzo IP e la porta.

UDP

Non sempre si ha la necessità di creare una connessione affidabile. Ci sono delle applicazioni, come visto precedentemente, per cui non si ha la necessità di dover creare una connessione stabile, ma si contempla la possibilità di perdere pacchetti.
Esiste un protocollo di comunicazione meno reliable di TCP, ossia l'**UDP**. L'header di un segmento UDP prevede:

- 16 bit per la source port;
- 16 bit per la destination port;
- 16 bit per la lunghezza del messaggio;
- 16 bit per il checksum.

Il checksum è un meccanismo che permette di verificare se l'informazione è stata ricevuta correttamente o meno.
L'UDP è un sistema essenziale, contenente i servizi necessari per mandare i pacchetti, senza precisare neanche l'ordine di invio di questi pacchetti. Ogni segmento viene mandato indipendentemente. Non vi è un RTT perché non c'è necessità di stabilire una connessione, gli header sono di piccola dimensione e non ci sono controlli di congestione.
L'UDP trova applicazioni per lo streaming multimediale, in quanto si ha tolleranza per la perdita di pacchetti per i meccanismi usati, come la compressione MPEG. Viene anche utilizzato per il DNS, SNMP e per HTTP/3. HTTP/3 in particolare per il trasferimento UDP aggiunge alcuni controlli di congestione e di affidabilità a livello applicativo.

Checksum

Ci sono diversi meccanismi di checksum. Ne mostreremo uno più semplice.
Per verificare che i dati sono stati inviati correttamente (un errore tipico sono i **flipped bits**, ossia bit invertiti), si utilizza un'operazione matematica per indicare qual è il risultato della somma delle informazioni che stiamo inviando se il contenuto è corretto. Se le informazioni che stiamo trasferendo ha checksum pari a 11, e devono essere ricevuti due numeri la cui somma deve fare 11, allora nel momento in cui la somma di tali numeri è diversa da 11 viene richiesto di inviare nuovamente i pacchetti.

Il checksum tuttavia dev'essere trasmesso, pertanto il checksum può anche essere affetto dello stesso problema che deve controllare che non esista. Generalmente il checksum viene inviato con un API più robusta, utilizzando un'instanziazione di invio di messaggio realizzata apposta per il checksum. Inoltre il problema del checksum è che è sintattico **ma non semantico**: d'altronde la somma 11 la possiamo ottenere facendo $5+6$, $3+8$, $4+7$ e così via. Di conseguenza è un meccanismo che sicuramente permette un controllo, ma è piuttosto superficiale.

Trasferimento affidabile

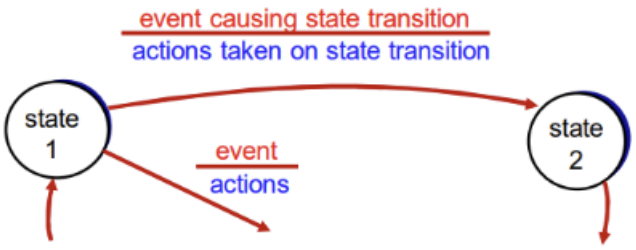
L'ideale sarebbe avere un canale affidabile che presi i pacchetti con gli header li invii da un client ad un server. Nella realtà è impossibile avere un canale totalmente affidabile, perché ci saranno quasi sempre sbalzi di tensione, flipped bits e segnali rumore. Ci sono anche dei limiti fisici dei mezzi utilizzati.
Il problema generale quindi è fornire un **protocollo di trasferimento dati affidabile**, ossia un **reliable data transfer protocol**, in grado di implementare un'astrazione di un canale affidabile per la comunicazione dei dati su un livello superiore, posto al di sopra di un canale che è inaffidabile.
Si può tuttavia lavorare a livello di software per gestire l'inaffidabilità del canale comunicativo. Questo non riduce a 0 i rischi della comunicazione unreliable, ma rende molto più robusta la connessione.
Inoltre, **il sender e il receiver non conoscono lo stato dell'altro**. Ciò è possibile solamente con messaggi del tipo SYN e ACK. Ma questi messaggi devono essere inviati correttamente: anche questi meccanismi devono sottostare ai problemi che cercano di risolvere.

Reliable Data Transfer Protocol

Abbiamo diverse versioni, in virtù delle considerazioni riguardante il canale sottostante:

1. **RDT 1.0**: assume che il canale di invio e di ricezione del messaggio non presentino errori, senza utilizzare segnali ACK e NAK e senza packet loss. Si tratta di un sistema puramente teorico chiaramente, che si basa su una situazione del tutto idilliaca;
2. **RDT 2.0**: assume che ci possano essere errori nella trasmissione dei messaggi, ma non nella ricezione. Il controllo è caratterizzato tramite utilizzo di segnali ACK e NAK. Il problema nell'RDT 2.0 è che si assume che non ci siano errori di ricezione, non tenendo conto quindi di una possibile perdita di un segnale ACK;
3. **RDT 2.1**: è un miglioramento del 2.0 che assume che ci possano essere errori nel canale di ricezione. Questi errori vengono gestiti con un meccanismo di **temporizzazione** o di **ripetizione** dei segnali ACK / NAK. Aumenta così la reliability, ma si impatta la banda di rete;
4. **RDT 2.2**: utilizza solo un segnale ACK, in particolare una comunicazione continua di segnali ACK. Ogni pacchetto viene inviato con un segnale ACK corrispondente, e se un secondo segnale ACK non viene ricevuto, il pacchetto viene inviato nuovamente;
5. **RDT 3.0**: introduce un meccanismo di **timeout** inserendo il concetto di perdita di pacchetti.

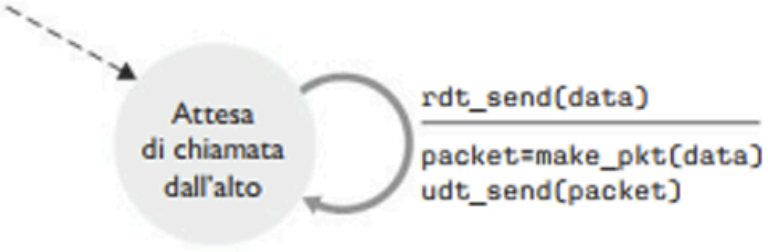
Il meccanismo delle RDT si basa sull'utilizzo di **macchine a stati finiti discrete**. Infatti è necessario sviluppare in maniera crescente il lato sender e receiver del RDT.
La macchina a stati finiti gestisce gli stati attivati da eventi. L'evento è indicato sopra la linea orizzontale, mentre le azioni intraprese in seguito all'evento sono specificate sotto la linea. Ogni cambio di stato è attivato da un evento.
Vogliamo caratterizzare un meccanismo software che permette la gestione del canale forward, backward ed eventuali segnali ACK e NACK.



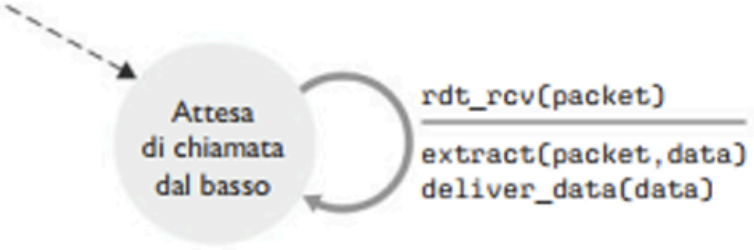
Trasferimento dati affidabile su un canale perfettamente affidabile: RDT 1.0

Il protocollo di versione 1.0 presuppone un canale perfetto senza distorsione dei bit e senza perdita di pacchetti. Non c'è bisogno quindi di rinvio di pacchetti né di segnali ACK e NAK.
Si utilizza una FSM per il sender e una FSM per il receiver. In particolare, il sender ha uno stato "**Wait for call from above**", mentre il receiver ha uno stato "**Wait for call from below**".
Il meccanismo con cui sender e receiver rimangono in attesa nello stesso stato si chiama **polling**. Lo stato di polling richiederebbe

che la CPU controlli un indirizzo di memoria in cui si aspetta qualche valore specifico. Questo tuttavia comporta un locking del bus di sistema. Si inserisce allora un **delay** che limita il numero di accessi al bus da parte della CPU. Se non si inserisce il delay, nessun altro evento potrà essere gestito. Quando osserviamo la macchina a stati, si tiene conto quindi del fatto che la CPU blocca il bus, e quindi il nostro codice non riesce a gestire gli eventi.



a. rdt1.0: lato mittente



b. rdt1.0: lato ricevente

Trasferimento dati affidabile su un canale con errori sui bit: RDT 2.0

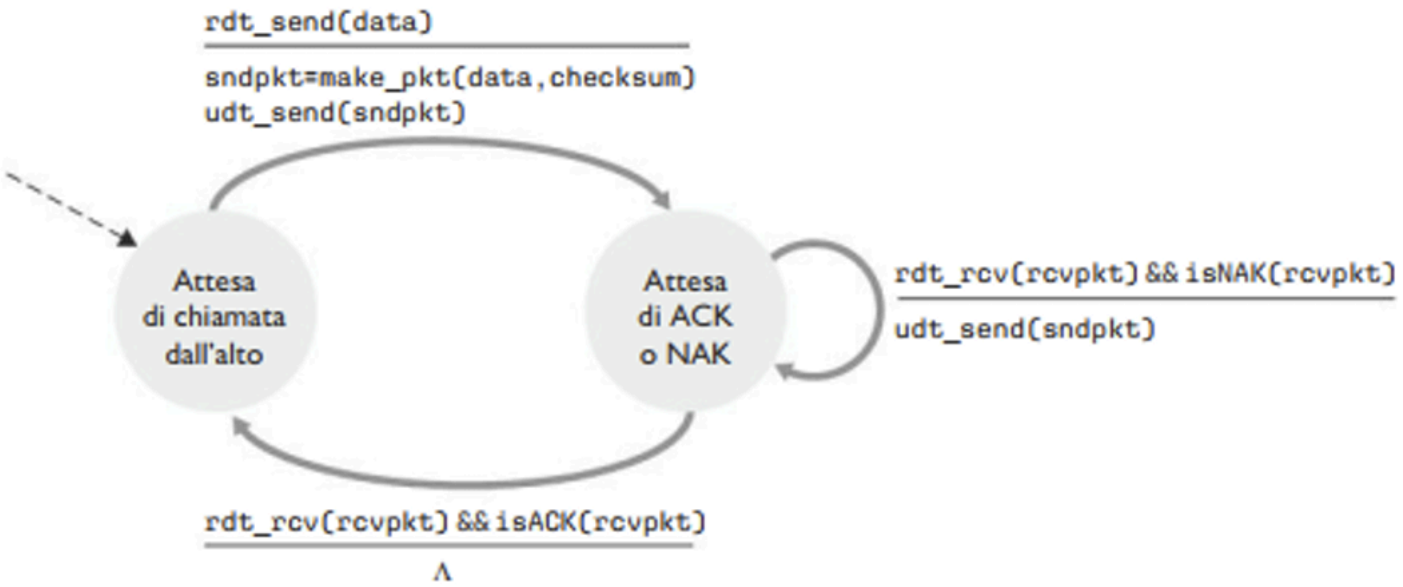
Nella versione 2.0 si assume che ci possano essere errori nella trasmissione dei messaggi, ma che non ci siano perdite di pacchetti. L'errore nella trasmissione dei messaggi a livello pratico è rappresentato da un errore sui bit. Per fare il recover degli errori utilizzano i segnali:

- **ACK**: il receiver comunica al sender che il pacchetto ricevuto è corretto;
- **NAK**: il receiver comunica al sender che il pacchetto ricevuto presenta errori. Il sender allora trasmette nuovamente il pacchetto.

Questo tipo di segnali permettono al receiver di notificare il mittente circa la ricezione dei messaggi, ricevendo nuovamente in caso il pacchetto. I protocolli RDT basati su ritrasmissioni sono noti come **protocolli ARQ** (**Automatic Repeat reQuest**). I protocolli ARQ necessitano di tre funzionalità aggiuntive:

- rilevamento dell'errore;
- feedback del destinatario;
- ritrasmissione.

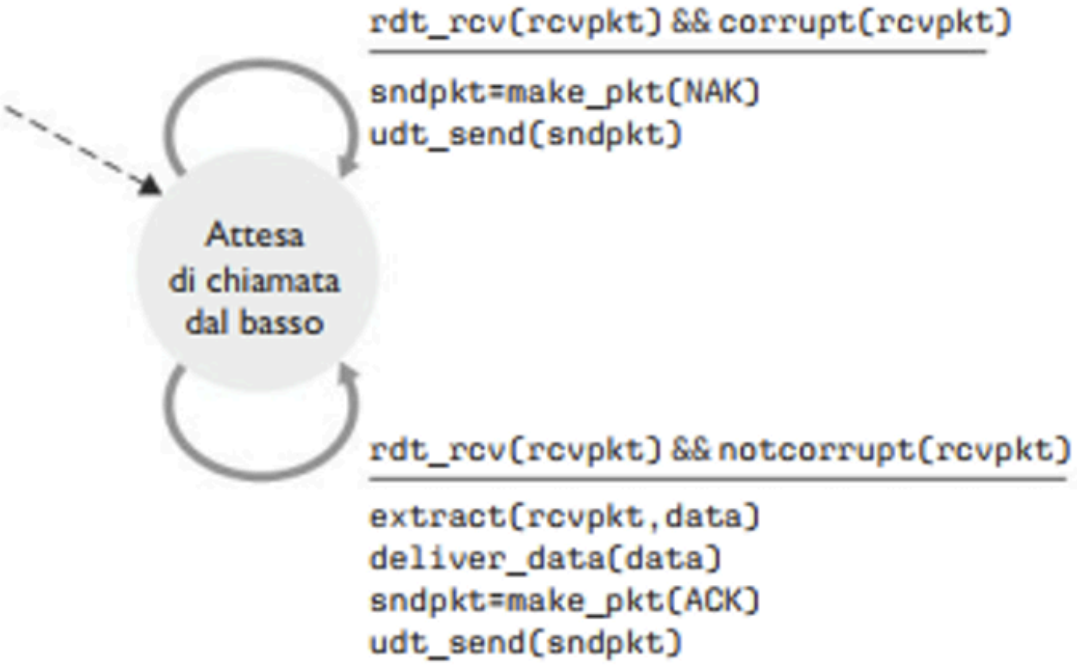
La FSM del sender è il seguente:



La macchina presenta due stati. In quello di sinistra, che è lo stato iniziale, la macchina attende i dati da raccogliere dal livello superiore. Nel momento in cui i dati devono essere inviati (`rtd_send(data)`), il mittente crea un pacchetto, includendo il checksum, e invia il pacchetto. Passa così allo stato di destra, in attesa di un segnale ACK e NAK. Se riceve un ACK, il sender sa che il pacchetto più recente è stato ricevuto correttamente, e passa così allo iniziale Λ. Se riceve un NAK, il protocollo ritrasmette l'ultimo pacchetto e attende il segnale ACK (o NAK).
Con tale protocollo si verificano le seguenti problematiche:

- si suppone che non ci sia una perdita di pacchetti;
- ci possono essere degli errori nella trasmissione del segnale NAK, che può diventare un segnale ACK per qualche errore: in tal caso il sender non inoltra nuovamente il pacchetto e assume che non ci siano stati problemi.

Nel caso del receiver:



Il receiver attende la ricezione dei pacchetti. Se il pacchetto ricevuto è corrotto, invia un segnale NAK e ritorna nel suo stato originale. Se il pacchetto ricevuto è corretto, i dati vengono estratti e viene inviato un segnale ACK.
In questo caso i problemi si presentano se il segnale ACK/NAK è corrotto. Infatti il sender non ha modo di sapere se il destinatario ha ricevuto correttamente l'ultimo blocco di dati trasmessi. Per risolvere questo problema il sender potrebbe inviare nuovamente il pacchetto perso, tuttavia introducendo un problema di **pacchetti duplicati**: il receiver, infatti, non sa se l'ultimo ACK o NAK sia stato ricevuto correttamente, pertanto non sa a priori se il pacchetto in arrivo è un duplicato o contiene dati nuovi.

Risoluzione della corruzione dei segnali ACK/NAK: RDT 2.1

Una soluzione al problema proposto consiste nell'aggiungere un campo al pacchetto dati, obbligando il sender a numerare i pacchetti con un **numero di sequenza**. Risulta sufficiente utilizzare un numero a 1 bit, dato che vogliamo solo capire se il

pacchetto ricevuto rappresenta una ritrasmissione o meno.
Le FSM per sender e receiver sono le seguenti:

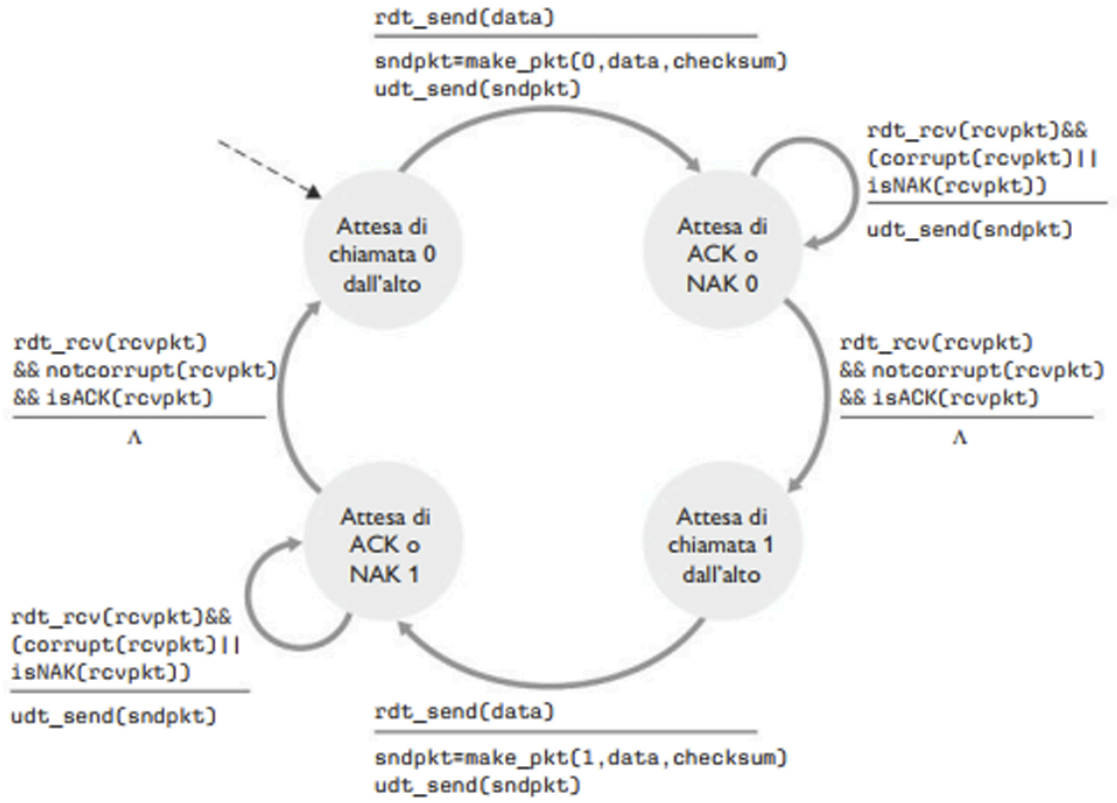


Figura 3.11 Mittente rdt2.1.

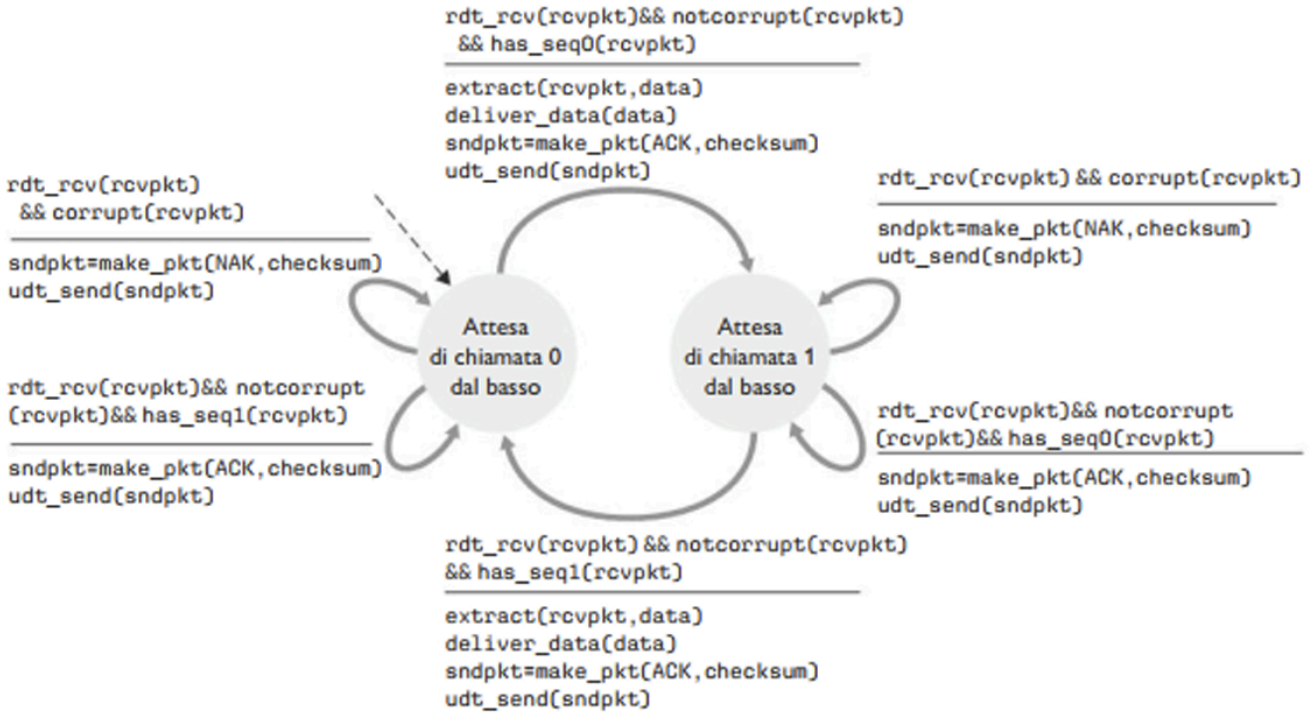


Figura 3.12 Ricevente rdt2.1.

Le FSM presentano quindi il doppio degli stati, in quanto il protocollo deve riflettere il fatto che il pacchetto inviato o ricevuto abbia numero di sequenza 0 o 1. In particolare, una volta che il sender crea il pacchetto con numero di sequenza 0, attende un segnale ACK o NAK con numero di sequenza 0. Se riceve un NAK o un segnale corrotto, allora rimane in attesa, altrimenti alterna stato: il prossimo messaggio avrà numero di sequenza 1.
Il receiver invece presenta due stati: attesa per una chiamata 0 e attesa per una chiamata 1. La FSM del receiver commuta stato solo nel caso in cui il pacchetto ricevuto non è corrotto e ha numero di sequenza corrispondente al numero di sequenza atteso. Altrimenti, non cambia stato e:

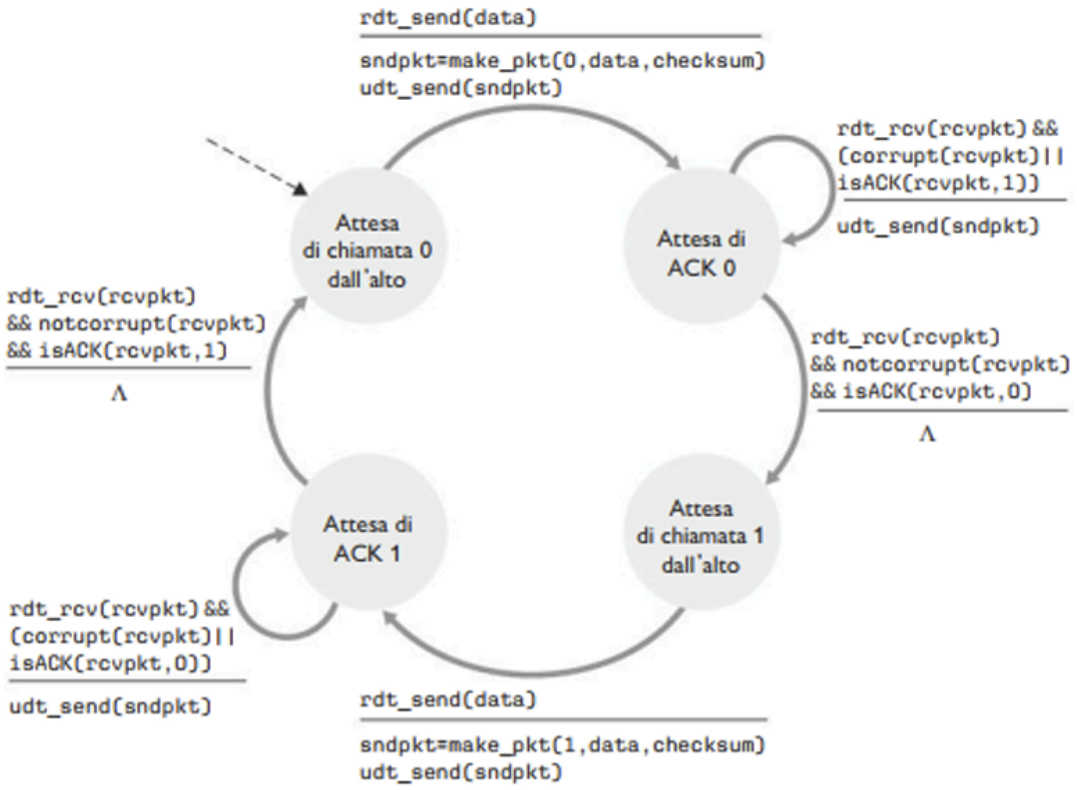
- manda un NAK se il pacchetto è corrotto;
- manda un ACK se il pacchetto è fuori sequenza. In questo caso si invia un ACK per il fatto che il sender ha ritrasmesso un pacchetto per il fatto che l'ACK non è stato ricevuto correttamente. Il pacchetto viene scartato e si invia l'ACK.

Con l'aggiunta della sequenza al pacchetto abbiamo un controllo aggiuntivo oltre al checksum. Dal punto di vista del receiver, a parità di correttezza dei dati il receiver può controllare se ci sono dei duplicati in base allo stato 0 o 1. Da notare che il receiver non può sapere se l'ultimo ACK/NAK è stato ricevuto correttamente dal sender.

Rimozione del segnale NAK: RDT 2.2

Il problema che ci si è posti riguarda la necessità di avere un segnale NAK. Si potrebbe in effetti utilizzare sempre un segnale ACK. Creare una differenza tra ACK e NAK rende più facile creare errori: quello che li differenzia spesso è un singolo bit. Se questo flag presenta un errore, passare da ACK a NAK è molto più semplice. Possiamo allora evitare di utilizzare il NAK e semplicemente inviare un ACK per l'**ultimo pacchetto ricevuto correttamente**. Il receiver quando tutto va bene manda l'ACK; se qualcosa non va bene, il receiver manda l'ACK con l'ultimo pacchetto ricevuto. Quando il sender riceve un ACK duplicato, il sender ritrasmette l'ultimo pacchetto, **che non è quello dell'ACK del receiver**.

Figura 3.13 Mittente rdt2.2.



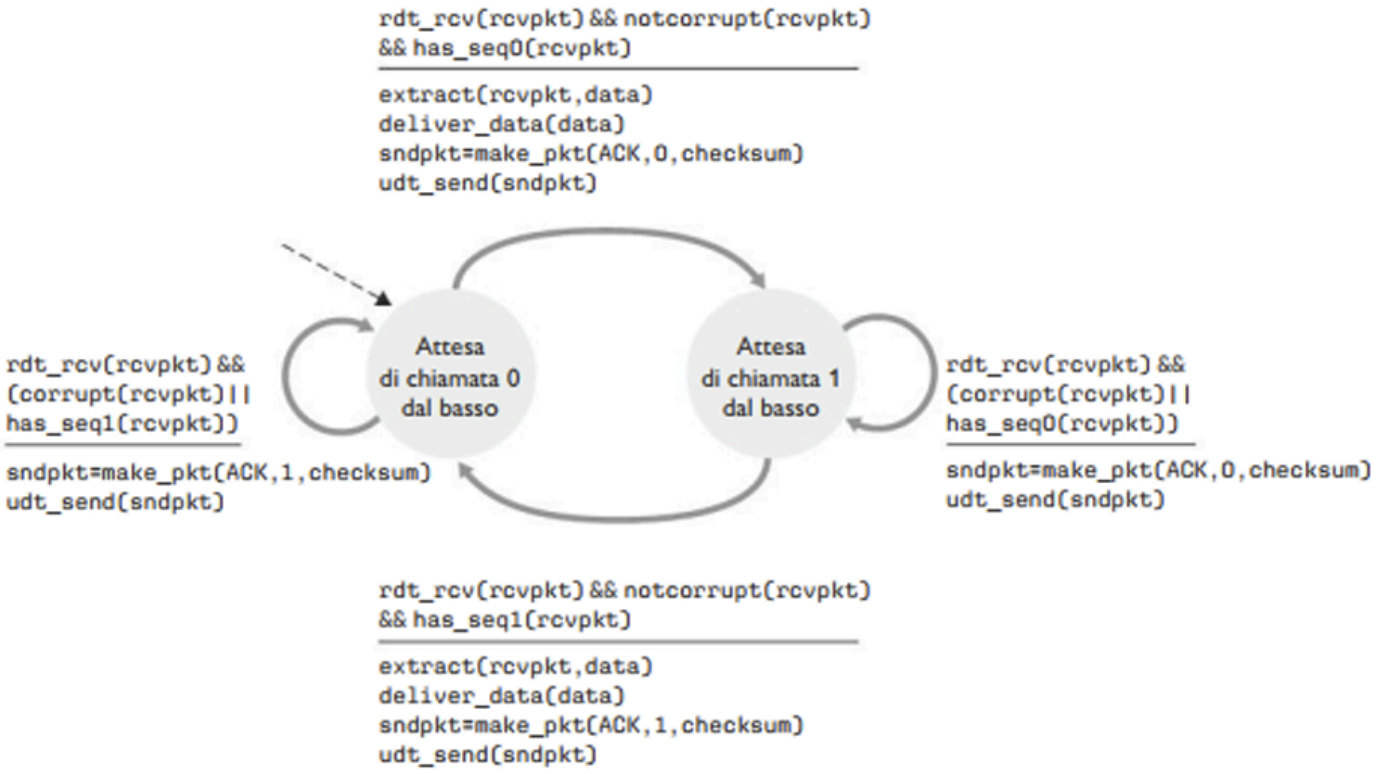
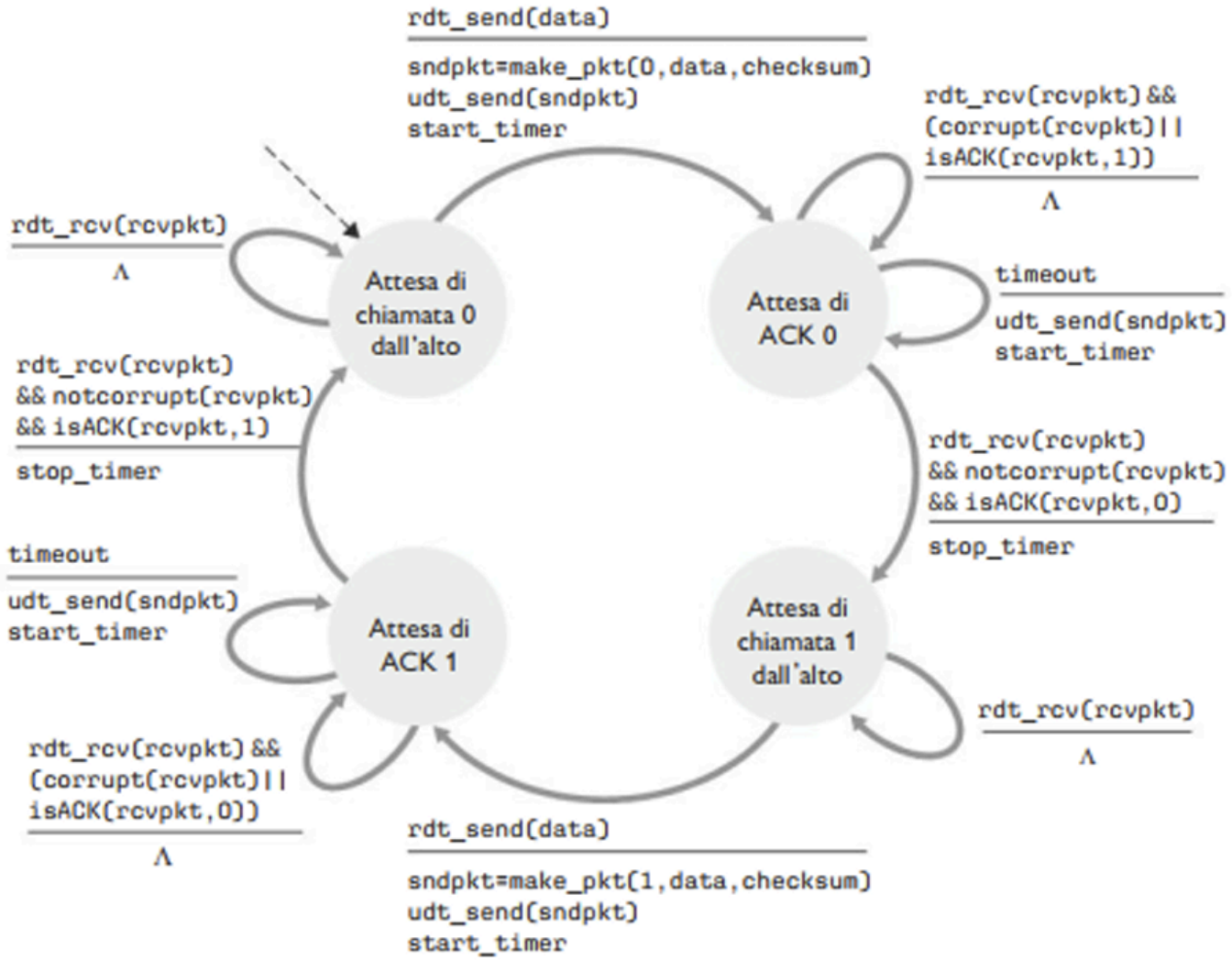


Figura 3.14 Ricevente rdt2.2.

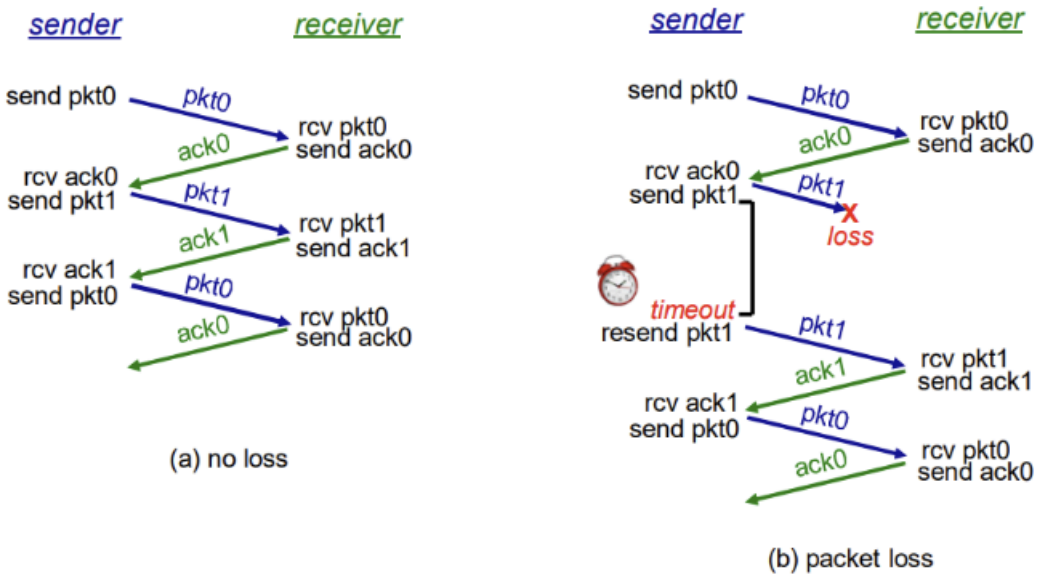
Gestione della perdita di pacchetti: RDT 3.0

Il vantaggio di RDT 2.2 è che tramite l'utilizzo di un toggle e con la sequenza di pacchetti si possono gestire molte problematiche. Un problema tuttavia è che **non viene gestita la perdita di pacchetti**. Non è detto infatti che i pacchetti arrivino corrotti: possono anche perdersi. I mezzi di propagazione infatti possono risentire del modo in cui questi vengono collocati, possono soffrire di cali di tensione, danni per usura (quindi ulteriori resistenze). Il segnale quindi viene dissipato nella trasmissione.
Nella versione 3.0 di RDT si assume che il canale sottostante possa perdere pacchetti. Si fa partire un timer dopo che il sender invia il pacchetto e si verifica se in questo lasso di tempo si riceve un ACK. Se il segnale ACK non è stato ricevuto in tempo, il pacchetto viene inviato nuovamente. Dal lato sender si ha la seguente FSM:

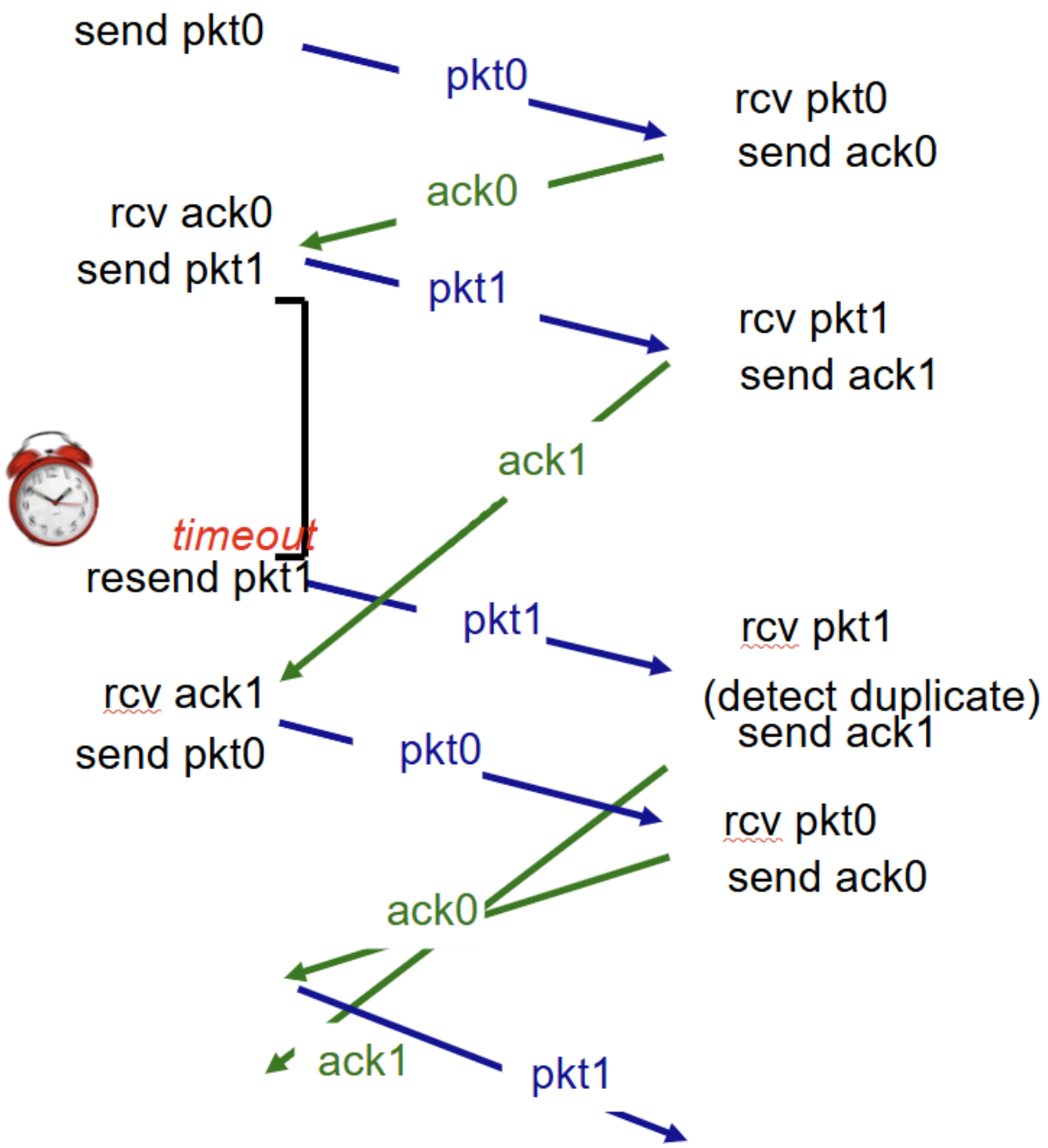


Se nello stato 0 il sender riceve un ACK 1 oppure il timer del sender va in timeout, il pacchetto viene inviato nuovamente.

L'RDT 3.0 a livello teorico opera come segue:



Nella realtà, tuttavia, si può presentare il seguente fallimento:



Il fallimento si verifica perché il segnale ACK1 non arriva in tempo. Il pacchetto 1 viene quindi inviato nuovamente, e il receiver individua un duplicato e invia un ACK1. Tuttavia poiché il primo segnale ACK1 è stato ricevuto in ritardo, il sender interpreta l'ACK1 vecchio come l'ACK dell'ultimo pacchetto. In pratica quindi **il sender interpreta gli ACK1 ricevuti come ACK1 dei pk1 appena inviati**.
La situazione può stabilizzarsi solamente sincronizzando nuovamente le connessioni.

05 - Throughput

> *Il **throughput** è una misura della capacità di trasmissione del mezzo.*

Cominciamo introducendo un esempio.
Sia data una **bandwidth** (**larghezza di banda**) di 10 Megabit per secondo (Mbps). La larghezza di banda esprime il numero di bit che il canale riesce a trasmettere per unità di tempo. In un secondo allora trasmettiamo 10^7 bit, pertanto un bit viene trasmesso in $0.1\ \mu s$.
Consideriamo ora la trasmissione di un frame di 1500 byte. Portiamo l'informazione a bit, e sapendo che un byte è 8 bit, si ha che 1500 byte corrispondono a 12000 bit. Per trasmettere un frame ci metteremo $12000 \cdot 0.1 = 1200\ \mu s$.
Chiaramente dobbiamo tenere in considerazione una distanza tra i due host. In particolare, supponiamo che lo **spazio di trasmissione** sia pari a $1\ \text{km}$. Dobbiamo calcolare anche la **velocità di propagazione** del segnale, fortemente dipendente dal mezzo che utilizziamo e dalla sua usura. In questo caso, supporremo che $v \simeq 200.000\ \text{km/s}$. Supponiamo tale velocità sia costante. Avremo allora che il segnale si propaga in un tempo pari a $5\ \mu s$.
Ora che abbiamo queste informazioni, possiamo dedurre quanti sono i bit che possiamo contemporaneamente far viaggiare sul canale. Basta che moltiplichiamo la larghezza di banda per il tempo di propagazione: si ottiene quindi $10^7 \cdot 5 \cdot 10^{-6} = 50$ bit inviati simultaneamente nel canale. Abbiamo tutte le informazioni per calcolare il **tempo di trasferimento**, ossia il tempo che un frame necessita per essere trasferito (senza considerare altre variabili quali tempi di connessione, ACK, ecc.): basta moltiplicare la dimensione del frame per il tempo necessario per trasferire un bit e sommare il tempo necessario per la propagazione del segnale. Si ottiene quindi $1200 + 5 = 1205\ \mu s$.
I calcoli che abbiamo fatto sono puramente teorici, perché possono esserci anche ritardi e congestioni. Quello che abbiamo calcolato è stato un **throughput teorico**.
Nella realtà una connessione stabile richiede la creazione della connessione, che richiede un RTT. Bisogna infatti considerare un **tempo di elaborazione**, dovuto al tempo per il receiver (solitamente un server) di inviare correttamente il segnale ACK al processo client corretto. Il segnale di ACK solitamente è un segnale a 64 byte. Questo richiede un tempo pari a $51.2\ \mu s$. Aggiungendo un tempo stimato di elaborazione pari a $3\ \mu s$ e un RTT pari a $10\ \mu s$, aggiungiamo alla quantità $1200\ \mu s$ il tempo di elaborazione più il tempo RTT più il tempo necessario per inviare il segnale ACK, quindi $1200 + 3 + 5 \cdot 2 + 51.2 = 1264.2\ \mu s$. Notare che l'RTT pari a 10 microsecondi fa riferimento al tempo per stabilire la connessione, sia all'andata sia al ritorno, quindi è dato dalla somma del tempo per stabilire la connessione in andata e dal tempo per stabilire la connessione in ritorno.
In virtù dei calcoli effettuati, possiamo ottenere il **throughput effettivo** prendendo il numero di bit da trasferire e dividendolo per il tempo impiegato. Si ottiene così:

$$\frac{12000}{1264.2 \cdot 10^{-6}} = 9.492\ \text{Mbps}$$

Quindi siamo vicini al throughput teorico.
Siamo partiti da un RTT pari a $5\ \mu s$. Parlare di microsecondi tuttavia in un trasferimento dove la distanza è molto alta e dove i mezzi impiegati per la trasmissione dei dati non sono molto avanzati non è opprtuno; sarebbe opportuno parlare anche di millisecondi. Immaginando un RTT ora pari a $20\ ms$, si ottiene un tempo pari a

$$1200 + 20000 + 3 + 51.2 = 21254.2\ \mu s$$

Il trasferimento richiede quindi $10000 + 1200 + 3$ microsecondi necessari per trasmettere il frame in andata, mentre per trasmettere l'ACK abbiamo bisogno di $10000 + 51.2$ microsecondi. La nostra banda adesso risulta fortemente penalizzata. Infatti il throughput effettivo è:

$$\frac{12000}{21524.2} = 0.575\ \text{Mbps}$$

che è decisamente inferiore rispetto al throughput teorico. Basta quindi inserire delle latenze e la velocità di trasmissione crolla.
La formula generale da seguire allora è la seguente:

$$T = T_{\text{frame}} + RTT + T_{\text{elaborazione}} + T_{\text{ACK}}$$

Verificare i tempi: comando ping

Una utility per misurare l'RTT necessario per connettersi al server è `ping`. `ping` permette di inviare pacchetti tramite il protocollo ICMP, permettendo di calcolare la percentuale di pacchetti persi e l'RTT.

Esercizio

Dobbiamo trasferire utilizzando un protocollo TCP tra il nodo A e il nodo B con un tempo di propagazione pari a $7.5\ ms$ un frame di 2000 byte con una bandwidth pari a $15\ \text{Mbps}$. Il tempo di elaborazione è pari a $10\ \mu s$. Lo spazio di trasmissione tra A e B è pari a

$1500\ m$, e la velocità di propagazione del segnale è pari a $150000\ \mathrm{km/s}$. Vogliamo calcolare il tempo di trasmissione del frame.
Il frame è di 2000 byte, pari a 16000 bit. La bandwidth è pari a $15\ \mathrm{Mbps}$, il che significa che un bit viene trasmesso in un tempo pari a $T_{bit} = \frac{1}{15000000} = 0.067\ \mu s$. Il nostro frame allora viene trasferito in un tempo pari a $1072\ \mu s$. Il tempo di propagazione del segnale è dato da $\frac{1,5}{150000} = 10\ \mu s$. In più dobbiamo calcolare il tempo per trasferire il segnale ACK, un messaggio di 512 bit, tempo pari a $512 \cdot 0,067 = 34.3\ \mu s$. L'RTT è pari a $7500\ \mu s$. Di conseguenza il tempo necessario è pari a:

$$T_{\text{trasferimento}} = 1072 + 15000 + 10 + 34.3 = 16116.3\ \mu s$$

L'effettivo throughput di conseguenza è dato da:

$$\frac{16000}{16116.3} = 0.997\ \mathrm{Mbps}$$

Round Trip Time e tempo di propagazione del segnale

Il tempo di propagazione è il tempo con cui un segnale si trasmette da un punto ad un altro. La connessione in realtà non è detto che avviene in prima istanza. Inoltre il TCP presenta il meccanismo di Three Way Handshake, con SYN, SYN ACK e ACK. Questi meccanismi introducono ritardi, ed è probabile che l'RTT è anche **maggiore** al tempo di propagazione del segnale nel mezzo in andata e in ritorno.
Assumeremo nella maggioranza dei casi che il tempo di propagazione (in avanti e indietro) coincida con l'RTT. Se questi valori sono diversi, invece, **si considera il maggiore dai due nel calcolo**.

Precisazioni sui risultati

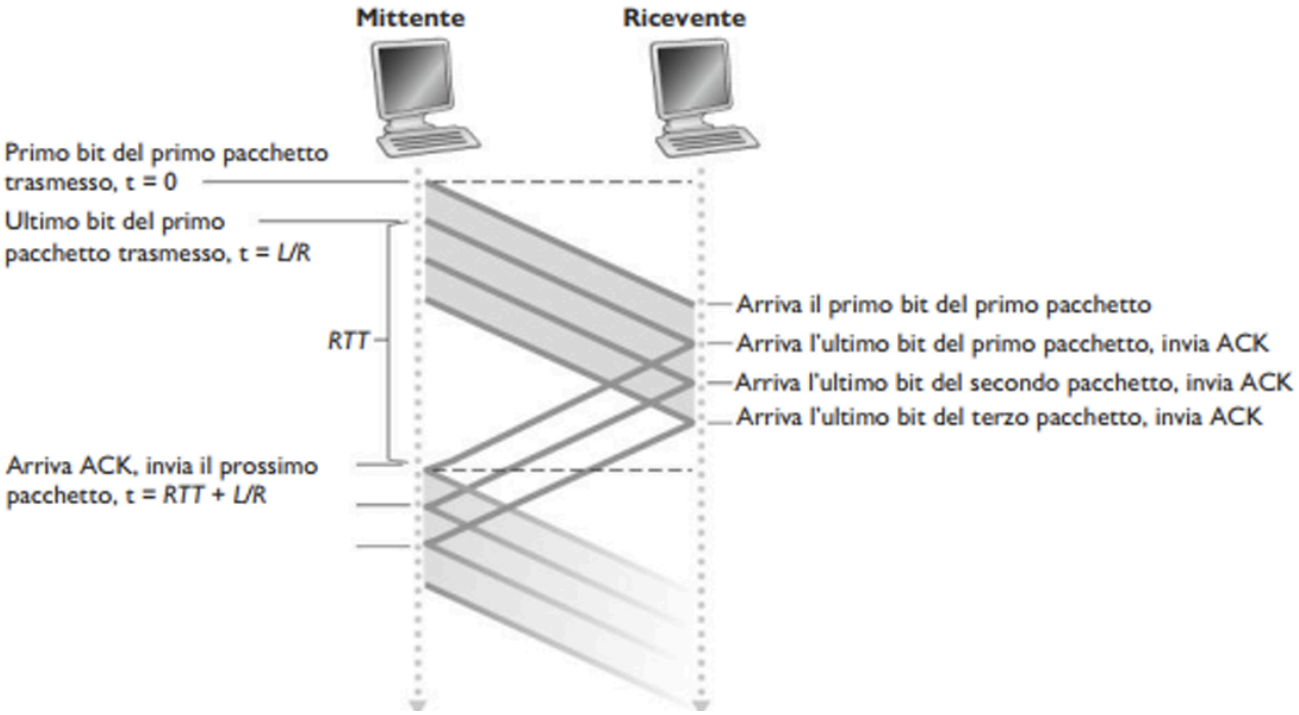
Se il bandwidth effettivo è superiore rispetto al bandwidth teorico, il bandwidth teorico funge da **collo di bottiglia** e il bandwidth effettivo è quello teorico fornito.

Modalità pipeline

C'è un'altra modalità che permette il miglioramento del throughput, che è la modalità **pipeline**. Invece di inviare un singolo frame, si definisce una **dimensione di finestra**. Si manderanno tanti frame pari alla dimensione di finestra. I frame possono anche avere dimensione diversa, ma il delay tra un frame e un altro è dato dal tempo per spedire il frame successivo a quello inviato più recentemente.
L'utilizzo della modalità pipeline è chiaramente più vantaggioso della modalità esposta precedentemente, nota come modalità **stop and wait**. Supponiamo infatti di trasferire pacchetti di 8000 bit (1000 byte) con una bandwidth di 1 Gigabit per secondo.
Consideriamo un RTT di $30\ ms$ complessivo e ignoriamo eventuali tempi di elaborazione (e per semplicità, anche il pacchetto ACK: il motivo di tale esempio è mostrare l'assurdità di riduzione di performance). Il tempo teoricamente richiesto per mandare il pacchetto è pari a 8 microsecondi. Considerando l'RTT impiegheremo 30 millisecondi. Il throughput effettivo è dato da:

$$\frac{8000}{30.008} = 267\ \mathrm{kbps}$$

che praticamente significa che di tutta la banda a 1 Gigabit per secondo, riusciamo solamente a usare 267 kilobit per secondo. Non è chiaramente la situazione ideale. Il problema risiede che inviamo un frame, aspettiamo esso venga ricevuto, e poi ne inviamo un altro (appunto, stop and wait).
Considerando l'esempio del paragrafo precedente (quello con RTT = $20ms$.) Consideriamo una window size pari a 10. Il primo frame viene inviato al tempo $t = 21254.2$. Supponiamo ogni frame sia della stessa dimensione, ossia 12000 bit. Poiché ogni frame impiega 1200 microsecondi per essere inviato, ogni frame viene inviato in sequenza, con un **delay** pari al tempo necessario per trasmettere il frame precedente. Si ha che il tempo finale è dato da $21254.2 + 9 \cdot 1200 = 32054.2$.
Il vantaggio risiede nel fatto che **non è necessario inviare un ACK per ogni frame inviato nel window**. Nel momento in cui riceve il primo ACK, il sender può inviare altri frame in una nuova window.



Come si nota nella foto soprastante, dopo che il receiver invia l'ACK per il primo frame inviato, il sender può già inviare un secondo pacchetto.

06 - Protocolli per la pipeline, TCP

Protocolli per la pipeline

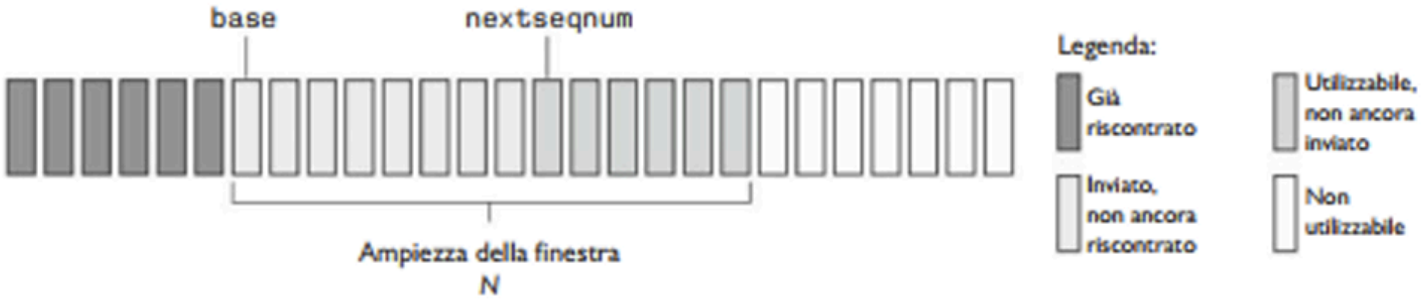
Abbiamo visto come il throughput con modalità **stop and wait** è fortemente influenzato dall'RTT. Possiamo migliorare il throughput:

- riducendo l'RTT;
- **utilizzando la pipeline**.

Nella gestione della pipeline è necessario **controllare il flusso di connessione**. Più precisamente, bisogna assicurarsi che gli ACK vengano gestiti correttamente, verificando che i pacchetti non siano inviati in un ordine sbagliato.
Risulta necessario in particolare aumentare i **numeri di sequenza**. Nel meccanismo RDT 2.1 e successivi infatti si utilizza un bit per il numero di sequenza, ma anche una modalità stop and wait. Poiché mandiamo più pacchetti insieme, è necessario che ogni pacchetto in transito abbia un numero di sequenza univoco. La quantità di numeri di sequenza necessari dipende dal modo in cui il protocollo di trasferimento dei dati reagisce ai pacchetti smarriti, alterati o troppo in ritardo. Pertanto si identificano due approcci di base per risolvere gli errori in pipeline: **Go-Back-N** e **ripetizione selettiva**.

Go-Back-N

In un protocollo **Go-Back-N** (**GBN**) il mittente può trasmettere più pacchetti senza dover attendere ACK, ma non può avere più di N pacchetti in attesa di ACK. La visione del sender in tale protocollo è descritta dalla foto sottostante:



Indichiamo con `base` il **numero di sequenza del pacchetto più vecchio che non ha ricevuto ACK**. Indichiamo con `nextseqnum` **il più piccolo numero di sequenza inutilizzato** (ossia **il numero di sequenza del prossimo pacchetto da inviare**). Possiamo allora individuare quattro gruppi di numeri di sequenza:

- un gruppo di numeri di sequenza più piccoli di `base`, che identificano i pacchetti che sono stati già inviati e hanno ricevuto un ACK;
- un gruppo di numeri di sequenza compreso tra `base` e `nextseqnum - 1`, che identifica i pacchetti che sono stati inviati ma non hanno ricevuto ACK;
- un gruppo di numeri di sequenza compreso tra `nextseqnum` e `base + N - 1`, che identifica i pacchetti che non sono ancora stati inviati ma sono pronti all'invio se arrivassero dal livello di applicazione;
- un gruppo di numeri di sequenza maggiori o uguali a `base + N`, che identifica i pacchetti che non possono essere inviati.

L'intervallo di numeri di sequenza ammissibili per i pacchetti trasmessi ma che non hanno ricevuto ACK, includendo i pacchetti utilizzabili, prende il nome di **finestra**. Il valore N prende il nome di **window size** e il protocollo GBN prende il nome di **protocollo a finestra scorrevole** (infatti si dice che è un algoritmo a **sliding-window**). Il motivo per cui si limita il numero di pacchetti a N risiede nei meccanismi di controllo di congestione.

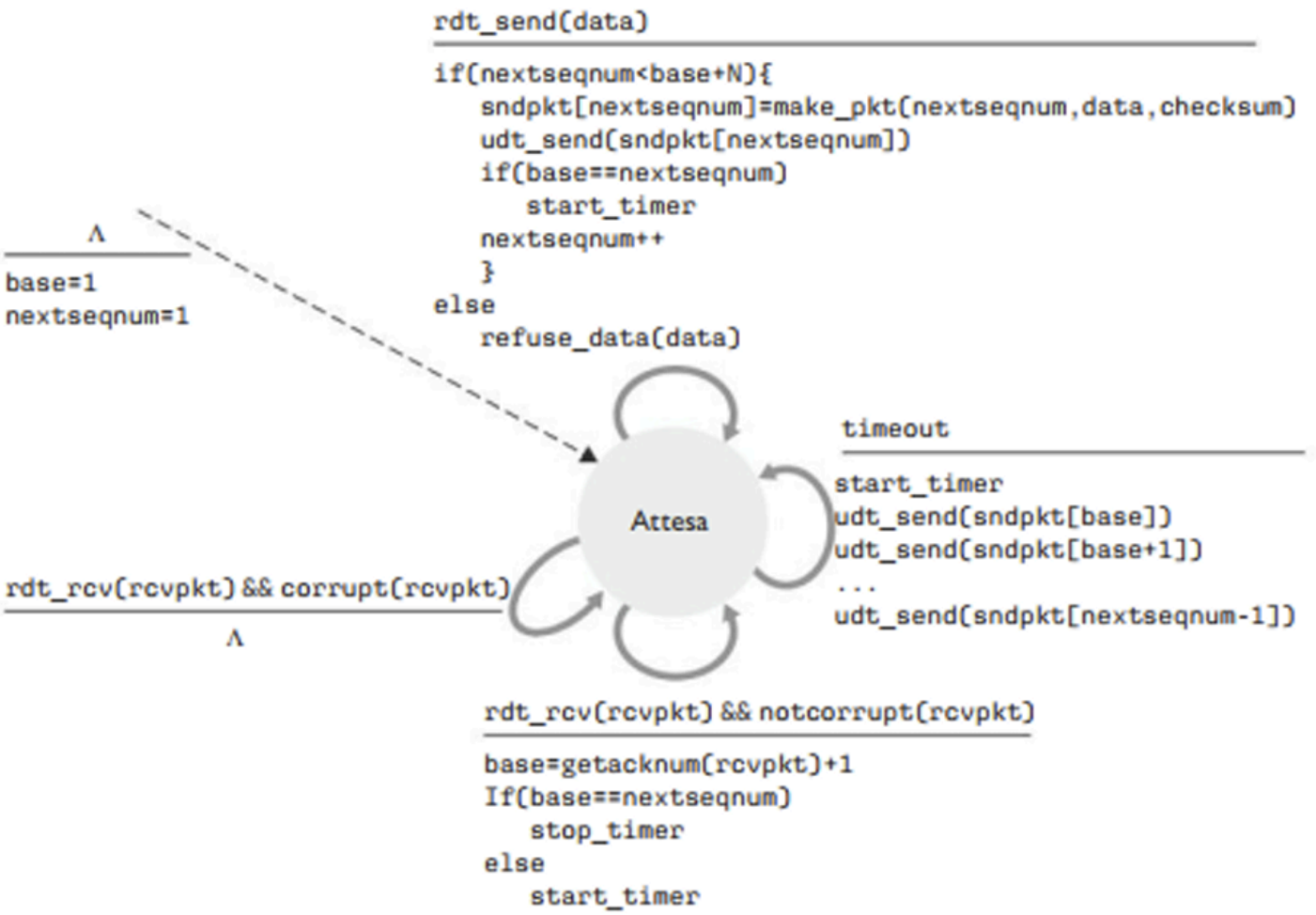
Mittente GBN

Analizziamo il mittente GBN:

- ad un'invocazione dall'alto di invio di un pacchetto il mittente controlla se vi siano pacchetti N in sospeso senza ACK, ossia se la finestra è piena. Se non è piena, crea e invia un pacchetto, aumentando `nextseqnum`. Se la finestra è piena, memorizza il pacchetto in un buffer e ritenterà più tardi l'invio;
- l'ACK di un pacchetto con numero di sequenza n viene considerato come **ACK cumulativo**, ossia **indica che tutti i pacchetti con numero di sequenza minore o uguale a n sono stati correttamente ricevuti**;
- al timeout il mittente invia **tutti** i pacchetti spediti che ancora non hanno ricevuto un ACK. Per il timeout si può implementare un timeout per il pacchetto trasmesso meno di recente. Se si riceve un ACK ma ci sono ancora pacchetti trasmessi e non

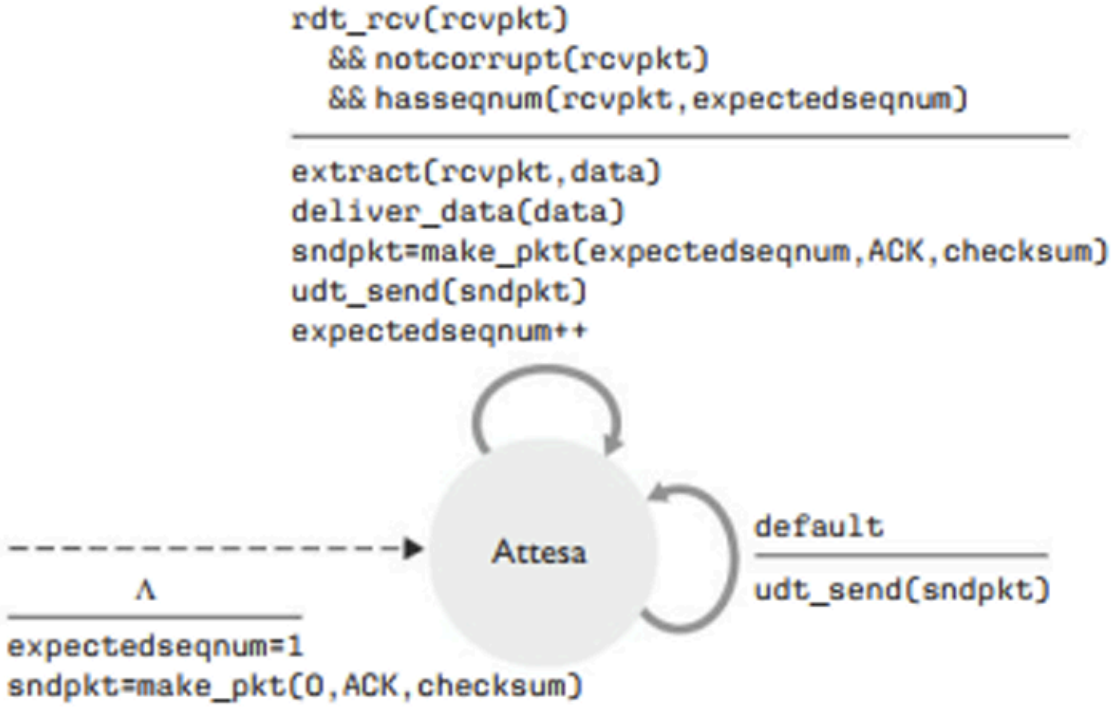
riscontrati, il timer viene fatto ripartire. Altrimenti il timer viene fermato.

Possiamo utilizzare una FSM estesa per rappresentare il sender:



Destinatario GBN

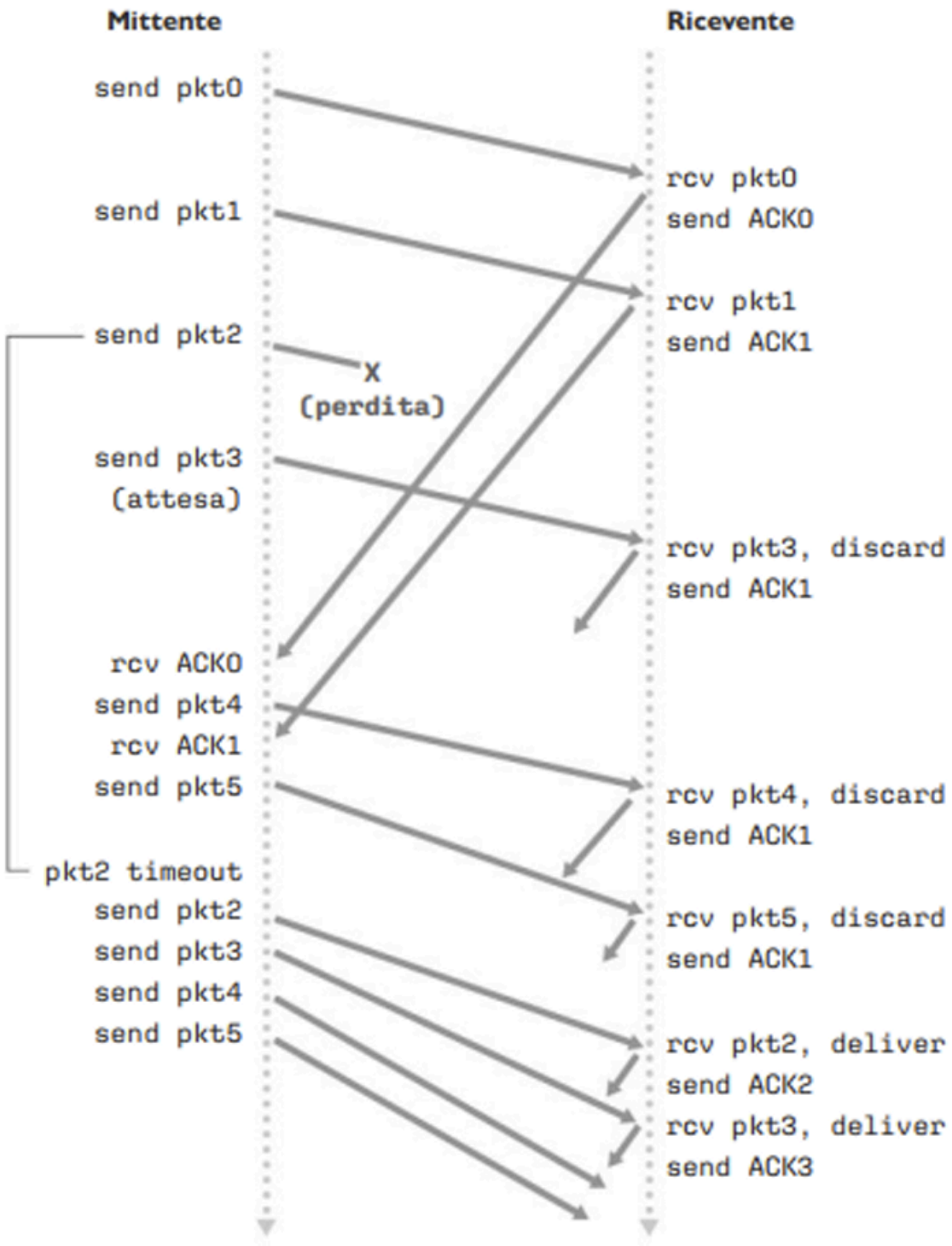
Il receiver GBN è più semplice:



Il receiver in questo caso tiene conto solo dell'ultimo pacchetto ricevuto correttamente e per cui è stato inviato un ACK. Il numero di sequenza di tale pacchetto è indicato con `rcv_base` (in foto invece è indicato con `expectedseqnum`). Se un pacchetto non viene ricevuto correttamente (ad esempio, **non è in ordine**) allora invia un ACK dell'ultimo pacchetto ricevuto correttamente. Da qui l'idea

dell'uso di ACK cumulativi: poiché i pacchetti vengono consegnati al livello superiore uno per volta, se l'n-esimo pacchetto è stato ricevuto e consegnato, tutti i pacchetti con numero di sequenza inferiore sono stati anch'essi consegnati. I pacchetti fuori sequenza, inoltre, **vengono scartati**.

Operatività GBN

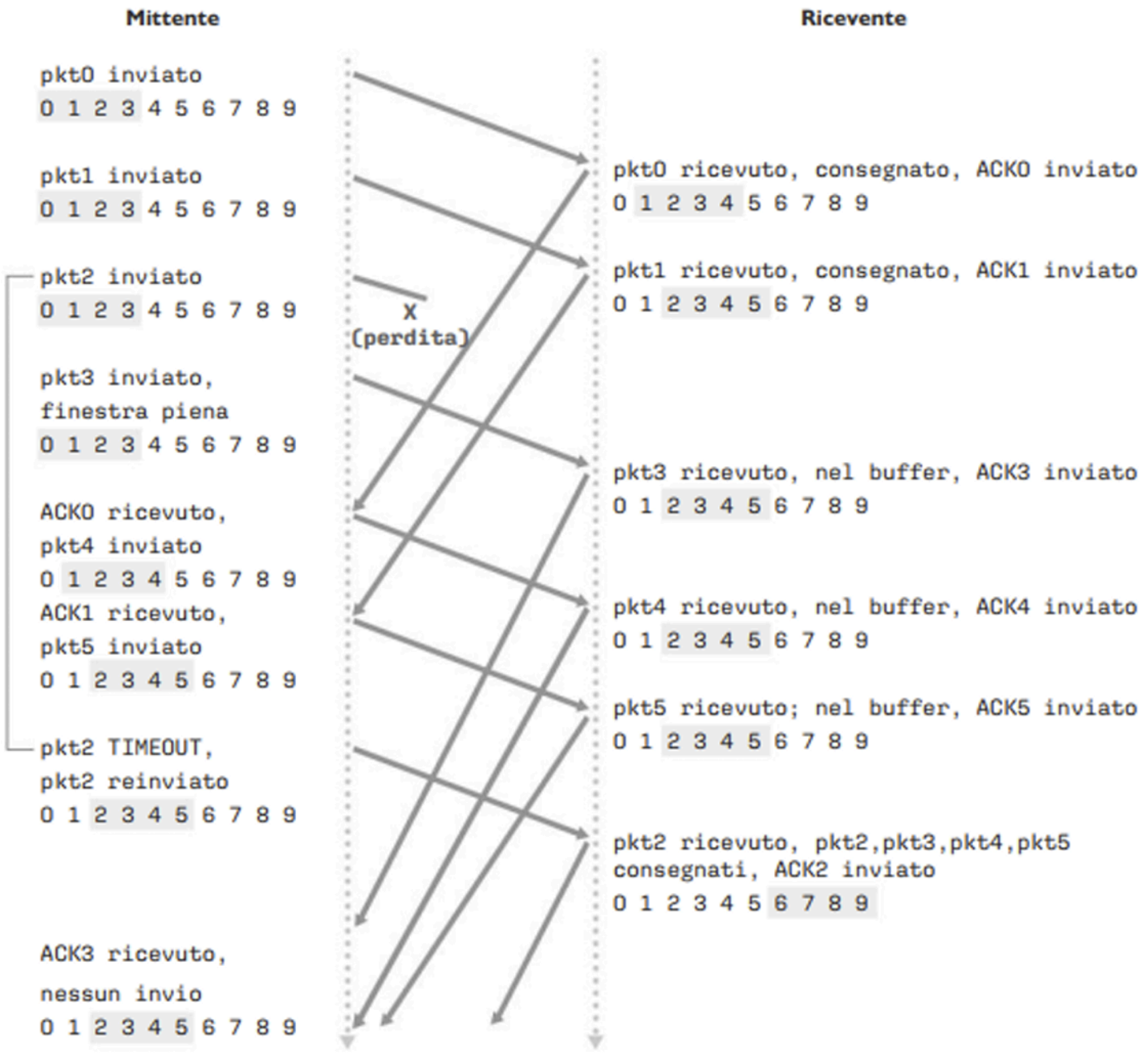


L'immagine soprastante mostra l'operatività del protocollo GBN con $N = 4$. A causa della window size, il sender può inviare solo i pacchetti con numero di sequenza da 0 a 3. Quando il receiver invia l'ACK del pacchetto con numero di sequenza 0, la finestra si sposta di un posto, aumentando `base` di 1 e inviando quindi il pacchetto con numero di sequenza pari a 4; lo stesso viene fatto quando viene inviato l'ACK di 1. Supponiamo il pacchetto 2 venga perso. Il receiver scarta automaticamente 3, 4 e 5 in quanto il numero di sequenza non è pari a 2 in nessuno dei casi. Al timeout, il sender invia nuovamente i pacchetti $2, 3, 4, 5$ e l'operazione continua.
Da notare che quando il receiver invia l'ACK con numero di sequenza pari a 1, il sender **ignora gli ACK duplicati**.

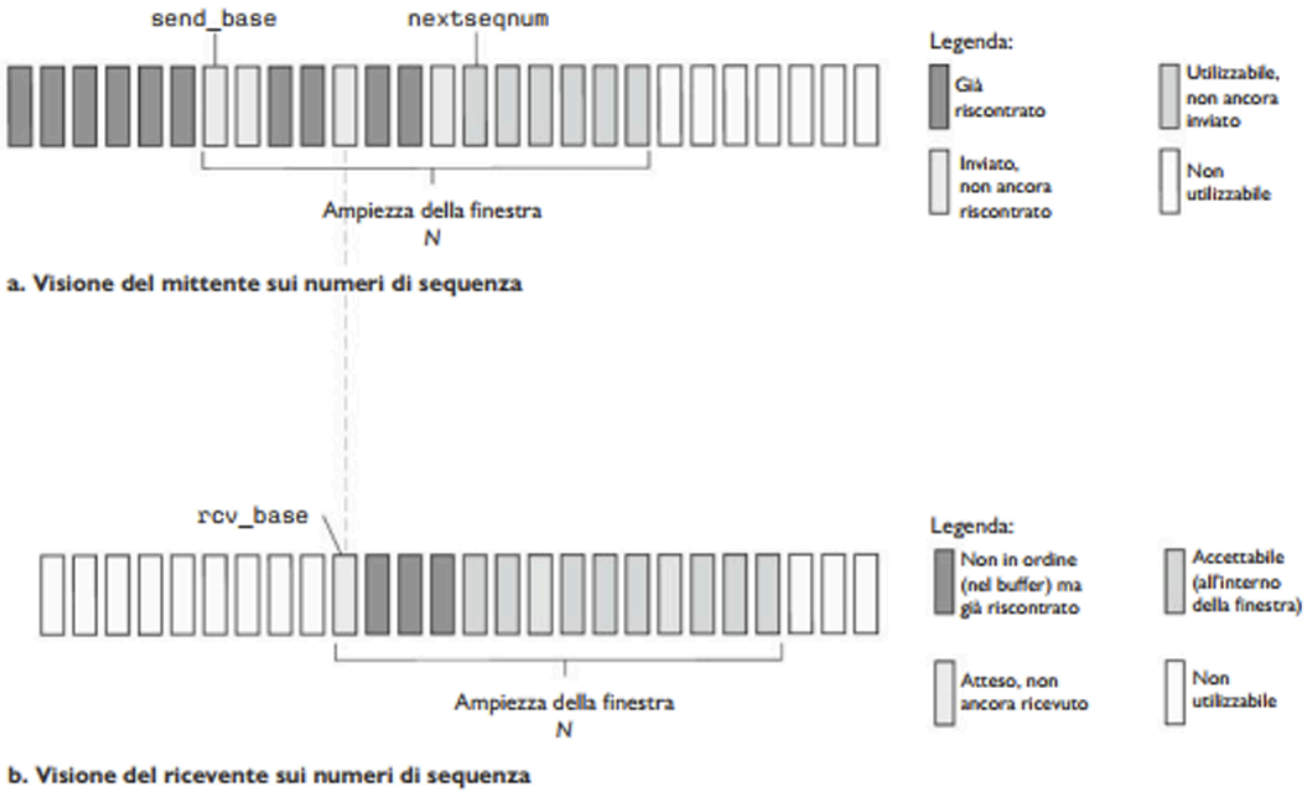
Selective repeat

Un problema del protocollo GBN risiede nel fatto che se nella pipeline si trovano molti pacchetti, anche un errore su un solo pacchetto può causare numerose ritrasmissioni, molto spesso inutili. La pipeline quindi si **satura** a causa di queste ritrasmissioni. Un'alternativa alla GBN è il **protocollo a ripetizione selettiva** (**SRP**, **Selective-Repeat protocol**), che evita le ritrasmissioni

facendo ritrasmettere al mittente solo i pacchetti persi o alterati. Si stabilisce quindi una window size N per limitare i pacchetti privi di ACK, ma a differenza della GBN, il sender avrà già ricevuto gli ACK di qualche pacchetto in finestra. Infatti il receiver SR invia un ACK per i pacchetti correttamente ricevuti, **anche se fuori sequenza**: tali pacchetti infatti vengono memorizzati in un buffer (tipicamente usato anche in sistemi che sfruttano UDP). Il buffer chiaramente è **limitato**, quindi se il buffer è troppo piccolo alcuni pacchetti si perderanno.



Risulta importante notare che il receiver spedisce un ACK dei pacchetti già ricevuti con numeri di sequenza al di sotto dell'attuale base della window (ossia più piccoli di `rcv_base`). Questa notifica è necessaria, in quanto, ad esempio, se non si propaga un ACK per il pacchetto con numero di sequenza `send_base` il sender potrebbe ritrasmetterlo anche se il receiver lo ha già ricevuto. Questo perché **gli ACK non sono cumulativi e i pacchetti fuori sequenza vengono memorizzati e riconosciuti**. Se non viene ritrasmesso l'ACK, la finestra del sender non può avanzare.
Nel caso di packet loss dell'ACK del segmento con numero di sequenza `send_base` il receiver sposta la sua finestra ma il sender no. Ciò significa che **le finestre del mittente e del destinatario non sempre coincidono**.



a. Visione del mittente sui numeri di sequenza

b. Visione del ricevente sui numeri di sequenza

Mittente SR

Analizziamo il sender SR:

- quando riceve dati dai layer soprastanti il mittente SR opera come il mittente GBN: verifica se la finestra è piena, creando un pacchetto con numero di sequenza `nextseqnum` se non è piena, altrimenti memorizza nel buffer il pacchetto da inviare e ritarda l'invio;
- il meccanismo di timeout è simile al GBN, ma **è necessario utilizzare un timer logico per pacchetto** per il fatto che gli ACK non sono cumulativi;
- alla ricezione di un ACK, il sender etichetta il pacchetto come ricevuto nel caso in cui sia in finestra. Se il pacchetto ha numero di sequenza `send_base`, allora sposta la finestra verso il numero di sequenza più piccolo del pacchetto che non ha ricevuto ACK.

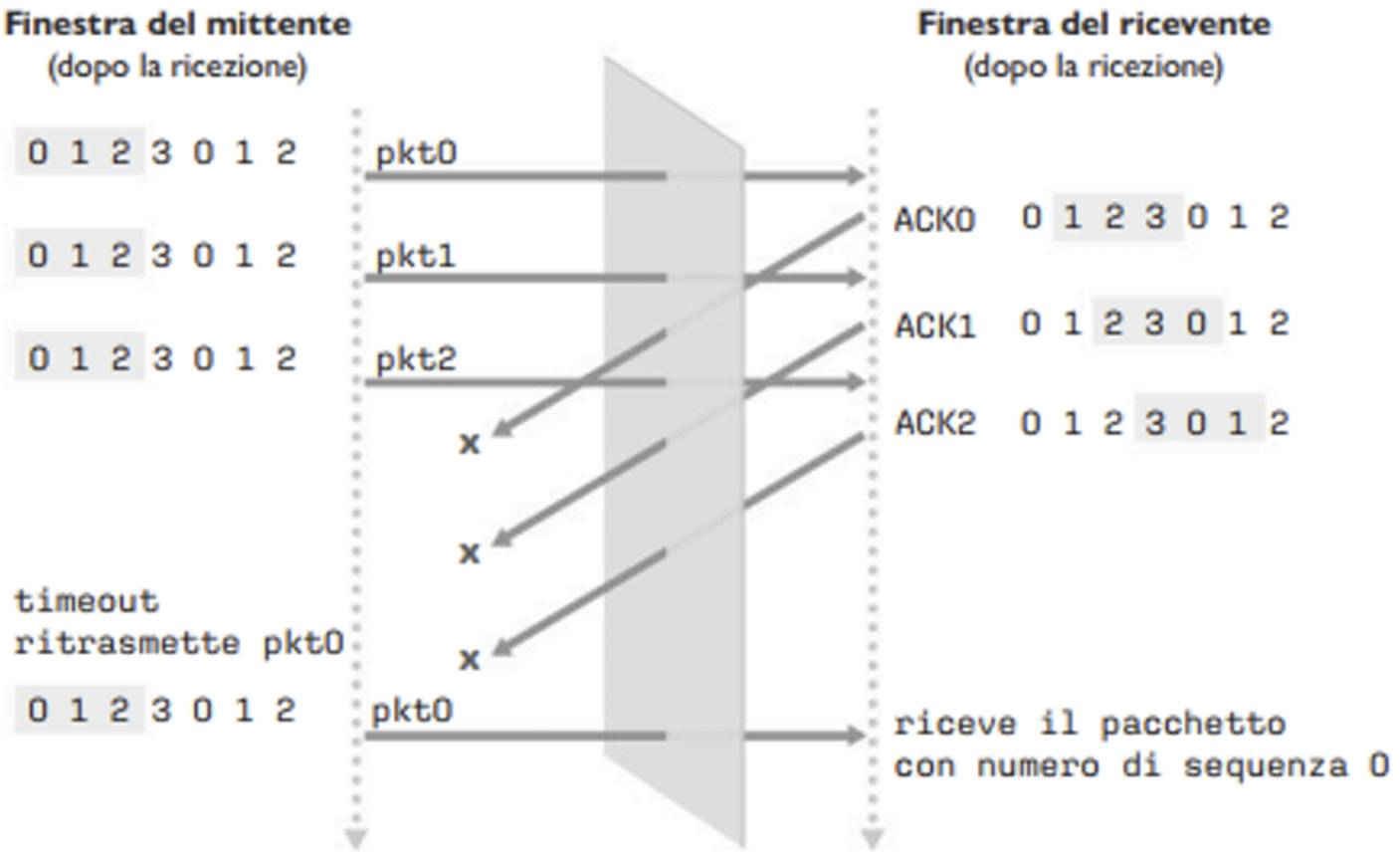
Destinatario SR

Analizziamo il receiver SR:

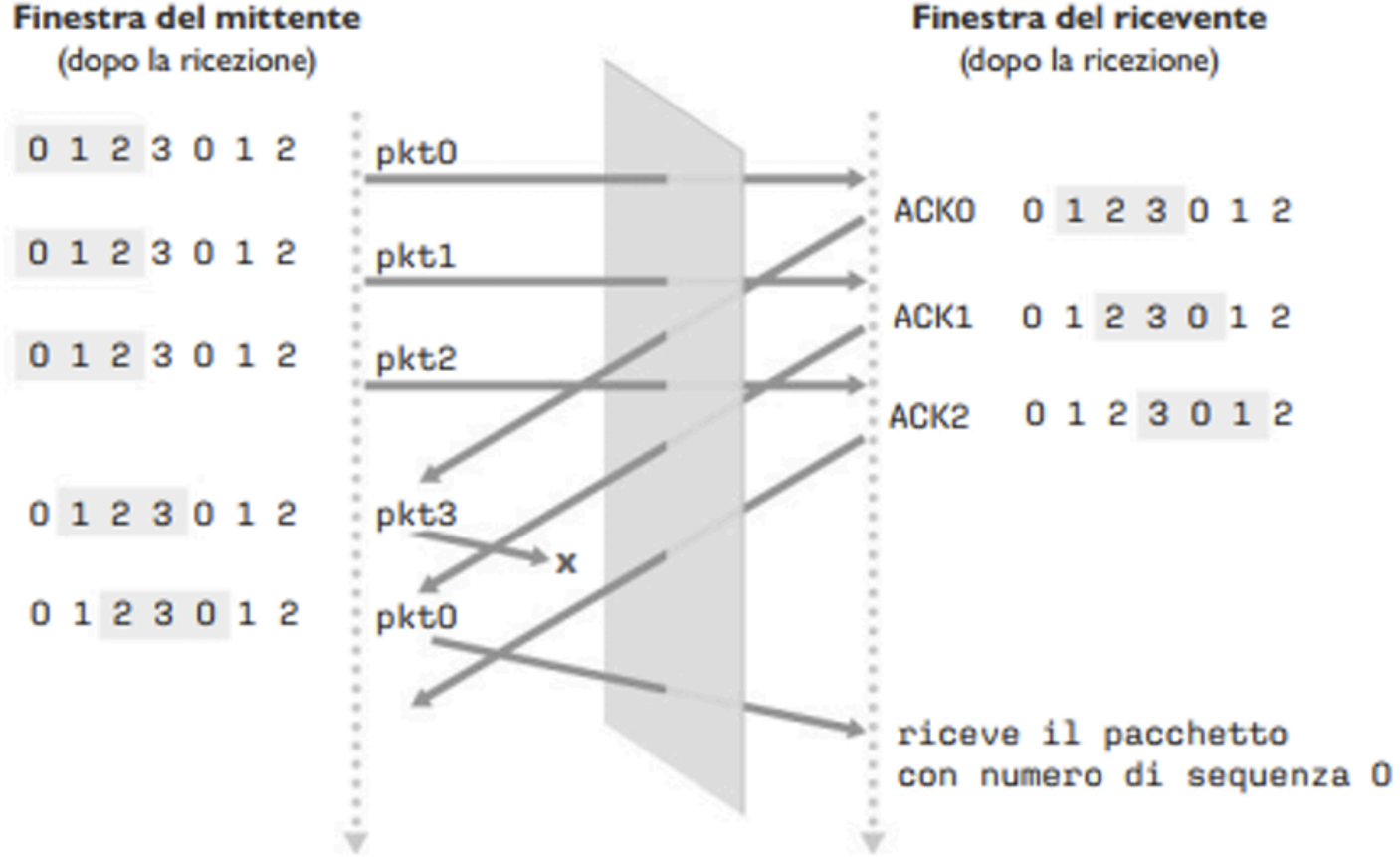
- a differenza del receiver GBN, il receiver SR accetta ogni pacchetto con numero di sequenza presente nella finestra (in range `[rcv_base, rcv_base + N - 1]`. Al sender viene restituito un ACK selettivo e il pacchetto viene inserito nel buffer. Le informazioni saranno passate al layer soprastante solo quando verrà ricevuto il pacchetto con numero di sequenza `rcv_base`: in tal caso, viene anche spostata la finestra;
- se viene ricevuto un pacchetto con numero di sequenza precedente la finestra (in range `[rcv_base - N, rvc_base - 1]`), si invia comunque un ACK, anche se il pacchetto è stato riscontrato;
- in ogni altro caso, **viene ignorato il pacchetto**.

Problema con il selective repeat

La mancanza di sincronizzazione tra le finestre del sender e del destinatario ha conseguenze importanti quando si utilizza un intervallo finito di numeri di sequenza.
Indichiamo con **Kseq** il numero totale di numeri di sequenza dei pacchetti. Ad esempio, se Kseq è uguale a 4, avremo i numeri di sequenza $0, 1, 2, 3$. Supponiamo di avere Kseq pari a 4 e $N = 3$. Supponiamo che i pacchetti $0, 1, 2$ siano trasmessi e ricevuti correttamente e che il receiver invii gli ACK. A questo punto, la finestra del destinatario si sposta per ricevere quarto, quinto e sesto pacchetto, che presentano numeri $3, 0, 1$. Consideriamo ora i due scenari:



a.



b.

Nel primo scenario nessun ACK viene ricevuto. Al timeout allora il mittente invia nuovamente il pacchetto 0. Nel secondo caso gli ACK vengono ricevuti correttamente. Notiamo che in entrambi i casi il receiver è pronto per ricevere il pacchetto 0. Tuttavia:

- nel primo caso, il pacchetto 0 è una **ritrasmissione del primo pacchetto**;
- nel secondo caso, il pacchetto 0 è la **trasmissione originaria del quinto pacchetto**.

Per il receiver **entrambi gli scenari sono identici** e non esiste alcun modo per distinguere la ritrasmissione del primo pacchetto dalla trasmissione originaria del quinto. Il receiver quindi non riesce a riconoscere correttamente in pacchetti. Affinché si possano distinguere correttamente i pacchetti, basta rispettare il seguente vincolo:

$$N \leq \frac{1}{2}\mathrm{Kseq}$$

TCP

Il **Transmission Control Protocol** è un protocollo di trasmissione **orientato alla connessione** (**connection-oriented**) che richiede che prima che i processi effettuino uno scambio di dati si effettui un **handshake**. Durante l'handshake vengono inviati alcuni segmenti preliminari per stabilire i parametri del successivo trasferimento dati.
Il TCP si occupa di:

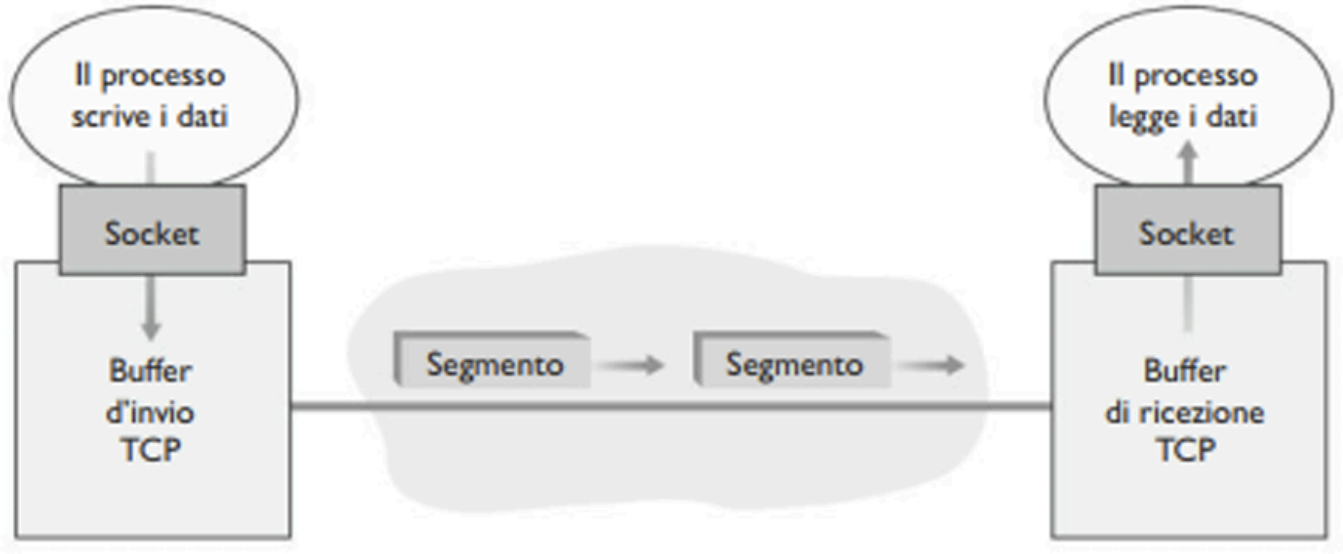
- **addressing** e **multiplexing**. Chi si occupa di far comunicare i processi nel client e i processi nel server è il transport protocol tramite i concetti di socket, multiplexing e demultiplexing.
- **connection establishment**, **management** e **termination**;
- **data handling** e **packaging**;
- **data transfer**;
- **reliability** e **transmission quality services**, con i meccanismi che abbiamo visto;
- **flow control** e **congestion avoidance**.

Il TCP **non** mantiene i message boundaries, non garantisce sicurezza e comunicazione. TCP trasmette un flusso continuo di byte, ma tocca all'applicazione delimitare i messaggi. Non si occupa neanche di definire il **throughput**, o di garantire il throughput ad un certo livello.
Il TCP è:

- **connection-oriented**: ci dev'essere una connessione stabilita;
- **bidirezionale**;
- **multidirezionale e endpoint-identified**;
- **affidabile**;
- **acknowledged**;
- **stream-oriented**: non si mandano **messaggi semantici**, ma una serie di **pacchetti**;
- **data-unstructured**: TCP manda byte, e sarà l'applicazione a ricostruire l'informazione;
- **data-flow-managed**.

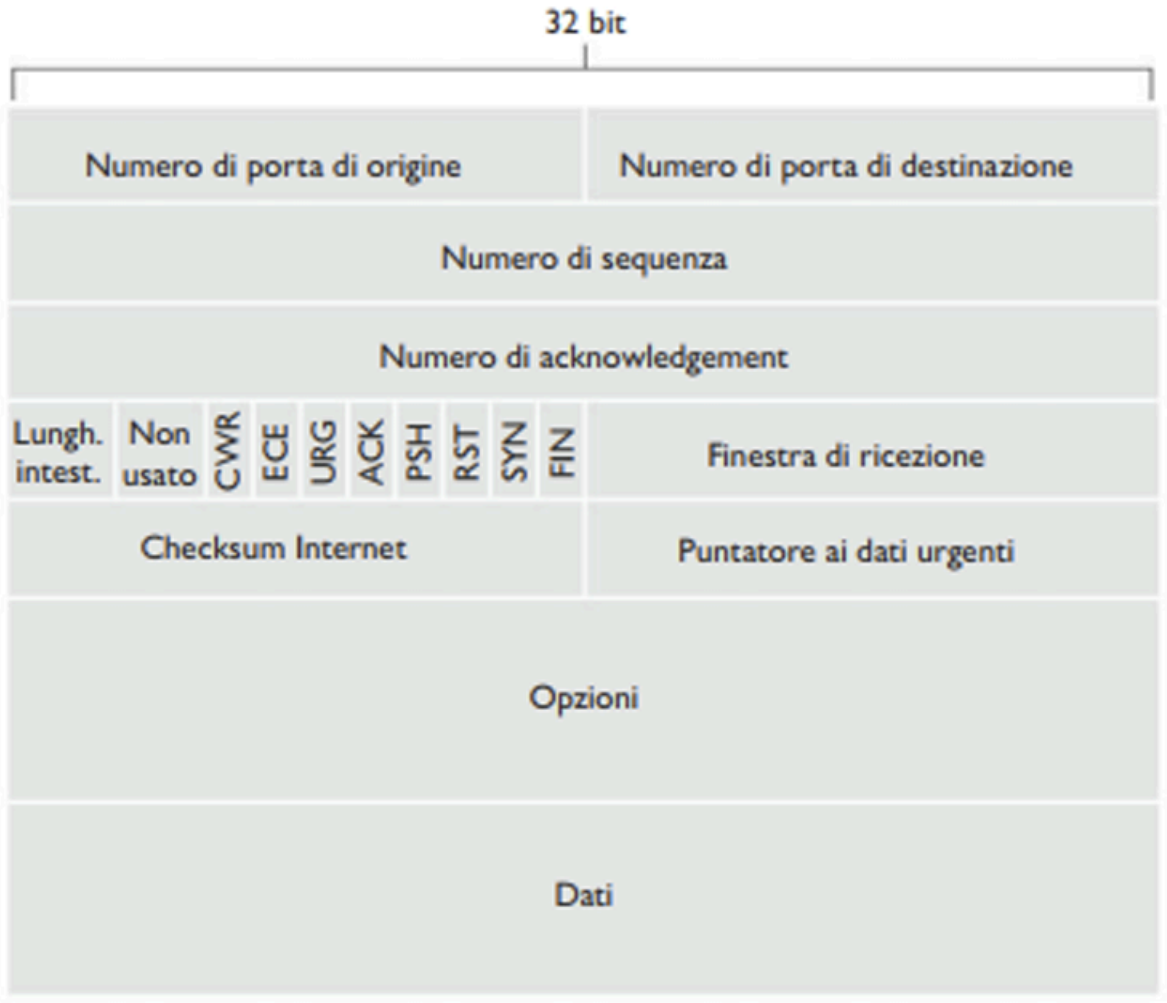
Segmento TCP

TCP accoppia ogni blocco di dati del client a header TCP, formando **segmenti TCP** che verranno passati al livello di rete, dove verranno incapsulati in datagrammi IP. Il server TCP invece riceve un segmento e lo memorizza in un buffer di connessione TCP. Il layer applicativo legge il flusso di dati dal buffer.
Sia il lato client sia il lato server hanno un buffer di invio e ricezione.



Il segmento TCP contiene header (campo intestazione) e un campo dati. La dimensione del campo dati è limitata dalla **Maximum Segment Size** (**MSS**) (che verrà analizzata successivamente). Gli header TCP contengono i seguenti campi:

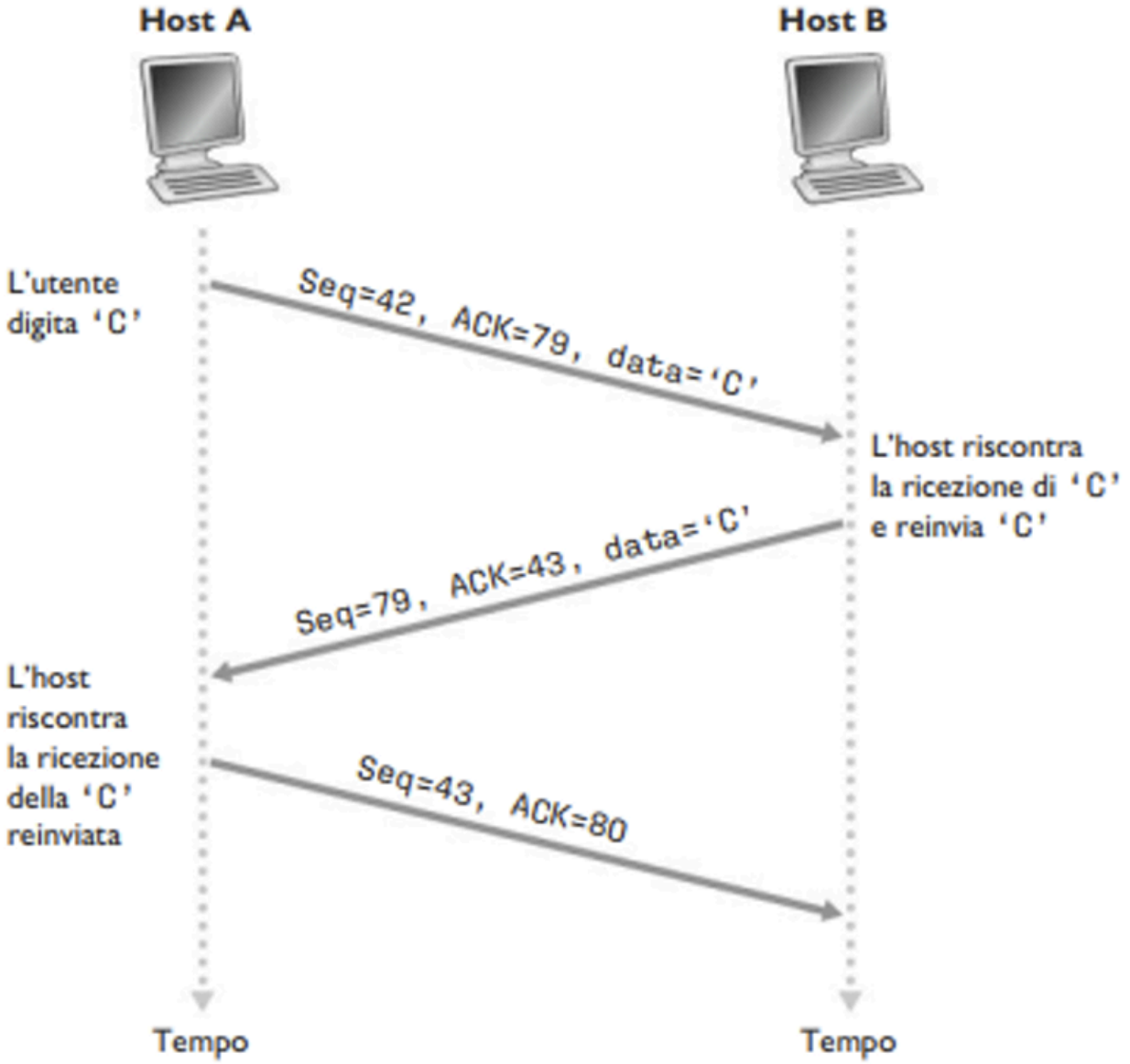
- **numeri di porta di origine e destinazione**, usati per multiplexing e demultiplexing dei dati e a 16 bit;
- **checksum**, con lo stesso scopo che ha nel caso del segmento UDP e a 16 bit;
- **numero di sequenza** e **numero di acknowledgment**, entrambi a 32 bit e usati per implementare il meccanismo di RDT (oggetto di studio nel prossimo paragrafo);
- **finestra di ricezione**, un campo a 16 bit per il **controllo di flusso**;
- **lunghezza dell'header**, a 4 bit che specifica la lunghezza dell'intestazioen TCP in **multipli di 32 bit**. Poiché ci sono più opzioni per TCP, l'intestazione TCP ha lunghezza variabile, ma solitamente è di **almeno** 20 byte;
- **opzioni** (**options**), facoltativo e di lunghezza variabile. Permette a sender e receiver di stabilire un valore MSS preciso o viene usato come fattore di scala per la window nelle reti ad alta velocità. Viene anche usata per il **time-stamping**, un meccanismo che permette agli endpoint di mantenere una misura dell'RTT, permettendo quindi di aggiustare in maniera dinamica di modificare i timer per il timeout;
- **flag**, campo a otto bit che contiene:
  - il bit **ACK**, usato per indicare che il valore nel campo di acknowledgment è valido (i.e. il segmento contiene un ACK per un segmento ricevuto con successo);
  - i bit **RST**, **SYN** e **FIN** usati per impostare e chiudere la connessione;
  - i bit **CWR** e **ECE**, usati per i controlli di congestione;
  - il bit **PSH**, usato per indicare che il destinatario deve inviare immediatamente i dati al livello superiore;
  - il bit **URG**, che indica che nel segmento sono presenti dati marcati come **urgenti**.
- **puntatore ai dati urgenti**, un puntatore a 16 bit che indica la posizione dell'ultimo byte di dati urgenti.



Solitamente URG e PSH non vengono utilizzati.

Numero di sequenza e numero di ACK

Il numero di sequenza e il numero di ACK sono i campi più importanti del TCP. TCP infatti vede i dati come un **flusso di byte ordinati ma non strutturati**. I numeri di sequenza fanno riferimento al flusso di byte trasmessi, **non alla serie di segmenti trasmessi**. Pertanto il numero di sequenza per un segmento è il numero nel flusso di byte del primo byte del segmento.
Poiché TCP è **full-duplex**, un host può contemporaneamente inviare e ricevere dati da un altro host. Se un host A invia segmenti ad un host B, i segmenti che provengono da B devono avere un numero di sequenza relativo ai dati che fluiscono da B ad A. Più precisamente, **il numero di ACK che A scrive nei propri segmenti è il numero di sequenza del byte successivo che A attende da B**. Quindi supponendo che A riceva da B un segmento contenente, ad esempio, i byte da 0 a 535 e un altro segmento che contiene i byte da 900 a 1000, il prossimo segmento che A invia conterrà nel campo ACK il valore 536.
Dato che TCP effettua l'ACK solo dei byte fino al primo byte mancante nel flusso, si dice che TCP offre **ACK cumulativi**. In questo modo il protocollo TCP permette risparmio di banda. Il meccanismo degli ACK cumulativi permette di risparmiare anche sugli ACK precedenti che non sono stati ricevuti dal sender.



Come si vede nell'esempio soprastante, che mostra una connessione Telnet, A invia un segmento con numero di sequenza 42 e ACK 79. Il prossimo segmento inviato da B ha numero di sequenza 79 e numero di ACK 43: questo numero corrisponde al numero di sequenza del prossimo segmento inviato da A.

Connessione TCP

La connessione TCP è **punto a punto** e non permette **multicast**, ossia trasferimento di dati da un mittente a molti destinatari in un'unica operazione.
La connessione TCP può essere divisa in due fasi: una fase di **apertura di connessione** e una **fase di chiusura**.

Fase di apertura

La fase di apertura di connessione viene detta **handshake a tre vie** (**three-way handshake**, anche chiamato **three-phase handshake**, ossia **handshake a tre fasi**).

Primo step

In un primo passo, il client TCP invia un segmento al client TCP del server. Il segmento non contiene dati a livello applicativo, ma uno dei bit dell'header, ossia il **bit SYN**, è posto ad 1: per tale motivo il segmento prende il nome di **segmento SYN**. Il client inoltre sceglie un **numero di sequenza iniziale** a caso `client_isn` e lo pone nel campo di sequenza del segmento SYN iniziale. Il segmento viene poi incapsulato in un datagramma IP e inviato al server.

Secondo step

Successivamente al passo due quando il datagramma viene ricevuto dal server, esso estrae il segmento TCP SYN e invia un segmento di connessione approvata al client TCP. In questo segmento:

- il bit SYN è posto ad 1;
- il campo ACK assume valore `client_isn + 1`;
- il numero di sequenza iniziale `server_isn` viene scelto dal server e inserito nel segmento.

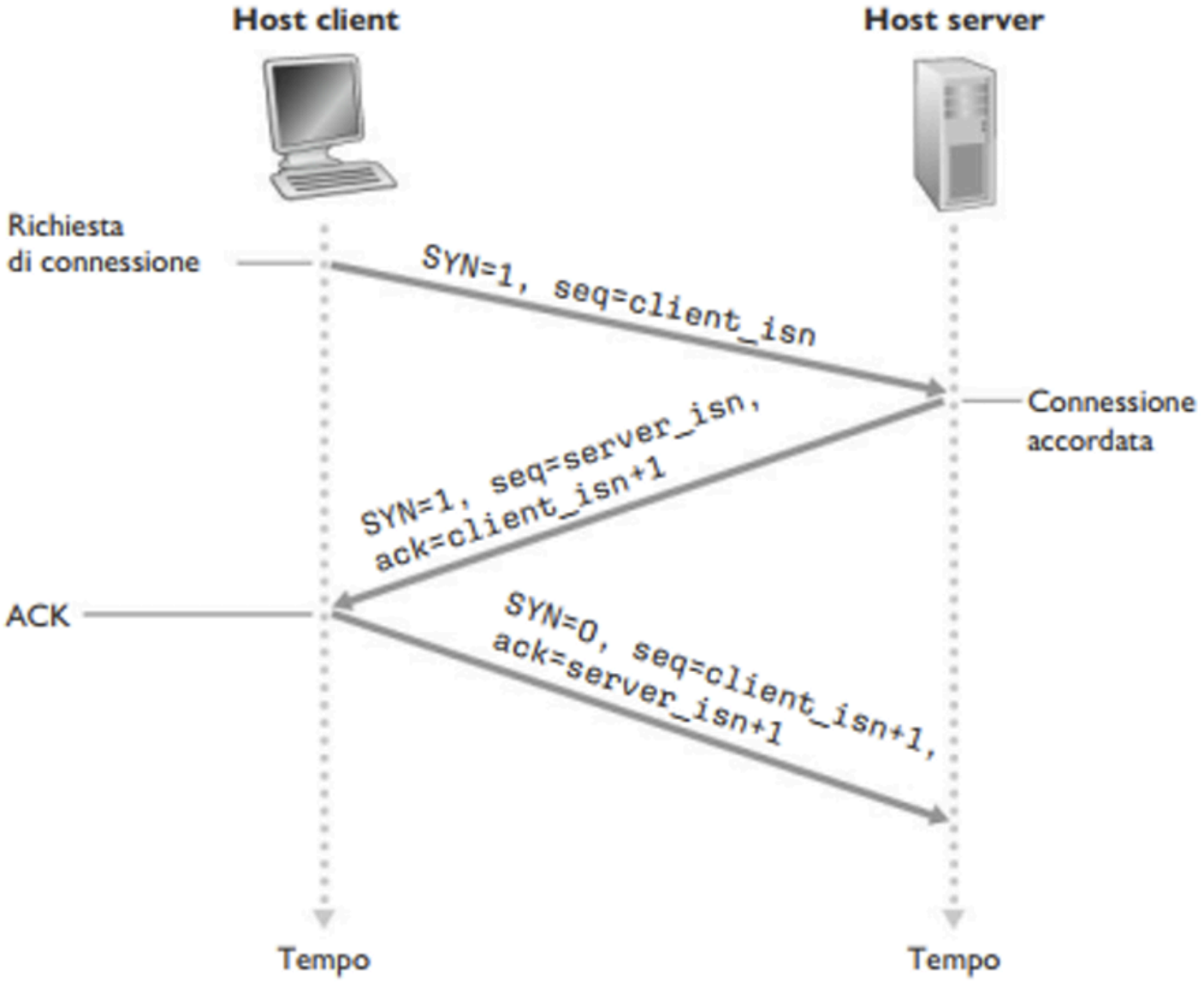
Il segmento inviato prende il nome di **segmento SYNACK**.

Terzo step

Alla ricezione del segmento SYNACK il client invia al server un altro segmento in risposta al segment odi connessione approvata del server. In tale operazione il client:

- pone nel campo ACK il valore `server_isn + 1`;
- pone il bit SYN a 0.

La connessione quindi è stabilita e i client e il server possono scambiare informazioni. Una volta che la connessione è instaurata, il client invia un flusso di dati utilizzando la socket. TCP poi dirige i dati verso il **buffer di invio** della connessione e passa i segmenti al livello di rete. Il server che riceve i segmenti li memorizza in un **buffer di ricezione** e li invia al layer di applicazione tramite il socket del processo ricevente.



Fase di chiusura

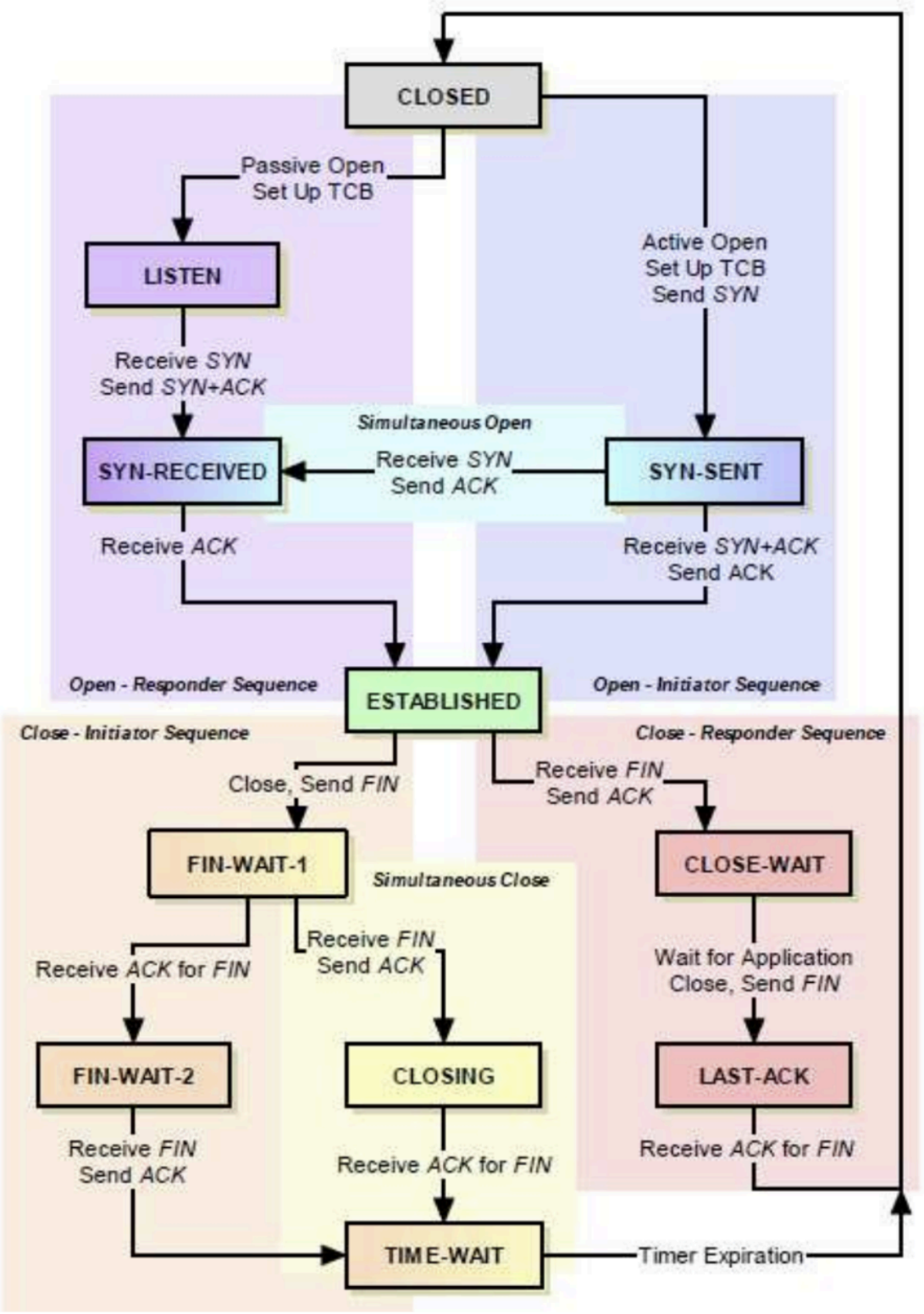
La fase di chiusura di connessione si basa su un **handshake a quattro vie** (**four-way handshake**), visto come un **doppio two way handshake**. Ciascuno dei due processi che partecipano alla connessione può terminarla inviando un pacchetto speciale. In particolare, supponiamo che il client voglia chiudere la connessione. Allora:

1. il client TCP invia un segmento TCP speciale, il cui header presenta il bit **FIN** posto ad 1;
2. quando il server TCP riceve il segmento, invia un ACK al client;

3. il server spedisce quindi il proprio segmento di shutdown con il bit FIN uguale a 1;
4. il client invia un ACK a quest'ultimo segmento del server. A questo punto le risorse di entrambi gli host sono deallocate.

Stati TCP

Nel corso della connessione TCP i protocolli TCP negli host attraversano vari **stati TCP**:



Nell'immagine il lato sinistro rappresenta il server TCP, mentre il lato destro il client TCP. Quando il client vuole stabilire una connessione, invia il segmento SYN ed entra in stato `SYN_SENT`, mentre il server crea un socket in ascolto ed entra in stato `LISTEN`. Alla ricezione del segmento, entra in stato `SYN_RECEIVED` inviando il segmento SYNACK. Dopo che il server riceve l'ACK, client e server entrano in stato `ESTABILISHED` che indica che la connessione è stabilita.
Quando il client invece vuole chiudere la connessione, invia il segmento con bit FIN posto ad 1 ed entra in stato `FIN_WAIT_1`, attendendo un ACK dal server. Quando il server invia l'ACK, esso entra in stato `CLOSE_WAIT`. Ricevendo l'ACK il client entra in stato `FIN_WAIT_2`. Quando il server invia il suo segmento con FIN a 1, entra in stato `LAST_ACK`. Il client invia l'ACK e transita in stato `TIME_WAIT`. Nello stato `TIME_WAIT`, il client attende tipicamente un tempo compreso tra 30 secondi e 2 minuti in cui può inviare nuovamente l'ultimo ACK nel caso in cui venga perso. Dopo l'attesa, la connessione è chiusa e si passa allo stato `CLOSED`.

MSS, RTT e throughput

Non conta solamente la reliability, ossia controllare il flusso e l'evitare di perdere i pacchetti, ma abbiamo anche un'altra necessità: il **throughput**. Per massimizzare il throughput, bisogna lavorare su alcuni campi:

- dimensione di segmento TCP;
- timeout.

Definiamo il **Maximum Segment Size** e il **Maximum Transmission Unit** rispettivamente come la **dimensione massima di segmento** e l'**unità trasmissiva massima**. Generalmente il valore MSS viene determinato in virtù della lunghezza del frame più grande che si può inviare nel layer di collegamento dal sender host: questa lunghezza è il valore MTU.
Il TCP richiede che tutte le reti trasferiscano pacchetti con **almeno** 576 byte, quindi significa che l'MSS minima è pari a 536. Ciò deriva dal fatto che 40 byte sono occupati dall'Header dell'IP e da quello del TCP (20 a testa).
Rimane il **timeout**, ossia quel tempo minimo che il sender deve aspettare tale che se un ACK non è stato ricevuto dal sender, questi deve rinviare il pacchetto. Chiaramente il timeout dev'essere superiore dell'RTT, ma:

- se il timeout è troppo piccolo, scatta troppe volte, inserendo latenza;
- se il timeout è troppo grande, c'è una reazione lenta alla perdita di pacchetti.

Per determinare il timeout, introduciamo il **SampleRTT**. Il SampleRTT è la misura del tempo tra la trasmissione del segmento fino a quando non si riceve l'ACK. Se non ci sono ritrasmissioni, si utilizza l'**algoritmo di Karn** (ciò implica che **la stima non è realmente accurata**, perché è come se si ignorasse la presenza di ritrasmissioni). Se ci sono ritrasmissioni, si utilizzano tecniche statistiche:

1. si prende una history profonda e si analizza la variazione del SampleRTT;
2. si esegue una media pesata.

Si utilizza una **EWMA** (**Exponential Weighted Moving Average**), che permette di calcolare quanto i campioni del passato influenzano il presente. In particolar modo, utilizzare una media mobile pesata esponenziale riduce l'influenza dei campioni del passato in modo tale che essa decresca in maniera esponenzialmente rapida.
Si ha allora che è possibile calcolare un RTT con la formula:

$$\text{Estimated_RTT}_n = (1-\alpha)\cdot \text{Estimated_RTT}_{n-1} + \alpha \cdot \text{Sample_RTT}_n$$

Un valore tipico di α è 0.125. Il valore di alpha viene modulato in virtù dei controlli congestione.
La stima dell'RTT è un evento statistico. Se stimiamo l'RTT ora, abbiamo un valore, ma tra un minuto ne avrà probabilmente un altro. Poiché è un evento variabile, si calcola la **deviazione standard** dell'RTT:

$$\text{DevRTT}_n = (1-\beta)\cdot \text{DevRTT}_{n-1} + \beta \cdot |\text{SampleRTT}_n - \text{EstimatedRTT}_n|$$

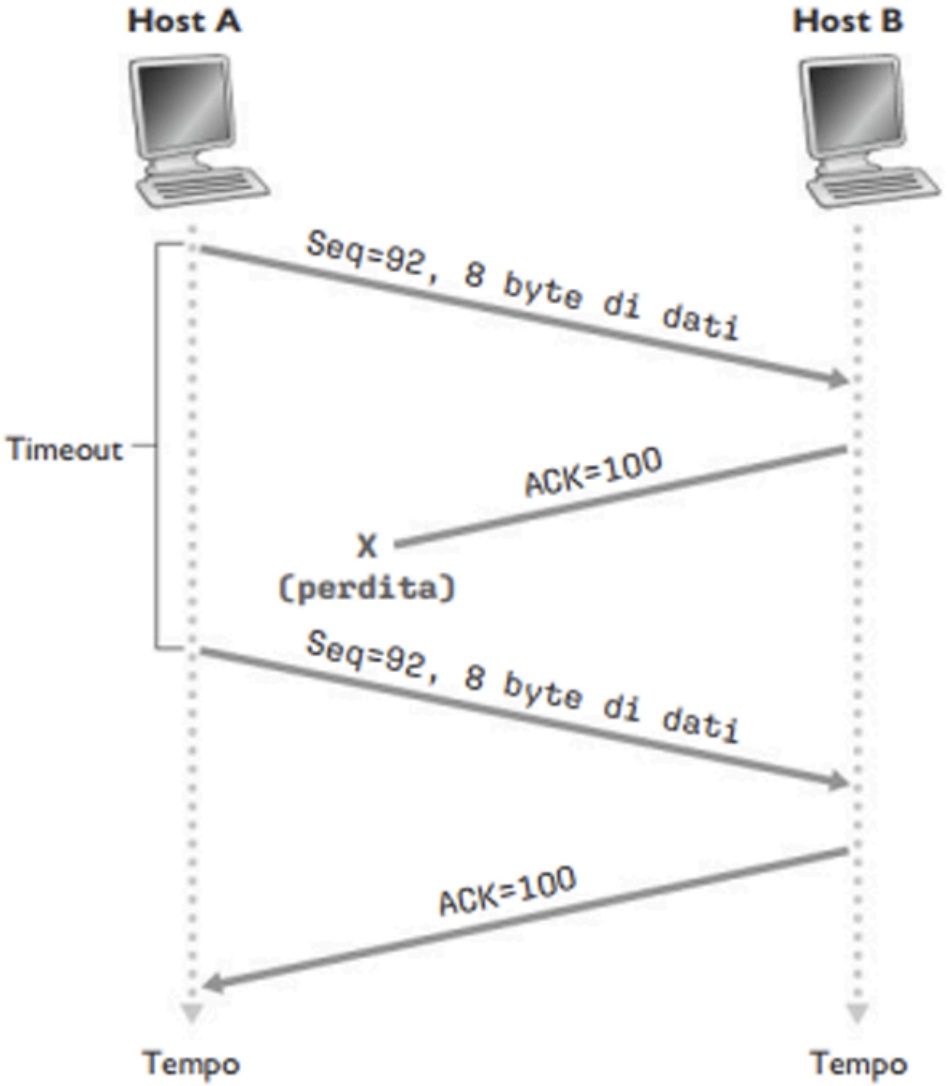
Valore tipico di β è 0.25.
Calcoliamo infine l'**RTO**, ossia il **tempo di ritrasmissione** (**Retransmission Timeout**), che corrisponde al tempo di attesa prima che un segmento TCP venga ritrasmesso se non è stato ricevuto l'ACK corrispondente. Questo è il **timeout** che cerchiamo.
Notiamo che il timeout non è statico, ma cambia durante la connessione. L'RTO si calcola come segue:

$$\text{RTO} = \text{EstimatedRTT} + 4 \cdot \text{DevRTT}$$

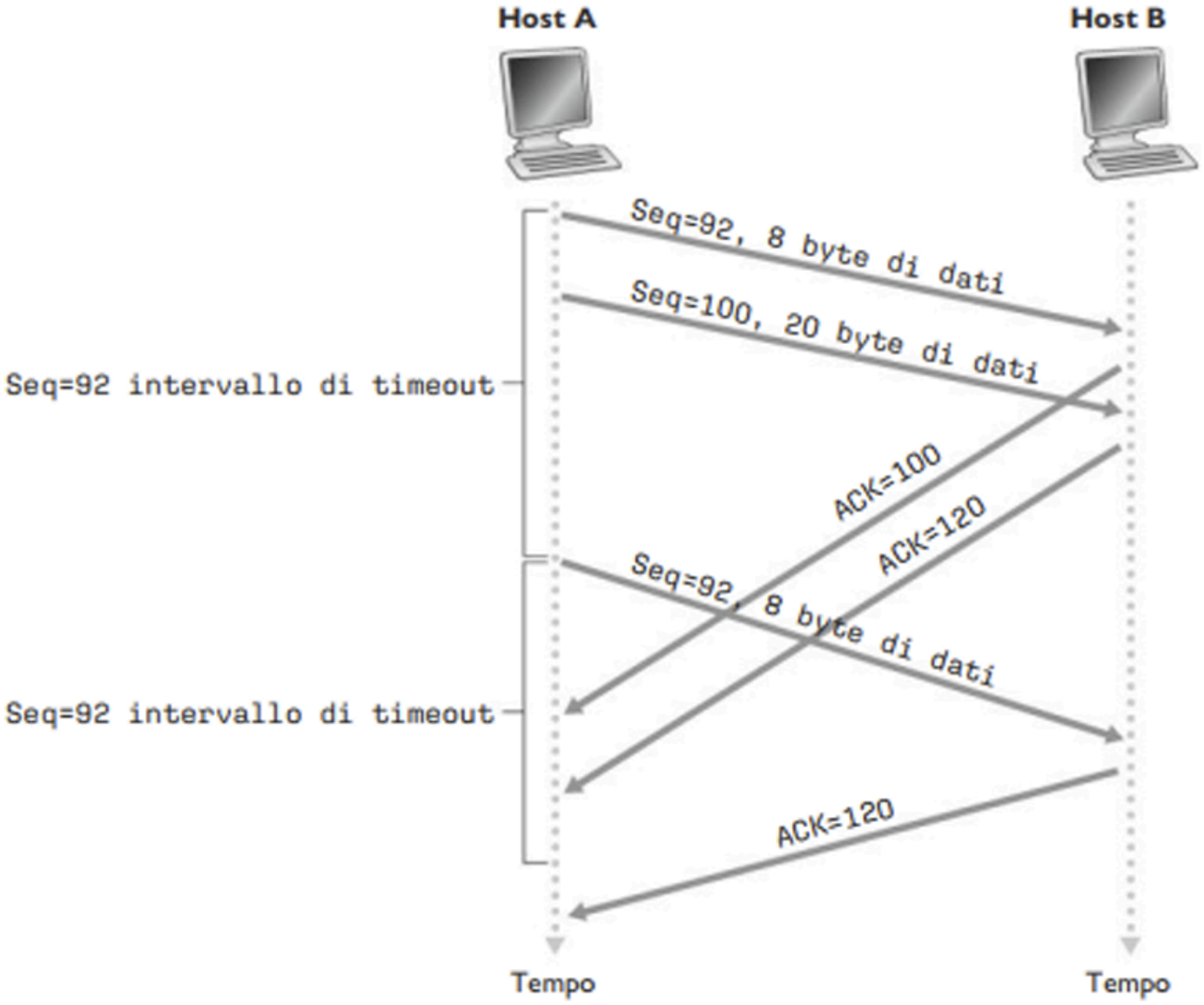
07 - Controllo di flusso e di congestione

Timer

TCP crea un servizio di **trasposto dati affidabile**. Per quanto abbiamo visto con i protocolli RDT ad ogni singolo segmento trasmesso era assegnato un timer. In questo modo, tuttavia, la gestione dei timer richiederebbe considerevoli risorse aggiuntive. Si utilizza pertanto un unico timer di ritrasmissione. Dopo l'arrivo di un ACK, un nuovo timer viene inizializzato, collegato al prossimo segmento di cui non è stato mandato un ACK.
Generalmente quando un segmento ACK viene perso, il sender TCP invia dopo il timeout nuovamente il segmento e attende per un segmento ACK, come mostrato in foto:



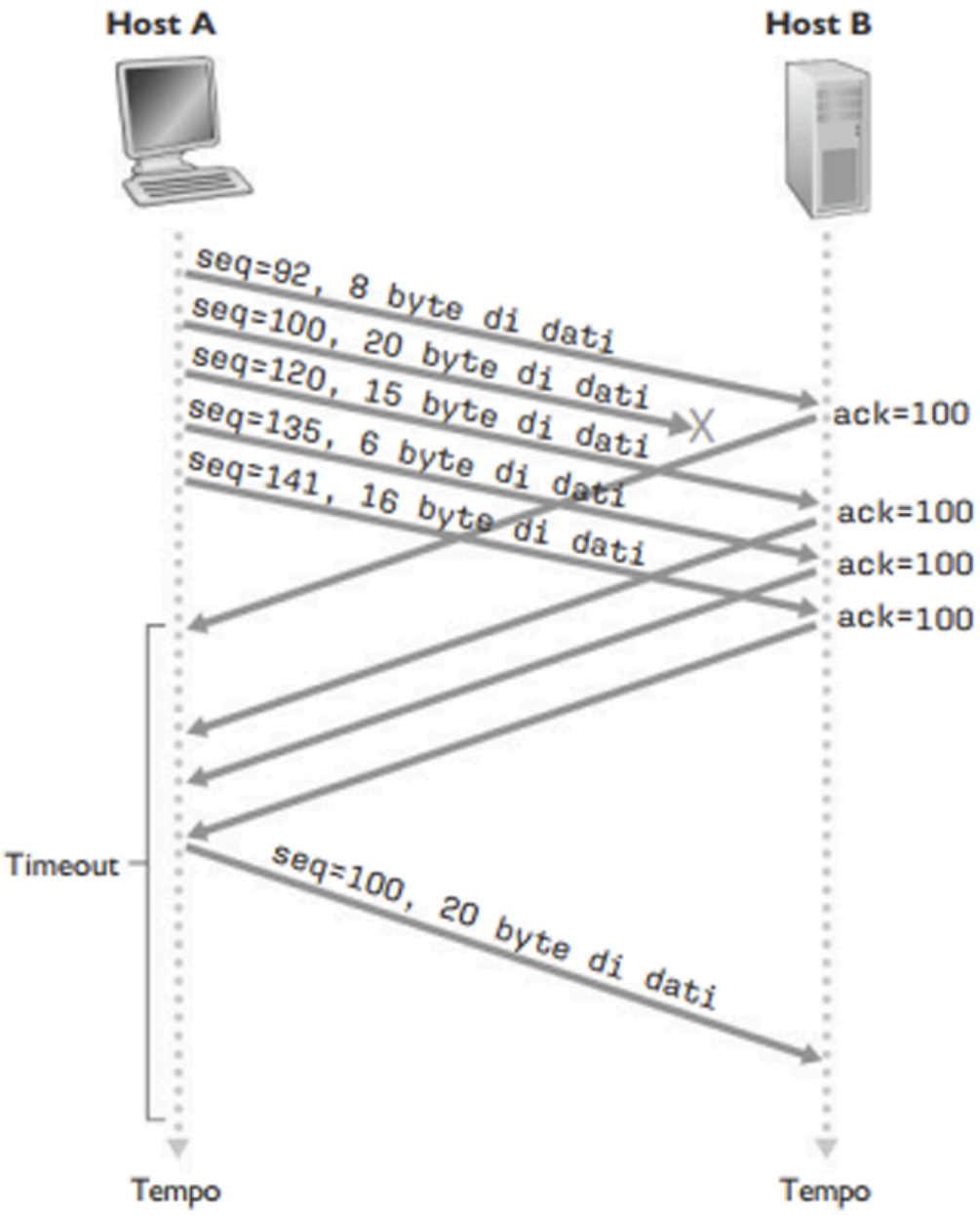
Poiché TCP utilizza ACK cumulativi, per evitare la ritrasmissione di pacchetti il receiver manda un ACK con numero di sequenza pari all'ultimo segmento TCP ricevuto. In questo modo se ad esempio un sender invia più di un segmento usando la pipeline, non si ha la necessità di ritrasmettere ogni singolo pacchetto.



Nell'esempio soprastante, il sender invia due segmenti TCP con numero di sequenza $92, 100$, e attende rispettivamente un ACK 100 e un ACK 120 (ricordando che il numero di sequenza del pacchetto è il numero del primo byte del segmento da inviare, mentre il numero di ACK che il sender scrive nel segmento è il numero di sequenza che attende dal receiver). Il sender va in timeout, sebbene il receiver abbia ricevuto entrambi i segmenti; ritrasmette così il primo segmento. Tuttavia, il receiver ha ricevuto correttamente entrambi i segmenti, pertanto invia un ACK con numero di sequenza 120. Tale ACK comunica la ricezione di entrambi i pacchetti.

Fast Retransmit

Il periodo di timeout può essere relativamente lungo, pertanto quando viene smarrito un segmento il tempo di attesa per la ritrasmissione può causare latenze. Tuttavia è possibile individuare se un pacchetto è stato perso **prima che si verifichi il timeout** tramite gli **ACK duplicati**.
Quando il receiver riceve un segmento TCP con numero di sequenza superiore a `rcv_base` rileva un segmento mancante. Il receiver allora invia un ACK relativo all'ultimo byte che ha ricevuto in ordine, duplicando così un ACK. In molti casi il sender TCP invia un gran numero di segmenti, pertanto se uno di questi segmenti viene smarrito vi saranno molti ACK duplicati. Il sender interpreta tre ACK duplicati per lo stesso segmento come indice che il segmento successivo è stato perso. Il sender allora esegue una **ritrasmissione rapida** (**fast retransmit**), rispedendo il segmento mancante prima del timeout.



TCP Flow Control

In una connessione TCP i byte ricevuti correttamente e in sequenza vengono posizionati nel buffer di ricezione. Il processo nel layer di applicazione legge i dati dal buffer, ma non necessariamente nell'istante in cui vengano ricevuti. Potrebbe anche succedere che il network layer invii dati più rapidamente rispetto alla rapidità con cui il layer di applicazione rimuove dati dal buffer di ricezione. Accade quindi che il sender **mandi in overflow** il buffer di ricezione: se ciò accade, il buffer non riesce a memorizzarli e i dati vengono persi.

Una delle grosse differenze tra TCP e UDP è il **controllo di flusso**. TCP offre un servizio di control-flow per evitare saturazioni del buffer, paragonando la frequenza di invio del mittente con quella di lettura dell'applicazione ricevente. Il controllo di flusso viene garantito facendo mantenere una variabile chiamata **Receive Window** (indicata con `rwnd`), ossia **finestra di ricezione**, che fornisce al mittente informazioni relative allo spazio libero nel buffer del receiver.
Indichiamo con `RcvBuffer` la dimensione del buffer del receiver, con `LastByteRead` il numero dell'ultimo byte letto dall'applicazione nel buffer e con `LastByteRcvd` il numero dell'ultimo byte copiato nel buffer di ricezione. Possiamo garantire che non si avrà buffer overflow se si ha:

$$\text{LastByteRcvd - LastByteRead} \leq \text{RcvBuffer}$$

da cui si deduce che la quantità di dati presenti all'interno del buffer è data da $\text{LastByteRcvd - LastByteRead}$. Allora possiamo dire che la finestra di ricezione `rwnd` è data da:

$$\text{rwnd} = \text{RcvBuffer - [LastByteRcvd - LastByteRead]}$$

La variabile `rwnd` viene comunicata all'host sender nel campo finestra di ricezione presente nel segmento TCP.

A sua volta il sender memorizza `LastByteSend`, ossia l'ultimo byte mandato, e `LastByteAcked`, ossia l'ultimo byte per cui è stato ricevuto un ACK. La differenza `LastByteSend - LastByteAcked` esrpime la quantità di dati spediti per cui non è stato ricevuto un ACK. Il buffer overflow viene evitato se viene rispettata la condizione:

$$\text{LastByteSend - LastByteAcked} \leq \text{rwnd}$$

Questo meccanismo sembra perfetto, ma in realtà esiste un problema nel momento in cui $\text{rwnd} = 0$. Infatti se $\text{rwnd} = 0$ e il receiver non ha più segmenti da inviare, egli non verrà notificato del fatto che il buffer può essersi liberato, pertanto l'host sender non potrà più inviare nuovi dati se è necessario. TCP allora prevede che il sender invii segmenti a 1 byte di dati quando $\text{rwnd} = 0$. Il receiver rispondera con degli ACK, impostando il campo intestazione per la finestra di ricezione con `rwnd`.

Algoritmo di Nagle

La risoluzione del caso $\text{rwnd} = 0$ nasconde un grande overhead. Infatti i pacchetti hanno un header di 40 byte (20 per TCP, 20 per IP), risultando in un pacchetto di 41 byte per contenere solo 1 byte di informazione. Questo spesso accade nelle sessioni **Telnet** dove la pressione di molteplici tasti porta alla generazioni di singoli byte di dati. Su collegamenti lenti, i pacchetti transitano contemporaneamente portando ad un **collasso di congestione**.

Nel 1984 John Nagle introdusse un algoritmo che memorizza i pacchetti di piccole dimensioni nel buffer e le invia tutte insieme. Più nello specifico, fino a quando c'è un pacchetto per cui non è stato ricevuto un ACK, il sender deve memorizzare nel buffer di invio i segmenti fino a quando non si ha una sufficiente quantità di dati da inviare.

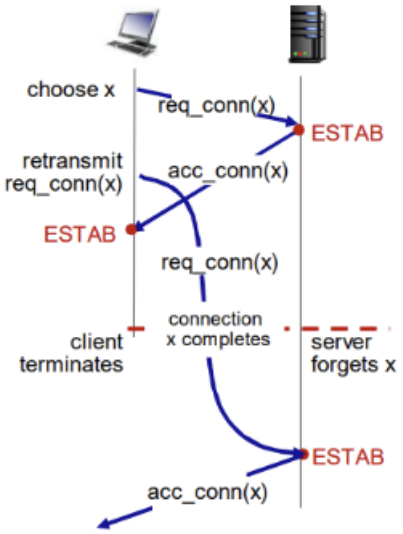
L'algoritmo può essere illustrato con lo pseudocodice:

```
if available_data > 0 then
    if rwnd >= MSS and available_data >= MSS then
        send_a_MSS_segment
    else
        if waiting_for_an_ack == true then
            enqueue_data_in_buffer
        else
            send_data
        endif
    endif
endif
```

L'algoritmo tuttavia non opera bene per il fatto che TCP utilizza un meccanismo di **ACK ritardati**. In generale se un receiver riceve un segmento con numero di sequenza atteso, e tutti i dati sono stati già riscontrati correttamente, il receiver attende fino a 500 millisecondi, in modo tale da evitare l'invio di più ACK. Questo algoritmo è definita in **RFC 1122**. Con entrambi i meccanismi attivi, applicazioni che eseguono due scritture sulla connessione TCP più una lettura che non viene eseguita fino a quando la seconda scrittura ha raggiunto la destinazione soffrono di un delay costante di 500 millisecondi. Nagle propose di evitare operazioni del tipo *write-write-read*, e di inserire nel buffer di invio di TCP le scritture e inviarle tutte singolarmente. Infine, in applicazioni in cui si richiede **bassa latenza e alta reattività** è **sconsigliato l'utilizzo di Nagle**, in quanto **aumenta le latenze**.

Three Way Handshake e possibile Two way handshake

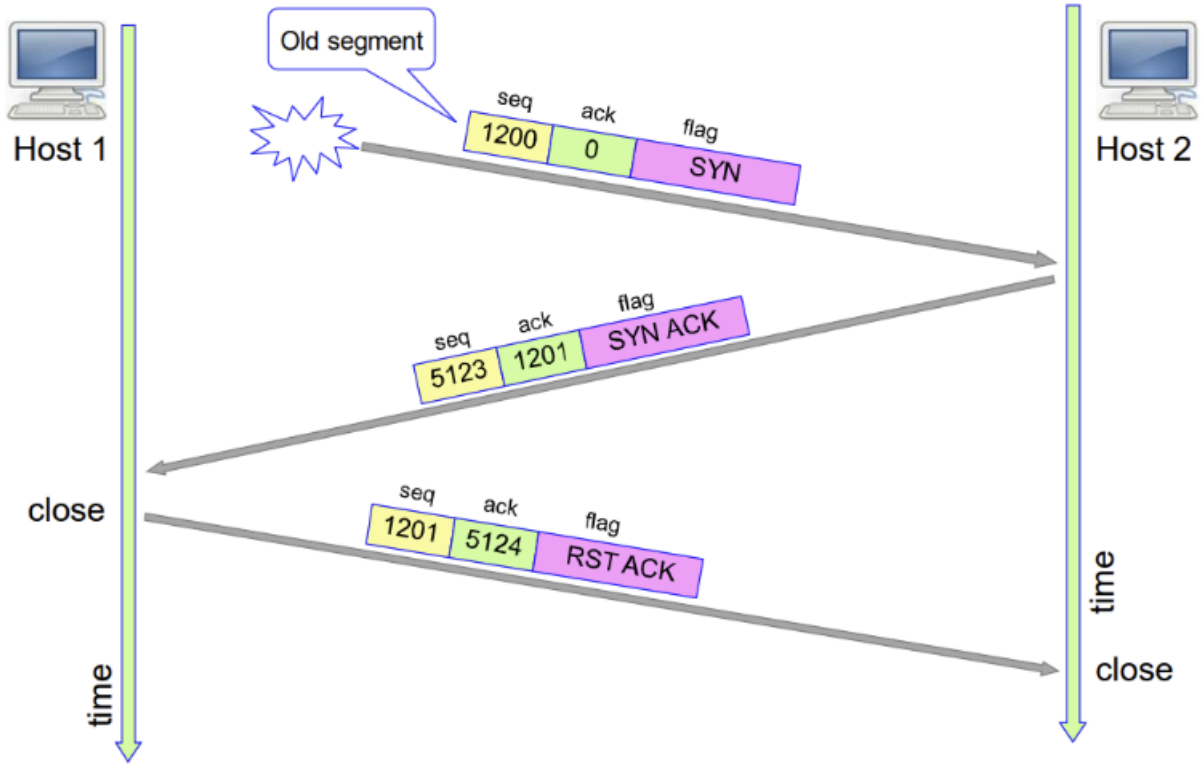
Per emulare il comportamento umano, inizialmente il Three-Way handshake era stato concepito come un **Two-Way handshake**, quindi un **handshake a due vie**. Tuttavia, quello che può accadere è uno scenario di **half connection**.
Supponiamo che il messaggio ACK da parte del receiver arrivi in ritardo; il sender vada in timeout. Per il server tuttavia la connessione è stabilita. Il client invia nuovamente una richiesta per stabilire una connessione, e che per qualche motivo il client dopo che la richiesta viene inoltrata vada in crash. Il pacchetto viene inviato, e il server stabilisce la connessione inviando un ACK. Ma poiché il server ha già stabilito la connessione e il client è in crash, **l'ACK non verrà mai ricevuto**. Questa connessione è **half-open**, ossia **apertà per metà**, in quanto il client è in crash, mentre il server è in attesa.



Il meccanismo del Three-Way handshake risulta quindi necessario per evitare tali scenari.

Gestione del three way handshake

Quando il sender va in crash e riceve un segmento SYNACK come risposta ad un segmento vecchio, il sender pone a 1 il **flag RST** nel campo intestazione. In questo modo la connessione viene chiusa. Con il TCP three-way si evita quindi che un elemento di terze parti possa inserirsi nella connessione. Poiché i numeri di sequenza di un host devono corrispondere con i numeri di ACK dell'altro, si possono determinare anomalie nella connessione facilmente.



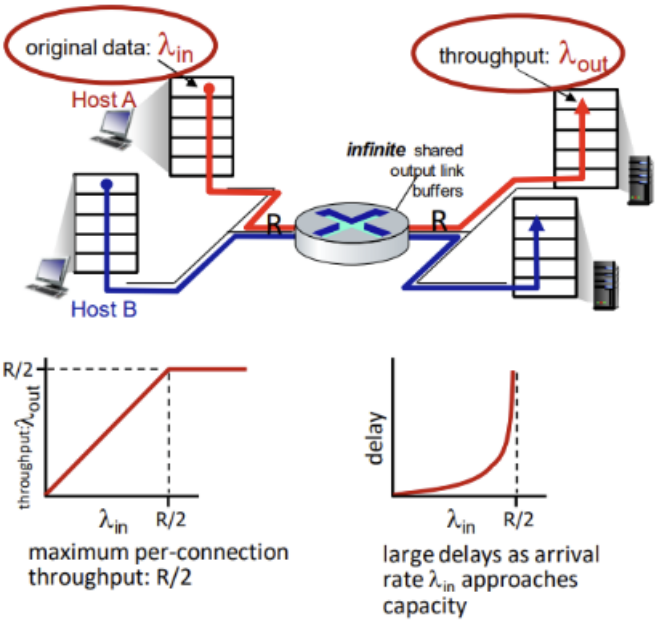
Analisi dei costi di congestione

Risulta necessario avere algoritmi in grado di calibrare sia il flusso sia la congestione. Il problema di flusso lo abbiamo già analizzato. Il problema di congestione invece è differente: non si tratta di latenza di rete, ma di **traffico di rete**, per cui indipendentemente dal flusso si crea latenza.
Per analizzare la congestione, analizziamo tre scenari:

1. uno scenario in cui due host comunicano tramite un router con **buffer illimitati**;
2. uno scenario in cui due host comunicamo tramite un router con **buffer limitati**;
3. uno scenario in cui quattro host comunicano tramite **più router**, con **percorsi composti da più collegamenti** e **buffer limitati**.

Primo scenario

Distinguiamo due host, A e B, che inviano dati tramite un router intermedio. Ipotizziamo che A stia inviando dati a una frequenza media di λ_{in} byte al secondo: questa grandezza indica il **tasso di trasmissione di pacchetti dal sender**. In questa situazione, supponiamo che ciascuna unità di dati venga mandata **solo una volta**, senza quindi la necessità di avere ritrasmissioni. L'host B opera in maniera simile, trasmettendo i dati ad un tasso pari a λ_{in}. I dati inviati transitano attraverso un router a **buffer infiniti** (scenario chiaramente irrealistico), con capacità di trasmissione pari ad R.\
Analizziamo ora il **throughput per connessione** (λ_{out}), ossia il tasso di dati che vengono consumati dal receiver, in virtù del tasso di trasmissione. Poiché il router trasmette con capacità R, ogni host può trasmettere con un tasso λ_{in} massimo pari a $\frac{R}{2}$, conseguenza della condivisione della capacità di trasmissione. Il throughput del ricevente equivale al tasso di invio del mittente fino a quando questo non supera il limite enunciato precedentemente. Superato questo limite, il throughput non aumenta: il router infatti non può trasmettere ad un tasso superiore rispetto alal sua capacità. Inoltre, ottenere un throughput pari a $\frac{R}{2}$ non è un ottimo risultato, in quanto il ritardo cresce di più. Quando si supera il limite imposto dalla capacità del router e dal numero di collegamenti che utilizzano il router, il numero di pacchetti in coda cresce e il ritardo tende a infinito.



Secondo scenario

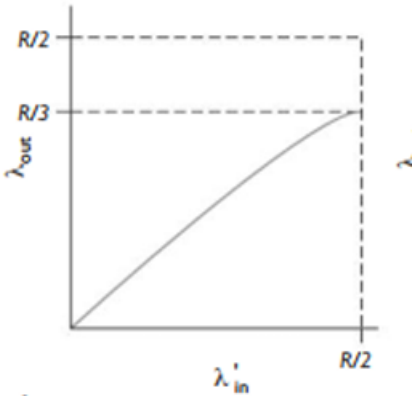
Modifichiamo lo scenario precedente, assumendo stavolta che la dimensione dei buffer nei router sia limitata. Quando i buffer del router sono pieni, i pacchetti che giungono vengono scartati, costringendo l'host sender a ritrasmetterli. La nozione di "tasso di invio" non può essere semplice come quella precedente. Distinguiamo allora due accezioni di questa locuzione:

- **tasso di trasmissione**, che indica sempre il numero di byte trasmessi alla socket per unità di tempo, λ_{in};
- **carico offerto alla rete** (**offered load**), che indica il tasso al quale il transport layer invia segmenti, λ'_{in}.

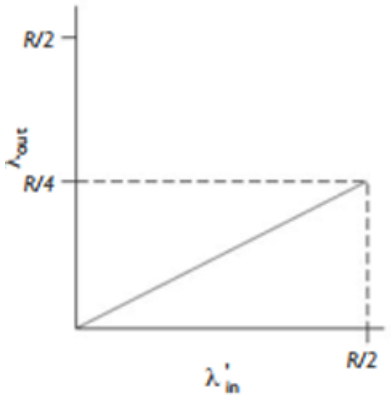
Le due grandezze differiscono per il fatto che nel caso in cui si abbiano ritrasmissioni, il tasso di trasmissione non coincide con il carico offerto alla rete: infatti λ_{in} denota **solo i dati originari, non le ritrasmissioni**. Per comprendere meglio le variazioni di prestazione, considereremo tre casi:

1. un caso in cui il sender determini **automaticamente** quando il router ha spazio a disposizione nei buffer (**perfect knowledge**);
2. un caso in cui il sender ritrasmette pacchetti quando è certo di averli persi (**some perfect knowledge**);
3. un caso in cui il sender ritrasmette pacchetti anche quando è andato in timeout prematuramente (**un-needed duplicates**).

Notiamo che sebbene il primo caso sia totalmente irrealistico, i due restanti non sono scenari lontani dalla realtà. Nel primo caso si ha infatti che, poiché il sender è in grado di determinare lo spazio rimanente nel buffer, $\lambda'_{in} = \lambda_{in}$ e il throughput sarebbe λ_{in}.
Se il sender ritrasmette pacchetti quando è certo di averli persi (ad esempio, se il timeout è sufficientemente grande da poter determinare con certezza che il pacchetto è perso) le prestazioni mutano, come mostra il grafico sottostante:

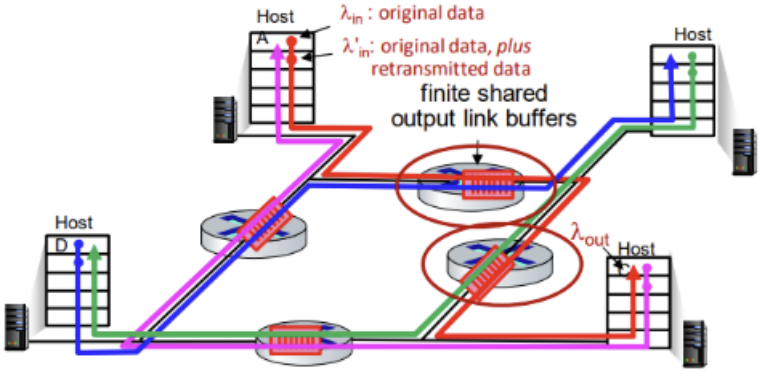


Sull'asse y è riportato il throughput per connessione, mentre sull'asse x il carico offerto (che è diverso dal tasso di trasmissione, poiché ai dati originali si aggiungono le ritrasmissioni). Il grafico, in particolare, mostra che con un carico offerto di $\frac{R}{2}$ si raggiunge un throughput di $\frac{R}{3}$. Questo indica che circa $\frac{1}{3}$ dei pacchetti viene perso. L'esempio mostra quindi che quando il sender deve ritrasmettere i pacchetti perduti, si ha un prezzo da pagare in termini di throughput. La perdita di pacchetti può essere dovuta anche a overflow dei buffer del router: questo è un costo della congestione.
Oltre alle ritrasmissioni di pacchetti persi, si possono anche avere ritrasmissioni **non necessarie** dovute a **timeout prematuri**. Il receiver, tuttavia, potrebbe già aver ricevuto correttamente il pacchetto, ma un timeout prematuro costringe il sender a inviare nuovamente un pacchetto. Questo mostra un ulteriore costo della congestione: ritrasmissioni non necessarie da parte del sender a causa di ritardi possono costringere il router a usare la larghezza di banda del collegamento per instradare copie non necessarie di un pacchetto. In questo caso, la curva che esprime il throughput in virtù del carico offerto è ancora più bassa:

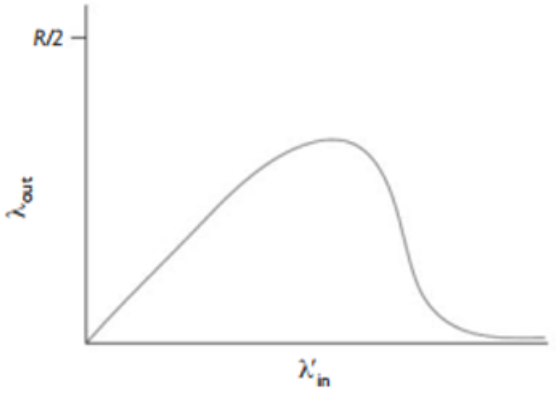


Terzo scenario

Supponiamo di trovarci nel seguente scenario:



Abbiamo quattro sender che inviano pacchetti, instradati in percorsi multi-hop (quindi attraverso più router). Ogni host utilizza meccanismi di timeout e ritrasmissione, implementando il servizio RDT. Supponiamo, inoltre, che i quattro router trasmettano a capacità R. Consideriamo la connessione tra l'host in alto a sinistra e l'host in basso a destra, utilizzando i router centrale-alto e destro. Per il router destro, consideriamo anche il traffico scambiato tra i due host.
Se i valori λ_{in} sono molto piccoli, gli overflow di buffer sono estremamente rari e il throughput sono paragonabili al carico offerto alla rete. Per valori più alti di λ_{in} (e quindi anche di λ'_{in}) la situazione è molto diversa. Il traffico che giunge al router a destra non può presentare un tasso maggiore della capacità di collegamento tra il router centrale-alto e il router a destra. Però se il carico offerto alla connessione è alto per ogni host, a causa del limitato spazio nei buffer e della **competizione** per questo, il throughput diminuisce. Nel caso limite, il buffer vuoto del router a destra viene immediatamente riempito dai pacchetti inviati dall'host in alto a destra e il throughput della connessione viene annullato. Il grafico a seguire sintetizza il discorso:



\

Controllo di congestione

Esistono due principali metodi per il controllo di congestione:

- **controllo end-to-end**. Il livello di rete non fortnisce supporto al livello di trasporto, pertanto il controllo di congestione dev'essere implementato a livello di trasporto, controllando la perdita di segmenti TCP e diminuendo eventualmente l'ampiezza della propria finestra;
- **controllo di congestione network-assisted**. Il livello di rete fornisce un feedback esplicito al sender sullo stato di congestione della rete, come un bit che indica traffico o direttamente informando il sender sulla frequenza trasmittiva che il router può supportare. Ad esempio, un router può inviare al mittente un **chokepacket**, ossia un **pacchetto di strozzatura**, che comunica al sender che il router è congestionato. Alternativamente, il router imposta un campo nel pacchetto che dev'essere transitato per indicare lo stato di congestione.

Per il controllo di congestione, TCP adotta un approccio end-to-end, imponendo a ciascun sender un limite al tasso di invio sulla propria connessione in virtù della congestione percepita.

Finestra di congestione

Per ridurre la velocità di invio sulla propria connessione, TCP utilizza la **finestra di congestione**.
Abbiamo già visto che per il controllo di flusso basta che valga la condizione:

$$\text{LastByteSend - LastByteAcked} \leq \text{rwnd}$$

Aggiungiamo la variabile di finestra di congestione `cwnd`. La quantità di dati che non hanno ancora ricevuto ACK non può eccedere il minimo tra i valori di `cwnd` e `rwnd`:

$$\text{LastByteSend - LastByteAcked} \leq \min\{\text{rwnd}, \text{cwnd}\}$$

Se consideriamo una situazione in cui possiamo trascurare perdita di pacchetti e ritardi di trasmissione, e assumendo che il vincolo di finestra di ricezione venga rispettato, approssimativamente all'inizio di ogni RTT il sender invia un numero di byte pari a `cwnd`. Si ha un tasso di invio pari a $\frac{\text{cwnd}}{\text{RTT}}$ byte/s. Modificando il valore di `cwnd` il sender può quindi regolare la velocità di invio dei dati sulla connessione. Per modulare il valore di `cwnd`, TCP segue i principi guida:

- **un segmento perso implica congestione, quindi i tassi di trasmissione del sender TCP devono essere decrementati**;
- **la ricezione di ACK non duplicato indica che il tasso di trasmissione può essere aumentato**.

Algoritmo per il controllo di congestione: AIMD

L'algoritmo per il controllo della congestione di TCP è un algoritmo della forma **incremento additivo, decremento moltiplicativo** (Additive-Increase Multiplicative-Decrease, AIMD). L'algoritmo si basa su tre fasi principali:

1. **slow start**;
2. **congestion avoidance**;
3. **fast recovery**.

Slow start

Nel momento in cui la connessione TCP è stabilita, il valore di `cwnd` è impostato a 1 MSS (quindi il tasso di trasmissione è inizialmente pari a $\frac{\text{MSS}}{\text{RTT}}$). Se vi è più banda disponibile, il sender vorrebbe scoprirla (si dice che il sender fa un **probe**, ossia vuole

sondare la velocità di trasmissione), pertanto il valore di `cwnd` **è aumentato di** `1` **ogni volta che per un segmento trasmesso si riceve un ACK**. Quindi inizialmente si invia un segmento, e se questo riceve un ACK, `cwnd = cwnd + MSS`. Successivamente si inviano due segmenti, e se per ciascuno si riceve un ACK, `cwnd` è raddoppiato. Notiamo quindi che ad ogni passo si raddoppia la quantità `cwnd`, quindi sebbene sia uno "slow start", il valore `cwnd` aumenta in maniera **esponenziale** quando tutti i pacchetti sono trasmessi correttamente.
L'aumento esponenziale termina nel caso di perdita di pacchetti (id est timeout a causa di congestione). In tale evento `cwnd` viene impostato nuovamente a 1 MSS, e una seconda variabile, la `ssthresh` (slow start threshold, soglia di slow start), viene impostata a metà del valore della finestra di congestione quando questa è rilevata. Dopo che il valore `ssthresh` è impostato, l'aumento esponenziale terminerà anche quando $\text{cwnd} \geq \text{ssthresh}$. Infine, nel caso di triplo ACK duplicato, TCP opera una fast retransmission e va in **fast recovery**.

Congestion avoidance

Al fine di evitare la congestione, in fase di **congestion avoidance** il sender incrementa il valore di `cwnd` di 1 MSS per RTT, invece di raddoppiarlo. L'aumento **lineare** termina quando si verifica un timeout o un triplo ACK duplicato. Nel primo caso, il valore di `cwnd` è impostato a 1 MSS e `ssthresh = cwnd/2`; nel secondo caso, `cwnd` viene dimezzato e `ssthresh` è sempre pari a `cwnd/2` al momento del triplo ACK.

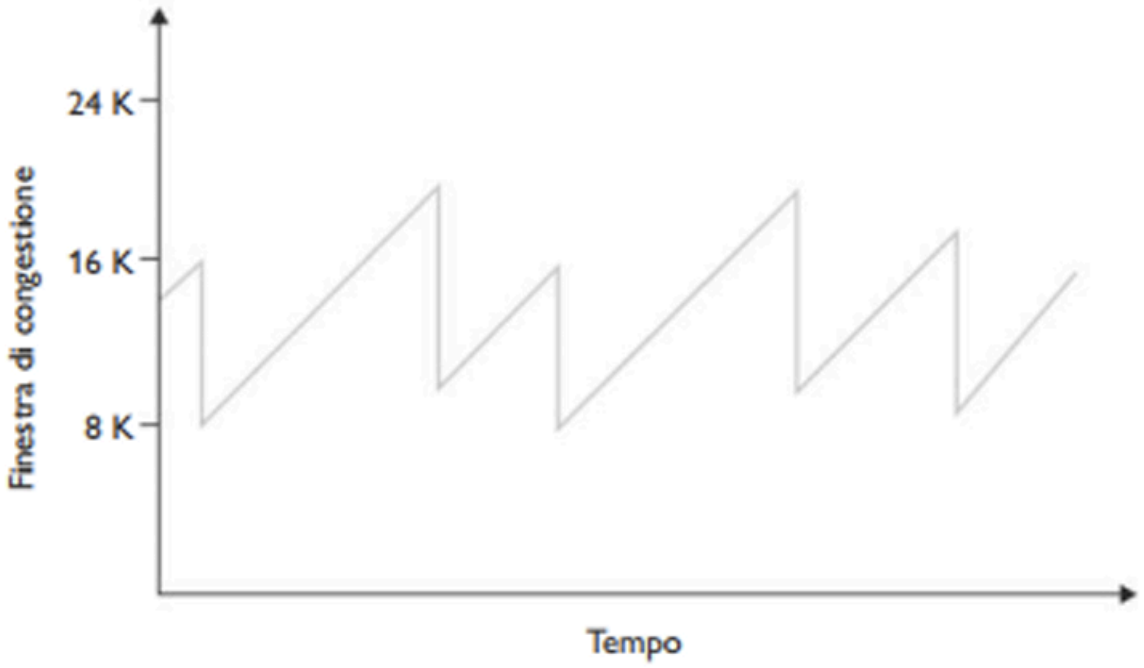
Fast recovery

Nel caso di triplo ACK si passa dallo stato di congestion avoidance allo stato di **fast recovery**, in cui `cwnd` viene aumentato di 1 MSS per ogni ACK duplicato del segmento perso. Pertanto, `cwnd` alla fine sarà uguale a $\frac{\text{cwnd}}{2} + 3 \cdot \text{MSS}$. Se arriva un ACK per il segmento perso, TCP entra in congestion avoidance, mentre se si verifica un timeout si passa a slow start impostando `cwnd` a 1 e `ssthresh` a metà di `cwnd`.
Da precisare che fast recovery **non è un componente obbligatorio**. Infatti esistono due versioni di TCP in virtù del controllo di congestione:

- **TCP Tahoe**, che non implementa fast recovery e a prescindere da qualsiasi evento di perdita imposta `cwnd` a 1 MSS;
- **TCP Reno**, che adotta fast recovery.

Osservazioni

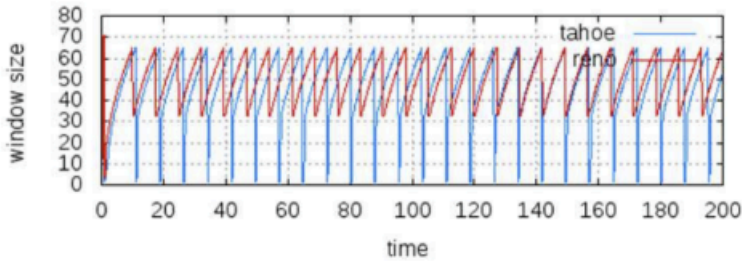
Il controllo tramite AIMD porta ad un comportamento noto come **saw tooth**, ossia "a dente di sega", in quanto guardando i grafici sulla rilevazione di larghezza di banda si ha un aumento lineare e una diminuzione moltiplicativa, più precisamente con un fattore 2 nel caso di TCP Reno, come mostra il grafico a seguire:

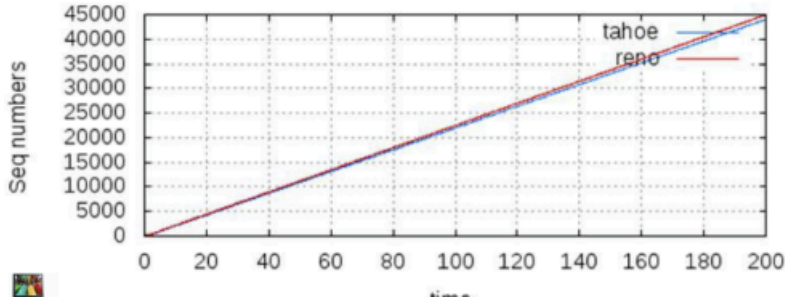


Inoltre, non possiamo dire semplicemente che TCP Tahoe è peggiore di Reno o viceversa, ma tutto dipende dalle caratteristiche del canale di trasmissione, quali:

- quantità di pacchetti persi;
- larghezza di banda;
- delay.

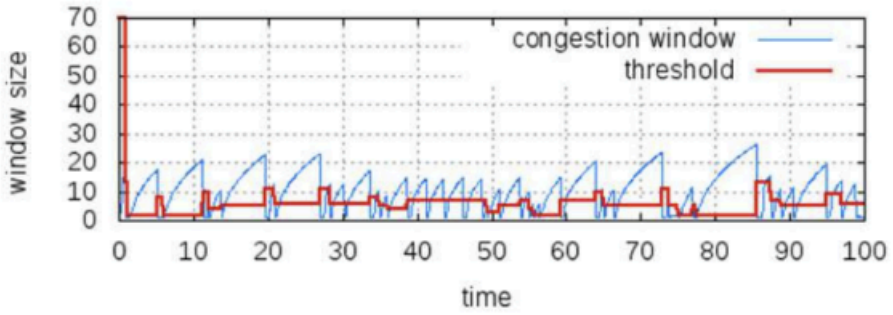
Ad esempio, se supponiamo di avere un canale lossless con un delay costante, si osserva una situazione simile:

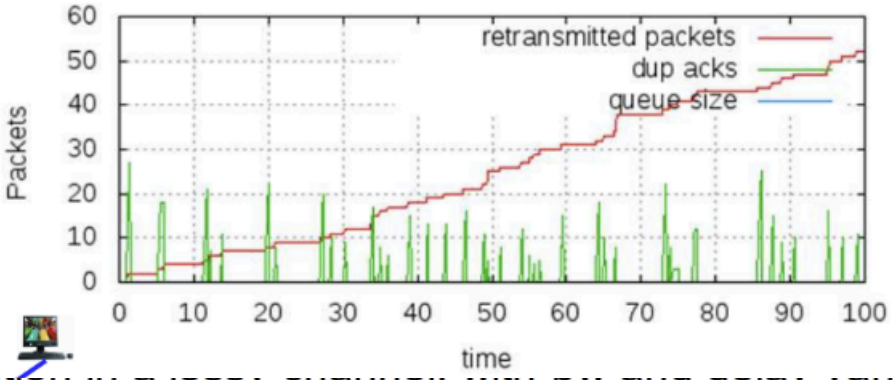




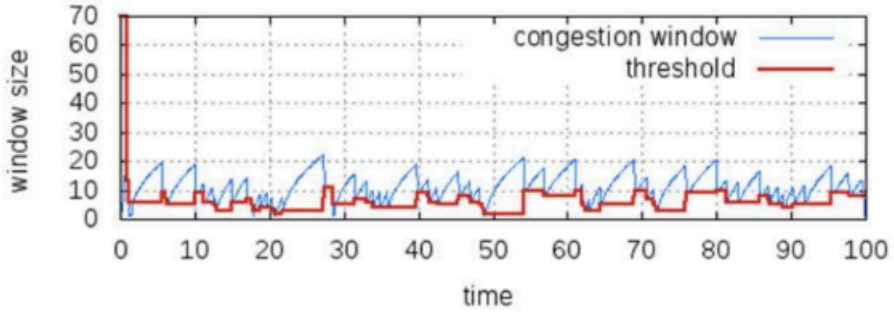
Si può vedere che Reno è leggermente superiore rispetto a Tahoe, specialmente per il fatto che la window size viene dimezzata invece di essere impostata a 1 MSS. Tuttavia, se abbiamo una larghezza di banda e un delay variabili, le cose cambiano:

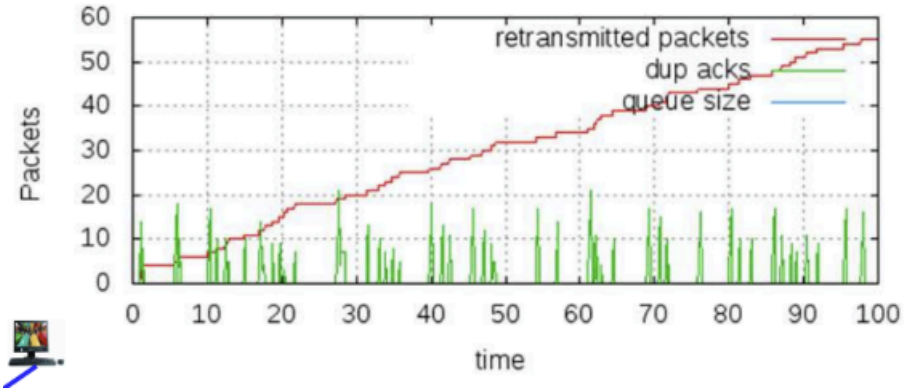
TCP Tahoe





TCP Reno





La prima immagine mostra le performance di TCP Tahoe, la seconda TCP Reno. Il primo grafico mostra la window size e la threshold, il secondo attenziona la quantità di pacchetti ritrasmessi. Anche se Reno sia superiore a Tahoe in virtù di numero di pacchetti trasmessi, la quantità di pacchetti ritrasmessi da parte di Reno è superiore. Tutto quindi dipende dalle caratteristiche del canale.

08 - ECN, Fairness

ECN

Oltre al controllo di congestione end-to-end esiste un approccio alternativo che permette alla rete di segnalare esplicitamente a sender e receiver una congestione: la **ECN** (**Explicit Congestion Notification**) Si tratta di un controllo di congestione **assistito dalla rete**.
Se un router è congestionato, può impostare a 1 due bit presenti nel campo "Type of Service" presente all'interno dell'header del datagramma IP e invia il pacchetto contrassegnato al receiver. Il receiver informera il mittente impostando a 1 il bit ECN Echo del segmento ACK. Il bit ECN viene anche usato dal sender per segnalare che sia sender sia receiver possono utilizzare il meccanismo di ECN.
I bit utilizzati da ECN nell'header IP permettono quattro configurazioni possibili:

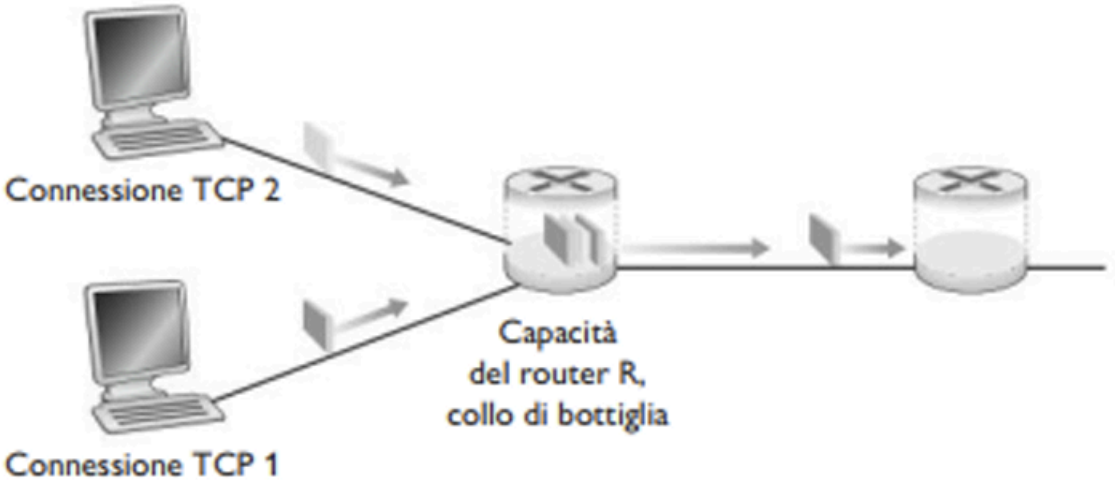
- 00 significa "Non-ECN capable Transport";
- 01 ECN capable transport: l'host supporta l'ECN;
- 10 indica che il source supporta l'ECN;
- 11 indica che c'è un problema di congestione.

Chiaramente la congestione non riguarda un solo sender, ma tutti i sender che attraversano il router che ha impostato a 1 entrambi i bit del ToS.

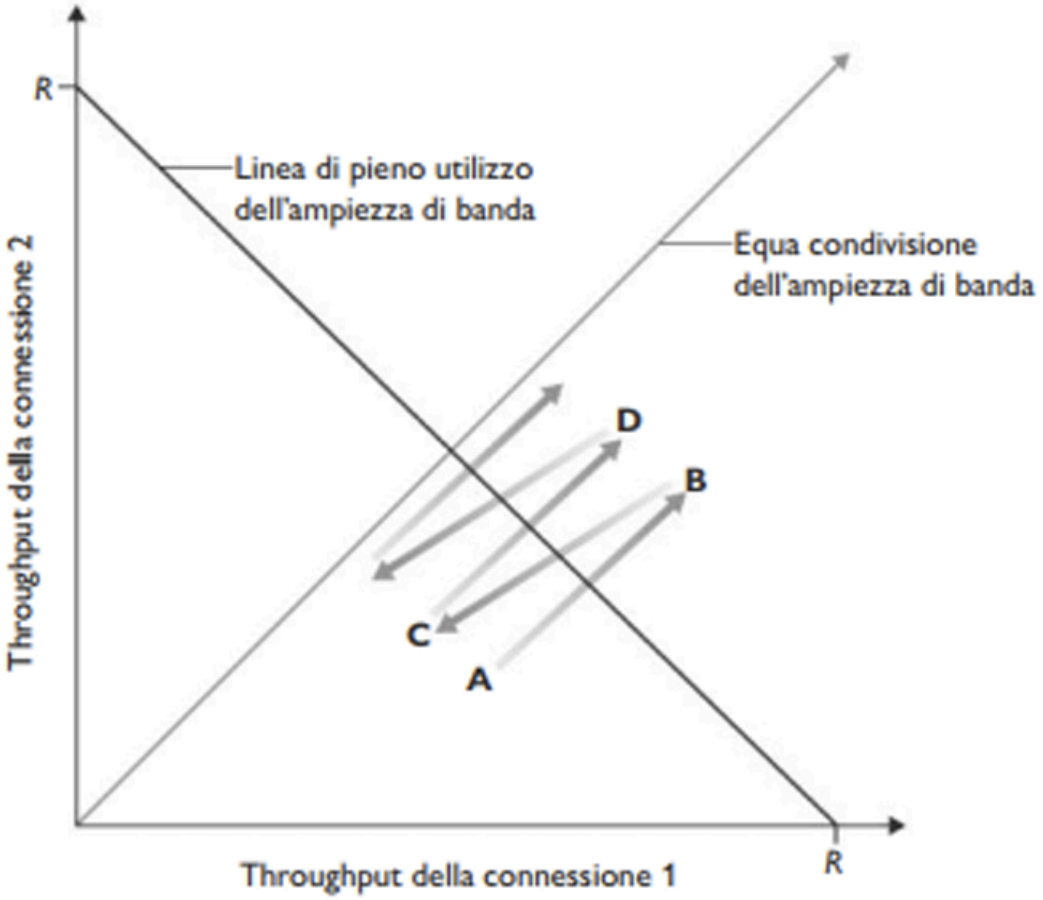
TCP Fairness

Vorremmo che il meccanismo TCP sia tale che date K connessioni TCP che passano attraverso un collegamento con capacità trasmissiva pari a R, si ha che **ogni connessione ottenga la stessa porzione di banda del collegamento**. In altre parole, la velocità media di ogni connessione dev'essere $\frac{R}{K}$.
Potremmo chiederci allora se il controllo AIMD sia un algoritmo fair. La risposta chiaramente deve tener conto del fatto che le connessioni possono avere diverse dimensioni di finestra. Tuttavia, è possibile mostrare in maniera intuitiva il perché il controllo di congestione di TCP riesca a offrire la stessa porzione di banda a connessioni in competizione.

Supponiamo di avere la seguente situazione:



Abbiamo due connessioni che utilizzano lo stesso router con capacità R. Assumiamo che si abbiano gli stessi valori di MSS e RTT, che non vi siano altre connesssioni che attraversano il collegamento e che le connessioni operino in congestion avoidance.
Sapendo che ciascuna connessione può avere un throughput massimo pari ad R, aumentare il throughput di una connessione significa diminuire quello delle altre connessioni. Con due connessioni, dovremmo avere che entrambe le connessioni hanno $\frac{R}{2}$ di throughput per una perfetta situazione di equilibrio. Tracciamo allora un grafico, in cui sugli assi x, y tracciamo i throughput delle connessioni:



Vorremmo allora che il throughput cada sulla bisettricee del primo quadrante, in modo tale che entrambe le connessioni condividano la stessa capacità. Inoltre, vorremmo che tale punto sia l'intersezione tra la bisettrice e la curva che esprime il limite di utilizzo della banda.
Supponiamo che i throughput di entrambe le connessioni corrispondano al punto A. Poiché siamo sotto la curva, le connessioni aumentano `cwnd` di 1 MSS per RTT (ricordiamo che siamo in congestion avoidance). Il throughput "avanza" sulla semiretta parallela alla bisettrice del primo quadrante, fino a quando non si verifica una perdita di pacchetti, causata da un triplo ACK. Quando si ha una perdita di pacchetti, le connessioni dimezzano `cwnd`. Successivamente procedono. Procedendo in questo modo, il punto P che esprime i valori di throughput delle connessioni si avvicina sempre di più alla bisettrice del primo quadrante. Quando il punto si trova sulla bisettrice, si ha che P è di coordinate $x = y$, quindi se x diventa $\frac{x}{2}$, segue che $y = \frac{x}{2}$ e siamo sempre sulla bisettrice.

Nella pratica è impossibile avere connessioni con lo stesso RTT, e in particolare si può notare che le connessioni con RTT inferiore sono in grado di acquisire più rapidamente larghezza di banda su un particolare collegamento non appena la banda si libera. Le connessioni con RTT più alto quindi avranno un throughput inferiore.

09 - Network Layer

Network Layer

Il network layer si occupa principalmente di **instradare** e **inoltrare** i pacchetti incapsulati dal livello transport, i **datagrammi**. "Instradare" e "inoltrare" non sono due sinonimi:

- instradare (**routing**) è il procedimento secondo cui i pacchetti seguono un determinato percorso per partire dall'host sender e arrivare all'host receiver. Per fare questo trasferimento, possiamo seguire più percorsi diversi, ognuno con una certa efficienza, latenza e bandwidth. L'efficienza di ogni percorso dipende da quante sottoreti vengono attraversate, e da come sono strutturate anche fisicamente queste sottoreti. Il problema del routing è un problema complesso;
- inoltrare (**forwarding**) indica l'effettivo **smistamento** dei pacchetti. Con il termine forwarding quindi si fa riferimento all'azione locale con cui il router trasferisce pacchetti da un'itnerfaccia di ingresso a quella di uscita.

Ogni dispostivo è dotato di un **indirizzo MAC**, un identificativo esclusivo del dispositivo. Sopra questo indirizzo è definito un indirizzo logico, l'**indirizzo IP**, ed identifica in maniera univoca un nodo nella rete.

Centrale all'interno del network layer è il **router**, un commutatore di pacchetto. Il router prende un pacchetto da uno dei suoi collegamenti in ingresso e lo ritrasmette su uno di quelli in uscita.

Data plane e control plane

Dividiamo le funzionalità del router su due piani: il **data plane** e il **control plane**.
Il **data plane** determina come il datagramma che arriva nella porta di input del router viene inoltrato alla porta di output. Potrebbe sembrare che non abbia alcuna importanza lo switching, ma quando il numero di pacchetti aumenta significativamente, è importante fare un forwarding dei pacchetti bilanciato, in quanto potremmo trovare alcune porte sovraccaricate e altre che hanno molta banda disponibile. La scelta dell'interfaccia di output (quindi della porta di output), deve tener conto del **control plane**, responsabile delle decisioni di instradamento e della logica **network-wide**.
Per inoltrare i pacchetti, i router utilizzano una **tabella di inoltro** (**forwarding table**). Quando si riceve un pacchetto, il router estrae dall'header dei valori che vengono usati come indice della forwarding table. Il risultato indica a quale **interfaccia di uscita** indirizzare il pacchetto. L'inoltro del pacchetto ad una determinata interfaccia, e quindi i campi della forwarding table, è gestito nel piano della **control plane**. La control plane è la logica network-wide che **determina come un datagramma viene instradato tra i vari router**. Per configurare le tabelle di inoltro, esistono due approcci: la **per-router control plane** e la **SDN control plane**

Per-router control plane

Il control plane calcola le forwarding table utilizzando degli algoritmi di routing, mentre il data plane usa la forwarding table per inoltrare pacchetti in tempo reale. I router tra loro scambiano informazioni costruendo una **visione della rete**: ogni router esegue un algoritmo di routing, che genera una forwarding table locale per quel router in base alla topologia di rete. Gli algoritmi più utilizzati sono gli algoritmi di Dijkstra e di Bellman-Ford.

Le forwarding table vengono di volta in volta aggiornate in virtù delle visioni di rete degli altri router. In particolare, i router si scambiano messaggi di un protocollo di instradamento.

Software-Defined Networking (SDN) control plane

Il control plane in questo caso è centralizzato in un **controllore remoto**, separato fisicamente dai data plane che resta nei dispositivi di rete. All'interno del router il firmware (**control agent**) riceve istruzioni dal remote controller, aggiornando la forwarding table locale. Il Remote controller è un controllore che conosce l'intera topologia di rete, prendendo decisioni su quali percorsi devono seguire i pacchetti e su come aggiornare le forwarding table.
La costruzione della forward table dev'essere più rapida rispetto al forwarding dei pacchetti per garantire che non vi siano ritardi nella rete. Notiamo inoltre che **il control agent del router non ha autonomia**, ma segue le istruzioni che riceve dal controllore. La rete quindi si dice **software-defined**, perché è il remote controller che calcola tutte le tabelle di inoltro.

Servizi a livello IP

Abbiamo sempre un modulo di servizio offerto dal livello di rete, che si concentra sulle tipologie di servizi forniti ai datagrammi che viaggiano in rete. Ad esempio, per singoli datagrammi, i modelli di servizio garantiscono:

- **consegna garantita**: il pacchetto è garantito ad arrivare a destinazione. Non è sempre disponibile nelle reti reali come internet, ma è possibile garantire ciò nelle sottoreti;

- **consegna garantita con meno di 40 millisecondi**: il pacchetto non solo arriva, ma arriva entro un certo tempo massimo, ossia 40 millisecondi.

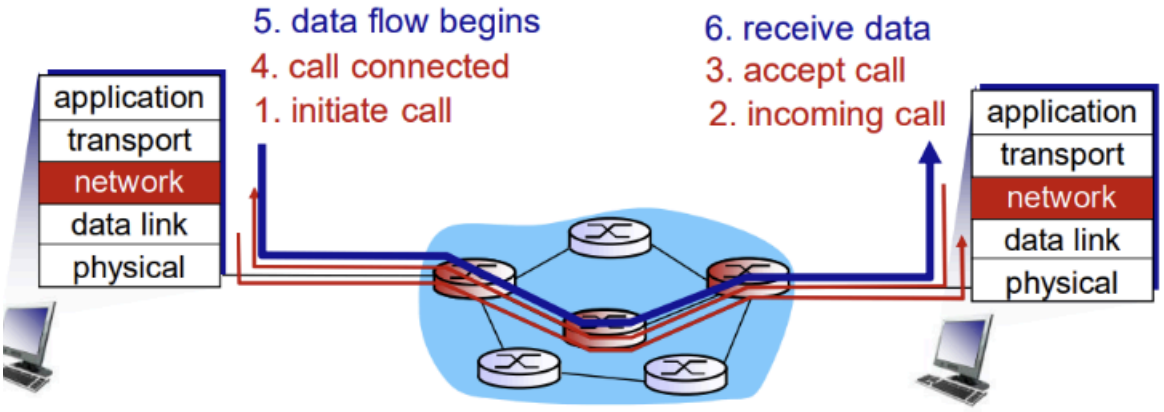
Sono previsti nei protocolli network anche servizi per un **flusso** di datagrammi:

- **garanzia di consegna in ordine**: i datagrammi arrivano **nell'ordine corretto**, ossia come sono stati inviati;
- **banda di rete minima garantita**: la rete garantisce una larghezza di banda minima. Questo garantisce che il canale ha un certo throughput, fornendo un lower bound;
- **garanzia sul jitter massimo**: il **jitter** è la variazione di ritardo tra pacchetti. Garantisce quindi che la variazione di ritardo sia sotto un certo limite, utile per applicazioni sensibili al tempo.

Su internet si utilizza il modello **best effort**. Nel modello best-effort non si hanno controlli sulla banda o sulla congestione, ma si prende per dato che qualcuno controlli il flusso. L'ipotesi che si fa quindi è che la rete sia organizzata e si cerca di fare il meglio per instradare i pacchetti. Ma non c'è nessuno che controlla realmente che ciò avvenga: tutto questo viene demandato al livello di trasposrto e al livello di applicazione. Non sarà quindi il network in prima istanza o in maniera efficiente ad accorgersi che i pacchetti vengono persi, ma il transport, con i servizi ad esempio forniti da TCP.

Virtual circuit

In alternativa al servizio basato su datagrammi si può usare il servizio di **Virtual Circuit**. Il servizio Virtual Circuit è un servizio orientato alla connesione, in cui la modalità di trasferimento avviene tramite un circuito fisso dedicato tra due nodi.
Un host vuole iniziare una comunicazione, richiedendo di **inizializzare una call**. Il messaggio viene raggiunto dal destinatario, che viene notificato della chiamata in arrivo (**incoming call**). Successivamente, il destinatario **accetta la chiamata**, e la rete conferma che la rete è stabilita con il mittente. A questo punto viene stabilito un flusso di dati dal mittente al destinatario.



Differenze tra Datagram e Virtual circuit

Ogni datagramma è indipendente nel modello a datagrammi, mentre nei virtual circuit viene stabilita una connessione.
Nel modello di servizio a datagrammi i router sono stateless, quindi le informazioni relative all'IP address è memorizzata in ogni pacchetto. Nel virtual circuit, i router memorizzano lo stato per ogni connessione.
Nel modello a datagrammi i pacchetti seguono un percorso diverso, quindi non si ha informazioni sul routing. Nel virtual circuit, ogni circuito dev'essere identificato, e si ha un routing predefinito per ogni pacchetto.
Non si hanno effetti se un router fallisce nel modello a datagrammi, perché si può avere un nuovo routing tramite router vicini, mentre per il virtual circuit bisogna ricreare il virtual circuit se un router del percorso fallisce.
Infine i controlli di congestione sono più semplici per il virtual circuit.

Network-layer service model

Internet utilizza il modello di servizio best-effort. Nelle reti ATM invece si utilizza il modello **Constant Bit Rate**.
ATM è una tecnologia di rete a commutazione di celle, usato in reti aziendali e telecomunicazioni. Divide i dati in celle di 53 byte e usa un modello a circuito virtuali, creando prima un percorso e poi trasmettendo i dati. Permette di garantire un bit-rate costante, garantendo che non vi siano perdite di pacchetti, che vi sia ordine di trasmissione di questi e timing.
Esiste anche la ATM che utilizza un servizio di **Available Bit Rate**, garantendo una bandwidth minima ma utilizzando la banda di rete disponibile. Non si hanno garanzie di timing e perdite di pacchetti, ma garantisce ordine di trasmisisone.
Esistono anche altri due modelli di servizi per internet:

- **Intserv Guaranteed**: prenota risorse e, ogni volta che sarà necessario istanziarle, causerà prelazione. Può essere esteso su internet, ma è molto costoso;
- **Diffserv**: tratta i pacchetti in modo differenziato utilizzando dei meccanismi di priorità. Il Diffserv viene usato molto poco.

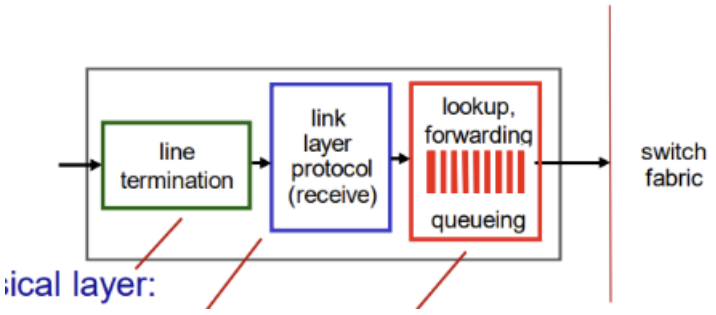
In internet viene utilizzato il best-effort per alcuni motivi:

- **semplicità**: il meccanismo è molto semplice da implementare, non si hanno prenotazione di risorse e nessuna gestione dello stato nei router. Questo ha reso internet scalabile e facile da far crescere;
- **provisioning of bandwidth**: le reti moderne hanno molta larghezza di banda, e non serve QoS se la rete è abbastanza veloce per tutti. Con abbastanza banda, la mancanza di garanzie è meno problematica;
- **servizi distribuiti del layer di applicazione replicati**: molte funzioni di affidabilità, gestione di ordine, ritrasmissione e altro vengono spostate al livello applicativo o al livello di trasposrto. Le applicazioni quindi si adattano alla rete, e non è la rete ad adattarsi all'applicazione.

Router

Un router è costituito da:

- un **routing processor**, che può essere del router stesso o dell'host. Si occupa di calcolare le tabelle di routing e della gestione di configurazione, policy, rete, ecc. Si occupa inoltre di instradare il traffico di dati, **eseguendo le funzioni del piano di controllo**. **Viene implementato via software**;
- un **high-speed switching fabric** (struttura di commutazione), che è un forwarder che inoltra i pacchetti reali da un'interfaccia (o porta) ad un'altra. Si occupa quindi di connettere fisicamente le porte in ingresso a quelle di uscita;
- **porte di input**, o **interfacce di ingresso**. Svolgono diverse funzioni:
  - a livello fisico, svolgono la funzione di terminazione di un collegamento in ingresso al router;
  - a livello di collegamento svolgono funzioni necessarie per operare nel livello di network;
  - a livello di rete, svolgono la funzione di ricerca e di inoltro (**lookup and forwarding**) in modo tale che il pacchetto inoltrato nella struttura di commutazione esca sulla porta corretta. Le porte utilizzano le informazioni relative ai campi dell'header e utilizzano la forwarding table nella memoria dell'interfaccia di input;



- **porte di output**, o **interfacce di uscita**. Memorizzano i pacchetti per trasferirli sul collegamento in uscita.

Quasi tutte le componenti sono implementate via **hardware**, per ovvi motivi: basta pensare al fatto che se si ha un collegamento a 100 Gbps e un datagramma di 64 byte, la porta di input ha solo 5.12 ns per elaborare il datagramma. Un'implementazione software non può performare con queste tempistiche. Le funzioni del piano di controllo, invece, sono implementate via software, in quanto operano sulla scala dei millisecondi.
I pacchetti che arrivano giungono tramite la linea di terminazione, appartenente al livello fisico. I pacchetti giunti arrivano al protocollo del link layer, fornendo informazioni relative al protocollo di rete utilizzato. Verificate e rimosse le informazioni, arrivano i datagrammi, che devono essere inoltrati alle porte di uscita. I datagrammi arrivano ad una certa frequenza, quindi può essere necessario inserire in un **buffer** i datagrammi che giungono. La decisione di forwarding si basa su due approcci:

- **destination-based forwarding**: il forwarding si basa solo sull'indirizzo IP di destinazione;
- **generalized forwarding**: il forwarding dipende da diversi campi dell'header. In questo caso si considerano anche QoS, protocollo di trasporto, etichette e priorità.

Destination-based forwarding

La porta di ingresso utilizza le informazioni della tabella di inoltro per determinare la porta di uscita a cui dirigere il pacchetto tramite la switching fabric. Una copia della forwarding table del router è memorizzata su ogni porta di ingresso: in questo modo il routing processor non viene invocato alla ricezione di ogni pacchetto in ogni porta.
Con il destination-based forwarding l'inoltro è basato esclusivamente sull'indirizzo IP del destinatario. In questo caso la forwarding

table è costituita da due colonne: destination address range e link interface. La prima colonna definisce un intervallo di indirizzi IP, la seconda invece definisce l'interfaccia verso la quale i datagrammi vengono inoltrati.

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

Notiamo che in questo esempio non è necessario neanche definire dei range, ma semplicemente anche dei **prefissi**, e individuare l'interfaccia di destinazione facendo un confronto tra prefissi:

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

Per inoltrare i pacchetti si utilizza un algoritmo **first-match**: si confronta l'indirizzo IP del segmento ricevuto e lo si confronta con gli indirizzi IP della tabella di inoltro. Al primo match, viene inoltrato il datagramma all'interfaccia corrispondente al range.
Chiaramente se definiamo una forwarding table tale che è possibile associare ad uno stesso indirizzo più di un intervallo, la first-match si ferma, come suggerisce il nome, al primo accoppiamento, che potrebbe individuare anche un **range più ampio**, quindi meno "preciso" (intendendo con "preciso" il fatto che sia poco ampio: più è preciso, meno è ampio, pertanto è meno appesantito).

Il first-match ha un impatto, quindi, sulla congestione delle interfacce di output. Si può evitare questo problema utilizzando il **longest prefix match**: quando si cerca la entry nella forwarding table, si cerca il più lungo prefisso per l'indirizzo che fa matching con l'indirizzo di destinazione. In questo modo si cerca un match ottimale, cercando un maggior numero di bit coincidenti nel prefisso.
Il longest prefix matching non può essere tuttavia effettuato tramite una ricerca lineare su grandi tabelle, ma necessita di alcuni algoritmi di ricerca più veloci. Inoltre, bisogna porre attenzione anche ai tempi di accesso alla memoria. Spesso per la ricerca si utilizzano le **TCAM** (**Ternary Content Address Memories**).

Content Address Memory

Una CAM è un tipo speciale di memroia utilizzata principalmente per applicazioni in cui è necessario eseguire ricerche ad alte velocità. Si tratta di una memoria **associativa**, in cui si confronta la chiave di ricerca con i contenuti memorizzati in memoria; il risultato della ricerca è l'indirizzo di memoria del match.
A differenza della RAM, che utilizza celle semplici per memorizzare i bit, la CAM necessita di un cpmponente aggiuntivo, ossia di un **circuito di confronto** per determinare il match tra il bit chiave e il bit memorizzato. Gli output di ogni match devono essere combinati per restituire un segnale di match completo. Ne segue quindi che poiché ad ogni ciclo di clock si attiva il meccanismo di confronto, la CAM è più costosa e ha un maggior consumo energetico. Tuttavia, il vantaggio risiede nel fatto che **eseguire una ricerca e ottenere un matching richiede un solo ciclo di clock**.
Una **TCAM** (**Ternary CAM**) aggiunge anche la possibilità di utilizzare una **wildcard** "don't care", che si indica con X. Ad esempio, la parola $10XX0$ in una TCAM permette il matching di $10000, 10010, 10100, 10110$. Chiaramente si aggiunge anche il costo di aggiungere ad ogni cella il terzo stato "don't care".

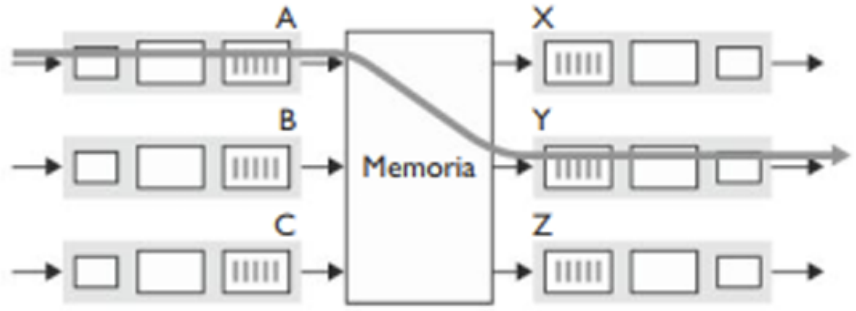
Switching fabric

La switching fabric è uno smistatore di pacchetti. Esistono diversi tipi di switching fabric:

- in memoria;
- via bus;
- con interconnessione.

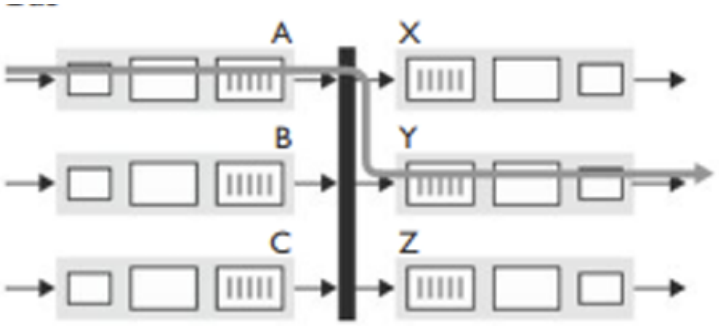
Switching via memoria

Si tratta di uno switching sotto il controllo diretto della CPU. Quando un pacchetto giunge da una porta di ingresso, viene generato un interrupt e il pacchetto viene copiato nella memoria di sistema. Si estrae così l'indirizzo di destinazione e una volta determinata la porta di uscita e si copiava il pacchetto nel buffer di questa. Il sistema è molto smeplice, ma chiaramente ha alcune limitazioni, tra cui l'**impossibilità di inoltrare più di un pacchetto per volta**: infatti solo un'operazione di scrittura o lettura può essere effettuata ogni volta, dato che dev'essere utilizzato il bus di sistema.



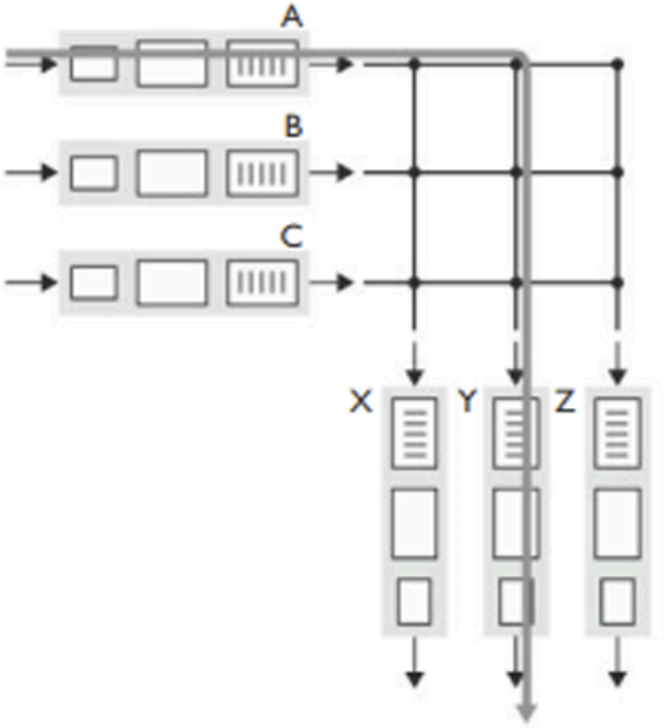
Switching via bus

In uno switching via bus viene utilizzato un bus condiviso senza l'intervento del processore di instradamento. Semplicemente, le porte di ingresso trasferiscono il pacchetto direttamente alle porte di uscita aggiungendo un'etichetta di commutazione al pacchetto. Tale etichetta indica la porta di uscita: ogni porta di uscita riceverà il pacchetto, ma solo la porta corrsipondente all'etichetta lo accetterà. L'etichetta verrà poi rimossa.
Anche in questo caso il sistema è limitato dal bus, sia a livello di larghezza di banda della commutazione che a livello di pacchetti che attraversano il router: soltanto un pacchetto per volta infatti può attraversare il bus.

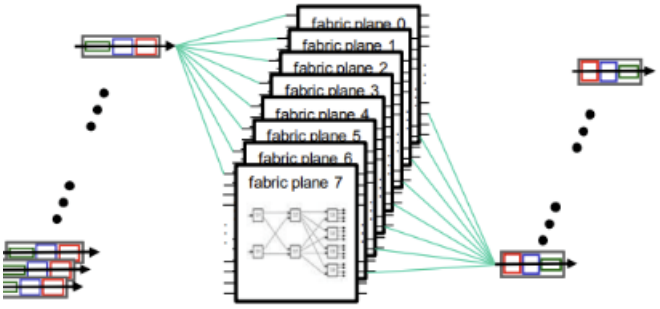


Switching a interconnessione

Per superare la limitazione dell'utilizzo di un singolo bus, si può utilizzare una **rete di interconnessione**, tipica ad esempio delle architetture multicore. Si utilizza un **crossbar switch**, ossia una **matrice di commutazione**, che utilizza, se vi sono n porte di ingresso e n porte di uscita, $2 \cdot n$ bus, divisi a metà tra verticali e orizzontali. Ogni punto di intersezione può essere aperto o chiuso dal controller della struttura di commutazione. Questo permette di poter inoltrare pacchetti che provengono da porte di input diverse a porte di output diverse in maniera contemporanea. Tuttavia, se due porte di input inoltrano un pacchetto alla stessa porta di output, una delle due porte deve attendere che il primo trasferimento sia completato.

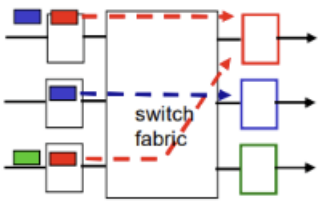


Alcune reti di interconnessione riescono ad aggirare il problema di trasmettere più pacchetti attraverso la stessa porta di output contemporaneamente. In particolare, si utilizza una commutazione non-blocking a tre stadi. Inoltre, la capacità di commutazione può scalare sfruttando il **parallelismo**. Essenzialmente si utilizzano N strutture di commutazione in parallelo, e la porta di ingresso spezza il pacchetto in K **chunk** più piccoli, inviandoli (si dice che fa uno **spray** dei chunk) attraverso K delle N strutture alla porta di uscita selezionata, come segue:

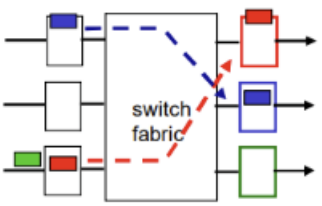


Port queueing

Se la switching fabric è più lenta rispetto alle porte di input combinate, è necessario mettere in coda i datagrammi in input. Il meccanismo di queueing può causare delay a causa dell'**Head of Line Blocking**: un pacchetto in coda di ingresso deve attendere il trasferimento attraverso la struttura perché un pacchetto che lo precede lo blocca. Questo lo notiamo nell'esempio sottostante, dove il pacchetto colorato in verde deve attendere che il pacchetto rosso venga inviato alla porta rossa, **nonostante non ci sia contesa nella porta di uscita verde**:



output port contention: only one red datagram can be transferred. lower red packet is *blocked*



one packet time later: green packet experiences HOL blocking

Quando i datagrammi arrivano dalla switching fabric alle porte di output più rapidamente del tasso di trasmissione, è necessario anche nel caso delle porte di output accodare i pacchetti. Questo accade anche quando la switching fabric è molto rapida rispetto alla velocità di linea delle porte, perché tutto dipende dal tempo di trasmissione del pacchetto. Bisogna quindi usare un **buffer** per

memorizzare i pacchetti. E se non ci sono buffer disponibili, bisogna decidere quali datagrammi buttare. Risulta necessario introdurre una **politica di buffering**.
Il problema per ora riguarda la dimensione del buffer:

- un buffer molto grande permette di conservare più datagrammi, ma aumenta la latenza;
- un buffer più piccolo non permette di memorizzare molti datagrammi ma causa meno latenza.

RFC 3439 suggerisce che il buffer medio dev'essere calcolato utilizzando la formula:

$$\frac{RTT \cdot C}{\sqrt{N}}$$

dove con C si intende la capacità del collegamento e con N il numero di flussi.
Rimane da gestire la politica di buffering:

- se il buffer è pieno, possiamo fare un **drop dei pacchetti**. Abbiamo due soluzioni:
  - il **tail drop**: scarta i pacchetti in arrivo quando la coda è piena senza tener conto di priorità o tipi di traffico. Può causare tuttavia burst di perdite, facendo collassare TCP: il protocollo di trasporto chiederà infatti ritrasmissioni di stessi pacchetti. Droppare pacchetti in questo modo non è efficiente;
  - la **priority queueing**: il buffer è diviso in più **code**, ognuna con priorità diversa. I pacchetti vengono etichettati con un livello di priorità, oppure il sistema di priorità può essere implementato all'interno del router.
- fare un **marking** dei pacchetti in modo tale da segnalare al sender la presenza di poco spazio nel buffer e quindi di ridurre il ritmo di invio. Questo può essere fatto utilizzando i bit ToS di IPv4 (traffic class in IPv6).

Bufferizzati i pacchetti, la porta di output deve trasmetterli. Dobbiamo quindi fare uno **scheduling** dei pacchetti, e abbiamo quattro approcci:

- **FCFS**: opera con metodologia FIFO, senza utilizzare meccanismi di priorità e trasmettendo i pacchetti nell'ordine di arrivo. Tuttavia può essere negativo se esistono pacchetti importanti che vengono inseriti in coda e non vengono inviati;
- **priority scheduling**: il traffico in arrivo viene classificato e messo in coda in base alla priorità. Si inoltrano poi i pacchetti alla base della priorità, svuotando prima la coda con priorità più alta e poi quelle con priorità più bassa. Questo approccio evita congestioni TCP, ma richiede calcoli in virtù della priorità dei pacchetti;
- **round robin**: anche in questo caso il traffico in arrivo viene classificato, ma per smaltire le code il router ciclamente invia un pacchetto per coda. Si prende, ad esempio, un pacchetto dalla coda 1, poi un pacchetto dalla coda 2 e così via;
- **weighted fair queueing** (**WFQ**): l'accodamento equo ponderato è la strategia che trova attualmente vasto impiego. Anche in qeusto caso si ha un servizio ciclico, ma con WFQ le classi possono ricevere un servizio differenziato. A ciascuna classe è assegnato un peso w, e WFQ garantisce che **i pacchetti che devono essere spediti ricevono una frazione di servizio pari a** $\frac{w}{\sum w}$. In virtù del peso, quindi, si garantisce che una certa classe di priorità abbia sempre una determinata larghezza di banda.

10 - Network Layer, 2

Formazione dei datagrammi

Quando i dati dal transport layer vengono passati al network layer, si ha una fase di incapsulamento, in cui vengono aggiunti degli header per memorizzare le informazioni sul routing, definizioni di control e data plane e la definizione del datagramma. Si ha anche la definizione dell'ICMP, sebbene esso non riguardi direttamente l'IP protocol.

Indirizzo IPv4

L'**indirizzo IP** è un indirizzo logico per identificare un nodo nella rete.
Il primo standard utilizzato per gli indirizzi IP è l'**IPv4**, uno standard di indirizzamento a 32 bit. Caratterizza un codice logico di associazione del nodo nella rete mediante 32 bit.
Inizialmente si poteva formare qualunque tipo di indirizzo per ogni dispositivo; tuttavia, col tempo si cercò di creare delle **sottoclassi** di indirizzi.
Ogni indirizzo IP è costituito da due campi:

- il **net**, che dà informazioni relative a quante sottoreti sono disponibili;
- l'**host**, che dà informazioni sul numero di nodi presenti nella sottorete.

Aumentare i bit assegnati a net aumenta il numero di sottoreti ma diminuisce il numero di host; il viceversa è altrettanto vero. Si possono quindi distinguere alcune classi di indirizzi IP, come mostra la tabella:

Classe	Da	A
A	$1.0.0.0$	$127.255.255.255$
B	$128.0.0.0$	$191.255.255.255$
C	$192.0.0.0$	$223.255.255.255$
D	$224.0.0.0$	$239.255.255.255$
E	$240.0.0.0$	

Le reti di classe A vengono utilizzate nelle grosse aziende, dove è necessario avere poche sottoreti ma un grande numero di host. I primi 8 bit sono utilizzati per identificare la rete.
La classe B è tipica dei servizi di streaming. I primi 16 bit sono utilizzati per identificare la rete.
La classe C viene spesso utilizzata per avere un numero buono di host ma anche un numero di sottoreti, quindi tipica di aziende in cui si hanno divisioni interne in gruppi. I primi 24 bit sono utilizzati per identificare la rete.
La classe D è usata per il **multicast**, ossia una trasmissione uno-a-molti di un pacchetto.

Alcuni indirizzi IP inoltre sono riservati:

- $0.0.0.0$, che fa riferimento all'host stesso che sta navigando. Si tratta di un indirizzo IP che viene assegnato ad un nodo che è stato appena aggiunto alla rete, ma a cui non è stato ancora concesso lo scambio di informazioni. Esiste l'host nella rete ma non è ancora indicizzato;
- $255.255.255.255$, utilizzato per il **broadcasting**, ossia invio di messaggi a tutti gli host che si trovano nella stessa sottorete. Invia messaggi a tutti i dispositivi della rete locale. Questo vale per la sottorete di riferimento;
- $127.0.0.0$ fino a $127.255.255.255$. Sono indirizzi riferiti al **localhost**. Più precisamente $127.0.0.1$ corrisponde al **loopback**;
- $10.0.0.0$ fino a $10.255.255.255$, una lista di indirizzi IP privati;
- $169.254.0.0$ fino a $169.254.255.255$, utilizzati quando vi sono problemi nell'assegnazione degli IP.

Due nodi all'interno della rete devono essere tali da avere indirizzi IP distinti. L'agenzia IANA (Internet Assigned NUmbers Authority) aveva il compito di assegnare gli indirizzi. Inizialmente infatti gli indirizzi IP erano in numero limitato. Nel 1992 si crearono i RIR (Regional Internet Registries) per gestire gli indirizzi e i nomi di dominio assegnati ad ogni area. Infine nel 1998 l'organizzazione no-profit ICANN (Internet Corporation for Assigned Network and Numbers) venne fondata sotto il controllo del dipartimento del commercio americano per coordinare i RIR di tutto il mondo. Attualmente ci sono 5 RIR:

- APNIC, per l'Asia;
- ARIN, per l'America;
- LACNIC, per l'America latina;
- RIPE NCC, per l'Europa;

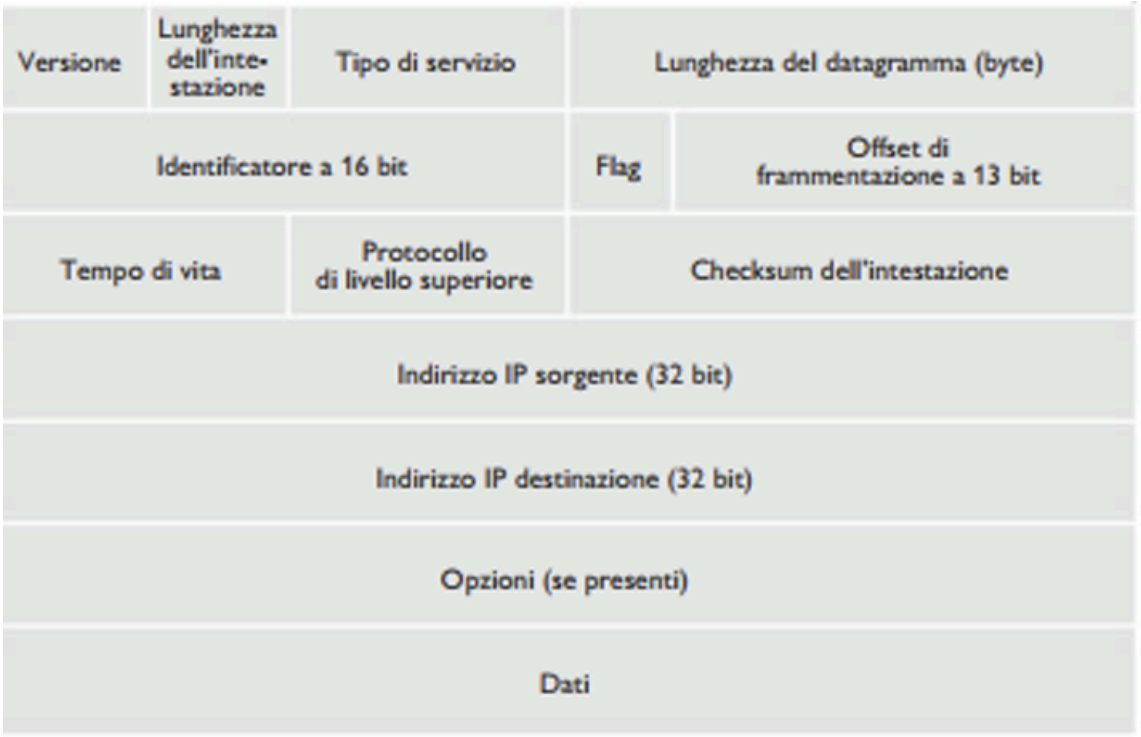
- AFRINIC, per l'Africa.

Ogni ISP ha un set di indirizzi che tipicamente varia (alcuni sono statici, altri dinamici) e che viene fornito dal RIR.

Formato del datagramma IPv4

All'interno dell'incapsulamento del segmento fornito dal transport layer, il network fornisce il suo header con i seguenti campi:

- **IHL**, di quattro bit e che indica la lunghezza dell'header in multipli di 4 byte. Il valore minimo è pari a 5, in quanto l'header ha lunghezza di almeno 20 byte. L'informazione è necessario per capire quando iniziano le **options**, in modo tale da dividere header e dati;
- **versione**, a quattro bit. Indica la versione dello standard utilizzata;
- **ToS**, Type of Service, campo a 8 bit utilizzato per distinguere diversi tipi di datagrammi (basso ritardo, alto throughput, affidabilità);
- **datagram length**, che specifica la lunghezza del datagramma;
- **ID**, identificatore del pacchetto a 16 bit;
- **flags**, 3 bit di flag;
- **TTL**, Time To Live, definito come il numero di hop massimo prima che il pacchetto venga scartato. Il valore di partenza è impostato dal sender e viene diminuito di uno ad ogni hop. Non tutti i pacchetti che vengono trasmessi dal router hanno lo stesso TTL, ma vi sono controlli top-level che aiutano il network a impostare correttamente il valore del campo;
- **protocol**, che specifica il protocollo di trasporto utilizzato (TCP, UDP, ICMP. Quest'ultimo dà informazioni di diagnostica ed è un livello minimale);
- **header checksum**, ossia il checksum dell'header. Si prendono tutte le informazioni dell'header, che vengono mappate con numeri decimali, e si crea il checksum, come l'OR logico di questi valori o come la somma di questi valori. L'operazione matematica eseguita con tutti questi valori numerici permette di ottenere questo valore. Chi riceve il pacchetto applica l'operazione matematica, specificata nel layer di trasporto, e verifica se il risultato è corretto. Se il risultato non combacia, il pacchetto è scartato;
- **options**, che include opzioni variabili per il datagramma.



Frammettazione e riassemblamento dei datagrammi IP

Se l'IP datagram da trasmettere è molto più grande rispetto a quello che il router può processare (ad esempio, l'MTU dell'ethernet è di 1500 byte, se ne voglio inviare 4000 c'è un problema), è necessario **frammentare** il datagramma. I datagrammi vengono poi riassemblati dal receiver.

Per gestire la frammentazione si utilizzano i bit di flag dell'header IPv4. In particolare, ci sono due flag importanti:

- DF, **Don't Fragment**: se impostato ad 1, il datagramma **non** verrà frammentato, e verrà scartato se necessario;
- MF, **More Fragments**: se impostato ad 1, specifica che ci sono ancora frammenti dopo questo. Se impostato a 0, il frammento

inviato è l'ultimo.

Esistono anche 13 bit di offset, che specificano l'inizio del frammento rispetto all'inizio del pacchetto originale. L'offset è misurato a multipli di 8 byte.

Esempio

Supponiamo di dover trasmettere un pacchetto di 4000 byte, con un MTU di 1500 byte massimo. Se il pacchetto da 4000 arriva con il flag DF settato a 1, questo viene scartato.
Impostiamo allora DF a 0, e inviamo il primo segmento con MF a 1 e offset a 0. La lunghezza del segmento inviato è di 1480 byte (20 per gli header). Il secondo frammento ha MF a 1 e offset a 185 (ossia $\frac{1480}{8}$). La lunghezza di questo segmento è sempre di 1480. Per l'ultimo, DF è posto a 0, ma MF è posto a 0. L'offset stavolta è di 370. Gli ID dei pacchetti sono **uguali per ogni frammento**.

Ogni pacchetto inviato non deve corrompersi in nessuno dei campi ID, flag o offset:

- se cambia l'offset, il ricongiungimento dei pezzi non può essere eseguito correttamente;
- se cambia l'ID, i frammenti vengono considerati come frammenti di pacchetti diversi;
- se cambiano le flag, potremmo avere dei frammenti scartati, o considerare il frammento ricevuto come finale se MF passa da 1 a 0.

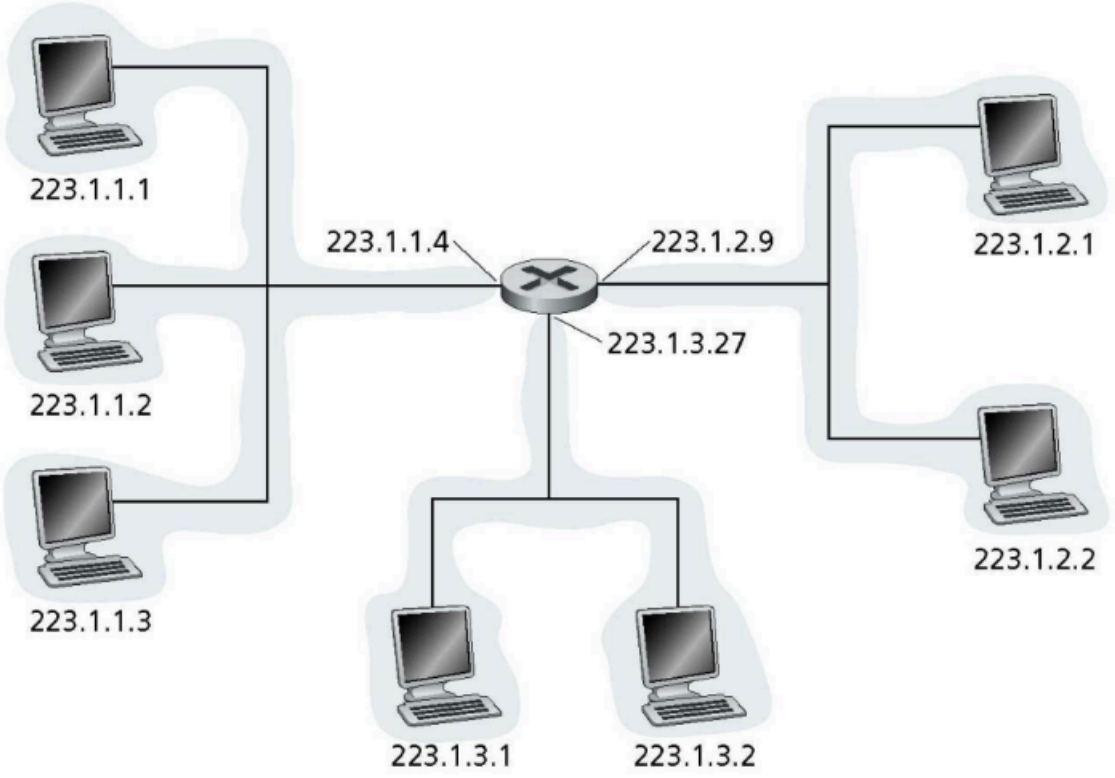
Indirizzamento IPv4

Il problema con l'indirizzamento **classful**, ossia utilizzando le classi A, B, C, D, E per indirizzare i nodi nella rete, risiede nel fatto che le suddivisioni dei bit di rete e dei bit di host sono rigide. Se un'azienda ha bisogno di 32 indirizzi IP ma assegniamo un indirizzo di classe C, l'azienda sprecherà $254 - 32 = 222$ indirizzi. Pertanto si utilizza una suddivisione meno rigida nota come **CIDR** (**Classless Inter-Domain Routing**).

CIDR prevede una notazione con barra per indicare quanti bit sono dedicati alla parte di rete. Ad esempio, $223.1.1.x/24$ è in notazione CIDR e si legge: "la rete inizia da $223.1.1.0$ e ha una maschera di sottorete di 24 bit". Gli host assegnabili vanno dall'IP $223.1.1.1$ a $223.1.1.254$. Il numero totale di host quindi è dato da

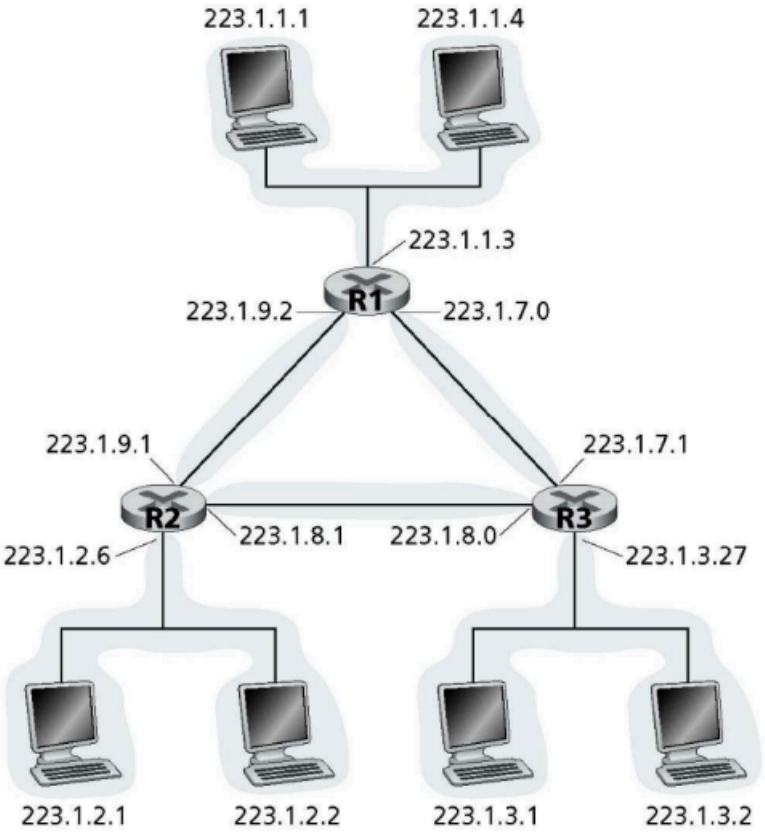
$$2^8 - 2 = 254 \text{ indirizzi IP}$$

Questo perché **ogni indirizzo che termina in 0 identifica la sottorete, mentre in 255 identifica un broadcast per la sottorete** (da non confondere con il broadcast alla sottorete locale, con $255.255.255.255$). Consideriamo la rete:



I dispositivi hanno indirizzo IP di classe C, che ha una **maschera di rete** tipicamente pari a $255.255.255.0$. Gli indirizzi IP sono determinati in virtù della sottorete a cui sono connessi gli host. Quindi, nella sottorete sinistra, gli IP saranno tutti del tipo $223.1.1.X$. Vi è da precisare, inoltre, che tali indirizzi IP non sono assegnati direttamente sugli host, ma sono associati all'**interfaccia**, ossia il confine tra l'host e il collegamento fisico. Ne segue quindi che poiché il router ha più interfacce, **tutte le interfacce devono avere un proprio indirizzo IP**.

Vediamo ora una differente struttura:

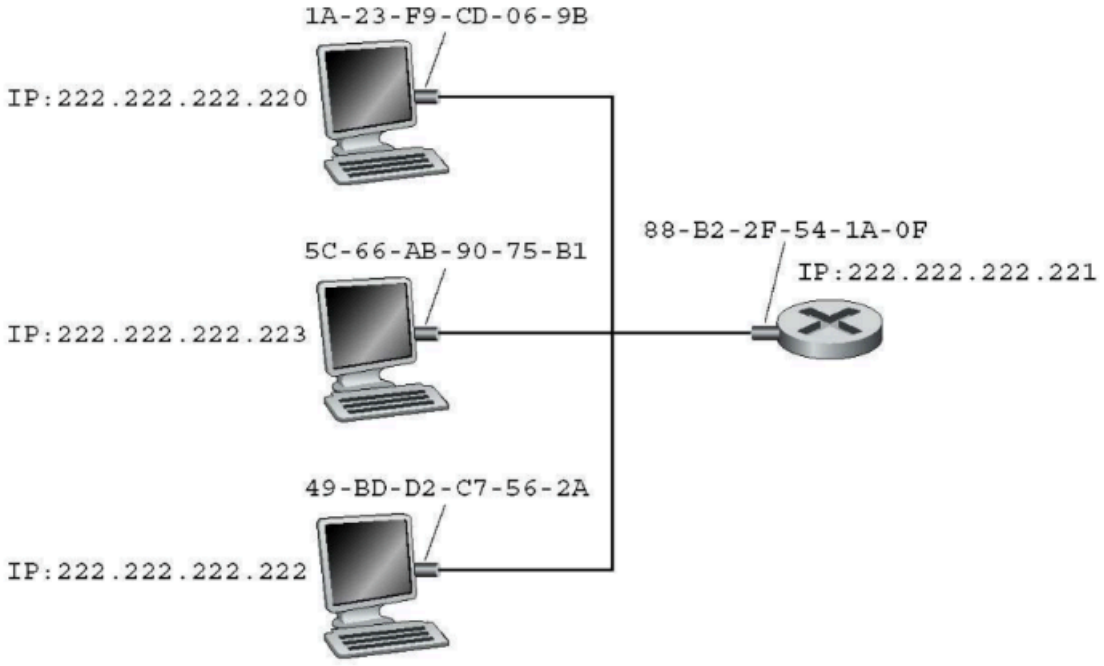


In questo caso i dispositivi non possono comunicare direttamente attraverso un'unico router con le altre sottoreti.

MAC Address

Ogni dispositivo è identificato logicamente tramite l'indirizzo IP, mentre il suo reale identificativo è il **MAC Address** (**Media Access Control**). Si tratta di un identificatore a 6 byte (48 bit), tipicamente in formato esadecimale, assegnato dalla fabbrica alla scheda di rete. Il MAC Address è associato alla scheda di rete similmente a come il codice IMEI è associato ad un telefono. Si tratta di un codice **univoco**, che identifica in maniera esclusiva la scheda di rete. All'interno del dispositivo sono presenti delle aree di memoria a sola lettura (Flash, EPROM, ecc.). Queste memorie di sola lettura possono essere riprogrammabili e quindi aggiornate. Il MAC Address è inserito in un'area di memoria a sola lettura che **non** può essere riprogrammato.
In passato è stato provato a modificare il MAC Address, ma generalmnete risulta quasi impossibile modificare questo indirizzo.
Nelle attività forensi, per tracciare il comportamento di un host non si utilizza l'indirizzo IP, bensì l'indirizzo MAC. Mentre l'indirizzo IP infatti può essere mascherato, ad esempio usando proxy, VPN o catene di proxy, il MAC Address identifica univocamente il dispositivo fisico in sé.
Il MAC non è quindi un indirizzo del network layer, bensì del data-link layer.

Consideriamo allora la seguente rete:



Supponiamo l'host con IP 222.222.222.220 voglia comunicare con 222.222.222.222. Nel layer di network il primo host può inviare direttamente i dati al secondo. Al livello di data link, tuttavia, il primo host non conosce il MAC del secondo host, informazione necessaria per l'incapsulamento nel data link layer. Utilizza allora il **protocollo ARP** (**Address Resolution Protocol**) e chiede tramite un messaggio broadcast qual è il dispositivo che ha quell'indirizzo IP, in attesa di ricevere l'indirizzo MAC corrispondente. L'indirizzo MAC viene inviato dal dispositivo che riconosce di avere quell'indirizzo IP. L'informazione dell'indirizzo MAC viene utilizzata per il layer 2.

Protocollo ARP

Il protocollo ARP viene utilizzato per tradurre l'indirizzo IP in un indirizzo MAC, in modo tale da poter inviare correttamente un **frame Ethernet** al dispositivo corretto nella rete locale.

Chiaramente se ci troviamo all'interno di una sottorete con numerosi host, è molto scomodo chiedere ogni volta l'indirizzo MAC. Si creano quindi delle **tabelle ARP**, delle cache che velocizzano la comunicazione. Se si ha già comunicato con un indirizzo IP, allora si inserisce un record nella tabella, associando a tale indirizzo logico l'indirizzo MAC. All'interno della tabella è presente una colonna "Type", che specifica se la voce è **statica**, ossia configurata manualmente, o **dinamica**, tramite ARP. Se la voce è dinamica, si aggiunge un'informazione, "Expiry", che specifica il tempo rimanente prima che la voce venga eliminata dalla tabella. Un expiration time molto piccolo annulla i vantaggi nell'utilizzo della cache; un expiration molto grande crea rischi di comunicazione.
Supponiamo infatti di avere un host che richiede il MAC dell'indirizzo 192.168.1.2. Un host esterno potrebbe rispondere alla richiesta, sebbene il suo IP sia differente, mandando il suo MAC. Si crea quindi un **sabotaggio**.

Assegnazione indirizzi IP: RARP e BOOTP

In **RARP** (**Reverse ARP**) il client conosce l'indirizzo MAC ma non conosce l'indirizzo IP. Il client allora invia un messaggio broadcast, chiedendo qual è il suo indirizzo IP. I dispositivi ascoltano la richiesta, **ma solo il server RARP comunica l'indirizzo IP**: la risposta è in **unicast**, inviata direttamente al richiedente. La comunicazione broadcast del messaggio inviato dal client viene semplicemente utilizzata in quanto il client non conosce a prescindere l'indirizzo del server, pertanto è più comodo inviare un messaggio broadcast.
In questo caso, tuttavia, la criticità risiede nel fatto che il server RARP potrebbe essere attaccato: se il server viene attaccato, è possibile modificare l'associazione MAC Address - IP Address.

Il RARP è stato col tempo rimpiazzato dal **BOOTP**, precursore del DHCP. Il BOOTP nacque dall'esigenza di avere client **diskless**: i nodi della rete non sono host con sistemi operativi, ma possibilmente anche **terminali**. Questi client allora eseguono una richiesta di **bootstrap**, ossia di **avvio di sistema**, richiedendo un indirizzo IP e un'**immagine di sistema da eseguire**: si avvia così un TFTP (Trivial FTP, FTP minimale) per scaricare l'immagine del sistema operativo e avviare l'OS direttamente dalla rete. Il BOOTP allora è utilizzato per richiedere la configurazione di rete del dispositivo. Oggi è sostituito dal DHCP.

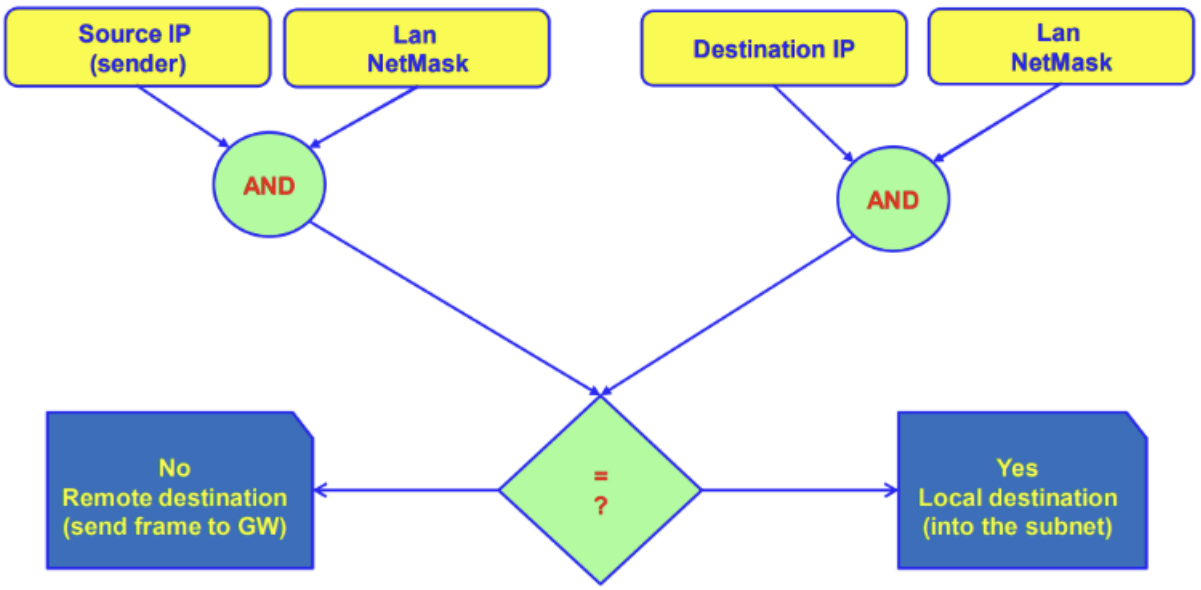
Tabella di routing

Se un host deve comunicare con un altro, ma non conosce gli hop necessari per arrivare ad indirizzi con subnet diversa, può utilizzare una **tabella di routing**. Una tabella di routing contiene le associazioni Indirizzo IP e next-hop. Le associazioni dicono quindi che una particolare destinazione può essere raggiunta trasmettendo i pacchetti a un router specifico, che rappresenta quindi l'hop, ossia la tappa, successiva verso la destinazione finale.

```
IPv4 Tabella route
===========================================================================
Route attive:
     Indirizzo rete            Mask          Gateway       Interfaccia Metrica
          0.0.0.0          0.0.0.0    192.168.1.1    192.168.1.13     35
        127.0.0.0        255.0.0.0      On-link         127.0.0.1    331
        127.0.0.1  255.255.255.255      On-link         127.0.0.1    331
  127.255.255.255  255.255.255.255      On-link         127.0.0.1    331
       172.28.0.0    255.255.240.0      On-link        172.28.0.1   5256
       172.28.0.1  255.255.255.255      On-link        172.28.0.1   5256
    172.28.15.255  255.255.255.255      On-link        172.28.0.1   5256
      192.168.1.0    255.255.255.0      On-link       192.168.1.13    291
     192.168.1.13  255.255.255.255      On-link       192.168.1.13    291
    192.168.1.255  255.255.255.255      On-link       192.168.1.13    291
        224.0.0.0        240.0.0.0      On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0      On-link       192.168.1.13    291
        224.0.0.0        240.0.0.0      On-link        172.28.0.1   5256
  255.255.255.255  255.255.255.255      On-link         127.0.0.1    331
  255.255.255.255  255.255.255.255      On-link       192.168.1.13    291
  255.255.255.255  255.255.255.255      On-link        172.28.0.1   5256
===========================================================================
```

Maschere di sottorete

Abbiamo visto la notazione CIDR per conoscere la subnet. Possiamo allora utilizzare la **maschera di sottorete** e calcolare quale sono tutte le possibili combinazioni di indirizzi IP.
La **maschera di sottorete**, o **subnet mask**, viene utilizzata per riconoscere la rete di appartenenza. Si ottiene facendo un AND logico tra la maschera di sottorete, ossia un indirizzo IP tale che tutti i bit della sezione net sono posti a 1 e i restanti a 0. La subnet mask ad esempio permette di capire se il destination IP identifica un'interfaccia presente all'interno della stessa sottorete. Basta infatti operare nella seguente maniera:



11 - Network Layer, 3

Assegnazione degli indirizzi: DHCPv4

Ogni dispositivo viene riconosciuto attraverso l'indirizzo IP. L'indirizzamento può essere statico o dinamico. Il protocollo **DHCPv4** (**Dynamic Host Configuration Protocol**) consente l'assegnazione dinamica degli indirizzi. Esso funziona attraverso un server centralizzato con mirroring in modo tale che se il server va giù si ha sempre qualcuno che assegna indirizzi IP.

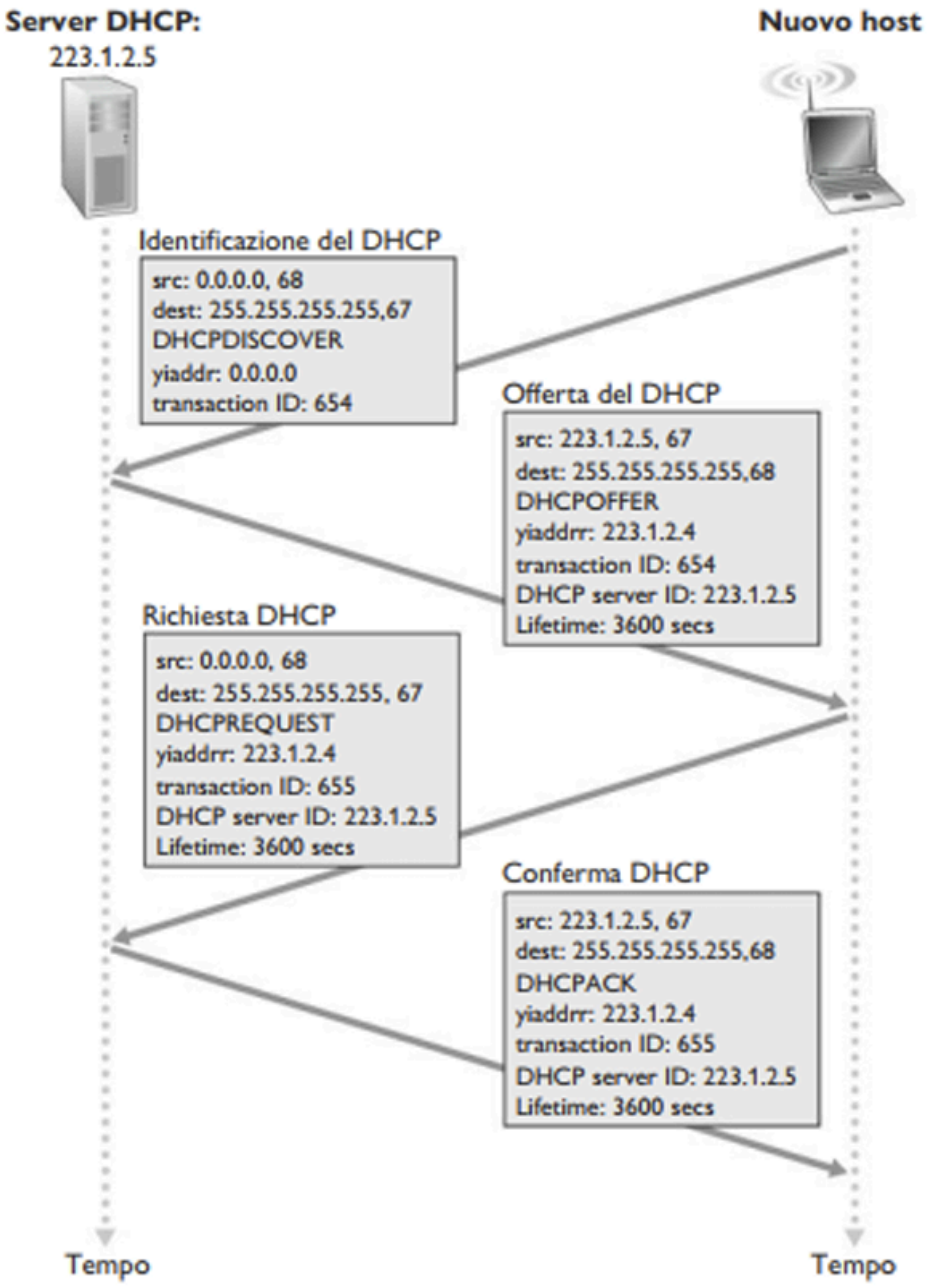
Essenzialmente DHCP consente a un host di ottenere un indirizzo IP in maniera automatica tramite una **Four-way handshake**. Oltre all'indirizzo IP, l'host riesce anche a ricevere informazioni riguardanti la sua subnet mask, il gateway e il server DNS locale. Il protocollo viene spesso chiamato protocollo **plug-and-play** o **zero-conf** (dove "conf" è l'abbreviazione di "configuration") in quanto la connessione degli host alla rete è automatizzata. Solitamente, inoltre, è presente un server DHCP per sottorete, ma in caso contrario è necessario un **agente di relay DHCP** che conosca l'indirizzo di un server per una determinata rete; tale agente di relay può essere anche un router.

Four-Way handshake per DHCP

DHCP utilizza un meccanismo a quattro vie per l'assegnazione di indirizzi IP agli host:

1. **DHCP server discovery**: un host appena connesso invia un pacchetto UDP nella porta 67. Il messaggio inviato prende il nome di **DHCP discover** e viene inviato in **broadcast** all'interno della sottorete. La DHCP discover contiene i campi di sorgente e destinazione (0.0.0.0 e 255.255.255.255,67), l'ID di transazione e un campo `yiaddr`, utilizzato per specificare quale IP può utilizzare l'host. In questo caso, il campo è impostato a 0.0.0.0;
2. **DHCP offer**: il server DHCP risponde al client con una **DHCP offer** inviata in broadcast. La DHCP offer in questo caso contiene nel campo `yiaddr` l'IP offerto al client richiedente e specifica l'indirizzo IP di sé stesso. Aggiunge anche una **lifetime**, ossia la durata dell'indirizzo IP. Allo scadere della lifetime, il client dovrà richiedere un nuovo indirizzo IP;
3. **DHCP request**: il client sceglie l'offerta del server DHCP (o dei server, se vi è più di un server) e risponde con una **DHCP request**. Specfica quindi nella `yiaddr` l'indirizzo richiesto;
4. **DHCP confirm**: il server risponde con un **DHCP ACK** per confermare i parametri richiesti.

Dopo la DHCP ACK, l'interazione è completata e il client può usare l'indirizzo IP per la lifetime.



NAT: Network Address Translation

Poiché la crescita della rete Internet fu esplosiva, ci si è resi conto che indirizzare 2^{32} dispositivi non era sufficiente. Quando i telefoni sono diventati smartphone, ossia nodi attivi in rete tramite la SIM, il problema si fece ancora più concreto. Per tamponare la crescita di dispositivi si utilizza la **NAT**: invece di assegnare un indirizzo IP ad ogni nodo, ad ogni dispositivo appartenente alla stessa rete viene assegnato uno stesso indirizzo IP a tutti i dispositivi, e cambia solamente la porta. Ad esempio, per i dispositivi con IP 10.0.0.1 fino a 10.0.0.100 possiamo assegnare un solo indirizzo IP tramite il router NAT. Di indirizzo IP quindi se ne utilizza uno solo. Quando internet deve comunicare con un dispositivo nella rete, invia i pacchetti al router, e il router smista correttamente i pacchetti al dispositivo specifico.

Ci sono diversi vantaggi nell'utilizzo di NAT:

- si ha un solo indirizzo IP per tutti i dispositivi;
- si può cambiare l'indirizzo IP degli host nella rete locale senza notificare la rete intera;
- si può cambiare l'ISP senza modificare gli indirizzi IP all'interno di reti locali;
- i dispositivi all'interno della sottorete non sono visibili dall'esterno: la rete infatti vede solo il router NAT.

Un router NAT in maniera trasparente deve:

- rimpiazzare l'indirizzo IP sorgente e la porta di ogni datagramma in uscita con l'IP del NAT e una nuova porta;
- ricordare in una **NAT translation table** ogni coppia IP Address, porta - NAT IP Address, nuova porta;
- rimpiazzare l'indirizzo IP NAT e la porta dei datagrammi in ingresso con il source IP address e la porta originali.

Ad esempio, supponiamo che un host con indirizzo 10.0.0.1 richieda una web page da un server HTTP. L'host apre una porta, 3345 ad esempio, e invia il datagramma IP al router NAT. Il router NAT modifica l'indirizzo IP sorgente con il proprio indirizzo e la porta con un nuovo numero di porta, nel nostro caso 5001. A questo punto il pacchetto viene inviato al server. Il server risponde al router NAT, indicando tuttavia come porta e indirizzo di destinazione quelli del router NAT. Il router, una volta ricevuto il pacchetto, consulta la translation table, modifica i campi indirizzo e porta e inoltra il datagramma nella rete domestica.

Con l'avvento di **IPv6**, che permette di avere indirizzi IP a 128 bit, non si ha nessuna carenza di indirizzi IP, come nel caso IPv4. Non si avrebbe quindi la necessità di avere il NAT. Inoltre, il NAT **viola la end-to-end argument** a causa della manipolazione della porta a livello di network layer. Un nodo dev'essere in grado di comunicare con un altro nodo, mentre con il NAT si comunica con il NAT, e poi il NAT comunica con il dispositivo. Quindi **il nodo non conosce l'esistenza dell'altro nodo direttamente**.

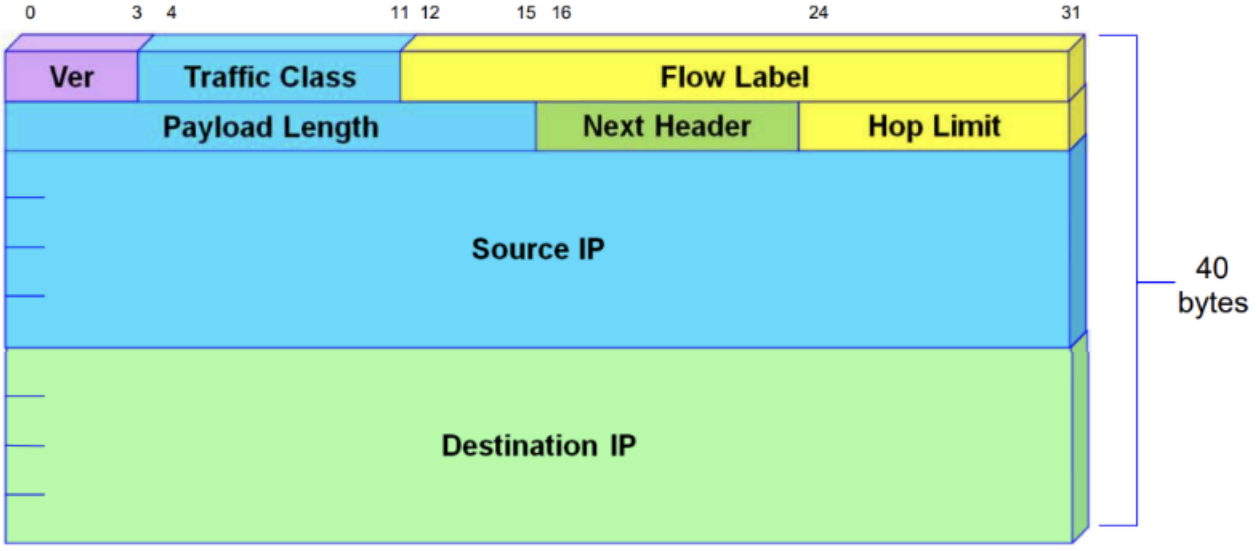
IPv6

L'idea che 32 bit fossero sufficienti per indirizzare i dispositivi in internet si rivelò ben presto una supposizione errata. IPv4 infatti ha solamente $2^{32} = 4$ miliardi di indirizzi dispobivili. **IPv6** invece offre $2^{128} = 340$ undecilioni di indirizzi. Dovrebbe essere chiaramente un numero più che sufficiente di indirizzi.

IPv6 offre nuovi servizi, tra cui:

- sicurezza;
- configurazione automatica Plug and Play;
- QoS;
- multicast addressing;
- mobile host addressing.

Formato Datagramma



Ogni Datagramma IPv6 contiene i seguenti campi:

- **versione**: campo a 4 bit. Per IPv6 è chiaramente uguale a 6;
- **classe di traffico**: campo a 8 bit simile al ToS per attribuire priorità a certi datagrammi;
- **etichetta di flusso**: campo a 20 bit per identificare un flusso di datagrammi;
- **lunghezza del payload**: campo a 16 bit che indica il numero di byte nel datagramma a seguire dall'intestazione. I 16 bit identificano un intero senza segno;
- **intestazione successiva**: campo che identifica il protocollo a cui vengono consegnati i dati del datagramma (TCP, UDP, ecc.);
- **limite di hop**: campo che indica il numero massimo di hop. Il contenuto del campo è decrementato di 1 da ciascun router;
- **indirizzi sorgente e destinazione**: indirizzi a 128 bit che identificano sorgente e destinazione.

I campi frammentazione/riassemblaggio, checksum e opzioni sono stati rimossi:

- la frammentazione in IPv6 non è consentita su router intermedi. Se un router riceve un pacchetto IPv6 troppo grande, il router invia al mittente un messaggio ICMP per indicare l'errore;
- il checksum è implementato già a livello di trasporto, pertanto per migliorare l'elaborazione dei datagrammi è stato rimosso il checksum a livello di rete;
- il campo opzioni non fa parte dell'intestazione IP standard.

Indirizzi IP in IPv6

Gli indirizzi nel protocollo IPv6 sono identificatori a 128 bit per interfacce e set di interfacce. Ci sono tre tipi di indirizzi:

- **unicast**: è un identificatore per una singola interfaccia. Un pacchetto inviato ad un indirizzo unicast viene consegnato all'interfaccia identificata da quell'indirizzo;
- **anycast**: è un identificatore per un set di interfacce. Un pacchetto inviato ad un indirizzo anycast viene consegnato a una delle interfacce. Più precisamente, viene inviata all'interfaccia **più vicina** identificata dall'indirizzo;
- **multicast**: è un identificatore per un set di interfacce. Un pacchetto inviato ad un indirizzo multicast viene inviato a **tutte** le interfacce identificate da quell'indirizzo.

Ogni interfaccia, inoltre, necessita di un indirizzo unicast link-local per la comunicazione all'interno di una sottorete.

La rappresentazione degli indirizzi IPv6 prevede l'utilizzo di 8 numeri a 16 bit. Questi numeri vengono rappresentati in un formato **esadecimale**, come l'indirizzo ABCD:EF01:2345:6789:ABCD:EF01:2345:6789. I prefissi degli indirizzi utilizzano la notazione CIDR con la rappresentazione indirizzo-ipv6/lunghezza-prefisso. La lunghezza del prefisso è specificata con un intero decimale. Se alcune parti dell'indirizzo sono poste a 0 (ad esempio, 2001:0000) si può anche porre un unico 0. Se le restanti parti dell'indirizzo sono poste a 0, basta utilizzare "::". Quindi per rappresentare il prefisso esadecimale 20010DB80000CD3 si possono usare le rappresentazioni:

- 2001:0DB8:0000:CD30:0000:0000:0000:0000/60;
- 2001:0DB8::CD30:0:0:0:0/60;
- 2001:0DB8:0:CD30::/60.

Il tipo di indirizzo IP può essere identificato dal prefisso:

Tipo di indirizzo	Prefisso binario	Notazione IPv6
Non Specificato	00...0 (128 bit)	::/128
Loopback	00...1 (128 bit)	::1/128
Multicast	11111111	FFXX::/8
Link-Local unicast	1111111010	FE80::/10
Global Unicast	altrimenti	

Indirizzi unicast

Gli indirizzi unicast si dividono in indirizzi unicast globali, site-local e link-local. Con RFC 4291, i site-local sono stati deprecati, ma venivano utilizzati per la comunicazione di nodi all'interno della stessa organizzazione e con prefisso FEC0::/48. I link-local unicast invece utilizzano un prefisso FE80::/10.
I nodi IPv6 possono non avere una conoscenza dettagliata della struttura dell'indirizzo unicast. Un host può considerare l'indirizzo come un numero a 128 bit. Un host più sofisticato, invece, può dividere l'indirizzo in n bit di prefisso e $128 - n$ bit per l'ID dell'interfaccia.
L'ID dell'interfaccia, come prevede RFC 2373, può essere creato dall'host utilizzando l'**Extended Unique Identifier-64** (**EUI-64**). Si utilizza il MAC Address a 48 bit, dividendolo in due porzioni di 24 bit ciascuno:

- **Organizationally Unique Identifier** (**OUI**);
- **Network Interface Card** (**NIC**) specifier.

Una volta diviso in due parti, si pone al centro la sequenza a 16 bit FFFE. Infine, i 2 bit meno significativi del byte più significativo dell'OUI possono essere impostati in virtù delle caratteristiche dell'unicast:

- il bit più significato tra i due viene posto a 0 se l'indirizzo unicast è **globalmente unico**. Se invece è **amministrato localmente**, viene posto a 1;
- il bit meno significativo tra i due identifica se l'indirizzo è **unicast** o **multicast**. Per gli indirizzi unicast si pone il bit a 0, per il multicast a 1.

Ottenuta la sequenza a 64 bit, RFC 2373 indica come porre i primi 64 bit:

- i primi 10 bit corrispondono al prefisso FE80, ossia 1111111010;
- i restanti 54 bit sono tutti posti a zero.

Per gli indirizzi unicast globali, RFC 4291 utilizza un formato specifico, definendo n bit per il prefisso di routing globale, m bit per l'ID di subnet e $128 - n - m$ bit per l'ID di interfaccia. Definisce inoltre i valori suggeriti di n ed m, ossia $n = 48, m = 16$. Inoltre i primi tre bit della parte di prefisso sono sempre posti a 001.
All'interno di un indirizzo unicast globale è possibile **mappare** un indirizzo IPv4, ponendo i primi 96 bit a 0 e nei restanti 32 bit l'indirizzo. In alternativa, si possono porre i primi 80 bit a 0, codificare i successivi 16 bit con FFFF e i restanti 32 bit con l'indirizzo IPv4.
Per i site-local unicast, i 54 bit centrali possono essere utilizzati per la subnet ID, sebbene siano deprecati.

Indirizzi anycast

Un indirizzo anycast può essere assegnato a più di un'interfaccia (e quindi a più nodi). Un pacchetto inviato ad un indirizzo anycast viene ricevuto dall'interfaccia più vicina.
Per gli indirizzi anycast si utilizza lo spazio di indirizzamento unicast, pertanto a livello sintattico gli indirizzi anycast sono indistinguibili dagli indirizzi unicast. Tuttavia, l'indirizzo di anycast è predefinito. Avendo n bit di prefisso che identificano la subnet, i restanti $128 - n$ bit sono tutti posti a 0.

Gli indirizzi anycast possono essere utilizzati per individuare un set di router collegati ad una certa subnet, oppure per trovare un set di router utilizzati da una certa organizzazione.

Indirizzi multicast

Gli identificatori multicast sono caratterizzati da un prefisso a 16 bit, in cui:

- i primi 8 bit sono posti a 1 (FFXX::/8);
- i successivi 4 bit sono **flag**;
- i rimanenti quattro bit sono **scope value**, utilizzati per limitare lo scope del gruppo multicast.

I restanti 112 bit sono utilizzati per identificare il **group ID**, ossia il gruppo multicast che identifica l'insieme di nodi nella rete.
Valori noti per lo scope del gruppo multicast sono i seguenti:

Valori	Scope	Estensione dello scope
1	Interface-local	Si estende su una sola interfaccia su un nodo ed è utile solo per loopback
2	Link-local	Si estende sulla regione topologica dello scope unicast corrispondente
4	Admin-local	Il più piccolo scope configurato dall'amministrazione
5	Site-local	Si estende su un singolo sito
8	Organization-Local	Si estende su multipli siti di una stessa organizzazione
E	Global	Lo scope si estende globalmente

I valori 0,3,F sono riservati, mentre i rimanenti sono non assegnati, permettendo quindi ad amministratori di definire scope aggiuntivi.

Tra gli indirizzi multicast distinguiamo tre gruppi:

- **well-known**, indirizzi predefiniti utilizzati per specifici servizi e protocollli IPv6 (routing, neighbor discovery, ecc.). Hanno prefisso FF00::/8;
- **transient**, utilizzati per sessioni di comunicazione temporanee. Hanno prefisso FF02::/16;
- **solicited-node**, generati automaticamente per ogni indirizzo unicast o anycast e utilizzati per neighbor discovery e per **Duplicate Address Detection** (**DAD**). La neighbor discovery rimpiazza l'ARP di IPv4 come vedremo a breve. Hanno prefisso FF02::1:FF00:0/104.

Incapsulamento

L'incapsulamento di un datagramma IPv6 è specificato nell'RFC 2464. Di default la MTU size è impostata a 1500 byte. Ogni datagramma IPv6 viene incapsulato in un frame Ethernet. In particolare, l'intestazione Eternet deve specificare nel campo tipo il valore esadecimale 86DD.

Neighbor Discovery Protocol

Al fine di determinare gli indirizzi al livello di collegamento i nodi di rete utilizzano il **Neighbor Discovery Protocol** (**NDP**). I nodi utilizzano questo protocollo anche per capire quali vicini sono raggiungibili o meno ("neighbor" significa "vicino di casa", per cui faremo riferimento ai nodi con il termine "vicino"). L'RFC 4861 definisce l'NDP per IPv6.

Il protocollo riesce a risolvere i seguenti problemi:

- **router discovery**, ossia come localizzare un router;
- **prefix discovery**, ossia come scoprire il set di indirizzi raggiungibile (i.e. la sottorete di appartenenza);
- **parameter discovery**, ossia come dedurre parametri (MTU, limite hop, ecc.);
- **Address Autoconfiguration and resolution**, ossia come configurare un indirizzo in maniera stateless e come determinare l'indirizzo a livello di collegamento a partire dall'indirizzo IP, quindi **come trovare un vicino**;
- **next-hop determination**, ossia come trovare il prossimo hop;
- **neighbor unreachability detection**, ossia come determinare se un vicino non è raggiungibile;
- **duplicate address detection** (**DAD**), ossia come determinare se un indirizzo è già usato;
- **redirection**, ossia come un router informa un host su un miglior hop per raggiungere la destinazione.

NDP si basa su cinque tipi di pacchetti **ICMP**:

- **Router Solicitation**: quando un'interfaccia si attiva, gli host possono inviare questo messaggio per richiedere ai router di generare immediatamente dei messaggi Router Advertisement immediatamente, piuttosto che aspettare l'orario schedulato;
- **Router Advertisement**: i router notificano la loro presenza. Questa notifica viene inviata periodicamente o in risposta ad una Router Solicitation. Il messaggio di advertisement contiene i prefissi usati per determinare se un altro indirizzo appartiene alla stessa rete e le configurazioni sugli address, valori di limite hop consigliati e anche specificare agli host di utilizzare DHCPv6 o una stateless address configuration ;
- **Neighbor Solicitation**: un nodo invia questo pacchetto per verificare se un suo vicino è ancora raggiungibile o per verificare un DAD;
- **Neighbor Advertisement**: un vicino risponde ad una solicitation. Un nodo può anche inviare degli advertisement per notificare di un cambiamento di indirizzo a livello di collegamento;
- **Redirect**: un router invia un messaggio di redirect per informare gli host della possibilità di ottenere un hop migliore per raggiungere la destinazione. Un router, inoltre, può inviare un redirect ad un host per notificare che il destinatario è un vicino.

Per risolvere un indirizzo, gli host eseguono un multicasting di un Neighbor Solicitation, chiedendo al nodo target di inviare l'indirizzo a livello di collegamento e includendo il suo indirizzo. Il nodo target restituisce il suo indirizzo attraverso un messaggio Neighbor Advertisement unicast. Per proteggersi dallo spoofing, RFC 3972 definisce il **Secure Neighbor Discovery protocol** (**SEND**). Ogni nodo genera una coppia di chiavi RSA, e la chiave pubblica viene utilizzata per la generazione dell'IPv6, legando quest'ultimo crittograficamente alla chiave. Quando un nodo invia il messaggio Neighbor Advertisement, deve includere la firma digitale RSA con la chiave privata e la chiave pubblica. Il nodo che vuole risolvere l'indirizzo verifica che IPv6 sia coerente con la chiave pubblica e verifica anche la firma RSA per assicurarsi che il messaggio provenga da un mittente legittimo. In questo modo **il problema dell'ARP spoofing viene risolto**.

Messaggi NDP per il Duplicate Address Detection

RFC 4862 inoltre specifica come utilizzare la Neighbor Solicitation per la DAD. In breve, un'interfaccia si unisce ad un indirizzo multicast di tutti i nodi (nota: l'interfaccia invia un messaggio report seguendo il protocollo **Multicast Listener Discovery**, utilizzato dai router e dagli switch per capire quali host fanno parte di un determinato gruppo. I messaggi che arrivano ad un indirizzo multicast vengono inoltrati ai membri del gruppo, che sono "iscritti" a questo indirizzo multicast) e anche all'indirizzo multicast solicited-node dell'indirizzo desiderato:

- il primo collegamento assicura al nodo di ricevere Neighbor Advertisement da altri nodi che potrebbero usare l'indirizzo;
- il secondo collegamento viene utilizzato per capire direttamente se due nodi stanno utilizzando lo stesso indirizzo.

Da notare come l'utilizzo esclusivo del secondo collegamento non sia sufficiente: ci possono essere dispositivi che usano l'indirizzo target ma che non si uniscono al solicited-node multicast address.
L'interfaccia interessata all'acquisizione dell'indirizzo invia messaggi Neighbor Solicitation, con IP target pari all'indirizzo che si vuole controllare (detto **tentative address**), l'indirizzo IP sorgente pari all'indirizzo non specificato (in notazione IPv6 `::`) e indirizzo destinazione al solicited-node multicast dell'indirizzo target. Qualunque nodo che riceve la Solicitation per un tentative address e non sta eseguendo una DAD **non deve rispondere con un Advertisement**: se un nodo risponde con un Advertisement, significa che l'indirizzo target è utilizzato da quest'ultimo.
Evitare che i nodi rispondano con un Advertisement assicura che il processo di DAD sia sicuro. Il nodo che effettua la Solicitation inoltre può ricevere il suo stesso NS in quanto il nodo fa un loopback dei pacchetti multicast. Se un nodo riceve una Solicitation da

un altro nodo, allora **un altro nodo ha generato lo stesso tentative address e sta eseguendo una DAD**. Negli altri casi, l'indirizzo è unico e può essere utilizzato.

Header opzionali

IPv6 permette di codificare informazioni opzionali a livello di rete in header separati. Questi header aggiuntivi vengono piazzati tra l'intestazione fissa di IPv6 e l'header del livello più alto. Un valore, il **Next Header value**, identifica il tipo di header di estensione. Gli header opzionali non vengono processati da nessun nodo fino a quando il pacchetto raggiunge il nodo target (se un multicast, ogni nodo target). RFC 2460 specificava in realtà che l'header opzionale **Hop-By-Hop options** dovesse essere processato da ogni nodo che riceveva il pacchetto. RFC 8200 specifica che i solo i nodi configurati appositamente processino le opzioni dell'header.

Altri header vengono utilizzati per la sicurezza (Authentication e Encrypted security payload header). Un header opzionale è anche il **fragmentation header**. IPv6 di norma non concede la frammentazione, con una MTU minima di 1280 byte e di norma impostata a 1500 byte. Il sender tuttavia può frammentare i pacchetti utilizzando questo header, frammentando solamente la parte successiva all'header. Ad ogni frammento del payload viene aggiunta la parte non frammentabile (ossia gli header). Inoltre, l'header di frammentazione presenta informazioni simili a quelle viste in IPv4:

- **Next Header value**, a 8 bit e indicatore del successivo header;
- **reserved**, un padding a 8 bit;
- **fragment offset**, che ha lo stesso significato dell'offset di IPv4;
- altri due bit **reserved**;
- bit **M** o flag **more fragments**, che indica la presenza di più frammenti dopo quello inviato;
- **identification number**, che a differenza di IPv4 è a 32 bit.

Per quanto riguarda l'**Authentication Header**, esso garantisce autenticità e correttezza del pacchetto ricevuto. Permette di proteggere contro attacchi quali **inserimento di header aggiuntivi**. L'header contiene:

- **next header value** a 8 bit;
- **payload length**, a 8 bit;
- **reserved**, a 16 bit e solo di padding;
- **security parameters index**, un valore arbitrario a 32 bit per identificare la **security association** del ricevitore per autenticare il pacchetto (id est l'algoritmo di cifratura usato, un token crittografico);
- **sequence number**, un numero crescente di sequenza a 32 bit per evitare **attacchi di replay**. Questo garantisce che se un pacchetto viene sniffato (ad esempio contiene l'hash di una password) allora non può essere riutilizzato;
- **authentication data**, di lunghezza variabile multiplo di 32 bit usato per check di integrità.

Se invece vogliamo crittografare i dati, utilizziamo l'**Encrypted security payload Header**. Abbiamo:

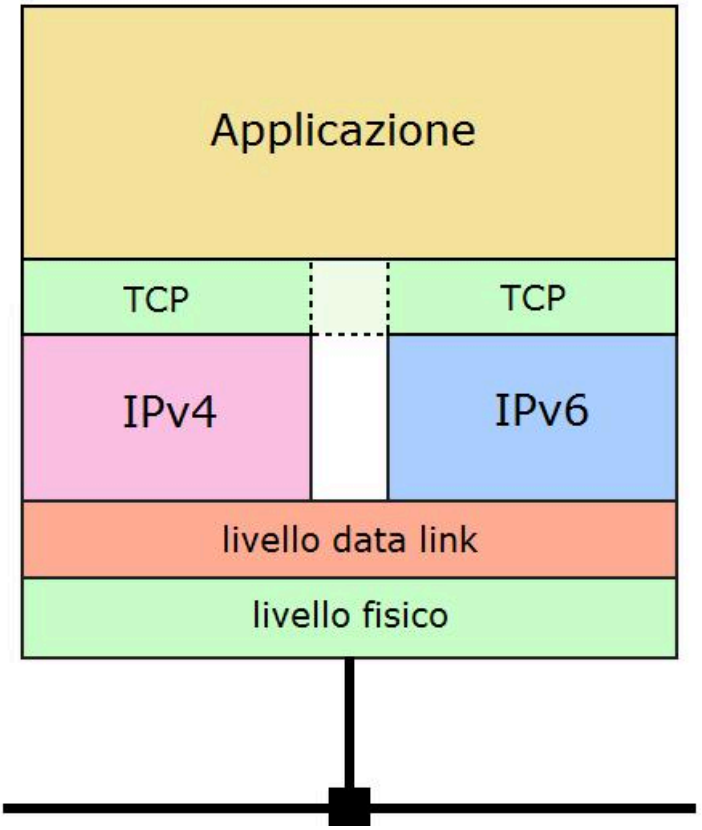
- **security parameters index**, a 32 bit per identificare la crittografia utilizzata e quindi la security association per decifrare il pacchetto;
- **sequence number**, numero di sequenza crescente a 32 bit per evitare attacchi di replay;
- **dati crittografati**;
- **padding**;
- **lunghezza di padding**;
- **next header value**;
- **authentication data** per check di integrità a lunghezza variabile di 32 bit.

Transizione IPv4 a IPv6

Passare dalla versione 4 alla versione 6 il protocollo IP richiede un aggiornamento massiccio dei dispositivi di rete. Non è quindi possibile eseguire una transizione spontanea della rete da IPv4 a IPv6. Si cerca per ora di aggirare la non interpretabilità di IPv6 via software. Le soluzioni adottate fino ad ora sono due: **dual stack** e **tunneling**.

Approccio dual stack

L'approccio dual stack prevede l'utilizzo di un doppio stack IP nella pila di protocollo. Lo smistamento dei pacchetti IPv4 e IPv6 è associato a un diverso EtherType specificato nel livello data link: 0800 per IPv4 e 86DD per IPv6. Sebbene sia semplice da implementare, aumenta la complessità di rete, aggravando il lavoro di router e switch e non risolvendo il problema della diminuzione di pacchetti.



Tunneling

L'approccio alla transizione IPv4/IPv6 più diffuso è il **tunneling**. Se due nodi IPv6 vogliono comunicare attraverso un insieme di router intermedi IPv4 (tale insieme si chiama **tunnel**, da cui il nome dell'approccio), il nodo sender al lato di invio del tunnel inserisce l'intero datagramma IPv6 nel campo dati del datagramma IPv4. I router instradano il datagramma IPv4, ignari della presenza del datagramma IPv4. Il receiver riceverà il datagramma IPv4, determina che contiene il datagramma IPv6 osservando che il **campo del numero di protocollo nel pacchetto IPv4 è pari a 41**. Il datagramma verrà quindi estratto e instradato correttamente. Poiché è difficile realizzare sulle reti globali questa tecnica, il tunneling è limitato a comunicazioni in reti locali.

ICMP

L'**Internet Control Message Protocol** è un protocollo utilizzato per la trasmissione di informazioni di controllo all'interno di una rete. Viene incapsulato direttamente in IP ed è molto utilizzato per strumenti come ping e traceroute.
ICMP esiste sia per IPv4 che per IPv6, e in IPv6 trova importante applicazione per i messaggi del Neighbor Discovery Protocol.
Generalmente ICMP include i campi:

- tipo, campo a 8 bit che specifica il formato dei messaggi;
- codice, campo a 8 bit che qualifica il messaggio;
- checksum, campo a 16 bit. In questo caso è il complemento a uno troncato a 16 bit del complemento a uno della somma dei bit del messaggio con il campo tipo;
- dati, campo a lunghezza variabile e specifico in base al campo "tipo" e "codice".

Firewall

I Firewall sono strutture hardware o software che schermano accessi ad una subnet dall'esterno. A differenza di un proxy, un firewall è un sistema intelligente che analizza il traffico di rete, decidendo ciò che entra e ciò che esce (nel caso di un trojan o di controlli remoti, ciò che esce dev'essere controllato).

Solitamente tra la rete locale e la WAN si crea una terza sottorete detta **Demilitarized Zone** (**DMZ**). Questa zona contiene i sistemi che sono isolati dalla rete interna ma che devono essere raggiungibili dall'esterno, come i server pubblici. Generalmente un firewall, analizzando i pacchetti, può compiere una di queste azioni:

- allow, ossia far passare il pacchetto;
- deny, ossia bloccare e rispedire il pacchetto al mittente;
- drop, ossia bloccare e non notificare il mittente del rifiuto del pacchetto. Spesso se non si lascia passare un pacchetto, il firewall lo droppa per risparmiare banda.

Oltre a filtrare pacchetti, i firewall sono in grado di **rilevare intrusioni** tramite euristiche, scatenanto reazioni automatiche. A volte un firewall può anche fungere da NAT per rendere inaccessibili i calcolatori sulla rete interna mascherandone gl iIP.

Il firewall inoltre si dividono in due gruppi:

- firewall **host-based**, un'applicazione software che monitora il traffico uscente ed entrante;
- firewall **network-based**, un **componente hardware** posto sul confine di una rete in modo da filtrare tutto il traffico che questa scambia con l'esterno. Questi firewall sono dotate di due schede di rete e di un OS minimale.

In un contesto aziendale conviene utilizzare un firewall network-based, in quanto il firewall host-based è eseguito sullo stesso sistema operativo da proteggere. Un malware potrebbe disabilitare il firewall, generando problemi di sicurezza. Il firewall host-based tuttavia è in grado di sapere quale applicazione è in ascolto su una porta, bloccando solo una connessione e facendo passare quella generata da un'altra applicazione, invece di bloccare l'intero dispositivo.

Algoritmi di routing

Al fine di instradare correttamente i pacchetti da sorgente a destinazione si utilizzano degli **algoritmi di routing**. Tipicamente il percorso migliore da sorgente a destinazione è quello che ha un costo minimo, ma a volte è necessario considerare fattori esterni, quali le **policy** (come preferenza di router).
Per modellare il problema del routing si utilizzano i **grafi**, coppie $G = (V, E)$ dove V è un insieme di vertici che rappresentano i router ed E è un insieme di archi che rappresentano i collegamenti tra i router. Nel nostro caso, inoltre, utilizzeremo dei grafi non direzionati e cercheremo di capire qual è il percorso minimo.

In generale, gli algoritmi di routing sono classificabili come centralizzati o decentralizzati:

- gli **algoritmi centralizzati** calcolano il percorso a costo minimo tra sorgente e destinazione sfruttando una **conoscenza globale della rete**. Sono anche noti come algoritmi **link-state**, poiché l'algoritmo è conscio del costo di ciascun collegamento di rete;
- gli **algoritmi decentralizzati** calcolano il percorso a costo minimo tra sorgente e destinazione in modo **distribuito**. Tramite uno scambio di informazioni con i nodi adiacenti, un nodo di rete gradualmente calcola il percorso a costo minimo. In questa categoria è di spicco l'algoritmo **distance-vector**, in cui ogni nodo memorizza i costi dei collegamenti tramite un vettore.

Gli algoritmi inoltre si distinguono in statici e dinamici:

- un algoritmo è **statico** se i percorsi cambiano molto raramente (e in tal caso, cambiano come risultato di un intervento umano);
- un algoritmo è **dinamico** se i percorsi vengono aggiornati al variare del volume di traffico o della topologia di rete.

Si possono anche classificare gli algoritmi come **sensibili** o **insensibili al carico**:

- nei primi i costi dei collegmaenti variano per riflettere il livello di congestione;
- nei secondi, che sono gli attuali algoritmi utilizzati, il costo di un collegamento non riflette il suo livello di congestione.

Flooding

Il protocollo di instradamento più semplice link-state che viene utilizzato anche per sapere se esiste un percorso da sorgente a destinazione è il **flooding**. Nel flooding ogni nodo invia un pacchetto a tutti i suoi vicini eccetto al vicino da cui proviene il pacchetto. Si può pensare al flooding come ad un algoritmo di BFS che tuttavia non tiene conto dei nodi a cui sono arrivati i pacchetti. La tecnica del flooding è usata per **conoscere informazioni relative alla topologia di rete e condividerle con i vicini**. Sebbene riesca a capire se un percorso esista o meno, **in presenza di cicli causa una saturazione della rete**, in quanto in un ciclo i pacchetti verrebbero inoltrati continuamente e si duplicherebbero all'infinito.
Per ovviare al problema dei pacchetti duplicati, esistono diverse soluzioni:

- impostare un **hop count** sul pacchetto;
- utilizzare un ID per i pacchetti. Ogni nodo aggiunge il proprio ID al pacchetto, e se un nodo riceve un pacchetto con lo stesso ID di un altro lo droppa;
- rendere il grafo aciclico;
- utilizzare lo **spanning tree**, ossia un **albero ricoprente** che contiene tutti i vertici del grafo ma solo un sottoinsieme di archi.

Distance Vector Routing

L'algoritmo di routing distance vector è un algoritmo **iterativo**, **asincrono** e **distribuito**:

- **iterativo** perché il processo si ripete fino a quando non si ha uno scambio di informazione tra vicini;

- **asincrono** perché nessun nodo deve stare al passo con gli altri;
- **distribuito** perché ogni nodo riceve informazioni dai vicini.

L'algoritmo utilizzato nel distance vector per aggiornare le stime delle distanze è l'algoritmo di **Bellman-Ford**.

L'algoritmo di Bellman-Ford risolve il problema del problema dei cammini minimi a singola sorgente, individuando anche eventuali cicli di peso negativo che invaliderebbero le reali stime. Ad ogni passo l'algoritmo aggiorna le stime dei percorsi minimi, basandosi sulla proprietà di **disuguaglianza triangolare**, per cui il costo minimo del percorso dal nodo x al nodo y è dato dal costo di uno dei vicini di x più il costo minimo del percorso dal vicino di x scelto al nodo y. Diciamo allora che si verificherà la seguente condizione:

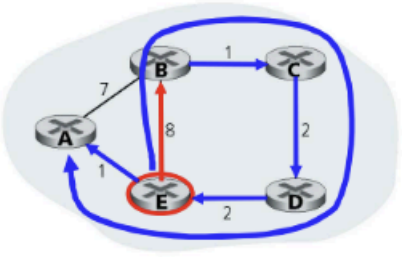
$$\delta(x, y) = w(x, v^*) + \delta(v^*, y)$$

dove $\delta(x, y)$ individua il peso del cammino minimo da x a y e v^* è il vertice che rende minima la somma.
L'algoritmo di Bellman-Ford fornisce le righe della tabella di inoltro di ogni nodo. Infatti se il nodo x vuole inviare un pacchetto a y, dovrebbe prima inoltrarlo a v^*. La tabella di inoltro del nodo x specifica quindi v^* come router successivo.

Ogni nodo x quindi mantiene:

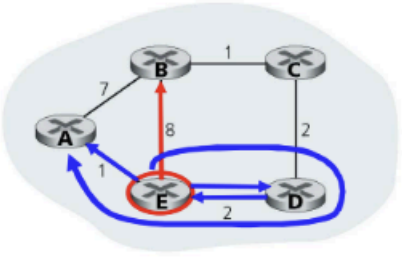
- per ciascun vicino v il costo dell'arco (x, v);
- il vettore delle distanze D_x del nodo x, contenente la stima dei costi dei percorsi minimi verso tutti i nodi del grafo di rete;
- i vettori delle distanze D_v dei vicini v del nodo x, contenenti le stime dei costi dei percorsi minimi verso tutti i nodi del grafo di rete.

Ogni tanto un nodo invia una copia del proprio vettore delle distanze ai vicini. Quando un nodo riceve un vettore delle distanze, allora esegue l'algoritmo di Bellman-Ford e aggiorna il proprio vettore.



$D^E()$	A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

destinazione



$D^E()$	A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

destinazione

Nella tabella soprastante, il vettore delle distanze D_e indica:

- nelle righe la destinazione;
- nelle colonne il nodo da attraversare.

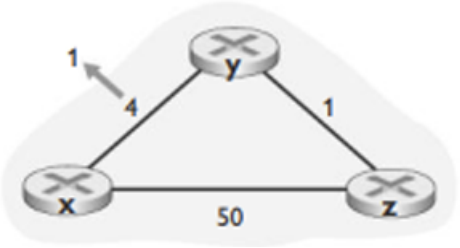
Ogni singolo valore esprime il costo per giungere al nodo destinazione passando per il nodo specificato dalla colonna.

Chiaramente il nodo e conserverà le informazioni relative al vettore delle distanze dei vicini. Inizialmente il vettore delle distanze conterrà solamente pesi infiniti ad eccezione delle destinazioni raggiungibili direttamente dal nodo stesso. Poi applicherà Bellman-Ford per calcolare le altre distanze.

Count-to-infinity problem, poisoned reverse

Se un nodo rileva dei cambiamenti nei costi, aggiorna il proprio vettore delle distanze e lo inoltra ai vicini. Questo cambiamento, tuttavia, può portare a dei problemi; più precisamente, se il costo di un collegamento aumenta, l'applicazione dell'algoritmo di Bellman-Ford influisce pesantemente sulle prestazioni di rete.

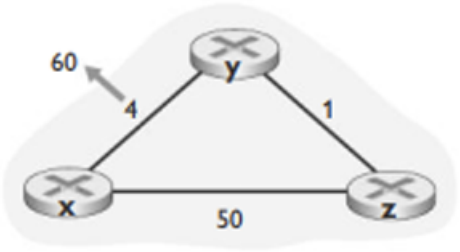
Supponiamo di avere la seguente rete e di modificare il costo del collegamento da y a x da 4 a 1:



Sia y il nodo che rileva tale modifica di costo:

1. y aggiorna il proprio vettore delle distanze e notifica i vicini;
2. z riceve l'aggiornamento. Aggiorna quindi la propria tabella e calcola un costo minimo verso x. Poiché il costo minimo prima era 5, il nuovo costo minimo (2) viene comunicato ai vicini inviando il vettore delle distanze;
3. y riceve l'aggiornamento di z e aggiorna la propria tabella. I costi minimi non cambiano quindi nessun messaggio viene scambiato.

Supponiamo ora di trovarci in questa situazione:



Ancora una volta, y rileva la modifica del costo:

1. y aggiorna la tabella delle distanze. Il valore memorizzato per il percorso minimo verso x tuttavia è 6. Questo perché il percorso minimo da z a x ha ancora peso 5 in quanto z non è a conoscenza della modifica di costo. Si ha un **instradamento ciclico**. Il percorso $y \rightsquigarrow x$ infatti è il percorso $y \rightarrow z \rightarrow y \rightarrow x$ (nota: non significa che il percorso effettivo è questo, ma questa è la stima del costo minimo. Infatti un pacchetto viene instradato dai nodi, quindi y lo instraderà verso z e z lo instraderà verso y e il pacchetto rimane bloccato);
2. la tabella viene inoltrata a z. z sa che il costo del percorso minimo $y \rightsquigarrow x$ è 6, pertanto applicando l'algoritmo di Bellman-Ford sa che può raggiungere x considerando il percorso $z \rightarrow y \rightsquigarrow x$. Il costo aumenta a 7 e la tabella viene inoltrata;
3. y riceve la tabella modificata di z, e riapplicando l'algoritmo di Bellman-Ford ottiene che il costo del percorso minimo $y \rightsquigarrow x$ è 8. La procedura continua fino a quando non si arriva a 51.

Se l'aumento fosse stato considerevole, ci sarebbe stato un **conteggio infinito**. Il problema presentato infatti prende il nome di **count-to-infinity problem**, ossia **problema del conteggio infinito**.
Il problema del conteggio infinito può essere risolto utilizzando una tecnica nota come **inversione avvelenata** (in inglese **poisoned reverse**). L'idea è molto semplice, per cui la applicheremo all'esempio illustrato precedentemente. Poiché z instrada tramite y per giungere a x, z comunica a y che **la distanza verso x è infinita**. In questo modo y non instraderà mai verso z, aggiornando il valore di costo del percorso minimo verso x a 60. La tabella viene quindi passata a z, che cambia percorso e utilizza il collegamento che lo collega a x direttamente, con un costo di percorso minimo pari a 50. La tabella aggiornata viene inviata a y, che applica l'algoritmo di Bellman-Ford e ottiene come costo del percorso minimo 51.

Sebbene l'inversione avvelenata risolva il problema con due nodi, la tecnica **non risolve il problema con cicli con più di due nodi**.

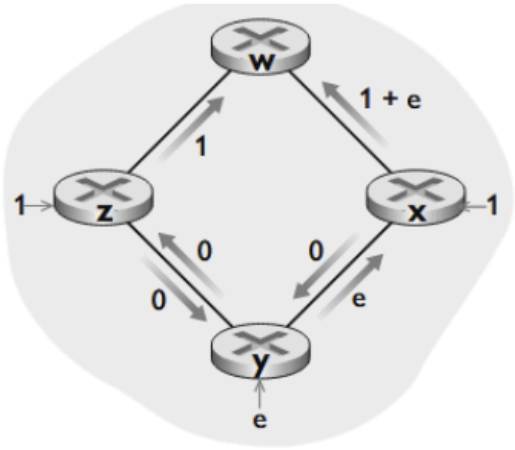
Link-State Routing

In un link-state routing la topologia di rete e i costi dei collegamenti sono già noti. Ogni nodo di rete invia periodicamente informazioni sullo stato dei collegamenti a tutti gli altri nodi della rete con un **link-state broadcast**.

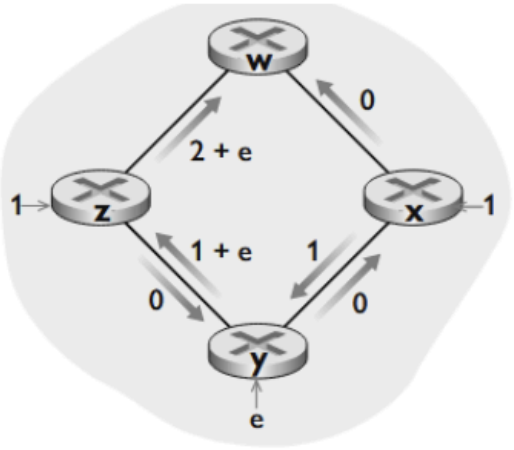
Per il calcolo dei percorsi si utilizza solitamente l'**algoritmo di Dijkstra**, un algoritmo simile ad una esplorazione in ampiezza di un grafo a partire da una singola sorgente ma che utilizza una coda di priorità minima. La priorità di un nodo, in particolare, è stabilità in virtù del costo del percorso minimo dalla sorgente al nodo considerato. Si estrae quindi il nodo con priorità più bassa (id est con costo stimato più basso), e per ogni vicino di tale nodo si aggiorna la stima del costo del percorso minimo verso i vicini. Se l'aggiornamento della stima ha effettivamente mutato il costo del percorso minimo, si aggiorna la priorità del nodo all'interno della coda di priorità minima.

Oscillazioni

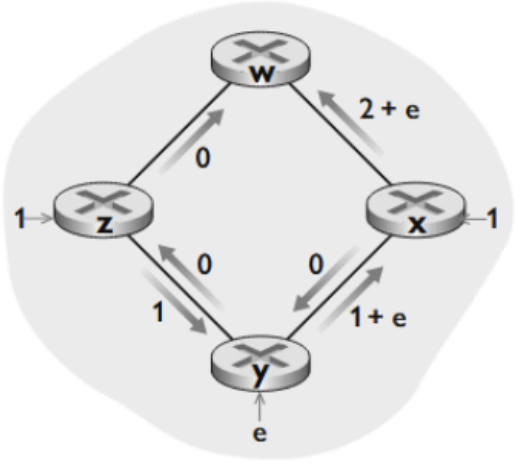
Nel link-state routing possono verificarsi delle **oscillazioni**. Consideriamo la seguente topologia di rete, in cui **i costi sono uguali al carico di trasporto sul collegamento**:



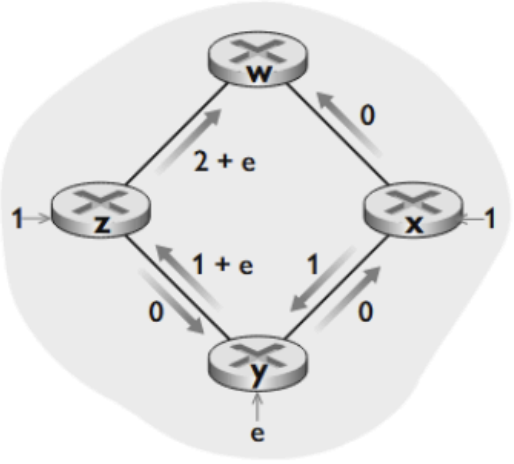
a. Instradamento iniziale



b. *x*, *y* determinano un percorso migliore verso *w*, in senso orario



c. *x*, *y*, *z* determinano un percorso migliore verso *w*, in senso antiorario



d. *x*, *y*, *z* determinano un percorso migliore verso *w*, in senso orario

Il router y vuole inviare una quantità di traffico pari ad e verso w. In virtù dei costi attuali, y deduce, applicando Dijkstra, che il percorso migliore è $y \to z \to w$, con costo 1. Anche x deduce che per inviare traffico pari a 1 a w conviene eseguire un giro in senso orario $x \to y \to z \to w$. Si ottiene quindi che x, y inviano pacchetti in senso orario. La successiva iterazione dell'algoritmo di Dijkstra tuttavia vede i nodi x, y, z instradare il traffico in senso antiorario. Si continua quindi a **oscillare**, cambiando continuamente l'instradamento del traffico.

Per evitare le oscillazioni, basta assicurarsi che non tutti i router lancino l'esecuzione dell'algoritmo di Dijkstra nello stesso istante.

Confronto tra routing link-state e distance vector

Il distance vector routing è un meccanismo decentralizzato in cui i messaggi vengono scambiati solamente tra i vicini. Questi messaggi contengono informazioni riguardanti tutte le destinazioni. Il link-state soffre del problema di infinity counting, che nel caso a due nodi può essere aggirato col poisoned reverse. Si inviano inoltre messaggi nel momento in cui si ha un cambiamento dei costi. Infine, il distance vector non è robusto contro nodi malevoli.

Il link-state routing è un meccanismo centralizzato in cui i nodi inviano messaggi broadcast riguardanti i collegamenti propri. Si hanno problemi di sincronizzazione, ossia l'oscillazione, che può essere aggirata evitando l'esecuzione dell'algoritmo di Dijkstra nello stesso istante. Messaggi sullo stato dei collegamenti devono essere inviati periodicamente. Infine, il sistema del link-state è robusto agli attacchi.

Routing in Internet

Gli algoritmi di instradamento analizzati hanno visto la rete semplicemente come una collezione di router interconnessi, ma per motivi di **scalabilità** (al crescere del numero di router, il costo per la comunicazione di informazioni di rete aumenta) e di **autonomia amministrativa** (ogni ISP vuole gestire in maniera autonoma i propri router) i router sono organizzati in **sistemi**

autonomi (**AS**, Autonomous Systems). I router di un AS eseguono gli algoritmi di routing e hanno informazioni sugli altri router dello stesso AS. Gli algoritmi di routing eseguiti all'interno di un AS prendono il nome di **protocolli di instradamento interno al sistema autonomo**. Per la comunicazione tra AS, si utilizzano **protocIli di instradamento inter-AS**.

Protocollo RIP

Il **Routing Information Protocol** (**RIP**) è un protocollo di routing intra-AS basato sui distance vector che evita i routing loop adottando come metrica un limite di hop. Il massimo numero di hop consentito è 15, pertanto un hop equivalente a 15 viene considerato come un costo infinito del route (quinid la rotta è inacessibile).

RIP utilizza i distance vector, con aggiornamenti alle tabelle di routing ogni 30 secondi. Le tabelle di routing contengono:

- indirizzo di destinazione;
- numero di hop fino a raggiungimento di destinazione;
- prossimo hop;
- timeout;
- timer per la garbage-collection.

Nella prima versione RIP supportava due tipi di messaggi:

- `REQUEST`, per richiedere informazioni ai vicini;
- `RESPONSE`, per inviare informazioni di routing.

La seconda versione di RIP introdusse nuove funzioni quali il supporto alla poison reverse, indirizzamento CIDR e meccanismi di autenticazione, nonché la possibilità di inserire nel datagramma del protocollo il successivo hop.

RIPng, basato sulla seconda versione di RIP, è invece utilizzato per IPv6. Ha le stesse funzioni di RIPv2 ma non supporta l'autenticazione.

Protocollo OSPF

A causa di scalabilità di bassa qualità si preferisce utilizzare come protocollo di routing intra-AS il protocollo **Open Shortest Path First**. Si tratta di un protocollo link-state che usa flooding per inviare le informazioni dello stato dei collegamenti e che utilizza Dijkstra per la determinazione del percorso minimo.
Ogni router è dotato di un **database link-state** che contiene record link-state. I database vengono aggiornati e sincronizzati tramite **Link-State Advertisement** (**LSA**). Il protocollo individua allora tre fasi:

- una prima fase di discovery dei vicini e di verifica (**HELLO**);
- una seconda fase di scambio di informazioni link-state (**EXCHANGE**);
- una terza fase di propagazione dei cambiamenti (**FLOODING**).

I messaggi scambiati con OSPF sono di cinque tipi:

- **Hello**: è una trasmissione periodica per la neighbor discovery;
- **Database Description**: usato per lo scambio di informazioni link-state;
- **LS Request**: richiesta di parti specifiche da parte di un database link-state di un vicino;
- **Link-State Update**: trasferimento di LSA ai vicini;
- **Link-State Acknowledgments**: ACK per segnalare la ricezione di un link-state update.

OSPF presenta numerosi vantaggi:

- supporto all'autenticazione;
- possibilità di poter utilizzare più percorsi con lo stesso costo;
- supporto integrato per unicast e multicast. Più precisamente, per il multicast si utilizza **MOSPF**, ossia multicast OSPF, che è un'estensione di OSPF che aggiunge un tipo di annuncio nuovo sullo stato dei collegamenti al meccanismo di broadcast;
- **supporto alle gerarchie**. Il sistema autonomo può essere strutturato in maniera gerarchica, configurando l'AS in più aree. In ogni area si individa uno o più **router di confine d'area** che instrada i pacchetti indirizzati all'esterno. Tra tutte le aree se ne individua una detta **area di dorsale**, che instrada il traffico tra le altre aree dell'AS e che contiene tutti (ma non solo) i router di confine del sistema autonomo.

BGPv4

Il **Border Gateway Protocol** (**BGP**) è un protocollo di routing inter-AS che permette la coordinazione tra più sistemi autonomi. Il protocollo permette a ciascun router di:

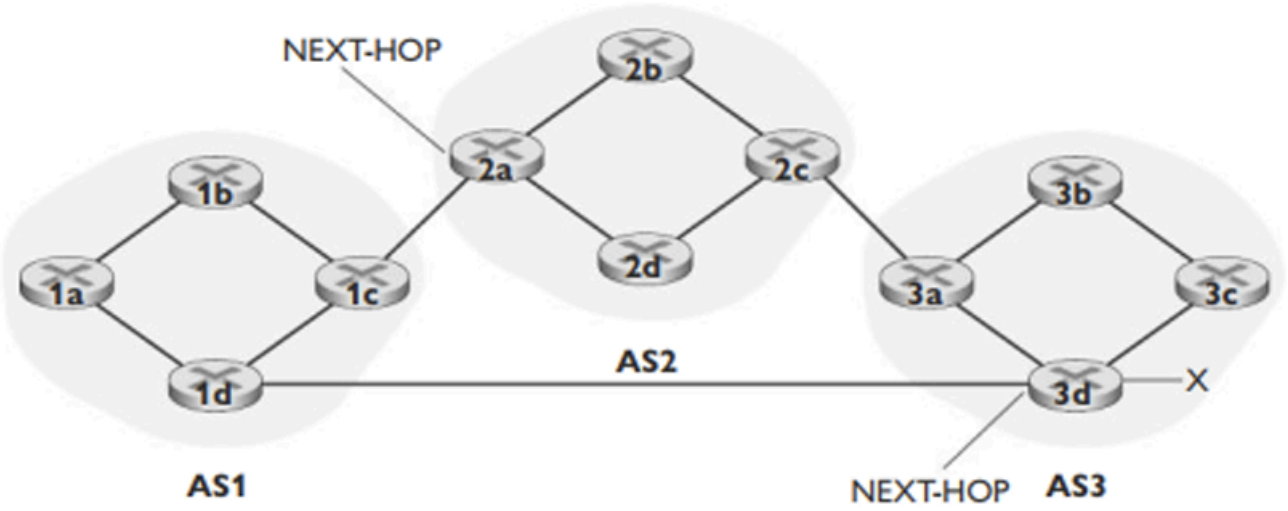
- ottenere informazioni sulla raggiungibilità di **prefissi di sottorete** da parte di AS confinanti. Poiché i record delle routing table infatti corrispondono a destinazioni interne ad un AS, in BGP i pacchetti vengono inoltrati a **prefissi** CIDR che rappresentano una sottorete;
- determinare i percorsi ottimi verso le sottoreti.

Lo scambio di informazioni tra router avviene tramite connessioni TCP, dette **sessioni BGP**. Se la sessione BGP coinvolge due router di AS diversi, la sessione si dice **esterna**; in caso contrario, si dice **interna**. Due router che si scambiano informazioni BGP vengono detti **vicini BGP**. Un router che supporta BGP viene detto **BGP speaker**.

La selezione della rotta migliore avviene tramite lo scambio di prefisso e di **attributi BGP**. Attributi e prefisso insieme formano la **rotta**. Due attributi molto importanti sono:

- `AS-PATH`, che elenca i sistemi autonomi attraverso i quali è passato l'annuncio del prefisso;
- `NEXT-HOP`, che riporta l'interfaccia del router che ha cominciato l' `AS-PATH`. Indica quindi l'indirizzo IP del router che dev'essere utilizzato come next hop verso la destinazione specificata.

Consideriamo ad esempio la seguente situazione:



Supponiamo che 1d voglia comunicare con la sottorete X, localizzata all'interno del sistema autonomo AS3. Per distribuire le informazioni relative alla raggiungibilità del prefisso X, AS3 invia un messaggio BGP ad AS2 con l'annuncio dell'esistenza della sottorete: denotiamo tale messaggio con "AS3 X". Questo messaggio viene inoltrato ad AS1 da AS2: "AS2 AS3 X". Ogni AS quindi riesce anche a conoscere i percorsi per raggiungere X.
Poiché nella realtà sono i router a comunicare, invece di comunicare solamente attraverso quali sistemi autonomi è passato l'annuncio, si comunica con l'attributo `NEXT-HOP` l'indirizzo dell'interfaccia. Di conseguenza, il router 1d conosce due rotte:

- IP dell'interfaccia di 2a, AS2 AS3, X;
- IP dell'interfaccia di 3d, AS3, X.

Tra le rotte comunicate si sceglierà la migliore in virtù del costo.

12 - Data Link Layer

Data Link Layer

Il data link layer è il livello che si occupa da fare da intermediario tra la parte più astratta e la parte più fisica del modello TCP/IP. Questo permette di connettere dispositivi.
Ogni dispositivo è dotato di un indirizzo fisico univoco: il **MAC Address**. L'indirizzo IP infatti viene assegnato per un intervallo temporale finito, individuando a livello logico un host per una sezione di tempo finita. Chi associa MAC Address e Indirizzo IP è il Data Link Layer. Questo vale sia per le reti fisiche sia per le reti wireless.

Servizi

Il data link layer offre un servizio di trasmissione di un **flusso di bit** proveniente dal livello di rete:

- raggruppa i bit dal livello fisico per formare i frame. Questa fase prende il nome di **data framing**;
- si occupa di gestire l'accesso al mezzo fisico, interpretando i segnali elettrici. Poiché inoltre ci possono essere n trasmissioni, riesce anche a gestire i collegamenti broadcast;
- offre una **consegna affidabile** dei dati. Sebbene possa sembrare superfluo, in questo caso gli errori dipendono molto anche a livello fisico:
  - distorsione di segnale dovuto a cali di tensione;
  - il passaggio dei dati tramite collegamenti elettrici (i.e. la corrente nei cavi) usura il canale di comunicazione. Ad esempio, dispositivi che assorbono elevata quantità di corrente risentono particolarmente dell'usura (nota: alcuni cavi possono anche sciogliersi al calore a causa dell'effetto Joule).
  - il passaggio di corrente induce un campo elettromagnetico, e il campo elettromagnetico può indurre anche un campo elettrico, causando possibili interferenze.
    Anche in questo caso quindi la trasmissione dev'essere affidabile;
- gestisce gli errori;
- regola il flusso di dati da sorgente a destinazione.

Il data link layer incapsula il datagramma proveniente dal livello di rete in un **frame**, aggiungendo un header e un trailer. Viene implementata la tecnica di indirizzamento MAC, che permette di identificare i nodi all'interno di una LAN. Se il canale è condiviso, inoltre, si gestisce anche l'accesso al canale di trasmissione. Per il rilevamento degli errori utilizza meccanismi come il checksum o il CRC, eventualmente ritrasmettendo i frame per mantenere la trasmissione affidabile. Ne segue quindi che i servizi del data link devono essere implementati anche nel router.

Il Data link layer deve gestire la consegna affidabile tra nodi adiacenti. I controlli tuttavia sono ridotti se si hanno viaggi su canali robusti. Tuttavia, sulle reti wireless, che sono più **rumorose**, il tasso di errore è più alto.
Il sender incapsula quindi il datagramma in un frame, e aggiunge i bit per il controllo degli errori, per il RDT e per il controllo di flusso. Il receiver invece controlla gli errori ed estrae il datagramma, passandolo ai layer più alti.

Concetto di rumore, attenuazione

Ogni misurazione teorica non è mai uguale al segnale realmente misurato a causa della presenza di **rumore**, una forma di distorsione di segnale. Il rumore può essere causato da molti fattori, quali resistenze, saldature del canale, ecc. L'effetto del rumore viene tamponato tramite dei **filtri**.
Il rumore ha delle componenti di intensità molto basse, ma frequenze altissime. Il rumore tipico delle simulazioni è il rumore **gaussiano** o **bianco**, con una distribuzione a campana.

Le reti wireless sono molto rumorose in quanto si ha un overlap di comunicazioni su bande a frequenza simile che causano rumore. Il data link deve correggere eventuali errori, ad esempio bit mutati da 0 a 1.

Oltre al rumore bisogna anche tener conto dell'**attenuazione di segnale**. Dei vincoli fisici possono causare attenuazione di segnale, che viene tamponata utilizzando dei **ripetitori di segnali**. Il data link allora deve occuparsi della ricezioni di questo tipo di errori, ritrasmettendo il segnale o droppando un frame.

Controllo di flusso

Il data link è il layer che si occupa di inserire e prelevare i dati nel canale. Se non vi è un controllo di flusso, si può facilmente avere un overflow di dati inviati, causando perdite, da cui deriva latenza. Il controllo di flusso tra due data link sono fondamentali per evitare overflow ma anche underflow.

Half duplex, full duplex

Il data link può operare in full duplex o half duplex. In half duplex i nodi non possono trasmettere dati allo stesso momento.

Implementazione del data link sull'host

Ogni host implementa il link layer utilizzando un chip oppure all'interno della **Network Interface Card** (**NIC**). Quest'ultima implementa il data link layer e il physical layer.
La network interface card è una scheda di espansione per il calcolatore. Generalmente il calcolatore comunica con le schede di espansione tramite il **controller** del dispositivo, che permette di semplificare la comunicazione tra sistema operativo e dispositivo.

Oggi le schede madri sono dotate di un chip che fornisce le funzioni del data link e del physical layer. Il supporto del layer di data link è un misto tra hardware, software e firmware.

Data Framing

Un problema nella comunicazione tramite uno stream di bit è la distinzione tra una sequenza di bit 0 oppure di assenza di comunicazione. A livello di trasporto e di rete può essere compreso, ma nel data link risulta più difficile.
Nel data link si cerca di evitare che ci siano porzioni di bit stream che siano sempre dello stesso tipo (quindi solo 1 o solo 0). Infatti in questi casi è difficile capire se il canale è difettoso (ad esempio, è avvenuto un cortocircuito) oppure se si ha una run di 1 o 0. Per evitare questo sono stati introdotti protocolli diversi da quelli classici.

Three Level Encoding

Di particolare importanza sono i protocolli di trasmissione di segnale a multi livelli. Il protocollo **three level encoding** prevede l'utilizzo di tre livelli di segnali:

- alto;
- zero;
- basso.

Il protocollo multilivello segue tre regole principali:

- se il prossimo bit è 0, non si ha transizione;
- se il prossimo bit è 1 e il livello di segnale è diverso da zero, allora il segnale passa a zero;
- se il prossimo bit è 1 e il livello di segnale è uguale a zero, il segnale passa ad un livello che è l'**opposto del livello più recente diverso da zero**. Quindi se prima il livello di segnale per il bit 1 era alto, il successivo livello di segnale è basso.

Poiché le transizioni avvengono solo quando un bit 1 viene trasmesso, una codifica multilivello richiede relativamente una bassa larghezza di banda. Nel caso peggiore si hanno run di bit 1, ma ogni 4 bit si ha un ciclo completo. Nello stesso intervallo temporale quindi si possono codificare più informazioni. Si ha anche una **sincronizzazione del clock**, perché i cambiamenti di livello possono essere sfruttati per recuperare il segnale di clock.
La **clock recovery** viene utilizzata per temporizzare il receiver utilizzando il flusso di dati provenienti dal sender. I flussi di dati infatti vengono inviati in **trasmissione asincrona**, ossia senza un clock, specialmente quando sono inviati ad alta velocità. Il ricevitore deve quindi generare un clock con un oscillatore e sincronizzarlo con il flusso di dati. Affinché ciò sia possibile il flusso di dati deve avere frequenti transizioni per correggere errori dall'oscillatore del receiver. Il receiver dev'essere sincronizzato in quanto si potrebbe campionare il segnale con un tempismo errato: in tal caso la decodifica del segnale sarebbe errata.

La Fast Ethernet (100 BASE TX) utilizza una codifica a tre livelli di segnale: +1V, 0V, -1V. Solitamente i valori non sono statici, ma possono essere cambiati in range di tollerabilità del segnale: ad esempio, +1V individua il range $[1, 1.2]$, 0V individua $[0, 0.2]$ e così via. La Gigabit Ethernet (1000 BASE T) utilizza una codifica a **cinque livelli**: +1V, +0.5V, 0V, -0.5V, -1V. Questo metodo permette di trasmettere due bit contemporaneamente, aumentando la velocità dei dati. Questo meccanismo tuttavia è più sensibile al rumore, in quanto i range sono più stretti.

4B5B Encoding

Il meccanismo del three level encoding non è **DC-balanced**, ossia le cariche elettriche non sono equilibrate sul canale. Se ci sono inoltre lunghe sequenze di zero, si ha un problema di perdita di sincronizzazione.

Una soluzione è la **codifica a blocchi**. I dati inviati sono divisi in blocchi di lunghezza fissa. Ogni blocco è rimpiazzato con una versione leggermente più lunga presa da una tabella di codifica; tale codifica è un pattern unico che garantisce la produzione di un numero minimo di transizioni.\

Una codifica utilizzata per Fast Ethernet è la **codifica 4B5B**, dove ogni blocco di 4 bit viene sostituito con un blocco di 5 bit. Ne segue quindi che si richiede una bandwidth che sia il 25% più capiente.

8B10B Encoding

La codifica 4B5B evita che vi siano lunghe run di bit consecutivi, ma non assicura la DC-balance. Una codifica che assicura la DC balance è la **codifica 8B10B**, garantendo che ogni sequenza di 8 bit diventi una sequenza di 10 bit in cui il numero di bit 1 è uguale al numero di bit 0.

Nella mappatura la sequenza a 8 bit viene divisa in due sottosequenze: una a 5 bit e una a 3 bit. Queste sezioni vengono mappate in modo tale da essere codificate in 6 e 4 bit rispettivamente: concatenate, si ottiene la sequenza di 10 bit. Si specificano inoltre anche 12 caratteri di controllo per indicare inizio o fine di un frame, skip e altre condizioni del link layer.

Quantità di informazioni

Ogni simbolo trasporta un certo numero di informazioni misurabile in bit. Un sistema a quattro livelli utilizza due bit per associare ogni codifica ad un livello. Se abbiamo n livelli, possiamo utilizzare $\log_2 n$ bit.
Se utilizziamo tre livelli, allora utilizziamo un numero di bit pari a $\log_2 3 = 1.5849\ldots$ quindi dobbiamo usare 1 bit.

Supponiamo di avere allora una trasmissione a velocità fissa di 12 **step** per secondo, dove uno step fa riferimento ad un possibile livello. Vogliamo allora capire qual è il **bitrate**, ossia il numero di bit per unità di tempo. Per capirlo, dobbiamo sapere quanti sono gli step per simbolo.
Se si ha uno step per simbolo, allora le combinazioni possibili per gli step sono 3, ma quelle che si possono utilizzare sono solo 2, in quanto utilizziamo un solo bit come visto precedentemente. Si ha un solo bit per simbolo, quindi il nostro bitrate è dato dal numero di step per secondo moltiplicato per il numero di bit. In questo caso, si ha $12 \cdot 1 = 12$ bps.
Se si hanno due step per simbolo, si utilizzano 3 bit per simbolo, in quanto il numero di bit da usare è $\log_2 9$, dove 9 esprime **tutte le combinazioni di livello possibili**. Allora si ha che per unità di tempo si possono inviare $\frac{12}{2} = 6$ step, e ogni step ha 3 bit. Quindi si ottiene un bitrate pari a $6 \cdot 3 = 18$ bps. In generale il **limite teorico di bitrate** è pari a:

$$12 \cdot \log_2 3 = 19.01955001$$

Rilevamento degli errori e correzioni

La error detection individua un errore d itrasmissione in un frame, ma non riesce a correggerlo. Talvolta per correggere l'errore utilizziamo delle informazioni ridondanti. Ad esempio, fornire nome, cognome, data e luogo di nascita, non è necessario trasmettere anche il codice fiscale, in quanto è calcolabile. Questo tuttavia può avere senso come **controllo di errore**: se nella trasmissione infatti qualche valore anagrafico è errato, possiamo renderci conto dell'errore tramite il dato ridondante, che in questo caso è il codice fiscale inviato.

EDC (Error detection and correction bits)

Il metodo più semplice per il rilevamento degli errori sfrutta la ridondanza. Ai dati del payload viene associata un informazione ridondante (i.e. quanti bit 1 ci sono). Trasmettendo quindi l'**EDC** insieme ai dati, ossia i bit di rilevamento e di correzione, il receiver si può rendere conto dell'errore.

Bit di parità

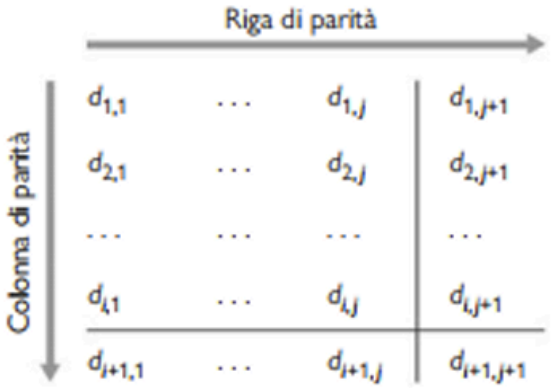
Il più semplice meccanismo dell'EDC è il **bit di parità**. Il bit di parità è uguale a 0 se il numero di bit 1 è pari, 1 altrimenti. Il bit di parità tuttavia fornisce un errore sintattico ma non semantico: il numero di bit potrebbe essere pari dal lato sender e receiver, ma non è sufficiente. Infatti potrei avere le sequenze `111000` e `101010`, e in entrambi i casi il bit di parità è uguale a 1. Si tratta tuttavia di un controllo macroscopico.

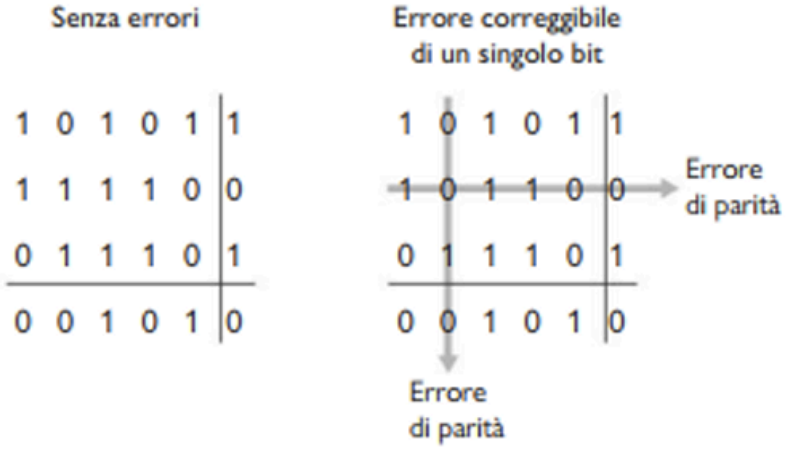
Le RAM ad esempio hanno il controllo parità. Quando trasmette dati attraverso il bus, se il bus è danneggiato o c'è elevata polvere si possono creare capacità passive, che accumulando carica causano rumore ed errori. La RAM allora utilizzava il controllo tramite bit di parità.

Parity checking bidimensionale

Oltre a individuare l'errore la tecnica di parity checking può essere utilizzata per la **correzione**. Possiamo infatti suddividere i bit dei dati in una matrice bidimensionale di i righe e j colonne. Per ciascuna riga e colonna è stato calcolato un valore di parità. I bit in più, che sono di numero $i + j + 1$, contengono bit per la rilevazione dell'errore.

Lo schema illustrato prende il nome di **schema di parità bidimensionale**. I bit di parità della colonna e della riga contenente il bit errato sono in grado di individuare l'errore, permettendo al receiver non solo di individuare l'errore, ma anche di correggerlo:





CRC

La **Cyclic Redundancy Check** (**CRC**) sfrutta l'idea di prendere uno stream da inviare e associargli un **polinomio**. Il polinomio è la struttura algebrica più semplice da gestire e da manipolare. Si prende lo stream di bit e, se n è il numero di bit, si considera un polinomio di grado $n-1$. Si associa poi allo stream di bit un polinomio tale che il termine di grado i, per $i \in [0, n-1]$, ha coefficiente pari al bit i-esimo. Usiamo le operazioni di XOR e di AND tra coppie di bit come somma e prodotto per i termini del polinomio. Generalmente si usa lo XOR per discriminare quando due segnali contemporaneamente vengono immessi in una porta logica: se ciò avviene, la porta viene posta a 0.

Supponiamo di avere le sequenze di bit `10010111` e `01011011`. Le sequenze individuano i polinomi $x^7 + x^4 + x^2 + x + 1$ e $x^6 + x^4 + x^3 + x + 1$. La somma tra i bit porta alla sequenza `11001100`, cioè il polinomio $x^6 + x^4 + x^3 + x + 1$. Il prodotto invece viene calcolato seguendo le regole polinomiali del prodotto. Nel nostro caso, si ottiene il polinomio $x^{13} + x^{11} + x^8 + x^7 + x^5 + x^4 + 1$, corrispondente alla sequenza di bit `10100110110001`. In questo modo si calcolano somma e prodotto dei polinomi individuati dalle sequenze di bit.

Allora sia $M(x)$ il messaggio originale in forma di polinomio. Sia $G(x)$ il **polinomio generatore dello standard di rete** (ad esempio CRC-32). Sia r il grado del generatore $G(x)$ e $R(x)$ il polinomio:

$$R(x) = x^r \cdot M(x) \mod G(x)$$

La divisione è calcolata **senza riporti** e la sottrazione **coincide sempre con l'operazione logica XOR**.
Il messaggio inviato allora è:

$$M'(x) = x^r \cdot M(x) + R(x)$$

e poiché $R(x)$ è il resto di una divisione per $G(x)$, che è di grado r, si ha che $\deg R < \deg G$, per cui nella somma $x^r \cdot M(x)$ e $R(x)$ **non si sovrappongono termini di grado uguale**. Questo garantisce che $M(x)$ è inviato correttamente.

Notiamo anche che $x^r \cdot M(x)$ è uguale a:

$$Q(x) \cdot G(x) + R(x)$$

dove $Q(x)$ è il quoziente della divisione per $G(x)$. Sostituendo questa espressione a quella precedente si ottiene:

$$M'(x) = Q(x) \cdot G(x)$$

che dimostra che **il messaggio è divisibile per il polinomio generatore**. Quindi il criterio utilizzato per verificare se il messaggio è stato inviato correttamente è il seguente:

> *Se il ricevitore divide il messaggio ricevuto per il polinomio generatore e ottiene resto nullo, il messaggio è stato inviato correttamente*.

Vantaggi CRC

CRC è in grado di individuare tutti gli errori a singolo bit e a doppio bit:

- a singolo bit in quanto il polinomio viene modificato in maniera significativa in caso di un bit switch;
- a doppio bit se la distanza tra i bit modificati è minore della lunghezza del CRC. Ad esempio se la lunghezza del CRC è di 16 bit, allora gli errori a doppio bit vengono individuati con una distanza minore di 16 bit.
- errori di bit se questi errori sono in numero dispari, perché si avrebbe un resto diverso per la divisione col polinomio generatore;
- errori su sequenze di bit di lunghezza inferore ad n se n sono i bit del polinomio generatore del CRC.

Per capire l'efficacia di CRC, basta notare che per un CRC a n bit, la probabilità di non identificare errori è pari a 2^{-n} circa. Ma non si tratta chiaramente di un meccanismo foolproof: **alcuni errori specifici, ad esempio che generano resto nullo per la divisione col polinomio generatore, non vengono identificati.** Sono tuttavia errori molto rari.

Il CRC previsto da IEEE 802.3 è il CRC-32, usato in ZIP, PNG, MPEG e altri standard.

13 - Error detection e correction

Correzione errori

Correggere un errore è più difficile di individuarlo, in quanto è difficile distinguere la semantica di un messaggio: in un treno di bit non vi è alcuna semantica.

Anche solo riconoscere l'errore in un bit richiede un approccio di una certa complessità. Si preferisce allora lavorare principalmente sulla **robustezza della trasmissione**, riducendo il rischio di errori. Questo tuttavia non elimina il problema della correzione di errore.

Distanza di Hamming

Introduciamo la **distanza di Hamming**. Date due codeword, definiamo distanza di Hamming tra le due codeword come il numero di bit che sono diversi in stessa posizione. Ad esempio, 1001 e 1000 hanno una distanza di Hamming pari ad 1.
Un vocabolario è un set di codeword. Il vocabolario si dice **completo** se contiene tutte le 2^n codeword possibili.

Consideriamo allora il vocabolario costituito dalle codeword `000000, 000111, 111000, 111111`. La distanza di Hamming in questo vocabolario è 3. La distanza di Hamming di un vocabolario la definiamo come la distanza di Hamming minima tra due codeword all'interno di un vocabolario. La distanza di Hamming può essere utilizzata per correggere gli errori. In generale, per correggere e errori, abbiamo bisogno di un vocabolario con una distanza pari a:

$$d = 2e + 1$$

Per individuare e errori, la distanza dev'essere pari a:

$$d = e + 1$$

La correzione è fatta con una base **probabilistica**. Ad esempio, per correggere un bit abbiamo bisogno di una distanza pari a 3; essendo infatti le codeword più vicine, un bit è più facilmente identificabile.

Ad esempio, supponiamo di avere un vocabolario con le codeword `1111100000, 0000000000, 0000011111, 1111111111`. La distanza di Hamming è 5, quindi possiamo:

- individuare 4 bit di errore;
- correggere 2 bit di errore.

Il numero di codeword ci permette di determinare **quali sono i simboli accettati**. Nell'esempio soprastante sono 4 i simboli accettati. Per cui si utilizzano effettivamente 2 bit per il simbolo e 8 bit per la ridondanza.

I quesiti da risolvere sono i seguenti:

- avendo m bit di dati, quanti bit di ridondanza r sono necessari per la correzione dell'errore?
- partendo da m bit di dati e sapendo come calcolare r, come possiamo costruire un vocabolario per correggere singoli errori?

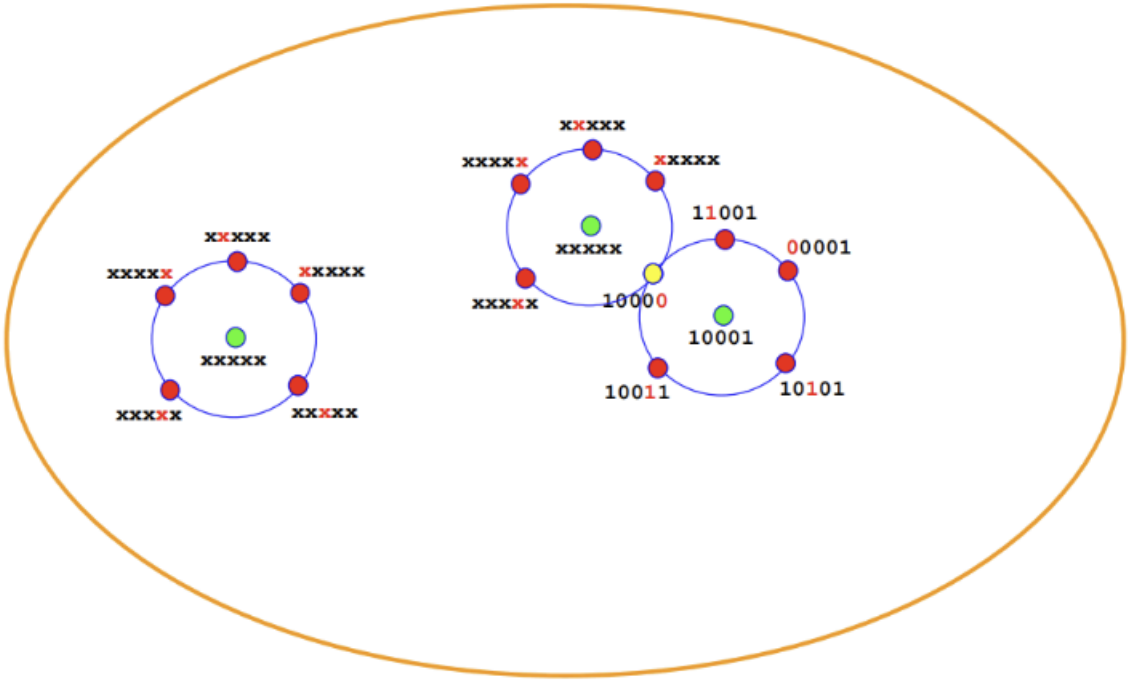
Ridondanza

Consideriamo un vocabolario con m bit di dati e r bit di controllo. Supponiamo che $m + r = n$. Tutte le possibili combinazioni sono 2^n, con 2^m combinazioni valide.

Supponiamo di avere la codeword valida `10001`. Cambiando solo un bit per volta, possiamo costruire altre n codeword, tutte non valide (`00001` , `11001` , `10101` , `10011` , `10000`). Considerando un bit commutato come errore, la distanza di Hamming dei codici originali dev'essere pari a 3 per la formula illustrata precedentemente. Se abbiamo un bit come errore, e ricordando che n sono i bit in una codeword e m i bit di dati, allora è sufficiente scegliere m ed n tali che:

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

Questo perché abbiamo n codeword non valide, 1 sola codeword valida, e questo discorso lo facciamo per 2^m codeword possibili. Ciò vale supponendo che non ci siano situazioni in cui **una codeword errata si può ottenere switchando un bit di due codeword diverse:**



In questo caso, la pallina gialla indica che quella codeword è errata, ma **come facciamo a ricostruire la codeword corretta?** C'è allora un problema nel vocabolario scelto.

Ritornando all'esempio, notiamo che $n = m + r$, da cui deduciamo che:

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

Da notare che la formula illustrata vale solamente con $d = 3$. Se ad esempio $d = 5$, allora dobbiamo essere in grado di correggere due errori. Quindi supponendo di avere 2^m combinazioni valide, possiamo generare codeword errate scegliendo di switchare un bit o due bit. Allora possiamo generare n codeword errate switchando un bit e $\binom{n}{2}$ codeword errate scegliendo due bit a caso. Quindi la disuguaglianza diventerebbe:

$$(n + \binom{n}{2}) \cdot 2^m \leq 2^n$$

Da cui deduciamo che:

$$(n + \frac{(n^2 - n)}{2}) \cdot 2^m \leq 2^n \implies \frac{n^2 + n}{2} \cdot 2^m \leq 2^n$$

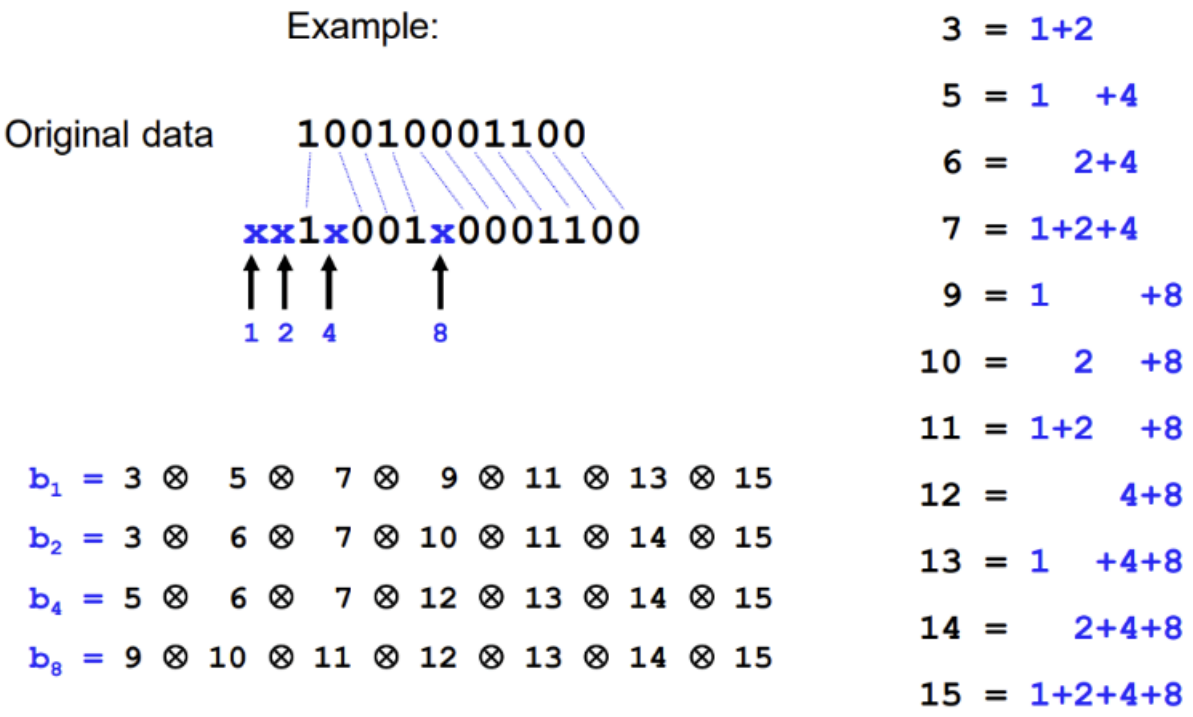
In generale come formula per correggere fino ad e errori basta scegliere m ed r in modo tale che:

$$\sum_{i=0}^{e} \binom{m+i}{r} \leq 2^r$$

Disposizione dei bit di controllo

Ancora non sappiamo utilizzare le codewords per rilevare e correggere gli errori. I bit di controllo vengono posti in posizioni tali da poter ricostruire ogni posizione del bit come somma di posizioni. Ossia, i bit di dati sono posti in posizioni tali che l'indice della posizione si può ottenere come somma di indici di posizione dei bit di controllo. Ad esempio, `10010001100` è la codeword dei dati originali. Inseriamo allora i bit di controllo in posizione 1,2,4 e 8, infatti si può ottenere che il bit dati 3 è in posizione 1+2, 5 è in posizione 4+1, 6 è in posizione 4+2 e così via.
(Nota: in questo caso abbiamo $r = 4, m = 11$.)
Per calcolare i bit di controllo, si esegue lo XOR di tutte le posizioni che coinvolgono il bit di posizione x. Nell'esempio precedente, la posizione 3 si calcola con 1+2, pertanto sia nel bit di controllo 1 che nel bit di controllo 2 si ha uno XOR del bit 3.

Le RAM implementano il meccanismo di error correction code. Generalmente nelle RAM le tecniche di correzione vengono implementate nel caso di errori dovuti a cali di tensione o qualsiasi altro tipo di problema.



Gestione del canale: protocolli a suddivisione di canale

Il canale non è solo di un host, ma ci possono essere diversi host che vogliono comunicare. Bisogna stabilire delle politiche di accesso al canale, che specificano tempismi e occupazione di banda per ogni host connesso al canale.

Per la gestione di accessi multipli, esistono diverse politiche:

- FDMA, ossia Frequency Division Multiple Access. Gli host comunicano tra loro in conversazioni private.
- TDMA, Time Division Multiple Access. A intervalli un host ha accesso al canale.
- CDMA, Code Division Multiple Access. Tutti gli host comunicano tra di loro trasmettendo un codice.

FDMA

La FDMA consiste nella suddivisione della banda di frequenza disponibile in un numero di sottobande che occupano, in frequenza, una banda più piccola. Ognuno di questi sottocanali è assegnato ad un host, che lo utilizzerà per trasmettere le informazioni. Il sistema è semplice e senza sincronizzazione, ma è inefficiente se un utente non ha sempre dati, in quanto in tal caso si ha un sottocanale sprecato.

TDMA

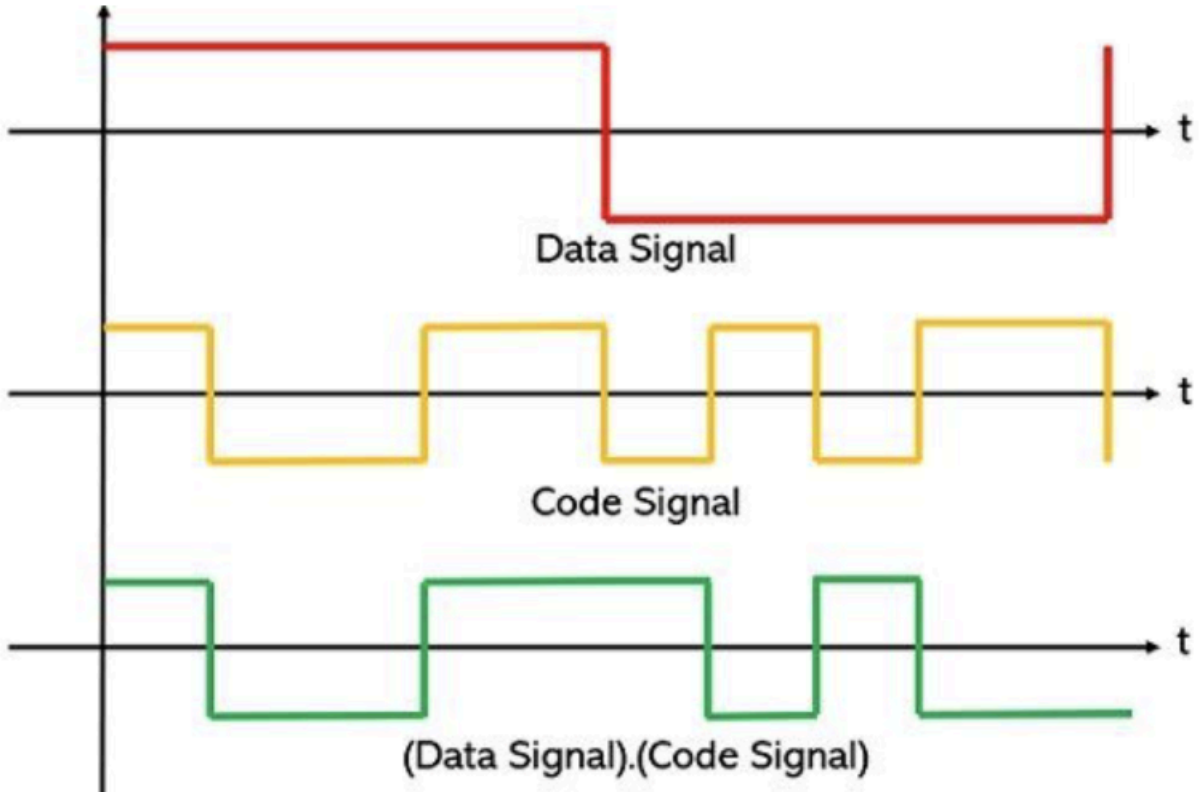
Nel TDMA gli host hanno accesso nel canale per una frazione di tempo determinata. Il tempo viene diviso in slot temporali durante i quali un solo host può trasmettere. Ad ogni host viene assegnato uno slot temporale. Se il meccanismo è ben sincronizzato si tratta di una politica molto efficiente senza sovrapposizioni. Tuttavia, se un host è pronto ma il canale è occupato, bisogna attendere che il canale si liberi: l'host va in idle. Pertanto la TDMA richiede una sincronizzazione stretta tra gli host.

CDMA

Nel CDMA un host invia le informazioni insieme ad un codice, permettendo al receiver di costruire i dati originari inviati utilizzando il codice del sneder.

Il CDMA è una tecnica che sfrutta il principio dello **spettro diffuso**: i segnali vengono modulati in modo tale da convertire il segnale originale in un segnale di ampiezza di banda maggiore. Ogni host allora utilizza un codice unico, e il destinatario utilizza il rispettivo codice per reperire i dati correttamente.

Per combinare i messaggi con il codice solitamente si utilizza un approccio noto come **direct sequencing**. Per la trasmissione del segnale l'host sender combina il codice univoco con i dati da inviare, modulando il segnale binario dei dati con il segnale del codice, come segue:



L'host ricevente demodula il segnale utilizzando lo stesso codice che ha usato l'host sender. I dati che riceve l'host sono esprimibili con la formula:

$$Z_{i,m} = d_i \cdot c_m$$

dove d_i indica i dati inviati e c_m il valore dell'm-esimo bit del codice. Il receiver può ricostruire il dato calcolando:

$$d_i = \frac{1}{M} \sum_{m=1}^{M} Z_{i,m} \cdot c_m$$

Da notare che anche se ci sono interferenze o più sender, i bit inviati dai diversi sender si sommano. Tuttavia, se un sender invia un dato codificato Z^s, allora il receiver può comunque ricostruire la sequenza inviata da quel sender specifico utilizzando quei dati codificati. Questo tuttavia richiete che il sender scelga opportuni codici.

Gestione delle collisioni: protocolli ad accesso casuale

Altri protocolli utilizzati per l'accesso multiplo sono i **protocolli ad accesso casuale**. I protocolli ad accesso casuale vengono utilizzati quando nodi trasmettono alla massima velocità consentita sullo stesso canale, e gestiscono situazioni in cui pacchetti trasmessi nello stesso canale alla stessa frequenza **collidono**. Quando i pacchetti collidono, non si può dedurre cosa riceva il destinatario. In questi casi il nodo ritrasmette dopo un tempo random.

Pure ALOHA

Il più semplice protocollo è il **pure ALOHA**. Se si ha un frame pronto, si trasmette subito senza preoccuparsi che il canale sia occupato. Ovviamente questo aumenta significativamente la probabilità che si verifichino collisioni, poiché ogni host opera nello stesso modo.
Si fissa un tempo entro il quale bisogna ricevere un ACK. Il trasmittente aspetta un tempo pari ad un RTT in attesa di ricevere un ACK. Se l'ACK non è stato ricevuto, vuol dire che c'è stata una **collisione**. In caso di collisione si fissa un tempo random all'interno di un certo range e ritrasmette il pacchetto andato in collisione dopo il tempo random. Potrebbe accadere che l'ACK può collidere con un altro ACK: in tal caso si ritrasmette con un nuovo tempo random.

In generale si può mostrare che Pure ALOHA non è un protocollo molto efficiente: infatti esso utilizza con il meccanismo di time slot un massimo pari al 18.4% di banda disponibile.

Slotted ALOHA

In alternativa al Pure ALOHA si utilizza uno **slotted ALOHA**. Il tempo viene diviso in **slot** di durata pari alla durata di un frame. Tutti gli host dell'insieme condividono una stessa "griglia" temporale. L'importante è inviare il frame all'inizio del primo slot utile, ma non vi è alcun carrier-sense, ossia non si ha allineamento con la fine di uno slot. L'allineamento con il primo slot utile avviene tramite due tecniche:

- sincronizzazione di clock;
- header.

Quando non vi è il carrier-sense, non si ha alcun meccansimo che misura il livello di tensione del canale per capire se il canale è libero o meno. Per cui si verificano più facilmente le collisioni.

Quando un nodo ha un frame pronto, si aspetta l'inizio dello slot successivo. Se il frame è ricevuto corettamente, il receiver invia un ACK entro lo stesso slot o in un mini-slot dedicato. Se si rileva una collisione (ossia non si riceve l'ACK), il nodo sceglie un intero k in $[0, K-1]$ e ritenta dopo k slot. Viene aumentata così la robustezza, in quanto si rimuove la possibilità di overlapping, ma si introducono latenze. Si potrebbero diminuire le latenze scegliendo in maniera ottimale il valore di k.

Il massimo valore K per la scelta del valore random cresce dopo ogni collisione: se un host rileva una collisione, allora deve aumentare il numero di time slot che si fanno passare perché il time slot atteso non è sufficiente.

I vantaggi nell'utilizzo dello slotted ALOHA sono i seguenti:

- un singolo nodo può trasmettere continuamente alla massima velocità del canale;
- è decentralizzato, in quanto solo gli slot nei nodi devono essere sincronizzati;
- è semplice da implementare.

Ci sono tuttavia i seguenti svantaggi:

- le collisioni causano perdite di slot;
- alcuni slot possono essere in idle;
- i nodi potrebbero individuare una collisione non necessariamente aspettando il time slot impostato;
- si ha la necessità di clock synchronization.

In generale i meccanismi che sfruttano slot temporali sono efficienti in virtù della frazione di **slot riusciti**, ossia slot in cui un nodo è in grado di trasmettere: la perdita di slot quindi riduce l'efficienza del protocollo. Nel caso di slotted ALOHA, si può dimostrare che la banda di rete utilizzata è pari al 37% un miglioramento rispetto a pure ALOHA, ma ancora insufficiente.

14 - CSMA, Ethernet

CSMA

Il **CSMA**, ossia **Carrier-Sense Multiple Access**, consente la gestione dei casi di collisione in tre modi:

- **1-persistent**. Se il canale è libero, trasmette il frame immediatamente. Il nodo misura la tensione nel canale per capire se un pacchetto è in trasmissione o meno. Se viene rilevata che la tensione è al di sotto di una soglia, detta **squelch threshold**, allora si può usare il canale per la trasmissione. Se il canale invece è occupato, aspetta che si libera e poi trasmette subito. Il meccanismo ha un'alta probabilità tuttavia di collisione se molti nodi attendono contemporaneamente che il canale si liberi;
- **p-persistent**. La trasmissione avviene con probabilità p, mentre con probabilità $1-p$ si aspetta il prossimo slot di tempo e ripete la procedura. Nel momento della trasmissione quindi un host ha probabilità p di trasmettere nel canale, e $1-p$ probabilità di non trasmettere e di rimanere in attesa del prossimo slot. La trasmissione di più nodi contemporaneamente avviene con più difficoltà, migliorando le performance di 1-persistent;
- **n-persistent**. Se il canale è occupato l'host attende un tempo casuale prima di ritentare l'accesso. Se il canale è libero trasmette subito. Si riduce la congestione ma si introducono latenze, in quanto attendendo un tempo casuale quando il canale è occupato, a differenza di 1-persistent dove appena il canale è libero si trasmette.

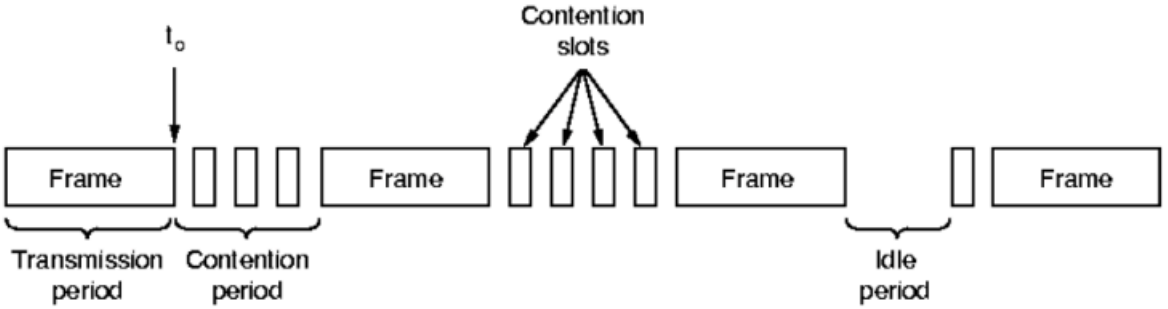
Uno tra i migliori CSMA a livello di throughput è il CSMA 0.01-persistent.

CSMA-CD

Il CSMA non rimuove la possibilità di collisioni, per cui spesso viene esteso con **CSMA-CD**, ossia CSMA con **Collision Detection**. Il protocollo non solo ascolta il canale prima di trasmettere ma si accorge delle collisioni durante la trasmissione.

Un nodo decide di iniziare la trasmissione, caricando il frame sul bus o sul mezzo di comunicazione del canale fisico. Quando comincia la trasmissione, se il nodo rileva del rumore o di segnali disturbati allora c'è stata una collisione. Il nodo invia in tal caso un **segnale di jamming con backoff**, che informa tutti gli host della collisione. Si aspetta allora un tempo casuale e poi si ritenta.

Per ridurre le collisioni, CSMA-CD adotta un meccanismo di **divisione temporale del periodo di contesa del canale**. Essenzialmente quando i nodi rilevano una collisione **annullano la propria trasmissione** e scelgono uno slot temporale casuale tra quelli disponibili: se due o più nodi scelgono lo stesso slot, si verifica una collisione. Il tempo di attesa prima della ritrasmissione prende il nome di **periodo di contesa del canale**.



La scelta del **tempo di backoff**, ossia dell'intervallo di tempo da attendere, dev'essere importante:

- se l'intervallo è piccolo è possibile che valori scelti casualmente siano simili in una rete di tanti nodi;
- se l'intervallo è troppo grande, i nodi possono aspettare un tempo molto grande e lasciare il canale inattivo.

Per la scelta del tempo di backoff, la IEEE prevede l'utilizzo dell'algoritmo di **binary exponential backoff**, ossia si sceglie un valore di tempo compreso tra 0 e 2^i-1 e lo si moltiplica per il tempo di uno slot (solitamente il tempo di propagazione di un certo numero di bit. In Ethernet è il tempo per la trasmissione di 512 bit). Per ogni collisione i aumenta di uno fino ad arrivare a 10. Alla 16esima collisione viene comunicato un errore di trasmissione.

Nella trasmissione di pacchetti i payload devono essere caratterizzati da livelli di segnali molto precisi. Quando si verifica una collisione, l'overlap di tensione e corrente è un segnale assolutamente privo di caratterizzazione elettrica, con una corrente che

individua una **noise burst**.

Protocolli senza collisione

Protocollo bit map

Il tempo viene diviso in mini slot di durata di 1 bit-time. Ad ogni stazione è associato un ID univoco che va da 0 a $n-1$ dove n è il numero di stazioni. **Una stazione che ha un frame pronto trasmette un singolo bit 1 nel minislot corrispondente al proprio ID**. Chi non ha nulla da trasmettere invia 0.

Alla fine della trasmissione si avrà una **bitmap** che permette di **far conoscere a tutte le stazioni chi ha qualcosa da trasmettere o meno**. Si attende un tempo t **superiore al tempo di propagazione massimo** in modo tale che tutti i nodi abbiano ricevuto la bitmap correttamente. In ordine crescente di numero identificativo, ogni stazione trasmetterà il proprio frame, **senza collisioni**.

L'approccio è chiaramente poco scalabile.

Binary backward counting

Un'altra alternativa è il **binary backward counting**, utilizzato ad esempio in CAN bus. Il CAN bus è uno dei primi connettori a bus utilizzati in sistemi embedded, specialmente utilizzato nella robotica. Il CAN bus è un meccanismo robusto che individua le collisioni in maniera efficace, per cui trova molte applicazioni nel settore automobilistico.

Il meccanismo di binary backward counting assume che **le latenze di trasmissione siano trascurabili, in modo tale che ogni host può vedere i bit inviati nel canale in maniera istantanea**. Il tempo viene diviso in slot di 1 bit-time. Ad ogni host è associato un ID binario. Ad ogni slot time ogni host invia nel canale un bit del suo ID, partendo dai bit più significativi ai bit meno significativi. Se un host trasmette zero ma legge uno, allora cede il canale. Quindi se ad esempio quattro host inviano `0010, 0100, 1001, 1010`, allora i primi due host cedono il canale, in quanto il bit più significativo degli altri due host è posto ad 1 e quindi è dominante; pertanto rimangono solamente gli ultimi due. A questo punto si confrontano i successivi bit per decidere quale host invia il frame nel canale.

Il metodo elimina le collisioni, ma soffre di alcuni problemi. Se due messaggi sono uguali, il primo che ha messo i bit passa il frame. Inoltre non segue realmente i meccanismi di priorità. Infine, se dovesse capitare che qualcuno dei nodi scrivesse un valore che non è né zero né uno perche viene attenuato dal canale, il meccanismo di binary backward counting non sa come interpretare questo segnale. Questa situazione manda in tilt il sistema.

Taking turns protocol

Nel protocollo taking turns ogni nodo manda 1 nel canale per segnalare di voler trasmettere i frame nel canale. I nodi che hanno inviato il bit 1 si inseriscono in un **logic ring**, ossia un anello logico, e si definisce quali tra questi nodi è il primo a trasmettere. Al nodo che deve trasmettere viene assegnato un **token** con un tempo di trasmissione massimo. Il token poi viene passato agli host che appartengono al ring.

Chiaramente non è un sistema privo di problemi:

- se un host del ring va in guasto, il meccanismo di token-passing non funziona;
- un host che decide di trasmettere **non si può unire al token ring e non potrà mai connettersi al canale per trasmettere i suoi frame**.

Il meccanismo taking turns allora opera bene quando includiamo un'intera subnet nel meccanismo di token passing.

Ethernet

La IEEE è l'ente di standardizzazione importante per l'ingegneria elettronica. Si occupa anche di standardizzare alcuni protocolli, in particolare alcuni standard a livello fisico sono deifniti in IEEE 802. L'802.3 stabilisce le specifiche per una corretta implementazione di CSMA/CD, mentre l'802.11 stabilisce le specifiche per le wireless LAN.

IEEE 802.3

Nell'802.3 vengono stabiliti anche i tipi di encoding da utilizzare per la trasmissione di segnali, in virtù anche della velocità di trasmissione e del sistema da implementare. Solo chi implementa lo stesso standard di comunicazione può ricevere correttamente i nostri messaggi.

Ethernet CSMA/CD

Nell'IEEE 802.3 è anche prevista la definizione del protocollo CSMA-CD. L'Ethernet riceve il datagramma dal layer di network, e crea un frame. Successivamente se ethernet vede che il canale è in idle comincia la trasmissione, altrimenti attende fino a quando il canale è in idle. Se si individua un'altra trasmissione viene inviato un segnale di jam e si entra in una **binary (esponenziale) backoff**:

- dopo la m-esima collisione, si sceglie un valore K nell'intervallo di interi che va da 0 fino a $2^m - 1$. Si attende allora $K \cdot 512$ tempi di bit, e si verifica se può essere eseguita la trasmissione;
- più sono le collisioni, maggiore è l'intervallo di backoff.

Il livello fisico contiene comparatori ad alta impedenza. Se la tensione differenziale dall'esterno supera la soglia di **squelch**, il CRS (Carrier Sense) sale a 1 e il MAC capisce che qualcuno sta già trasmettendo. Un canale fisico ad alta impedenza è tale che la resistenza impedisce il passaggio di corrente. Quando il carrier sense sale a 1 quindi tutti i nodi capiscono che non possono trasmettere nel canale.

Tecnologie ethernet

Le tecnologie standardizzate si presentano con la seguente denominazione:

- un numero, che indica la velocità dello standard (10, 100, 1000, 10G, ecc.);
- BASE, che riferisce la fatto che il mezzo fisico trasporta solo traffico Ethernet;
- un'etichetta che fa riferimento al mezzo fisico, ad esempio "T" si riferisce ai doppini in rame.

Storicamente ethernet venne concepita come un segmento di **cavo coassiale**, cavo composto da un singolo conduttore di rame posto al centro (detto **anima**) e da un dielettrico, isolante di corrente elettrica che separa l'anima da uno schermo di metallo che blocca interferenze. Il cavo coassiale viene utilizzato per gli standard 10Base5 e 10Base2, che rpevedono una connessione via cavo per l'ottenimento di una rete locale e l'utilizzo di Ethernet CSMA/CD. Tuttavia:

- 10Base5 utilizza un cavo coassiale spesso, supportando una lunghezza massima per cavo di 500 metri, connettendo fino 100 stazioni per segmento;
- 10Base2 utilizza un cavo coassiale snello, supportando una lunghezza massima di 185 metri, connettendo fino a 35 stazioni per segmento e senza la necessità di un hub.

Il 10Base5 venne successivamente sostituito dal 10Base2. Entrambi poi vennero sostituiti dal 10BaseT, che utilizza un cavo **UTP** (**Unshielded Twisted Pair**). 10BaseT supporta sempre 10Mbps, ma è un sistema molto economico e con una portata di 100m per cavo. Il 10BaseF, invece, è basato sulla **fibra ottica** ed è molto buono in quanto ogni segmento può raggiungere una lunghezza di 2km e quindi è la soluzione migliore per collegare router di edifici diversi.

Negli anni '90 Ethernet fu standardizzato a 100Mbps, e vennero introdotti:

- 100Base-T4 e 100Base-TX, che utilizzano cavi twisted pair con una lunghezza massima per segmento di 100 metri. Il vantaggio principale di T4 era il costo, perché utilizza un UTP di categoria 3, mentre TX permette full duplex. Inoltre, T4 utilizza un encoding 8B6T, mentre TX 4B5B;
- 100Base-FX, in fibra ottica e con una lunghezza massima di 2k.

Un'estensione degli standard Ethernet è Gigabit Ethernet, detto anche IEEE 802.3z, che offre una velocità di trasmissione di 1Gbps. Gli standard Gigabit sono:

- 1000Base-SX e 1000Base-LX, che utilizzano fibre ottiche. La differenza tra i due è che il primo è utilizzato per trasmissioni su cavi corti (massimo 550 metri) utilizzando una fibra multimodale (cioè che consente di trasferire più segnali contemporaneamente), mentre LX è usato per le trasmissioni con distanze più lunghe (5000 metri massimo);
- 1000Base-CX, che utilizza due coppie di **STP** (**Shielded Twisted Pair**). Supporta un segmento di lunghezza massima di 25 metri;
- 1000Base-T, che utilizza quattro coppie di UTP e una lunghezza massima di 100 metri. Il vantaggio è che l'UTP è più economico rispetto ad altri cavi. Inoltre:
  - non utilizza 4B5B;
  - utilizza una trasmissione full duplex;
  - utilizza un encoding a 5 livelli invece di 3;
  - introduce un **forward error correction**.

Tipologie di cavo

Il modo in cui è realizzato un cavo influenza le capacità di trasmissione. Principalmente ci sono alcuni fattori da tenere in considerazione:

- il passaggio di corrente induce un **campo magnetico**. A sua volta un campo magnetico può indurre un campo elettrico e quindi indurre una corrente. La corrente indotta ha un verso che è l'opposto rispetto al verso della corrente che scorre nel filo. Ne segue quindi che la corrente indotta si oppone al passaggio di corrente. Inoltre, anche cavi adiacenti possono causare interferenze. Questo tipo di interferenze prende il nome di **diafonia** (nel caso di cavi si chiama anche **crosstalk**);
- più è lungo il cavo, maggiore è la resistenza. Quando vogliamo trasmettere un segnale ma la distanza è maggiore della portata del cavo, è necessario utilizzare dei **ripetitori**.

I cavi solitamente presentano alcune componenti fondamentali, quali la **jacket**, ossia il materiale isolante, e i cavi conduttori. Non tutti i cavi hanno la foil, ossia la schermatura del cavo, realizzata con un materiale apposito per impedire le interferenze.
I cavi più semplici sono gli **UTP** (**Unshielded Twisted Pair**). UTP è un cavo costituito da coppie di fili di rame intrecciati non schermati. L'intreccio dei fili di rame ha lo scopo di ridurre le interferenze e il crosstalk. UTP è presente in diverse categorie in virtù della capacità di trasmissione e di numero di intrecci.
Successivamente ci sono i cavi **FTP** (**Foiled Twisted Pair**), che presenta una foil che protegge tutti i cavi, ma non le singole coppie. Invece, i cavi **STP** (**Shielded Twisted Pair**) utilizzano la foil su ogni singola coppia di cavi. **SFTP** (**Shielded Foiled Twisted Pair**) utilizzano la foil sia su ogni singola coppia sia sul gruppo di coppie, e garantiscono una massima protezione dalle interferenze a fronte di maggiori costi.

Indipendentemente dalle schermature e dalla composizione del cavo distinguiamo i cavi di rame in diverse categorie:

- Cat5, che supporta fino a 100Mbps e opera sui 100Mhz di frequenza;
- Cat5e (Cat5 enhanced), che segue maggiormente gli standard IEE e supporta fino a 1Gbps, operando sempre su 100Mhz di frequenza;
- Cat6, che supporta fino a 10Gbps e opra sui 250Mhz di frequenza;
- Cat6a, che supporta fino a 10Gbps, ma con maggiori distanze e sulla frequenza di 500Mhz.

RJ-45

Ogni cavo termina con dei connettori di tipo **RJ-45** schermati. I connettori vengono innestati nell'interfaccia del dispositivo e sono del tipo 8P8C, ossia 8 posizioni 8 contatti. Per il connettore esistono due chemi di cablaggio, ossia T568A e T568B, che differiscono solo per un'inversione delle coppie 2 e 3. Solitamente per gli impianti nuovi si preferisce il T568B in quanto ammesso dallo standard ANSI/TIA.

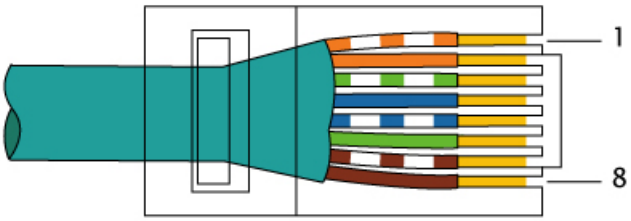
Ad ogni pin corrisponde un cavo, secondo la tabella (nota: si tratta dello schema di colori per T568B) sottostante:

Pin	Colore	Utilizzo
1	bianco/arancione	Trasmissione (Tx+)
2	arancione	Trasmissione (Tx-)
3	bianco/verde	Ricezione dati (Rx+)
4	blu	Bidirezionale dati (Bi3+)
5	bianco/blu	Bidirezionale dati (Bi3-)
6	verde	Ricezione dati (Rx -)
7	bianco/marrone	Bidirezionale dati (Bi4+)
8	marrone	Bidirezionale dati (Bi4-)

I simboli '+' e '-' indicano la polarità del segnale nel canale. Supponendo ad esempio che un bit 1 viene trasmesso a 5 volt, viene trasmesso in un canale '+' a +5 volt e nel corrispondente canale '-' a -5 volt. Avere i segnali con polarità opposta in un twisted pair permette di ridurre il rumore, in quanto la differenza di voltaggio tra i due segnali equivale al doppio del voltaggio del segnale da trasmettere, assicurando robustezza e ricezione del segnale.

I cavi inoltre si distinguono in **cavi diretti** e **cavi incrociati**:

- quando la disposizione dei doppini è uguale in entrambe le estremita del cavo si ha un **cavo diretto**. I cavi diretti vengono utilizzati per collegare dispositivi non simili, ad esempio il computer con un router;
- quando i doppini incrociano le loro posizioni alle estremita si ha un **cavo incrociato** o **crossover**. I cavi incrociati vengono utilizzati per collegare due pc o due router e funge quindi da prolunga.



EIA/TIA-568B

\

Frame Ethernet

Il frame ethernet è costituito da:

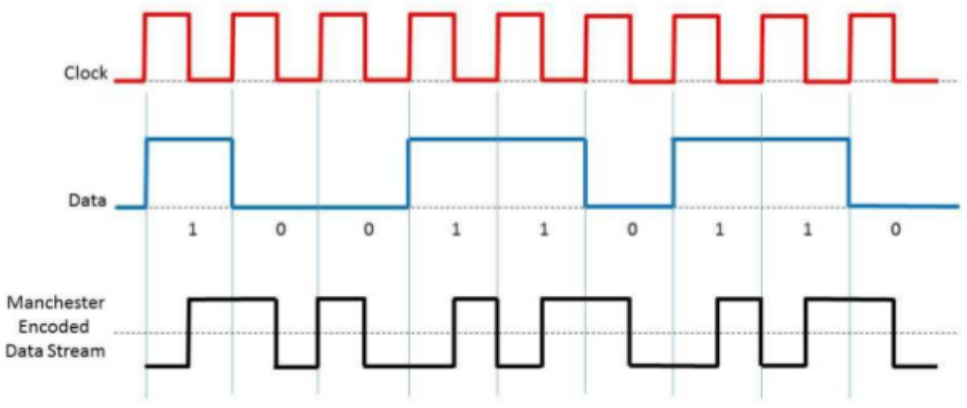
- **preambolo**, a 7 byte. Ogni byte è pari a 10101010 per sincronizzare il clock della scheda di rete del ricevente con quella del trasmittente;
- **start frame delimiter**, 1 byte. Il byte è la sequenza 10101011 per indicare l'inizio del frame e rompere il pattern di bit del preambolo;
- **destination MAC address**, 6 byte;
- **source MAC address**, 6 byte;
- **type / length**, 2 byte. Identifica il tipo di connessione (EtherType, IPv4, IPv6, ARP, ecc) se il valore è superiore a 1500. Se invece il valore è inferiore a 1500, identifica la lunghezza del data field;
- **data field**, fino a 1500 byte e minimo 46 byte. Se non si raggiunge la dimnesione minima, si utilizza un padding, mentre se si supera la dimensione massima il frame viene frammentato;
- **frame check sequence**, a 4 byte. Si tratta del controllo CRC per rilevare l'errore nel frame.

La dimensione dell'header è

Codifica di Manchester (10Base*)

Per combinare l'informazione del clock con i dati in un'unica trasmissione si utilizza la **codifica di Manchester** (che prende il nome dall'università di Manchester in cui fu ideata). Ogni bit non viene codificato come un singolo segnale alto o basso, ma come un segnale basso seguito da uno alto, o alto seguito da uno basso.

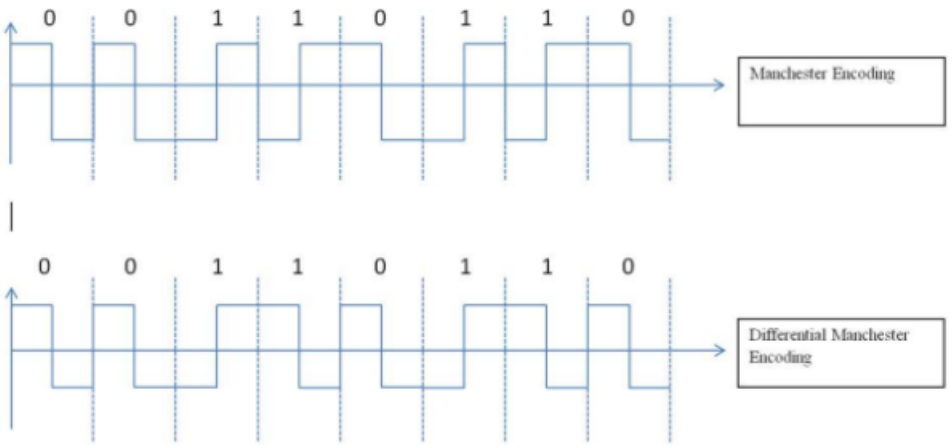
Nella codifica di Manchester i dati controllano la fase di un carrier a onde quadre, mantenendo sempre una transizione al centro di ogni periodo di bit. Nella pratica, quando bisogna trasmettere un bit, esso viene codificato in modo tale da avere una transizione. IEE 802.3 prevede che si rappresenti il bit 0 con un segnale alto seguito da un segnale basso, e il bit 1 come un segnale basso seguito da un segnale alto. A livello di codifica questo si traduce in un'operazione di XOR tra il segnale del clock e il segnale del dato da trasmettere. Poiché ogni dato viene trasmesso ad ogni rising edge del clock, si ottiene la trasmissione in foto:



Il problmea della codifica di Manchester risede nell'**importanza della polarità del segnale trasmesso**, pertanto se il segnale codificato viene invertito allora bisogna invertire il modo in cui si codifica il dato. L'ambiguità viene risolta utilizzando una **codifica di Manchester differenziale**.
Nella codifica di Manchester differenziale utilizza la presenza o l'assenza di transizioni per indicare il valore logico. L'idea generale è quella di rappresentare l'informazione non tanto nel voltaggio del segnale, bensì nella transizione. Tale transizione è identificata ad ogni rising edge del segnale di clock. IEEE 802.5 specifica la codifica di Manchester differenziale per le LAN a token ring:

- ogni bit 0 è identificato da una transizione;
- ogni bit 1 è identificato dall'assenza di transizione.

La codifica allora diventa come in foto:



Come si osserva, durante la trasmissione dei bit 1 il segnale non presenta transizioni. Si garantisce quindi sempre una transizione per bit ma non si è dipendenti dalla polarità del segnale.

15 - Gigabit, dispositivi di link

Gigabit Ethernet

Il protocollo che gestisce l'ethernet Gigabit è IEEE 802.3:

- 802.3z quando è su fibra;
- 802.3ab quando è con cavi di rame twisted pair.

Lo standard gigabit rimuove l'encoding 4B5B. Questo encoding infatti crea un overhead del 25%, in quanto trasmettiamo un bit in più ogni 4 bit. Rimuovendo 4B5B rimuoviamo questo overhead. Si passa così da 100 Mbps a 125 Mbps.
Gigabit inoltre utilizza tutte e 4 le coppie allo stesso momento **come canali bidirezionali**. Utilizza inoltre una **trasmissione full duplex**, cioè trasmette e riceve contemporaneamente. Ricezione e trasmissione avvengono su canali separati, pertanto non si hanno problemi di collisioni, per cui CSMA/CD non viene utilizzato.
Gigabit non utilizza MLT-3, usata per fast Ethernet, bensì la **PAM-5**. La PAM-5 utilizza 5 livelli per baud invece di 3, dove i livelli sono -2, -1, 0, 1, 2. Poiché utilizziamo quattro canali di trasmissione, abbiamo quindi che per ogni canale possiamo trasmettere un totale di combinazioni di segnali pari a $5^4 = 625$ combinazioni. Poiché $\log_2(625) \approx 9$, si ha che trasmettiamo:

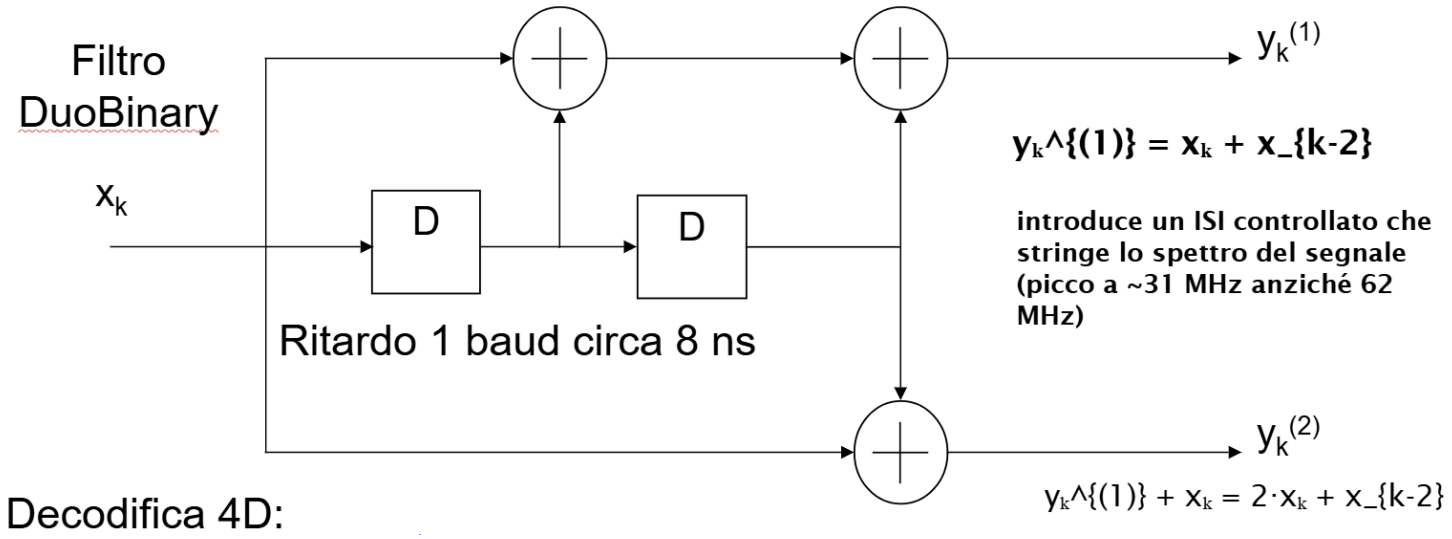
$$125 \cdot 10^6 \text{ baud} \cdot 9 \text{ bit} \approx 1 \text{ Gbit lordi}$$

Per quanto riguarda il frame, Gigabit mantiene la retrocompatibilità:

- la MTU è 1500 byte;
- c'è il preambolo a 8 byte per la sincronizzazione;
- ci sono 12 byte alla fine del frame per l'**interpacket gap**. Se non si dà un gap avvengono delle **sbavature** di frequenze, ossia un tipo di interferenza del segnale. Le interferenze di segnale, e quindi i bit switch, vengono gestiti tramite il **forward error correction**.

Codifica 4D-PAM5, Duo Binary e decodifica Viterbi

A causa dell'alta velocità di trasmissione, gli impulsi dei simboli trasmessi possono sovrapporsi con quelli vicini creando delle interferenze. L'encoding dei dati in gigabit non avviene semplicemente usando PAM-5, ma usando **4D-PAM5**. Lo schema opera codificando per ogni simbolo due bit di dati per canale, per un totale contemporaneo di $2^8 = 256$ word di bit valide possibili: infatti supponendo di avere uno step per simbolo, possiamo inviare 2 bit ogni volta perché abbiamo 5 livelli possibili, e in totale abbiamo 4 canali che possiamo usare simultaneamente.
Tuttavia abbiamo visto che con PAM5 in quattro canali abbiamo $5^4 = 625$ combinazioni possibili. Questo permette a 4D-PAM5 di avere **ridondanza** e di poter introdurre meccanismi di Forward Error Correction. Per identificare le combinazioni corrette tra le 625 combinazioni possibili si usa un **codificatore convoluzionale a stati**. Essenzialmente si utilizza un **algoritmo di codifica convoluzionale a 4 dimensioni in stile Trellis**, in modo tale che **la sequenza corretta è codificabile solo tramite una serie finita di transizioni tra simboli**. Le dimensioni fanno riferimento al numero di stati della **macchina a stati finiti** da cui si ricava il Trellis.
Per ridurre le interferenze ad alte velocità si può restringere lo spettro del segnale tramite un **filtro DuoBinary**:



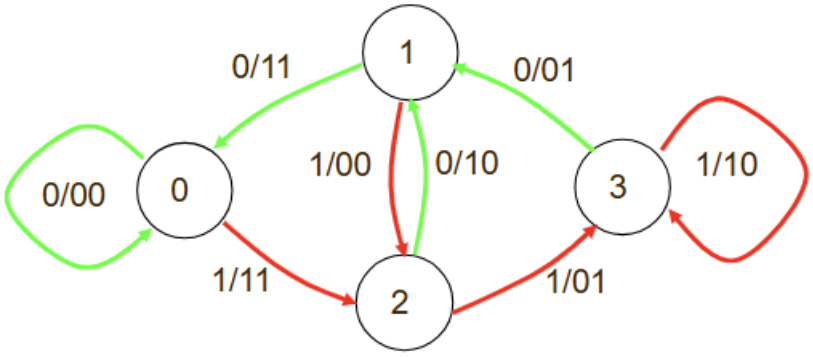
Decodifica 4D:

Il segnale di partenza x_k viene processato da due filtri, dei **delay**. Ogni delay introduce un ritardo di 8 nanosecondi. Il segnale inviato inizialmente x_k viene processato producendo due segnali risultanti:

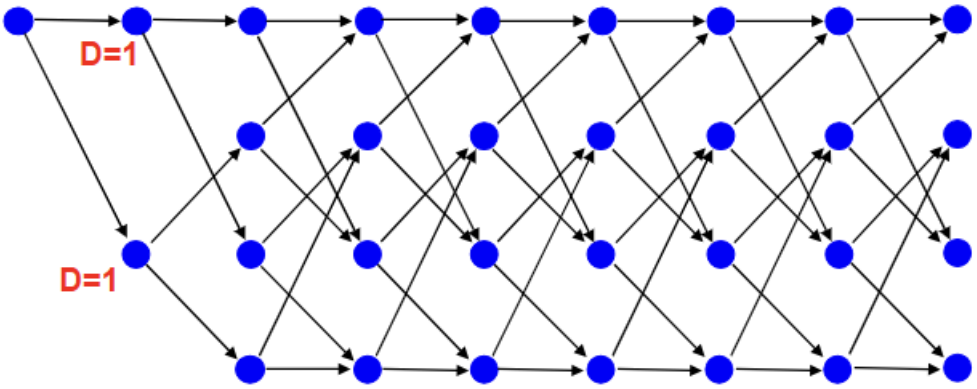
- il segnale $y_k^{(1)}$ è ottenuto sommando il segnale x_k al segnale x_{k-2}, ossia il segnale che riceve due delay da 8 nanosecondi;
- il segnale $y_k^{(2)}$ è ottenuto sommando il segnale $y_k^{(1)}$ al segnale x_k.

Sostanzialmente si inserisce un **ISI controllato**, in modo tale che prima che parta il secondo frame si possa controllare l'interferenza. Il ricevitore applicherà una **equalizzazione** per ottenere la stima dei simboli trasmessi. Tuttavia è possibile che **il segnale trasmesso possa contenere errori**. Per ricostruire il messaggio corretto il receiver usa un **decodificatore Viterbi**.
Da notare che in questo caso parliamo di **ricostruzione del messaggio**, ma non di **correzione dell'errore**, per il fatto che **la decodifica Viterbi si basa su una correzione per verosimiglianza**. Nella decodifica Viterbi si va a seguire lo stesso **grafo degli stati finiti**, ossia il **trellis**, che viene usato durante la codifica. Il messaggio che viene ricostruito è quello più verosimile a quello ricevuto. Questa somiglianza viene misurata tramite la **distanza di Hamming**: in base a tutti i percorsi del trellis, si va a scegliere il percorso che ricostruisce la sequenza tale che **la distanza di Hamming tra la sequenza ricostruita e quella ricevuta è minima**.

In virtù di quanto detto allora si costruisce il seguente codificatore a quattro stati:

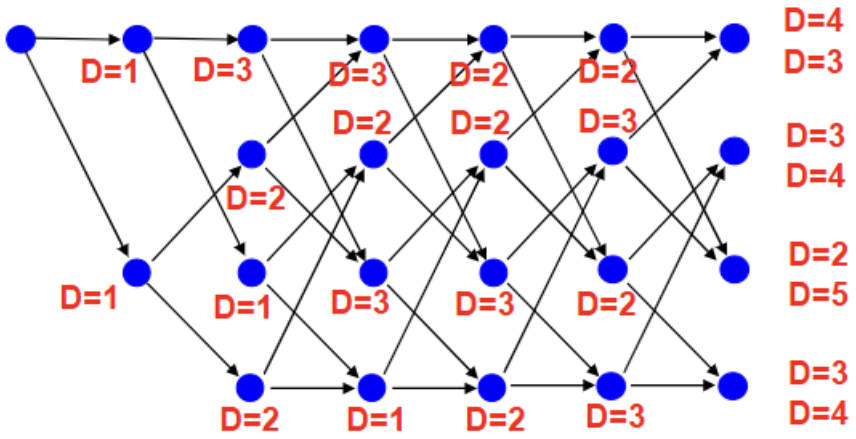


Supponiamo la sequenza di simboli `00 11 01 01 11 11` subisca la trasformazione per interferenza `01 11 01 11 11 11`.
Costruiamo il trellis relativo alla macchina a stati:

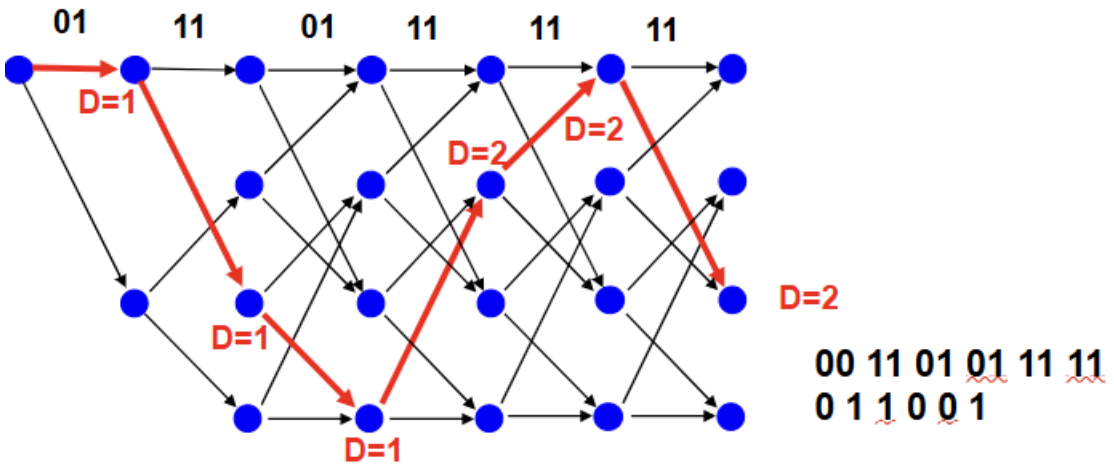


Computer Netw

Analizzando tutti i percorsi, otteniamo la somma di distanze:



Chiaramente la distanza minima è pari a 2:



Che corrisponde al treno di bit originale. Da qui possiamo allora ricostruire il treno di bit ricevuto.

Strutture Ethernet

Per connettere le varie sottoreti, Ethernet prevede l'utilizzo di hub e bridge. Hub e bridge sono due strutture diverse:

- hub lavora più a livello fisico. Abbiamo una caratterizzazione della connessione tra i nodi più vicina al livello fisico che a livello di collegamento. Bridge invece lavora sul data link;
- l'hub ripete e rigenera il segnale su tutte le porte, pertanto l'**hub è una struttura passiva che esegue forwarding**. Invece il brdige **filtra e inoltra frame in base al MAC address, pertanto è un dispositivo intelligente**;
- l'hub ha un unico **dominio di collisione**. Il dominio di collisione rappresenta le aree della nostra rete in cui può avvenire una collisione; conoscere i domini di collisione è fondamentale per la risoluzione dei problemi di traffico di rete. Non è detto che questo dominio sia unico. **Nei bridge ogni porta crea idealmente un proprio dominio di collisione**;
- a livello di broadcast, sia hub che bridge hanno un unico dominio di broadcast;
- l'hub non filtra il traffico in ingresso, inoltrando ogni frame in ingresso su tutte le porte. Il **bridge invece impara gli indirizzi MAC e invia ogni frame solo sulla porta di destinazione nota**;
- hub ha alte collisioni, una sicurezza bassa e il throughput è condiviso da tutti. Il bridge, invece, presenta meno collisioni e dedica la larghezza di banda ad ogni segmento;
- l'hub non ha tabella di forwarding, il bridge sì.

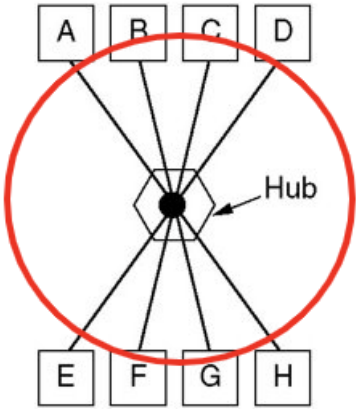
Oggi l'hub è obsoleto, mentre il bridge è alla base di tutti gli switch moderni.

Quando si collegano più bridge nell'infrastruttura di rete si può creare un **loop** di livello 2. Il loop può essere un problema perché il frame può circolare all'infinito e la rete si può saturare a causa di un **broadcast storm**, ossia di una tempesta di pacchetti.
Per risolvere questo problema, si utilizzano gli **spanning tree**. Gli spanning tree identificano quali sono gli unici collegamenti che possono fare forwarding, identificando un sottoinsieme di porte da bloccare in modo tale da lasciare un **albero di collegamenti attivi**. Poiché per definizione un albero è un grafo aciclico connesso, si evita il problema del loop. Nella definizione dello spanning tree, si individuano sempre 3 fasi:

1. **elezione del root bridge**, ossia il bridge con **identificativo più basso**. Ogni identificativo è composto da una priorità numerica e dal MAC address. Identificato il root bridge, tutti i bridge cercheranno il più breve percorso verso questo;
2. **calcolo del costo**. Ogni collegamento ha un costo, e ogni bridge calcola il costo cumulato del cammino più corto per raggiungere il root;
3. **assegnazione dei ruoli alle porte**. Ogni bridge ha una **root port** che è la porta con costo minimo verso il root. Tale porta è sempre attiva. Su ogni collegamento la porta con cammino più veloce verso il root viene pure mantenuta attiva e gestisce il traffico su quel segmento di rete. Tale porta è la **designated port**. Tutte le altre porte sono **bloccate**.

Hub

L'hub è un dispositivo che ripete tutti il segnale in ingresso su ogni porta senza alcuna distinzione di indirizzo. Poiché ogni trasmissione concorre allo stesso mezzo condiviso, si ha un unico dominio di collisione.
In foto è rappresentata una rete con una **topologia a stella**, ossia dove tutti i nodi sono collegati ad un nodo centrale, che in questo caso è l'hub:



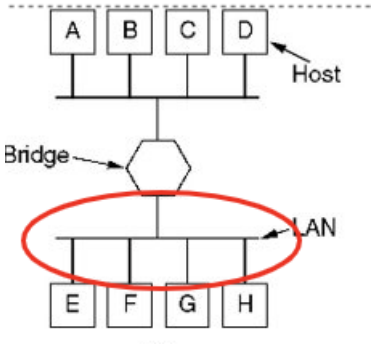
Bridge

Un bridge è in grado di gestire il traffico tra segmenti di una stessa rete o tra due reti. I dispositivi pertanto sono connessi tramite un bus di comunicazione ad una porta del bridge. Quando il bridge riceve un frame su una porta, cerca di capire se il destinatario si trova sullo stesso segmento del mittente o no:

- in caso positivo, il bridge non inoltra il frame, in quanto il destinatario l'ha giò ricevuto per la condivisione del bus di comunicazione;
- in caso negativo, inoltra il frame verso il segmento di appartenenza del destinatario. Se non si conosce tale segmento, si esegue flooding.

Il bridge mantiene una tabella di forwarding per ciascuna porta per capire dove inoltrare il frame, e tramite il traffico progressivo sul bridge esegue un **auto-apprendimento**. L'apprendimento si basa sull'**analisi dell'indirizzo MAC sorgente**. Quando il bridge riceve un frame da una porta, analizza il MAC sorgente e annota nella tabella di forwarding l'associazione MAC interfaccia.

Il bridge presenta un numero di domini di collisioni pari al numero di segmenti che il dispositivo connette. Nella foto sottostante si hanno due bus, quindi due segmenti, per cui i domini di collisione sono due:



Solitamente il bridge per definizione connette due segmenti di rete connessi a hub.

Switch

Uno switch è un dispositivo che trasmette i pacchetti ricevuti solamente a uno o più dispositivi destinatari. A differenza di un bridge, lo switch gode di una **maggiore espandibilità** in termini di numero di porte.
Lo switch esegue operazioni di **inoltro** e **filtraggio**:

- con **inoltro** si intende l'individuazione dell'interfaccia verso cui il frame dev'essere diretto, nonché il successivo invio;
- con **filtraggio** si intende la funzionalità che determina se un frame dev'essere scartato o meno.

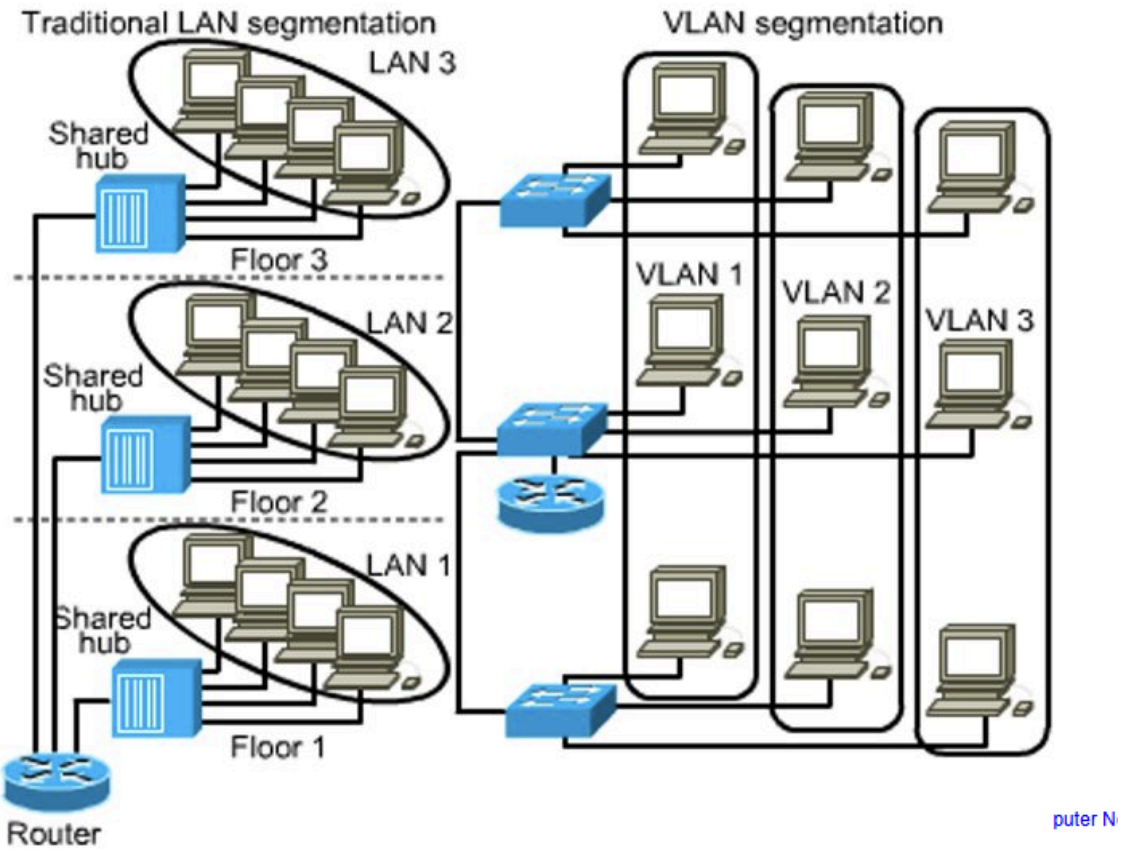
Una **switch table** contiene l'associazione MAC address-interfaccia-tempo. L'etichetta temporale semplicemente indica il momento in cui la voce per un determinato nodo è stata inserita in tabella. Anche lo switch è dotato di un meccanismo di auto-apprendimento per l'aggiornamento della switch table. Il campo temporale, in particolare, viene utilizzato per cancellare un indirizzo sorgente da cui non si ricevono pacchetti dopo un periodo di tempo detto **aging time**, permettendo quindi di dedurre se un calcolatore viene sostituito.

Uno switch presenta tanti colllision domain quante sono le porte. Infatti un collegamento da un nodo `A` ad un nodo `B` tramite switch è un collegamento full duplex isolato. L'inoltro, inoltre, avviene sulle singole porte e non sul bus, come nel caso del bridge. Praticamente, in full duplex non si hanno collisioni quando si utilizzano gli switch.

VLAN

Quando va fatto un cablaggio strutturato in un edificio per rendere disponibile una linea ethernet a dei nodi, è necessario strutturare il passaggio dei cavi, la connessione degli switch con il router in maniera efficiente. Supponiamo inoltre che per qualche motivo l'edificio cambia di esigenza. Ad esempio, la struttura ha esigenze diverse, pertanto deve interconnettere i vari nodi in maniera differente.

Invece di cambiare il sistema di cablaggio (lavoro abbastanza improponibile anche a livello edilizio, non solamente a quello economico), possiamo utilizzare una **VLAN**. La **VLAN** è una LAN virtuale. La interconnessione è garantita tramite la rimappatura delle tabelle di forward del bridge, interconnettendo i vari nodi in maniera diversa in cui sono cablati fisicamente. Quindi fisicamente i nodi sono cablati in un modo, ma virtualmente in un altro. Ad esempio:



Se prima quindi il cablaggio è per piani (a sinistra), ora il cablaggio è per settori (a destra). Si possono quindi connettere gli host di sopra con quelli di sotto creando routing differenti specialmente se i bridge sono connessi fisicamente tra loro.

Lo standard che si occupa di gestire la VLAN è 802.1Q. Gli scopi della VLAN sono molteplici:

- **risparmio**, perché si riutilizza l'equipaggiamento esistente;
- **flessibilità**, in quanto non si creano realmente nuove connessioni fisiche, garantendo un movimento fisico facile tra gli utenti;
- **miglioramenti di performance**, in quanto creando dei routing statici nella VLAN, quest'ultima è confinata. Ad esempio, se prima si avevano 5 computer per piano in un sistema a 3 piani, una VLAN limitata a 2 computer per piano migliora il broadcast;
- **sicurezza**, in quanto gli utenti di VLAN differenti non vedono i data frame di utenti di altre VLAN.

Le VLAN sono connesse tra loro tramite **routing a livello di rete**. Uno switch può essere partizionato logicamente in più parti, assegnando le porte ad una VLAN. Per identificare le VLAN, nell'header a livello DLL che si inviano c'è un identificatore di VLAN. Questo accade inserendo nel frame Ethernet un tag a quattro byte. Tale specifica è sempre definita in 802.1Q.
I quattro byte prevedono:

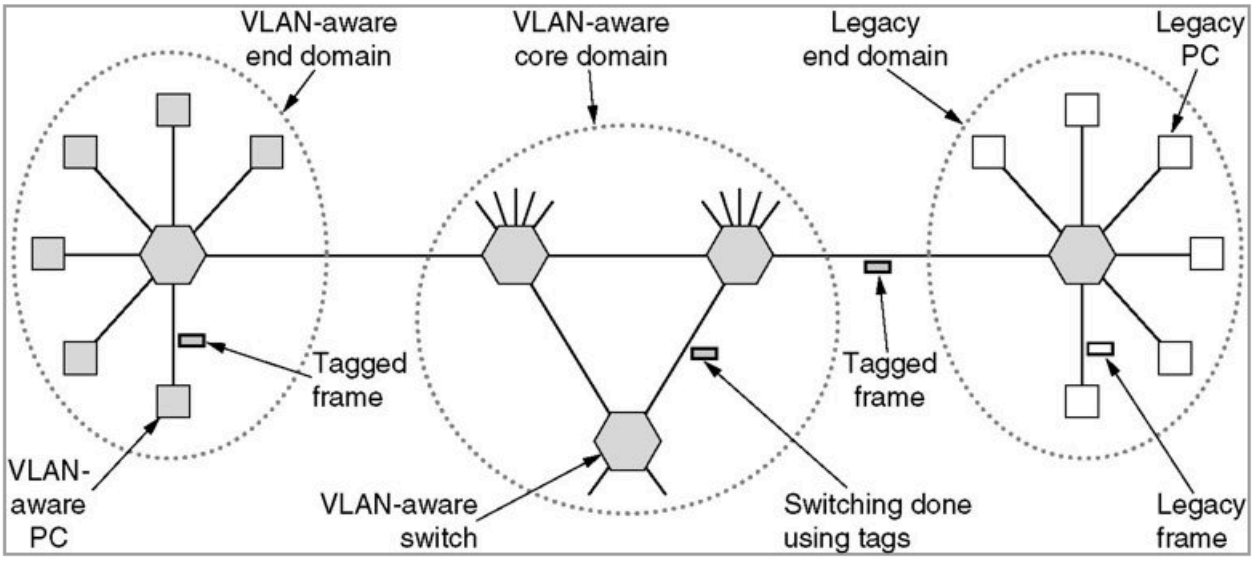
- 16 bit per il **TPID**, ossia il **Tag Protocol Identifier**;
- 3 bit sono per il **priority**, ossia per il livello di priorità;
- 1 bit per il **CFI**, ossia il **Canonical Format Indicator**, e indica se il frame è tagged o untagged. Il frame posto a 1 indica che si supporta il meccanismo di VLAN;
- 12 bit per il **VID**, ossia il **VLAN identifier**.

Distinguiamo due tipi di VLAN:

- **VLAN Untagged**, che indica che **i frame su quella porta non contengono informazioni sulla VLAN**, quindi sono frame Ethernet standard. I dispositivi potrebbero non interpretare i tag VLAN, per cui lo switch utilizza una porta per l'identificazione della VLAN, e i frame che arrivano vengono "puliti", ossia il tag VLAN viene rimosso. Le porte di inoltro a dispositivi VLAN-unaware sono dette **porte access**;

- **VLAN Tagged**, che si trovano sulle porte attraverso cui è necessario far passare il traffico di più VLAN. Il dispositivo ricevente può identificare la VLAN corretta e inoltrare il frame alla porta adeguata. Le porte che supportano il traffico tagged sono dette **porte trunk**.

Consideriamo l'esempio:



A sinistra abbiamo una VLAN a stella. Lo switch riceve il frame taggato, pertanto riconosce prima se il frame è appropriato alla VLAN e poi gestendo l'invio all'host corretto.
A destra si ha una LAN con **strutture legacy**. I sistemi legacy sono sistemi che non supportano i nuovi protocolli o nuove configurazioni, ma solo quelle precedenti. La struttura a destra è una vecchia rete che non supporta le estensioni VLAN. Ignora quindi tutti i pacchetti tagged in quanto la formattazione dei frame non è conforme e quindi queste strutture legacy gestiscono solo i frame legacy. VLAN e LAN legacy quindi possono coesistere.

C'è da specificare che ogni porta è associata ad **un'unica VLAN**, e non può essere associata a due VLAN, altrimenti si hanno problemi di collisioni.