Appunti di Reti di Calcolatori

*Appunti di **Giuseppe Pitruzzella** - Corso di Internet Security @ DMI, UniCT*

Teoria 📚

Laboratorio 💻

Introduzione

*Appunti di **Giuseppe Pitruzzella** - Corso di Reti di Calcolatori @ DMI, UniCt*

Cos'è Internet?

Internet puo' essere inteso come una **rete pubblica di calcolatori** che interconnette miliardi di dispositivi.
Questi dispositivi sono chiamati *host* o sistemi periferici (end system) e sono connessi tra loro grazie a rete di collegamenti (*commutation link*) e *commutatori di pacchetti* (packet switch).
I sistemi periferici accedono ad Internet attraverso un **ISP** (Internet Service Provider), un insieme di commutatori di pacchetto e di collegamenti. Questi collegamenti possono essere di molti tipi, alcuni di essi riguardano fili di rame, fibra ottica o onde elettromagnetiche.

La comunicazione tra due macchine (o meglio tra due sistemi periferici) inizia con la suddivisione delle informazioni, ognuna con la rispettiva intestazione. L'insieme delle informazioni è chiamato **pacchetto**.
I pacchetti viaggiano attraverso la rete alla destinazione, che riassembla quest'ultimi per ottenere l'informazione originale.
Un commutatore di pacchetto prende un pacchetto in ingresso e lo ritrasmette in uscita.
I *commutatori di pacchetto* odierni sono **router** e **link-layer switch** (commutatori a livello di collegamento).

Interfaccia socket

Ogni dispositivo collegato ad Internet, fornisce un interfaccia **socket**, quale specifica un insieme di regole secondo cui un programma su un sistema periferico A possa chiedere ad Internet di recapitare dati ad un programma eseguito su un sistema periferico B.
Per fissare il concetto utilizziamo la seguente analogia: se $Alice$ volesse inviare una lettera a Bob, allora non basterebbe scrivere la sola lettera per poi lanciarla dalla finestra, bensì è necessario seguire alcune regole, ovvero inserire la lettera in una busta, scrivere i recapiti del destinatario per poi imbucare quest'ultima. Possiamo vedere queste "*regole*" come l'interfaccia socket del servizio postale.

I protocolli

Tutto ciò che abbiamo elencato fin'ora, ovvero sistemi periferici, commutatori di pacchetto ed altro, fa uso di **protocolli**. I principali protocolli in Internet sono due, *TCP* (Transmission control protocol) ed *IP* (Internet protocol); quest'ultimo specifica il formato dei pacchetti scambiati tra router e sistema periferico. I due sono noti sopratutto con il nome collettivo di TCP/IP.
Anche per questo argomento, per capire meglio, utilizziamo un analogia: se $Alice$ volesse chiedere l'ora a Bob, sicuramente per prima cosa vi sarà un saluto da $Alice$ verso Bob, e probabilmente un altro da Bob verso $Alice$. Certamente in base a quest'ultimo possiamo

capire se è il caso o meno di fare la richiesta riguardo l'ora. In ogni caso se tutto va bene, allora avviene la richiesta di $Alice$ ed in seguito una risposta da parte di Bob.
I protocolli in Internet sono molto simili ai protocolli umani con la differenza che, chi invia messaggi ed intraprende azioni sono componenti hardware o software di un dispositivo. Troviamo protocolli in qualunque attività riguardi Internet.

> Formalmente con protocollo intendiamo: uno specifico formato ed ordine di messaggi scambiati tra due o più entità.

Terminologia

Abbiamo descritto i dispositivi con il nome di sistemi periferici o host, e bene però fare un ulteriore distinzione in quanto quest'ultimi (host) si dividono in (i) **client**, ovvero colui che richiede servizi e (ii) **server**, colui che è in attesa di erogare un servizio. Oggigiorno la maggior parte dei server è collocata in grandi data center, una struttura adibita ad ospitarli fisicamente.

Reti di accesso

Con reti di accesso si intende la rete che connette fisicamente un sistema al suo edge router (router di bordo), che è il primo router sul percorso dal sistema d'origine a un qualsiasi altro sistema di destinazione.
Reti di accesso relative ad **accessi residenziali** riguardano:

- **DSL** (Digital subscriber line), generalmente fornito dal gestore telefonico insieme al servizio di telefonia fissa. In questo modo il gestore telefonico assume il ruolo di ISP. Si noti che il modem DSL dell'utente utilizza la linea fissa per scambiare dati con un *DSLAM* (Digital subscriber line access multiplex), quale si trova nella centrale locale della compagnia. I dati digitali prima di essere inviati al DSLAM devono essere convertiti in segnali ad alta frequenza per poter essere trasmessi attraverso il cavo telefonico. Arrivati al DSLAM, egli stesso riconvertirà i dati in digitale.
- **FTTH** (Fiber to the home), una tecnologia che vanta ancora maggiori velocità. Come suggerito dal nome, il concetto di FTTH è semplice: fornire fibra ottica dalla centrale locale direttamente alle abitazioni. La rete di distribuzione ottica più semplice è chiamata **fibra diretta**, in cui una singola fibra collega una centrale locale a un'abitazione. Tuttavia, di solito una fibra uscente dalla centrale locale è in effetti suddivisa per essere condivisa da molte abitazioni e solo quando arriva relativamente vicina alle abitazioni viene ancora una volta suddivisa in più fibre, ognuna dedicata a un utente. Esistono due architettura in grado di eseguire questa **suddivisione**: **AON** (Active Optical Network) e **PON** (Passive Optical Network). Ci concentriamo maggiormente sulla più utilizzata PON. Secondo quest'ultima archietettura, ogni *abitazione* possiede un terminatore chiamato **ONT** (Optical Network Terminator), un terminatore a cui ogni utente si collega attraverso il proprio router. ONT si collega a sua volta ad uno **splitter** di quartiere, una singole fibra con il compito di collegare più abitazioni, ognuna con il proprio ONT. Questa singola fibra all'interno dello splitter si collega all'altro capo ad un **OLT** (Optical Line Terminator), situato all'interno della

centrale *locale* della compagnia telefonica. L'OLT si connette ad Internet attraverso un router della compagnia telefonica.

- **Cavo**, la quale utilizza le infrastrutture esistenti della televisione via cavo. Un'abitazione richiede un accesso a Internet via cavo alla stessa azienda che le fornisce il servizio di televisione via cavo.

Canali di comunicazione

Gli utenti, quindi le applicazioni, hanno bisogno di un **canale virtuale** affidabile e privo di errori. Un canale virtuale è implementato attraverso un **canale fisico**. Un canale fisico può essere:

- **Simplex**: quando la comunicazione avviene in una sola direzione ed è quindi composto da un trasmettitore ed un ricevitore.
- **Half duplex**: quando la comunicazione avviene da ambedue le parti, ma con il vincolo che venga effettuata uno alla volta.
- **Full duplex**: quando la comunicazione avviene in entrambe le parti e nello stesso momento. Quest'ultimi sono difficili da sviluppare.

Alcuni **problemi** derivati dai canali di comunicazione sono i seguenti:

- I messaggi a basso livello non possono avere lunghezza arbitraria.
- Un trasmettitore non deve sommergere un ricevitore lento.
- L'ordine dei messaggi in spedizione ed arrivo deve essere.
- Deve essere determinato il percorso migliore, seocndo una specifica metrica.

Tipi di rete

Impariamo adesso a distinguere le varie tipologie di rete, ovvero:

- **Punto-punto**, nient'altro che una singola connessione dalla macchina A alla macchina B. Con questo tipo di rete, i pacchetti passano per macchine intermedie ma non possono essere letti.
- **Broadcast**, la quale si riferisce ad una serie di macchine connesse tra di loro. Un pacchetto inviato da una delle macchine viene letto sequenzialmente da tutte le macchine presenti. Alcune tipologie di rete broadcast sono:
    - **Bus condiviso**, attraverso cui tutti possono leggere e scrivere ma trasmette una sola macchina alla volta.
    - **Anello**, secondo cui ogni macchina possiede due interfacce (una per leggere, l'altra per scrivere) e le informazioni girano in senso orario o antiorario. Questo tipo di connessione può essere visto come tanto connessioni *punto-punto*. Un **problema** (*software*) con questo tipologia di rete è che le macchine intermedie che consegnano un messaggio da mittente a destinatario, se volessero parlare, troverebbero il canale occupato.

- **Stella**, il più comune rispetto ai precedenti, è formato da un **hub** che collega con una connessione *punto-punto* le macchine presenti. Si noti che è piu simile ad un bus condiviso rispetto che quello ad anello per via della *non-intelligenza* dell'hub.

Tassonomia di rete

- $1m$: **PAN**, personal area network;
- $10m\ -\ 1km$: **LAN**, local area network;
- $10km$: **MAN**, metropolitan area network;
- $1\,000km$: **WAN**, wide area network;
- $100\,000km$: **Internet**.

Reti per le telecomunicazioni

Esaminati i *confini di Internet*, approfondiamo ora in dettaglio il nucleo della rete: una maglia di **commutatori di pacchetti** e **collegamenti** che interconnettono i sistemi periferici di Internet.

Commutazione di pacchetto

Le applicazioni distribuite scambiano messaggi di diverso tipo; essi, infatti, possono avere funzione di controllo o possono contenere dati relativi ad un immagine o un file audio. La macchine sorgente ha il compito di suddividere tale informazioni in parti più piccole, note come **pacchetti**. Questi pacchetti viaggiano attraverso **commutatori di pacchetto**, i quali possono essere *router* o *link-layer switch*.
La maggior parte dei commutatori di pacchetto utilizza la **trasmissione store-and-forward**, il che implica che egli non invierà verso la destinazione alcun bit di un pacchetto $i-esimo$ prima che egli stesso non abbia ricevuto ogni bit relativo a quest'ultimo pacchetto (l'intero pacchetto).
Per fissare il concetto pensiamo ad una rete formata da due macchine A e B collegate da un unico router. Quest'ultimi potranno inviare dei pacchetti di dimensione fissa pari ad L con un velocità costante per ogni collegamente pari ad R.
Se A volesse inviare tre pacchetti verso B, ognuno con dimensione pari ad L, allora a tempo $t_1 = \frac{L}{R}$ (con $R\ bps$ uguale alla velocità con cui viene inviato ogni bit) l'*intero* primo pacchetto sarà stato ricevuto dal router, il quale darà inizio alla trasmissione verso la destinazione. A questo punto a tempo $t_2 = 2 * \frac{L}{R}$, B riceve per intero il primo di tre pacchetti e, nel frattempo, il router riceve anche il secondo pacchetto. A tempo $t_3 = 4 * \frac{L}{R}$, B riceve per intero i tre pacchetti.

> Il caso generale indica, dati N di collegamenti ed $N-1$ router, un tempo per la trasmissione di un pacchetto pari a $D_{end_to_end} = N * \frac{L}{R}$.

Ogni commutatore di pacchetto fa uso, inoltre, di un **buffer di output** per conservare i pacchetti da inviare su un determinato collegamento. Di conseguenza, oltre al ritardo relativo alla trasmissione *store-and-forward* si aggiungono i ritardi di accodamento di pacchetti all'interno del buffer. Inoltre, poichè il buffer ha dimensione finita, può capitare che un pacchetto in arrivo trovi il buffer pieno, quindi si verifichi una **perdita di pacchetto**.

Un altra caratteristica di un commutatore di pacchetto risponde alla seguente domanda, ovvero: come riesce un commutatore di pacchetto a determinare il corretto collegamente in uscita? Il tutto è possibile attraverso una **tabella di inoltro**, attraverso cui un router può mettere in relazione l'indirizzo della destinazione (contenuto nell'intestazione del pacchetto) ed i suoi collegamenti in uscita.

> Per capire meglio il ruolo del router, pensiamo ad un analogia secondo cui $Alice$ voglia effettuare un viaggio da Caltanissetta verso Catania senza far alcun utilizzo qualsivoglia mappa. Fondamentalmente, il compito di $Alice$ sarà quello di fermarsi ad ogni stazione di servizio per chiedere indicazioni verso Catania o verso il posto conosciuto dall'addetto più vicino a Catania.

Commutazione di circuito

Nelle reti a commutazione di circuito, le risorse richieste lungo un percorso per consentire la comunicazione tra sistemi periferici, sono *riservate* per l'intera durata della sessione di comunicazione.
Un esempio sono le reti telefoniche, dove il primo passo prevede che venga stabilita una connessione "a regola d'arte" per la comunicazione tra mittente e destinatario (questa connessione è detta circuito).
Quando la rete stabilisce un circuito, riserva e garantisce anche una velocità di trasmissione costante.
Un circuito all'interno di un collegamento è implementato mediante:

- **FDM** (*Multiplexing a divisione di frequenza*), secondo cui viene dedicata una banda di frequenza a ciascuna connessione per la durata della connessione stessa (un esempio di FDM sono le radio FM).
- **TDM** (*Multiplexing a divisione di tempo*), attraverso cui il tempo viene suddiviso in frame (intervalli) di durata fissa, quest'ultimi poi vengono suddivisi ancora una volta in un numero fisso di porzioni. Durante la creazione di una connessione, la rete dedica a quest'ultima uno slot di tempo per ogni frame. Questi slot saranno dedicati esclusivamente a quella connessione.

> [$Chiarimento$] Come funzionano le reti a commutazione di circuito, come le reti con circuiti virtuali e come le reti datagram?

Commmutazione di pacchetto vs Commmutazione di circuito

Si supponga che gli utenti condividano un collegamento da 1Mbps e che ognuno di essi alterni (per il 10%) periodi di attività in cui genera dati con una velocità costante di 100Kbps, a periodi in cui non genera alcun dato (per il restante 90%).
In questo caso, con la commutazione di circuito sarebbe necessario riservare 100Kbps in ogni istante, che l'utente sia attivo o meno. Il che indica che vi possono essere al più 10 utenti collegati.
Con la commutazione di pacchetto invece il numero di utenti collegati è molto maggiore poichè gioca con il fattore probabilistico.
Per esempio, con 35 utenti collegati, la probabilità che 11 di loro siano attivi è dello 0,0004%.
In generale la commutazione di pacchetto fornisce le stesse prestazioni della commutazioni di pacchetto, con la differenza che supporta più del triplo degli utenti.

Rete di reti

Architettura a livelli

Come abbiamo detto all'inizio, un architettura a livelli (o strati) è stato il modello utilizzato da progettisti per implementare protocolli, hardware e software.
Vi sono protocolli per ogni livello.
Considerati insieme, i protocolli dei vari livelli sono detti pila di protocolli.
La pila di protocolli di Internet consiste di cinque livelli: applicazione, trasporto, rete, collegamento e fisico.
Descriviamo brevemente la funzione di ogni livello, che vedremo nello specifico nei prossimi capitoli.
Il livello applicativo (applicazione) è la sede delle applicazioni di rete e dei suoi relativi protocolli come HTTP, FTP ed SMTP.
Possono essere scambiati pacchetti attraverso applicazioni tra diversi sistemi periferici.
Questi pacchetti, in questo livello, verranno chiamati messaggi.
Il livello di trasporto aiuta nel trasferimento dei messaggi precedentemente descritti, questo grazie ai suoi due protocolli: TCP ed UDP.
TCP offre un servizio connection-oriented ed include garanzie di consegna, controllo di flusso, controllo della congestione ed altro ancora.
UDP invece è un servizio connection-less e non offre quasi nulla rispetto ciò che offre il precedente.
Chiameremo i pacchetti a livello di trasporto segmenti.
Il livello di rete si occupa di gestire i datagrammi, ovvero i segmenti arrivati in questo livello.
Il compito del livello di rete è mettere a disposizione il servizio di consegna per il segmento.
Inoltre comprende anche l'indirizzo IP.
Il livello di collegamento instrada (esegue operazioni di routing) il datagramma attraverso una una serie di router tra sorgente e destinazione. Chiameremo frame il pacchetto a livello di collegamento.
Il livello fisico ha il ruolo di trasferire i singoli bit che formano il frame da un nodo al successivo. Questo avviene mediante un effettivo mezzo trasmissivo come il doppino o fibra ottica.

Modello ISO OSI

Negli anni 70 l'azienda ISO propose il modello OSI con sette livelli.
Quest'ultimi sono: applicazione, presentazione, sessione, trasporto, rete, collegamento e fisico.
Introduce quindi il livello di presentazione e sessione.
Il livello di presentazione ha il ruolo di fornire servizi che consentono alle applicazioni di interpretare il significato dei dati scambiati (comprende servizi come cifratura e compressione dei dati).
Il livello di sessione fornisce la delimitazione e la sincronizzazione dello scambio di dati.
Questo modello non fu mai sviluppato se non nel suo lato teorico.

Incapsulamento

Con questo paragrafo spiegamo il concetto di incapsulamento durante la trasmissione di un pacchetto.
Iniziamo con il messaggio M a livello applicativo, quale viene passato al livello di trasporto.
Quest'ultimo concatena al messaggio un intestazione H_t, quali informazioni per far si che il pacchetto arrivi all'applicazione giusta e bit per il rilevamento di errori; il tutto servirà al livello di trasposto del sistema periferico ricevente.
Il livello di trasposto passa il segmento a livello di trasporto (ovvero M con H_t) al livello di rete, che aggiunge una sua intestazione H_n.
Così facendo passerà al livello inferiore, il livello di collegamento. Anche questo aggiungerà la sua intestazione H_l.
Infine verrà inviato il tutto al ricevente grazie al livello fisico.
A questo punto possiamo distinguere due campi: una serie di intestazioni ed il payload (ovvero il carico utile).
Tipicamente con payload intendiamo un pacchetto proveniente dal livello superiore.
[*NB*] Il nome delle intestazioni nel libro è formato dalla prima lettera del nome del livello in inglese.

Livello Applicativo

Introduzione

Architetture

Il principio delle applicazioni di rete si trova nello sviluppo di programmi che verranno poi eseguiti su dei sistemi periferici.
Un esempio sono le applicazioni web, dove due programmi diversi comunicano tra loro; questi sono il browser eseguito da un host, ed il web server eseguito da un host chiamato anch'esso web server.
Parliamo quindi di un architettura client-server in cui vi è un host sempre attivo, chiamato server, che risponde alle richieste degli host client.
Spesso un server non è in grado di rispondere alle richieste di tutti i client, per questo motivo molte aziende utilizzano dei data center, quali possono ospitare centinaia di migliaia di host, in modo da formare un grande server virtuale. Un azienda come Google utilizza dai 50 ai 100 data center.
In alternativa a questa architettura esiste l'architettura P2P, in cui la comunicazione avviene tra coppie di host (tra peer ovvero pari) come computer fissi o portatili. Nelle connessioni P2P chiamiamo client chi scarica il file e server chi lo invia. Si noti che a volte un processo in P2P può essere sia client che server.
In generale (ciò riguarda tutte le architetture) chiamiamo client chi avvia la comunicazione (cioè contatta) e server chi attende di essere contattato.

Processi e socket

Rivediamo il concetto di socket di cui abbiamo parlato nel primo capitolo, adesso dal punto di vista dei processi applicativi.
Possiamo intendere la socket come l'interfaccia tra livello di applicazione e livello di trasporto.
Diciamo che un processo invia messaggi nella rete e riceve messaggi dalla rete grazie ad un interfaccia software chiamata socket.
Per capire meglio, utilizziamo un analogia.
Possiamo vedere una casa come un processo e le interfacce come le porte della casa.
Dunque un processo che vuole spedire un messaggio ad un altro processo, spedisce il messaggio attraverso una porta, supponendo che dopo vi sia un modo per cui il messaggio venga trasportato fino alla porta di destinazione.
Affinchè il processo venga inviato è necessario disporre di due indirizzi, uno che indentifichi l'host, un'altro che indentifichi il processo.
Identifichiamo il primo nell'indirizzo IP, un numero di 32 bit che possiamo pensare identifichi uinvocamente ogni host.
Identifichiamo il secondo nel numero di porta di destinazione.
Parleremo meglio di entrambi nei capitoli successivi.

Protocolli e servizi di trasporto

Molte reti mettono a disposizione molti protocolli di trasporto. Nel progettare un applicazione è necessario scegliere un protocollo a livello di trasporto che ci soddisfi, poiché ognuno di essi fornisce diversi servizi.Non a caso classifichiamo i protocolli di trasporto in base ai servizi che offre, ovvero:

- **Trasferimento di dati affidabile**.
  Sappiamo bene che un pacchetto può andar perso a causa del trabocco di un buffer o scartato a causa di alcuni bit corrotti.
  Un protocollo che fornisce garanzia riguardo la consegna di dati si dice che fornisce trasferimento di dati affidabile ed è necessario ad applicazioni che non possono permettere che alcun dato venga perso.
- **Throughput**.
  Possiamo distinguere due tipi di throughput, istantaneo e medio. Con throughput istantaneo intendiamo la velocità a cui B riceve i bit del file da A. Con throughput medio intendiamo la velocità diviso il tempo necessario affinché B riceva tutti i bit del file. In generale se un protocollo di trasporto fornisce servizio di throughput, si vuole intendere un throughput disponibile garantito (quindi una soglia minima sotto cui non si scenderà).
  Un esempio è un applicazione di telefonia Internet che codifica la voce a 32 kbps ed ha necessità che i dati vengano ricevuti a quello stesso tasso.
  Le applicazioni che necessitano di questo requisito sono dette applicazioni sensibili alla banda.
  Chi invece non necessita di questo servizio sono le applicazioni elastiche.
- **Temporizzazione**.
  Con il servizio di temporizzazione intendiamo il ritardo massimo con cui ogni bit viene ricevuto.
  Questo interessa le applicazioni interattive in tempo reale.
- **Sicurezza**.
  Un protocollo di trasporto può offrire più servizi riguardanti la sicurezza, alcuni di questi sono la cifratura dei dati da parte del mittente (poi decifrati dal destinatario), la riservatezza e l'integrità dei dati ed altri ancora.

Internet fornisce due tipi di protocolli a livello di trasporto, ovvero

- **TCP**,
  - **Servizio orientato alla connessione**.
    Con questo servizio, TCP fa in modo che client e server si scambino informazioni di controllo a livello di trasporto prima che i messaggi a livello di applicazione comincino a fluire.
    Questa procedura, detta handshaking, allerta e prepara client e server alla partenza dei pacchetti.
    Dopo l'handshaking possiamo dire che esiste una connessione TCP, tra le socket dei due processi. Quest'ultima (connessione) è full duplex, il che indica che i due processi possono scambiarsi contemporaneamente messaggi.
    L'applicazione, infine, deve chiudere la connessione quando termina di inviare messaggi.

    - **Servizio di trasporto affidabile**.
      Pur avendo descritto questo servizio poco prima, ricordiamo che quest'ultimo offre garanzia riguarda la ricezione (con il giusto ordine) dei dati. Controllo della congestione. Questo servizio verrà descritto bene nel prossimo capitolo.

- **UDP**, un protocollo molto più semplice rispetto al precedente poiché offre un modello minimalista con pochi servizi a disposizione.
  UDP infatti è un protocollo *connection-less* (non possiede, quindi, una procedura di *handshaking*) e non prevede trasporto dati affidabile o controllo della congestione.

> Si noti che sia per TCP che UDP, non si sono menzionati servizi come la temporizzazione o il throughput. Il motivo è riguarda semplicemente l'assenza di quest'ultimi nei protocolli odierni. Ciò non significa, però, che applicazioni di telefonia o in tempo reale non possono esistere in Internet poiché esse riescono a convivere bene anche senza quest'ultimi. Per esempio per Skype si utilizza UDP poichè può tollerare perdite ma hanno requisiti minimi riguardo la velocità. Scegliendo questo quindi si evitano tutti i messaggi aggiuntivi dati dal controllo della congestione e si favorisce maggiore velocità. Non sempre però UDP può essere utilizzato al meglio per la telefonia, poiché alcuni firewall sono impostati in modo da bloccare il traffico proveniente da questo protocollo, motivo per cui si tende ad utilizzare TCP come opzione di riserva.

Protocolli a livello applicazione

Vediamo adesso alcuni protocolli a livello applicativo. Questi non sono da confondere con le applicazioni di rete, in quanto un protocollo a livello applicativo è solo una parte di quest'ultima.

HTTP

Introduzione

HTTP, descritto nell'$RFC\ 1945$ ed $RFC\ 2616$, nasce come un modo per *richiedere* e *ricevere* file da una macchina all'altra. Il protocollo è implementato in due *programmi*

- *Client*, rappresentato da un *browser web* installato all'interno di in un sistema periferico A. Attraverso il browser ed apposito *URL* è possibile richiedere una determinata pagine al *web-server*, che può trasferire una pagina al client stesso.
- *Server*, rappresentato da un *web-server* in un sistema periferico B, il quale ospita pagine Web referenziabili tramite *URL*. Un web-server è potenzialmente sempre attivo ed in grado di rispondere a richieste provenienti da molioni di diversi browser.
  E' importante notare che i server HTTP non mantengono informazioni sui client, il che indica che HTTP è classificato come protocollo *state-less* (senza memoria di stato).

Connessioni persistenti e non persistenti

In molti applicativi in Internet, client e server comunicano per un lungo periodo secondo una serie di richieste. E' importante per gli sviluppatori dell'applicativo, capire se far sì che ogni richiesta deve essere inviata su una connessione TCP separata o devono essere inviate tutte sulla stessa connessione TCP. Nel primo approccio si dice che l'applicativo utilizza connessioni non persistenti, nell'altra che utilizza connessioni persistenti. HTTP utilizza di default delle connessioni persistenti, tuttavia è possibile per client e server essere configurati per utilizzare connessioni non persistenti.

Le connessioni non persistenti in HTTP 1.0

HTTP 1.0 era caratterizzato da connessioni **non persistenti**. Descriviamo passaggio per passaggio come avveniva il trasferimento di una pagina web dal server al client.

- Il processo client HTTP inizializza una connessione TCP con il server `www.qualcosa.com` sulla porta 80, ossia la porta di default per HTTP. Associato alla connesione TCP vi sarà una socket per il client ed il server.
- Il client HTTP, tramite la propria socket, invia al server un messaggio di richiesta HTTP che include il percorso `qualcosina/home.index`.
- Il processo server HTTP riceve il messaggio di richiesta attraverso la propria socket associata alla connessione. Recupera l'oggetto `/qualcosina/home.index` dalla memoria, lo incapsula in un messaggio per poi inviarlo al client.
- Il processo server HTTP comunica a TCP di chiudere la connessione. TCP chiuderà quindi la connessione ma solo quando sarà certo che il file è stato ricevuto dal client.
- Dopo che il client ha ricevuto il messaggio desiderato, TCP chiude la connessione. Si noti vengano ripetuti i primi quattro passaggi per ogni oggetto (e.g. immagini) all'interno del documento.

> Possiamo definire il **Round Trip Time** (*RTT*) come il tempo impiegato da un pacchetto per viaggiare da client a server e poi da server a client. Ciò include, ovviamente, ritardi di propagazione, accodamento nei router e nei commutatori intermedi ed elaborazione del pacchetto.

Come funziona HTTP dalla versione 1.1

Con HTTP 1.1 sono state introdotte le **connessione persistenti**, il che indica che il server può inviare l'intera pagina web in una sola connessione TCP. Il server quindi chiuderà al connessione quando essa rimane **inattiva** per un dato lasso di tempo.

Formato dei messaggi HTTP

Messaggio di richiesta

- `GET /qualcosina/page.html HTTP/1.1`, rappresentato da metodo, url e versione HTTP, indicano la *riga di richiesta*;
- `Host: www.qualcosa.com`, specifica l'host su cui risiede l'oggetto ed è chiamata rida d'intestazione (*header lines*);
- `Connection: close`, per indicare che il browser non utilizzerà connessioni persistenti;
- `User-agent: Mozilla/5.0`, per indicare il tipo di browser che effettua la richiesta;
- `Accept-language: it`, il che indica la lingua desiderabile per la pagine Web.

> Alla fine di un messaggio di richiesta HTTP vi è sempre una sezione per il *corpo del messaggio* (*entity body*), la quale rimane vuota per `GET` ma viene utilizzato per esempio con `POST` quando si sta compilando un form */

Per quanto riguarda il campo metodo della *riga di richiesta*, si noti che esistono oltre a GET (utilizzato per richiedere una pagina), i seguenti metodi:

- `POST`, metodo che permette l'inserimento di testo nel corpo, da utilizzare per compilare un form.
- `HEAD`, metodo simile a GET con la differenza che, se usato, HTTP tralascia l'oggetto richiesto e risponde con un messaggio.
- `PUT`, consente agli utenti di inviare un oggetto in un percorso specifico.
- `DELETE`, consente la cancellazione di un oggetto da un server.

Messaggio di risposta

- `HTTP/1.1 200 OK`, composto dalla versione di HTTP, codice di stato e messaggio di stato, rappresenta la *riga di stato*;
- `Connection: close`, indica che la comunicazione verrà chiusa dopo il messaggio e rappresenta la riga di intestazione;
- `Date: Tue, 30 Mar 2021 15:41:10 GMT`, composta da ora e data di creazione ed invio del messaggio;
- `Server: Apache/2.2.3 (CentOS)`, analogo rispetto UserAgent, indica il WebServer utilizzato;
- `Last-Modified: Tue, 30 Mar 2021 15:11:10 GMT`, ossia ora e data in cui l'oggetto in questione è stato creato e/o modificato l'ultima volta. Esso è importanto per la gestione dell'oggetto nel proxy;
- `Content-Length: 6821`, ovvero il numero di byte dell'oggetto inviato;
- `Content-Type: text/html`, il quale indica che l'oggetto nel corpo è testo HTML;
- `(data data data data data ...)`, ossia il corpo stesso del documento richiesto.

Descriviamo adesso i vari codici di stato con i relativi messaggi.

- 200 OK: richiesta avvenuta con successa, viene inviata in risposta l'informazione desiderata.
- 301 Moved Permanently: Oggetto richiesto è stato trasferito di locazione in modo permanente. Il nuovo URL sarà specificato in *Location nel messaggio di risposta*.
- 400 Bad Request: Riguarda un errore generico, la richiesta non è stata compresa.
- 404 Not Found: Il documento richiesto non esiste sul server.
- 505 HTTP Version Not Supported: Il server non dispone della versione di HTTP richiesta.

Cookie

Abbiamo visto che i server HTTP sono privi di stato, ma è spesso auspicabile che quest'ultimi si ricordino degli utenti e li possono quindi autenticare. A questo scopo il protocollo HTTP adotta i cookie. La tecnologia dei cookie presenta quattro componenti:

- Riga di intestazione della risposta HTTP;
- Riga di intestazione della richiesta HTTP;
- Un file mantenuto nel sistema dell'utente e gestito dal browser;
- Un database sul sito;

Per fissare il concetto, ipotizziamo che $Alice$ contatti per la prima volta il sito *Amazon.com*, per cui quando giunge la richiesta al web server di *Amazon*, quest'ultimo crea un identificativo univoco per $Alice$ ed una voce nel suo database.
A questo punto nella risposta HTTP da parte del web server verrà aggiunta l'intestazione `Set cookie: xxxx`. Quando il browser di $Alice$ nota quest'ultima intestazione, aggiunge una riga al file dei cookie che gestisce, ossia una tupla composta dal nome dell'host ed il codice identificativo.
Si noti che sebbene *Amazon.com* non conosca il nome di $Alice$, conosce bene quando e quali pagine ha visitato. Le informazione riguardo $Alice$ saranno note ad Amazon quando lei sceglierà di registrarsi sul sito.

Sebbene i cookie semplifichino la vita per molti siti web, quest'ultimi sono fonte di controversie poiché considerati una violazione della privacy. Infatti, la giusta combinazione di cookie ed informazioni di un utente può rilevare molte informazioni su quest'ultimo.

Proxy

Una **web cache**, nota anche con il nome di **proxy server**, è un entità di rete che soddisfa richieste HTTP al posto del web server effettivo. Il proxy infatti ha una propria memoria su disco (una cache) in cui conserva copie di oggetti recentemente richiesti.
Per questo motivo, ogni richiesta di oggetto eseguita attraverso il browser viene prima diretta al proxy.

Supponiamo di voler richiedere l'oggetto http://www.qualcosa.com/qualcosaltro.gif e notiamo cosa succede passaggio per passaggio.

- Il browser stabilisce una connessione TCP con il proxy server ed invia una richiesta HTTP per l'oggetto specificato.
- Il proxy controlla la presenza di una copia dell'oggetto memorizzata localmente. Se l'oggetto viene rilevato, il proxy lo inoltra all'interno della risposta HTTP al browser.
- Se il proxy non dispone dell'oggetto, allora apre un connessione TCP con il server di origine, ossia a www.qualcosa.com. Poi ancora il proxy invia la richiesta dell'oggetto al server. Ricevuta la richiesta, il server invia l'oggetto al proxy.
- Quando il proxy riceve l'oggetto ne salva una copia all'interno della sua cache, inoltrando un'altra copia al browser.
  Un proxy server è generalmente acquistato ed installato da un ISP.

In definitiva un proxy, che svolge contemporaneamente la funzione di client e server, può ridurre i tempi di risposta per ogni richiesta e conseguentemente il traffico locale e globale in Internet.

GET Condizionale

Sebbene il proxy riduca i tempi di risposta, introduce un nuovo problema: la copia di un oggetto in cache potrebbe essere scaduta. Fortunatamente HTTP permette di verificare se i file all'interno della cache sono aggiornati con un **GET Condizionale**. Un messaggio di richiesta GET Condizionale è tale se usa il metodo GET ed include una riga `If-modified-since`.

Supponiamo che il $30\ Aprile$ vi è stata una richiesta, per la prima volta, ad un sito xyz.com, che ha portato il proxy a salvarne una copia nella sua cache. Se una settimana dopo viene fatta richiesta per quello stesso oggetto, è normale pensare che vi è la possibilità che questo sia stato modificato. Questo porta il proxy a controllare se quest'ultimo è stato o meno modificato con la seguente richiesta al server:

```
GET /qwe/rty.gif HTTP/1.1
Host: www.xyz.com
If-modified-since: Sat, 20 Mar 2021 15:23:24
```

Il che porta alla seguente *risposta* da parte del server:

```
HTTP/1.1 304 Not Modified
Date: Mon, 01 Mar 2021 09:41:00
Server: Apache/1.3.0 (Unix)
(Corpo vuoto)
```

Si noti che all'interno del messaggio di risposta non vi è, ovviamente, alcuna traccia dell'oggetto richiesto.
In questo caso il server comunica al proxy che può inoltrare senza alcun problema la sua copia in cache, poichè l'ultima modifica è avvenuta il primo Marzo.

FTP

FTP (acronimo di *File Tranfert Protocol*), è un protocollo a livello applicativo sviluppato dal MIT nel 1971 e definito in $RFC\ 114$.
FTP presenta, diversamente da HTTP, un *sistema di connessione con dati fuori banda*, il che indica che utilizza due connessioni differenti (entrambe TCP) su due porte distinte (20 e 21), rispettivamente per gestire i *dati* ed i *comandi*.
Conseguentemente, per esempio, se vi è un trasferimento in corso che desidero bloccare, piuttosto che inviare il comando tramite la connessione sulla porta 20 (trasferimento dati), in cui potrebbe rimanere bloccato, invio tramite la porta 21.
Inoltre, differentemente da HTTP, FTP prevede:

- Una connessione persistente, la quale vive finchè non viene chiusa dall'utente o scatta un timeout;
- Un trasferimento bidirezionale tra le due macchine.

Si noti che per utilizzare il protocollo FTP è necessaria un'autenticazione.

SMTP, IMAP e POP3

Introduzione alla posta elettronica

In questo paragrafo analizzeremo il funzionamento della posta elettronica. Vedremo quindi le tre componenti che la riguardano, ovvero:

- **UserAgent**;
- **Server di Posta** (o Mail Server);
- Il protocollo **SMTP**.

Spieghiamo meglio il tutto con un esempio, supponendo una comunicazione da $Alice$ verso Bob.
Entrambi utilizzeranno uno UserAgent come *Apple Mail* o *GMail*, i quali permettono di scrivere, leggere ed inviare messaggi di posta elettronica.
Scritto un messaggio da parte di $Alice$, il suo UserAgent lo invierà al suo server di posta, il quale lo inoltra in una coda di messaggi in uscita verso il server di posta di Bob. Quando Bob vuole leggere il messaggio, il suo UserAgent lo recupera dalla casella di posta del suo server di posta. Nel caso in cui $Alice$ non riuscisse a comunicare con il Server di Posta di Bob, trattiene il messaggio in coda, riprovando ogni trenta minuti. Se dopo alcuni giorni ancora non è possibile inviare il messaggio a Bob, quest'ultimo viene eliminato dalla coda di $Alice$, notificando il tutto.

SMTP

SMTP è il principale protocollo a livello applicativo per la posta elettronica su Internet. Egli nasce nel 1982 (molto prima di HTTP) ed è definito nell'$RFC\ 5321$ SMTP fa uso di TCP e si basa su:

- Un lato *client*, in esecuzione sul Mail Server del mittente;

- Un lato *server*, in esecuzione sul Mail Server del destinatario.

SMTP, quindi, agisce come **client** o **server** in base a chi è il mittente e chi il destinatario. Adesso possiamo descrivere meglio come avviene l'invio del messaggio da $Alice$ verso Bob.

- Come abbiamo già detto, dopo che $Alice$ scrive il suo messaggio grazie al suo UserAgent, questo viene *pushato* nel suo Mail Server e posto in una coda di messaggi in uscita;
- In seguito, il lato client di SMTP eseguito sul Mail Server di $Alice$ apre una connessione TCP (sulla porta 25) verso il lato server di SMTP sul Mail Server di Bob. Se il server è inattivo riprova più tardi.
- Dopo la procedura di *handshaking*, il client SMTP invia il messaggio sulla connessione TCP.
- Dopo che il server SMTP ha ricevuto il messaggio, questo viene posto nella casella di posta di Bob, attraverso protocolli di accesso alla posta come **POP3**.

Lo scambio di messaggi tra client SMTP e server SMTP si basa su comandi come `HELO`, `MAIL FROM`, `RCPT TO`, `DATA` e `QUIT`, tutti molto auto-espicativi. Oltre a questi messaggi relativi a questi comandi viene talvolta inviato un messaggio contenente un solo `.`, ad indicare che il messaggio è terminato.

Si noti che solitamente SMTP non utilizza Mail Server intermedi per inviare la posta. Inoltre, SMTP utilizza sempre **connessioni persistenti**, il che indica che in caso di più messaggi alla stessa persona, questi verranno inviati sulla stessa connessione TCP.

Confronto tra SMTP e HTTP

Abbiamo visto come sia **SMTP** che **HTTP** trasferiscono file da un host all'altro. Sostanzialmente, l'unica cosa in comune tra i due protocolli si basa sulla tipologia di connessione adottata, infatti entrambi utilizzano connesioni persistenti.

Relativamente alle differenze, possiamo notare che HTTP è principalmente un **protocollo pull**, infatti gli utenti attirano a sè le informazioni che qualcuno carica nel server e la connessione inizia dalla macchina che vuole ricevere il file.
D'altra parte, SMTP è un **protocollo push**, infatti il Mail Server di invio *pusha* i file al server e la connessione viene iniziata da chi vuole spedire il file.

Una seconda differenza riguarda **ASCII**, infatti SMTP deve comporre l'intero messaggio (l'intero corpo) in ASCII a 7 bit, dai carattere ai dati binari come le immagini. HTTP non impone questo vincolo.
Una terza ed ultima differenza riguarda l'incapsulamento dell'oggetto. In HTTP, infatti, ogni oggetto viene incapsulato nel proprio messaggio di risposta, mentre in SMTP tutti gli oggetti vengono incapsultati in unico messaggio.

> Si noti che se l'userAgent di Bob si trova nel suo PC locale, lo stesso non vale per il suo Mail Server. Se così fosse infatti quest'ultimo dovrebbe rimanere sempre accesso e connesso ad Internet. Una procedura poco pratica. Per questo motivo

> si tende ad utilizzare un Mail Server condiviso, sempre attivo e gestito dall'ISP, in cui ogni utente possiede la sua casella di posta.

POP3

POP3 è un protocollo di accesso definito in $RFC\ 1939$ che entra in azione quando lo UserAgent (i.e. client) apre una connessione **TCP** verso il suo Mail Server sulla porta 110.
Egli si basa su tre fasi: *autorizzazione*, *transazione* ed *aggiornamento*.
Durante la $fase\ 1$, lo UserAgent invia il nome utente e la password (*in chiaro*) per autenticare l'utente, mentre nella $fase\ 2$ recupera i messaggi; in seguito dopo il comando `quit` e la conseguente chiusura della sessione POP3, inizia la fase di aggiornamento dei messaggi.
Si noti che per ogni transazione POP3, lo userAgent invia comandi ed il server reagisce con due possibili risposte:

- `+OK`, se tutto è andato bene;
- `ERR`, in caso di errori.

Un UserAgent che utilizza POP3 può spesso essere configurato (dall'utente) per "scaricare e rimuovere" oppure "scaricare e mantenere".
Nel primo caso dopo aver letto il messaggio da un dispositivo, non sarà possibile rileggerlo una seconda volta da un secondo dispositivo. Ciò non avviene con la configurazione "scarica e mantieni".

IMAP

Con l'acesso tramite POP3, dopo aver scaricato i messaggi sulla propria macchina è facile pensare che questi possano essere spostati e suddivisi in delle cartelle. Questo paradigma riguardante messaggi e cartelle, purtroppo, causa dei problemi. Il protocollo IMAP risolve questo problema.
IMAP è un altro protocollo di accesso alla posta, più complesso di POP3, in grado di associare ogni messaggio sul server ad una cartella. I messaggi in arrivo sono associati innanzitutto alla cartella INBOX del destinatario, il quale può poi spostare il messaggio in una nuova cartella.
Si noti che una funzione importante di IMAP si basa sulla lettura di singole parti di messaggio, utile in caso di connessione lenta.

Posta elettronica sul Web

Oggi è in costante crescita l'utilizza della propria email mediante browser web. Grazie a questo servizio lo UserAgent diventa il proprio browser, che comunica con la propria casella mediante protocollo HTTP.
Quando un mittente o un destinatario vuole eseguire il push o il pull di un messaggio di posta, allora egli utilizza HTTP. D'altra parte, per eseguire l'invio del messaggio da un Mail Server all'altro è ancora necessario il protocollo SMTP.

DNS

Gli host Internet possono essere identificati in vari modi, infatti, nomi degli host come www.google.com risultano semplici da ricordare, ma quest'ultimi sono identificati anche dai indirizzi IP.
Un indirizzo IP consiste di $4\ byte$ (in cui ogni $byte$ è espresso con un numero da 0 a 255 ed è separato da punti) e presenta una rigida struttura gerarchica, questo perchè leggendolo da sinistra verso destra otteniamo informazioni sempre più specifiche riguardo ad esso; utilizzando un analogia sarebbe come leggere un indirizzo postale dal basso verso l'alto.
Un esempio di indirizzo IP è `185.199.109.153`.

Esistono quindi due modi per identificare gli host, noi umani preferiamo il primo perchè più semplice da ricordare, i router preferiscono il secondo. Per questo motivo deve esser svolto un lavoro di traduzione, compito svolto dal **DNS** (acronimo di *Domain Name System*).

Il **DNS** è (i) un **database** distribuito implementato in una gerarchia di **DNS Server** e (ii) un **protocollo** a livello applicativo che consente agli host di interrogare il database. Con DNS Server intendiamo tendenzialmente macchine UNIX che eseguono un software chiamato **BIND**.
Il protocollo DNS utilizza **UDP** come protocollo a livello di trasporto e la porta 53.

Inoltre, il DNS viene comunemente utilizzato anche da altri protocolli a livello applicazione come **HTTP** ed **SMTP**.

Vediamo passo dopo passo, come il DNS, riesce a svolgere il compito di traduzione.

- L'utente richiede l'URL www.something.com;
- E' necessario ottenere il suo indirizzo IP, per cui il browser estrae il nome dell'host passandolo al lato client dell'applicazione DNS;
- Il client DNS, quindi, invia una query contenente l'hostname ad un DNS Server.
- Il client DNS prima o poi riceverà una risposta che include l'indirizzo IP corrispondente.
- Ricevuto l'indirizzo IP dal DNS, il browser può dare inizio ad una connessione TCP verso il processo server HTTP collegato alla porta 80 di quell'indirizzo IP.

Altri servizi offerti dal DNS sono:

- **Host Aliasing**.
  Sappiamo che un host dal nome complicato può avere uno o più sinonimi.
  Un esempio, `relay.west-coast.enterprise.com` potrebbe avere due sinonimi, quali `enterprise.com` e `www.enterprise.com`.
  In questo caso, chiamiamo il primo nome, più lungo e complicato, *hostname canonico*.
  Il DNS quindi può essere invocato per trovare l'*hostname canonico* a partire da un sinonimo.
- **Mail Server Aliasing**.
  Il concetto esposto per l'host aliasing vale anche per l'email. Infatti l'hostname del server gmail potrebbe essere molto più complicato rispetto al semplice `gmail.com`.
  Un'applicazione di posta può invocare il DNS per ottenere il nome canonico di un sinonimo fornito.
- **Distribuzione del carico di rete**.
  Il DNS viene anche utilizzato per distribuire il carico tra server replicati; in questo

modo i siti con molto traffico vengono replicati su più server. Ad ogni *hostname canonico* sono associati un insieme di indirizzi IP.

> Tutte le query DNS ed i messaggi di risposta sono inviati attraverso **datagrammi UDP** diretti alla porta 53.
> Vengono utilizzati più DNS Server poichè utilizzarne uno solo causerebbe problemi come: (i) guasti, uno solo colpirebbe l'intera Internet (ii) un volume di traffico notevole a causa di centinaia di milioni di richieste, (iii) problemi di locazione del server, quale si troverebbe vicino a pochi e lontano a molti.

Gerarchia dei DNS Server

Come abbiamo già detto, il DNS utilizza e deve utilizzare un grande numero di DNS Server. Possiamo distinguere i DNS server in una gerarchia formata da 3 classi:

- **Root Server**.
  In Internet ne esistono 400, dislocati in tutto il mondo e gestiti da 13 organizzazioni diverse, il loro compito è fornire gli indirizzi IP dei server TLD.
- **TLD (*Top-Level Domain*) Server**.
  Si occupano dei domini di primo livello come `com`, `org`, `net`, `edu` e `gov` ed inoltre a quelli relativi ai vari paesi come `it`, `fr` e `de`. Il loro compito è fornire gli indirizzi IP dei server autoritativi.
- **Server Autoritativi**.
  I Server Autoritativi contengono le associazioni di nomi ad indirizzi IP, per ogni organizzazione. Un organizzazione può quindi implementare il proprio server autoritativo o pagare un fornitore per ospitare questi record nei suoi server.
- **Server Locale**.
  Ultimo ma non per questo meno importante, sono i DNS Server locali. Ogni ISP ne possiede uno, dunque, quando un host si connette ad un ISP, questo gli fornisce un indirizzo IP tratto da uno o più dei suoi DNS server locali.

Consideriamo adesso un esempio per capire meglio il funzionamento del DNS, supponendo che un host voglia l'indirizzo IP di `web.dmi.unict.it`.

- Dapprima viene inviato un messaggio di richiesta DNS al proprio DNS locale, che contiene il nome da tradurre.
- Il server locale allora contatta un root server. Quest'ultimo prende nota del suffisso `it`, restituendo al server locale un elenco di indirizzi IP per i TLD server responsabili di `it`.
- Il server locale quindi contatta con la stessa richiesta uno di questi TLD server.
- Il TLD server prende nota del suffisso unict.it e risponde restituendo l'indirizzo IP del server autoritativo per l'Università di Catania.
- Infine verrà inviato un ultimo messaggio con la stessa richiesta al server autoritativo, che fornirà l'indirizzo IP desiderato.

Si noti che può capire che il TLD server non conosca il server autoritativo in questione, in questo si passerà ad un altro TLD server.

DNS Caching

Una caratteristica di fondamentale importanza è relativa al **caching**. Il DNS server, che riceve una risposta DNS contenente la traduzione, può infatti memorizzare quest'ultima in *cache*.

Record e messaggi DNS

I server che implementano i database, memorizzano i **Record di Risorsa** (RR, Resource Control). Un Record di Risorsa è una tupla contenente i seguenti campi: (`Name`, `Value`, `Type`, `TTL`). In particolare, **TTL** è il Time To Live, ossia il tempo residuo prima che la risorsa venga eliminata dalla cache. Il significato di Name e Value dipende da Type.

- `Type = A`, per cui segue che `Name` sarà il nome dell'host e `Value` il suo indirizzo IP.
- `Type = NS`, per cui segue che `Name` sarà il nome di un dominio (come something.com) e `Value` l'hostname del suo DNS server autoritativo (come `dns.something.com`).
- `Type = CNAME`, per cui segue che `Name` sarà il sinonimo del nome canonico, quale è rappresentato da `Value`.
- `Type = MX`, per cui segue che `Name` sarà il sinonimo di un Mail Server canonico, quale è rappresentato da `Value`.

SNMP, Simple Network Management Protocol

Pur non essendo oggetto di esame, menzioniamo velocemente SNMP, un protocollo per monitorare e svolgere azioni su un dispositivo. E' formato da un Agent che comunica con il Sistema Operativo e di un Manager, colui che invia le richieste dall'esterno.

Livello di Trasporto

Introduzione

Un *protocollo* a livello di trasporto mette a disposizione una **comunicazione logica** tra due processi applicativi di host differenti. Con comunicazione logica intendiamo quella comunicazione che, dal punto di vista dell'applicazione, avviene come se gli host che eseguono i processi fossero direttamente connessi. Questo ovviamente non è vero, ma avviene poichè gli applicativi non si preoccupano dell'infrastruttura fisica sottostante.
I protocolli a livello di trasporto sono implementati nel sistema periferico e *non* nel router.
Come abbiamo visto in precedenza il livello di trasporto (lato mittente) converte i messaggi ricevuti in segmenti (o pacchetti a livello di trasporto).
Questo avviene spezzando se necessario il pacchetto ricevuto in parti più piccole, aggiungendo un intestazione ad ognuno di essi.
Volendo utilizzare un analogia per capire meglio il concetto di comunicazione logica, supponiamo che un gruppo di n ragazzi scriva n lettere ad un'altro gruppo di n ragazzi. I due gruppi abitano in due case diverse ed in due città diverse.
Un ragazzo per gruppo ha il compito di inviare o ritirare le lettere e distribuirle.
In questo caso il servizio postale fornisce comunicazione logica tra le due case mentre il ragazzo A ed il ragazzo B provvedono alla comunicazione logica tra i due gruppi di ragazzi.
Possiamo intendere quindi i messaggi dell'applicazione come le lettere nelle buste, i processi come i ragazzi, gli host come i condomini che abitano le rispettive case, il protocollo a livello di trasporto come A e B, mentre il protocollo a livello di rete come il servizio postale (compresi di postini).

Panoramica del livello di trasporto

Internet, e più in generale una rete TCP/IP, mette a disposizione del livello applicativo due diversi protocolli:

- **UDP**, che fornisce un servizio non affidabile e non orientato alla connessione.
- **TCP**, che fornisce un servizio affidabile ed orientato alla connessione, il che lo rende per ovvii motivi più complesso del precedente.
  Uno sviluppatore sceglie tra i due durante la creazione della socket. Anche se ne parleremo meglio nei prossimi capitoli, facciamo un breve cenno ai protocolli a livello di rete.
  Sappiamo che un pacchetto a livello di rete è chiamato *datagramma* e che il protocollo utilizzato a livello di rete è **IP**.
  Il modello di IP si basa sul *best effort* (i.e. massimo sforzo), il che indica che farà del suo meglio ma comunque non fornisce alcuna garanzia sulla consegna dei segmenti che gli arrivano; in altre parole, non assicura che arriveranno a destinazione o nel giusto ordine, per questo motivo si dice che IP offre un servizio non affidabile.
  Il principale compito di TCP ed UDP è quindi estendere il servizio fornito da IP. In particolare per trasformarlo da consegna tra due sistemi periferici a consegna tra processi in esecuzione su sistemi periferici.

> Questo passaggio descritto sopra, anche chiamato da host-to-host a process-to-process, viene detto multiplexing e demultiplexing a livello di trasporto.

Multiplexing e demultiplexing

Nell'host destinatario il livello di trasporto ha il compito di consegnare i dati dei segmenti al processo appropriato.
Ricordiamo che un processo può gestire una o più socket, attraverso le quali i dati fluiscono dalla rete al processo e viceversa.
Il livello di trasporto, quindi, non invia i dati direttamente al processo bensì alla **socket**.
Ad ognuna di queste socket è associato un *identificatore univoco*, con un formato dipendente dal protocollo scelto (i.e. TCP o UDP).

Definiamo il **demultiplexing** come il compito di trasportare i segmenti verso la giusta socket, mentre il multiplexing come il compito, svolto dal *mittente*, di radunare dati da diverse socket e incapsulare ognuno di questi con intestazioni per creare dei segmenti e passarli al livello di rete.
Richiamando l'**analogia** precedente, diciamo che il ragazzo A effettua un *operazione di multiplexing* quando raccoglie le lettere per poi spedirle, mentre il ragazzo B effettua un *operazione di demultiplexing* quando leggendo il nome consegna la lettera al corretto destinatario.

Il multiplexing quindi richiede (i) che le socket abbiano un identificatore univoco e (ii) che ciascun segmento presenti campi che indichino la socket a cui va consegnato il segmento. I campi a cui facciamo riferimento sono (i) il campo del numero di *porta di origine* ed (ii) il campo del numero di *porta di destinazione*.
In definitiva, possiamo affermare che ogni socket nell'host deve avere un *numero di porta*, in modo tale che quando un segmento arriva all'host, il livello di trasporto possa dirigere il segmento alla socket corrispondente. Infine la socket consegnerà i dati al processo.

Numeri di porta

I **numeri di porta** sono numeri di $16\ bit$ che vanno da 0 a 65535.

- Porte da 0 a 1023 sono chiamati numeri di porta noti (well-known port number)
- Porte da 1024 a 49151 sono chiamate numeri di porta registrati (registered ports)
- Porte da 49152 a 65535 sono porte dinamiche o private.

UDP

Il primo protocollo di trasporto che studiamo è **UDP**, un protocollo che svolge fa il *lavoro minimo* che un protocollo di trasporto debba fare; svolge la funzione di **multiplexing** / **demultiplexing** ed una forma di **controllo degli errori** molto semplice.

In particolare, il lavoro svolto da UDP è prendere i messaggi dal processo applicativo, aggiungere il *numero di porta di origine* e di *destinazione* per il *multiplexing/demultiplexing*, per poi aggiungere altri 2 piccoli *campi* e passare il tutto al livello di rete. Chiaramente, non

esiste *handshaking*, il che rende UDP un protocollo *non orientato alla connessione*.
Un **esempio** di protocollo applicatico che utilizza UDP è **DNS**.

Quali sono i motivi per preferire UDP rispetto a TCP?

- **Controllo più fine a livello applicativo**;
  UDP per la sua natura permette impacchettare e trasferire al livello sottostante i dati ricevuti con tempi inferiori rispetto TCP, il quale non curante dei tempi richiesti, ritarda l'invio dei pacchetti dati i suoi meccanismi di controllo di congestione ed invia segmenti finchè non viene notificata la ricezione per quest'ultima. UDP è per questo motivo preferibile all'interno di applicativi in tempo reale, quali non permettono eccessivi ritardi ma permettono la perdita di un pacchetto.
- **Nessuno stato di connessione**;
  TCP include uno stato della connessione nei sistemi periferici, il che include buffer di ricezione ed invio, parametri per il controllo della congestione, sul numero di sequenza e di ACK.
  UDP, d'altra parte, non conserva lo stato della connessione e non tiene traccia di quest'ultimi parametri. Per questo motivo, un server può generalmente supportare più client attivi quando l'applicazione utilizza UDP rispetto a TCP.
- **Minor spazio utilizzato per l'intestazione del pacchetto**, poiché l'intestazione dei pacchetti TCP aggiunge 20 byte, UDP solo 8.

> Si noti che anche se UDP nasce senza alcun servizio di trasporto affidabile, quest'ultimo servizio può essere fornito se presente nell'applicazione stessa (per esempio con meccanismi di notifica). Un esempio è QUIC (Quick UDP Internet Connection) utilizzato dal browser Chrome.

Struttura dei segmenti di UDP

La struttura di un **segmento UDP** si basa sui seguenti **campi**:

- **Numeri di porta**, di origine e destinazione (entrambi da 16 bit), necessari per effettuare *multiplexing* / *demultiplexing*.
- **Campo lunghezza**, formato da 16 bit, presenta il numero di byte del segmento UDP (*intestazione* + *dati*).
- **Checksum**, formato da 16 bit, per verificare se vi sono errori nei bit che formano il segmento, quindi in altre parole che non siano stati alterati durante il trasferimento. Nel lato mittente il campo checksum viene riempito con il complemento a uno della somma di tutte le parola di memoria da 16 bit nel segmento (l'eventuale riporto finale viene sommato al primo bit). Ricordiamo che il complemento a uno si effettua convertendo i bit 0 in 1 e viceversa. Nel lato ricevente invece verrà effettuata la somma delle parole di memoria più il campo checksum. Se non ci sono errori (rappresentati da bit 0) il risultato dovrà essere formato da soli bit 1. In caso di errori, alcune implementazioni di UDP si limitano a scartare il segmento danneggiato, altre lo trasmettono all'applicazione con un avvertimento.

- **Campo dati**, formato da 32 bit, ossia la parte che contiene il messaggio di richiesta o di risposta.
  Il campo di intestazione di UDP, è formato da 4 campi da 2 byte ciascuno.

Il trasferimento affidabile

Un canale affidabile implica che nessun bit è corrotto o va perduto, per cui tutti i bit sono consegnati nell'ordine di invio.
In questo paragrafo vedremo come sviluppare un trasferimento affidabile per mittente e ricevente, utilizzando modelli di macchine a stati finiti via via più complesse.
Assumeremo quindi che i pacchetti vengano consegnati nel giusto ordine, ma vi sia la possibilita che vengano persi.
Utilizzeremo chiamate come `rdt_send()` (con `rdt` acronimo di *Reliable Data Transfer*) per indicare il trasferimento di dati da parte del mittente, `rdt_rcv()` quando il pacchetto raggiunge il ricevente e `deliver_data()` per inviare i dati al livello superiore. Consideriamo, inoltre, un trasferimento dati unidirezionale poichè un trasferimento dati bidirezionale sarebbe più noioso in termini di spiegazione.

RDT 1.0

Iniziamo la trattazione di RDT con $RDT\ 1.0$, il quale ci mostra la soluzione più semplice e banale per il trasferimento affidabile su un canale affidabile, il tutto attraverso $2\ FSM$ (i.e. *macchina a stati finiti*), una per il mittente ed una per il destinatario.

- **Mittente**

  1. `rdt_send(data)` -> Evento che causa la transizione. Crea un pacchetto contenente i dati ricevuti dal livello superiore grazie alla chiamata `make_pkt(data)`; in seguito, lo invia sul canale attraverso `udt_send(packet)`.

- **Ricevente**

  1. `rdt_rcv(packet)` -> Evento che causa la transizione; prevede che i pacchetti vengano raccolti dal livello sottostante ed i dati estratti dal pacchetto attraverso `extract(packet, data)`. I dati vengono poi passati al livello superiore secondo `deliver_data()`.

All'interno di RDT ; 1.0 tutti i pacchetti fluiscono dal mittente verso il ricevente poiché per ipotesi disponiamo di un *canale perfettamente affidabile*, il che implica che nulla possa andare storto.

> Usiamo la lettera "Λ" a destra o sinistra della freccia se rispettivamente l'evento non causa alcuna transizione o viceversa una transizione si verifica senza il determinarsi di un evento.

RDT 2.0

Un modello più *realistico* riguarda la possibilità che i bit possano essere corrotti, errore che può verificarsi quando il pacchetto viene trasmesso, inserito nel buffer o ritrasmesso. Analizziamo una piccola analogia. All'interno di una chiamata e per ogni frase, i due interlocutori al telefono potrebbero scambiarsi dei "messaggi" di notifica come "*OK*" (ACK), per intendere che la frase è stata compresa oppure "*Ripeti*" (NAK) se la frase non è stata compresa.
Nell'ambito delle reti di calcolatori, protocolli basati sul *ritrasferimento* sono chiamati protocolli *ARQ*.
Protocolli ARQ gestiscono il rilevamento di errori con 3 funzionalità aggiuntive:

- **Rilevamento dell'errore**, un meccanismo in grado di rilevare gli errori sui bit. Ricordiamo che UDP utilizza per questo il campo checksum.
- **Feedback del destinatario**, ovvero le risposta di notifica positiva (ACK) o negativa (NAK). Tali pacchetti di notifica potrebbero essere costituiti da un solo bit, per esempio 0 per NAK ed 1 per ACK.
- **Ritrasmissione**, in quanto un pacchetto con errori sarà ritrasmesso dal mittente.

Le tre funzionalità descritte predentemente compongono $RDT2.0$, il quale è formato dalle seguenti 2 FSM:

- **Mittente**
    1. `rdt_send(data)` -> Una volta ricevuti i dati dal livello sopra, viene creato il pacchetto insieme al checksum per poi inviare il tutto. In seguito si passa al secondo stato sottostante.
    2. Attesa di ACK o NAK -> Il mittente resta in attesa di una notifica da parte del ricevente, il quale se NAK comporterà la ritrasmissione dell'ultimo pacchetto ed il ritorno nel *secondo stato*, quindi l'attesa di una nuova notifica; se ACK avviene il passaggio al *primo stato*, in quanto egli saprà che il pacchetto spedito più di recente è stato ricevuto senza alcun problema. Si noti che finché il mittente si trova nel secondo stato egli sarà impossibilitato ad inviare nuovi pacchetti, il che rende il protocollo $RDT\ 2.0$ ed analoghi dei protocolli $stop - and - wait$.

- **Ricevente**
    1. `rdt_rcv(packet)` && `corrupt(packet)` -> Ricevuto un pacchetto da parte del mittente, allora se quest'ultimo è corrotto (`corrupt(packet)`) allora viene creato ed inviato un pacchetto contenente NAK.
    2. `rdt_rcv(packet)` && `not_corrupt(packet)` -> Ricevuto un pacchetto da parte del mittente, allora se quest'ultimo non è corrotto (`not_corrupt(packet)`) allora vengono estratti i dati al suo interno per poi inviare quest'ultimi al livello superiore, quindi viene creato ed inviato un pacchetto contenente ACK.

RDT 2.1

Se $RDT\ 2.0$ sembra funzionare bene, in realtà così non è; dimentichiamo che anche ACK o NAK potrebbero essere corrotti. Risolviamo questo problema attraverso $RDT\ 2.1$.
Considerando l'**analogia** basata sulla telefonata, notiamo che la risposta "*OK*" o "*Ripeti*"" da parte di A possa non esser stata capita, per cui segue la domanda di B: "*Che cos'hai detto?*" a cui a sua volte potrebbe succedere un "Che cos'hai detto tu?" da parte di A e così via...
Per risolvere il problema andiamo ad aggiungere un numero identificativo ai pacchetti. Questo numero identificativo avrà valore 0 o 1. Il mittente invia il primo pacchetto con numero identificativo 0 e si mette in attesa di un ACK da parte del ricevente. Il ricevente invierà un ACK e si metterà in attesa del pacchetto 1 o un NAK se riceve un pacchetto corrotto. Il mittente, se riceve una risposta corrotta o se riceve un NAK reinvia il pacchetto, se riceve un ACK va ad inviare il pacchetto 1. Potrebbe capitare che la risposta corrotta sia un ACK, in questo caso, quando il ricevente riceve nuovamente un pacchetto con il codice precedente reinvierà l'ACK per il pacchetto precedente. In questo modo il mittente si risincronizzerà e potranno continuare con la comunicazione.

RDT 2.2

All'interno di $RDT\ 2.2$, il destinatario, per segnalare al mittente di aver ricevuto un pacchetto corrotto, spedisce un ACK per l'ultimo pacchetto ricevuto correttamente, invece di inviare un segnale NAK. Così facendo il mittente che riceve due ACK per lo stesso pacchetto(ACK duplicati) sa che il destinatario non ha ricevuto quello successivo e quindi lo ritrasmette.

RDT 3.0

Supponiamo che il canale oltre a danneggiare i bit possa anche **perdere pacchetti**.
Soluzione è in questo caso quella di utilizzare un **timer**: il *mittente* inizializza un *contatore* ogni volta che invia un pacchetto, motivo per cui se non riceve un ACK per quel pacchetto prima che il timer termini allora egli ritrasmette quest'ultimo. Stimare il tempo da settare per un timer è molto difficile in quanto non si può mai avere la certezza che questo sia abbastanza *grande* da non introdurre troppi pacchetti duplicati ed abbastanza *piccolo* da non perdere troppo tempo prima di ritrasmetterlo. $RDT\ 3.0$ introduce un *timer* e le primitive ad esso associate al fine di gestire la *perdita di pacchetti*.

Pipelining

RDT 3.0 risulta corretto dal punto di vista dell'affidabilità, tuttavia è inefficiente relativamente alle prestazioni. A causare le cattive prestazioni di quest'ultimo è la strategia da egli utilizzata: *stop and wait*, per cui il mittente non può spedire il pacchetto $i+1$ se non ha prima ricevuto un ACK per il pacchetto i.

Una soluzione si basa sull'applicazione del metodo **pipelining**, per cui è possibile inviare pacchetti senza aspettare l'ACK per il pacchetto precedente da parte del ricevente (e.g. se il mittente invia 3 pacchetti alla volta senza aspettare l'ACK, implica triplicare il *throughput*).

E' possibile *implementare il pipelining* attraverso: **Go-Back-N** e **Ripetizione Selettiva**.

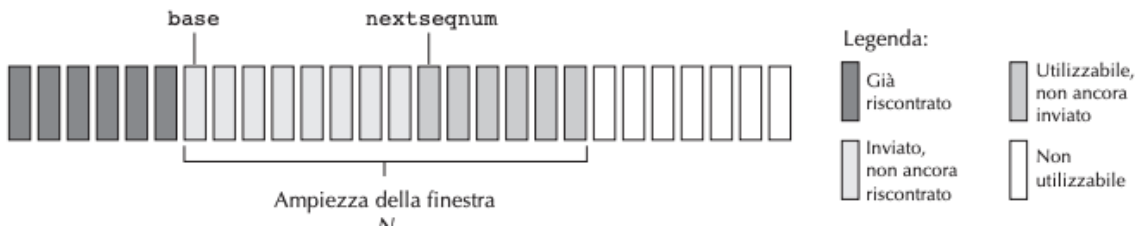
Go-Back-N (GBN)

Una prima implementazione del pipelining si basa su **Go-Back-N**, il quale permette al mittente di poter inviare più pacchetti senza rimanere in attesa di un ACK; tuttavia, il numero di pacchetti inviati dal mittente che non hanno ancora ricevuto un ACK non può essere maggiore di N.

All'interno della *pipeline*, possiamo distinguere 4 diversi segmenti (i.e. *intervalli*). Possiamo definire `base` come l'indice del pacchetto più vecchio ed ancora senza ACK, mentre `nextseqnum` come l'indice del prossimo pacchetto da inviare al destinatario. Gli intervalli sono i seguenti:

- $(0, base-1)$, formato da tutti i pacchetti inviati che hanno ricevuto un ACK da parte del ricevente.
- $(base, nextseqnum-1)$, formato da tutti i pacchetti inviati che *non* hanno ricevuto un ACK da parte del ricevente.
- $(nextseqnum, base+N-1)$, formato da tutti i pacchetti che possono essere ancora inviati al ricevente.
- $(base+N, \dots)$, formato da tutti i pacchetti che *non* possono essere inviati al ricevente se il mittente non riceve prima un ACK per un pacchetto all'interno dell'intervallo $(base, nextseqnum-1)$.

Per questo motivo, N è spesso chiamata ampiezza della finestra ed il protocollo **GBN** definito **protocollo a finestra scorrevole**.



Per concludere, denotiamo le caratteristiche di mittente e destinatario GBN descritti attraverso le rispettive *FSM*. Un **mittente GBN** deve rispondere a 3 tipi di evento:

- *Invocazione dall'alto*, per cui se vengono ricevuti dati dal livello soprastante, egli crea ed invia il pacchetto se e solo se la finestra non è piena, ovvero non vi sono già N pacchetti in attesa di ACK. Se la finestra è piena, i dati ricevuti vengono ritornati al livello soprastante, il quale riprova più tardi.
- *Ricezione di un ACK*, il quale viene definito **cumulativo** ed indica che un ACK per il pacchetto n implichi la corretta ricezione di tutti i pacchetti con indice minore di n.
- *Evento di timeout*, che nel caso in cui si verifichi prevede che il mittente invii nuovamente tutti i pacchetti spediti che non hanno ancora ricevuto un ACK. Se il mittente riceve un ACK ed esistono ancora pacchetti in $(base, nextseqnum-1)$, il timer viene fatto ripartire. Altrimenti, se dopo la ricezione di un ACK non esistono più pacchetti in $(base, nextseqnum-1)$, il timer viene stoppato.

D'altra parte, le azioni del destinatario sono le seguenti:

- Se un pacchetto con un indice n viene ricevuto correttamente ed in ordine (ossia l'ultimo pacchetto ricevuto possedeva indice $n-1$), allora il destinatario invia un ACK per il pacchetto con indice n.
- In tutti gli altri casi, il destinatario *scarta il pacchetto* ed invia al mittente un ACK per l'ultimo pacchetto ricevuto correttamente. Ciò accade banalmente anche se un pacchetto è stato ricevuto correttamente ma non è in ordine. Il motivo è il seguente: pur ipotizzando che il pacchetto $i+1$ (*non in ordine*) venga conservato in attesa del pacchetto i, se il pacchetto i viene perso durante la trasmissione, allora GBN impone di rispedire sia i che $i+1$ (i.e. da i in poi). Conseguentemente, il destinatario può scartare il pacchetto non in ordine a fronte di una maggiore semplicità d'uso.

> L'operazione di ritrasmissione può risultare dispendiosa, tuttavia permette al destinatario di conservare il solo numero di sequenza successivo per il prossimo pacchetto.

Ripetizione Selettiva

Il problema di GBN esiste nel momento in cui l'ampiezza della finestra è abbastanza grande da dover rispedire una quantità ingente di pacchetti nel caso in cui uno di essi viene perduto.
Un protocollo a **ripetizione selettiva** risolve quest'ultimo problema considerando una finestra in cui possono esistere pacchetti che hanno ricevuto e che non hanno ricevuto l'ACK.
In questo senso, il destinatario che riceve un pacchetto non in ordine, invia un ACK al mittente e mantiene quest'ultimo in un buffer finché non sono stati ricevuti i pacchetti mancanti. In altre parole, il ricevente invia al livello soprastante il tutto (il pacchetto ricevuto più i pacchetti nel buffer) se ha ricevuto fino al pacchetto `rcv_base`, ossia il primo pacchetto non in ordine.
D'altra parte, la finestra del mittente scorre esclusivamente nel caso in cui venga ricevuto un ACK per il pacchetto inviato meno di recente ed ancora in attesa dI ACK.

TCP

TCP è il protocollo di Internet a livello di trasporto. TCP gode delle seguenti **caratteristiche**:

- E' **orientato alla connessione**, in quanto prima di effettuare lo scambio dei dati, i processi effettuano un *handshake*: una serie di segmenti utili a stabilire i parametri per il trasferimento dei dati. In particolare, parliamo di un *handshake a tre vie*, così definito poiché tre sono i segmenti necessari ad instaurare una connessione tra client e server.
- E' in **esecuzione** all'interno delle due macchine periferiche e non negli elementi intermedi come router e switch.

- Offre un servizio **full-duplex**, motivo per cui i dati tra 2 host, per esempio $Alice$ e Bob, possono fluire da $Alice$ verso Bob ed allo stesso tempo da Bob verso $Alice$.
- Offre un servizio **punto-punto**, motivo per cui è possibile un trasferimento tra un singolo mittente ed un singolo destinatario.
- *Non* offre un servizio **multicast**, motivo per cui il trasferimento da un mittente verso più destinatari non è possibile.

Come viene stabilita una **connessione** tra 2 host secondo TCP? Come abbiamo già scritto, instaurare una connessione prevede un handshake a tre vie, il quale si compone di (i) una richiesta di connessione del client verso il server, (ii) una conseguente risposta dal server e (iii) un terzo ed ultimo segmento può trasportare payload. Una votla concluso l'handshake, client e server possono scambiarsi dati provenienti dal livello applicativo.

Come avviene il **trasferimento** dei dati su una connessione TCP tra 2 host? Instaurata una connessione è possibile l'invio dei dati, motivo per cui il processo client può inviare un flusso di dati attraverso la socket, il quale può essere intesa come il punto di contatto tra il processo ed il protocollo a livello di trasporto. Non appena i dati sono nella mani di TCP, egli dirige i dati al **buffer di invio**, all'interno del quale saranno conservati finché TCP non li preleva per inviarli. Secondo un RFC, TCP afferma di prelevare i dati dal buffer "*quando è più conveniente*".
Quando i dati vengono ricevuti da parte del destinatario, quest'ultimi vengono memorizzati all'interno del **buffer di ricezione**, da cui può leggere il processo applicativo.

Qual è la **massima quantità di dati** posizionabile e prelevabile nel buffer di invio? La quantità massima di dati, talvolta definita come dimesione massima di segmento (i.e *MSS*), è impostata considerando la relativa intestazione e sopratutto la lunghezza massima per una frame (i.e. *MTU*); per questo motivo, dato l'utilizzo di Ethernet (con $MTU = 1500\ byte$) ed un intestazione normalmente pari a $40\ byte$, segue un $MSS = 1460\ byte$.
E' importante notare che MSS non è la massima dimensione di un segmento TCP, bensì la massima quantità di dati proveniente dal processo applicativo posta all'interno del segmento TCP (payload).

Com'è formato un **segmento TCP**? Un segmento TCP è formato dalla seguente struttura:

- **Numero di porta di origine**, un campo formato da $16\ bit$;
- **Numero di porta di destinazione**, un campo formato da $16\ bit$;
- **Numero di sequenza** per il determinato segmento, un campo formato da $32\ bit$ ed utilizzato da mittente e destinatario per implementare il trasferimento affidabile. Più nello specifico, il numero di sequenza per un dato segmento può essere inteso come l'indice del byte all'interno del flusso di byte. In altre parole, ipotizziamo un flusso di byte da inviare da $Alice$ verso Bob uguale a $500\,000\ byte$, con un $MSS = 1000\ byte$ ed il primo byte del flusso sia numerato con 0; segue che il primo segmento avrà numero di sequenza uguale a 0, il secondo uguale a 1000, il terzo 2000 e così via.
- **Numero di ACK**, un campo formato da $32\ bit$ ed utilizzato da mittente e destinatario per implementare il trasferimento affidabile;
- **Campo finestra di ricezione**, un campo formato da $16\ bit$ ed utilizzato per il controllo di flusso (il numero di byte che il destinatario è disposto ad accettare). Il controllo di flusso è un servizio utile affinché il mittente non saturi il buffer del destinatario;

- **Campo lunghezza dell'intestazione**, un campo formato da $16\ bit$ ed utilizzato per specificare la lunghezza dell'intestazione TCP, la quale possiede lunghezza variabile a causa del campo di opzioni.
- **Campo checksum**, un campo formato da $16\ bit$;
- **Campo opzioni**, un campo formato da lunghezza variabile ed utilizzato in modo facoltativo da mittente e destinatario per concordare alcuni parametri per il trasferimento dei dati;
- **Campo flag**, un campo formato da $6\ bit$ e formato a sua volta dai seguenti bit:
    - **ACK**, utilizzato per indicare che il valore trasportato nel *campo di ACK* è valido, ossia che il segmento contiene un ACK per un segmento che è stato ricevuto con successo.
    - **RST**, **FIN** e **SYN**, utilizzati per impostare e chiudere la connessione;
    - **PSH**, se pur quasi mai utilizzato, se 1 indica che il destinatario dovrebbe inviare immediamente i dati appena ricevuti al livello superiore;
    - **URG**, se pur quasi mai utilizzato, indica che nel segmento esistono dati "*urgenti*"

- **Campo puntatore a dati urgenti**, un campo formato da $16\ bit$, il quale punta all'ultimo byte di dati urgenti.

Come reagisce TCP alle possibile **perdite di pacchetti**? Per far fronte alle possibili perdite di pacchetti, TCP fa uso di un **timer**. Definito **RTT** come il tempo che intercorre tra l'invio del segmento e la ricezione del suo ACK, è facile notare che il valore del timer per TCP deve essere maggiore di RTT.

Com'è possibile **stimare** il valore di **RTT**? Il valore di RTT, denotato $SampleRTT$, viene valutato esclusivamente per un solo pacchetto degli n pacchetti inviati per cui non si è ancora ricevuto ACK (non appartenenti all'insieme di pacchetti da ritrasmettere), il che indica approsimativamente il calcolo di $SampleRTT$ ad ogni RTT. I valori RTT possono variare ad ogni calcolo, motivo per cui TCP effettua una stima di quest'ultimi, calcolando una *media ponderata* uguale ad $EstimatedRTT = (1 - \alpha) * EstimatedRTT + \alpha * SampleRTT$. Si noti che la $EstimatedRTT$ attribuisce maggiore importanza ai campione recenti rispetto i campioni passati. Quest'ultima media ponderata è meglio definita con il nome di **EWMA** (acronimo di *Exponentially Weighted Moving Average*), una media pesata esponenziale mobile.
Oltre ad avere una stima di RTT, è anche importante possedere la misura della sua variabilità; per questo motivo, possiamo definire $DevRTT = (1 - \beta) * DevRTT + \beta * (SampleRTT - EstimatedRTT)$. Si noti che $DevRTT$ non è altro che un EWMA della differenza tra $SampleRTT$ ed $EstimatedRTT$; inoltre, minori saranno le *fluttuazioni*, minore sarà il valore di $DevRTT$.

Come viene calcolato il valore per il timer in TCP? Abbiamo notato esistere un legame tra il timer ed i vari parametri legati a loro volta al concetto di RTT. Possiamo affermare che il valore per il timer, denotato $TimeoutInverval$, è uguale al tempo medio per RTT più un certo margine moltiplicato per il numero di fluttuazioni. In simboli matematici, $TimeoutInverval = EstimatedRTT + 4 * DevRTT$.

TCP utilizza un solo timer, anche in presenza di riavvia il timer ogni qual volta riceve un ACK ed esistono ancora dei pacchetti in attesa di ACK. In caso di *timeout* del timer, il pacchetto viene ritrasmesso e conseguentemente viene riavviato il timer.

> E' importante notare che, secondo un RFC, il valore iniziale per il timer è uguale ad $1\ s$. In caso di *timeout* del timer, il valore del timer viene raddoppiato per poi essere aggiornato (secondo la formula vista) alla ricezione del prossimo segmento.

Quali sono gli **eventi principali** all'interno di TCP? Esistono 3 eventi principali all'interno di TCP:

- **Arrivo di dati dal livello applicativo**, per cui TCP incapsula i dati all'interno di un segmento e lo passa ad IP. Se non esiste già un segmento in attesa di ACK, quindi il timer non è in funzione, allora quest'ultimo viene attivato per il segmento appena passato ad IP. E' utile pensare che il timer sia legato al segmento in attesa di ACK inviato più nel passato.
- **Timeout per il timer**, per cui TCP ritrasmette il segmento che lo ha causato, riavviando poi il timer;
- **Ricezione di un ACK**, ossia l'arrivo di un segmento di ACK con un valore y valido nel campo ACK. In questo caso, TCP confronta il valore y con la sua variabile $SendBase$. Ricordiamo che $SendBase$ è l'indice per il pacchetto inviato più nel passato ed ancora in attesa di ACK. Poiché TCP utilizza ACK cumulativi, il valore y conferma la ricezione di tutto ciò che è precedente ad esso. Pertanto, se il valore y è maggiore o uguale di $SendBase$, allora l'ACK si riferisce ad uno o più segmenti che non avevano ancora ricevuto conferma. Il mittente, quindi, aggiorna la variabile $SendBase$.

TCP è un protocollo GBN o SR? TCP può essere inteso come un ibrido tra i protocolli di tipo GBN ed SR.
Ricordiamo che gli ACK TCP sono cumulativi e che i segmenti ricevuti correttamente, ma in modo disordinato, non vengono notificati singolarmente dal destinatario. Quindi, il mittente TCP deve solo memorizzare il numero di sequenza più basso tra i byte trasmessi che non hanno ancora ricevuto acknowledgment ($SendBase$) e il numero di sequenza del successivo byte da inviare ($NextSeqNum$). In questo senso, TCP somiglia molto a un protocollo di tipo GBN.
Tuttavia, esistono delle differenze tra TCP e GBN. Supponiamo che il mittente invii $0, \dots, N$ segmenti al destinatario, i quali arrivano in ordine e senza errori; il destinatario risponde con degli ACK per i segmenti da $0, \dots, n-1$, finché un ACK per un segmento $n < N$ viene perso. In questo scenario, il protocollo GBN ritrasmetterebbe tutti i pacchetti da n ad N. TCP, invece, ritrasmette al più il segmento n; TCP, inoltre, potrebbe addirittura non ritrasmettere alcun segmento se l'ACK per il segmento $n+1$ arriva al mittente prima del timeout del timer.
Una modifica proposta per TCP è il *riscontro selettivo*, per cui il destinatario può inviare un ACK selettivo per i segmenti non in ordine, piuttosto che uno cumulativo per l'ultimo segmento senza errori ed in ordine.
Il riscontro selettivo e la scelta nativa di TCP di non ritrasmettere tutti i segmenti da n ad N nel caso dello scenario precedente, rendono quest'ultimo simile anche ad un generico protocollo SR.

In definitiva e come detto inizialmente, TCP può essere inteso come un ibrido tra i protocolli di tipo GBN ed SR.

Cos'è la **ritrasmissione rapida** in TCP? Uno dei problemi legati alle ritrasmissioni di pacchetti è relativo al periodo di timeout del timer, il quale può rivelarsi relativamente lungo. Fortunatamente, il mittente può rilevare la perdita ancor prima che scada il timer: attraverso gli ACK **duplicati**.
Quando TCP riceve un segmento non in ordine, ovvero con numero di sequenza maggiore rispetto a quello aspettato, rileva un segmento mancante nel flusso; per questo motivo, il destinatario invia un ACK relativo all'ultimo segmento ricevuto in ordine, duplicando così l'ACK per un segmento.
Si noti che TCP invia un ACK duplicato per ogni pacchetto ricevuto non in ordine, motivo per cui se il mittente invia $0, 1, 2, 3$ segmenti ma il destinatario riceve 0, e poi 2 e 3, allora egli invia l'ACK per 0 per poi inviare due volte l'ACK per il segmento 1.

Cos'è il **controllo di flusso**? Il controllo di flusso è uno dei servizi offerti da TCP, utile ad evitare che il mittente saturi il buffer del destinatario. Si basa banalmente sul confronto tra la velocità di invio da parte del mittente e la velocità di lettura dei dati da parte del ricevente.
L'implementazione del servizio di controllo di flusso è possibile mediante una variabile conservata dal mittente: la **finestra di ricezione**, utile a mostrare al mittente lo spazio libero all'interno del buffer del destinatario. Chiaramente, essendo TCP un protocollo *full-duplex*, questa variabile è conservata da entrambi gli host nella connessione.
Supponendo l'invio di un file di grande dimensioni da $Alice$ verso Bob.

- Bob alloca un buffer di ricezione per la connessione, cui dimensione è denotata con $RcvBuffer$ e legge ogni tanto dal buffer.
  Bob tiene traccia di due variabili: (i) $LastByteRead$, uguale all'ultimo byte letto da Bob, (ii) $LastByteRcvd$, uguale all'ultimo byte copiato nel buffer di Bob.
  Chiaramente, poiché è possibile mandare in overflow il buffer di Bob, la dimensione del buffer $RcvBuffer$ deve essere maggiore o uguale alla differenza tra i byte ricevuti (i.e. $LastByteRcvd$) ed i byte letti (i.e. $LastByteRead$).
  Definiamo la variabile finestra di ricezione con il parametro $rwnd = RcvBuffer - (LastByteRcvd - LastByteRead)$. Bob comunica $rwnd$ ad $Alice$ scrivendo il suo valore all'interno del campo apposito del segmento. Inizialmente il valore di $rwnd$ è uguale ad $RcvBuffer$.
- $Alice$, d'altra parte, tiene traccia di due variabili: (i) $LastByteSent$, uguale all'ultimo byte inviato a Bob, (ii) $LastByteAcked$, uguale all'ultimo byte per cui si è ricevuto ACK da Bob. Possiamo calcolare il numero di bytes senza ACK inviati da $Alice$ attraverso $NoAckedBytes = LastByteSent - LastByteAcked$.

Se il numero di bytes senza ACK, $NoAckedBytes = LastByteSent - LastByteAcked$, viene mantenuto minore o uguale rispetto la variabile finestra di ricezione (i.e. $NoAckedBytes \leq rwnd$) allora non vi è pericolo di *overflow* per il buffer di Bob da parte di $Alice$.

Supponiamo che ad un certo tempo i, il buffer di Bob sia pieno. Supponiamo, quindi, che ad un certo tempo $i + 1$, $Alice$ comunichi (mediante ACK per un segmento) che si è liberato dello spazio all'interno del buffer. Cosa succede se l'ACK per il segmento comunicante lo spazio libero viene perso? Chiaramente i due interlocutori risulterebbero bloccati.
Per risolvere quest'ultimo problema, TCP utilizza un secondo timer: **Persistent Timer**, il

quale scatta nel momento in cui $rwnd = 0$, ovvero il buffer del destinatario è pieno. Al timeout del timer, il mittente invia un messaggio di sveglia al ricevente; se quest'ultimo non risponde al messaggio, la connessione viene chiusa.

Per evitare che il mittente possa inviare bytes non appena riconosce il minimo "*spiraglio*" nel buffer, la **soluzione di Clark** prevede che il destinatario non comunichi subito la variabile finestra di ricezione nel caso in cui si sia liberato poco spazio.

> E' importante non confondere il controllo di congestione con il controllo di flusso. Infatti, entrambi, per motivi diversi, causano un rallentamento al mittente ma il primo avviene a causa di ritardi dovuti al traffico nella rete.

Cos'è l'**algoritmo di Nagle**? Ad un certo tempo si pensò fosse una pratica inefficiente quella in cui fosse possibile inviare segmenti anche e solo per un singolo byte (e.g. Telnet). Per evitare ciò, Nagle propose l'algoritmo omonimo.
L'algoritmo di Nagle afferma che, dopo aver inviato un segmento a tempo i, posso inviare un segmento a tempo $i + 1$ se: (i) esiste un quantitativo x di dati che devono essere inviati verso il destinatario, (ii) esiste $rwnd > MSS$, quindi lo spazio libero all'interno del buffer del destinatario è maggiore rispetto la dimensione massima per il payload nel segmento e (iii) esiste $x > MSS$, ovvero il numero x di byte da inviare è maggiore di quelli inseribili all'interno del segmento. Se una di queste condizioni non è rispettata, il mittente ha tempo di accumulare dati finché non arriva l'ACK per il segmento inviato a tempo i.
L'algoritmo di Nagle permette di sfruttare al meglio un segmento, migliorando le prestazioni del sistema; d'altra parte, l'utilizzo dell'algoritmo prevede un decremento dell'interattività per il sistema.

Come viene **stabilità una connessione** TCP? Oppure, come avviene l'**handshake a tre vie** in TCP? Supponiamo che un processo in un host (client) voglia inizializzare una connessione con un altro processo in un altro host (server). Inizialmente, il processo client comunica con il lato client TCP, il quale a sua volta instaura una connessione TCP con il server mediante i seguenti passi:

1. Invia un segmento speciale al TCP lato server. Questo primo segmento speciale è chiamato **segmento SYN** ed è caratterizzato da: (i) nessun payload, (ii) bit $SYN = 1$ e (iii) campo numero di sequenza uguale ad un valore $ClientSeqNum$ generato più o meno casualmente.
2. Il server riceve il segmento SYN, motivo per cui alloca il buffer e le variabili per la connessione. In seguito, risponde al client con un segmento speciale: un **segmento SYNACK**, caratterizzato da: (i) nessun payload, (ii) bit $SYN = 1$, (iii) $ACK = ClientSeqNum + 1$ e (iv) un campo numero di sequenza uguale ad un valore $ServerSeqNum$ scelto dal server stesso.
3. Il client riceve il segmento SYNACK, motivo per cui alloca il buffer e le variabili per la connessione. In seguito, risponde al server con un terzo ed ultimo segmento speciale, caratterizzato da: (i) possibilità di contenere payload, (ii) bit $SYN = 0$, (iii) $ACK = ServerSeqNum + 1$.

Come viene **terminata una connessione** TCP? Una connessione tra due host può essere stabilità e talvolta può essere anche terminata. Entrambi gli interlocutori hanno la possibiltà

di chiudere la connessione.
Supponiamo che l'host client voglia chiudere la connessione con il server. Fare ciò è possibile mediante i seguenti passi:

1. Il client invia al server un segmento speciale caratterizzato dal bit $FIN = 1$.
2. Il server riceve il segmento speciale e spedisce il proprio segmento speciale caratterizzato anch'esso da bit $FIN = 1$.
3. Il client riceve il segmento speciale da parte del server e risponde con un ACK verso il server.

Terminare la connessione implica la deallocazione di buffer e variabili per entrambi gli interlocutori.
E' importante notare che la chiusura della connessione è **unidirezionale**, diversamente all'apertura che è invece *bidirezionale*. E' possibile, quindi, chiudere la connessione da $Alice$ verso Bob ma tenere aperta la connessione da Bob verso $Alice$. Se il client chiude la connessione verso il server, il server può continuare ad inviare dati verso il client, ottenendo i relativi ACK ma mai dei dati.
Esiste, inoltre, la possibilità che $Alice$ pensi che la connessione sia ancora aperta in modo bidirezionale mentre Bob pensa che è aperta in modo unidirezionale verso di egli; ciò avviene se il segmento FIN di Bob, che vuole terminare la connessione, viene perso.

Cos'è il **controllo di congestione**? Possiamo definire la congestione di rete come il tentativo da parte di troppe sorgenti di inviare dati a ritmi troppo elevati, motivo per cui possiamo definire il controllo di congestione come un metodo per adeguare l'attività dei mittenti in relazione al traffico.
TCP utilizza un controllo di congestione **end-to-end**; infatti, il livello IP non offre alcun supporto riguardo il controllo di congestione, motivo per cui TCP deve necessariamente utilizzare un approccio end-to-end ed essere in grado di individuare la congestione attraverso la perdita di segmenti, alle quali consegue il decremento dell'ampiezza della propria finestra.
TCP consiste nell'imporre a ciascun mittente un **limite alla velocità di invio** sulla propria connessione in funzione della congestione di rete percepita. Se il mittente TCP si accorge di condizioni di *scarso traffico* sul percorso che porta alla destinazione, incrementa il proprio tasso trasmissivo; se, invece, percepisce traffico lungo il percorso, lo riduce. Tale approccio solleva tre domande:

- Come può il mittente TCP **limitare la velocità** di invio del traffico sulla propria connessione? TCP è in grado di ridurre la velocità di invio del traffico sulla connessione attraverso l'aggiunta di una variabile per gli interlocutori: la finestra di congestione (i.e. `cwnd`). Nello specifico la quantità di dati che non hanno ancora ricevuto ACK non può essere inferiore al minimo tra `cwnd` ed `rwnd`.
  $NoAckedBytes = LastByteSend - LastByteAcked \leq min\{cwnd, rwnd\}$.
- Come **percepisce** la **congestione** sul percorso che porta alla **destinazione**? TCP è in grado di individuare la congestione attraverso la perdita di segmenti, ovvero l'occorrenza di un timeout o la ricezione di 3 ACK duplicati da parte del destinatario. Si noti che TCP considera gli ACK come indicazione che i segmenti sono arrivati con successo motivo per cui in base alla frequenza con cui arrivano al mittente, egli decide in che misura ampliare la finestra di congestione. In altre parole, più alta è la

frequenza con cui arrivano gli ACK e con più rapidità verrà ampliata la finestra di congestione. Poiché TCP utilizza gli ACK per "*temporizzare*" gli incrementi per la finestra di congestione, possiamo affermare che TCP sia **auto-temporizzato**.

- Quale **algoritmo** dovrebbe essere usato dal mittente per variare la velocità di invio in funzione della congestione end-to-end? Il celebrato **algoritmo di controllo di congestione di TCP** si basa su 3 **componenti** principali:
    - **Slow start**, componente obbligatoria per i mittenti TCP. Quando si stabilisce una connessione TCP, il valore di `cwnd` viene in genere inizializzato a 1 MSS, il che comporta una velocità di invio iniziale di circa MSS/RTT. . Dato che la banda disponibile alla connessione può essere molto più grande di MSS/RTT, durante la fase iniziale, detta **slow start**, il valore di cwnd parte da 1 MSS e si incrementa di 1 MSS ogni volta che un segmento trasmesso riceve un ACK. Questo processo ha come effetto il raddoppio della velocità trasmissiva a ogni RTT. Quindi, in TCP, la velocità di trasmissione parte lentamente, ma cresce in modo *esponenziale* durante la fase di slow start. Tuttavia, quando termina la crescita esponenziale, ovvero la fase di slow start? Slow start può terminare in 3 modi:
        1. Nel caso in cui vi sia un timeout, motivo per cui `cwnd` viene posto ad 1 ed una nuova variabile di stato `ssthresh` uguale a $cwnd/2$. Successivamente a questo evento, inizia un nuovo processo di slow start.
        2. Nel caso in cui, per evitare di raggiungere nuovamente una velocità per cui è nata in precedenza congestione, `cwnd` è uguale ad `ssthresh`. Successivamente a questo evento, TCP entra nella fase *congestion avoidance*.
        3. Nel caso in cui vengono rilevati 3 ACK duplicati. Successivamente a questo evento, TCP opera una *ritrasmissione rapida* ed entra nello stato di *fast recovery*.

    - **Congestion Avoidance**, componente obbligatoria per i mittenti TCP. Come abbiamo già scritto, TCP entra nello stato di congestion avoidance per evitare di raggiungere nuovamente una velocità per cui è nata in precedenza congestione, motivo per cui adotta un approccio più conservativo nell'incremento di `cwnd`, incrementando quest'ultimo di 1 MSS per ogni RTT. Questa operazione di incremento può essere implementata attraverso un incremento di `cwnd` di $MSS * (MSS/cwnd)$ per ogni ACK. Quando **termina** congestion avoidance? Questa seconda fase, che prevede un incremento lineare di `cwnd`, termina in 2 modi:
        - Nel caso in cui vi sia un timeout, motivo per cui `cwnd` viene posto ad 1 ed `ssthresh` uguale a $cwnd/2$. Successivamente a questo evento, inizia un nuovo processo di slow start.
        - Nel caso in cui vengono rilevati 3 ACK duplicati, motivo per cui `cwnd` viene posto ad $cwnd/2$ ed `ssthresh` uguale a $cwnd/2$. Successivamente a questo evento, TCP entra nello stato di *fast recovery*.

    - **Fast recovery**, componente suggerita ma non obbligatoria per i mittenti TCP. In questa terza ed ultima fase, il valore di `cwnd` è incrementato di 1 MSS per ACK

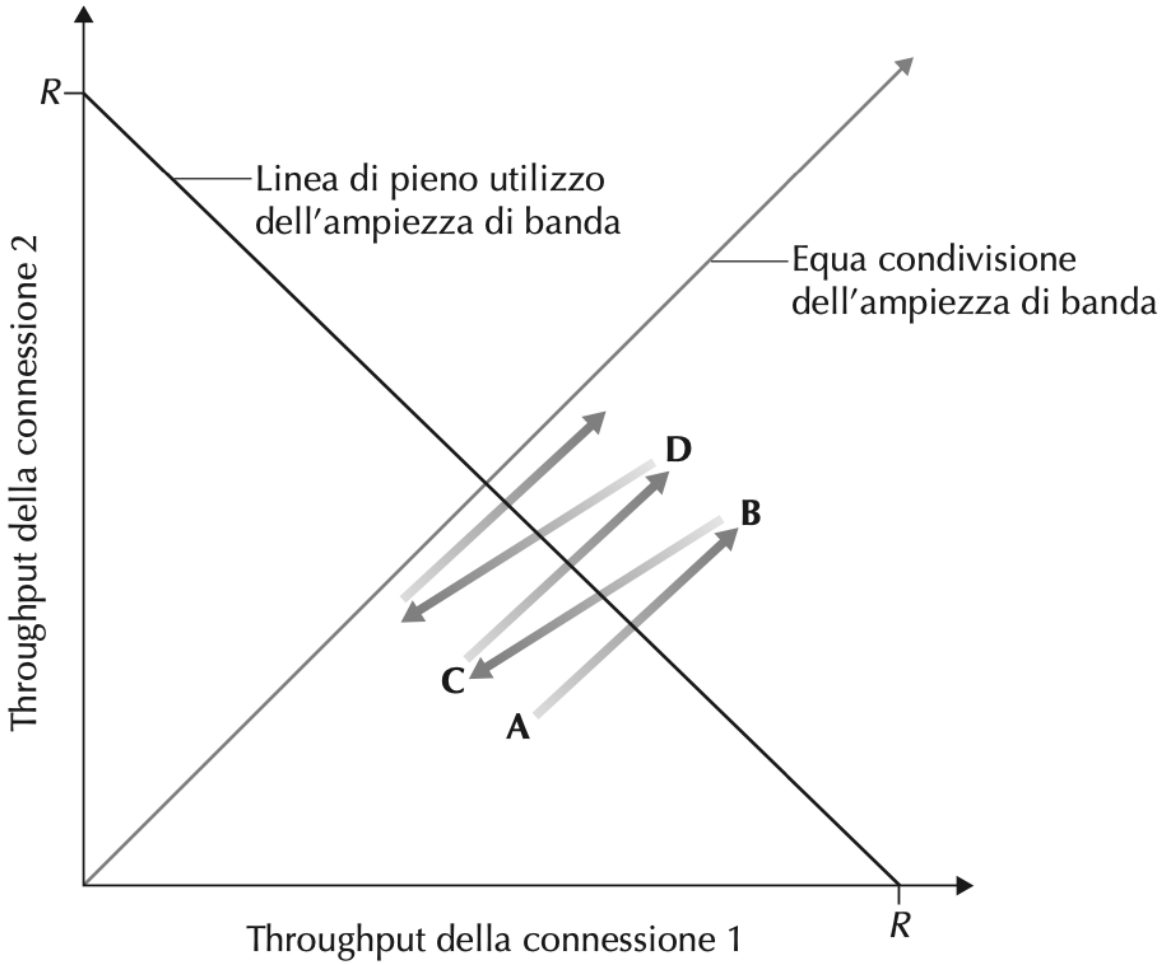
relativo al segmento perso. Se vengono rilevati 3 ACK duplicati nella fase di fast recovery, allora il valore di `cwnd` viene decrementato per poi eseguire una transizione a *congestion avoidance*. Se si verifica un **timeout** nella fase di fast recovery, allora dopo aver impostato il valore di `cwnd` ad 1 ed `ssthresh` uguale a $cwnd/2$, viene effettuata una transizione verso *slow start*. E' importante notare che la fase di fast recovery è consigliata ma non obbligatoria, motivo per cui una prima versione di TCP, ossia **TCP Tahoe**, non faceva uso di quest'ultima fase. **Tahoe**, infatti, impostava `cwnd` ad 1 ed entrava in slow start per ogni evento di perdita. D'altra parte, la versione più recente di TCP, ossia **TCP Reno**, adotta fast recovery. **Tahoe** e **Reno**, al dì fuori dell'adozione di fast recovery, hanno comportamento abbastanza simile; tuttavia, nel caso in cui vengano inizializzate 2 comunicazione TCP all'interno dello stesso canale di comunicazione, allora Reno è in grado occupare più banda rispetto Tahoe.

L'algoritmo di controllo di congestione di TCP precedentemente descritto è spesso indicato come **incremento additivo, decremento moltiplicativo** (**AIMD**, acronimo di *additive-increase multiplicative-decrease*)

> L'algoritmo Vegas tenta di evitare la congestione mantenendo allo stesso tempo un buon throughput. L'idea alla base di Vegas è (i) rilevare la congestione nei router tra origine e destinazione prima che si verifichi un evento di perdita e (ii) abbassare la velocità in modo lineare quando si profila l'imminente perdita di un pacchetto.

Fairness

Un meccanismo di congestione è *fair* se la velocità di trasmissione media per ogni connessione su un collegamento è uguale - o, quasi - per ognuna di esse; in altre parole, se ogni connessione ottiene la stessa porzione di banda.



Supponendo che TCP stia suddividendo la banda in in modo uguale tra 2 connessioni, allora il throughput dovrebbe trovarsi sulla bisettrice in figura.
Il nostro obiettivo è far sì che il throughput sia situato vicino l'intersezione tra la bisettrice ed il segmento tracciato a partire dai punti $(R1,\ 0)$ e $(0,\ R2)$, ossia il punto $(\frac{R}{2},\ \frac{R}{2})$.
In questo modo avremo equa suddivisione della banda.

> I punti $(R1,\ 0)$ e $(0,\ R2)$ indicano rispettivamente l'utilizzo di massimo throughput per la connessione 1 e l'utilizzo di massimo throughput per la connessione 2.

In particolare il segmento tracciato a partire dai punti $(R1,\ 0)$ e $(0,\ R2)$ può essere descritto dall'equazione $y = -x + R$, il che indica una retta con $m = -1$, perpendicolare alla bisettrice.

Come può essere dimostrata la fairness? Supponendo di trovarci a tempo i al punto $A\ (x,\ y)$, ad miglioramento consegue lo spostamento ad un punto $B\ (x + 1,\ y + 1)$, il che benché a sua

volta indichi *nessun miglioramento* e *nessun peggioramento*, il quale si nota attraverso il calcolo della retta passante per i 2 punti, una retta con coefficiente angolare pari ad 1.

D'altra parte, se a partire dal punto B $(x+1, y+1)$ si verifica un peggioramento tale per cui ci spostiamo ad un punto C $(\frac{x+1}{2}, \frac{y+1}{2})$, il che indica permette il raggiungimento di *maggiore equità* rispetto le 2 connessioni.
In particolare, possiamo notare che la retta passante per i 2 punti sia tale da avere termine noto $q = 0$ (i.e. $y = mx$), il che indica una retta passante per l'origine ed il punto C $(\frac{x+1}{2}, \frac{y+1}{2})$ si trovi a metà tra B $(x+1, y+1)$ e $(0, 0)$.

E' importante notare che le 2 connessioni TCP lavorano in modo indipendente, non comunicando in nessun modo. Il sistema appena descritto, infatti, funziona in modo autonomo attraverso un rispetto reciproco tra le connessioni.

> Senza fairness non potrebbe esistere Internet. Infatti, in tale caso, tutte le connessioni successive alla prima non avrebbero modo di ottenere alcuna porzione di banda.

Livello di rete

Introduzione

Il **livello di rete** si occupa principalmente di individuare un **percorso** tra un computer *mittente* e un computer *destinazione* e di trasmettere i pacchetti dalla sorgente alla destinazione attraversando diversi router. La ricerca del percorso (*routing* o *instradamento*) si basa su una **tabella di instradamento** preparata manualmente o automaticamente.
I router si occupano della spedizione dei pacchetti in base all'indirizzo indicato. Esistono, quindi, delle **tabelle di routing** che permettono di scoprire subito dove spedire il pacchetto.

Datagram e Circuiti virtuali

All'interno del livello di rete ci sono due diversi servizi per l'instradamento dei pacchetti:

- **Datagram**.
  In datagram, ogni pacchetto contiene al suo interno l'indirizzo di destinazione completo e ogni "nodo nella rete" effettua le operazioni di routing (capire la strada migliore per giungere a destinazione). Sostanzialmente, ogni pacchetto contiene un indirizzo di destinazione e uno di arrivo che non è detto che siano necessariamente uguali a quelli precedenti o successivi. Per questo motivo, i pacchetti possono prendere strade diverse e arrivare in modo non ordinato o non arrivare, spetterà ai livelli superiori effettuare i vari controlli.
  Si noti che in genere, datagram tende ad appensantire il lavoro dei router (*contro*). Tuttavia (*pro*), con datagram, se un router dovesse guastarsi, verrebbero persi esclusivamente i pacchetti passanti per quel punto (i quali potranno essere recuperati successivamente non appena il sistema nota la perdita).
- **Circuito virtuale**.
  Nei circuiti virtuali il tipo di indirizzamento è lo stesso del datagram ma il calcolo del percorso migliore viene effettuato solo all'inizio. In particolare, durante il calcolo del percorso, svolge un ruolo fondamentale un particolare "*pacchetto esploratore*" che è l'unico nei circuiti virtuale a contenere l'indirizzo di destinazione completo come in datagram (i pacchetti successivi non dovranno conoscere la destinazione ma il percorso tracciato dal pacchetto esploratore). Quindi, in un sistema a circuito virtuale si crea un tubo di flusso tra le macchine sorgente e destinazione con tutti i router attraversati. Il vantaggio è che i pacchetti seguiranno sempre quella strada.

Si noti che attraverso i circuiti virtuali, esistono diversi benefici (*pro*), ovvero: (i) un circuito virtuale è più semplice e veloce e le info vengono associate tramite id ad una tabella; (ii) il controllo di congestione è semplificato poiché ogni pacchetto si assicura all'inizio delle risorse interne al router.
Purtroppo un grande svataggio dei circuiti virtuali si basa sul fatto che se un router ha un problema, allora l'intero percorso deve essere ricalcolato.

Indirizzi IPv4

Un indirizzo **IPv4** è un indirizzo a 32 byte, il che indica un valore al più pari a 4 miliardi. Gli indirizzi IPv4 furono distinti (per intervalli) in 5 classi: A, B, C, D ed E.
In generale, possiamo distinguere due parti di un indirizzo:

- La parte di **rete**, all'interno della quale si svolgono operazioni di routing;
- La parte di **host**, riferita alla *LAN*.
  Si noti che la differenza tra le varie classi di indirizzi si basa essenzialmente sul *numero di bit dedicati alla rete*, i quali saranno in numero via via maggiore da A verso E.
  Esistono alcuni **indirizzi speciali**, come per esempio `255.255.255.255` (formato da soli 1 in binario), il quale indica il **broadcast**.

Com'è formato un **pacchetto** IPv4? Un pacchetto con indirizzo IPv4 assume una determinata struttura. Idealmente, il pacchetto può essere suddiviso in **righe**:

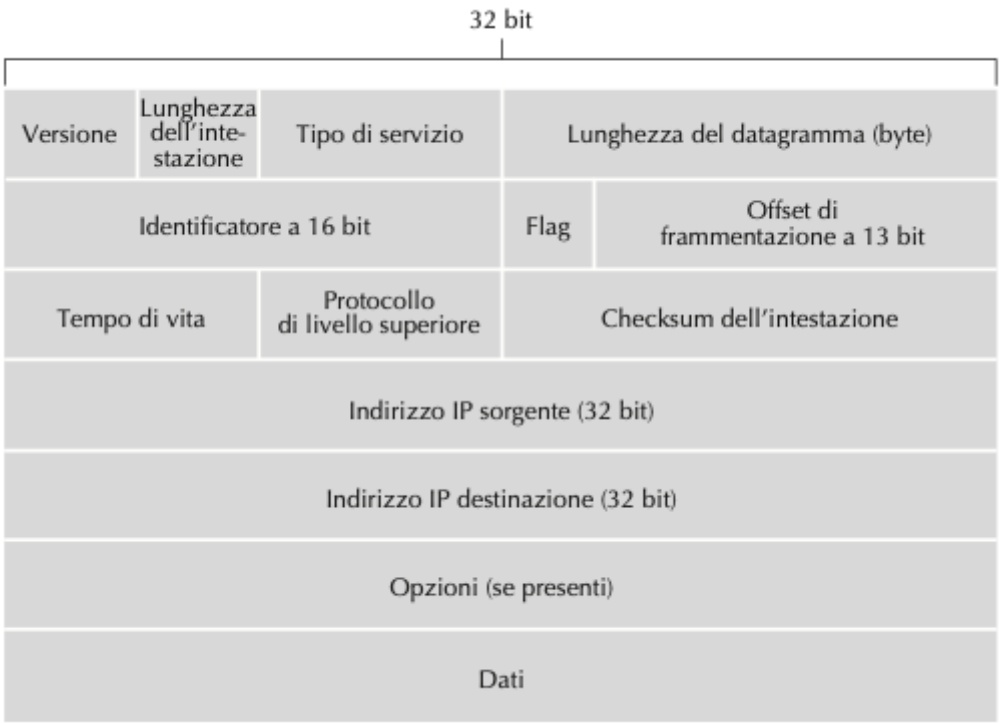
- *Riga 1*, dedicata all'**intestazione** contiene:
  - **Versione**, un campo di $4\,bit$, utili a specificare la versione del protocollo IP; necessario poiché differenti versioni di IP (i.e IPv4 ed IPv6 posseggono diversi formati di datagramma).
  - **Dimensione dell'intestazione**, un campo di $4\,bit$, utile a specificare dove finisce l'intestazione ed iniziano effettivamente i dati.
  - **Tipo di servizio**, un campo di $8\,bit$, utile a distinguere i diversi tipi di datagramma.
  - **Lunghezza del datagramma**, ovvero la lunghezza totale del datagramma, intestazione più dati. Pur essendo formato da un $16\,bit$, quindi in grado di rappresentare una dimensione massima uguale a $65\,535\,byte$ spesso, per non superare la dimensione massima per Ethernet, i datagrammi non superano i $1500\,byte$.

- *Riga 2*, dedicata principalmente alla **frammentazione** contiene:
  - **ID** del datagram, utile per la frammentazione);
  - **Flags** (i.e. DF, MF);
  - **Offset di frammentazione**, un campo di $13\,bit$ per indicare la posizione del frammento all'interno dell'intera sequenza.

- *Riga 3*, contiene:
  - **TTL**, ossia un contatore di salti che viene decrementato da ogni router e che se zero lo scarto di quest'ultimo (in altre parole, *TTL* indica il "*tempo di vita massimo di un pacchetto*");
  - **Protocollo di trasporto utilizzato**, un campo riservato per l'omonima funzione (e.g. TCP o UDP);
  - "**Header Checksum**", ossia un checksum del preambolo presente nella prima riga (utile per verificare la presenza di errori).

- *Riga 4*
    - **Indirizzo sorgente**, un campo a $32\,bit$;
- *Riga 5*
    - **Indirizzo di destinazione**, un campo a $32\,bit$;
- *Riga 6*
    - **Options**, un campo facoltativo (se vuoto HTL=0);
- *Riga 7*
    - **Data**, un campo contenente il payload, ovvero i dati, spesso il pacchetto al livello di trasporto (dati applicativi e intestazione del protocollo di trasporto).

> Le prime 5 righe misurano $20\,byte$, aggiungendo i parametri *TCP* (non considerando il campo option) la dimensione finale sarà $40\,byte$ (28 usando *UDP*)



Frammentazione

Se un pacchetto di dimensione N arriva ad un router e deve essere trasmesso su un link di uscita con MTU < N, allora il pacchetto dovrà essere *frammentato* e *ricostruito* nella macchina di destinazione.

> MTU è acronimo di *Maximum Transfer Unit* ovvero la dimensione massima della frame che può viaggiare a livello DLL.

I frammenti appartenenti allo stesso pacchetto presentano lo stesso **ID**; l'**offset** ci permette di capire dove si pone il frammento rispetto all'origine.
E' importante notare che i flag descritti all'interno della seconda riga coprono un ruolo relativo alla frammentazione. Infatti, relativamente a **DF** (i.e. *don't fragment*), questo indica se il pacchetto può essere frammentato o meno, mentre **MF** (i.e. *more fragment*) indica se il determinato frammento è l'ultimo della sequenza di frammenti.

Indirizzamento IPv4

Generalmente un host ha un solo collegamento con la rete; quando un host vuole inviare un datagramma, lo fa su qeust'ultimo collegamento. Il confine tra l'host ed il collegamento (fisico) è detto **interfaccia**.
Un router, differentemente da un host, poiché ha il compito di ricevere i datagrammi da una parte ed inoltrarli dall'altra, necessita di almento 2 interfacce.
Chi possiede un indirizzo IP univoco non è l'host o il router bensì l'interfaccia.

Un **indirizzo** IP di tipo IPv4 è un indirizzo lungo $4\,byte$ e viene solitamente rappresentato attraverso la **notazione decimale puntata**, in cui ogni $byte$ viene suddiviso dal simbolo $.$ e rappresentato nella sua forma decimale.

I seguenti **indirizzi** IPv4 sono stati riservati a specifiche funzioni:

- $0.0.0.0$, un indirizzo inizialmente non valido e da non utilizzare ma nel tempo diventato un indirizzo che rappresenta l'host stesso;
- $0000...xxxx$, per indicare un indirizzo all'interno della stessa sottorete. E' possibile, infatti, raggiungere una macchina presente all'interno della nostra stessa sottorete attraverso l'utilizzo di un indirizzo che prevede soli 0 a meno della parte relativa all'host. Si noti che inserire esclusivamente 255 all'interno della parte relativo all'host implica effettuare un broadcast all'interno della sottorete.
- $255.255.255.255$, per indicare il *broadcast*, secondo cui tutte le macchine riceveranno il pacchetto. Si noti che questo indirizzo è spesso bloccato dalla maggior parte dei router.
- $127.0.0.0$ -> $127.255.255.255$, per indicare un indirizzo di loopback; quest'ultimi sono utili ad identificare realmente l'host (i.e. *this-host*). Quasi sempre si trova nel formato $127.0.0.1$, il quale indica la propria macchina.
- $169.254.0.0$ -> $169.254.255.255$, una serie di indirizzi che la macchina host può utilizzare nel momento in cui nessuno riferisce ad egli una indirizzo IP. Tecnicamente, chiamiamo quest'ultimi indirizzi *zero-conf*. Quest'ultimo non è in grado di navigare in Internet.
- $10.0.0.0$ -> $10.255.255.255$, per indicare un indirizzo IP privato, utili nel momento in cui vogliamo creare una LAN all'interno della nostra abitazione. Indirizzi di questo tipo non sono raggiungibili dal mondo esterno.

- $172.16.0.0$ -> $172.31.255.255$, e $192.168.0.0$ -> $192.168.255.255$, per indicare ancora una volta un IP privato. Indirizzi di questo tipo sono spesso forniti dai router domestici.

> Attraverso un indirizzo di loopback, i pacchetti inviati a partire dal livello applicativo verso il livello di rete, una volta arrivati a quest'ultimo livello, vengono rispediti nuovamente verso il livello applicativo.

Maschera di sottorete

Ad un certo tempo si decise di eseguire una **maggiore suddivisione** all'interno dell'indirizzo IPv4, motivo per cui nascono le **maschere di sottorete**.
Una maschera permette di suddividere un indirizzo in **rete** più **sottorete** ed **host**, secondo cui rete più sottorete rappresentassero la parte di indirizzo all'interno della quale si effettuasse *routing*, mentre l'host indentifica la macchina stessa.

> Il compito di una maschera di sottorete è dividere in 3 parti l'indirizzo IPv4, facilitando la ricerca e l'individuazione della macchina nella rete.

Com'è formata la maschera? Sostanzialmente, la maschera non è altro che una sequenza di $x\ bit\ 1$ seguiti da una serie di $32 - x\ bit\ 0$. In particolare, x è un valore compreso tra 1 e 31, ed indica il numero di bit *su cui non applicare la maschera*.
Se per esempio avessimo un indirizzo con maschera 23, allora ciò indica che solo i primi $23\,bit$ dell'indirizzo saranno visibili, il che non vale per i restanti 9 (i quali non saranno visibili).
Una maschera di sottorete viene descritta secondo la seguente notazione: `a.b.c.d/x`.

Una maschera di sottorete viene applicata secondo un operazione di *AND* tra l'indirizzo è la maschera e permette di capire se la macchina si trova all'interno della nostra rete o fuori rispetto essa.

$$\begin{array}{rr} 11000000.10101000.00100000.01100001 & AND \\ 11111111.11111111.11111111.11100000 & = \\ 11000000.10101000.00100000.01100000 & \end{array} \qquad \begin{array}{rr} 192.168.032.097 & AND \\ 255.255.255.224 & = \\ 192.168.032.096 & \end{array}$$

> Secondo $RFC\ 3021$ è possibile utilizzare la maschera 31 per i soli collegamenti *punto-punto*, mentre una maschera 32, se pur in modo insensato, può essere utilizzata per un collegamento che prevede un solo indirizzo IP (i.e. non permette di comunicare "*con il resto del mondo*").

Assegnamento di indirizzi IP

L'assegnamento degli IP era originariamente compito di una sola autorità: **IANA** (acronimo di *Internet Assigned Numbers Authority*). Con il passare del tempo Internet crebbe sempre di più, motivo per cui nascono autorità che svolgono questo compito prima a livello **regionale** (a livello di continente), ossia **RIR**, poi anche a livello **locale**, ovvero **LIR**.

L'autorità locale responsabile dell'assegnamento di indirizzi sul territorio locale italiano è **GARR-LIR**. GARR è un organizzazione italiano responsabile della gestione della rete rispetto le strutture universitarie.

Indirizzi MAC

A livello LAN (rappresentabile da un ristretto gruppo di macchine) svolgono un ruolo di rilievo gli indirizzi MAC. Gli indirizzi MAC sono sostanzialmente degli indirizzi "*in teoria univoci ma modificabili*" assegnati alle schede di rete dal proprio produttore, il quale si riserva un blocco di indirizzi.
Indirizzi MAC sono formati da $6\,byte$ solitamente rappresentati in formato esadecimale separando ciascun ottetto con un trattino o con i due punti. I primi $3\,byte$ identificano il costruttore della scheda di rete.

$$00 - 08 - 74 - 4C - 7F - 1D$$

Un indirizzo MAC non fornisce nessuna informazione riguardo la posizione della macchina o la LAN di appartenenza, infatti, è scollegato dalla LAN ed è legato al solo costruttore della scheda di rete della macchina.

> All'interno di una comunicazione nel canale di una LAN non interessa l'IP ma il MAC, il quale informa su quale macchina è diretta la nostra frame (i.e. il pacchetto al livello DLL)

Il protocollo ARP

Il protocollo **ARP** (acronimo di *Address Resolution Protocol*) è un protocollo al livello **rete** il cui scopo è ottenere l'indirizzo MAC di una macchina di cui è noto l'indirizzo IP.
Un pacchetto IP, infatti, può arrivare a destinazione solo se è noto l'indirizzo MAC della macchina destinataria.

> Possiamo definire il protocollo ARP come un "*traduttore*" da IP a MAC.

Il mittente, che vuole conoscere l'indirizzo MAC del destinatario a partire dall'indirizzo IP di quest'ultimo, invia una `ARP REQUEST` in broadcast, utilizzando
$FF - FF - FF - FF - FF - FF$ come indirizzo MAC. In questo modo, tutte le schede di rete riceveranno il messaggio ma l'unico a rispondere sarà la macchina con l'indirizzo IP uguale a quello descritto nell'`ARP REQUEST`.
Ogni coppia `IP-MAC` verrà conservata all'interno di una tabella (i.e `ARP CACHE`), quest'ultima contenente le colonne relative a IP, MAC e TTL, quest'ultimo un timer utile a scartare informazioni troppo vecche.
Questa tabella fungerà da cache per gli indirizzi MAC, e per questo sarà caratterizzata dalla poca memoria e la necessità di processi di rimozione e aggiunta (e.g. strategia FIFO).
Per ovvi motivi, prima di ogni `ARP REQUEST`, ogni host controlla che questa non sia già presenta all'interno della propria `ARP CACHE`.

Come funziona tecnicamente il protocollo ARP? Essenzialmente il protocollo presenta 3 fasi:

1. `ARP REQUEST`, una richiesta in cui viene inserito all'interno della *FROM* l'indirizzo IP di cui si vuole conoscere l'indirizzo MAC ed il suo indirizzo MAC proprietario; nella sezione *TO* l'indirizzo MAC $FF-FF-FF-FF-FF-FF$. In questo verrà effettuata una broadcast, motivo per cui tutti analizzeranno la richiesta ed una sola macchina sarà colei che non scarterà la frame (poichè "*proprietaria*" dell'indirizzo IP nella FROM).
2. `ARP REPLY`, secondo cui la macchine che non scarta la frame può rispondere, comunicando il suo indirizzo MAC. Egli inserisce il proprio IP ed indirizzo MAC all'interno della *FROM*, mentre inserisce i dati del precedente mittente all'interno della sezione *TO*.
3. In seguito alla risposta, l'informazione precedentemente non conosciuta verrà conservata all'interno di `ARP CACHE`.

$ARPRequest:\ From:\ 192.168.0.1\,(00:00:1A:3E:43:55)\ To:\ 192.168.0.3\,(FF:FF:FF:FF:FF:F$

$ARPReply:\ From:\ 192.168.0.3\,(D1:01:CA:13:37:B7)\ To:\ 192.168.0.1\,(00:00:1A:3E:43:55)$

I **problemi** di ARP sono rappresentati dalle sue gravi falle di sicurezza, infatti:

- ARP **non richede autenticazione**, motivo per cui l'host che invia una richiesta deve "fidarsi" che l'indirizzo MAC arrivato sia effettivamente del leggittimo proprietario, il che crea le premesse per numerose vulnerabilità;
- ARP è **stateless**, il che indica che un `ARP REPLY` possa non essere conseguenza di alcuna `ARP REQUEST`, ancora una volta premessa di notevoli attacchi;
- Un host che riceve un pacchetto ARP, deve sempre aggiornare la sua `ARP CACHE`.

Altri protocolli

RARP

Il protocollo **RARP** (acronimo di *Reverse Address Resolution Protocol*), che differentemente da ARP non è un protocollo distribuito, è un protocollo usato per risalire all'indirizzo IP conoscendo l'indirizzo fisico (MAC).
Il suo funzionamento si basa su un server RARP che associa un indirizzo MAC ad un indirizzo IP. L'utilizzo di questo server è anche il motivo del suo scarso utilizzo.

BOOTP

Il protocollo **BOOTP** è un protocollo che permette l'installazione di un sistema operativo su una macchina tramite rete; il tutto a partire dalla conoscenza dell'indirizzo MAC di quest'ultima macchina (per questo motivo, BOOTP è spesso utilizzato insieme a RARP).
Il trasferimento avviene tramite **TFTP** (una versione *senza autentificazione* di *FTP*).
L'utilizzo di BOOTP era fortemente legato al costo degli HDD.

Come avviene la comunicazione?

Supponiamo che una macchina A abbia la necessità di comunicare con un'altra macchina B, allora esistono due possibili scenari:

- B si trova all'interno della stessa LAN di A, per cui in tal caso dovrà ottenere il MAC del destinatario per avviare la comunicazione a livello DLL, motivo per cui invia una richiesta ARP, rimanendo in attesa di risposta da parte del destinatario.
- B non si trova all'interno della stessa LAN di A, motivo per cui la macchina A indirizza il pacchetto al *router* che, leggendo il pacchetto IP crea una frame DLL da spedire verso la LAN di destinazione.

Com'è possibile per il mittente capire se il ricevente si trova all'interno della sua stessa LAN? Sostanzialmente, ciò si basa sull'applicazione di operazioni di AND e XNOR tra *indirizzi* e *maschere*. In particolare, viene effettuato:

1. L'operazione di **AND** tra l'IP del destinatario e la maschera del destinatario;
2. L'operazione di **AND** tra l'IP del destinatario e la maschera del mittente;
    - Se il risultato dell'operazione 1 e 2 è uguale, allora le macchine appartengono alla stessa sottorete (i.e. LAN), motivo per cui verrà effettuata una richiesta ARP.
3. L'operazione di **XNOR** (operazione per cui risultato è uguale ad 1 se i bit sono uguali, quindi entrambi 0 o 1). A questo punto, se il risultato presenta una sequenza di soli 1, allora la destinazione di trova definitivamente nella sottorete del mittente.

> Se il mittente dovesse scoprire che il destinatario si trova in un'altra LAN, dovrà comunque fare una richiesta ARP. In particolare, effettua una richiesta ARP inserendo come MAC di destinazione il router di uscita (quindi non sarà fatto in broadcast) e come indirizzo IP l'indirizzo del destinatario. A quel punto sarà il router di destinazione a effettuare nuovamente la richiesta ARP.

Tabelle di routing

Ogni macchina, quindi non solo il router, possiede un *particolare database* chiamato **tabella di routing**, contenente *informazioni utili* per muoversi all'interno della rete. Generalmente, una tabella di routing è formata da:

- Rete di destinazione con maschera;
- IP del router da contattare per raggiungere la destinazione;
- Una **metrica**, per capire se un uscita è più conveniente di un'altra.
- Talvolta, vi è anche un indirizzo di *loopback* ed *interfaccia*;

Un algoritmo di routing è responsabile di riempire una tabella di routing.

DHCP

DHCP (acronimo di *Dynamic Host Configuration Protocol*) è un protocollo per l'*assegnazione dinamica* di un indirizzo IP per l'host che si è appena collegato ad una *LAN plug-and-play*, il che indica che non richiede configurazione.

Ogni indirizzo IP all'interno di una LAN deve essere compatibile con la LAN stessa, ma cosa succede nel caso in cui utilizziamo un dispositivo mobile, in cui è possibile spostarsi da una LAN all'altra? Esistono 2 **soluzioni**: (i) configurare manualmente il dispositivo (i.e. macchina) o (ii) utilizzare DHCP.

Come funziona DHCP? Il funzionamento di DHCP richiede l'utilizzo di un **DHCP Server** e si compone di 4 fasi:

1. **Fase di scoperta** (alias *DHCP Discovery*), all'interno della quale la nuova macchina invia un messaggio in broadcast a cui deve rispondere i soli server DHCP. L'obiettivo all'interno di questa fase è trovare almeno un server DHCP disponibile ad offrire una configurazione. All'interno del messaggio inviato dalla nuova macchina saranno presenti i seguenti campi:
    - Un campo per l'**Indirizzo Sorgente**, inizialmente settato a $0.0.0.0 : 68$ (con 68 uguale alla porta per il DHCP Client).
    - Un campo per l'**Indirizzo Destinazione**, che poichè broadcast sarà composta da $255.255.255.255 : 67$ (con 67 uguale alla porta per il DHCP Server).
    - Un campo dedicato alla "**fase DHCP**" (in questo caso, contenente *DHCPDISCOVER*);
    - Un campo **yiaddr**, settato a $0.0.0.0$ che successivamente conterrà l'indirizzo offerto dal server DHCP.
    - Un campo per l'**ID** della transazione, utile nel caso in cui più macchine diverse richiedano un indirizzo IP.

2. **Fase di offerta** (alias *DHCP Offer*), all'interno della quale il server DHCP risponde con un offerta DHCP contenente la nuova configurazione. All'interno del messaggio inviato dal server DHCP (inviato in broadcast), saranno presenti i seguenti campi:
    - Un campo per l'**Indirizzo Sorgente**,
    - Un campo per l'**Indirizzo Destinazione**, ancora broadcast.
    - Un campo dedicato alla "**fase DHCP**" (in questo caso, contenente *DHCPOFFER*);
    - Un campo **yiaddr**, che adesso contiene l'indirizzo offerto dal server DHCP.
    - Un campo **lifetime**, che indica per quando tempo il server DHCP mette a disposizione il determinato indirizzo IP.
    - Un campo per l'**ID** della transazione, utile nel caso in cui più macchine diverse richiedano un indirizzo IP.
    - Un campo per l'**ID** del server DHCP.

3. **Fase di richiesta** (alias *DHCP Request*), all'interno della quale la macchina richiedente l'indirizzo può accettare o meno l'offerta del server DHCP. All'interno del messaggio inviato dalla nuova macchina saranno presenti i seguenti campi:
    - Un campo per l'**Indirizzo Sorgente**, inizialmente settato a $0.0.0.0 : 68$ (con 68 uguale alla porta per il DHCP Client).
    - Un campo per l'**Indirizzo Destinazione**, che poichè *broadcast* sarà composta da $255.255.255.255 : 67$ (con 67 uguale alla porta per il DHCP Server).
    - Un campo dedicato alla "**fase DHCP**" (in questo caso, contenente *DHCPREQUEST*);
    - Un campo **yiaddr**, che contiene ancora l'indirizzo offerto dal server DHCP.
    - Un campo per l'**ID** della transazione, adesso incrementato.
    - Un campo per l'**ID** del server DHCP.

4. **Fase di accettazione** (alias *DHCP ACK*), secondo cui la macchine è ufficialmente autorizzata ad usare l'indirizzo IP. All'interno del messaggio inviato dalla nuova macchina saranno presenti i campi visti in precedenza; l'unico valore che cambia è il campo relativo alla fase DHCP, adesso uguale a *DHCPACK*.

Quali sono i **problemi** di DHCP?
Il suo problema principale è dato dal timer di **lifetime**. Infatti, per un indirizzo per cui è scaduto il lifetime non esiste alcun *rilasco esplicito*, motivo per cui il client può rilasciare l'indirizzo ed effettuare una nuova richiesta DHCP oppure non far nulla.
Il server DHCP, in ogni caso, allo scadere del *lifetime* può ri-utilizzare e ri-assegnare l'IP ad un altra macchina, il che potrebbe far sì che vi siano più macchine con lo stesso indirizzo IP.

> In IPv6 è previsto il protocollo DHCP con leggere modifiche.

NAT

A questo punto, è chiaro che ogni dispositivo che desideri comunicare in rete necessiti di un indirizzo IP. Tuttavia, la proliferazione delle sottoreti dovrebbe implicare una allocazione di indirizzi (da parte dell'ISP) per ognuna delle sottoreti, ma ciò non è possibile a causa del numero limitato di indirizzi. Per risolvere questo problema, nasce il **protocollo NAT**.

Un router abilitato al NAT non appare come router al mondo esterno, bensì come un unico dispositivo con un unico indirizzo IP. In questo modo è possibile per una sottorete utilizzare tranquillamente il proprio *spazio di indirizzamento* (i.e. i propri indirizzi privati), il quale ha senso solo per i dispositivi all'interno della sottorete; inoltre, per ovvi motivi, è permesso che più sottoreti abbiano lo stesso spazio di indirizzamento privato.

> Un router ottiene il suo indirizzo IP univoco attraverso l'utilizzo del server DHCP dell'ISP. In seguito è il router stesso a mandare in esecuzione un server DHCP che posso fornire l'indirizzo privato ad ogni macchina all'interno della sottorete.

Come **funziona** il protocollo NAT? Essenzialmente, ogni qual volta una macchina all'interno della sottorete, con il suo indirizzo privato ed un suo numero di porta, vuole comunicare in rete, egli invia un datagramma verso il router, il quale effettua la richiesta utilizzando il suo indirizzo IP (univoco) ed un numero di porta scelto dal router stesso.
Tutte queste informazioni vengono salvate all'interno di una **tabella di traduzione NAT**, all'interno del router NAT.

Supponiamo che una macchina A, con indirizzo $10.0.0.1$ e numero di porta 3345, all'interno di una rete domestica voglia richiedere una pagina Web, quest'ultimo con indirizzo $128.119.40.186$ e numero di porta 80. Dunque, la macchina A, con il suo indirizzo e numero di porta, invia un datagramma verso il router NAT, che genera un nuovo numero di porta 5001.
Il router NAT, quindi, effettua la richiesta secondo il suo indirizzo IP, $138.76.29.7$, ed il numero di porta generato 5001.

Index	Prot.	Local IP	Local Port	Global IP	Global Port
1	TCP	$10.0.0.1$	3345	$138.76.29.7$	5001

All'invio della **risposta** da parte del Web Server, il router NAT consulta la tabella di traduzione NAT usando l'indirizzo IP di destinazione ed il numero di porta di destinazione per ottenere l'indirizzo IP privato della macchina richiedente.

> Notiamo che,

Quali sono i **problemi** di NAT? I problemi per NAT sono i seguenti:

- Tutte le comunicazioni relative alle macchina private escono in Internet con lo stesso indirizzo IP, come se tutte le operazioni fossero state svolte da una singola macchina;
- Essendo il campo per il numero di porta lungo $16\ bit$, il protocollo NAT può supportare al più $65\,000$ connessioni simultanee con un solo indirizzo IP. Una LAN abbastanza grande potrebbe saturare il numero di connessioni possibili, le quali potrebbero essere aumentate fornendo più indirizzi IP per lo stesso router.
- Generata una entry con una determinata porta, quest'ultima non può essere utilizzata per comunicare finchè non viene rimossa dalla tabella.
- Se la comunicazione inizia dall'esterno (e non dall'interno), il router non può capire quale sia la macchina a cui inoltrare il tutto. Per questo motivo, possiamo affermare che NAT funziona finché la comunicazione inizia dalla LAN.

UPnP

Una soluzione all'ultimo problema di NAT è **UPnP**, un protocollo che permette alla macchina che può essere ricevere comunicazioni provenienti dall'esterno di inviare un messaggio al server NAT e richiedere il set di una riga all'interno della tabella.
In questo modo, tutte le comunicazione che iniziano dall'esterno verso il router (con una data porta) verranno inoltrate di default dal router verso la determinata macchina.
Per ovvi motivi (richiamando anche il terzo problema di NAT), nel momento in cui la macchina richiede il set di una entry con una determinata porta all'interno della tabella,

quest'ultima porta non potrà più essere utilizzata da nessun altra macchina; fare ciò restituisce un errore.

IPv6

IPv6 nasce a causa dei **limiti** mostrati da **IPv4**, ovvero:

- L'esaurimento degli indirizzi da parte di IPv4;
- Un routing che non riflette la posizione geografica;
- L'implementazione di nuovi servizi che mostrarono dei limiti all'interno di IPv4;

Quali sono i **miglioramenti** introdotti da IPv6?
IPv6 introduce molteplici cambiamenti rispetto IPv4, ovvero:

- **Indirizzamento esteso**, aumentando la **dimensione** dell'indirizzo IP da 32 a $128\,bit$; in questo modo, gli indirizzi diventano praticamente inesauribili. IPv6 supporta 3 tipi di indirizzi: *Unicast*, *Multicast* ed *Anycast*; quest'ulimi consentono la consegna di un datagramma ad un qualsiasi host all'interno di un gruppo.
- **Intestazione ottimizzata** di $40\,byte$, possibile attraverso la rimozione o la resa opzionale di alcuni campi presenti all'interno di IPv4. L'intestazione di lunghezza uguale a $40\,byte$ è fissa e ciò consente una più rapida elaborazione.
- **Etichettatura dei flussi**, secondo cui viene presentato il concetto di **flusso** ed è possibile un'etichettatura di pacchetti appartenenti a flussi particolari (e.g. audio, video).

Com'è formato un **pacchetto** IPv6? Un pacchetto IPv6 può essere suddiviso in 5 righe:

- *Riga 1*.
  - **Versione**, un campo a $4\,bit$ che identifica il numero di versione di IP, il quale, come si può facimente immaginare, per IPv6 è uguale a 6.
  - **Classe di traffico**, un campo ad $8\,bit$ simile a *TOS* in IPv4, utile ad attribuire priorità ad un datagramma.
  - **Flow Label** (alias *etichetta di flusso*), un campo a $20\,bit$ utilizzato per identificare un flusso.

- *Riga 2*.
  - **Dimensione del payload**, il quale ha sostituito il campo *Total Length*, un campo a $16\,bit$ che indica il numero di $byte$ presenti dopo l'intestazione fissa di $40\,byte$.
  - **Next Header** (alias *intestazione successiva*), il quale ha sostituito il campo *Protocol* ed indica il protocollo a cui verranno consegnati i dati (e.g. TCP o UDP).
  - **Hop Limit**, il quale ha sostituito TTL e rappresenta un valore che viene decrementato di 1 ogni qual volta che il datagramma viene inoltrato da un router; se il valore raggiunge 0, il datagramma viene eliminato.

- *Riga 3*.
    - Indirizzo IP **Sorgente**;
- *Riga 4*.
    - Indirizzo IP **Destinazione**;
- *Riga 5*.
    - *Payload* (i.e. dati).

Quali campi sono stati **rimossi**? I campi che sono stati rimossi sono:

- Campi relativi **frammentazione**, ovvero **ID**, **Flags** ed **Offset**, la quale essendo un operazione onerosa, in IPv6 non viene consentita sui router intermedi ma esclusivamente dalla sorgente o destinazione.
- **Opzioni**, ovvero **ToS** ed **Header Length**, cui rimozione ha permesso un intestazione di lunghezza fissa a $40\,byte$.
- **Header Checksum**, poiché dal momento che protocolli a livello di trasporto e collegamento calcolano il loro checksum, si pensò fosse ridondante ed oneroso calcolare un checksum anche a livello di rete.

Quali campi sono stati **modificati**? I campi che sono stati modificati sono:

- **Total Length**, il quale diventa **Payload Length**;
- **Protocol**, il quale viene adesso gestito diversamento e diventa **Next Header**;
- **TTL**, il quale diventa **Hop Limit**;

Quali campi sono stati **aggiunti**? I campi che sono stati aggiunti sono:

- **Traffic Class**;
- **Flow Label**, aggiunto in IPv6 con l'idea di permettere un utilizzo di *circuito virtuale*; indica al router di non analizzare l'indirizzo di destinazione bensì la propria tabella per indirizzare il traffico. In questo modo, pacchetti relativi allo stesso flusso possono seguire la stessa strada in ordine. Purtroppo, Flow Label non viene molto utilizzato;

Com'è formato un **indirizzo** IPv6? Gli indirizzi IPv6 sono composti di $128\,bit$ e sono rappresentati come 8 gruppi, separati da due punti (`:`), di 4 cifre esadecimali (ovvero $16\,bit$) in cui le lettere vengono scritte in forma minuscola. Ad esempio `2001:0db8:85a3:0000:1319:8a2e:0370:7344` rappresenta un indirizzo IPv6 valido. E' possibile utilizzare notazione come `::`, i quali indicano una sequenza contigua di 0.

Come viene **suddiviso** un indirizzo IPv6? Un indirizzo IPv6 viene suddiviso nel seguente modo:

- 001, primi tre bit dell'indirizzo, indicano il tipo di indirizzo;

- $Global\ Routing\ Prefix$, ovvero i successivi $45\,bit$, indicano il *sito* (i.e. regione) a cui l'indirizzo si riferisce;
- $Subnet\ ID$, ossia i successivi $16\,bit$, indicano la sottorete e sono utili a raggiungere velocemente la macchina di destinazione;
- $Interface\ ID$, ossia i restanti $64\,bit$;

Com'è possibile **suddividere** gli indirizzi IPv6? Gli indirizzi IPv6 si suddividono in tre tipi di base:

- 010, indirizzi **unicast**, che possiamo a loro volta suddividere in;
    - $1111\,1110\,10$ :: ($FE80$ ::), indirizzi locali di **canale** (*link-local*), indica che l'indirizzo è valido esclusivamente sullo specifico link fisico e sono progettati per l'uso su un collegamento locale singolo (rete locale). Può indicare un'azienda o un ente. I router non inoltrano i pacchetti con un indirizzo di origine o di destinazione contenente un indirizzo locale di collegamento (non possono essere utilizzati in Internet).
    - $1111\,1110\,11$ :: ($FEC0$ ::), indirizzi locali di **sito**, specifica che l'indirizzo è valido esclusivamente all'interno dell'organizzazione locale. Indirizzi *site-local* sono stati deprecati secondo $RFC\ 3879$. Non possono essere utilizzati in Internet.
    - :: $1/128$, - l'indirizzo di **loopback** è un indirizzo associato al dispositivo di rete che ripete come eco tutti i pacchetti che gli sono indirizzati. Corrisponde a $127.0.0.1$ in IPv4;
    - ::, l'indirizzo composto da tutti zeri, detto **unspecified address**, viene utilizzato per indicare *l'assenza di indirizzo* e viene utilizzato esclusivamente a livello software, corrisponde a $0.0.0.0$ in IPv4;

- $1111\,1111$ :: ($FF00$ ::), indirizzi **multicast**, specifica una serie di interfacce, possibilmente in più ubicazioni. Il prefisso utilizzato per un indirizzo multicast è ff. Se un pacchetto viene inviato ad un indirizzo multicast, una copia di tale pacchetto viene distribuita ad ogni membro del gruppo. Il sistema operativo IBM i attualmente fornisce il supporto di base per l'indirizzamento multicast.;
- $1111\,1111$ :: ($FF00$ ::), indirizzi **anycast**, specifica una serie di interfacce, possibilmente in diverse ubicazioni, che condividono tutte un singolo indirizzo. Un pacchetto inviato ad un indirizzo anycast è recapitato solo al membro più vicino del gruppo anycast. IBM® i può inviare indirizzi anycast, ma non può essere un membro di un gruppo unicast..

Altri indirizzi speciali per IPv6 sono:

- :: $151.97.1.1$, un esempio di indirizzo **IPv4** mappato all'interno della gerarchia IPv6;
- 010 ::, indirizzi riservati ai **service providers**;
- 100 ::, indirizzi **geografici**, utilizzati per *routing*;

Quali sono gli **header opzionali** all'interno di un pacchetto IPv6? 0. Tutte le **informazioni aggiuntive**, non previste nell'header di base, possono essere inserite utilizzando i

cosiddetti **extension header**. Essi sono dei campi opzionali che pertanto sono presenti nel pacchetto solo quando necessario, a differenza di IPv4 che prevedeva eventuali opzioni aggiuntive all'interno di *Option* (il che rendeva l'header per quest'ultimo variabile). In particolare, le informazioni contenute nelle estensioni devono essere esaminate soltanto dalla sorgente e dalla destinazione, ma non dai nodi intermedi. Questo rende più veloce l'instradamento. L'unica eccezione a questa regola è costituita dall'opzione Hop-by-Hop, che contiene informazioni che devono essere esaminate da tutti i nodi lungo il cammino. Ogni extension header include al proprio interno un campo **Next Header** che identifica ciò che segue immediatamente dopo.
Inoltre, ogni header opzionale è caratterizzato da un valore univoco (ad esempio TCP è 6); generalmente è preferibile seguire un ordine preciso con cui elencare i vari header opzionali. Idealmente si forma una *lista-linkata* in cui ogni nodo contiene le informazioni e un "link" (*next-header*) che "annuncia" il tipo successivo.

Come viene gestita la **frammentazione** all'interno di IPv6? Sappiamo che in IPv4, la frammentazione avviene all'occorrenza: ogni router intermedio deve frammentare i datagram che risultano troppo grandi per attraversare il link successivo; il mittente non viene a conoscenza di tali frammentazioni intermedie. D'altra parte, con IPv6 la frammentazione viene effettuata in modalità end-to-end: la sorgente si occupa di dimensionare opportunamente i datagram e la destinazione si occupa di riassemblarli, senza alcun coinvolgimento dei router intermedi. IPv6 richiede che ogni link consenta il passaggio di pacchetti pari almeno a $1280\,byte$.

Cosa possiamo dire rispetto la **sicurezza** all'interno di IPv6? In IPv6 esistono 2 specifici extension header che forniscono servizi di sicurezza: "IP Authentication Header (AH)" e "IP Encapsulation Security Payload" (ESP), parti della suite di protocolli IPSec.

- **IP Authentication Header** (AH) ha il compito di fornire ai datagrammi IP integrità e autenticazione, ma non confidenzialità. AH non fornisce confidenzialità affinché possa essere usato anche in luoghi dove l'importazione, l'esportazione e l'uso della crittografia è limitato. AH supporta sicurezza tra due o più host, tra due o più gateway, tra host e gateway.
- **IP Encapsulation Security Payload** (ESP) fornisce integrità, autenticazione e confidenzialità ai datagrammi IP. ESP supporta sicurezza tra due o più host, tra due o più gateway, tra host e gateway.

Sia AH che ESP possono essere utilizzati in due modalità: Transport Mode e Tunnel Mode; nel primo caso viene fornita protezione solo ai dati, mentre nel secondo caso viene fornita protezione all'intero datagramma IP.

Com'è stata gestita la **transizione da IPv4 ad IPv6**? Se i nuovi sistemi IPv6 sono retro-compatibili e supportano IPv4, lo stesso non vale per quest'ultimo. Non è possibile aggiornare tutte le macchine con IPv4 ad IPv6, motivo per cui è necessario che anche quest'ultime dispongano di un modo per gestire pacchetti IPv6. Soluzione è stata fornita dal **tunneling**. Supponiamo che una macchina A IPv6 voglia comunicare con una macchine B IPv6 e per farlo debba passare per alcuni router IPv6 ed n router IPv4, che chiameremo *tunnel*. Essenzialmente, secondo il tunneling, l'ultimo router IPv6 all'interno del tragitto che deve essere percorso dal datagramma IPv6 incapsula quest'ultimo (i.e. il datagramma IPv6) all'interno del campo dati di un nuovo datagramma IPv4. In seguito, il primo nodo IPv6 che decreta la fine del tunnel determinerà che il datagramma IPv4 che ha ricevuto ne contiene a

sua volta uno di tipo IPv6, il tutto a partire dal numero di protocollo definito all'interno dell'omonimo campo, ossia 41. Dunque, il nodo IPv6 estrae il pacchetto IPv6 e continua l'instradamento.

DHCP in IPv6

È simile a IPv4 ma c'è qualche differenza:

- Nella fase di *DHCP Discovery*, il messaggio è in **multicast** (il broadcast tendeva ad intasare la rete) e l'indirizzo sorgente non è più 0.0.0.0 ma è un indirizzo local-link valido (in questo modo il mittente è identificato);
- Nella fase di *DHCP Offer*, avviene in **unicast**;
- Nella fase di *DHCP Request*, avviene in **multicast**;
- Nella fase di *DHCP Ack*, avviene in **multicast**;
- I dispositivi richiedenti sono identificati dagli indirizzi **EUI-64**.

NDP (Neighbour Discovery Protocol)

NDP è un protocollo introdotto con IPv6 per scoprire chi sono i *vicini* o per scoprire se ci sono router all'interno della rete locale, quindi capire come fare al meglio il routing. Inoltre, dentro NDP abbiamo inserito nativamente anche i protocollo **ARP**, ICMP e ICMP Redirect Message.

> Il protocollo ICMP (ICMPv4 ed ICMPv6) è utile ad inviare messaggi di controllo a livello di rete. Non utilizza porte e viene identificato da un codice inserito all'interno del pacchetto. Un esempio di messaggio ICMP è `ping`.

> Un datagramma IPv6 all'interno di una frame **Ethernet** prevede, all'interno della frame, che il campo Ethertype venga settato al valore: $86DD$. In questo modo, è possibile capire velocemente se la frame trasporta un datagramma di tipo IPv6 o IPv4; quest'ultime vengono riconosciute da un valore uguale a 0800.

Gli algoritmi di routing

Il routing è il compito fondamentale del livello di rete. Consiste nella ricerca del percorso verso una destinazione attraverso la rete.
Gli algoritmi di routing sono eseguiti dai router per decidere su quale linea di output trasmettere i dati in ingresso e hanno come obbiettivo finale l'aggiornamento delle tabelle di routing.

Gli algoritmi di routing non sono altro che algoritmi su grafi che si possono suddividere in due categorie:

- **Statici**: il grafo (i.e. la rete) viene esaminato in un determinato istante e i percorsi vengono *calcolati* sul grafo in quel preciso momento, senza tenere conto di variazioni dei pesi all'interno della rete. L'informazione viene poi distribuita a tutti i nodi partecipanti;
- **Dinamici**: la rete viene esaminata continuamente per poter prendere le decisioni migliori; sfortunatamente, questo approccio è particolarmente oneroso in termini di risorse e tempo e potrebbe causare congestione.

Generalmente, se usati correttamente, gli algoritmi dinamici performano meglio durante il routing. Inoltre, è possibile fare un'ulteriore divisione:

- **Locali**: i nodi agiscono conoscendo solo le informazioni di nodi "vicini". (informazioni che potrebbero non essere congrue a quelle di altri punti nella rete);
- **Globali**: si agisce avendo una visione "globale sulla rete"; il problema è che potrebbero servire troppe informazioni;

Flooding

L'algoritmo di **flooding** rappresenta, in linea teorica, uno dei migliori algoritmi di routing e riesce a trovare la strada migliore in poco tempo senza conoscere un grosso numero di informazioni.

Come **funziona** flooding? All'inizio, il nodo mittente invia a tutti i nodi a lui connessi una copia del pacchetto; successivamente i nodi che hanno ricevuto la copia lo inoltrano ai nodi collegati a loro (come ha fatto il mittente), *escludendo* il nodo da cui ha ricevuto il pacchetto stesso. Conseguentemente, in pochi passi, è possibile trovare il percorso migliore (poiché prova tutti i possibili percorsi). In particolare, il numero di passi risulta essere il *diametro* del grafo.

Qual è il **problema** di flooding? Il problema di flooding si basa sul fatto che la sua applicazione può far sì che due nodi possano inviarsi reciprocamente una copia, il che avviene poichè i due nodi non sono a conoscenza che l'altro ha già ricevuto una copia del pacchetto.
Per questo motivo, il numero di copie del pacchetto all'interno della rete tende a crescere fino ad inondare l'intera rete, con la possibilità che possano pure tornare al nodo di origine.

Com'è possibile **risolvere il problema** del flooding? E' necessario bloccare il routing da parte di flooding prima che quest'ultimo inondi l'intera rete, per farlo possiamo applicare una delle due soluzioni:

- Conoscendo a priori il diametro del grafo, settare un **Hop Limit** pari al diametro del grafo. Purtroppo, non è sempre semplice conoscere il diametro del grafo.
- Inserire un **ID** per ogni pacchetto, attraverso cui è possibile far capire ai nodi che i pacchetti che gli stanno arrivando sono tutte copie; questa soluzione potrebbe

causare congestione ed è molta onerosa poiché ogni nodo dovrebbe tenere in memoria tutti i pacchetti che ha visto ed effettuare i vari confronti.

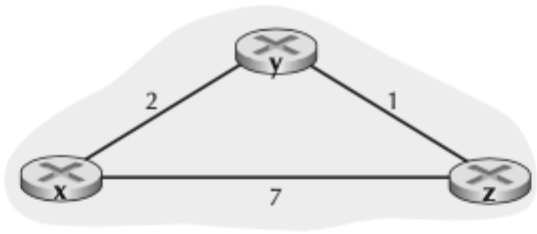
Distance Vector

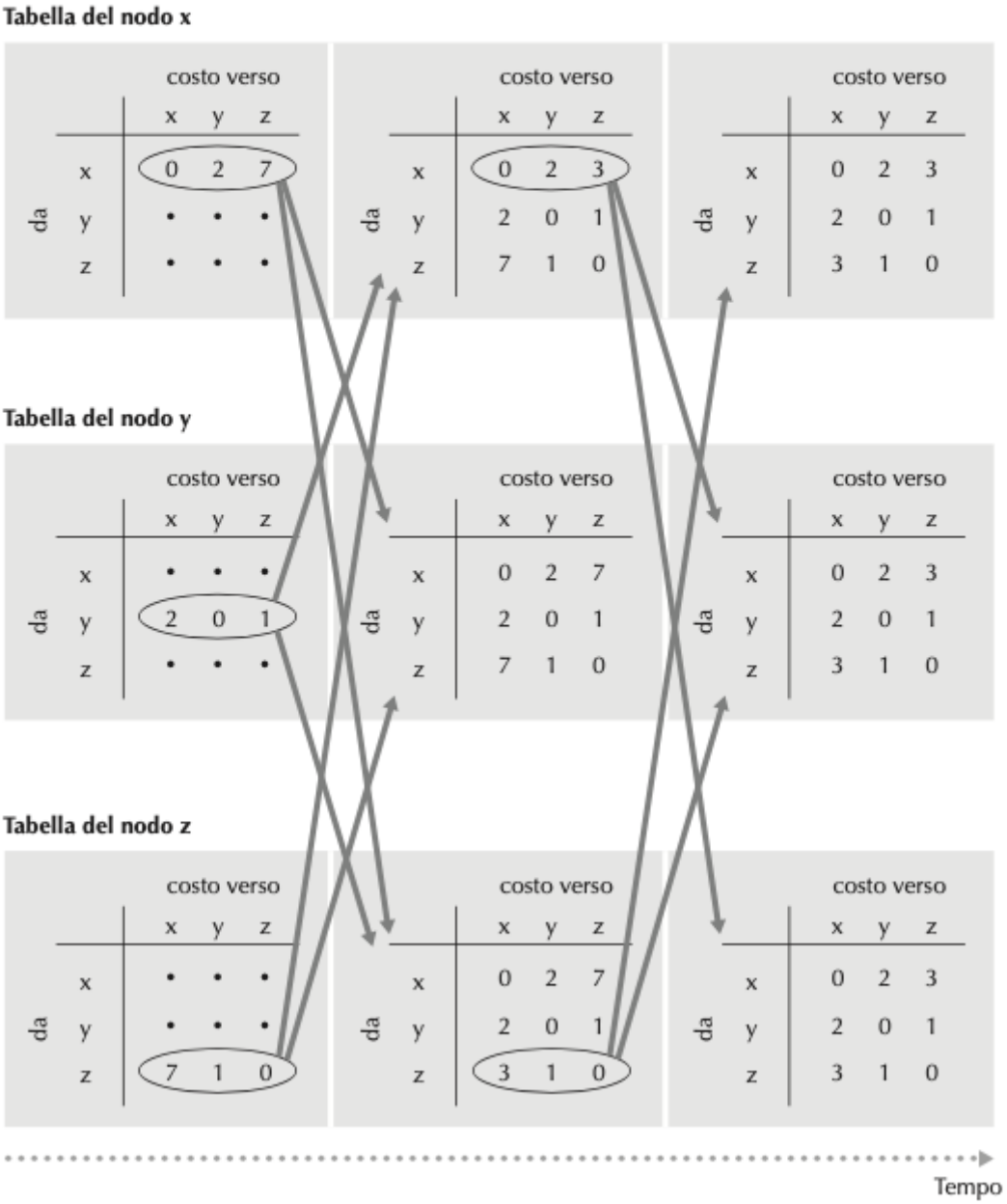
Distance Vector è un algoritmo di routing basato sull'algoritmo di **Bellman-Ford**. È una soluzione **statica** e **locale**, cui obiettivo è trovare il *cammino minimo* tra due nodi.

> Distance Vector è **distribuito**, poiché ciascun nodo riceve parte dell'informazione da uno o più dei suoi vicini; (ii) **auto-terminante**, poiché non vi è alcun segnale che il calcolo debba fermarsi, semplicemente si blocca; (iii) **asincrono**, poiché non richiede che tutti i nodi operino al passo con gli altri.

Distance Vector, come abbiamo già scritto, si basa sull'algoritmo di Bellman-Ford, quindi calcola il percorso minimo da x verso ogni y nel grafo secondo la nota formula di Bellman-Ford: $d_x(y) = min_v\{c(x,v) + d_v(y)\}$. Tale relazione è piuttosto intuitiva. Infatti se, dopo aver viaggiato da x a v, consideriamo il percorso a costo minimo da v a y, il costo del percorso sarà $c(x,v) + d_v(y)$. Dato che dobbiamo iniziare viaggiando verso qualche vicino v, il costo minimo da x a y è il minimo di $c(x,v) + d_v(y)$ calcolato su tutti i nodi adiacenti v.

Come **funziona** Distance-Vectors? Ogni router $\mathcal{X}$ conserva una tabella di routing $D^{\mathcal{X}}()$ chiamata "*tabella delle distanze*" (i.e. una *matrice rettangolare*, a volte *quadrata*) formata da tante *righe* quanti sono i nodi presenti nella rete e tante *colonne* quante sono le possibili uscite. La distanza può essere misurata con vari tipi di **metriche**, come ad esempio il numero di salti (hop), il ritardo in millisecondi o il numero di pacchetti in coda lungo un cammino.
Periodicamente i router scambiano la propria tabella delle distanze con i router vicini. Quando un router x riceve da un vicino v la sua tabella delle distanze, confronta per ogni destinazione y la propria distanza con quella stimata dal vicino, a cui viene sommata la distanza rispetto al vicino stesso $d(x,v)$; se il secondo valore è minore, modifica la tabella di routing inserendo tale valore come distanza e la linea che porta al vicino come linea di uscita. A questo punto, se al confronto segue un aggiornamento del vettore delle distanze del nodo x, allora x invia il proprio vettore delle distanze ai suoi vicini.





Com'è possibile **costruire la tabella**? Ogni tabella di instradamento posseduta da un nodo x (i.e. $\mathcal{D}_x$) è formata dal vettore delle distanze per il nodo x ed i vettori delle distanze di tutti i suoi vicini v, quindi $\mathcal{D}_x = [D_x(X),\, D_x(Y),\, D_x(Z)]$.
Se volessimo calcolare la tabella per figura soprastante per il nodo x, quindi calcolare $\mathcal{D}_x$, al primo passo avremo una tabella formata da $0, 2, 7$ (uniche distanze conosciute dal nodo x) ed ∞ per le restanti distanze non conosciute.
Dopo l'inizializzazione, ciascun nodo invia il proprio vettore ai suoi vicini come mostrato dalle frecce dalla prima alla seconda colonna delle tabelle. Per esempio, il nodo x invia il suo vettore delle distanze $D_x = [0, 2, 7]$ ai nodi y e z. Dopo aver ricevuto gli aggiornamenti, i nodi ricalcolano il vettore delle distanze. Per esempio, il nodo x calcola:

- $D_x(x) = 0$

- $D_x(y) = min\{c(x,y) + Dy(y), c(x,z) + Dz(y)\} = min\{2+0, 7+1\} = 2$
- $D_x(z) = min\{c(x,y) + Dy(z), c(x,z) + Dz(z)\} = min\{2+1, 7+0\} = 3$

Dopo aver ricalcolato i rispettivi vettori delle distanze, se i nodi hanno aggiornato il proprio vettore delle distanze, inviano la versione aggiornata ai propri vicini; notiamo, infatti, che solo i nodi x e z inviano aggiornamenti: il vettore delle distanze del nodo y non è cambiato e pertanto non è stato spedito.
Il tutto continua finché non vi è più alcun messaggio di aggiornamento.

Cosa avviene nel caso di un **miglioramento** del peso di un arco? ...

Quali sono i **problemi** di Distance Vector? Supponiamo che, dato il seguente grafo ed i seguenti vettori delle distante, ad un certo tempo t_i vi sia un peggioramente del peso di un arco (x, y), da 4 a 60.
0. Prima che il costo del collegamento cambi, segue y $\{x : 4, z : 1\}$, z $\{x : 5, y : 1\}$.

1. y rileva che il costo del collegamento è passato da 4 a 60 e calcola il suo nuovo percorso a costo minimo.
2. Ovviamente, con la nostra visione globale della rete, possiamo rilevare che questo nuovo costo attraverso z è errato. Ma l'unica informazione che il nodo y possiede è che il costo diretto verso x è 60 (e non più 4) e che z ha ultimamente detto a y di essere in grado di giungere a x con un costo di 5. Abbiamo, quindi, un instradamento ciclico: al fine di giungere a x, y fa passare il percorso per z, il quale a sua volta lo fa passare per y.
3. z riceve il nuovo vettore delle distanze di y,che indica che il costo minimo di y verso x è 6, sa che può giungere a y a costo 1 e quindi calcola un nuovo costo minimo verso x pari a $D_z(x) = min\{50+0, 1+6\} = 7$. Dato che il costo minimo di z verso x è aumentato, z informa y del suo nuovo vettore delle distanze.
4. Analogamente, dopo aver ricevuto il nuovo vettore delle distanze di z, y determina $D_y(x) = 8$ ed invia a z il suo nuovo vettore delle distanze; z allora determina $D_z(x) = 9$ e invia a y il suo nuovo vettore delle distanze, e così via.

Il questo specifico caso, il tutto proseguirà per 44 iterazioni (scambi di messaggi tra y e z), fino a quando z considera il costo del proprio percorso attraverso y maggiore di 50. Tuttavia, cosa sarebbe successo se il costo fosse aumento per un valore molto maggiore rispetto 60?
Questo è il **problema del conteggio all'infinito**.
L'idea è semplice: se z instrada tramite y per giungere alla destinazione x, allora z avvertirà y che la sua distanza verso x è infinita, ossia z comunicherà a y che $D_z(x) = +\infty$, anche se in realtà z sa che $D_z(x) = 5$, e continuerà a dire questa piccola bugia fintanto che instrada verso x passando per y.

Qual è la **soluzione** al problema? Soluzione al problema descritto è data da **Poison Reserve**. Attraverso quest'ultimo si evita la diffusione di informazioni false utilizzando ...

> Esiste una configurazione in cui Poisoned Reverse non risolve il problema del conteggio all'infinito? Sì, in tutte quelle configurazioni in cui esistono cicli che non riguardano semplicemente due nodi adiacenti.

Link State

In un instradamento link-state la topologia di rete e tutti i costi dei collegamenti sono noti, ossia disponibili in input all'algoritmo. Ciò si ottiene facendo inviare a ciascun nodo pacchetti sullo stato dei suoi collegamenti a tutti gli altri nodi della rete.
L'algoritmo di calcolo dei percorsi che presentiamo associato all'instradamento link- state è noto come **algoritmo di Dijkstra**.
L'algoritmo di Dijkstra calcola il percorso a costo minimo da un nodo (l'origine, che chiameremo u) a tutti gli altri nodi nella rete, è iterativo e ha le seguenti proprietà: dopo la $k-esima$ iterazione, i percorsi a costo minimo sono noti a k nodi di destinazione e, tra i percorsi a costo minimo verso tutti i nodi di destinazione, questi k percorsi hanno i k costi più bassi. L'algoritmo di instradamento *link-state* consiste in un passo di inizializzazione seguito da un ciclo che viene eseguito una volta per ogni nodo del grafo. Quando termina, l'algoritmo avrà calcolato il percorso minimo dal nodo origine u a tutti gli altri nodi.

Confronto tra LS e DV

Distance Vector	Link State
Decentralizzato, poiché ogni nodo esegue autonomamente il calcolo per i costi minimi	**Globale**, poiché si basa sulla "*fotografia*" dell'intero grafo (i.e. rete)
Basato su **messaggi** provenienti dai soli nodi **vicini**	Basato su **messaggi** provenienti da ogni nodo nel **sistema** (i.e. broadcast)
Informazioni riguardanti ogni nodo	**Informazioni** riguardanti solo i propri link
Problema del **conteggio all'infinito**	Problema delle **oscillazioni**
Invio di un **messaggio** all'aggiornamento della propria **tabella**	Invio **periodico** di un **messaggio**
Poco robusto agli **attacchi**	Robusto agli **attacchi** (e.g. dirottare il traffico)

RIP

Il protocollo **RIP** (acronimo di *Routing Information Protocol*) è basato su **Distance Vector** ed utilizza gli **hop** come metrica per il calcolo del peso di un arco nel grafo, ossia il numero di salti effettuati da un nodo all'altro. Si noti che RIP non tiene conto della situazione del singolo link.
Esistono due **versioni** di RIP:

- **RIPv1**, descritta all'interno di $RFC\ 1058$, prevede un numero massimo di salti pari a 15 , il che pone anche un limite al diametro del grafo, utile a velocizzare la convergenza. Secondo RIPv1, le tabelle di routing vengono scambiate ogni $30\ sec$; se un percorso non viene aggiornato per un tempo maggiore di $180\ sec$, la sua distanza è settata ad ∞ . Passati altri $120\ sec$ dal set ad ∞ scatta il *Garbage-Collection Timer*, per cui il nodo

irraggiungibile viene eliminato dalla tabella di routing.
All'interno di RIP, esistono due **tipi di messaggi** per RIPv1:

- *REQUEST*, per chiedere informazioni ai nodi adiacenti;
- *RESPONSE*, per inviare informazioni di routing;

Cosa contiene una **tabella** di routing secondo RIPv1? Ogni tabella contiene:

- Un indirizzo di destinazione;
- Una distanza dalla destinazione in *hop*;
- Un parametro *Next-Hop* uguale al router adiacente;
- Un *timeout*, un timer;
- Un *Garbage-Collection Timer*;

- **RIPv2**, descritta all'interno di $RFC\ 2453$, include il trasporto delle informazioni sulla maschera di sottorete, supportando così il CIDR (Classless Inter-Domain Routing). Per garantire la sicurezza degli aggiornamenti sono disponibili 2 metodi: autenticazione semplice con testo in chiaro ed MD5. Per mantenere la retrocompatibilità il limite di *hop count* rimane a 15;

OSPF

Il protocollo **OSPF** (acronimo di *Open Shortest Path First*) è basato su **Link State** e consente l'uso di differenti metriche per il calcolo del peso di un arco nel grafo.
Ogni nodo invia un messaggio ad un nodo vicino, rimanendo in attesa della risposta. In base al tempo della risposta il nodo può determinare la bontà del link. Il tempo di risposta, chiaramente, tiene conto *anche* di quanto traffico c'è nel link.
Inoltre, il tutto avviene contemporaneamente tra i due nodi, il che permette di bloccare eventuali imbrogli.

In seguito, ogni nodo prepara poi una sua tabella con i costi determinati per ogni suo link. La tabella viene poi inviata in **flooding** verso gli altri nodi del sistema. Si noti che la tabella è accompagnata da un ID, il quale permette che l'invio di quest'ultima all'interno della rete *non* tenda a crescere fino ad inondare l'intera rete, con la possibilità che possa pure tornare al nodo di origine.
Pur mantenendo un ID, se non fermato in tempo, il flooding potrebbe congestionare la rete. Per questo motivo utilizziamo **LSA** (acronimo di Link State Advertisement), ovvero informazioni di instradamento inviate via broadcast ogni qualvolta che si verifica un cambiamento nello stato di un link (e.g. una variazione di costo o un cambiamento di disponibilità) a tutti gli altri router nel sistema autonomo.

Quali sono i tipi di **messaggi** in OSPF?
Rispetto i messaggi in OSPF, distinguiamo 3 tipologie:

- **HELLO**, utile a scoprire e verificare i vicini;

- **EXCHANGE**, utile alla sincronizzazione iniziale del DB;
- **FLOODING**, utile all'aggiornamento del DB.

BGP

Il protocollo BGP è un protocollo di routing tra AS, cui scopo è realizzare dorsali di comunicazioni basate a partire da motivazioni che possono essere anche politiche.
Ogni AS possiede uno o più ASBR, ossie di router di bordo in grado di mettere in comunicazione due AS. Il protocollo BGP viene eseguito sui BGP Speaker, i quali non sono necessariamente dei ASBR.

Livello di collegamento

All'interno del livello di collegamento faremo riferimento al concetto di (i) **nodo**, ovvero un qualsiasi dispositivo ed (ii) al concetto di **canale di comunicazione** (o *link*), che collega nodi adiacenti.

Dov'è **implementato** il livello di collegamento? Il livello di collegamento è sostanzialmente implementato all'interno dell'hardware, in particolare all'interno della scheda di rete. Tuttavia esistono parti di esso implementate mediante software, il che indica che il livello di collegamento è una combinazione dei due.

Quali sono i **servizi offerti** dal livello di collegamento? Sebbene il servizio di base del livello di collegamento sia il trasporto di datagrammi da un nodo a quello adiacente lungo un singolo canale di comunicazione, protocolli a livello di collegamento possono includere i seguenti servizi:

- **Framing**.
  I protocolli a livello di collegamento incapsulano i datagrammi provenienti dal livello di rete all'interno di una **frame**, questa costituita da un campo dati (in cui viene inserito il datagramma) ed un campo d'intestazione. Si noti che la struttura della frame varia in base al protocollo.
- **Accesso al collegamento**, attraverso cui vengono specificate le regole per immettere le frame all'interno di un canale di comunicazione.
- **Consegna affidabile**.
  Abbiamo già visto che protocolli a livello di trasporto (e.g. TCP) implementano bene il servizio di consegna affidabile. Lo stesso può essere implementato all'interno del livello di collegamento per collegamenti ad elevato tasso di errore.
  Questo servizio può essere spesso considerato futile per collegamenti che presentano un basso tasso di errore, come per esempio la fibra ottica o il cavo coassiale.
- **Rilevazione e correzione degli errori**.
  All'interno di una frame possono talvolta verificarsi degli errori sui bit, causati dall'attenuazione di segnale o disturbi elettromagnetici.
  Molti protocolli di trasporto forniscono un modo per rilevare la presenza di quest'ultimi per poi effettuarne la correzione.

Framing dei dati

Immaginiamo di avere un canale di trasmissione come la *fibra ottica*; lo stato di questa trasmissione può essere *accesso* e *spento*, *presenza di luce* o *assenza di luce*. E' possibile, quindi, pensare di rappresentare un bit 1 come la presenza di luce ed un bit 0 come l'assenza di luce. Tuttavia, così facendo, non è possibile distinguere l'assenza di segnale ad una serie di bit 0. Questo tipo di trasmissione è chiamata **trasmissione sincrona**, caratterizzata dal fatto che viene sempre trasmesso qualcosa.

Una soluzione *banale* è l'adozione di un sistema che preveda non più 2 soli livelli, bensì 3 **livelli**. Secondo un sistema del genere, un primo livello rappresenterebbe il bit 1, un secondo livello l'assenza di segnale ed un terzo livello il bit 0. Utilizzare un livello esclusivamente per indicare l'assenza di segnale indica sprecare un terzo della banda totale (i.e. *problema di ridondanza*).

Esistono altre tecniche che permettono di *limitare la ridondanza*. Ad esempio, continuando ad usare 3 livelli, smettiamo di associare un bit ad un livello ben preciso. Secondo questo schema, un bit 1 corrisponde al cambio di livello (in altre parole, dal secondo livello centrale mi sposto al primo livello soprastante), un bit 0 prevede di rimanere nello stesso livello. Uno schema del genere limita le alte frequenze.

Quest'ultimo schema può essere implementato anche secondo 5 **livelli**, in cui a meno del livello assegnato all'assenza di segnale, per ogni livello viene associato non più un bit bensì una coppia di bit.

Un altra alternativa si basa sull'utilizzo della **codifica 4B5B**. Attraverso questa codifica, l'invio di una stringa di $4\,bit$ prevede di aggiungere un bit di *ridondanza* quindi l'invio di $5\,bit$, calcolata attraverso una tabella prestabilita.
Secondo 4B5B, non esistono sequenze dove vi sono solo bit dello stesso tipo; in altre parole, non vi sono sequenze di soli 1 o soli 0, motivo per cui più 0 non può che indicare l'assenza di segnale.
All'utilizzo di 4B5B consegue il **vantaggio** di permettere l'utilizzo di un sistema a 2 livelli ma anche lo **svantaggio** relativo ad uno spreco pari ad $\frac{1}{5}$. Per esempio, supponendo di voler trasmettere a $100\ Mbps$, aggiungendo un bit sul cavo andremo ad effettuare una comunicazione a $125\ Mbps$, uno spreco al 25% della banda totale.

Un ultima alternativa si basa sull'utilizzo della **codifica 8B10B**. Attraverso questa codifica, l'invio di una stringa di $8\,bit$ prevede di aggiungere un bit di *ridondanza* quindi l'invio di $10\,bit$, calcolata attraverso una tabella prestabilita.
Diversamente dalla codifica 4B5B, la codifica 8B10B gode della proprietà di essere **elettricamente neutra**, il che indica che il numero di $bit\,1$ viene mantenuto il più possibile uguale al numero di bit $bit\,0$. Questa proprietà evita che vi possa essere un *trasferimento di energia* tra un dispositivo e l'altro.
Quest'ultima codifica è utilizzata in sistemi come USB 3.0, SATA ed HDMI.
Attravero 8B10B, ogni stringa da $8\,bit$ viene suddivisa in un primo gruppo formato da $5\,bit$ ed un secondo formato da $3\,bit$, i quali vengono trasformati rispettivamente in gruppi di $6\,bit$ e $4\,bit$.
Concludiamo affermando che nella codifca 8B10B non vi è alcun *sbilanciamento*; casi di *squilibrio* sono possibili e vengono sistemati successivamente.

Rilevazione e correzione degli errori

Vediamo ora tre tecniche per rilevare gli errori nei dati trasmessi: **controllo di parità** (*parity check*), per illustrare l'idea di base della rilevazione e correzione degli errori ed il **controllo a ridondanza ciclica** (*CRC*).

Controllo di parità

La forma più semplice di controllo degli errori è il **controllo di parità**. Supponiamo che $Alice$ debba inviare un informazione D formata da $d\ bit$ a Bob. Applicando il controllo di parità, $Alice$ sceglie il suo valore in modo da rendere pari il numero totale di $bit\ 1$ all'interno dei $d\ bit$, adesso $d+1\ bit$ (vale il contrario per lo schema di parità dispari).
In questo modo, l'unica operazione da svolgere per il ricevente è il conteggio dei $bit\ 1$, i quali se dispari indicano che si è verificato *almeno* un errore.

Cosa succede se il numero verificatosi è pari? In questo caso il controllo di parità non può aiutarci ma pulò farlo una sua variante bidimensionale. Quest'ultima generalizzazione del controllo di parità si basa sul disporre i $d\ bit$ da inviare a Bob in una matrice composta da i righe e j colonne e calcolare il bit di parità per ogni riga e colonna. $Alice$, oltre ad inviare i $d\ bit$, invia a Bob anche gli $i+j+1$ *bit di parità*, i quali gli permettono di *identificare* e *correggere* il bit alterato.

CRC

Una tecnica di rilevazione dell'errore largamente utilizzata è basata sui **codici di controllo a ridondanza ciclica** (CRC, acronimo di *cyclic redundancy check*).
I codici CRC sono anche detti **codici polinomiali**, in quanto possibile vedere la stringa di bit da inviare come un polinomio cui coefficienti sono i bit della stringa.

Secondo CRC è necessario distinguere due stringe di bit:

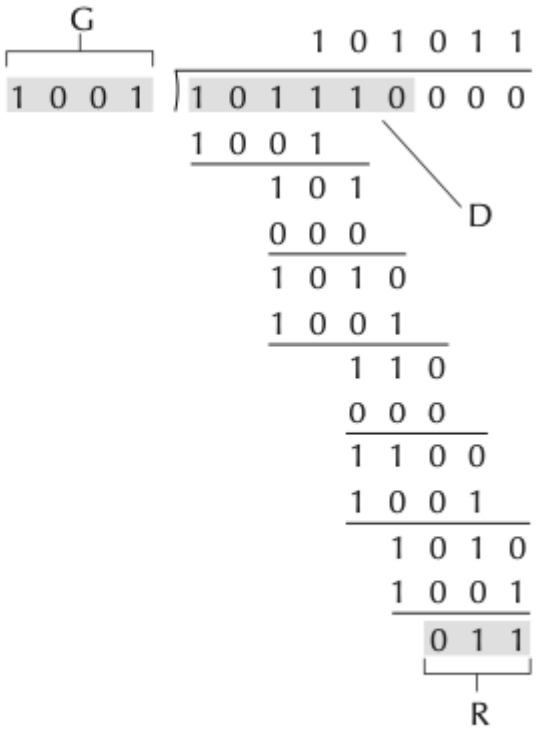
- D, composta dai $d\ bit$ da inviare;
- G, conosciuta come *generatore* e composta da $r+1\ bit$, un stringa di bit per cui si sono accordati mittente e ricevente.

> Sono stati definiti dei generatori standard di 8, 12, 16 e $32\ bit$. Lo standard CRC-32 a $32\ bit$, in numerosi protocolli IEEE del livello di collegamento, usa il generatore: $G_{CRC-32} = 100000100110000010001110110110111$

L'idea alla base di CRC è la seguente: data una stringa D (formata da $d\ bit$) da inviare, segue la scelta di R (formato da $r\ bit$ addizionali). L'unione di D e R deve formare un numero binario esattamente divisibile per G. Se il ricevente nota per la divisione un resto diverso da 0, allora egli sa che si è verificato un errore.

Tutti i calcoli di CRC sono eseguiti in aritmetica modulo 2, il che indica che addizione e sottrazione sono operazioni uguali ed equivalenti a svolgere uno *XOR*. D'altra parte, moltiplicazioni e divisioni come in base 2; la moltiplicazione di una stringa per 2^k equivale ad uno *shift a sinistra di k posizioni*.
Per questo motivo, possiamo intendere $D * 2^r\ XOR\ R$ è uguale alla stringa $d+r\ bit$.

Tornando alla questione cruciale, ovvero trovare R tale che $D * 2^r\ XOR\ R\ =\ nG$, ossia scegliere R in modo tale da che G sia divisibile per $D * 2^r\ XOR\ R$ senza resto.
Notiamo che se eseguiamo l'operazione di XOR di R per entrambi i membri dell'espressione otteniamo $D * 2^r\ =\ nG\ XOR\ R$. In altre parole, possiamo calcolare $R = resto\ di\ (D * 2^r\ /\ G)$



Esempio di calcolo di CRC.

In definitiva, CRC può rilevare errori a *burst* inferiori a $r + 1$ bit: ovvero tutti gli errori consecutivi di non oltre r bit saranno rilevati.

Distanza di Hamming

Possiamo definire la **distanza di hamming** come il numero di bit diversi fra una stringa di bit e l'altra. In altre parole, la distanza di hamming tra due *codeword* è uguale al numero di bit necessari per passare dall'una all'altra.
L'esempio sottostante prevede una distanza di hamming pari al numero di bit 1 presenti all'interno del risultato all'operazione di XOR, ovvero 2.

$$10001100 \ \ XOR \ \ 11000100 \ \ = \ \ 01001000$$

Un **vocabolario** per stringhe di n bit è un sotto-insieme di tutte le possibili combinazioni, ovvero un sotto-insieme delle 2^n stringhe di n bit. Possiamo definire la distanza di hamming per un vocabolario di *codeword* (i.e. stringhe di bit) come il minimo numero di bit diversi fra una *codeword* e l'altra.

Come avviene la **rilevazione di un errore**? Il mittente può generare solo codeword appartenenti al vocabolario; se il canale introduce un errore, allora la rilevazione dell'errore da parte del ricevente è banale.
In particolare, il tutto si basa sulla statistica poiché possiamo affermare che la probabilità che si verifichi un errore sia molto maggiore rispetto la probabilità che si verifichino due errori e così via...

$$P(Errori = 1) \ >> \ P(Errori = 2) \ >> \ P(Errori = 3) \ldots$$

Quindi, ipotizzando che il canale abbia introdotto un errore all'interno della codeword, possiamo supporre con forte probabilità che *la codeword inviata in origine dal mittente sia la codeword nel vocabolario con distanza di hamming minore rispetto la codeword con*

errore. E' importante specificare questa probabilità poiché la correzione di un errore attraverso la distanza di hamming viene effettuata esclusivamente su base probabilistica.

Consideriamo, per esempio, un vocabolario formato dalle seguenti codeword: $000000, 000111, 111000, 111111$, un sotto-insieme delle 64 codeword possibili. Chiaramente, la distanza di hamming fra le varie codeword è uguale a 3 o 6, per questo motivo possiamo definire distanza di hamming per il vocabolario pari al minimo dei due, ossia 3.
Supponendo che al ricevente arrivi una codeword uguale a 101000, egli può supporre con forte probabilità che la codeword inviata dal mittente fosse 111000, codeword per cui vi è distanza di hamming minima (i.e. 2).

A partire dall'esempio precedente, possiamo affermare che data una codeword non valida, nel caso peggiore egli avrà distanza di hamming pari ad 1 rispetto una codeword nel vocabolario.
Inoltre, è possibile affermare che *il vocabolario che permette di correggere errori singoli all'interno di una codeword è un vocabolario con distanza uguale a* 3.

In generale, per correggere un numero n di errori all'interno di una codeword è necessario un vocabolario con distanza $d = 2n + 1$.
D'altra parte, per effettuare un rilevamente di n errori all'interno di una codeword è necessario un vocabolario con distanza $d = n + 1$.

> Una codeword *illecita* (i.e. non presente all'interno del vocabolario) può avere distanza di hamming uguale ad 1 con più codeword all'interno del vocabolario?
> No, poiché altrimenti la distanza per il vocabolario sarebbe uguale non più 3 ma 2.

A partire da una stringa di $m\,bit$ di **dati**, quanti $r\,bit$ di **ridondanza** sono necessari per correggere singoli errori?
Supponendo stringhe di $n\,bit$, con $n = m + r$, segue un numero di codeword totali pari a 2^n, di cui solo 2^m saranno *lecite*. Notiamo che a partire da una stringa di bit contenuta all'interno dell'insieme di codeword lecite (2^m), seguono $n + 1$ possibili modi per generare codeword illecite.
Calcolare tutte le possibili codeword illecite per ogni codeword lecita è uguale a $(n + 1) * 2^m$.

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

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

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

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

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

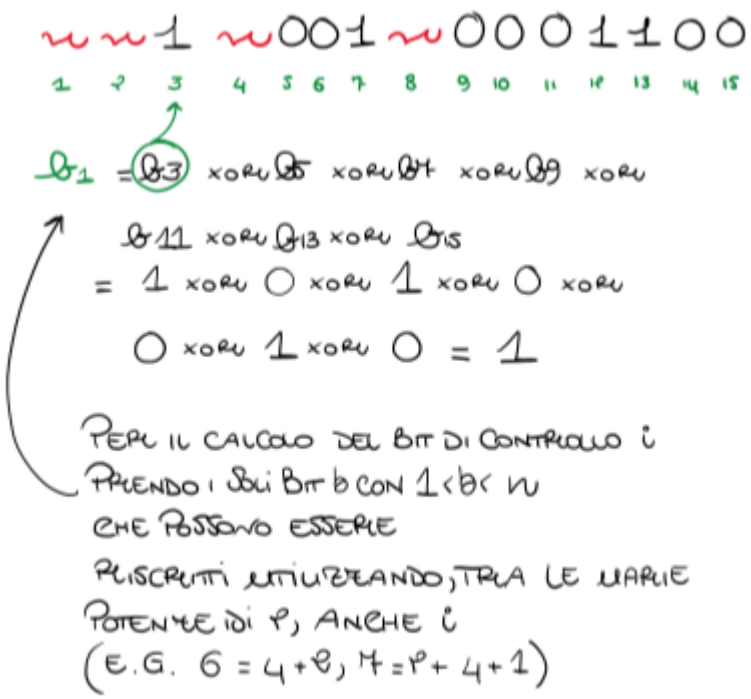
E' possibile costruire un diagramma rispetto tutti i possibili valori di m ed r; è importante notare che il valore di $n = m + r$ non è mai una potenza di 2, poiché nel momento in cui $m + 1 = 2^r - r$ allora avviene sia un incremento di m che un incremento di r.

Trovati gli $r\,bit$ necessari a correggere errori singoli, in che modo viene stabilita la **posizione** dei bit di controllo all'interno della codeword?
Sia data una codeword formata da n bit, fissata la posizione dei bit in ordine crescente da sinistra verso destra (da 1 verso n), possiamo fissare i bit di controllo come tutte quei bit cui

posizione i sia uguale ad una potenza di 2; tutti gli altri bit saranno bit di dati.

001**1**001**0**000

Abbiamo capito che un bit di controllo è sempre un bit in posizione i, con i uguale ad una potenza di 2, ma come **stabilire il bit di controllo**? Proviamo a capirlo con il seguente esempio.



Protocolli di accesso multiplo

Abbiamo osservato che esistono due tipi di collegamento di rete: *punto-a-punto* e *broadcast*. Il collegamento *punto-a-punto* è sostanzialmente costituito da un trasmittente ed unico ricevente, il collegamento *broadcast* può avere più nodi trasmittenti e riceventi connessi allo stesso canale condiviso. Il termine broadcast indica che, quando un nodo trasmette una frame, il canale lo diffonde e tutti gli altri nodi ne ricevono una copia. *Ethernet* e *Wireless LAN* sono esempi di tecnologie broadcast.

Il problema principale in un collegamento broadcast è determinare chi e quando debba parlare (i.e. trasmettere). Dato che tutti i nodi sono in grado di trasmettere frame, è possibile che due o più lo facciano nello stesso istante, per cui tutti i nodi riceveranno contemporaneamente più frame, generando una collisione e conseguentemente una perdita della frame.
Protocolli di accesso multiplo fissano le modalità con cui i nodi regolano le loro trasmissioni. In generale possiamo classificare praticamente tutti i protocolli di accesso multiplo in una di queste categorie: **protocolli a suddivisione del canale**, **protocolli ad accesso casuale** e **protocolli a rotazione**.

Protocolli a suddivisione del canale

Esistono tre tecniche per la suddivisione di un canale broadcast fra i nodi che lo condividono; dati N nodi ed una velocità di R bps, protocolli a suddivisione del canale sono:

- **TDM**, il quale suddivide il tempo in intervalli di tempo per poi dividere ogni intervallo in N slot temporali, uno per ogni nodo. Ogni volta che un nodo ha un pacchetto da inviare, egli trasmette i bit del pacchetto durante lo slot di tempo assegnatogli. Generalmente le dimensioni dello slot sono stabilite in modo tale da permettere al più l'invio di un solo pacchetto. Una volta che tutti hanno avuto l'opportunità di parlare secondo il loro slot, la sequenza si ripete. In definitiva, TDM ha il vantaggio di evitare le collisioni ed essere imparziale, ma anche lo svantaggio di mantenere sempre ed al più un tasso trasmissivo di $R \ / \ N$ bps, anche quando vi è un solo nodo che vuole trasmettere.
- **FDM**, il quale suddivide in N frequenza differenti, una per ogni nodo. Anche FDM ha il vantaggio di evitare le collisioni ed essere imparziale, tuttavia possiede anche lo svantaggio di permettere un tasso trasmissivo pari ad $R \ / \ N$ bps, anche quando vi è un solo nodo che vuole trasmettere.
- **CDMA**, terzo ed ultimo protocollo a suddivisione del canale, si basa sull'assegnare un codice ad ognuno degli N nodi. Ciascun nodo utilizza il proprio codice univoco per codificare i dati inviati e, se i codici sono scelti accuratamente, viene consentito la trasmissione simultanea da parte di nodi differenti.

Protocolli ad accesso casuale

I protocolli ad accesso casuale si basano sulla ritrasmissione di una frame ripetuta finchè questa non viene persa a causa di una collisione, quindi arriva a destinazione. La ritrasmissione non è immediata, bensì avviene dopo un periodo di tempo random. Protocolli ad accesso casuale sono i seguenti:

- **Slotted ALOHA**, primo e più semplice protocollo di accesso casuale, si basa sulla possibilità di inviare un frame da $L \ bit$ esclusivamente all'inizio di uno slot di tempo. Infatti, il tempo è suddiviso in $L/R \ secondi$, il che equivale ad avere la possibilità di trasmettere al più una frame per ogni slot. Inoltre, se avviene una collisione allora tutti i nodi devono esserne a conoscenza prima dello scadere dello slot. I **vantaggi** di Slotted ALOHA sono: (i) la possibilità di trasmettere ad $R \ bps$ nel caso in cui vi sia un solo nodo attivo ed (ii) è fortementente *decentralizzato* poichè ciascun nodo decide in modo indipendente quando ritrasmettere la frame in caso di collisione. Tuttavia, Slotted ALOHA presenta anche i seguenti **svantaggi**:
  - Gli slot su cui si verifica una collisione vengono sprecati;
  - Poichè ogni nodo decide in modo indipendente quando ritrasmettere, alcuni slot potrebbero risultare vuoti.
  - A causa della probabilità di successo per ognuno degli N nodi pari a $N * p * (1-p)^{N-1}$, segue un efficienza massima pari al 37%, il che indica una velocità di trasmissione pari al più a $0,37 * R \ bps$.

- **ALOHA**, secondo cui appena una frame va in collisione, il nodo la ritrasmette immediatamente con probabilità p. Per determinare l'efficienza di ALOHA, notiamo che ad ogni istante esiste un nodo con probabilità p. Supponendo che la trasmissione di una frame i inizi al tempo t_i, segue che affinché quest'ultima venga trasmessa con esisto positivo nessun altro nodo deve iniziare a trasmettere nell'intervallo $[t_{(i-1)}, t_0]$, poichè se così fosse si verificherebbe una sovrapposizione. La probabilità che nessun nodo inizi a trasmettere a $[t_{(i-1)}, t_0]$ è uguale a $(1-p)^{N-1}$. Analogamente, nessun nodo deve trasmettere mentre la frame i viene trasmessa, ovvero nell'intervallo $[t_0, t_1]$, il che porta la probabilità di successo uguale a $(1-p)^{2*(N-1)}$.

- **CSMA**, attraverso cui si trova una soluzione ai problemi di ALOHA. I due protocolli ALOHA, infatti, prevedono che un nodo non si curi del fatto che possa esserci qualche nodo che sta trasmettendo, il che equivale ad un maleducato che continua a parlare senza preoccuparsi che altri stiano conversando tra loro. Esistono due regole per evitare questo problema:

  - *Ascoltare prima di parlare*, che nel mondo delle reti si traduce nel **rilevamento della portante**; indica di aspettare che il canale si liberi prima di inziare a trasmettere.
  - *Se qualcun altro inizia a parlare insieme a noi, noi smettiamo di parlare*, che nel mondo delle reti si traduce nel **rilevamento della collisione**; indica che se un nodo che sta trasmettendo nota una frame che interferisce con la sua, allora egli arresta la sua trasmissione, aspettando un tempo casuale prima di ritrasmettere.

  Queste due regole sono alla base dei due protocolli che studieremo: **CSMA** e **CSMA/CD** (ovvero, *CSMA con il rilevamento della collisione*).

  Una domanda rispetto **CSMA** è la seguente: *Se esiste rilevamento della portante, continuano ad esistere le collisioni?* All'istante t_0, il nodo B rivela che il canale è inattivo, motivo per cui inizia a trasmettere. E' necessario un intervallo di tempo non nullo affinchè i bit trasmessi da B si propagano sul canale. Nel frattempo, a tempo $t_1 > t_0$, il nodo D, che non è stato ancora raggiunto dai bit trasmessi da B, vuole trasmettere sul canale e, pensando che il canale sia inattivo, inizia a trasmettere i suoi bit.
  Dopo un breve periodo, la trasmissione di B inizia ad interferire con quella di D.
  Poiché attraverso CSMA i nodi non godono della proprietà di rilevamento delle collisioni, entrambi i nodi continueranno a trasmettere l'interno frame danneggiato a causa della collisione.

  Cosa succede se lo **scenario** si verifica in **CSMA/CD**? Poiché attraverso CSMA/CD i nodi godono della proprietà di rilevamento delle collisioni, entrambi i nodi evitano di trasmettere l'inutile ed interno frame danneggiato a causa della collisione.
  Dopo che il primo nodo ha verificato la collisione, infatti, quest'ultimo smette di trasmettere la propria frame, aspettando un tempo *randomico* prima di ritrasmettere la frame.
  Il tempo randomico è scelto attraverso l'algoritmo di **binary exponential backoff**, per cui data l'$n-esima$ collisione durante la trasmissione, viene estratto randomicamente

un valore k dall'insieme $(0, \ldots, 2^{n-1})$. Si noti che anche *Ethernet* fa utilizzo di *binary exponential backoff*.

Qual è l'**efficienza** di **CSMA/CD**? Possiamo definire l'efficienza di CSMA/CD come la frazione di tempo media durante la quale i frame sono trasferiti sul canale senza collisioni in presenza di un alto numero di nodi attivi (i.e. nodi con un'elevata quatità di frame da inviare). Il tuto può essere riassunto dalla seguente **formula**:

$$Efficienza = \frac{1}{1 + \frac{5 * d_{prop}}{d_{trasm}}}$$

d_{prop} = tempo di propagazione massimo per un segnale fra una coppia di nodi;
d_{trasm} = tempo necessario per trasmettere una frame della maggior dimensione possibile.

- **CSMA/BA**
  ...

Protocolli a rotazione

Due proprietà auspicabili per un protocollo di accesso multiplo sono le seguenti:

- Quando vi un solo nodo che vuole trasmettere, allora vi deve essere throughput pari ad $R\ bps$.
- Quando vi sono N nodi che vogliono trasmettere, allora vi deve essere throughput vicino a $R/N\ bps$.
  Purtroppo, i protocolli ALOHA e CSMA possiedono la prima proprietà ma non la seconda. Dal tentativo di definire un protocollo che godesse di entrambe le proprietà, nascono i protocolli a rotazione. Noi studiamo due protocolli a rotazione, ovvero:
- Il **protocollo polling**, nel quale uno dei nodi, chiamato nodo principale, interpella a turno ogni nodo comunicandogli la possibilità di trasmettere (fino ad un massimo di frame). Un esempio di protocollo polling è *Bluetooth*. Il protocollo polling ha il vantaggio di eliminare le collisioni e gli slot vuoti, tuttavia presenta **svantaggi** come:
  - L'introduzione del ritardo per la notifica da parte del nodo principale. Se per esempio vi è un solo nodo attivo (i.e. che vuole trasmettere una frame), a causa del tempo impiegato dal nodo principale per notificare ciclicamente tutti i nodi, l'unico nodo attivo trasmette ad una velocità inferiore rispetto $R\ bps$.
  - Se vi è un guasto al nodo principale, allora l'intero canale diventa inattivo.

- Il **protocollo token-passing** (alias *token-bus*), in cui non esiste alcun nodo principale ma un messaggio di controllo (i.e. token) che circola fra i vari nodi seguendo un ordine prefissato. Se il token che riceve il token non ha frame da trasmettere allora procede all'inoltro del token verso il nodo successivo, altrimenti può trasmettere fino al numero massimo di frame trasmissibili. I problemi di token-passing sono i seguenti:
  - Il guasto di un nodo può mettere fuori servizio l'intero canale.

- Se un nodo non riesce ad inoltrare il token, occorre invocare procedure di recupero.

Ethernet

Ethernet, ideata negli anni $'70$ è senza dubbio la tecnlogia per LAN più diffusa, diffusione dovuto ai suoi enormi vantaggi in termini di costo, semplicità e velocità.

Cosa compone un **pacchetto Ethernet**? Un pacchetto Ethernet è composto da:

- Un **payload** (i.e. campo dati), il quale contiene il datagramma IP, che a sua volta può occupare dai 46 ai 1500 $byte$ (i.e. *MTU*); se il valore è maggiore rispetto il massimo numero di $byte$ allora l'host dovrà *frammentare*, se inferiore rispetto il minimo allora l'host dovrà riempire il payload con $byte$ che verranno poi scartati (i.e. *padding*).
- Un indirizzo di **destinazione**, composto da 6 $byte$, contiene l'indirizzo **MAC** di **destinazione**.
- Un indirizzo della **sorgente**, composto da 6 $byte$, contiene l'indirizzo **MAC** della **sorgente**.
- Un **tipo** (alias *EtherType*), composto da 2 $byte$, permette di specificare il protocollo di rete mediante il relativo numero identificativo (e.g. $0x0806$ per ARP).
- Un controllo a ridondanza ciclica (i.e. **CRC**), composto da 4 $byte$, consente al ricevente di rilevare e correggere un errore nei bit della frame.
- Un **preambolo**, composto da 8 $byte$ di cui 7 $byte$ avranno la forma 10101010 ed 1 $byte$ (l'ultimo) avrà la forma 10101011. Il loro compito e far sì che la schede di rete del ricevente possa sincronizzarsi con il clock della schede di rete del trasmittente. L'ultimo byte, che differisce per gli ultimi 2 bit avvisa la scheda di rete del ricevente che stanno per arrivare "*le cose importanti*".

Si noti che l'utilizzo del preambolo è strettamente legato all'utilizzo della codifica Manchester, in modo tale da poter sincronizzare correttamente i due interlocutori. La **codifica Manchester** è una codifica utilizzata nel passaggio dal livello fisico al livello di collegamento ed basata sulla variazione di segnale; in particolare, esistono due livelli. Attraverso quest'ultima codifica, il bit 1 viene rappresentato secondo il passaggio dall' "*alto verso il basso*"; viceversa, per quanto riguarda il bit 0.
Per carpire bene il passaggio da un bit all'altro è necessario che mittente e ricevente siano sincronizzati: il che è possibile attraverso il preambolo.

Notiamo, inoltre, che Ethernet offre un servizio (i) *senza connessione*, (ii) *privo di handshake* e (iii) *poco affidabile*; tutte queste *"mancanze"* permettono ad Ethernet il suo più grande **vantaggio**, ossia l'essere semplice ed economico.

Quali e quante sono le varie **tecnologie Ethernet**? Ethernet compare in molte forme differenti e con svariate denominazioni come: $10BASE-T$, $10BASE-2$, $100BASE-T$, $1000BASE-LX$ e $10GBASE-T$.
Ogni notazione si divide in 3 parti:

- **Velocità**, $10, 100, 1000$ o $10G$, rispettivamente per $10\ Mbps, 100\ Mbps, 1\ Gbps, 10\ Gbps$
- **Tipo di banda**, quasi sempre $BASE$, indica che il mezzo fisico trasporta solo traffico Ethernet;
- **Mezzo fisico**, come *cavi coassiali* (i.e. 5, più spesso o 2), *doppini in fili di rame intrecciati* (i.e. T) o *fibre ottiche* (i.e. F, FX e BX).

Storicamente, Ethernet venne concepito con i seguenti **cablaggi**:
- *Cavi coassiali*.
- $10BASE5$, spesso e rigido in quanto costituito da un pezzo di rame unico non intrecciato, una guaina isolante, una calza metallica di protezione e una guaina esterna per una protezione totale. La lunghezza massima per un segmento unico è 500 metri ma potevano essere collegati tra di loro con dei ripetitori/amplificatori di segnale (fino ad un massimo di 5 segmenti consecutivi, per una lunghezza totale uguale a $2,5km$). *Intercettare* il cavo interno era possibile attraverso una "*presa a vampiro*", la quale forava la guaina (in punti specifici) per fare contatto con la parte interna.
- $10BASE2$, più flessibile rispetto il precedente; per estendere il cavo era sufficiente tagliare quest'ultimo ed inserire un connettore (pur rovinando il segnale). I segmenti avevano una lunghezza pari a 185-$200\ m$ ed era possibile unire fino a 5 segmenti, per una lunghezza massima di $1\ km$.
- *Doppini in fili di rame intrecciati*.
- $10BASET$, cui segmenti avevano grandezza pari a 100 metri e potevano essere collegati fino a 1024 nodi. Lo schema di collegamento infatti prevedeva un hub centrale, un concentratore ed una connessione diretta tra scheda di rete e concentratore.
- *Fibra ottica*.
- $10BASEF$, la quale può trasmettere fino a $2\ km$ ed avere al più 1024 nodi anche se in realtà la comunicazione è punto-punto come per T.

Cos'è **Fast Ethernet**? Fast Ethernet (con standard $IEEE\ 802.3u$) è un termine collettivo per indicare un numero di standard Ethernet che trasportano il traffico alla velocità di 100 Mbps rispetto alla velocità originale Ethernet di 10 Mbps.
Cablaggi relativi a Fast Ethernet sono i seguenti:

- $100BASE-T4$, una prima implementazione della Fast Ethernet. Richiede quattro coppie twisted pair. Una coppia è riservata per la *trasmissione*, una per la *ricezione* e le restanti due coppie vengono utilizzate alternativamente in una direzione o nell'altra. Ogni coppia permette una velocità pari a 33 Mbps, il che permette a sua volta una velocità pari a $33 * 3$ Mbps verso una direzione (e.g. ->) e 33 Mbps nell'altra (e.g. <-). Attraverso $100BASE-T4$ viene abbandonata la codifica Manchester ed utilizzato un codice inusuale: *8B6T*, per convertire gruppi di $8\ bit$ in 6 cifre in *base-3*.
- $100BASE-TX$, molto simile al precedente a meno del cavo utilizzato, il quale è CAT5. $100BASE-TX$ usa la codifica 4B5B.
- $100BASE-FX$, una versione della Fast Ethernet su fibra ottica. Usa due cavi di fibra, uno per ricevere ed uno per trasmettere. Il traffico di rete usa completamente la banda di 100 Mbps, su un segmento di fibra full duplex fino a 2 km. $100BASE-FX$ usa la codifica 4B5B.

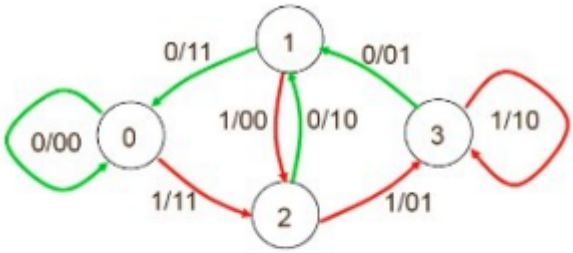
Cos'è **Gigabit Ethernet**? Gigabit Ethernet (con standard $IEEE\ 802.3z$ su *fibra* e $IEEE\ 802.3ab$ su *rame*) è l'evoluzione a 1.000 Mbit/s del protocollo Fast Ethernet operante a 100 Mbit/s.
Ovviamente ottenere tale velocità è semplice con l'utilizzo della **fibra ottica** ($BASE-F$).
D'altra parte, il tutto è possibile con qualche difficoltà anche con $BASE-T$.
Perché implementare sia fibra ottica che **doppini intrecciati**? Semplicemente perché ciò consente di non modificare il cablaggio relativo all'implementazione Ethernet precedente, il quale richiederebbe un costo maggiore.

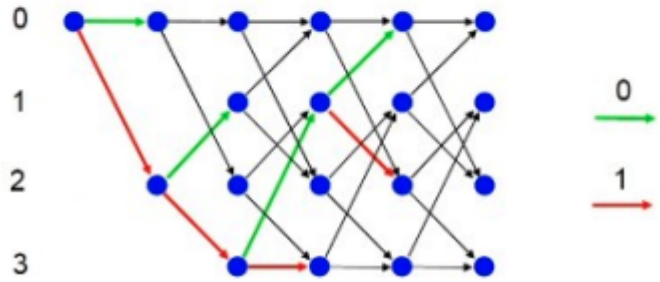
Per raggiungere la velocità di 1 Gbps a partire da 100 Mbps è necessario eseguire i seguenti 5 **passaggi**:

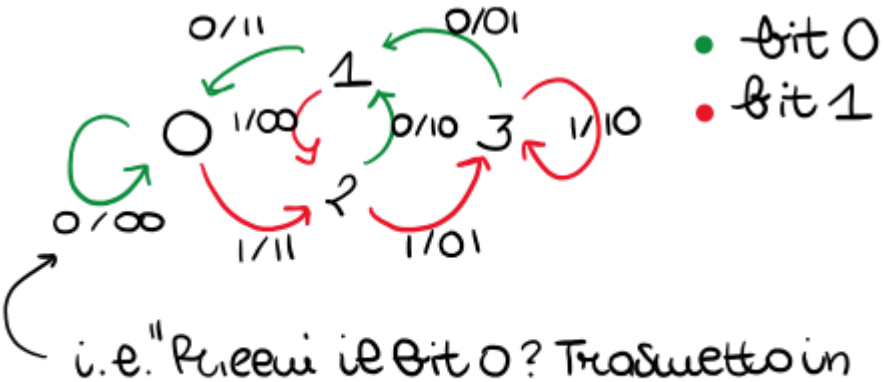
1. Rimuovere la codifica 4B5B, in questo modo passiamo da una velocità uguale a 100 Mbps ad una velocità pari a 125 Mbps;
2. Usare le quattro coppie disponibili simultaneamente, in questo modo passiamo da una velocità uguale a 125 Mbps ad una velocità pari a 500 Mbps;
3. Permettere una trasmissione full-duplex, in questo modo passiamo ad una velocità uguale a 500 Mbps *full-duplex*;
4. Passare da 3 a 5 livelli per *baud*, in questo modo passiamo ad una velocità uguale a 1000 Mbps *full-duplex*;
5. Utilizzare un FEC (acronimo di *Forward Error Connection*), in questo modo possiamo recuperare $6dB$.

> Un *baud* è la possibile variazione di segnale dallo stato precedente. Il *baud* al secondo rappresenta quante variazioni ci possono essere nell'arco di tempo. Ad ogni *baud* si può associare un solo bit oppure 2 bit nella codifica a 5 livelli.

Com'è certamente già notato, il primo passaggio per implementare il tutto all'interno di $BASE-T$ è rimuovere la codifica 4B5B; tuttavia, quale codifica è stata utilizzata? La codifica che è stata utilizzata è la **Codifica Trellis Viterbi**, o più semplicemente la *codifica di Viterbi*.
L'idea su cui si basa quest'ultima codifica è trasmettere $2\ bit$ per ognuno che bit entrante, introducendo così il 100% della ridondanza al fine di recuperare gli errori.
Il tutto è possibile grazie ad un automa a stati finiti, composto da 4 stati ($\{0,1,2,3\}$), ognuno dei quali caratterizzato da un comportamento e due frecce in uscita. L'automa è presente sia alla sorgente che alla destinazione, le quali concordano lo stato iniziale.
In base allo stato e ciò che di vuole trasmettere, invio la determinata coppia di bit.
L'automa può essere talvolta rappresentato anche dal "*traliccio*" presente nella seconda figura sottostate.

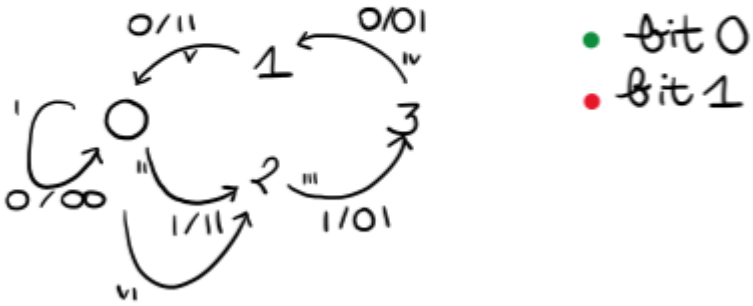

0/11
1
0/01
0/00
0
1/00
0/10
3
1/10
1/11
2
1/01


0
1
2
3
0
1



i.e. "Ricevi il bit 0? Trasmetto in output i bit 00, passando allo stato 0."

Supponiamo di ricevere in input una sequenza di bit uguale a 011001, con stato iniziale 0.



Output: 00 11 01 01 11 11

Se inseriamo degli errori (e.g. 01 11 11 01 11 11) allora ripercorrendo lo stesso percorso (i.e. traliccio) potremmo individuare facilmente gli errori.
Tuttavia, non sappiamo quale bit è quello sbagliato; per capirlo notiamo i possibili cammini e la relativa distanza di Hamming.
Il generale il percorso corretto è quello che mantiene distanza minima.

Collegamenti tra LAN differenti

> Cos'è una **LAN**? Una LAN è un sistema di bus condiviso in cui possono verificarsi collisioni, ovvero il dominio di collisione.

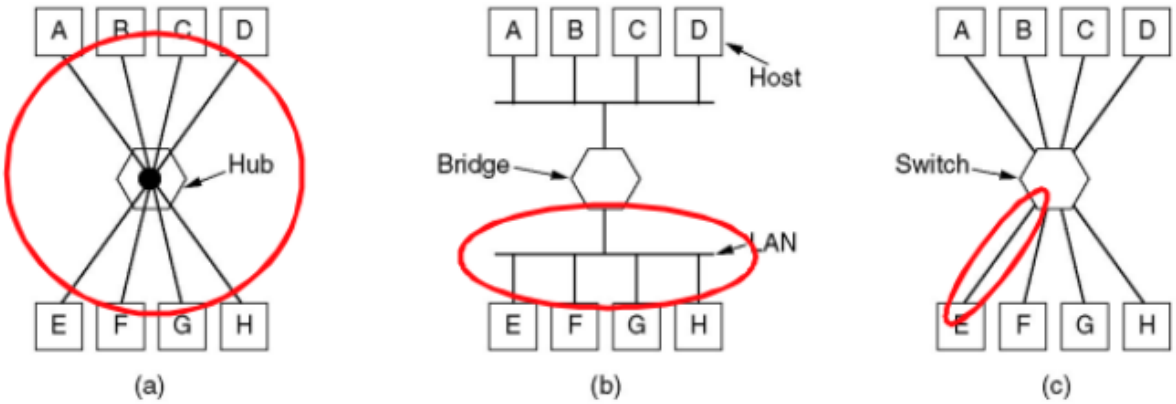
Relativamente a questo argomento possiamo distinguere tra vari dispositivi secondo vari livelli:

- Dispositivi di livello 1, il livello fisico.
  - **Repeater**, un amplificatore in grado di connettere più segmenti tra loro. Un repeater dispone di due porte, una per l'entrata ed una l'uscita, le quali gli permettono di ripetere il segnale tra un segmento ed un altro, con la possibilità di introdurre errori e collisioni
  - **Hub**, un dispositivo a livello fisico con una struttura a stella a cui si collegano più nodi, utilizzato all'interno di Ethernet $10BASE-T$. Un hub può essere inteso come un concentratore in grado di riprodurre elettricamente in uscita i segnali provenienti da una coppia, senza alcun interpretazione del segnale. In altre parole, come un repeater, ha il compito di ascoltare da una porta e ripetere il segnale sulle altre collegate in uscita, con la possibilità di introdurre errori e collisioni (e.g. arrivo di due frame contemporaneamente).

- Dispositivi di livello 2, il livello DLL.
  - **Bridge**, dispositivo che permette di trasformare le frame da un formato LAN (e.g. Ethernet) ad un altro. Attraverso un bridge, le frame provenienti dalla LAN E-H (Figura C. all'interno dell'immagine sottostante) vengono interpretate da qeust'ultimo e se necessario inoltrate verso la seconda LAN A-D.
  - **Switch**, un dispositivo intelligente con un processore, una memoria ed un firmware operativo, i quali permettono l'interpretazione del contenuto della frame Ethernet. Può essere pensato come un bridge specifico per Ethernet. Nello specifico, il compito di uno switch è analizzare l'intestazione di una frame e trovare la strada migliore per raggiungere la destinazione, preparando/utilizzando un percorso che colleghi le due macchine. Gli switch sono, infatti, dei dispositivi *plug and play*, motivo per cui non hanno bisogno di alcuna configurazione. Inizialmente, la tabella per uno switch è vuota ma acquisisce informazioni per ogni frame inviata/ricevuta. Un esempio è possibile attraverso l'immagine sottostante, in cui la macchina A desidera comunicare con la macchina H. In questo caso la frame, arrivata allo switch $B1$ verrà inoltrata su tutte le uscite e scartata dalle macchine non interessate; in questo modo la frame arriva allo switch $B2$, la quale a sua volta inoltra su ogni collegamento fino ad arrivare a $B3$, tramite cui la frame può arrivare ad H. Se H vuole rispondere ad A con una seconda frame, allora il percorso è già conosciuto dagli switch, motivo per cui non si effettua nessun inoltro su tutte le uscite, bensì sulle sole uscite necessarie. Dopo un dato tempo in cui uno switch non riceve frame da un indirizzo, detto *Aging Time*, il determinato indirizzo viene rimosso dalla tabella. Un **problema** relativamente agli switch esiste, ed è legato alla loro topologia. Al fine di prevenire dei cicli, non possono essere costruiti dei grafi; per questo motivo utilizziamo il metodo dello **spanning tree**. *Spanning Tree Protocol* permette la costruzione di un albero a partire da un

grafo ciclico; il tutto si basa sul set di un nodo root (e.g. il nodo con ID inferiore) e la definizione dei figli per ogni nodo. Ogni nodo avrà i figli necessari a rimuovere ogni ciclo nel grafo.

- Dispositivi di livello 3, il livello di rete.
    - **Router**, un dispositivo a livello di rete; differentemente da uno switch, necessita di una configurazione.



Indirizzamento flat vs gerarchico

Abbiamo notato che a livello di collegamento, gli indirizzi hanno un formato **flat**; il MAC Address, infatti, è scelto dalla prodotture della sceda di rete e non contiene alcuna informazione rispetto l'instradamento. Un indirizzo di rete, d'altra parte, è detto gerarchico, poiché il suo valore informa rispetto la posizione della destinazione; in altre parole, fornisce informazioni di instradamento.

L'indirizzamento flat è comodo (vantaggio), infatti, prevede che non vi siano regole di posizionamento, il che indica che l'indirizzo è fisso nell'host ed è inoltre più vantaggioso in termini di spazio di indirizzamento.
Tuttavia (svantaggio) non è possibile creare una LAN con milioni di host, poiché dispositivi come bridge e switch non sarebbero in grado di fornire i loro vantaggi; essi, infatti, imparano dalla rete, ed inizialmente necessitano di un invio in broadcast verso ogni collegamento.

In definitiva, è necessario capire bene come equilibrare bene il tutto, realizzando una struttura logica corretta raggruppando le macchine per tipologia di utenza. A partire da quest'ultimo concetto, nascono le VLAN.

VLAN (LAN Virtuali)

Supponiamo di avere una struttura a tre piani, in cui ogni piano corrisponde ad una LAN. Attraverso una VLAN è possibile connettere macchina che si trovano su LAN differenti. Per definire una VLAN è necessario utilizzare appositi switch, i quali prevedono delle porte suddivise in gruppi; ogni gruppo costituisce una VLAN.

Quali sono i **vantaggi** di una VLAN? Il vantaggio principale di una VLAN si basa sul permettere maggiori prestazioni, dati da:

- Flessibilità, permette di spostarsi su "*piani*" diversi senza alcuna connessione a LAN diverse.
- Prestazioni, il traffico broadcast viene confinato ad utenti appartenti alla stessa VLAN;
- Sicurezza, gli utenti appartenenti a VLAN diverse non vedono i reciproci frame dati.

Possiamo distinguere due diversi tipi di VLAN:

- **untagged**, ovvero sprovviste di un tag identificativo;
- **tagged**, definite con il protocollo $802.IQ$ ed introducono un nuovo campo all'interno di una frame Ethernet: VLAN. Il nuovo campo è formato da $4\ byte$; i primi $2\ byte$ sono utili a specificare il protocollo VLAN, i restanti 2 indicano (i) **priority**, per indicare il livello di priorità, (ii) **CFI**, per indicare se l'indirizzo MAC è canonico o meno ed infine (iii) il campo **VLANID**, il quale indica un indentificatore univoco per la VLAN.

Il protocollo di accesso multiplo conosciuto con il nome "CSMA/BA" è una variante del protocollo CSMA: un protocollo di accesso multiplo ad accesso casuale. CSMA/BA impone: (i) non trasmettere nulla nel caso in cui voglia trasmettere il bit 0 e (ii) trasmettere il bit 1, altrimenti. In questo modo non possono esistere collisioni, poiché se un bit 1 incontra un bit 0 allora *vince* il bit 1, mentre se un bit 1 incontra un bit 1 allora *vince* in ogni caso il bit 1. Ogni nodo, prima di trasmettere un bit, effettua un confronto tra il bit ricevuto ed il bit da trasmettere.

Il protocollo di accesso multiplo conosciuto con il nome "BITMAP" possiede, date N macchine, n slot. In questo modo, ogni macchina può inserire un bit nel proprio slot per indicare la volontà di trasmettere. A questo punto, le macchine con bit 1 nel proprio slot possono comunicare in ordine e senza alcuna possibile collisione.

Livello fisico

Introduzione

Il livello fisico è responsabile del trasporto dei segnali, possibile attraverso:

- Un canale trasmissivo, in grado di trasmettere segnale attraverso la deformazione di una sua caratteristica fisica. Possiamo riconoscere tre diversi tipi di canale trasmissivo, ovvero:
    - Canale trasmissivo perfetto, il quale non causa distorsioni o ritardi durante la propagazione del segnale;
    - Canale trasmissivo ideale, il quale causa esclusivamente un ritardo costante durante la propagazione del segnale;
    - Canale trasmissivo reale, il quale causa attenuazione e ritardi in funzione della frequenza del segnale;
- Un generatore di segnale;
- Un ricevitore di segnale, in grado di percepire il segnale.

Analisi di Fourier

Un qualsiasi segnale di periodo finito può essere rapprestnato per mezzo di una sommatoria infinita di onde sinusoidali (i.e. armoniche), ovvero la somma di seni e coseni.

In particolare, il segnale di periodo finito può essere espresso mediante la somma di sinusoidi e cosinusoidi opportunamente e rispettivamente pesato con un peso a_n e b_n; un segnale possiede frequenza pari all'inverso del suo periodo.
Si noti che il calcolo di a_n e b_n avviene mediante lo svolgimento di determinati integrali e che i due valori sono tali per cui il quadrato di sinusoide e cosinusoide sia uguale ad 1.

In definitiva, esprimere il segnale secondo questa espressione permette uno spostamente dal dominio del tempo al dominio delle frequenze, il quale a sua volta permette vantaggi quali il calcolo dello spettro per il segnale, ovvero l'influenza di ogni armonica al segnale.
In questo modo, è possibile notare come il segnale viene deformato una volta passato per il canale trasmissivo reale.

> Un onda quadra è necessaria ad esprimere un baund, quindi un bit.

> Raddoppiare le frequenze implica dimezzare il tempo e viceversa. D'altra parte, il raddoppio delle frequenze, che implica il raddoppio del *bitrate*, può portare al taglio di alcune parti del segnale da parte del canale trasmissivo, il che comporta un segnale a destinazione di difficile interpretazione.

Come abbiamo già scritto, un canale trasmissivo reale genera attenuazioni e ritardi rispetto il segnale. In particolare, notiamo che quest'ultimo è in grado di trasportare bene il segnale esclusivamente all'interno di un certo range di frequenze. Il segnale al di fuori del range, infatti, viene attenuato.
Per questo motivo, è utile far rientrare il segnale all'interno dello specifico range di frequenze, il che è possibile grazie alla **modulazione**.

Modulazione

La **modulazione**, per ricapitolare, è la tecnica utile a far rientrare il segnale all'interno del range di frequenze che il canale riesce a trattare meglio, senza troppe attenuazioni.
La tecnica di modulazione consiste nell'applicazione di un onda portante $p(t)$ ad un segnale espresso da $s(t)$, ottenendo una nuova funzione $g(t)$
L'onda rappresentata da $g(t)$ sarà il segnale che passa per il canale trasmissivo. Si noti che una volta arrivati a destinazione sarà necessario sottrare l'onda $p(t)$ a $g(t)$.
$p(t) = Asin(\omega * t + \phi)$
Inoltre, possiamo distinguere due tipi di modulazione, ovvero:

- **Modulazione in ampiezza**, il quale permette l'avvicinamento di due elementi lontani;
- **Modulazione di frequenza**, il quale permette una compressione del segnale nella sua parte positiva ed una dilatazione dello stesso nella sua parte negativa;
- **Modulazione in fase**, il quale permette un risultato analogo al precedente.

Quantizzazione

L'operazione di quantizzazione consiste nel limitare il numero di valori che il segnale può assumere. Infatti, generatore e ricevutore non riescono a generare livelli di segnale infiniti, motivo per cui è necessario trasmettere campioni finiti del segnale, limitati verticalmente ded orizzontalmente, quantizzando rispettivamente lungo l'altezza dell'onda e lungo il tempo.

Teorema di Nyquist

Il teorema di Nyquist afferma il legame tra larghezza di banda per un segnale e la quantità di informazioni trasportabili; in particolare, il tutto avviene secondo la seguente formula:

$$max\, bitrate = 2 * H * \log_2 V\ [b/s]$$

In particolare, con H si intende la larghezza di banda, mentre con V intendiamo il numero di livelli differenti verticali.

> E' importante notare che Nyquist non pone limiti rispetto alla quantità di informazioni (i.e. bit rate?) trasportabile da un segnale.

Rumore

Sappiamo che il ricevitore non dissipa tutta l'energia del segnale trasmesso dal generatore. Infatti, finché non viene completamente dissipata, una parte del segnale (i.e. parte non dissipata) continua a circolare all'interno del canale trasmissivo.
Questa parte del segnale rimasta in circolo nel canale è chiamata **rumore**. Il rumore causato dal invio di un segnale i, causa collisioni rispetto i segnali successivi $> i$.

> In Ethernet, un terminatore situato agli estremi del cavo permette la completa dissipazione del rumore.

Possiamo distinguere 2 tipi di rumore, ovvero:

- **Rumore termico**, dovuto al calore, il quale eccita gli atomi e fa sì che quest'ultimi generino oscillazioni. Alle oscillazioni conseguono disturbi elettromagnetici che si sovrappongono al segnale.
- **Interferenza elettromagnetica**, secondo cui un cavo che tende a comportarsi da *antenna*, assorbe energia elettromagnetica presente nell'aria, trasformandola in segnale elettrico che si va a sovrapporre al segnale nel cavo stesso. L'interferenza elettromagnetica può essere limitata attraverso:
  - **Cavo coassiale**, il quale se utilizzato permette di schermare le interferenze attraverso la protezione del conduttore interno, una protezione formata da un conduttore esterno chiamato *treccia*.
  - **Doppino intrecciato**, il quale poiché intrecciato permette al cavo di essere una pessima antenna, quindi limitare le interferenze.

Teorema di Shannon

Il teorema di Shannon afferma il massimo *bit rate* all'interno di un canale reale in cui esiste rumore.

$$max\, bitrate = H * \log_2 (1 + \frac{S}{N})\ [bit/s]$$

In particolare, con H si intende la larghezza di banda, con S si intende la potenza del segnale ed infine con N intendiamo la quantità di rumore presente all'interno del canale.

Attraverso il teorema è possibile dedurre che per ogni canale trasmissivo esiste un limite fisico rispetto il bit rate. Inoltre, maggiore è il bitrate e maggiore sarà labanda da trasmettere, mentre maggiore è la lunghezza del canale trasmissivo e maggiore sarà il rumore.

Per aumentare il bit rate, segue:

- Se H è costante, allora un incremento di S o un decremento di N;
- Se $\frac{S}{N}$ è costante, allora è necessario aumentare H.

> Per Nyquist, poiché non considera il rumore, più livelli indicano maggiore bit rate. Per Shannon, che considera l'esistenza del rumore, più livelli indicano maggiore rumore.

Analogico o digitale?

Essenzialmente, è possibile notare come non esista alcun analogico o digitale, bensì due modi distinti per vedere una trasmissione.

- In un segnale analogico abbiamo un numero enorme di livelli distinti, così tanti da rendere difficile il poter discriminare un livello dall'altro. L'amplificazione di un segnale analogico amplifica il rumore, il quale altera il segnale stesso.
- In un segnale digitale abbiamo un numero di livelli (i.e. stati) molto limitati - tipicamente piccole potenze di 2 - motivo per cui ogni livello è magiormente distante dall'altro, una caratteristica che non permette al rumore che un livello venga discriminato per un altro. L'amplificazione di un segnale digitale è possibile in maniera fedele, quindi senza introdurre errori.

Fibra ottica

Una fibra ottica è formulata da un tubo all'interno del quale passa un raggio luminoso che a contatto con la superficie del tubo stesso, genera una parte riflessa ed una parte rifratta. Quest'ultima parte (i.e. parte rifratta) può essere talvolta nulla se considerato un certo grado di incidenza.

Com'è formata una fibra ottica? Una fibra ottica è composta da:

- Una fibra, un tubo di vetro molto sottile e fragile; si noti che secondo una certa misura, una fibra può essere talvolta piegata di pochi centimetri. In particolare, ogni singola fibra è composta da due strati concentrici di materiale trasparente puro, ovvero:
  - Core, il nucleo cilindrico centrale, avente diametro compreso tra 8 e $50\mu m$.
  - Cladding, ossia un mantello situato attorno al *core* ed avente diametro uguale a $125\mu m$ (i.e. *micron*). Si noti che il cladding, rispetto al core, possiede indice di rifrazione diverso; questo gli permette di assorbire meglio la luce rifratta dal *core*.
- Una guaina in plastica, alias *coating*;
- Un tubo, alias cavidotto, formato da uno spessore uguale a $0.9\ mm$
- Una struttura in kevlar per permettere resistenza alla trazione;
- Una guaina esterna formata da uno spessore uguale a $2 - 3\ mm$

> Si noti che due cavi in fibra possono essere collegati l'uno con l'altro attraverso uno specifico strumento in grado unire microscopicamente qust'ultimi.

> Una fibra si collega ad una scheda di rete attraverso un terminatore in grado di centrare meccanicamente quest'ultimi due.

Quanti tipi di fibre esistono? Conosciamo due tipi di fibre, ovvero:

- Monomodali, in cui la luce avanza assialmente, il che consente distanze maggiori, un maggiore bit rate, ma anche costi più alti.
- Multimodali, le quali possono essere distinte a loro volta in:
  - Step index, all'interno del quale sono presenti più raggi in contemporanea, motivo per cui aumenta il rapporto tra la potenza del segnale ed il rumore, quindi diminuisce la qualità del segnale.
  - Graded index, all'interno del quale sono presenti più raggi in contemporanea che seguono però un andamento "*più continuo*"; motivo per cui aumenta il rapporto tra potenza del segnale e rumore, quindi se pur in misura minore rispetto al precedente, diminuisce la qualità del segnale .

Reti Wireless

Una prima tipologia di rete è quella wireless, caratterizzata dall'assenza di cavi o collegamenti fissi.
Una rete wireless è per definizione broadcast e prevede prestazioni peggiori rispetto le reti cablate.
La più grande difficoltà per una rete wireless è data dai disturbi sulla comunicazione.
Una rete wireless utilizza, per comunicare, onde radio AM, FM o trasmissioni LTE.

Reti telefoniche

Le reti telefoniche sono state progettate per la comunicazione tramite voce umana, motivo per cui non si prestano bene alle comunicazionei tra computer. Il canale viene suddiviso in slot, ed ogni slot è assegnato ad un utente. In questo modo è possibile far viaggiare telefonate diverse sullo stesso filo.
Normalmente, un provider telefonico inserisce nel canale un filtro *passa-basso*, il quale prevede di eliminare tutte le frequenze alte (i.e. $> 3K\,Hz$).

Modem

L'utilizzo di linee telefoniche, essenzialmente analogiche, avviene attraverso un modem, il quale è in grado di trasformare il segnale da digitale ad analogico e viceversa.
Soltamente, un modem prevede una portante di $2100\ Hz$, una frequenza che è possibile

ascoltare con l'orecchio umano. Poiché la frequenza è bassa, per la formula di Nyquist, è necessario inserire più livelli per aumentare il bit rate.
All'interno dei modem, venivano utilizzate due diversi tipi di modulazioni: in ampiezza ed in fase.

Costellazioni

L'utilizzo di questi due tipi di modulazioni nei modem, ha permesso l'ottenimento delle costellazioni, il quale permette la trasmissioni di più bit per *baud*.
In questo modo, sono possibile 16 combinazioni diverse per ogni *baud*. Se ogni punto rappresenta 4 bit, allora trasmettendo segnali a 2400 baud, il modem risulta essere in grado di trasmettere $2400 * 4 = 9600$ bit al secondo.

> Per comunicare tra loro, due modem devono avere la stessa costellazione.

In generale, l'utilizzo di una costellazione prevede l'aumento del throughput.

DSL

Il problema dei modem - ovvero, rendere inutilizzabile il canale telefonico durante il suo utilizzo - viene risolto dalle linee DSL.
Le linee DSL prevedono di lasciar libera una parte del canale èer le comunicazione telefoniche ed un piccolo spazio per evitare disturbi. Il resto del canale è suddiviso il blocchi da $4K$, utilizzabili in parallelo tra loro.
Esiste, quindi, un "*lato modem*" ed un "*lato telefonico*", i quali non interferiscono l'uno con l'altro poiché utilizzano rispettivamente un filtro *passa-alto* ed un filtro *passa-basso*.
Una DSL è in grado di colllegarsi alla centrale telefonica mediante DSLAM.

> Una specializzazione di DSL è ADSL, la quale permette un asimmetria rispetto *downstream* ed *upstream*.