reti di calcolatori

appunti lezioni + riassunto libro, A.A. 2020/2021[1]

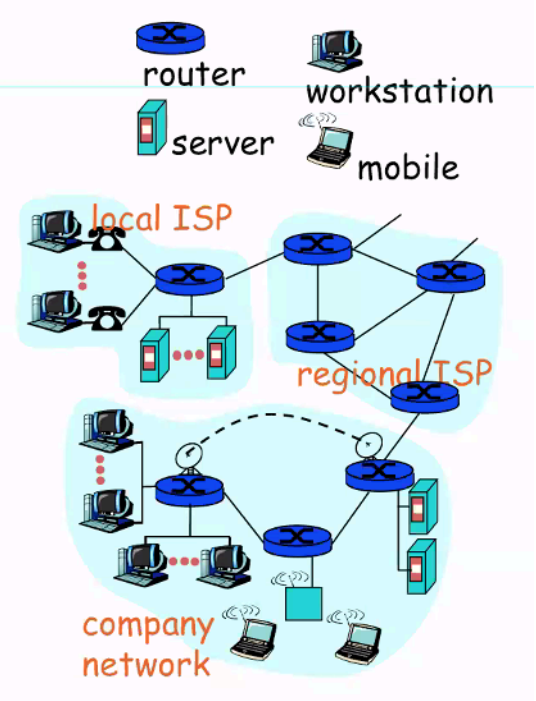
[1] **Disclaimer**. Si tratta delle trascrizioni di appunti presi durante le lezioni del corso di Reti di Calcolatori tenuto dalla prof. Pagani presso l'Università degli Studi di Milano nell'A.A. 2020/2021 ad opera di G.V e R.P.. Poiché si tratta di appunti trascritti potrebbero esserci diversi errori concettuali, passaggi fraintesi ed errori di battitura. Gli autori non si accolla l'onere di garantire la veridicità dei suddetti appunti né è perseguibile in alcuna maniera qualora studiarli non sortisca l'effetto desiderato.

introduzione

perché la rete è importante?
In 60 secondi si registrano milioni di connessioni tra facebook, youtube, snapchat, etc..
ormai non si può prescindere dall'uso della rete nello sviluppo di applicazioni interattive.
I provider italiani, nel periodo di quarantena, hanno visto salire il carico di rete del 50%.

cos'è una rete di calcolatori?
è una collezione interconnessa di calcolatori autonomi. Cioè una collezione di calcolatori (qualunque macchina capace di fare calcoli) autonomi collegata con dei mezzi di comunicazione che permettono di scambiare informazioni.
esistenza di macchine remote esplicita → esiste un server che dichiara un servizio web.

concetti fondamentali
host / end-system = apparati alla periferia della rete, qualcosa di utilizzabile con delle applicazioni che servono all'utente e che interagiscono con la rete → ospiti della rete che **usufruiscono di un servizio di rete** (smartphones, tablet, smart tv, etc...).
router = forward data through network, permette di **trovare una strada** da un end-system a un'altro.
dispositivi di rete (switch, bridges, hub, etc...).
link = tratta che collega 2 router, di fibra ottica, canali radio, satellitari.



utilizzo delle reti
- ➔ comunicazione = email, social network, videoconferenza, etc..
- ➔ condivisione risorse = tramite due modelli:
  - ◆ client-server (come Google Docs, DropBox che sono strutture cloud)
  - ◆ P2P (peer-to-peer come torrent)

aspetti positivi
- ➔ alta affidabilità data dalla replicazione (su altri server)
- ➔ alta potenza di calcolo

possibili problematiche
- ➔ sicurezza e privatezza dei dati : controllo degli accessi (**AAA** = Authentication Authorization Accounting). Mira a capire chi è l'utente che vuole usufruire di una determinata risorsa. Una volta fatta la prima A, l'user che tipo di permesso ha? La terza A definisce se il tipo di risorsa è gratuito o a pagamento, se debba essere contabilizzato e, dunque, addebitato all'utente.

- ➔ Il contenuto delle risorse è qualitativamente corretto? Si verifica l'integrità rispetto alla sorgente.
- ➔ qualità di servizio (QoS). "**Best effort**", ovvero la rete fa lo sforzo migliore per velocizzare i tempi di trasmissione dei dati.

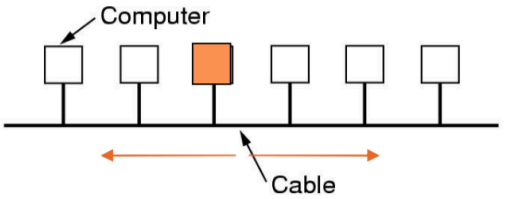
i diversi tipi di rete

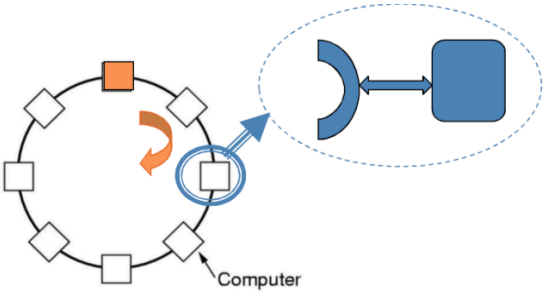
broadcast

E' la categoria più semplice di reti. Le informazioni vengono diffuse da un host e **chiunque può fruirne**. Caratterizzata da hardware, bridges e switch. La sorgente non necessita di conoscere quanti spettatori ci sono (es.: la rete delle radio e TV).
Esistono diverse tipologie di questa rete.

topologia a bus

Molti host collegati ad un unico cavo, detto dorsale della rete, che tutti condividono, e un **host** trasmette dati tramite un cavo di rame (segnale elettrico) o con fibra ottica (segnale luminoso). I dati si propagano su entrambe le direzioni del cavo. Tutti gli host collegati possono fruire di queste informazioni.





topologia ad anello

La dorsale è chiusa su se stessa in modo circolare e ad essa si collegano, con delle tratte di mezzo trasmissivo, i singoli host.

A seconda del tipo di trasmissione, inoltre, soltanto uno o alcuni end-system collegati alla dorsale possono ricevere/inviare. Si possono definire 3 *schemi di reindirizzamento*:

- **unicast** = la sorgente trasmette i dati dicendo che la destinazione è una e **una sola**. comunicazione tra 2 due entità, sorgente e destinazione.
- **broadcast**
- **multicast** = la sorgente trasmette un dato a cui viene allegata la informazione di controllo che il dato è destinato a un determinato **sottogruppo** di end-system tra quelli collegati. I sistemi che non sono membri del gruppo non la utilizzano, mentre quelli del gruppo ne hanno accesso (a differenza del broadcast dove tutti hanno accesso a tutto).

problema delle reti broadcast

Dato il mezzo trasmissivo condiviso bisogna comunicare uno alla volta, altrimenti si hanno delle **collisioni**, le quali possono compromettere l'informazione. Servono dei meccanismi di arbitraggio per risolvere la contesa nell'accesso al mezzo.
Oggi le reti broadcast sono costruite utilizzando degli **switch**, ovvero tratte di mezzo trasmissivo che controllano la propagazione di informazioni sul mezzo trasmissivo e controllano e gestiscono le contese (inoltre ci sono dispositivi che ricevono dati da più

stazioni e gli switch li immettono sulle tratte di mezzo trasmissivo in modo che non ci sia contesa). Altri apparati che evitano la contesa sono i **bridges**, in grado di connettere solamente due dispositivi tra di loro (hanno due interfacce di rete). Gli switch, invece, hanno più interfacce di rete.

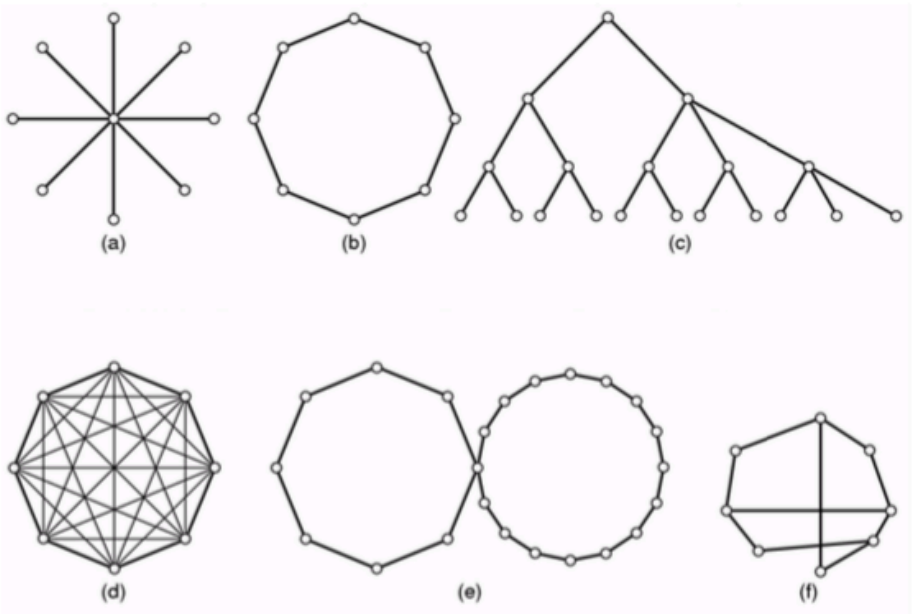
dominio di broadcast = è un insieme di host in una rete che possono scambiare dati a livello datalink, senza che questi debbano risalire fino al livello di rete in altri nodi dello stesso insieme, tali che se una di loro inviasse un messaggio in broadcast a livello datalink, sarebbe "ascoltata" da tutte le altre. Gli switch sono in grado di ricevere un'informazione da una **sorgente** con un **indirizzo broadcast** → deve essere inviato a tutti gli end-system collegati → **lo switch fa da broadcast**.

reti wireless = mezzo trasmissivo condiviso (onde radio che si propagano nelle tre dimensioni delle spazio), puramente broadcast.

point-to-point

Punto-punto (p2p) (solo nel caso con router).
//NB: attenzione a non confondere con P2P, peer-to-peer.

- Ogni tratta che collega i vari end-system costituisce una rete a se.
- Il router ha il compito di ricevere/trasmettere i dati.
- il dominio di broadcast in questo caso è la rete costituita da 2 apparati direttamente connessi da una tratta di mezzo trasmissivo.



topologie di reti p2p:

(a) centro stella = end-system con al centro un router, con il compito di ricevere segnali e capire la destinazione dei dati da trasmettere.
(b) anello = l'informazione fa il giro di tutto l'anello. In questo tipo di rete però ogni vertice decide se tenere l'informazione, trasmetterla al mezzo successivo. Ogni

segmento di questo ottagono è una rete a sé che contiene esclusivamente due apparati connessi ai lati.

(c) albero
(d) maglia completa = ogni apparato ha tante interfacce di rete (collegamenti) quanto è il numero degli altri apparati con cui ha un collegamento esclusivo. Per ogni coppia c'è un cavo-tratta riservato. Ovvero tutti i punti sono connessi, tutte le stazioni.
(e) 2 strutture ad anello connesse tra di loro
(f) maglia parziale = non a maglia completa, struttura più frequente (struttura di internet)

in generale le **reti "piccole" sono broadcast,** quelle "**grandi"sono p2p**.

fare routing = Letteralmente "fare instradamento", significa cercare un cammino all'interno della topologia di rete che permetta di portare un dato, attraverso l'aiuto di altri apparati intermedi, da una sorgente a una destinazione.
hop = passaggio di un messaggio da un apparato a un'altro.

come stabilire il cammino migliore ?

- numero di hop
- tempo in cui i dati vengono messi a disposizione = ritardo
- capacità dei link (tratta che collega due o più router)

I router hanno anche il compito di trovare il cammino migliore in funzione di quello che il traffico dati ha bisogno, con le cablature che hanno a disposizione, oltre al computare dati.

routing table = tabella di instradamento, in cui è ricordata per ogni destinazione nella rete, qual è la migliore interfaccia di uscita.

infrastrutture di rete cablate esistenti

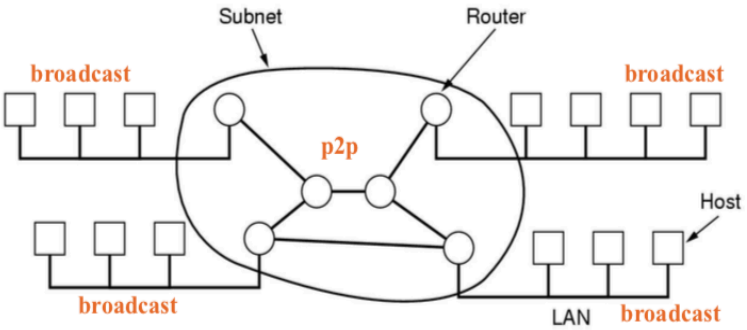
- **PSTN** = Public Switched Telephone Network; rete telefonica.
- **ISDN** = Integrated Services Digital Network; rete a larga banda, evoluzione della rete telefonica.

Queste reti connettono utenze domestiche a **ISP** = Internet Service Provider.

LAN

= Local Area Network.

- reti di taglia piccola (room/campus)
- sotto un unico amministratore di rete.
- broadcast.
- tempo calcolabile di trasmissione.
- non ci sono router.



intranet

= una collezione di LAN collegati tra loro con dei router. Un router può essere connesso a LAN o altri router connessi a LAN.

- collegamenti punto-punto con delle aree broadcast alla periferia.
- comuni ad una determinata azienda/campus.
- sotto un unico amministratore di rete, cioè un'unica autorità che impone le stesse politiche di instradamento/sicurezza/qualità su tutti gli apparati di rete che costituiscono la intranet.

WAN

= Wide Area Network.

- dimensione metropolitana/regionale/nazionale.
- rete punto-punto
- più amministratori perché diverse aree sono gestite da ISP diversi.

internet

= collezione di LAN e WAN eterogenee

- interconnessione di reti
- più amministratori
- nazione

Internet (i maiuscola)

= unica. interconnette tutte le reti del mondo.

unità di misura

il segnale è trasportato da **onde elettromagnetiche** (segnale elettrico), in particolare onde con una forma **sinusoidale** e **periodiche**, che si ripetono con una determinata frequenza. Periodo= T, frequenza= 1/T.
Per rappresentare 0 e 1 servono 2 onde. possono variare :

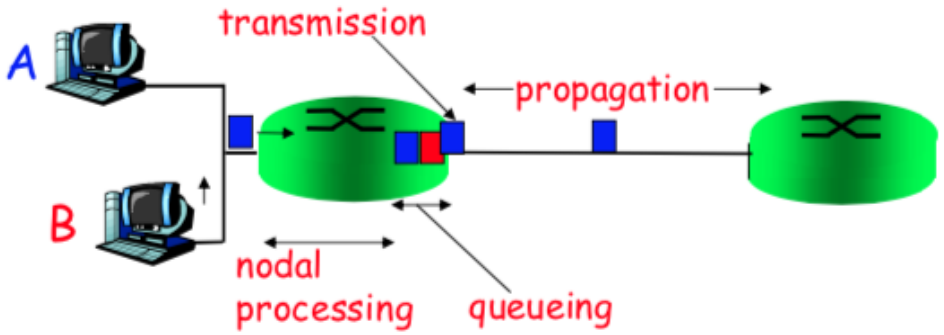
- in ampiezza (= M)
- in frequenza (diverse frequenze → diverse informazioni o diversi apparati con cui comunicare)

banda = indica lo spettro di frequenza utilizzate per la trasmissione di segnale elettromagnetico che codifica l'informazione, viene misurata in Hz (= numero di vibrazioni per secondo).
bit rate = numero di bit trasmissibili su un canale per unità di tempo. bps = bit per second mentre Bps = Byte per second.
baud rate = numero di volte per secondo in cui la scheda di rete riesce a cambiare il segnale elettromagnetico trasmesso sul canale. (cambiare la informazione emessa).
bit error rate = BER, probabilità di errore nell'interpretazione di un'onda. (es. rumore bianco)

parametri caratterizzanti
//velocità di propagazione =

- **in aria o fibra ottica = $3x10^8$ m/s = $3x10^5$ km/s**
- **in rame = $2x10^8$ m/s = $2x10^5$ km/s**

(importante per gli esercizi)



- **ritardo di trasmissione Tx** = quantità di dati da trasmettere / bit rate
- **ritardo di propagazione Tp** = distanza apparati / velocità di propagazione (dipende dalla natura del mezzo : rame, fibra ottica, wireless)
- **capacità del link** = bitrate x ritardo di propagazione Tp.
- **ritardo di elaborazione** = dipende da quanto è **veloce** il router.
- **ritardo di accodamento** = dipende dalla **quantità di lavoro accumulata sugli apparati di rete.** Ha un effetto valanga in alcuni casi, infatti il ritardo in coda cresce esponenzialmente con l'approssimarsi dell'**intensità di traffico** a 1. Si può creare del **traffico bursty** = (traffico distorto) i dati partono regolari ma arrivano in modo irregolare.

tassonomia dati

- testo = un tempo unico tipo di dato scambiabile
- immagini = rappresentare luminosità e colore e in funzione della qualità della img possiamo avere una risoluzione alta (codifica SVGA)
- video = sequenza di immagini (frame). 2 tipi di codifica : CBR(constant) e VBR(variable, alcuni frame descrivono solo cosa è cambiato da quello precedente → fa statistical multiplexing)
- audio

Alcuni tipi di traffico, che non sono solo testo, hanno delle richieste molto elevate da fare alla rete → bisogna fornire un buon servizio.

Quality of Service

➔ **streaming** = ha assoluta sensibilità al ritardo dei dati, che sono generati e consumati con continuità. Nelle applicazioni interattive (es.: Zoom) si considera il **round trip delay** = ritardo di viaggio di andata e ritorno.

➔ parametri di QoS :

- dimensione massima del package da sorgente
- affidabilità = numero/percentuale di pkt consegnati
- latenza = tempo che intercorre tra l'invio dei dati dalla sorgente all'arrivo, metà del round trip delay
- **jitter** (regolarità di flusso) = varianza dell'inter-arrivo dei pkt (il ritardo di inter-arrivo tra 2 pacchetti è la differenza di tempo di arrivo del primo bit del primo pacchetto al primo bit del secondo pacchetto).
- **throughput** = (qtà dati a destinazione / unità di tempo). Il suo valore viene controllato dal canale più stretto che c'è in rete, è determinato dal bottleneck link.

Si possono considerare 3 tipi di throughput:

- istantaneo = rate in un certo punto del tempo
- average = calcolo medio su tutta la trasmissione di una sequenza di dati
- goodput = throughput senza considerare le informazioni di controllo che la rete può aver aggiunto per funzionare.

regole di trasferimento dati

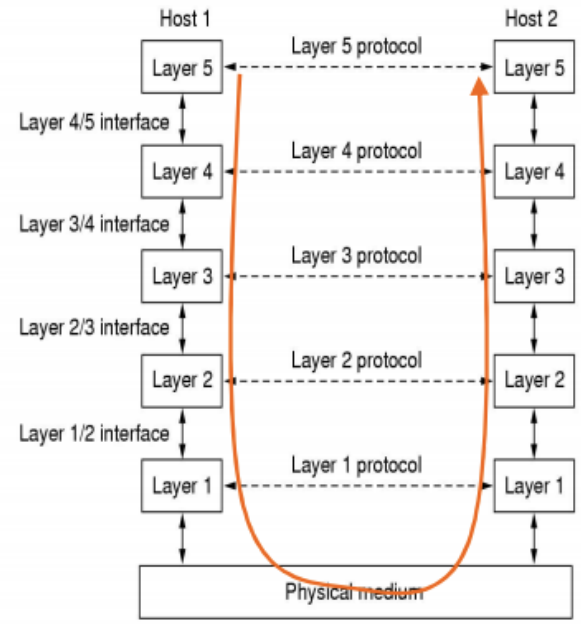
- simplex = dati viaggiano in una sola direzione
- half-duplex = i dati viaggiano alternativamente nelle 2 direzioni (senso unico alternato)
- full-duplex = i dati viaggiano in entrambe le direzione

la rete

come è fatta una rete?

è una architettura a livelli. dal basso verso l'alto:

- **hardware** = il mezzo fisico del trasporto dei dati (cavo in rame, fibra, etc...) → trasformare l'informazione digitale in un segnale elettrico.
- **layer di estrazione** che lavorano con l'HW.
- **layer di estrazione** che lavorano con le applicazioni.



stack di protocolli

servizi = ciò che c'è, su una stessa macchina, tra un livello e quello sottostante/sovrastante (**verticali**) → tutte le **primitive**, comandi che un livello mette a disposizione a un'altro livello
protocolli = dialogo tra 2 entità di pari livello residenti su macchine diverse (host sorgente- host destinazione, 2 router connessi da un pezzo di filo) (**orizzontali**) → insieme di regole che dice quali informazioni si scambiano e in che formato. Servono perchè due macchine possono essere diverse in termini di sistema operativo, architettura interna.
algoritmi = sono quello che c'è all'interno di ogni livello e servono per implementare i servizi forniti al livello superiore e i protocolli.

servizi + protocolli + algoritmi = **architettura**

comunicazione verticale

Quando un livello (n+1) deve passare dei dati (**SDU = Service Data Unit**) a un livello inferiore (n) per chiedere che vengano inviati dall'altra parte, vengono trasmessi attraverso uno dei possibili **punti di accesso a servizi** (es.: servizio best effort, QoS tipo A, etc…) che il livello n mette a disposizione a livello n+1.
Durante il passaggio al livello superiore deve dire a quello inferiore cosa fare di questi dati → ciò avviene grazie a delle **informazioni di controllo** (**ICI = Interface Control Information**). Queste hanno visibilità soltanto all'interno dell'apparato che stiamo considerando.
Una volta passati SDU e ICI da un livello all'altro vengono separati e il livello che attualmente li contiene implementa il proprio ICI → viene aggiunto un **header** (intestazione).
PDU = (**Protocol Data Unit di livello n**) dati del livello n+1 **+** header che implementa il protocollo del livello n.

Service Access Point = interfaccia che permette a un livello superiore di accedere ai servizi di un livello inferiore.

comunicazione orizzontale

/*ripasso di Sistemi Operativi

memoria paginata = la memoria viene divisa in pagine di uguale dimensione all'interno dei quali vengono scritti i dati. Facile da usare.

memoria segmentata = si divide dinamicamente la memoria in segmenti della dimensione che serve per i dati che devono essere allocati. Esistono 3 algoritmi per la gestione dei segmenti : Best Fit, First Fit e Worst Fit.*/

Negli apparati di rete si usa la **memoria paginata**. Ciò implica che esiste uno standard che impone, per tutti i dati, una **dimensione massima prefissata** (es.: 64 kB). Spesso nei livelli più alti arrivano quantità di dati più grandi di una pagina → si **frammentano i dati** in più pagine.

payload = tutto ciò che **non** è header di livello n.

L'architettura che sta appena sopra il mezzo trasmissivo è composta da :

- **tail** = unica del livello dell'architettura che sta appena sopra il mezzo trasmissivo. Particolare sequenza di bit. Serve per la comunicazione con l'altra parte. Comunica che i dati sono finiti e che d'ora in poi riceverà la portante.
- **header** = i primi bit sono una sequenza particolare che viene riconosciuta dell'entità di pari livello destinataria.
- **la portante del segnale** = onda sinusoidale, si attiva quando la scheda di rete è accesa. Per portare informazioni bisogna modulare l'onda in ampiezza e in frequenza. Si può anche fare una modulazione di fase → introdurre delle discontinuità nell'onda spostandola di un certo angolo.

funzionalità dei livelli

convenzione naming e addressing = non sono uguali a tutti i livelli. Indica con chi si vuole comunicare (es.: web server ateneo = indirizzo).

controllo di flusso = (orizzontale) l'entità mittente deve regolare la velocità con cui manda dati in modo da non sovraccaricare l'entità di pari livello.

controllo di congestione = (verticale) l'entità mittente deve regolare la velocità con cui manda dati in modo da non sovraccaricare l'entità sottostante.

multiplexing = significa prendere dati da tante sorgenti che stanno a livello superiore e "**incapsularli**" su un unico flusso di dati uscente al livello inferiore.

(de)multiplexing = inverso di multiplexing. Arrivano molte informazioni da livello inferiore → bisogna smistarle su diverse code per far arrivare i dati a diversi applicazioni destinatarie al livello superiore.

Si effettua la creazione di:

- **sessioni** = regola il trasferimento di diversi flussi di dati a destinazione.
- **connessioni** = regola il trasferimento di tutti i dati correlati a destinazione.

connection-oriented service

= si divide in tre fasi :

1. **handshaking** = apertura connessione, sorgente e destinazione vengono messe in relazione, serve per costruire la rotta dove viaggiano le informazioni. I terminali si mettono d'accordo su come scambiare i dati (in modo che il servizio sia orientato alla connessione) → informazioni puramente di controllo.
2. **scambio dati**
3. **chiusura connessione** = informazioni di controllo.

la caratteristica legata al servizio orientato alla connessione è **l'ordinamento**:

- ➔ la granularità di ordinamento (ovvero quanto grande è la porzione di dati da consegnare in ordine) può essere su una porzione di dati o byte per byte.
- ➔ **potrebbe** garantire anche **affidabilità** (non è una caratteristica del servizio orientato a connessione) → oltre a essere dati ordinati sono tutti presenti.
- ➔ è orientato al messaggio.

connectionless service

- ➔ non garantisce l'ordine.
- ➔ gestito indipendentemente, cioè non c'è alcuna relazione tra i dati che una sorgente manda a una destinazione.
- ➔ non esistono fasi di apertura/chiusura del servizio → comunicazione immediata
- ➔ meno **overhead** = sovraccarico di lavoro delle componenti di rete perchè non c'è lo scambio di informazioni di controllo.
- ➔ **acknowledged** = garantisce **affidabilità**. riconoscimento dell'arrivo del dato a destinazione.
- ➔ è orientato al byte.

tecniche di commutazione

*commutazione di **circuito**: come funzionano le reti sul cavo?*

sono reti p2p che hanno più interfacce rete per un router

- ➔ **non c'è ritardo di processing e di accodamento** e sicuramente arrivano in **ordine**
- ➔ **non è molto efficiente**, infatti dato che il **path è riservato** e vengono usati solo qualche kbyte (es.: per la codifica della voce), il resto della capacità del cavo è sprecata
- ➔ scarsa resistenza ai guasti.

i dati possono essere frammentati in:

- pacchetti
- messaggi (più resistenti ai guasti)

*commutazione di **pacchetto***

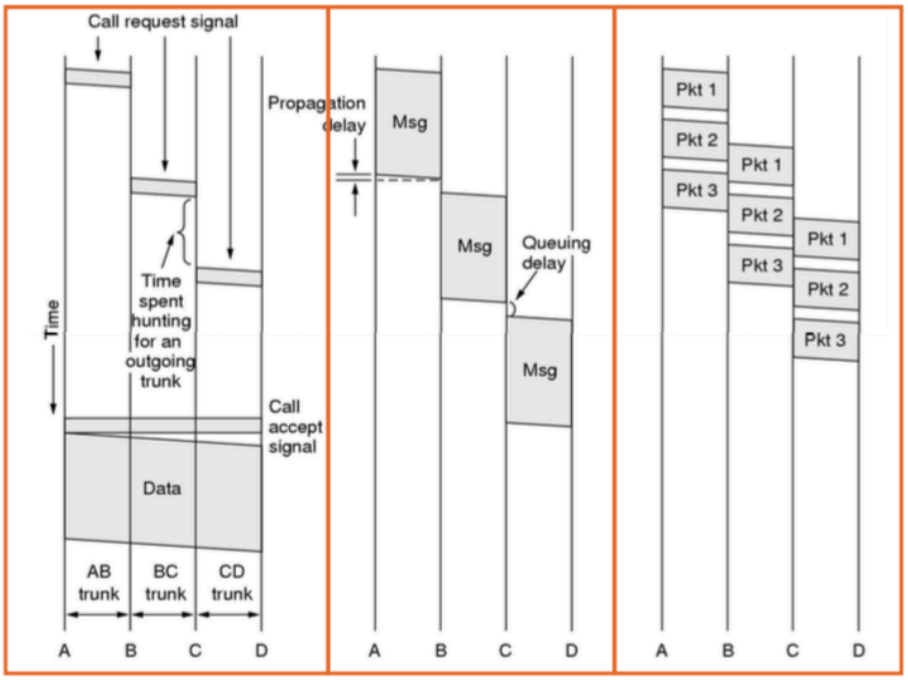
packet = i dati vengono rotti in pacchetti di dimensione massima **fissata** che dipende dalle caratteristiche della rete. Non esiste un cammino unico tra sorgente e destinazione, ogni pacchetto viene mandato sulla rotta migliore → tecnica **store-and-forward**. Ha in più il vantaggio che si può fare **pipelining**, cioè avere parallelismo. Questo riduce di molto la latenza.

Gli switch qui sono in grado di memorizzare i pacchetti, oltre che di inoltrarli. Il router possiede una coda dove possono essere memorizzati i pacchetti prima di essere inoltrati e si occupa di fare instradamento (store-and-forward).

*commutazione di **messaggio***

msg = insieme di dati con un significato per la sorgente. Hanno dimensione **arbitraria**. Vale la tecnica store-and-forward anche per i messaggi. Poco usata, di passaggio per arrivare alla commutazione di pacchetto.

confronto tecniche di commutazione (switching)



sono rettangoli "storti" a causa del tempo di propagazione del segnale nel mezzo trasmissivo da un dispositivo ad all'altro.

- **commutazione a circuito**, sull'asse temporale di B si può misurare il tempo che trascorre tra il momento in cui arriva l'ultima parte di informazione trasmessa da A a B e il momento in cui B rimanda avanti in direzione di D l'info ricevuta = **somma dei ritardi di accodamento+processing**. Una volta che il segnale di chiamata è stato accettato, il cammino da A a D può essere visto come un **filo unico** e **non** ci sarà più ritardi di accodamento e processing (solo un minimo ritardo di propagazione).
- c**ommutazione a messaggio**, i dati vengono inoltrati da un dispositivo a un altro
- **commutazione a pacchetto**, i pkt sono meglio dei msg perché hanno una dimensione massima fissata.

*tecniche di **instradamento***

routing a circuito virtuale
cerca di emulare il funzionamento della **commutazione di circuito.** La sorgente chiede al primo dispositivo che incontra di costruire un circuito per portare i suoi dati, che avranno uno specifico identificatore, a una destinazione. I circuiti virtuali che possono attraversare un apparato sono in numero molto inferiore a 2^{32}.
Prima che tutti i datagrammi di un messaggio possano essere inviati, è necessario impostare una connessione virtuale per definire il percorso. Non solo il datagramma deve contenere gli **indirizzi di sorgente e destinazione**, ma deve anche contenere un'etichetta di flusso, che identifichi il circuito virtuale che definisce il **percorso virtuale** che il datagramma deve seguire. Questo servizio si compone di tre fasi:

- setup (creazione del circuito virtuale);
- data transfer (trasferimento dei dati);
- teardown (chiusura del circuito virtuale).

routing a datagram
è la più usata ed è il default. Ogni pacchetto contiene un **indirizzo destinazione** ed è **instradato indipendentemente**. Per ogni pacchetto è scelta la routing table migliore al momento e, quindi, per diversi pacchetti appartenenti alla stessa sessione possono essere assegnati diversi path. Un router riceve un pacchetto, analizza lo header, e in base alla conoscenza che il router ha in quel momento della rete, decide a quale vicino inviare il pacchetto. Il nome datagram è usato perché simile a telegram (telegramma). L'arrivo dei dati e l'ordine non sono garantiti (poco affidabile).
//NB: attenzione a non confondere il servizio connection-oriented con la politica di commutazione per comunicazione circuit switching con le politiche di routing.

le architetture di rete standard

La **standardizzazione** è importante per la comunicazione degli apparati di rete poiché questi possono essere molto diversi tra loro. Ciò avviene grazie ai protocolli, che regolano la modalità di comunicazione tra apparati differenti. La standardizzazione è di protocolli (quindi è orizzontale, non verticale). Il firmware e il sistema operativo, infatti, si occupano dei servizi. La standardizzazione è operata da un'organizzazione super partes, ovvero organismi internazionali che la gestiscono.

ISO
= International Standard Organization, certifica la procedura per produrre un determinato progetto/servizio. Non certifica il livello di qualità ma il processo produttivo. Non è specifico per le reti ed è fondamentale per lavorare in ambito internazionale.

ITU-T
= International Telecommunication Union. Determina e standardizza l'hardware, ovvero come devono essere fatti i vari strumenti e di che materiali (le componenti della rete, cavi, frequenze, antenne...).

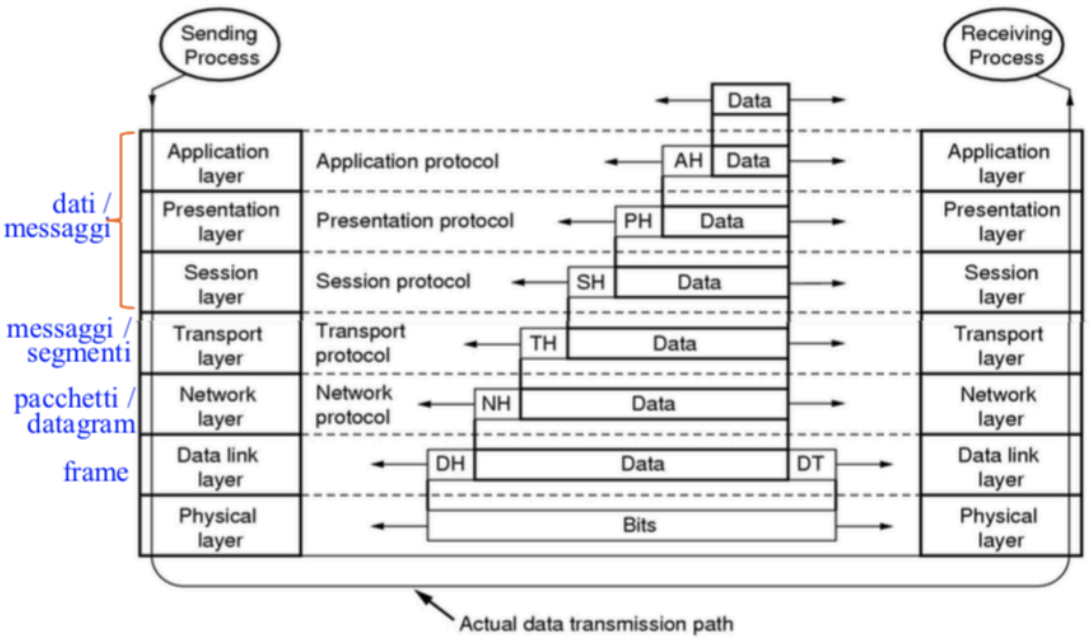
IETF
= Internet Engineering Task Force, è organizzata in gruppi di lavoro per risolvere problemi specifici legati alla realizzazione di internet. Nei working groups può partecipare chiunque e sono svariati, con diversi obiettivi e funzioni. IRTF (=Internet Research Task Force) nasce quando si nota una problematica e svolge un lavoro preliminare, ovvero verifica la portata del problema (anche a livello globale) e cerca se ci sono già soluzioni. Se il problema persiste, allora, nasce un IETF. I documenti che parlano del problema e soluzione sono gli "Internet Draft". Anche qui la gestione è aperta a tutta la comunità. Dopo 6 mesi la bozza termina. Può anche ripartire una nuova bozza. Altrimenti si arriva a un documento standard che conclude la faccenda e porta a una soluzione approvata (RFC).

IEEE
= Institute of Electrical and Electronic Engineers. Anche qui organizzata in working groups, non molto aperta. Ethernet, ad esempio, è un prodotto commerciale conforme allo standard IEEE. Conforme allo standard vuol dire due cose: gli standard devono essere scritti in modo chiaro e si può commercializzare un prodotto solo se conforme a quel determinato standard internazionale.

Il principio è *Divide et Impera*. In questo modo ogni livello più piccolo è più facile da gestire. Tuttavia, come già osservato nell'incapsulamento, ciò può creare diverse problematiche (informazioni di controllo, header, tail…).

Nella storia dei modelli standard ne sono stati pensati due:

- ➢ **OSI** = Open System Interconnection, pensato da ISO. Costituito da 7 livelli:
  - **7. livello applicazione.** protocolli che danno supporto alle applicazioni. Es.: un browser come Chrome quando si connette alla rete deve usare un protocollo (HTTP).
  - **6. livello di presentazione**. (sicurezza e privacy) ha come obiettivo trasformare i dati forniti dalle applicazioni in un formato standardizzato e offrire servizi di comunicazione comuni, come la crittografia, la compressione del testo e la riformattazione. Prima dei problemi legati alla sicurezza, in questo livello si operava il **marshaling** = modo di manipolare i dati in modo che siano comprensibili dall'altra parte. Ciò è legato alla macchina, poiché diverse macchine operano con delle differenze.
  - **5. livello di sessione**. gestisce le sessioni. Ovvero diversi flussi di traffico che vanno messi insieme. (Es.: sincronizzare la partita di calcio con il commentatore). Qui ci sono tutte le funzioni, tipicamente marcature temporali, che si occupano di gestire le sessioni (ovvero flussi di traffico differenti), ma che devono arrivare a sessione tutte insieme.
  - **4. livello trasporto.** primo livello in cui la comunicazione è end-to-end. La comunicazione è da end-system sorgente a end-system destinatario. E' una coperta che nasconde tutte le problematiche che ci sono sotto la rete. Svolge multiplexing: i dati che arrivano da diverse app sovrastanti vengono incastra nell'unica uscita di rete attiva. Inoltre fa demultiplexing (il contrario) e, quindi, fa naming. Sa distinguere le varie applicazioni che ci stanno sopra (sa da chi arrivano e a chi vanno i dati...). Fa controllo dell'errore. Vengono aggiunti bit di ridondanza per verificare se quello che si sta mandando è corretto. Non fa correzione di errore, nel senso che se c'è allora bisogna buttare via il dato. Infine controllo di flusso e di congestione dei dati (quantità di dati ecc…)
  - **3. livello network**. comprende le funzionalità che sono svolte dai router, che fanno connessioni punto punto. Si vede la topologia della rete e si cerca il cammino e si traduce da protocollo in uso dall'apparato a protocollo destinazione.
  - **2. livello data link**. appena sopra il filo; oltre allo header si aggiunge la tail. Ci si occupa di affidabilità, controlla se ci sono frame persi o danneggiati. Nella tail ci sono anche bit di ridondanza che fanno controllo errore. C'è anche controllo di flusso. (Si noti la ridondanza di funzioni con il livello quattro.) Se qualcosa va storto, è meglio accorgersene a questo livello piuttosto che il livello quattro. Qui c'è una vicinanza a livello geografico. Eppure il controllo di errore c'è anche a livello quattro per il fatto che magari l'errore può essere grande. **MAC** (medium access) gestisce le collisioni nella contesa di broadcast.
  - **1. livello fisico**. cioè il cavo. C'è lo hub, che fa amplificazione (anche pulizia) e broadcast a livello di segnale. Oggi si usano molto più gli switch.

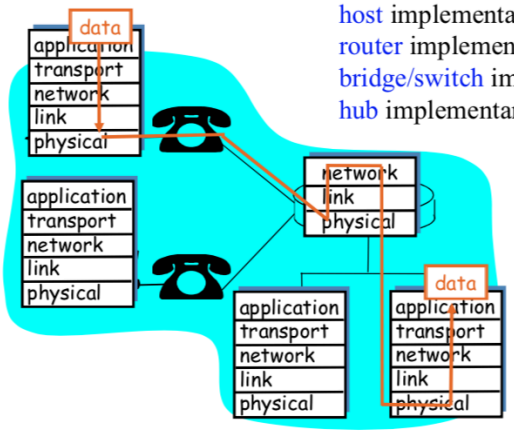


➢ **TCP/IP** =

- **5. livello applicazione.** protocolli per collegarsi ad una macchina remota, trasferimento di file, protocollo che è alla base dei comandi shell oggi. Traduce le stringhe in numeri che sono gli indirizzi della rete: **DNS** (Domain Name System). Ad esempio traduce “www” o “http”.
- **4. livello trasporto** : qui ci sono 2 protocolli:
  - **TCP** = garantisce che i dati arrivino nell’ordine in cui sono stati inviati, crea affidabilità, frammenta i dati (segmenti) e implementa il controllo di congestione e di flusso.
  - **UDP** = qui i dati si chiamano messaggi, è connectionless e best effort (non garantisce affidabilità).
- **3. livello network.** best-effort. Anche qui un servizio senza connessione e affidabilità. Ingloba le funzionalità di instradamento e controllo di congestione.
- **2,1.** Qui il livello 2 e 1 sono uniti generalmente nel **livello Host**

L’OSI è più recente del TCP/IP, quindi una sostituzione sarebbe costata cara; alcuni livelli nell’OSI, inoltre, non furono completamente definiti. Non riuscì a convincere della propria validità rispetto a TCP/IP.
OSI, tuttavia, è indipendente da protocolli e implementazioni, quindi è molto malleabile e adattabile. Nel tempo non è stato usato molto, anche se oggi sta venendo riconsiderato.

Sullo host o end-system ci sono tutti e sette i livelli. I router implementano i livelli dall’uno al tre; i bridges e gli



switches dall'uno al due; gli hub solo il primo.
Oggi le prime tre funzionalità di OSI sono implementate sulle schede di rete.

Accesso alla rete:

- **residenziale** = accedono ad un point-of-presence (PoP, punto di accesso di un ISP). Gli ISP sono organizzati con una struttura gerarchica.
- **aziendale** = non c'è un utente domestico ma un'organizzazione, azienda o università che ha la propria rete nelle proprie sedi e che si interconnette con il resto del mondo attraverso le dorsali fornite dall'ISP.

Come sono fatti gli ISP?

Ce ne sono di un primo livello, quelli internazionali, caratterizzati dal fatto che i POP (point of presence) sono collegati in modo diretto (cammino diretto tra un POP e l'altro). Gli ISP di livello 1 formano una *maglia completa.*
Sempre attraverso i POP ci si aggancia a ISP di livello nazionale, ovvero il secondo livello. Gli ISP di livello 3 sono regionali, locali. Questi ultimi devono passare attraverso i livelli 2. Gli ISP di livello 2 e 3 non formano maglie complete. Qui siamo nell'unità di misura dei Tbps.
I *Peering Point* sono i punti grazie ai quali possono comunicare le *dorsali.*

Terminologia:

mobile = l'applicazione o l'utente si possono connettere alla rete in punti diversi col variare del **tempo**.
nomadic = wireless network + mobile user
wireless = (tipicamente broadcast) fa riferimento alle caratteristiche del mezzo trasmissivo → è un'onda radio non guidata o infrarossi, e si diffonde in tutte e **tre le dimensioni**. assume le caratteristiche delle reti broadcast. Il bit-error-rate è molto più elevato rispetto alle reti wired, è un mezzo trasmissivo **fragile**. le reti wireless si classificano in:

- **single-hop** = tipico della telefonia cellulare. Due utenti possono comunicare attraverso il segnale di una stessa antenna del provider di servizi di rete (i due utenti non possono comunicare direttamente). C'è una triangolazione.
- **multi-hop (W-LAN)** = potrebbero addirittura non esserci antenne, cioè due utenti nel reciproco raggio radio possono direttamente comunicare (simile a bluetooth).

Le antenne sono connesse tra loro via una rete wired, e gli utenti sono nomadici. La potenza trasmissiva di una antenna varia a seconda del numero di dispositivi connessi ad essa.
handoff = quando un utente si sposta troppo e il segnale passa da un'antenna/cella all'altra. L'antenna vecchia comunica con quella nuova per trasmettere le operazioni che stava eseguendo l'utente.
bluetooth = distanza ristretta (10-30 metri), legato alla frequenza del segnale e alla frequenza trasmissiva dell'antenna
Wi-Fi = distanza di 100-150 metri

collegamento dispositivi

Una stazione, quando si accende, inizia a mandare un segnale detto **beacon** e alterna il proprio funzionamento tra periodi in cui manda il beacon per annunciare la propria presenza e periodi in cui ascolta i possibili segnali mandati da altri dispositivi. Ciò può portare alla associazione tra dispositivi che negoziano su come usare il canale, quali frequenze usare per la comunicazione, politiche di accesso.

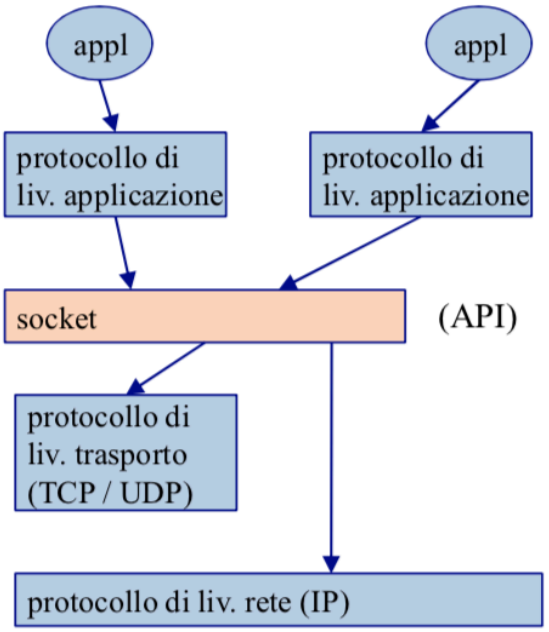
parametri delle reti Wi-Fi

- **BSSID** = parametro numerico che dà un indirizzo di rete dello access point. Può essere l'indirizzo di rete che ha iniziato la costruzione della rete (sono scritti nel firmware della scheda di rete → lvl 2).
- **SSID** = è una stringa con nome della rete.

livello applicazione: protocolli HTTP

//in questo corso si fa riferimento alla struttura ISO/OSI dove il livello applicazione è il livello più alto e fa già parte dell'architettura di rete (ovvero della parte del sistema operativo che si occupa di far funzionare la rete).

Bisogna distinguere tra :

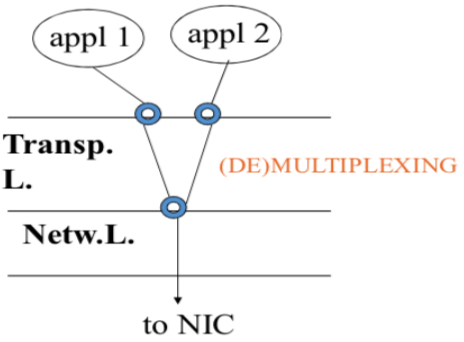


- **user agent** = (applicazioni propriamente dette) sono quelle che utilizziamo noi utenti → applicazioni utente in esecuzione sugli end-system in spazio utente (con privilegi di utente) e per funzionare hanno bisogno di utilizzare i servizi di rete (implementare il protocollo dell'applicazione). Non fa parte dello stack dell'architettura di rete.
- **protocolli di livello applicazione** = regola la comunicazione tra le entità di livello 7 (il più vicino all'utente) di end-system differenti. Può essere un "pezzo" di applicazione. Hanno bisogno di accedere ai service-access-point dei livelli sottostanti tramite le socket**.**
- **socket** = librerie che mettono a disposizione dei **comandi** che permettono di richiedere i servizi dei protocolli di liv. trasporto, e modificarne il comportamento (decidono se al trasporto ci sarà un servizio connection-less/oriented). Con le socket, inoltre, si può accedere direttamente dal livello applicazione al livello rete (perdo controllo di flusso e di congestione e dell'errore). Posso farlo grazie a socket raw, accessibili solo ad applicazioni di proprietà dell'utente amministratore di sistema.
  Le socket possono fare **multiplexing** sullo stesso host, usando 2 componenti di indirizzo di rete :
  - **nome** o indirizzo della specifica macchina su cui l'applicazione sorgente/destinataria dei dati è in esecuzione
  - **numero di porta** = interi corti senza segno su 16 bit, utilizzati per identificare una particolare connessione di trasporto tra quelle al momento attive su un calcolatore. In base al range cambia il significato :
    - 0-255 well-known per servizi standard. (web, mail, etc..) (standard ITF e IEEE)
    - 256-1023 per servizi in test (oggi c'è qualche servizio standard, perchè non bastano 256 per i well-known)
    - 1024-49151 Registered. Non standard, forniti da aziende. Ad esempio i servizi di Cloud, sono proprietari e non ci sono documenti che dicono come siano fatti all'interno.
    - >49151 effimere/dinamiche. Servono a sviluppatori, in cui non si usa la rete. Oppure vengono usate regolarmente da processi client per

comunicare in rete; nel momento in cui si chiude il processo la porta torna libera.

- **protocollo di livello trasporto** = svolge funzionalità di controllo dell'errore; possono essere di due tipi :
  - **TCP** → orientato alla connessione, garantisce che i dati arrivino a destinazione e che siano in ordine.
  - **UDP** → è un servizio connectionless, non garantisce l'ordinamento ed è inaffidabile.
- **protocollo di livello rete** = è sconsigliato passare direttamente da socket a protocolli di livello rete.

indirizzamento e (de)multiplexing



Ci sono 2 applicazioni in esecuzione sullo stesso host, entrambe devono generare dei dati e quindi chiedono servizio al livello di trasporto sfruttando i service access point.
Il livello di trasporto aggiunge ai dati di queste applicazioni il suo indirizzo di trasporto, (i dati delle applicazioni risiedono allo stesso indirizzo di rete), cioè i **numeri di porta.**
In seguito inoltra i dati al livello rete che aggiunge ai dati l'**indirizzo di rete** (partiti tutti da un unico host).
I servizi standard sono elencati nella RFC (Request For Comment). Secondo lo standard, per accedere a servizi web bisogna utilizzare la porta nota 80. La procedura per standardizzare i processi qual è? Per IETF vengono prodotte Internet Draft, esaminate dalla comunità e discusse; ad un certo punto si fa un prototipo (numeri di porta tra 256-1023).

come si distinguono le due applicazioni?
le applicazioni in esecuzione sono dei processi e hanno perciò un identificatore unico : il process ID = **PID**. Questa distinzione va bene solo in locale. Da remoto le applicazioni di distinguono per i **numeri di porta** diversi.

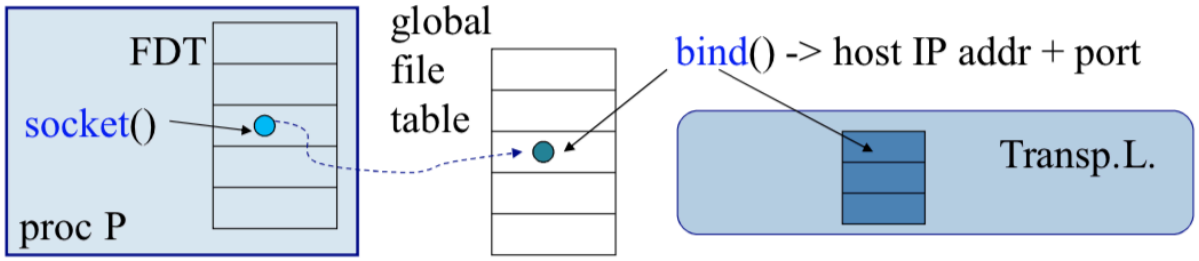
come si usano le socket ?
Le socket sono librerie di strumenti software (sia in java che in C) per accedere alle **funzionalità del livello trasporto** e si dicono API nel linguaggio di windows o syscall nel linguaggio di UNIX. La traduzione di socket è "presa elettrica", infatti è il punto di accesso per le applicazioni per accedere ai servizi forniti dal livello trasporto.
Queste API/syscall ci permettono di specificare la famiglia di protocolli che si vuole utilizzare, il tipo di servizio da chiedere e lo specifico protocollo.

- *socket(family, type, protocol)*: creazione end-point

La **socket()** alloca una entry nella **File Descriptor Table** (FDT) del **Process Control Block** (PCB) del processo, la entry (che diventa descrittore del socket) contiene un puntatore a

una struttura dati del sistema operativo chiamata **Global File Table** (GFT) che contiene tutti i dati per l'accesso a uno specifico file.
La successiva syscall/API è **bind()** che lega l'informazione dell'entry della GFT, puntata dell'identificatore di socket, con i buffer del kernel di sistema operativo che realizzano i servizi rete (legame con l'indirizzo del processo = l'indirizzo IP dell'host + il numero di porta).

- *bind(address)* a indirizzo noto all'esterno

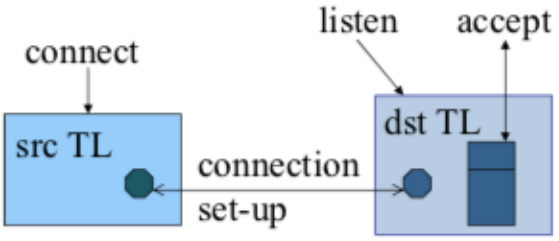


come si crea un canale di comunicazione usando un servizio connection oriented ?
Esiste una fase preliminare in cui i due processi che forniscono al livello superiore un servizio connection-oriented fanno **handshake**, negoziando i parametri per usare la connessione, prima del vero e proprio scambio di dati. La **connessione** è descritta da :

- associazione <proto, IPsrc, port-src, IPdst, port-dst>

ma in questo momento abbiamo solo il **protocollo** e determinato con la socket, **indirizzo** e **porta** del processo (src), determinato con la bind. La parte remota si stabilisce realizzando la connessione grazie a tre primitive TCP :

- **listen** = eseguito da chi mette a disposizione un servizio (server). Notifica al SO che non comunicherà per primo ma che è disponibile ad accettare richieste di connessione da parte di clienti. Il SO crea nel kernel un terzo buffer dove vengono salvate queste richieste.
- **connect** = un cliente specifica il server a cui si vuole connettere.
- **accept** = se non ci sono richieste il server si blocca su questa primitiva.



In seguito avviene il vero e proprio scambio di dati.
connection record = i parametri che sono stati negoziati vengono salvati dai livelli di trasporto connection-oriented di entrambe le parti.

primitive di servizio
Nel caso di **connectionless** le applicazioni per ogni socket hanno un buffer di invio e uno di ricezione. Quando un processo vuole mandare dei dati usa *sendto(msg, address)*. I dati sono presi da un buffer di caratteri del processo e copiati nel buffer di invio.
Per ricevere si usa *recvfrom(msg, address)* (i due parametri sono dei puntatori) è quasi simmetrico ma il processo è in stato di wait e in seguito passa a ready.

Nel caso di **connection oriented** si possono usare semplicemente *read(buffer)* e *write(buffer)*, perchè sappiamo già a che indirizzo inviare/ricevere.
Quando si chiude una connessione si usa *close()*.
Nb: i buffer ragionano sempre in modalità FIFO.

architettura client-server
Lato **server**:
- mette a disposizione delle risorse
- è in attesa su un indirizzo ben noto e dei clienti chiedano di usufruire dei suoi servizi
- controllo della sicurezza dell'accesso alle risorse tramite procedure di sicurezza del tipo AAA (= Authentication Authorization Accounting).
- deve gestire più clienti efficientemente e contemporaneamente → fa multitasking oppure ha a disposizione una server farm (insieme di server) →
    - **architettura cloud** = (es.: DropBox fornisce un servizio di memoria, anche condivisa) c'è un punto d'ingresso per le richieste dei clienti, ovvero un nodo server detto "broker" e manda le richieste al server opportuno.
    - **mirror** = lo smistamento delle richieste non è automatico ma la scelta è dell'utente in base al server più vicino geograficamente (i ritardi dipendono dagli hop → più è vicino meno hop).

Lato **client**:
- usa le risorse e quindi inizia la comunicazione
- può avere indirizzo qualsiasi
- se termina inaspettatamente il server deve continuare ad operare

architettura peer-to-peer (P2P)
Tutti i partecipanti coprono sia il ruolo di server che client (es.: torrent).
- il codice di un nodo è molto complesso perché ha la complessità del server
- l'infrastruttura è più resistente perchè un peer può chiedere servizio ad un altro peer → **graceful degradation** (più peer ci sono meno problemi si riscontrano). Il lavoro è distribuito e quindi throughput maggiore.

Ultimamente il traffico è diminuito perché è difficile trovare i peer che mettono a disposizione la risorsa desiderata.

condivisione file
approccio online :
- Network File System (NFS) = le modifiche sono immediatamente visibili a tutti ma ci possono essere problemi di concorrenza per l'accesso (si può verificare un bottleneck al server).

approccio offline :
- FTP, HTTP = non è richiesta la immediatezza e al massimo si dovrà fare il merge delle varie modifiche fatte su uno stesso contenuto.

World Wide Web
è il servizio on-demand che usa HTTP (e porta nota 80).
user agent = browser, applicazioni multiprotocollo: non c'è solo HTTP, ma anche FTP e altri protocolli per diverse operazioni. Il browser è, quindi, flessibile rispetto ai protocolli. Nel browser si indica cosa si vuole con una scritta del tipo : proto://hostname:port#/pathname. Questo tipo di scritta, che serve per l'indirizzamento, avviene tramite **URL (Uniform Resource Locator)**. URL è case sensitive, si può anche non specificare il pathname e si andrà ad accedere a public_html/index.html. Il browser è in grado di capire di che tipo è un contenuto (testo, multimedia, etc...) e usando plugin (helper application oppure external viewer) offre all'utente il contenuto nel modo migliore e più comprensibile.
Nb: il pathname nell'URL è relativo e non assoluto. Il servizio è on-demand o "pull". Il server non dà niente di sua spontanea volontà.

funzionamento di HTTP

Il browser (applicazione) ingloba una parte di codice che è il **client per il protocollo HTTP**. Il client tramite socket chiede servizi TCP/IP come protocollo di trasporto (quindi chiede servizio ad un determinato server web). La richiesta arriva al server che si è indicato nell'hostname e si arriva al file richiesto. *HTTP è un protocollo* ***stateless***: non ricorda le operazioni precedenti. Se il server non riesce a soddisfare le richieste diventa a sua volta client di altri server.
L'utente indica tramite il browser a quali contenuti vuole accedere. Il browser, per prima cosa, si rivolge a un **local resolver**, un servizio che cerca di tradurre le stringhe (www, http) in indirizzi numerici (livello di astrazione più basso) (in realtà il computer usa un unsigned int da 32 bit).
Per prima cosa cerca in un **file di corrispondenze locali**, al cui interno troviamo :

- l'indirizzo locale del pc
- l'indirizzo di rete associato alla scheda di rete del pc
- indirizzi dei server che vengono più cercati

Se il local resolver trova la corrispondenza [nome simbolico inserito dall'utente, indirizzo di rete] nel file i contenuti sono subito disponibili.
Se il local resolver non trova corrispondenza, allora il resolver **va in rete** a cercare questo valore. Utilizza il servizio di **UDP** che a sua volta fa uso di dati interni, come l'indirizzo di un server **DNS** (= Domain Name System) primario. I server DNS sono contattabili al numero di porta nota **53**. Il DNS, a fronte della richiesta del local resolver, consulta una base (locale in parte e distribuita in parte) di dati dando una risposta.
Una volta che il local resolver ha trovato l'indirizzo lo restituisce al browser, che sarà pronto alla navigazione.
Il browser passa al **client HTTP** l'indirizzo di rete appena scoperto e chiede di contattare il server con quell'indirizzo per recuperare il contenuto indicato nell'url messo dall'utente, utilizzando il servizio **TCP**. Si stabilisce una connessione tra client-http sull'end-system dell'utente e http-server che si vuole contattare, dato che TCP è connection-oriented.

formato del messaggio di **richiesta** HTTP

metodo = è il comando che il client manda al server, cioè gli dice cosa fare.
La richiesta ha il formato in quest'ordine :
1- riga di richiesta

- metodo
- URL
- versione http che si sta usando

2- righe di intestazione = (nome campo seguito dal valore) permettono di specificare delle informazioni aggiuntive come cookies e un elenco di cosa il browser client è capace di fare. Nelle righe di intestazione ci può essere il valore "*if modified since"*, in cui si fa una richiesta condizionale al server in cui la data di modifica nello header della copia del server deve essere successiva a quella mandata nella richiesta.
3 - <riga vuota>
4 - corpo entità = contiene le informazioni da inviare al server, viene usato dal metodo POST.
I metodi più utilizzati sono :

- GET <filename> → read file named (ma anche form «?»)
- HEAD <filename> → read header of webpage named
- PUT <filename> → write data to file named
- POST <filename> → append data to file named (form)
- DELETE <filename> → delete file named

formato del messaggio di **risposta** HTTP

La risposta ha il formato in quest'ordine :
1 - riga di stato :

- versione
- codice stato
- frase

2 - righe di intestazione (nome campo seguito dal valore).
3 - <riga vuota>
4 - corpo entità
I campi <codice stato> e <frase> sono delle informazioni che il server dà al client comunicando se è riuscito a portare a buon fine o meno la richiesta iniziale, a seconda del range di codice stato :

- 200-299 conferma la buona riuscita della richiesta
- 300-399 contenuto ricercato non si trova più sul server
- 400-499 la richiesta non va bene
- 500-509 errore da parte del server

richieste condizionali = un client può aggiungere una condizione alla richiesta. In questo caso il server invia la pagina richiesta solo se la condizione è soddisfatta, altrimenti informa il client della motivazione del mancato invio.

trasferimento di contenuti
Se più oggetti (i contenuti possono essere di diverso tipo : testo html, video, immagini, etc.) a cui si vuole accedere (es.: pagine Web) sono localizzati sullo stesso server, esistono due possibilità:

- recuperare ciascun oggetto usando una nuova connessione TCP = **connessioni non persistenti**
- recuperare tutti gli oggetti con un'unica connessione = **connessioni persistenti**

connessioni non persistenti

E' la connessione preferita dal client. Segue queste fasi:

1. il client apre una connessione TCP con il server, negoziando i parametri.
2. il client richiede al server un oggetto.
3. il server risponde con l'oggetto che torna indietro al client.
4. il client richiede al server di chiudere la connessione.
5. il client legge i dati e si riparte da 1.

Le connessioni non persistenti sono affette da **latenza** dato che per ogni oggetto bisogna fare setup della connessione, trasferimento dati e chiusura della connessione.
Il browser può usare più di una **connessione in parallelo** per poter trasferire contemporaneamente tutti gli oggetti all'interno di un documento. Anche in questo caso però può verificarsi della latenza e ci saranno molti messaggi di controllo per gestire la connessione (apertura+gestione oggetto+chiusura).
Si crea per ogni connessione un **connection record** che contiene tutte le informazioni che regolano lo scambio di info su una specifica connessione.

connessioni persistenti

La versione HTTP 1.1 prevede di default, a meno che il client non lo richieda specificatamente, l'uso di connessioni persistenti. In questo caso può succedere che :

- il client **non** chiude **esplicitamente** la connessione con quel server, ma si interrompe perché, per esempio, chiudendo il browser si accede a un altro server tramite link o perché se ne inserisce uno nuovo tramite l'url.
- se, dopo un certo tempo, non arriva una chiusura esplicita dal client allora la connessione viene automaticamente chiusa, sia da http-client che da http-server perché per un certo tempo non c'è stata alcuna attività su quella connessione. (time-out).

L'impiego di connessioni persistenti consente di **risparmiare tempo e risorse** (messaggi e info stato per diverse connessioni).

cookie

http è senza memoria (stateless). Si aggiunge **memoria** tramite il meccanismo dei cookies. Se i cookies sono abilitati allora fra le righe di intestazione della **risposta** del server può esserci un nome campo : "*Set_cookie*" seguito da un valore identificativo numerico *IDc* (identificatore numerico del client per il server).
Il database del server ricorda i dati del cliente associati a IDc (file scaricati, dati inseriti in form…) e, poi, l'applicazione (browser client) vede se il sito è già stato visitato e nel database dei cookies prende IDc e lo passa ad HTTP nei messaggi di richiesta (IDc inserito in richieste a successive interazioni col server).
come si generano i cookies ?

1. il server, quando riceve una richiesta, memorizza le informazioni relative al client in una stringa o in un file (es.: salva il nome di dominio del client, i contenuti di un cookie precedente, i dati inseriti in un form, etc...).
2. il server include il cookie nella risposta che invia al client;
3. il browser che riceve la risposta memorizza il cookie in una directory apposita.

Quando un client invia una richiesta ad un server, controlla nella **directory dei cookie** se c'è un cookie precedente e nel caso viene incluso nella richiesta; il server che la riceve capisce così che si tratta di un vecchio client.
-vantaggi: identificazione utenti, mantenimento stato, offerta personalizzata dei contenuti.
-svantaggi: violazioni privacy, data mining.

caching

Si può fare cache dei contenuti visitati dall'end-system dell'utente, in modo da facilitarne la fruizione = **caching locale**
ripasso di Sistemi Operativi
principio di località = se un processo sta utilizzando delle variabili/risorse a determinate locazioni di memoria del sistema, probabilmente per un certo tempo continuerà ad utilizzare i dati contenuti in quegli indirizzi.
Si è osservato che questo principio di località vale anche per le reti, ovvero, ci sono siti definiti **hotspot** perchè sono dei server che contengono informazioni che sono particolarmente accedute. In particolare vale per le persone di una stessa comunità (ad esempio un'azienda).
L'80% delle ricerche sul web riguarda il 20% dei contenuti.

proxy server

Questo ha portato ad ampliare l'architettura http con un'altra interazione con un **server http proxy** = è un **server a servizio di una determinata comunità**, scelto all'interno o vicino a un rete di tale comunità. Fa **caching** dei contenuti per tutta la comunità, cioè da qualunque end-system parta una richiesta per procurare un determinato contenuto web all'interno di una comunità, il proxy riceve quel contenuto e ne mantiene una copia per un certo periodo per tutti gli utenti.

L'http proxy ha come obiettivo :

- **ridurre la latenza** → non bisognerà aspettare molto tempo (rispetto al server originario) se il contenuto lo troviamo in locale o su un server vicino.
- **ridurre il traffico in rete** → non c'è bisogno di attraversare tutto Internet.

Come funziona un proxy server ?

- Tutti i browser di una comunità possono essere configurati per usare un http proxy che si trova nella rete della comunità o ai margini.
- Quando un browser genera una richiesta (parte un messaggio di http request) destinata al server che possiede quel contenuto, questa richiesta viene **incapsulata** una volta di più, in modo che il **destinatario** diventi l'indirizzo di rete del **server http proxy** che ha configurato.
- La richiesta arriva al http proxy che **controlla** nella propria cache se il proxy ha quel contenuto perché precedentemente :
    - lo ha cercato lo stesso utente (e lo ha portato lui stesso nella cache).
    - un'altro utente che si serve del medesimo proxy ha cercato lo stesso contenuto.
- Una volta trovato il contenuto lo fornisce come risposta.
- **Se il contenuto <u>non</u> si trova** il proxy non incapsula niente e fa partire la richiesta originaria http, come se l'avesse generata lui. La **risposta tornerà al proxy** aggiungendo il contenuto nella propria cache. Infine manda il contenuto al richiedente originario.

La definizione generica di proxy è: "entità che svolge lavoro al posto di qualcun altro in prossimità di chi ha bisogno di quel servizio". Attenzione a non confondere http proxy con il concetto più generale di proxy.

Si può trovare gerarchia di cache di livello :

- **basso** → una cache locale ad un end-system → riguarda i contenuti e gli interessi dell'utente che usa quello specifico end-system.
- **alto** → rappresentato dall'http proxy → serve un'intera comunità di più utenti. Ampiezza di memoria più elevata.

Ci sono due aspetti che determinano la durata di un contenuto in un cache :

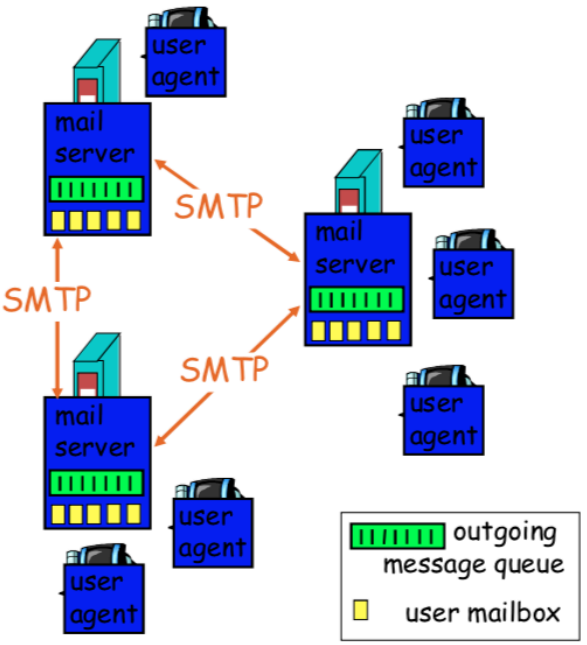
- l'ampiezza della cache in termini di memoria disponibile su disco. Quando la memoria è esaurita qualcosa viene eliminato seguendo politiche tipo FIFO o LFU.
- quanto è aggiornato il contenuto → permettere un update della pagina da parte del proxy solo dopo un certo intervallo di tempo dall'ultima modifica = **conditional get**

(Il proxy è server rispetto al browser e diventa cliente rispetto al server della richiesta originaria.)

Electronic Mail

Ci sono tre componenti dell'architettura di posta elettronica :

- **user agent** = è un software che risiede sull'end-system del client e permette di leggere/inviare mail e allegati. E' anche conosciuto come "mail reader" (es.: Outlook).
- **server di mail** = deve gestire i messaggi di posta in uscita da mandare ad altri server. Quando si manda una mail finisce al server di posta, non direttamente al destinatario utente. Quindi questi mail server hanno ruolo (come i proxy) sia di server sia di client. I mail server, di conseguenza, comunicano in coppia tra di loro, variando il loro ruolo a seconda che stiano inviando o ricevendo. Chi manda la mail ha il ruolo di client mentre fa da server vero è proprio chi riceve perché offre il servizio di ospitare il messaggi.

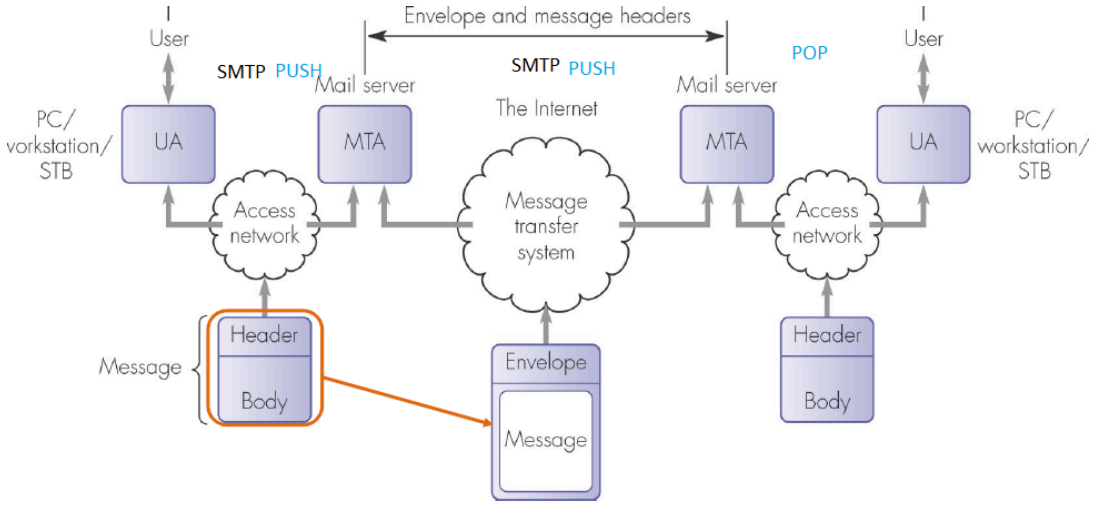


- **SMTP** = Simple Mail Transfer Protocol, cioè il protocollo di trasferimento di base. Realizza la comunicazione tra i server.

Il servizio di posta elettronica utilizza un servizio **persistente** e un protocollo a livello trasporto **TCP** → consegna ordinata e affidabile.

SMTP - Simple Mail Transfer Protocol

Tutti gli user agent (=UA) si interfacciano a uno **user agent server**, una parte di sw che risiede all'interno della applicazione che si sta usando per dialogare con mail server. Il messaggio composto dall'UA viene pushato via SMTP al mail server mittente (viene anche indicato come **MTA = mail transfer agent**). Il MTA mittente manda il messaggio al MTA che gestisce la mailbox destinataria dialogando via SMTP, aggiungendo delle proprie informazioni di controllo per il dialogo tra MTA, ovvero al livello applicazione che implementa il protocollo SMTP viene aggiunto un altro header che viene chiamato **envelope**.
Il MTA destinatario può esplodere un alias di email in un insieme di indirizzi, creando una copia di email per ognuno dei destinatari finali.
Il messaggio è composto da header (composto da valori chiave) e corpo.
SMTP trasferisce tutto come una sequenza di caratteri, riga per riga, e per indicare la fine di un messaggio usa il simbolo **CRLF.CRLF** (= Carriage Return and Line Feed, CR and LF are special characters, also referred to as \r\n, that are used to signify the End of Line (EOL). Significa avere un punto solo su tutta una riga).



SMTP command & response

Tutti i comandi sono lunghi **4 caratteri** perché non sono mai cambiati da quando sono stati creati, la banda disponibile non era molta e cercano di sfruttare al meglio il bitrate dei canali (non c'era bisogno di usare più di 4 caratteri).

I comandi sono :

- HELO → serve, una volta aperta la connessione TCP, per mandare il nome simbolico del MTA e quindi indicare quale **dominio** gestisce
- MAIL FROM → MTA client comunica al MTA server di avere una mail from e riferisce la mail del sender
- RCPT TO → MTA destinatario controlla che esista una mailbox con tale address
- DATA → dà inizio al trasferimento del messaggio
- QUIT
- RSET → c'è un'errore nella connessione e viene chiusa (sia SMTP che la TCP sottostante) e si riparte da capo

Le risposte sono :

- 220 → MTA ricevente è pronto a ricevere
- 221 → MTA ricevente è pronto a chiudere la connessione TCP
- 250 → svolto un comando con successo
- 354 → MTA del destinatario è pronto a ricevere il messaggio riga per riga
- 421/450/551 → risposte d'errore

```
Esempio di comando e risposte (S = MTA destinatario; C = MTA mittente) :
      S: 220 hamburger.edu
      C: HELO crepes.fr
      S: 250 Hello crepes.fr, pleased to meet you
      C: MAIL FROM: <alice@crepes.fr>
      S: 250 alice@crepes.fr... Sender ok
      C: RCPT TO: <bob@hamburger.edu>
      S: 250 bob@hamburger.edu ... Recipient ok
      C: DATA
      S: 354 Enter mail, end with "." on a line by itself
      C: Do you like ketchup?
      C: How about pickles?
      C: .
      S: 250 Message accepted for delivery
      C: QUIT
      S: 221 hamburger.edu closing connection
```

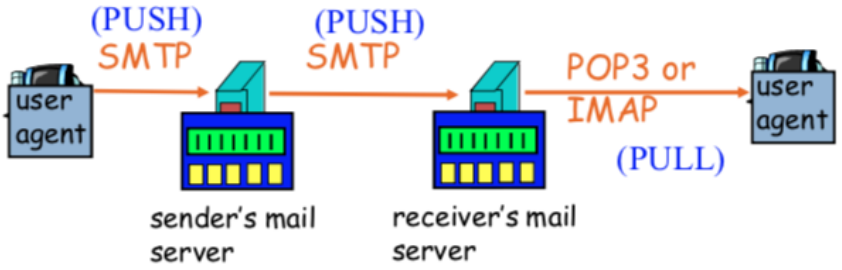
*Fasi della **consegna** di un messaggio di posta*

La consegna avviene in 3 fasi distinte (TCP e SMTP avvengono a livelli diversi della pila di protocolli):

1. **Apertura della connessione**: il MTA mittente apre connessione TCP con MTA destinatario e avviene la negoziazione per la fase di apertura della connessione TCP. Terminata, si apre la connessione SMTP e vengono scambiati ancora dati di controllo. Il server SMTP dà il via alla fase di apertura della connessione non appena il client effettua la connessione TCP con la porta nota 25.
   Inizia il dialogo tra i due MTA.
   Questa fase può essere suddivisa in 3 passi successivi:
   a. il server invia il codice 220 (service ready) o 421 (servizio non disponibile);
   b. il server si identifica tramite comando HELO seguito da nome del dominio;
   c. il server invia il codice 250 (comando richiesto completato) o altri codici a seconda della situazione particolare;
2. **Trasferimento del messaggio**: è l'invio dei dati veri e propri, attraverso un singolo messaggio dal MTA mittente a uno o più destinatari. L'MTA mittente passa tutto il buffer in attesa di invio e pesca tutti i messaggi per il dominio dell'MTA ricevente e glieli manda.
3. **Chiusura della connessione** : una volta che nella coda dell'MTA mittente non c'è più alcun messaggio si può chiudere la connessione. Dopodichè avviene anche la chiusura TCP sottostante.

mail access protocol

SMTP è un protocollo **push** infatti :



- push che viene utilizzato dallo UA per spingere i messaggi che devono essere spediti verso la MTA mittente.
- push da MTA mittente per spingere i messaggi a MTA destinatario.

Nell'ultimo passaggio si fa **pull** da parte dall'UA destinatario che chiede al proprio server se ci sono mail nuove. Questo perché lo **UA di un destinatario**, diversamente da un MTA che esiste sempre ed è sempre pronto in qualsiasi momento a ricevere un messaggio, sia che il cliente sia uno UA che un altro server, **non esiste sempre** e quindi MTA del destinatario **non può fare push**. Bisogna usare un approccio pull, quando l'utente destinatario vuole guardare la sua posta attiva lo UA e quest'ultimo chiederà il contenuto della propria mailbox dal proprio MTA di riferimento (in questo caso MTA viene chiamato **MAA = Message Access Agent**).

Quindi non si usa il SMTP ma 2 protocolli diversi:

- **pop3** → post office protocol. È molto semplice ma limitato. Comunica allo UA che è arrivato un messaggio e permette di fare:
  - download and delete → cancellare
  - download and keep → mantenere una copia
- **IMAP** → Internet Mail Access Protocol. È più complesso e permette più caratteristiche come mantenere dei folder di massaggi sul MAA e di scaricare solo lo header della mail o lo header e il corpo senza gli allegati (comodo per dispositivi con poche risorse). Supporta l'accesso utente da diversi host: non scarica le mail su host utente; POP3 fa lo stesso solo se configurato "download & keep".

come vengono gestiti gli allegati ? MIME

Multipurpose Internet Mail Extensions. E' un protocollo che serve per estendere il servizio fornito da SMTP per permettere il trasporto di altri tipi di dati. Aggiunge nella mail, sia nello header che nel corpo, delle informazioni di controllo per dialogare con il MIME a destinazione come:

- **MIME-version** = indica la versione di MIME.
- **Content-Type** = definisce il tipo di dati nel corpo.
- **Content-Transfer-Encoding** = definisce il metodo utilizzato per la codifica del messaggio in binario per il trasporto.
- **Content-id** = individua in modo univoco una parte del messaggio in messaggi composti da più parti.

- **Content-Description** = indica se il corpo del messaggio contiene un'immagine, un file audio o video.
- **Content-Length** = numero di caratteri ascii utilizzati.

Se gli allegati sono più di uno, lo header MIME lo indica con *multipart/mixed*. Tutti gli allegati sono separati e descritti dai 5 campi visti prima.

Ci sono due codifiche :

- **quoted printable** = trasforma testo ASCII arbitrario in testo senza caratteri indesiderabili (es.: CRLF.CRLF).
- **base64** = codifica tutto ciò che non è ASCII su 7 bit. L'oggetto che deve essere codificato viene considerato come una sequenza di bit. I bit vengono spezzati in gruppi da 6 bit → si ha a disposizione 64 valori che verranno codificati in ASCII char 'A'...'Z', 'a'...'z', '0'...'9', '+', '/'.

e-mail gateway

gateway = router più complesso. Oltre alle funzionalità a livello rete svolge anche funzionalità di livello **trasporto e applicazione**.
e-mail gateway = Traduce i messaggi in transito tra sistemi che usano diversi formati e protocolli. Manipola gli header. Procedura:

- msg trasferito da mail server a gateway dell'organizzazione
- gateway determina su quale rete deve essere inoltrato, in base a indirizzo destinazione
- gateway riformatta msg in funzione natura rete.

FTP vs. HTTP vs. SMTP

tutti e 3 usano TCP.

- HTTP :
    - **senza stato** → **non ricorda** quali siano le pagine e i contenuti visitati precedentemente dall'utente (si adottano i **cookies** per risolvere questo problema)
    - trasferisce informazioni di controllo **in-band** → usa **una sola connessione** (sulla porta 80) su cui viaggiano le richieste tra client e server e i dati da server a client.
    - ogni oggetto viene ottenuto tramite un trasferito **separato**.
    - oggetti di qualunque natura.
    - può usare connessioni TCP non-permanenti o permanenti.
    - protocollo **pull**.
- SMTP :
    - trasferimento asincrono di file.
    - si usa una connessione sola in cui tutti gli oggetti vengono trasferiti insieme.
    - mail deve essere sequenza di ascii char (7 bit): tutto deve essere convertito.
    - protocollo **push**.

- FTP : File Transfer Protocol (non più utilizzato per problematiche di sicurezza)
    - ha informazioni di stato → il server ricorda in quale directory del file system server sta lavorando ora il cliente
    - riconosce 2 tipi di file : quelli costituiti da caratteri di ASCII e quelli non, che verranno trasferiti come una sequenza di bit.
    - trasferisce informazioni di controllo **out-of-band** → usa **due connessioni**: una di controllo permanente e un'altra di dati non permanente.
    - lavora per ottetti di caratteri (gruppi da otto)

TELNET

= **TErminaL NETwork.** Protocollo di livello 7 pensato per **connettersi su una macchina remota**, con delle credenziali, e operare su quella macchina come se la stessi utilizzando in locale.
La prima operazione di telnet è quella di controllare l'identità del client. Usa numero di porta **23** (FTP 20 e 21) (25 per la posta elettronica).
Telnet può essere utilizzato come un protocollo **multiprotocollo**, cioè ci si può fingere client di un protocollo di livello 7 e dialogare non come se fosse telnet (usando una porta diversa da 23), questo può essere utile per vedere se ci sono falle nel protocollo. Può essere anche utile per configurare un dispositivo di rete. Allo stesso modo, FTP ha una variante TFTP più semplice, usata da amministratori di rete e di sistema per diagnosi o configurazione.
Esegue anche **marshaling** (capacità di cambiare il formato dei dati per permettere la comprensione tra due macchine). Non più utilizzato per problematiche di sicurezza, le informazioni vengono mandate in chiaro.

Secure Socket Layer

= successore di telnet e FTP. C'è la costruzione di un canale sicuro crittografato prima della comunicazione vera e propria (ssh-trans). *E' una collezione di protocolli,* quali :

- **Secure SHell** (=ssh) come sostituto di TELNET,
- **Secure CoPy** (=scp, permette di copiare un singolo file da/sulla macchina remota) e **sftp** (=secure FTP) come sostituti di FTP .

Secure SHell = ssh

È un'applicazione nata per sostituire TELNET e risolvere i suoi problemi di sicurezza.
Permette di aprire una shell su una macchina remota su cui ho un account per lavorare su quella macchina come se fossi in locale.
Il protocollo di livello applicazione è composto da 3 diversi componenti:

- **SSH-TRANS** → ha l'obiettivo di costruire un canale di comunicazione sicuro sfruttando la connessione offerta da TCP (il protocollo TCP trasmette tutte le informazioni in chiaro e quindi non è in grado di garantire privacy e confidenzialità). Ciò è possibile grazie ad un insieme di *tecniche crittografiche* che permettono al client di verificare che il server a cui si sta collegando sia quello legittimo.
- **SSH-AUTH** → è responsabile dell'autenticazione del client. Oltre alla normale autenticazione con nome utente e password, sono disponibili altri metodi di verifica, come l'accesso basato su coppie di chiavi crittografiche.
- **SSH-CONN** → offre i servizi veri e propri di livello applicazione: accesso al terminale, trasferimento file, esecuzione remota, creazione di tunnel.

Domain Name System

Il DNS traduce un **nome simbolico** in un **indirizzo di rete**, numero unsigned long su 32 bit, che vengono rappresentati come quattro numeri separati da un punto.
Abbiamo già visto che il browser prima di comunicare con il client http chiede aiuto al local resolver che cerca all'interno dei propri file delle corrispondenze nome simbolico-indirizzo IP che può tenere in locale, se non le possiede si passa a un servizio UDP al livello di trasporto (connectionless, non ordinato e inaffidabile) e contatta l'infrastruttura dei DNS sulla porta 53 e le chiede di effettuare la traduzione tra nome simbolico a indirizzo di livello 3.
E' più utilizzato dai protocolli di livello 7 (es.: viene usata dai browser che così si procurano l'indirizzo di livello 3 e lo passano al http-client) che dagli utenti, anche se usando da linea di comando `nslookup` si può fare una ricerca nel name system del nome simbolico che si inserisce come argomento.
La funzionalità di naming e addressing è aumentata, infatti ci sono 3 schemi di indirizzamento:

- nome simbolico → livello 7 applicazione, stringa
- indirizzo IP → livello 3 rete, unsigned long su 32 bit
- MAC address o HW address → livello 2 datalink, indirizzi su 48 bit unici per ogni scheda di rete

ARP *- Address Resolution Protocol*
ARP è un protocollo di livello ***2,5*** che ha la funzione di tradurre gli **indirizzi di rete** (IP, 32 bit) in **indirizzi MAC** (48 bit).
Gli **indirizzi di rete** hanno una struttura che rivela la posizione geografica dell'organizzazione a cui appartengono i dispositivi (host e router). Questi indirizzi sono usati da dispositivi che implementano il **livello 3** (network) dello stack OSI (quindi non da switch e bridge che si fermano al livello 2).
A **livello 2** (data link) sono usati i **MAC** address che non hanno alcuna semantica e servono a distinguere in modo univoco ogni scheda hardware di rete (ethernet, bluetooth, wifi...).
Quando un pacchetto (livello 3) scende nello stack e deve essere incapsulato in un frame (livello 2), nello header del frame deve essere indicato l'indirizzo (di livello 2) della destinazione. Da qui l'esigenza di ARP che traduca l'indirizzo di destinazione da indirizzo rete usato nel pacchetto a indirizzo MAC da inserire nel frame. Per questo motivo è un servizio analogo alla traduzione da nome simbolico in indirizzo rete operata da DNS.
I nomi simbolici erano unici un tempo, oggi sono cambiate: se ho un server particolarmente acceduto, lo copio (es.: server farm). Non si accede a una sola macchina, ma ad un gruppo: il nome non è più unico per un host, ma per un gruppo di host. Anche il DNS, oggi, è capace di smistare il carico su diversi host/server.

gerarchia nomi simbolici

Da destra verso sinistra. Sotto la root troviamo in ordine i seguenti campi.

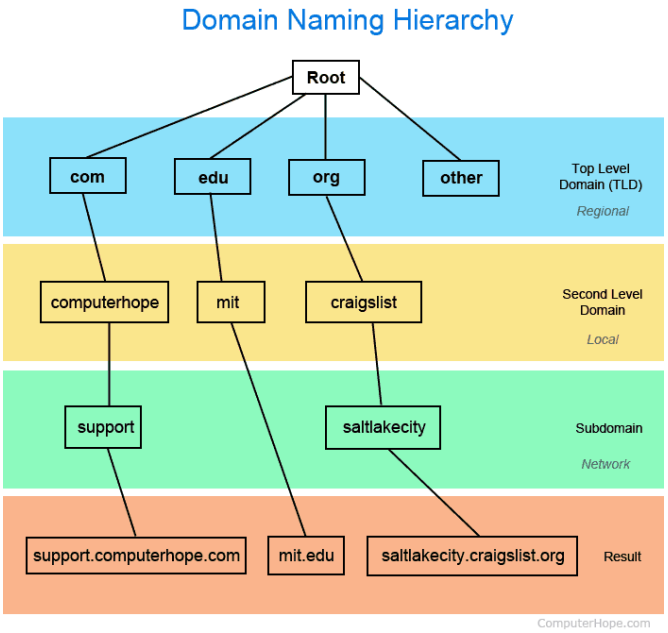
-Top Level Domain = TLD, che si differenziano in

- nomi **generici** : .com, .edu, .gov, .mil, .org, .net, .int, etc..
- domini di **paese** : .uk, .us, .it, .jp, etc..

-**Second Level Domain**

-**Subdomain**

Si ha come risultato un identificativo detto **fully qualified domain name** che dice tutto: dal dispositivo particolare fino alla radice. Se si passa al resolver un nome incompleto cerca di risolverlo automaticamente.



Come memorizzare i dati su dispositivi fisici?

Si ripartiscono le informazioni relative ai nomi di dominio tra diversi calcolatori, i **DNS** (detti anche **name server**). L'infrastruttura del DNS è distribuita (non su un unico server centrale). Ci sono 13 nodi di root e ognuno corrisponde a svariati server fisici, dato che non è possibile memorizzare l'intera gerarchia dei nomi di dominio su un solo server. C'è un **albero di server** che riproduce la gerarchia. Tutte le root conoscono i server che gestiscono i TLD, che a loro volta conoscono i server dei domini sottostanti. L'unicità dei nomi a ogni livello implica che i fully qualified domain name siano unici. Il database è **parzialmente replicato** cioè l'informazione di corrispondenza da nome simbolico a relativo indirizzo IP si trova in più nodi, garantendo **robustezza**.

Fino a questo punto si è parlato di **server logici**: un server logico, in realtà, può corrispondere a più **server fisici**, per ogni nodo interno (non foglia) della gerarchia ci sono almeno 2 server fisici, uno primario che viene interrogato e uno secondario che viene tenuto per backup. Gli end-system (foglia) conoscono i 2 name server del proprio dominio.

rete autonoma = autonomous system, rete o interconnessione di reti che fa capo a un'unica organizzazione.

L'unicità di un name server è garantita dall'autorità **ICANN** = Internet Corporation for Assigned Names and Numbers. E' una organizzazione che coordina :

- l'infrastruttura di DNS,
- gli indirizzi IP,
- controllo dello spazio di allocazione dei nomi,
- assegnamento degli identificatori ai protocolli,
- gestione dei server di TLD (sia generici sia di paese) e dei root server.

Una divisione di ICANN è **IANA** = Internet Assigned Numbers Authority. E' responsabile per:

- il coordinamento globale dei root server,
- l'indirizzamento IP,
- le risorse di alcuni protocolli,
- la distribuzione dell'autorità per gestire i numeri che riguardano internet (gli indirizzi, i protocolli) tra 5 altre autorità, i **RIRs** = Regional Internet Registries, per i continenti del mondo.

I Registries (RIRs) a loro volta delegano a dei **Registrar nazionali** il controllo dell'unicità dei nomi. I Registrar gestiscono i top level domain e controllano l'unicità dei nomi dei figli. Registrar inserisce le informazioni nel DNS: il nome del dominio più i due server autoritativi. IANA inoltre assegna indirizzi hardware ai produttori **NIC** = Network Interface Card, cioè l'hw della scheda di rete (48 bit MAC o hardware).

root name server

Un **root server** viene contattato da un **local name server** che non sa "risolvere" il nome del main server che vuole contattare. Il root name server ha il compito di :

- contattare i name server che hanno autorità (*sono autoritativi*) per il nome che si sta cercando, se il mapping di quel nome non è noto al local name server.
- recupera il **mapping** e li restituisce al local name server o aiuta il local name server a recuperare il mapping.

Il local name server conosce il root name server perchè è configurata al momento del **bootstrap** del sistema da un file di configurazione nel sistema operativo.
Ci sono 13 root server logici gestiti da 12 organizzazioni (una di queste ARPA) e ognuno di questi server corrisponde a tanti server fisici replicati (se ne può vedere una mappa su https://root-servers.org/).
Politica di instradamento **anycast** = si interroga un server logico (o un pool di server fisici) prendendo la risposta da uno qualunque delle repliche del server logico.

DNS resource record

Per ogni nome è mantenuto uno (o più) resource record. Un **record di risorsa** si trova nel database mantenuto da ogni **name server**. E' composto da righe di 32 bit contenenti:

- **nome di dominio** (identificativo del record, può avere 255 caratteri al massimo)

- **tipo** (come interpretare il campo valore: determina informazioni mantenute nel record)
- **classe** (tipo di rete) (di fatto ha valore solo IN, cioè Internet)
- **TTL** (=time to live, tempo di vita di questo record) (segnala per quanto tempo deve rimanere in cache l'informazione) (generalmente 2 giorni)
- **valore** (informazione relativa al nome del dominio).

Ad ogni nome di dominio (un nodo) è associato un record di risorsa. Il database del server dei nomi non è altro che una collezione di record di risorsa.
I nomi sono autoritativi se sono assegnati a discendenti, altrimenti, se il nome viene trovato in una cache, è non-autoritativo.
Un nome di dominio è una concatenazione di label di byte (max. 63 per label), ogni informazione è divisa dell'altra da un byte contenente il numero di byte dell'informazione successiva.

DNS query and reply
i messaggi DNS sono di due tipi: **interrogazione** (query) e **risposta** (response). Hanno lo stesso formato:

- query header (12 byte)
- query (domain) name
- query type
- query class

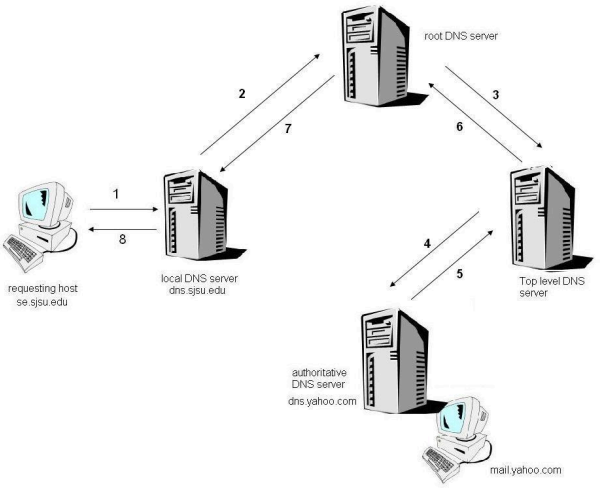
In più, se è una risposta, sono presenti in coda dei record di risorsa per dare tutte le informazioni trovate su quel nome di dominio.
I messaggi hanno, nel campo query header, vari campi che li descrivono:

- **id** (numero pseudocasuale)
- **un flag** (indica se è risposta o query, se la risposta è autoritativa o meno e indicano la procedura di risoluzione desidereta tra ricorsiva e iterativa)
- altri quattro campi che definiscono il numero di ciascun tipo di record nel messaggio (numero di domande o risposte, numero di resource record autorevoli e addizionali).

risoluzione (locale)
Il processo che permette di associare un **indirizzo IP** ad un **nome noto** è detto **risoluzione** di un nome. Un host che vuole associare un nome a un IP, o viceversa, si rivolge a un programma client DNS, detto **resolver**. Il resolver invia una richiesta al server DNS più vicino; questi, se ha un informazione, comunica l'indirizzo o il nome cercato al resolver, altrimenti o si rivolge ad un altro server o comunica al resolver l'indirizzo di un altro server a cui fare riferimento. Ricevuta la risposta, il resolver l'analizza per accertarsi che non ci siano errori e trasmette il risultato al processo che aveva chiesto la risoluzione.
La risoluzione può essere iterativa o ricorsiva.

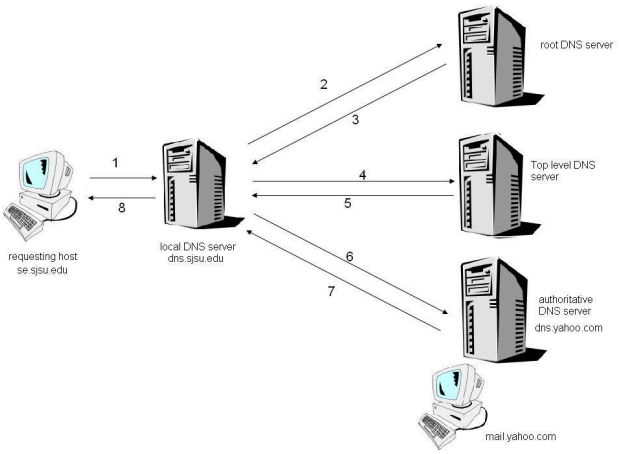
Risoluzione ricorsiva

Es.: abbiamo un programma applicativo eseguito su un host di nome X -sorgente-, che vuole conoscere l'IP di un altro host, chiamato Y -destinatario-.



X, a questo punto, applica la tecnica di risoluzione: il programma applicativo chiede al **resolver DNS** l'indirizzo IP, ma dato che quest'ultimo non lo conosce, esegue una query al **server DNS locale** di X (quello più vicino). Ipotizziamo che neanche il server DNS locale conosca l'IP di Y. Invia, quindi, la query a un server **DNS root**, il cui indirizzo IP è noto a questo server DNS locale. Il DNS radice, non conosce a sua volta l'IP di Y, ma conoscendo il **server del suo dominio top-level**, gli invia la query. Il server di dominio top-level non conosce l'associazione nome/IP (in questo caso), ma conosce l'IP del **server DNS locale** di Y. La query viene così inviata a questo nuovo server che conosce l'IP dell'host destinatario. L'IP viene ora inviato al server DNS top level e, poi, al server DNS locale di X e restituito a X.

Risoluzione iterativa



Il **local resolver** che non trova la corrispondenza all'interno del proprio file chiede aiuto al **name server locale** che chiede alla **radice**, se non ha la risposta in cache restituisce al name server locale l'indirizzo IP del **top level Domain** a cui ci si deve rivolgere, invece di delegare la query lei stessa (la radice) al top level domain. Di conseguenza il local name server del richiedente si occupa di andare al server top level di competenza che fa la stessa cosa della radice, se non ha la risposta in cache rende al local name server l'indirizzo IP di uno dei suoi figli. I nodi intermedi non si ricordano niente, a differenza della risoluzione ricorsiva. Meno carico di memoria, di processing ma anche di caching perchè la reply generata dal main server va direttamente a questo.

risoluzione ricorsiva vs iterativa

caching della risposta:

- nella risoluzione ricorsiva viene fatto da **tutti i nodi intermedi**;
- nella risoluzione iterativa la risposta rimane solo nella **cache del name server locale** che può aiutare solo le richieste dello stesso host;
- la risoluzione ricorsiva è vantaggiosa per le richieste successive.

risoluzione inversa **//non è presente sul libro. Non la chiede.**
Invece di risolvere un nome simbolico per ottenere un indirizzo IP corrispondente fa il contrario. Coinvolge solo un tipo di server come mail server e file server per autenticare sorgenti richieste. Si ha un indirizzo IP e si vuole sapere a che nome simbolico corrisponde. Gli indirizzi IP di 32 bit sono divisibili in bit più significativi in base alla loro posizione: a sinistra si trova la parte più generica e corrisponde all'identificatore di rete mentre a destra si trova la parte più specifica l'identificatore di host.

DNS Dinamici
Il problema è che l'infrastruttura dei DNS è statica. Però anche l'informazione nei vari resource records lo è. Normalmente il valore di una time to live è circa due giorni. Negli apparati mobili, tuttavia, non è proprio vero. Con i dispositivi mobile, spostandosi, ci si muove in punti diversi della rete e l'IP cambia. Bisogna saper cambiare l'associazione tra IP e nome del server con il DNS. Viene chiesto servizio a un **provider** esterno ai DNS che offre un servizio di registrazione (name server solo per indirizzi dinamici). Ogni volta che l'host cambia IP, il client invia al server provider il nuovo indirizzo e quest'ultimo l'aggiorna. Il provider usa un proprio dominio, ed è cliente dei DNS dinamici e server degli utenti. RFC 2136 è l'estensione che permette ad un name server di agire da supporto per trovare le info sui servizi mobili.

livello trasporto

Il livello di trasporto nell'architettura OSI è il livello 4, sotto di esso c'è la rete. E' il livello di astrazione più basso che non vede la topologia della rete e regola la comunicazione end-to-end, cioè regola la comunicazione dei dati delle applicazioni tra due end-system terminali di una comunicazione. Offre ai livelli superiori dei servizi che sono **indipendenti** dalla natura della rete sottostante.
Ci sono due protocolli di trasporto, che vengono scelti dall'applicazione utente:

- **TCP** = Transmission Control Protocol. Connection-oriented, ordinato e affidabile. Affronta anche problemi dovuti alla natura della rete, fallimenti, guasti, controlla se ci sono errori. Garantisce il controllo di flusso e congestione.
- **UDP** = User Datagram Protocol. Connectionless, non ordinato e inaffidabile.

Il livello trasporto, con entrambi i protocolli, si occupa dell'indirizzamento di livello trasporto.

controllo/gestione degli errori
TPC per garantire l'**ordinamento** mette un numero di sequenza appiccicata ai vari pezzi di dati e poi il destinatario li riordina.
Il **controllo degli errori** è un po' più complesso. L'errore si può verificare quando l'informazione passa in un cavo:

- in fibra ottica, bit error rate di 10^{-9} (meno probabile)
- in rame, bit error rate di 10^{-6}
- oppure via wireless: il bit error rate è di 10^{-3}.

Dato che il livello di rete (IP) è inaffidabile, è necessario implementare l'**affidabilità** al livello trasporto, che ha la responsabilità di:

- rilevare e scartare i pacchetti corrotti;
- tenere traccia dei pacchetti persi e scartati e gestirne la rispedizione;
- riconoscere i pacchetti duplicati ed eliminarli;
- bufferizzare i pacchetti fuori sequenza fino a quando arrivano quelli mancanti.

Per fare questo servono degli **algoritmi** (implementati dai protocolli) che ricadono nella categoria **ARQ** = **Automatic Repeat reQuest**. Tutti gli algoritmi fanno uso, nel dialogo tra le parti comunicanti, di informazioni di controllo:

- **ACK** = **acknowledgment**, riscontro positivo, permettono alla destinazione di dire alla sorgente cosa ha ricevuto. Il sender manda un dato e se non riceve un ACK o non lo riceve entro un certo tempo allora assume che il dato non sia arrivato a destinazione e quindi lo trasmette nuovamente.
- **NAK** = **negative acknowledgment**, riscontro negativo. Se un receiver si accorge di ricevere qualcosa di errato o di non ricevere quello che attende, allora manda un NAK al sender. Se però tutta la parte finale della trasmissione è persa un receiver potrebbe non rendersi conto perchè non vede dei buchi.

Alcune considerazioni:
ACK è pesante. NAK e ACK possono essere persi come gli altri dati. I NAK soffrono di "haste acknowledge" (fretta di riconoscere): se tutta la parte finale della trasmissione di un contenuto si perde un destinatario potrebbe non rendersene conto.
Ci sono due tipi di schemi:

- **idle RQ** = semplice ma poco efficace
- **continuous RQ** = complicati ma efficienti

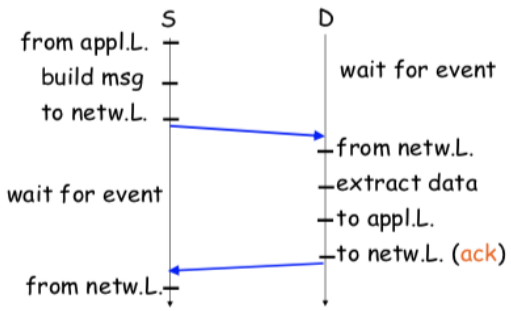
protocolli connection oriented
Bisogna fare alcune assunzioni :

- al livello applicazione i dati sono sempre disponibili: il livello trasporto non si ferma ad aspettare dei dati;
- non viene considerato il contenuto dei dati né il processing per il controllo degli erorri;
- i messaggi portano un numero di sequenza = *#seq*;
- i dati che arrivano dal livello superiore hanno una dimensione massima;
- il messaggio ha un **message header = <*kind, #seq, ack*>**;
- il canale fisico può perdere messaggi.

Il livello di trasporto può usare un **timer** per gli ACK. Questo timer deve essere grande almeno tanto quanto il RTT= round trip time.

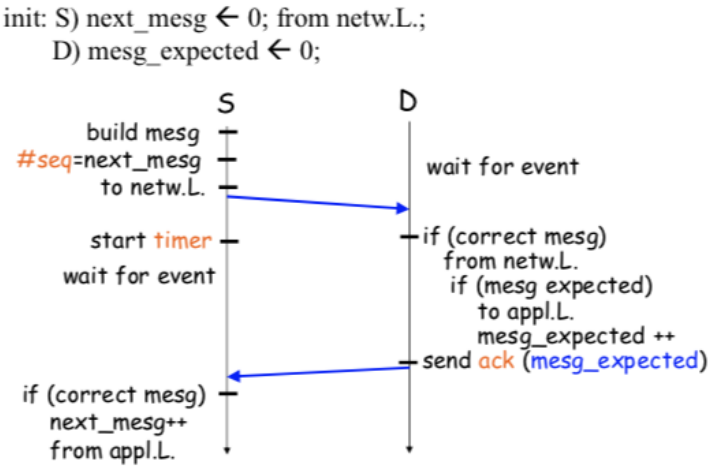
very silly protocol → inaffidabile

Si studia un protocollo semplice, privo di connessione e controllo degli errori e di flusso. Si suppone che il destinatario sia in grado di gestire immediatamente tutti i pacchetti ricevuti (niente sovraccarico).
Il livello trasporto mittente riceve un messaggio dal proprio livello applicazione, ne crea un pacchetto e lo invia (fornisce un servizio per il suo livello applicazione). Il livello di trasporto destinatario riceve il pacchetto dal proprio livello rete, ne estrae i dati e lo consegna al proprio livello applicazione. Il mittente non può inviare pacchetti fino a quando il suo livello applicazione non ha un messaggio da spedire e, analogamente, il destinatario non può consegnare al proprio livello applicazione fino a quando non riceve un pacchetto (il protocollo UDP corrisponde quasi del tutto a questa descrizione).
Se nella rete c'è un errore non viene segnalato [...]. Se la sorgente è più veloce della destinazione, si vanno a perdere dei dati.



silly protocol
Molto simile al precedente, ma con alternanza stretta tra sorgente e destinatario. Ci sono dei tempi morti, quindi non è molto efficiente. Se perdo dati, inoltre, si crea un deadlock.

PAR - Positive Acknowledgment with Retransmission **//non è presente sul libro.**



//PAR è Idle RQ, può essere visto come una versione semplificata di Stop-and-Wait, semplicemente aggiungendo rispetto a PAR la possibilità di usare il NAK.

E' un algoritmo semplice ma funziona per dare un servizio ordinato e affidabile.

All'inizio la sorgente memorizza un numero 0 per numerare i prossimi dati che devono partire da livello applicazione, mentre la destinazione memorizza che i prossimi dati attesi devono avere un numero di sequenza 0. La sorgente prende dati dall'applicazione e costruisce un messaggio con un header che ha come #seq (=numero di sequenza) il valore *next* pari inizialmente a 0. Passa i dati a livello rete, fa partire un timer e aspetta che si verifichi un evento che può essere:

- la scadenza del timer
- la ricezione di un ACK

La destinazione rimane in attesa e se arriva un messaggio corretto, lo prende dal livello rete, ne estrae i dati e il #seq dallo header. Se era il messaggio atteso lo passa al livello applicazione e incrementa il numero di messaggio atteso. Infine (anche nel caso in cui il messaggio non era corretto) manda un ACK a cui viene allegato il valore del messaggio atteso. Quando la sorgente riceve un ACK per il messaggio che ha appena mandato, incrementa il #seq da allegare al messaggio e prende altri dati dal livello applicazione. Se l'ACK è per un altro messaggio o scade il timer il sender invia nuovamente il messaggio con gli stessi dati e con lo stesso #seq effettuando una **ritrasmissione**.

Aspetti **positivi** di PAR:

- Il timer (più o meno grande come round trip time) previene il deadlock che si verificava nel silly protocol nel caso in cui si presenta una perdita di dati o di ACK .
- il #seq previene la consegna di duplicati se si perdono ACK. Se la sorgente rinvia i dati, in caso di mancato ACK, la destinazione li scarta, avendoli ricevuti correttamente in precedenza e aspettandosi un #seq superiore.

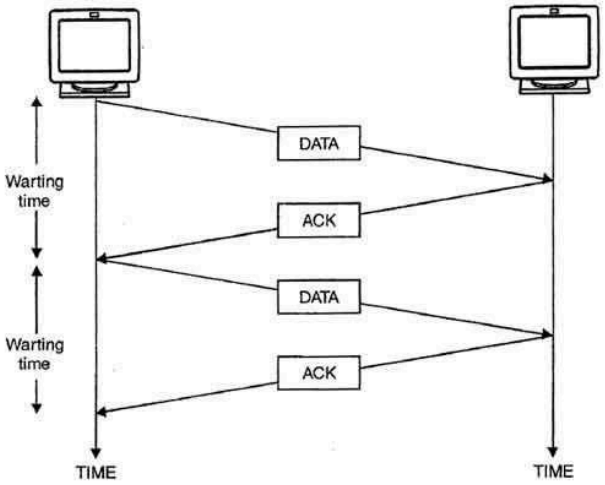
In questo caso l'ACK non è anonimo ma è numerato, dice anche che cosa è arrivato.

L'upper bound di MAX_SEQ è limitato, 1 bit, quando finisce si riparte dall'inizio. Il range di #seq, in sostanza, deve essere limitato e uguale, così l'informazione nell'header sarà uguale a tutti i livelli (ciò è utile anche per la frammentazione). Quando si raggiunge il valore massimo di #seq si riparte da capo.

Ma come fa la destinazione a capire che un messaggio con un #seq uguale a un precedente è un nuovo messaggio e non un duplicato del precedente? Questo grazie all'ACK che dice cosa si aspetta (msg_expected).

Stop-and-wait → idle RQ

Meccanismo connection-oriented, utilizza sia il controllo di flusso sia degli errori.
Il sender e il receiver usano una **finestra scorrevole** (MAX_SEQ limitata) di dimensione 1.
Viene settato un **contatore di ritrasmissione** a 1. Il sender fa partire un timer e aspetta:



Stop & Wait Method.

-arrivo ACK → se arriva per il messaggio ricevuto il campo ACK ha un valore che è uguale al valore next message vuol dire che il ricevente ha ricevuto il messaggio. Quindi il sender preleva dati dal livello superiore, li mette nel buffer sovrascrivendo i dati precedenti, incrementa modulo 2 il numero di sequenza da allegare ai nuovi dati e inizializza nuovamente il contatore di ritrasmissione a 1.
-arrivo NAK o scadere del timeout → se riceve qualcosa che ha il campo NAK non vuoto, e in particolare contiene il valore pari al numero di sequenza del messaggio che ha mandato, o scade il timeout, il sender incrementa il **contatore di ritrasmissione**. Se questo è superiore a un certo valore massimo segnala un errore a livello applicazione ed esce, altrimenti il sender costruisce un nuovo segmento identico a quello che aveva inviato prima.

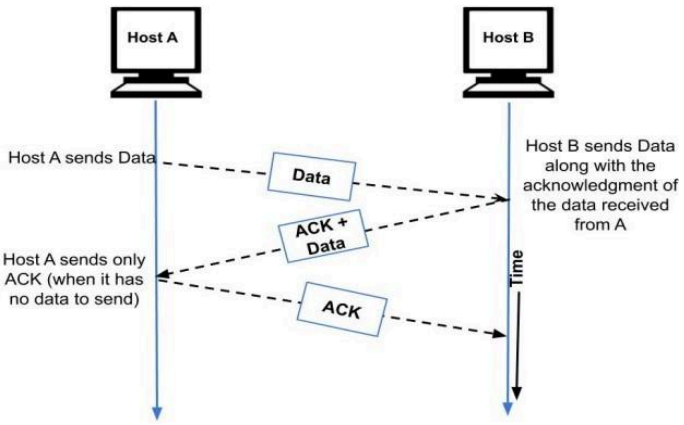
Il receiver aspetta un evento che può essere:

- messaggio errato → viene buttato via e si manda un NAK.
- messaggio corretto → Se è il messaggio atteso si passano i dati al livello applicazione, si manda al network layer un messaggio di ACK con associato il numero di sequenza del messaggio ricevuto e si incrementa il **numero di messaggi ricevuti.** Altrimenti se il messaggio non è quello che si attendeva (cioè è un duplicato) viene scartato, si decrementa il numero di messaggi ricevuti e si manda un ACK.

Osservazioni:

- il mittente deve tenere una copia del pacchetto fino a quando non ne riceve un ACK.
- Per potere gestire i pacchetti duplicati lo Stop and Wait, come PAR, utilizza i numeri di sequenza e di riscontro. Nel meccanismo Stop and Wait *il numero di riscontro indica, in aritmetica modulo 2, il numero di sequenza del prossimo pacchetto atteso dal destinatario.*
- la comunicazione è unidirezionale: S manda a D e D manda indietro soltanto ACK o NAK (in httt non era così).

Protocolli bidirezionali: ***piggybacking***

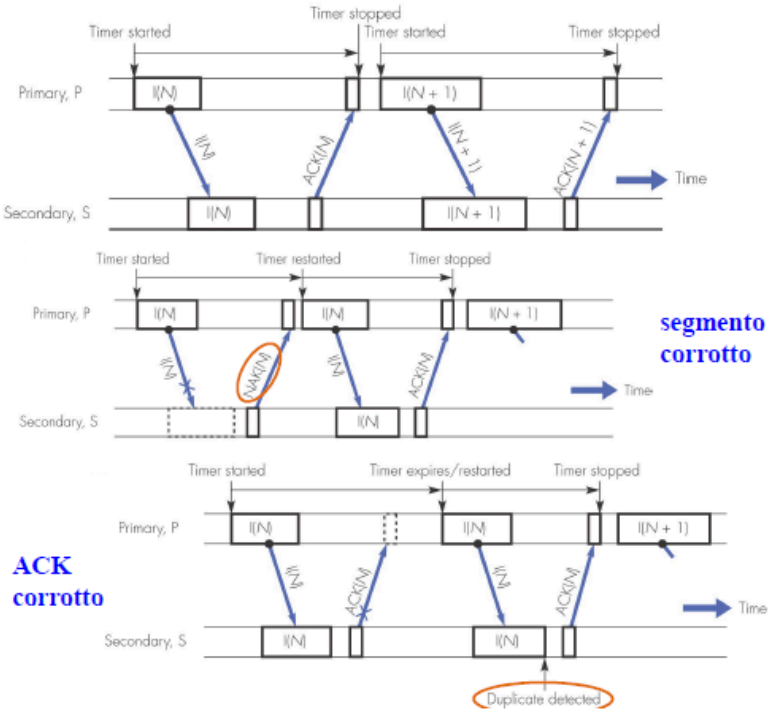


I meccanismi visti finora erano unidirezionali: i pacchetti scorrono solo in una direzione, così come i riscontri. In realtà i pacchetti possono scorrere in entrambe le direzioni (**full-duplex**). Il piggybacking è un meccanismo adottato per migliorare l'**efficienza** dei protocolli bidirezionali. Quando un pacchetto trasporta dei dati da A a B, può trasportare anche ACK relativi ai pacchetti ricevuti da B (e viceversa).
Si manda un ACK da solo quando non ci sono dati da mandare e si rischia di far scadere il timer causando una ritrasmissione da parte dell'altro host.

situazioni di errore

segmento corrotto: il secondario manda un NAK al primario che rimanda lo stesso segmento e fa ripartire il timer. Il secondario riceve il segmento e manda ACK. Il primario ora può mandare il segmento+1.

ACK corrotto: un ACK può far capire che il ricevente ha ricevuto più messaggi e si chiama ACK cumulativo. A fronte dell'invio di un segmento con numero di sequenza *n* il secondario invia un ACK, questo può arrivare corrotto - o non arrivare proprio - al primario. A questo punto scade il timer e ritrasmette l'ultimo segmento con numero di sequenza *n*, il secondario capisce che è un duplicato, quindi lo butta e manda un ACK con numero di sequenza *n*. Il primario manda il segmento con numero di sequenza *n+1*.



Nb: l'ACK è cumulativo.

efficienza di stop-and-wait : utilizzo del canale in idle RQ

l=message size; R=rtt; b=bit rate;

transmission delay T_x = l/b

propagation delay T_p = R/2= (lunghezza del cavo)/(velocità di propagazione)

utilizzo(=occupazione) del canale = il canale viene utilizzato per un tempo pari al ritardo di trasmissione T_x. Non si trasmettono più dati per il tempo di propagazione T_p di quei dati al secondario e per un R la sorgente aspetta un ACK (non contato come utilizzo perché sono info controllo di dimensioni trascurabili).

utilizzo U $=(l/b) / (l/b + R) = l/(l+bR) = T_x/(T_x+2T_p)$

Se l < b*R l'utilizzo è inferiore al 50%. I messaggi devono essere grandi per non sprecare risorse di rete.

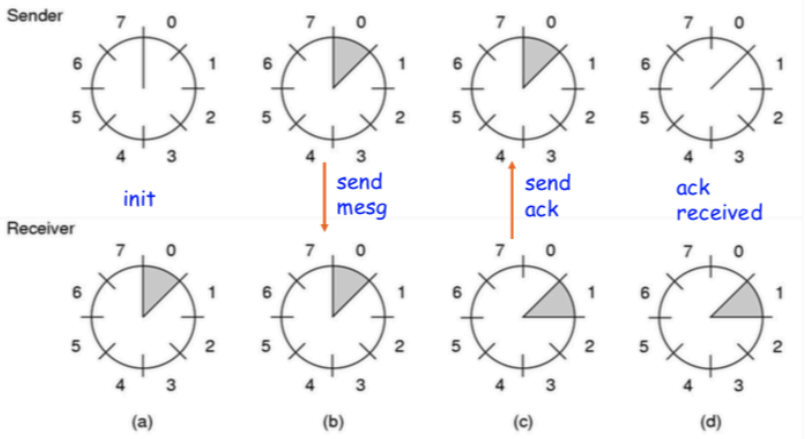
sliding window → continuous RQ

//alternativa a idle RQ, infatti è un idle RQ adeguato per link corti (rtt breve) e con basso bit rate (B piccola) → utilizzo superiore al 50%. Se si permette parallelismo tra i messaggi (ovvero permettere al sender di inviare altri messaggi mentre aspetta ACK) allora si può aumentare l'utilizzo:

- il sender mantiene una finestra aperta per i messaggi inviati in attesa di ACK. Quando arriva un ACK per il messaggio con il minimo numero di sequenza (il segmento più antico che è stato mandato) l'elemento con quel numero di sequenza viene buttato fuori dalla finestra e la si fa scorrere in avanti. Il sender ha il permesso di mandare un segmento nuovo.
- la finestra del receiver deve permettere di alloggiare il prossimo messaggio atteso.

In una finestra scorrevole che utilizza numero di sequenza di 1 bit le fasi sono:

a. Il sender non ha nulla nella finestra perché non ha ancora inviato alcun messaggio mentre il receiver ha una posizione nella sua finestra di ricezione perché sta aspettando un messaggio con numero di sequenza 0.
b. Il sender invia il primo messaggio e la sua finestra di invio diventa di dimensione 1 (perché inserisce il messaggio inviato in attesa di ACK). Il messaggio non è ancora arrivato e quindi la situazione del receiver rimane invariata.
c. Il messaggio arriva al receiver che lo controlla, lo consegna all'applicazione,manda un ACK verso il sender e fa scorrere in avanti la propria finestra (si mette ad aspettare il messaggio con numero di sequenza 1). Il sender sta ancora aspettando un ACK.
d. Il sender riceve l'ACK e di conseguenza elimina dalla finestra il messaggio 0.



continuous RQ

ci possono essere due approcci:

- **Go-Back-N** = semplice, il ricevente ha una finestra(/buffer) di ricezione che è uguale esattamente a 1, ovvero si aspetta soltanto il prossimo messaggio che dovrebbe consegnare al livello applicazione sovrastante. Se riceve messaggi duplicati e vecchi li butta via perché non servono. Ha un alto spreco bwth se rtt grande perché i messaggi non attesi vengono buttati e poi richiesti.
- **Selective-Repeat** = il ricevente mantiene un buffer di posizioni > 1, i messaggi fuori ordine vengono messi nel buffer in attesa di consegna e ACK. Il sender deve inviare solo i messaggi mancanti.

Con entrambi questi approcci è possibile realizzare il pipelining e aumentare l'utilizzo. Bisogna però fare un compromesso tra spazio occupato nel buffer (e conseguente computazione eseguita dal ricevente) e utilizzo della capacità della banda. In entrambi i casi c'è bisogno di un timer che quando scade fa rinviare tutti i pacchetti in attesa di riscontro. Dal lato sender c'è una **lista di ritrasmissione** (lista linkata) che ricorda quali sono i messaggi che sono stati trasmessi in attesa ACK. Questi messaggi sono gestiti in ordine FIFO.

GBN - Go-Back-N

//Lo Stop-and-Wait è in realtà un GBN particolare, con due soli numeri di sequenza e la finestra di invio ha dimensione 1. Nel meccanismo Stop and Wait il mittente deve attendere che il pacchetto precedente abbia raggiunto il destinatario e che il relativo riscontro sia stato ricevuto, prima di poter inviare il pacchetto successivo.

Go-Back-N è un meccanismo che, tramite la tecnica nel pipeling, permette di inviare più pacchetti prima di ricevere riscontri. Il mittente mantiene una copia dei pacchetti inviati fino a quando non riceve un riscontro. Fa anche uso di **ACK cumulativo**= conferma che tutti i frame con numero ≤ k siano arrivati. E' un miglioramento perchè è più resistente a perdita di ACK, anche se si perde un ACK di numero inferiore, è garantito arrivo tutti i frame. (Es: invio ack di 1 2 3 4 5 6, si perde ack(3) ma arriva ack(6), è tutto ok per via dell'ACK cumulativo e quindi non si rimanda nulla, risparmiando ritrasmissioni.)

Situazioni di errore:

- se c'è un **frame corrotto**: i successivi vengono scartati e ritrasmessi tutti da quello corrotto in poi.
- Se **ACK n corrotto**: se ACK n+1 o n+2 o n+3 arriva in modo corretto, sorgente considera anche frame senza ACK(conferma) validi, quindi **non** ritrasmette.

SR - Selective-Repeat

Il destinatario accetta messaggi nella finestra anche fuori ordine, ovvero il destinatario mette a disposizione una maggiore quantità di memoria rispetto a GBN che accettava soltanto il prossimo segmento che avrebbe dovuto consegnare a livello di applicazione. La consegna rimane però ordinata.
L'ACK può essere sia cumulativo che selettivo a un singolo segmento, dipende dalla implementazione.
Se ci sono errori si cerca di ritrasmettere esclusivamente i segmenti persi o corrotti.
Si usa un timer per decidere l'invio di un ACK in assenza di traffico di ritorno per evitare blocco (<rtt/2), ovvero per utilizzare al meglio la capacità del canale si manda uno header con dentro un ACK. Vengono usati anche negative ACK per velocizzare il recupero dei dati.
Il dimensionamento del timer lato sender dipende da cosa si vuole sfruttare meglio:

- se grande, il vantaggio del NAK è maggiore
- se piccolo, i NAK diventando inutili e si spreca bwth per ritrasmissioni extra

Situazioni di errore:

- caso **frame corrotto**: dopo aver ricevuto il frame N+1 corretto, il receiver manda un ACK(N+4) e il sender considera tutti i frame mandati come arrivati correttamente (ACK cumulativo). I NAK possono perdersi, quello che garantisce veramente l'arrivo di un frame correttamente sono i timer e gli ACK; NAK è opzionale, serve solo per velocizzare ritrasmissione e un protocollo senza NAK è comunque considerato corretto.
- caso di **ACK corrotti**: viene risolto quando il sender riceve ack(N+1), in questo modo sa che anche i precedenti frame non sono arrivati in modo corretto.

range numeri di sequenza
Ci sono dei limiti sulla dimensioni delle finestre della finestra di ricezione e di invio, legati ai valori che possono essere assunti dal dal massimo numero di sequenza. Il massimo numero di sequenza dipende dai bit che vengono programmati all'interno della struttura dell'header e questo dipende dalla descrizione standard del protocollo.

- GBN: la dimensione massima della finestra di invio deve essere uguale al massimo numero di sequenza, mentre la finestra di ricezione è uguale a 1.
- SR: la dimensione massima della finestra di invio e di ricezione deve essere (MAX_SEQ+1)/2.

//Dimostrazione
Ipotesi: j bit usati per rappresentare il #seq, quindi ho k (= 2^j = MAX_SEQ +1) valori per #seq, da 0 a (2^j -1=) MAX_SEQ. Es. se j=3 ho 2^3=8 valori da 0 a MAX_SEQ=7.

GO-BACK-N: ha finestra ricezione di taglia 1, e finestra di invio SW (sliding-window) di taglia massima MAX_SEQ.
Infatti, se taglia SW fosse > MAX_SEQ (per esempio MAX_SEQ+1), allora: se j=3, e dunque SW = MAX_SEQ+1 = 7+1 = 8, all'inizio sender ha 8 segmenti con #seq da 0 a 7 nella propria finestra. Supponiamo che li invii tutti, che il ricevente li riceva tutti, che mandi relativi ack ma tutti gli ack si perdono. Dopo l'invio, sender ha sempre gli 8 segmenti da 0 a 7 in attesa di ack; il ricevente - dopo aver ricevuto 7 - fa scivolare la propria finestra ad attendere il segmento successivo, con #seq 0. Quando a sender scade timer, re-invia 0 (e tutti i successivi); 0 è valore atteso da ricevente che tiene il segmento 0 vecchio (ritrasmesso) come fosse 0 nuovo e lo consegna all'applicazione, violando così ordinamento. ERRORE! Con taglia SW <= 7, finestra sender contiene segmenti da 0 a 6. Nel caso precedente, dopo aver ricevuto 6 il receiver si mette in attesa di 7. Quando riceve il vecchio 0 ritrasmesso lo riconosce correttamente come duplicato, lo scarta, e rimanda ack dicendo che fino a 6 ha ricevuto e attende 7. E tutto funziona.

SELECTIVE REPEAT: ha finestra di ricezione e di invio di taglia massima (MAX_SEQ+1)/2.
Infatti, se la taglia fosse come in Go-Back-N (7 nell'esempio), e si ripete il caso di prima: dopo aver ricevuto 6, il receiver fa scorrere la propria finestra andando ad occupare le successive 7 posizioni [7 0 1 2 3 4 5]. Quando al sender scade il timer, ritrasmette il vecchio segmento 0, che cade nella finestra del receiver il quale lo tiene come nuovo 0 e lo passa a livello applicazione. ERRORE! Con finestre (7+1)/2=4, il sender ha in finestra [0 1 2 3]; receiver li riceve ma tutti gli ack sono persi. A questo punto la finestra del receiver è scorsa alle successive posizioni [4 5 6 7] e quando sender rimanda 0 receiver lo riconosce correttamente come vecchio 0, lo scarta, e rimanda ack dicendo che ha ricevuto fino a 3 ed attende 4.

efficienza di GBN e SR: utilizzo del canale in continuous RQ
l=message size; R=rtt; b=bit rate;
transmission delay $T_x = l/b$
propagation delay $T_p = R/2$= (lunghezza del cavo)/(velocità di propagazione)
k = numero di frame nella finestra di invio
Tempo considerato per mandare e ricevere ACK = T_x+2T_p

$$\mathbf{U} = k \cdot T_x/(T_x+2T_p) = k/(1+2T_p/T_x) = kl/(l+bR)$$

U tende al 100% quando k·Tx tende a Tx+2Tp da cui si ottiene la dimensione ottimale della finestra di trasmissione.
E' indifferente che approccio si usi, sia Go-Back-N o Selective- Repeat, ma si differenziano per **efficienza** in recupero errore, anche se in entrambi il destinatario può ricevere altri k-1 messaggi prima che il sender se ne accorga.

controllo di flusso
Go-Back-N non richiede memoria al receiver che aspetta soltanto il prossimo segmento che deve consegnare in ordine, la gestione è più semplice, ci sono meno computazioni, non c'è controllo di flusso perchè il ricevente è predisposto a ricevere solo il segmento richiesto e il

resto viene “buttato via”, infatti non ci sono problemi di overflow di memoria. Questo protocollo è però più dispendioso in termini di banda.
Selective-Repeat può negoziare il dimensionamento della finestra lato receiver ed eventualmente ri-negoziabile dinamicamente -durante un conversazione la dimensione potrebbe cambiare e quindi essere più piccola dell’upper bound (MAX_SEQ+1)/2, negoziazione *max message size* per facilità bufferizzazione.

TCP - Transport Control Protocol

TCP è un protocollo di tipo:

- **two-way**= cioè full-duplex, supporta lo scambio di dati tra due parti coinvolte in una conversazione contemporaneamente.
- **connection oriented** → ordinato e affidabile
- **byte-oriented** (stream)= quello che viene scambiato è una sequenza di byte e TCP numera ognuno di questi byte. (è possibile trasferire anche un singolo byte, es.: un comando inviato in remoto fatto di un solo byte).

I dati sono trasferiti in segmenti i quali hanno una **maximum segment size (MSS)** tale per cui non è necessaria la frammentazione sulla rete (garanzia di connection oriented). TCP tenta sempre di riempire completamente il segmento alla MSS stabilita per ottimizzare la banda. Se non negoziato durante l'handshake la MSS è settata a **536 byte** (requisito minimo su Internet).
I segmenti non sono frammentabili e la MSS deve essere minore della più piccola protocol data unit inoltrabile dai router sul cammino.
Non vengono preservati i confini dei messaggi al livello applicazione, l'applicazione deve saper rimettere insieme i segmenti creati da TCP, ad esempio una jpeg viene frammentata a MSS e non trasmessa a dimensione intera.
Nota: i segmenti contengono i dati di una determinata applicazione eseguita da un determinato processo, non c'è alcuna possibilità che in un segmento siano multiplexati dati che arrivano da applicazioni diverse eseguite da processi diversi sul medesimo host.
Ci sono due possibili richieste che arrivano dalla applicazione a TCP:

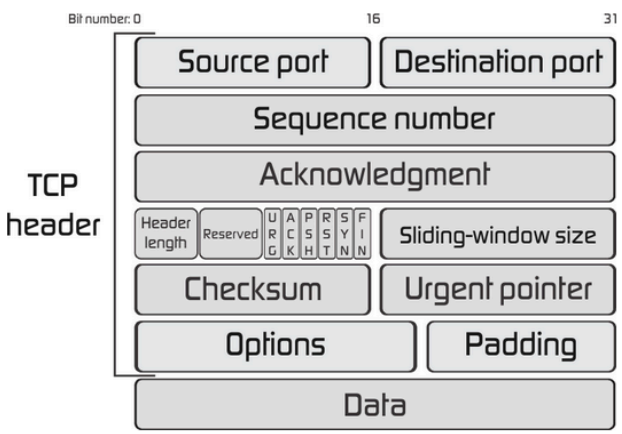
- **push**= TCP deve inviare un segmento, non completamente pieno. Serve per evitare che TCP attenda di riempire segmento.
- **urgent**= il dato urgent viene inserito nel segmento che TCP sta costruendo, senza che sia di MSS, e viene spedito così com'è. Serve ad applicazioni interattive a cui inviare un comando.

TCP include flow control e congestion control.
Ha un connection record: <ID connessione, initial seq#, window size, state, MSS, ...>

formato segmento

Il segmento TCP ha 20 byte fissi di header:

- **i numeri di porta**= degli unsigned number e si dividono in porta sorgente e porta destinazione (16 bit ciascuno)
- **numero di sequenza** (32 bit)
- **acknowledgment number**= è di tipo cumulativo, cioè memorizza qual'è il numero di byte che la destinazione sta attendendo.
- **header length**= lunghezza dell'header espressa in numero di parole da 32 bit, è specificata perché non è di lunghezza fissa.



Dice al TCP ricevente quanto devo andare avanti con le info di controllo prima che inizi il payload.
Il campo ***reserved*** che segue serve per rappresentare una dimensione di header più grande (6 bit)

- **window size**= è la dimensione della receiver window. La dimensione viene negoziata all'inizio della connessione. Può aumentare o diminuire dinamicamente in funzione del carico che ha una macchina in un dato momento. Es.: può iniziare con win=4k e diventare win=8k oppure win=2k. Tale dimensione viene sempre inviata al sender in modo da fare controllo di flusso, se invio win=0 il sender si ferma e aspetta.
- **checksum**= sono dei bit (16) di ridondanza che permettono a TCP di rilevare eventuali errori presenti nell'intestazione di un pacchetto a livello rete, durante il trasferimento tra dispositivi e controlla il payload. Non si sa quali sono i bit errati e bisogna quindi "buttare via" i segmenti.
- **urgent pointer**= dice dova si trova il dato urgent. Con code bit URG=1 setta offset(posizione) del dato urgent nel payload in modo che TCP ricevente non sia costretto a scansire tutto il payload per leggere dato urgent ma ci si arriva subito.

code bit e pseudo-header

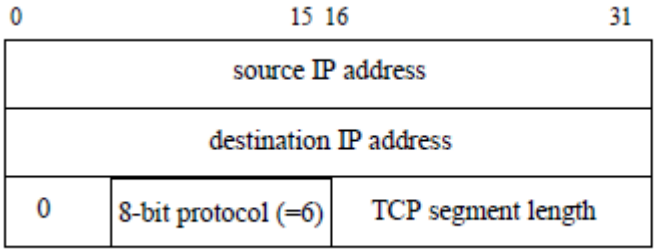
il **code bit** (o flag di controllo) contiene 6 bit di controllo, ciascuno dei quali ha un significato diverso:

1. URG: puntatore urgente valido, ci sono dei dati urgenti all'interno del payload
2. ACK: riscontro valido
3. PSH: richiesta di push
4. RST: significa che chi ha generato il segmento, sta resettando la connessione
5. SYN: sincronizzazione numeri di sequenza, si tratta di una apertura di connessione
6. FIN: chi ha generato questo segmento vuole terminare la connessione

Il **checksum** viene calcolato su una sequenza di bit che rappresentano delle informazioni collezionate nello **pseudo-header**, pseudo perchè **non** è presente nel segmento. Queste informazioni sono:

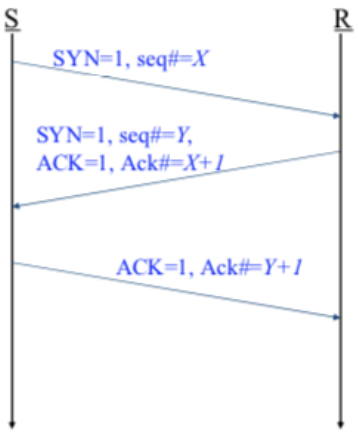
- indirizzo IP sorgente (fa parte dello header di livello 3, non è presente nel segmento TCP)
- indirizzo IP destinazione (header di livello 3)
- protocol ID (header di livello 3) (serve per fare multiplexing)
- lunghezza del segmento TCP
- header del segmento TCP, parte fissa più eventuali opzioni
- dati del payload, più un byte aggiuntivo (che può esserci o non esserci) in funzione che il numero di byte nel campo dati sia pari o dispari

Il checksum serve a TCP per verificare che i dati siano arrivati all'host giusto.



//La trasmissione orientata alla connessione nel protocollo TCP richiede 3 fasi: apertura della connessione, trasmissioni dei dati, e chiusura della connessione.

apertura connessione



Nel **connection record** è indicato il numero di sequenza iniziale, da cui viene poi calcolato lo **spiazzamento dei byte** che ci sono nei payload seguenti. I numeri di sequenza di S e R sono diversi (*calcolati con bit meno significativi dei rispettivi clock*). Viene ricordato per assicurare sicurezza e per evitare che ci sia confusione tra sessioni differenti tra un primario e un secondario oppure che una terza parte possa cercare di replicare una sessione tra sorgente-ricevente per propria utilità (malevola).

three-way handshake

La procedura di apertura della connessione in TCP è detta three way handshake. La procedura comporta i tre passi seguenti:

1. il client invia il primo segmento SYN nel quale viene impostato solo il flag SYN che serve per la sincronizzazione dei numeri di sequenza. Il client sceglie un numero random come primo numero di sequenza chiamato Initial Sequence Number (ISN) e lo invia al server. Questo non contiene dati utente, ma usa un numero di sequenza;
2. il server invia il secondo segmento, che contiene i due flag SYN e ACK e anch'esso non trasporta dati utenti, ma usa un numero di sequenza;
3. il client invia il terzo segmento ACK, che conferma l'avvenuta ricezione del secondo segmento mediante il flag ACK. Ogni processo fa riferimento al proprio clock e il counter viene incrementato ogni 4 o 8 microsecondi.

Questa procedura funziona anche in caso di malfunzionamenti o ritardi.

// secondo la versione degli appunti i passi sono:

1. S manda il primo segmento con il SYN bit alzato marcato con il proprio numero di sequenza iniziale X;

2. R riceve questo segmento e capisce che le due parti si vogliono sincronizzare, analizza i parametri proposti, algoritmi...risponde con eventualmente i propri parametri con segmento con SYN alzato e numero di #seq=Y e ACK;
3. S invia numero di sequenza X+1 con solamente l'ACK alzato.

trasferimento dati

Dopo aver stabilito la connessione è possibile trasferire i dati. La dimensione della finestra di ricezione viene aggiornata ad ogni segmento per ottimizzare il controllo di flusso (windows size advertisement). TCP lavora per cercare di riempire la MSS, eventualmente non rispettando i confini dei messaggi. Questo, ad esempio in un sistema distribuito multimediale, non è una buona soluzione e va evitata perché comporta ritardi (freeze).
Se si usano applicazioni interattive, potrebbe accadere, quindi, di inviare 1 byte di dati e 20 byte di header, e questo comporta un enorme spreco. Ci sono 3 soluzioni :

- MSS → TCP aspetta di raccogliere byte sufficienti dall'applicazione sorgente per riempire un segmento di MSS. Lento, efficace ma non interattivo.
- PUSH → l'applicazione invia pochi dati e questi devono arrivare all'altro lato subito. interattivo ma c'è comunque dello spreco.
- **algoritmo di NAGLE** = (si usa se un tipo di applicazione che lo consente, mediazione tra PUSH e MSS) il sender lento invia subito il primo carattere disponibile al receiver e, mentre attende l'ACK dal receiver, bufferizza i successivi dati che gli arrivano dal livello applicazione (es.: caratteri da tastiera). Appena arriva ACK, il sender invia intero buffer e ripete la procedura. L'algoritmo di NAGLE non va bene per inviare movimenti di mouse in remoto in quanto si potrebbe avere un funzionamento a scatti.

errore in scambio dati

TCP fa piggyback di ACK; se non ci sono dati (la comunicazione è half-duplex) manda ACK da solo (attesa fino a 200 ms per invio separato), per evitare che l'altra parte faccia una ritrasmissione inutile.
In caso di errori (non arriva un segmento o ACK) TCP ritrasmette:

- quando **scade RTO**(=retransmission timeout). Questa viene considerata una **congestione grave** (non sta passando più traffico tra le parti comunicanti).
- quando vengono **ricevuti 3 ACK consecutivi** con lo stesso ACK#. Il sender riceve 3 ACK con stesso numero quando il receiver sta ricevendo qualcosa, ma ha perso un segmento e gli stanno arrivando i successivi. Questa viene considerata una **congestione lieve**. Con "fast retransmit" ritrasmetto solo il segmento atteso segnalato da ACK#.

timer di ritrasmissione

Una stima del RTT viene calcolato dinamicamente controllando volta per volta quanto tempo passa tra l'invio di un segmento e la ricezione del su ACK. Si può calcolare la media con una formula di media smorzata:

$$RTT_{new} \leftarrow \alpha \cdot RTT_{old} + (1-\alpha) \cdot M$$

α è un peso (la normalizzazione della storia precedente, riassunta nel RTT), il valore ottimale di α è di circa 0,9 (7/8).
M è la misura del tempo trascorso tra l'invio dell'ultimo segmento e la ricezione del suo ACK. (Si da più peso ad α che ad M perchè potrebbe essere un outlier).
La misura M viene presa soltanto se l'ACK è ricevuto dopo la prima trasmissione del segmento. (Se viene ritrasmesso il medesimo segmento, quell'M non è valido in quanto non si potrebbe sapere se l'ACK che torna indietro si riferisce al primo invio o al secondo).
In caso di reinvio dello stesso segmento, il RTT non viene aggiornato ma, secondo l'algoritmo di Karn, si aggiorna il RTO, raddoppiando il suo valore.
L'algoritmo di Jacobson dice che non basta prendere RTO circa uguale a RTT ma è meglio usare la varianza per calcolare RTO:

$$D_{new} \leftarrow \alpha \cdot D_{old} + (1-\alpha) \cdot |RTT - M|$$

$$RTO \leftarrow RTT + 4 \cdot D_{new}$$

D è la varianza (o deviazione standard).
La soluzione è determinata considerando quanto stabilito dallo RFC 2988, secondo il quale:

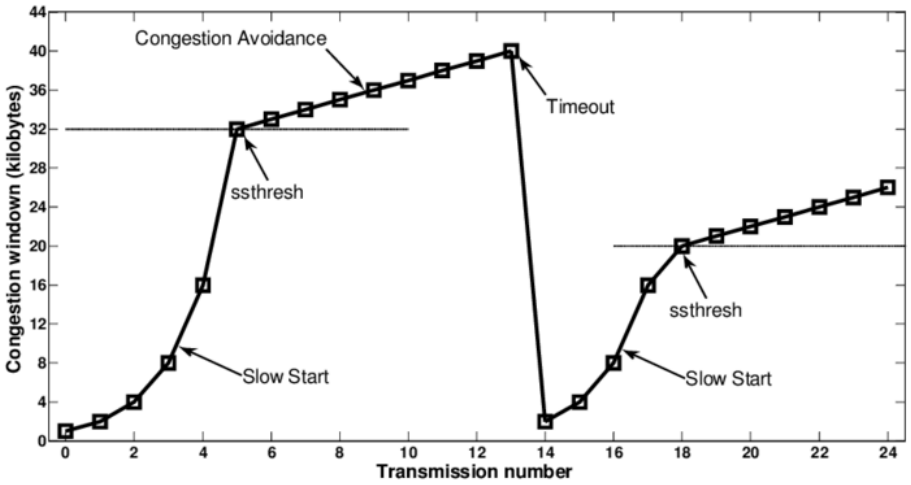
1. inizialmente RTO è posto uguale a 3 sec.
2. dopo la prima misura M si pone : RTT = M; D = M/2; RTO = RTT + 4D = 3M
3. per le misure successive l'aggiornamento è eseguito come sul libro di testo.

Nel caso si verifichi un errore, RTT non viene più aggiornato mentre RTO è raddoppiato. Quando supero la condizione di errore, invio un nuovo segmento e per esso ottengo una misura, allora RTT, D e RTO vengono reinizializzati come descritto al punto (2) precedente.

congestion window (algoritmo)
TCP cerca di non congestionare la rete. Il controllo di congestione è implementato grazie all'algoritmo **slow start** (input = comportamento della rete; output = congestion window).
Ci sono due fasi:

1. `slow start: cwnd++ /* cwnd = congestion window */`
2. `congestion avoidance: cwnd ← cwnd + (1/cwnd)`

La congestion window (o finestra di congestione) è solo un numero e viene calcolato con l'algoritmo di slow start. Questo numero puro viene considerato per determinare la taglia della finestra di invio **sendwindow=min{recv window, congestion window}**.
All'inizio della connessione la CW(congestion window)=1 e la send_window=1MSS. Si invia 1 segmento e se arriva ACK la finestra raddoppia e, se inviano 2 segmenti e se arrivano i 2 ACK, si raddoppia ancora (c'è una crescita esponenziale) fino alla soglia di **slow start threshold** (=SST). Da SST la crescita diventa lineare. Dopodichè c'è un'altra soglia che fa in modo che non cresca più.
La SST e la dimensione massima della finestra di invio vengono decise dal sistema, indipendentemente dalla dimensione della recv_window del ricevente. Per TCP max_send_window è 64K.



situazione di errore

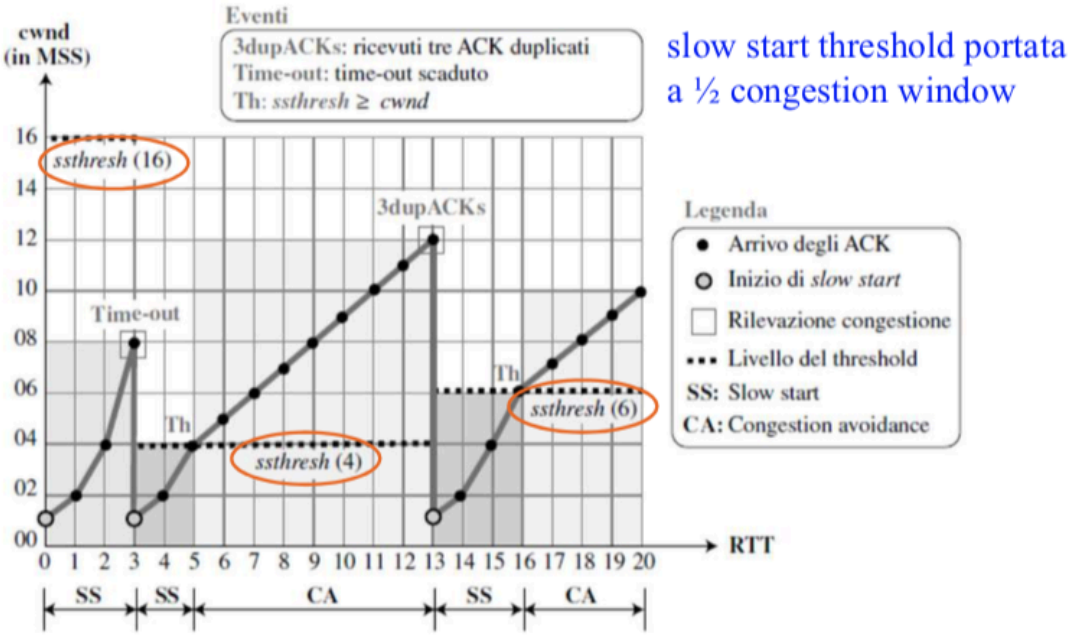
L'algoritmo slow start prevede che la finestra possa crescere liberamente fino alla seconda soglia (poi si appiattisce) ma la rete può congestionare prima, in due modi:

- congestione **lieve**: i messaggi passano, ma si fermano nei buffer e arrivano fuori ordine, questo è segnalato dalla ricezione di (3) ack duplicati.
- congestione **grave**: i messaggi non passano o vengono buttati via per buffer overflow. Questo è segnalato dalla scadenza del timer di ritrasmissione.

Diverse versioni di TCP reagiscono in modo differente a fronte di questi eventi.

gestione errori in TCP Taho

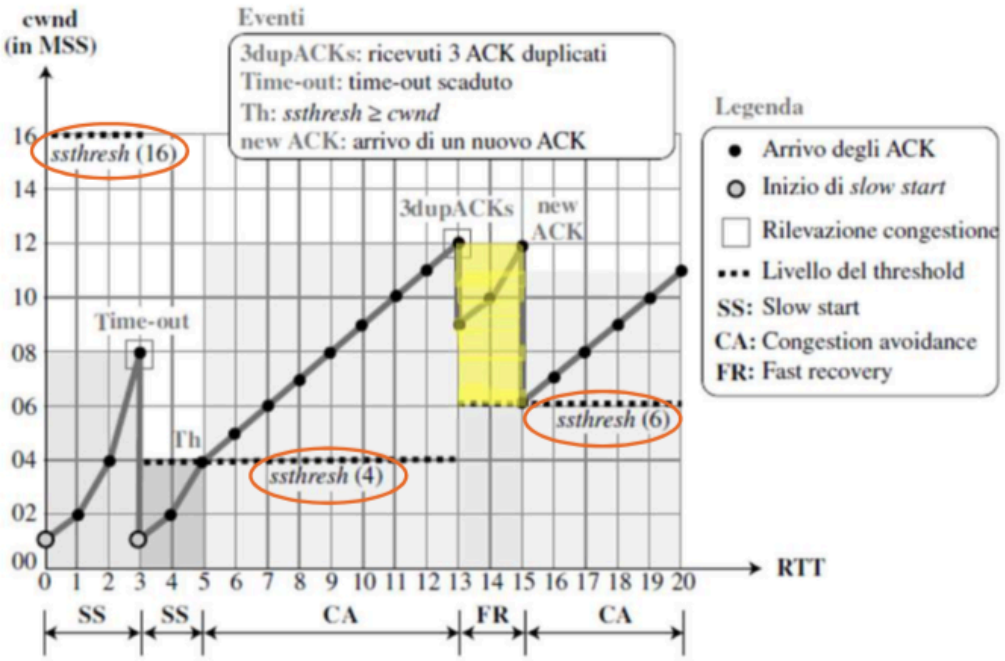
TCP Taho= prima versione di TCP, utilizza due algoritmi per la gestione di congestione. Si parte con *slow start*. Quando si arriva a congestione, TCP fa ripartire tutto impostando sstresh=cwnd/2. Se non c'è congestione fino a sstresh si passa a *congestione avoidance* (finché non finisce la connessione o c'è una nuova congestione). Se avviene congestione TCP imposta sstresh=cwnd/2 e riparte con *slow start*.



gestione errori in TCP Reno - Fast recovery

TCP Reno= al timeout (cioè quando non riceve riscontro), TCP passa a *slow start*. Ai tre riscontri duplicati TCP passa a fare *fast recovery* finché non arrivano altri riscontri duplicati (con **fast recovery**, la ssthresh=dimensione finestra di invio /2 e il valore di cwnd diventa sstresh+3 MSS). Quando TCP entra in *fast recovery* si possono verificare tre situazioni:

1. se continuano ad arrivare riscontri duplicati, cwnd cresce esponenzialmente
2. se arriva il timeout, TCP passa a *slow start* (si comporta come TCP Thao)
3. se arriva un nuovo riscontro, TCP passa a *congestion avoidance* (crescita lineare) e la dimensione di cwnd diventa = ssthresh.



chiusura della connessione

Sia il client che il server possono chiudere la connessione (solitamente è il client). Quando si verifica una richiesta di chiusura della connessione TCP potrebbe non aver ancora finito di trasmettere/ricevere. Una chiusura drastica cancellerebbe tutte le informazioni sulla connessione (anche dati sul buffer di invio che devono essere mandati, prima bisogna accertarsi dello stato dati pendenti).

Ci sono due opzioni di chiusura di connessione di TCP:

- **chiusura simmetrica**= si usa una sorta di **handshaking a tre vie**: il client (che vuole forzare la chiusura della connessione) invia un segmento con flag FIN=1 (che indica la chiusura) e #seq successivo al numero di byte inviato, il server TCP utilizza un segmento FIN+ACK e #seq adeguato. Il client TCP invia l'ultimo segmento ACK. Eventuali dati ancora in buffer di invio possono essere flushed (=TCP manda gli ultimi dati che ha nel buffer al server).
- **chiusura asimmetrica**= si usa una sorta **handshaking a quattro vie** con opzione **half-close**: uno dei due processi può smettere di inviare dati mentre ne sta ancora ricevendo (chiusura a metà). Ad esempio, ciò è necessario nell'ordinamento (client manda dati e server li ordina quando li ha tutti).

Il rilascio deve avvenire da entrambe le parti.

rilascio di connessione (errore)
Durante la disconnessione, potrebbero perdersi i dati di chi ha iniziato la chiusura. Per tale motivo l'iniziatore della chiusura (**active close**) fa partire un timer e manda FIN=1 e #seq. Se non riceve un ACK rimanda FIN=1e #seq. Quando scade il timer fatto partire alla chiusura attiva l'iniziatore chiude la connessione.
Quando un processo è coinvolto in una connessione e riceve dati, fa partire un timer di attesa detto **timer di keepalive** e se entro un certo tempo, ovvero 2 ore, non arrivano più dati dal sender, il receiver manda l'ultimo ACK che ha mandato al primario, se ancora non c'è una risposta per 10 tentativi il secondario chiude la connessione.
Dopo la chiusura di una connessione si attende per 30-120 secondi per evitare che segmenti provenienti da una vecchia connessione possano interferire con una nuova connessione, magari perchè nuova connessione ha medesimi parametri (es.: numero di porta).

altre caratteristiche di TCP
persistence timer= serve a TCP per evitare le situazioni di stallo (o **deadlock**). Quando receiver è congestionato, manda a sender **window=0** e sender si ferma. Quando receiver ha spazio, aggiorna la win size; ma se la nuova taglia della win si perde succede che receiver attende il sender e viceversa. La soluzione è che se sender riceve win=0 fa partire un persistence timer e dopo 60 secondi è il sender che invia al receiver una ***window probe***, ovvero un l'ultimo segmento che ha inviato, il ricevente manda un ACK nel cui header risiede anche la win size.
silly window= La finestra di ricezione si riempie e decresce lo spazio disponibile. Ha un certo punto il receiver invia al sender un ACK+informazione sullo spazio residuo della sua finestra=1 byte. Il sender mada 20 byte di header + 1 byte di payload. Il ricevente allora manda ACK e la sua finestra è 0 fino a che l'applicazione di sopra non fa liberare un po' di spazio. L'applicazione è lenta e tira fuori solo 1 byte. La situazione si ripete.
L'algoritmo di Clark dice: il receiver deve inviare un window update solo quando ha spazio libero in finestra ≥ min{ MSS, reciver_buffer/2}.

UDP - User Datagram Protocol

UDP è un protocollo di tipo:

- connectionless (non garantisce ordine o affidabilità) tutti i messaggi che arrivano dalla sorgente per la rete sono scorrelati
- privo di controllo di flusso e privo di controllo di congestione

I messaggi sono di qualsiasi dimensione, non si interessa alla frammentazione. Non aggiunge nulla ai servizi di IP, eccetto la comunicazione tra processi, ovvero l'indirizzamento di livello 4 e aggiunge il checksum (bit di ridondanza per verificare la correttezza dei dati).
È adatto, tuttavia, per alcune applicazioni. Ad esempio, **multicast**, ovvero intermedio tra unicast (la sorgente comunica con una sola destinazione), broadcast (la sorgente comunica con tutte le destinazioni). Si parla con un gruppo di apparati con un indirizzo di rete comune e che possono stare pure in reti differenti.
TCP non va bene per multicast, perché le destinazioni possono essere distanti e pure su reti differenti. Le prestazioni di rete possono variare molto. TCP farebbe molta fatica a gestire tutte queste differenze.
UDP, best effort, è leggero e semplice, così da non sovraccaricare nessuno. Analogamente va bene per il multimedia (dati video, audio e real time). TCP è sostanzialmente troppo pretenzioso, per esempio nello streaming (real time).
Rispetto a TCP, aggiunge indirizzamento di livello 4 (quello di livello 3 permette solo di trovare end system, quello di livello 4 di indirizzare una specifica applicazione: multiplexing e demultiplexing): fa come TCP aggiunge **checksum**, ovvero dei bit di ridondanza che verificano che quello che è sbucato fuori dalla rete e che è arrivato al livello 4 dal 3 sia corretto.
Le primitive di servizio sono *socket()*, *bind()* e *close()* come per TCP, ma, essendo una socket in UDP non connessa chiede che, ogni volta che è inviato un messaggio, deve essere specificato il destinatario, quindi si aggiunge *sendto()* e *receivefrom()*.

UDP – formato header

Sono 8 byte contenenti:

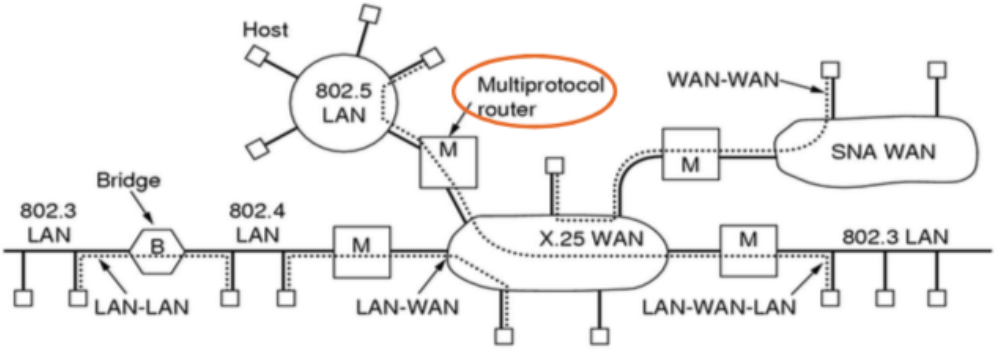
- 2 byte contengono la porta sorgente
- 2 byte contengono la porta destinazione
- 2 byte contengono la lunghezza dell'intero datagramma UDP (diverso da TCP)
- 2 byte contengono il checksum (calcolato sullo pseudo header che considera TCP)

- max datagram size = 65535 B, IP header = 20 B
- max UDP message = 65535 - 20 - 8 = 65507 B

Network Layer

Il livello di rete:

- deve interconnettere reti diverse, come dorsali (le reti proprietà di ISP nazionali, regionali o internazionali) e reti di accesso (reti di organizzazioni cittadine, LAN, WAN)
- è il più alto livello di astrazione, e di conseguenza può vedere apparati che usano protocolli diversi in quanto devono instradare su reti differenti, come: protocolli dello stack TCP/IP, DECnet (=Digital Equipment Corporation, al confine di una rete DECnet servivano apparati HW commutatori come i **multiprotocol router** che traducesse il formato delle protocol data unit da uno stack di protocolli al TC/IP a un'alltro stack di protocolli, quello proprietario degli apparati DECnet), Novell, ATM (=Asynchronous Transfer Mode, stack di protocolli pensato per lavorare con fibre ottiche, progetto poi fallito), AppleTalk (stack di protocolli proprietari di Apple),etc..



funzionalità

routing= I router sono gli apparati che hanno il compito di mettere insieme tutte queste cose. I router multiprotocollo sono interpreti tra le diverse strutture di protocolli delle reti. In più fa instradamento, ovvero routing (trova la rotta) e studia la topologia della rete (la sua rappresentazione come grafo). Il livello rete, tramite router, fa **fragmentation**, poiché tutte le reti non possono supportare tutte la stessa dimensione (65535 dimensione massima).
addressing= al livello 4 (trasporto) l'indirizzamento si fa tramite i numeri di porta per distinguere processi diversi che si trovano sullo stesso end-system mentre al livello 3 (rete) si usano gli indirizzi IP.
monitoraggio della rete e controllo degli errori = ICMP è un protocollo di supporto di IP.
traduzione tra reti diverse = Al livello 3 è possibile fare traduzione di indirizzi di rete e indirizzi MAC o HW (livello 2 o 1). Fa anche traduzioni tra stack e protocolli differenti usati in reti diverse e cambia il formato degli header e degli indirizzi. Può anche tradurre in situazioni di sicurezza, cifratura, QoS, multicast/broadcast, etc..

gateway

Il gateway è un router (apparato di livello 3), che sa capire anche protocolli di altri livelli, magari di livello 4 (o più alto). Capiscono codifiche audio e video, che stanno più in alto. Esistono due tipi di gateway :

- **full gateway**= connesso a reti che parlano protocolli differenti, fa traduzione tra formati di rete per tutte le reti che sono connesse.
- **half gateway**= sa fare solo due traduzioni: il linguaggio della rete in cui si trova e un linguaggio universale. Cos'è questo "esperanto"? È quello di TCP, UDP e IP.

//esempio
Supponiamo che un **full gateway** connetta reti DECnet, X.25 e Internet; allora deve tradurre nei due sensi:

DECnet <---> X.25,
DECnet <---> Internet,
X.25 <---> Internet,

ovvero deve saper fare 6 traduzioni.
Uno **half gateway** sta solo su una rete e su Internet. Quindi nell'esempio precedente avremmo bisogno di 2 half gateway:

- il 1° connesso alla rete DECnet e a Internet,
- il 2° connesso alla rete X.25 e a Internet,

e con Internet che permette la connessione tra i due attraverso di sé.
Il 1° half gateway sa tradurre nei due sensi:

DECnet <---> Internet,

il 2° traduce:

X.25 <---> Internet,

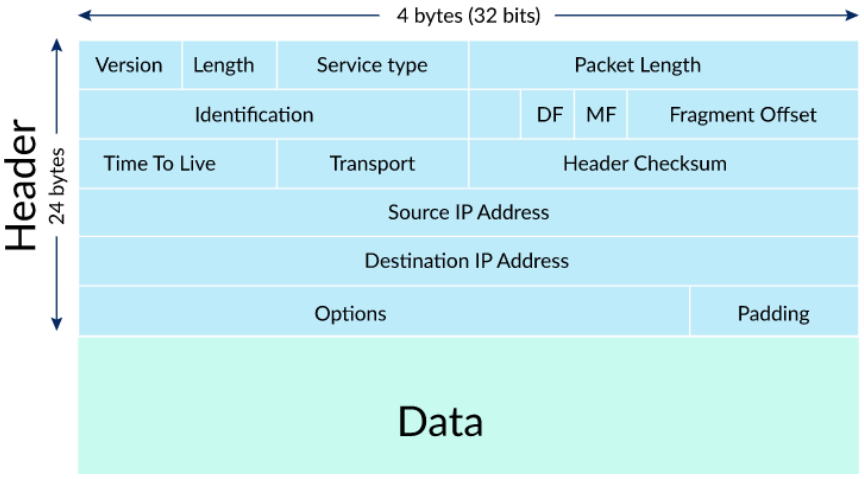
e ognuno dei due deve fare solo 2 traduzioni. Quindi sono meno complessi, di uso più generico, e decisamente meno costosi.

IP - Internet Protocol
Protocollo di livello rete per lo stack TCP/IP (di Internet). IP fornisce un servizio su reti interconnesse, best effort. I datagrammi hanno dimensione fino a 65535 byte (64KB). Fa instradamento attraverso una struttura gerarchica. La rete utilizza il Big Endian (byte più significativo a sinista).

Formato di datagrammi IPv4
I pacchetti utilizzati da IP sono detti datagrammi. Sono di lunghezza variabile, composti da due parti: header (intestazione) e campo dato (payload). L'intestazione è da 20 a 60 byte e contiene le informazioni essenziali per il routing e la consegna.
Un datagramma è composto da:

1. **versione** (4 o 6);
2. **lunghezza** intestazione (header);
3. **Type Of Service**= TOS; è un ottetto e gli otto bit si distinguono:
    a. 3 per la priorità (con cui il router ritrasmette il pacchetto sull'interfaccia di uscita o con cui scarta il pacchetto),
    b. 5 per capire se i dati vogliono un cammino con basso ritardo, alto throughput, o basso costo (di cammino), inutilizzato;
4. **lunghezza totale** del pacchetto (16 bit);

5. **identificazione** (16 bit), **flags**, **fragment offset**: costituiscono la frammentazione dei datagrammi.
    a. identification= identificatore comune a tutti i frammenti ottenuti da una protocol data unit di livello più alto se quella PDU è troppo grande per stare nei pacchetti IP.
    b. I flag sono **DF**=don't fragment (di default alzato) e viene usato per i segmenti TCP.
    c. **MF**=more fragment; indica se ci sono altri frammenti.
    d. **Fragment offset**= è la posizione del 1° byte del frammento nel messaggio originario (indica quale posizione bisogna occupare);
6. **Time To Live**=TTL, numero di router visitati per sapere quando scartare il datagramma (serve ad evitare che un pacchetto giri per la rete per sempre, arrivato a 0 il pacchetto viene scartato);
7. **protocollo** del livello superiore: serve per sapere a chi consegnare il payload estratto dal datagramma;
8. **checksum**= controllo errori solo dell'intestazione, deve essere ricalcolato a ogni hop poichè l'header cambia in quanto il campo Time To Live viene decrementato a ogni hop;
9. **indirizzi sorgente e destinazione**;
10. **opzioni**= sono rappresentate in notazione **TLV** = Type(1 byte dell'opzione), Length(1 byte del campo), Value(valore del campo).



Ci sono diverse opzioni:

- **security**= specifica quanto è riservato il datagramma.
- **source routing**= invece di avere la rotta calcolata da algoritmi, la sorgente del pacchetto dice quale rotta deve fare all'interno della rete. Si richiede che la applicazione sorgente conosca la topologia della rete. Si può fare in due modi:
    - **strict**= il cammino indicato dalla sorgente specifica tutti gli hop che devono essere fatti. C'è anche un puntatore al prossimo elemento.
    - **loose**= lascia libertà al pacchetto di fare il cammino che vuole, passando per dei punti di controllo. Segue gli IP, non il cammino.
- **record route**= ogni router aggiunge il proprio IP address.

- **timestamp**= il valore del proprio clock nel momento in cui è ricevuto e gestito il pacchetto.

Type of service (TOS byte): non è mai stato usato, in ipv6 è stato rimosso.

fragmentation

Reti diverse possono avere un diverso upper bound sulla taglia del pacchetto. Ci sono due opzioni:

1. **frammentazione non trasparente**= i frammenti rimangono frammentati finchè non raggiungono la loro destinazione.
2. **frammentazione trasparente**= ogni volta che i frammenti arrivano a un gateway vengono riassemblati.

IP addressing

Gli indirizzi IP sono rappresentabili in due modi: la rappresentazione utilizzata dal livello di rete è su 32 bit. In realtà li rappresento in rappresentazione decimale puntata. I 32 bit sono divisibili in quattro ottetti. Si separano con punti (i valori sono compresi tra 0 e 255).
Gli indirizzi devono essere unici (altrimenti non è più possibile l'instradamento). Con i DNS più moderni è possibile fare corrispondere un nome simbolico a più indirizzi IP, per bilanciare tra più server le richieste inviate al server, quindi il carico.
ICANN si occupa degli indirizzi, IANA, una sua sottodivisione, si occupa dei numeri, si occupa più da vicino della gestione degli indirizzi IP.
Oltre all'indirizzo IP e al nome simbolico c'è l'indirizzo di livello due, data link, che ha 48 bit, sulla scheda di rete. Questo si chiama MAC address o hardware address.
L'IP, di livello 3, ha una struttura (a differenza di MAC che è piatto):

- Network ID= id di rete (bit significativi);
- Host ID= id di host (meno significativi).

Quanto spazio dare a questi due id? Ci sono 5 schemi di assegnazione di indirizzi:

- **class based addressing**= è il più vecchio
- **subnetting**= serve quando ci sono reti grandi dove si trova un grande numero di apparati. Il subnetting permette di individuare una struttura logica all'interno della rete, dividendo la rete in sottoreti caratterizzati dal loro subnetwork id.
- **classless addressing** (**CIDR**)= recente, ha sostituito il class based addressing. Permette di usare meglio gli indirizzi di rete disponibili (che in IPv4 sono 2^{32}).
- **network address translation** (**NAT**)= si risparmia al massimo sul numero di indirizzi di rete assegnati.
- IPv6: indirizzi di 128 bit

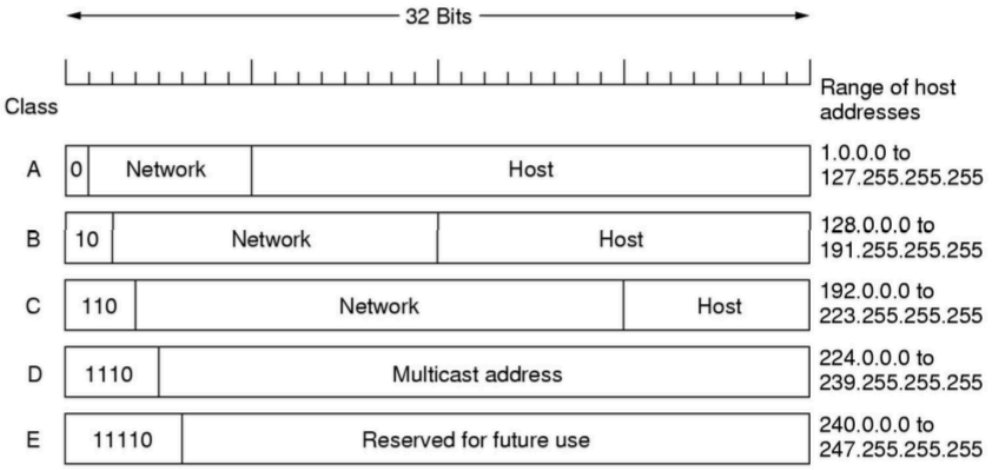
class-based addressing

Esistono delle classi di indirizzi predefinite per cui un indirizzo può avere solo la struttura determinata da quelle classi.

- Quando i bit nel netID sono diversi da 0 e hostID sono a 0, si indica la rete stessa, ovvero l'indirizzo di base.
- Al contrario, netID=0 e hostID è diverso da 0, allora si indica un determinato host che appartiene alla rete in cui ci troviamo.
- Se tutti i bit sono 1, allora questo è l'indirizzo **broadcast** della rete su cui ci troviamo (si vuole inviare un pacchetto a tutti gli apparati di livello 3 che stanno sulla stessa rete).
- Se invece netID non è tutti 0 o tutti 1, e la parte hostID è tutto 1, allora indico un **pacchetto** che deve andare in broadcast sulla rete destinataria.
- Infine, se i bit sono tutti a 0 questo è un **wildcard** indica qualunque indirizzo e non può essere assegnato a nessun host (es.: può indicare la disponibilità di un server con più interfacce di rete attive a ricevere richieste dai client su tutte le sue interfacce, qualunque sia il loro indirizzo).

Le classi di indirizzi sono:

A. 1° bit a 0 e i restanti 7 bit del primo ottetto vengono usati per il netID, il resto per lo hostID. 128-1 reti al mondo possono avere un indirizzo di classe A. Possono contenere 2^{24}-1 router o host. Da 1.0.0.0 a 127.255.255.255.
B. 14 bit (2 per identificazione delle reti, 10). 2^{14}-1 reti da 2^{16}-1 router o host. Da 128.0.0.0 a 191.255.255.255.
C. 21 bit (3 bit per id delle reti, 110). 2^{21}-1 reti da 255 router o host. Da 192.0.0.0 a 233.255.255.255
D. (1110) non è divisa in prefisso e suffisso ed è utilizzata per gli indirizzi di tipo multicast. Da 224.0.0.0 a 239.255.255.255
E. (11110) riservata per l'uso futuro. Da 240.0.0.0 a 255.255.255.254.



Una entry di routing table è fatta da:

- **campo destinazione**,
- **netmask**= un meccanismo utilizzato con le classi in modo tale da ridurre la lunghezza dell'informazione di instradamento. *Funzionamento*: il router riceve un

pacchetto destinato a una determinata rete, rete in cui lui non è. Estrae dallo header IP del pacchetto l'indirizzo IPv4 del campo destinazione. Incomincia a percorrere le righe della sua tabella di instradamento e per ognuna prende l'indirizzo destinazione del pacchetto e lo mette in end bit a bit con la netmask, un insieme di 32 bit, e controlla se il risultato è uguale al valore che in quella riga c'è per il campo destinazione. Se il risultato è uguale allora il pacchetto deve essere inoltrato sulla interfaccia indicata come next hop (o come output interface) per andare verso la sua destinazione (a un router che non è nella rete basta ricordare una sola riga nella tabella di instradamento per tutti gli apparati di un'altra rete).

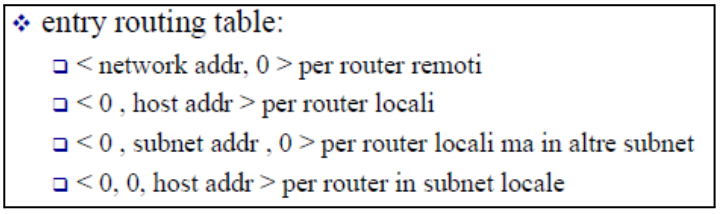
- **identificatore di next hop**,
- **informazione di metrica** (quanto costa viaggiare fino alla destinazione),
- **insieme di flag** (caratteristiche del cammino).

Perchè suddivisioni in classi? Per fornire (affittare) indirizzi IP in base al numero di host che possiede una LAN.
Il problema di questo tipo di indirizzamento è che si sprecano parecchi indirizzi (es.: si hanno 300 host, cosa si fa? Se si sceglie una classe B, si può fare broadcast su tutti mettendo tutti i bit a 1, ma si sprecano (65535-300) indirizzi. Se si usa una classe C è troppo piccola, solo 255 host. Le classi A e B sono molto grandi e la C è molto piccola).

subnetting

Una soluzione per ovviare allo spreco indirizzi è fare subnetting. Serve per suddivisione interna di una rete di classe A o B in sottoreti. Gli ultimi due ottetti vengono divisi e nei bit più significativi (a sinistra) si utilizza un subnetID, a destra HostID. Ciascun router non deve più ricordare tutte le righe della tabella di instradamento, ma solo quelle associate a una subnet.
Si risparmia anche spazio per le tabelle di routing, dato un indirizzo IP, assegnato da ICANN, usa una parte dei bit che identificano l'host come maschera di bit; quindi si fa routing a 3 livelli: network+subnet+host.
Per capire a quale sottorete deve venire indirizzato il pacchetto, viene fatto un and logico con la subnet mask, formata da:

- tutti 1 nella porzione scelta per il SubnetID
- 0 altrove



CIDR

=Classless Inter-Domain Routing. CIDR è stato creato per riuscire a vendere indirizzi di classe C che nessun amministratore voleva perché erano ritenuti troppo piccoli (solo 255 host).
Il CIDR usa un particolare schema di indirizzamento, con indirizzi di 4 ottetti **x.y.w.z./n** (n=numero di bit usati per il netID).

L'idea è di raggruppare classi C in insiemi da usare come spazio contiguo di indirizzi (in questo modo si possono avere più di 255 host). Vengono assegnati un range di indirizzi alle varie nazioni (es.:Europa da 194.0.0.0 a 195.255.255.255).

NAT

=Network Address Translation. NAT è un altro sistema per usare meglio gli indirizzi di rete. Gli host possono assumere qualsiasi indirizzo essi vogliano. C'è un gateway unico che fa da confine tra la rete interna e la rete esterna. Il compito del NAT è di trovare corrispondenza tra indirizzi esterni e interni. Gli host nella rete interna assumono l'indirizzo che vogliono, tanto all'esterno non lo vedono. Al di fuori si vede un solo indirizzo unico. C'è un indirizzo per organizzazione.
Nota: vengono distinti indirizzi TCP e UDP.
Come si trova la corrispondenza tra indirizzo esterno e interno? Il NAT si presenta all'esterno con IP address e un numero di porta. Sul gateway d'accesso, di confine, mantiene una tabella NAT in cui ricorda una quadrupla di corrispondenza:

`[<IPA,port#A >,<IPX ,port#AX>]`

- IP address dell'host interno,
- numero di porta del processo su quello host (interno che manda verso l'esterno),
- IP address pubblico del gateway NAT che mostra in Internet,
- numero di porta con cui si mostra all'esterno.

Modifica gli header del pacchetto uscente con suo il suo indirizzo IP pubblico e il suo numero di porta (nota: quindi *NAT sa lavorare a livello 3 (IP) e 4 (TCP))* e inoltra il pacchetto. Quando il pacchetto rientra da Internet, grazie alla relazione creata in precedenza, sostituisce l'indirizzo IP pubblico con quello dell'host e il numero di porta di NAT con quello originale dell'host e spedisce il pacchetto all'host di destinazione.

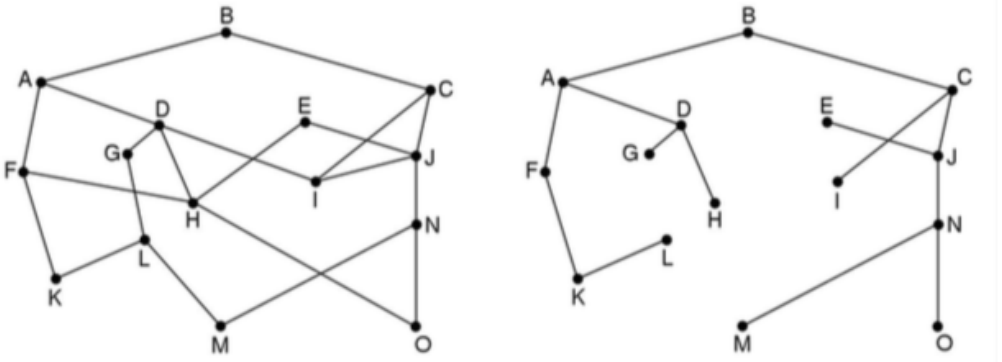
IP routing

Il problema del routing è quello di trovare il cammino “buono” nella rete per portare traffico da sorgente a destinazione. Questo si può trovare tramite diverse metriche: numero di hop, banda sul link, latenza. L’algoritmo che serve a trovare il cammino e il cammino stesso devono soddisfatte alcune proprietà:

- **correttezza**: path fornito deve esistere, no loop;
- **semplicità**: di computazione;
- **robustezza**: ci possono essere dei cambiamenti di topologia;
- **stabilità**: sembra in contraddizione col punto di prima. Non deve cambiare in continuazione il cammino;
- **fairness e ottimalità**: deve esserci un equilibrio tra questi due punti. Se tutto il traffico sceglie le rotte migliori (ottimalità), queste peggiorano.
- **politica statica oppure adattiva**: la statica non è una vera politica. La scelta è cablata e configurata dall’amministratore di rete. Le rotte non cambiano, anche se succedono guasti o cambi nella rete. Adattiva invece fa sì che gli algoritmi si adattino alla situazione della rete in ogni istante.

Non ci deve essere una singola entità centralizzata a occuparsi di routing, ma diversi router. Nell’header IP c’è il *source routing*, la possibilità di dire che cammini deve effettuare il datagramma. Tuttavia, non viene mai quasi adottato. Nella realtà nei router ci sono le **tabelle di instradamento** che si occupano di fare instradamento.
Come si deve comportare la globalità dei router e come si devono comportare le tabelle di instradamento? Ad esempio, se ci fossero contraddizioni tra le tabelle, un pacchetto rischierebbe di andare in loop. Ci deve quindi essere correttezza.

principio di ottimalità
Si immagini una struttura dati ad albero: se il router J è sul cammino ottimale tra S(=sender) e D(=destination), allora il cammino ottimale tra J e D si sovrappone a quello da S a D. (Ottimale non significa ottimo, significa che si è stato trovato il cammino minimo locale). Le decisioni che i diversi router prendono riguardo al miglior cammino sono coerenti.
Sink tree= è un albero privo di cicli, che raccoglie l’insieme dei cammini ottimali da tutte le sorgenti ad una destinazione. Il principio di ottimalità per qualsiasi destinazione, trova un sink tree.



routing statico

Gli amministratori di rete configurano manualmente le tabelle di instradamento nei router della loro rete indicando, per ogni destinazione, l'output interface che il router deve usare per inoltrare i pacchetti. Se qualcosa cambia bisogna nuovamente cambiare manualmente le tabelle. Non va bene per reti ignote o di grandi dimensioni.

Flooding

I pacchetti finiscono dappertutto. Vengono percorsi tutti i cammini esistenti, a parte quelli già fatti. Ovviamente c'è il problema della terminazione (TTL). Può diventare $O(n^2)$, quindi costoso in termini di banda. E' robusto e finchè non conosco la topologia della rete è ottimale, tra tutti i cammini prima o poi trova il cammino ottimo.
Per evitare di fare sempre flooding e inondando la rete, si potrebbe tenere traccia dei pacchetti in modo da inviarli all'uscita giusta. *Dopo quanto elimino la storia dei pacchetti già inviati?* Dopo un **tempo uguale a 2 volte il diametro della rete**, dove il diametro della rete è la distanza tra le 2 stazioni più lontane nella rete.
Si può usare anche **Selective Flooding**: output interface che più o meno vanno dalla parte giusta. I router non devono conoscere tutta la topologia ma qualcosa. Nelle rete wireless e mobili è una politica adottata spesso.

approccio Distance Vector - Bellman Ford

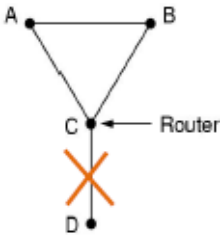
Non fa bilanciamento del carico, non è *robusto* in quanto se un router cade deve aspettare di imparare un altro percorso, è un algoritmo di tipo **greedy** quindi non è detto che riesca a determinare il cammino ottimo ma viene usato in internet perchè costa poco.
Dato un grafo (che rappresenta la rete), se due nodi sono direttamente connessi da un arco, allora la distanza tra i due nodi x e y è uguale al costo dell'arco che congiunge x e y. Se non sono vicini, allora la distanza tra x e y è il minimo su ogni y nel vicinato di x del costo per andare da x al suo vicino i più il costo (distanza cumulativa) tra il vicino i e la destinazione y.
D_{xy}= min per ogni *i* appartenente al vicinato di (x) $\{(c_{xi} + D_{iy})\}$.
RIP= Routing Information Protocol in Arpanet. Implementa l'approccio Distance Vector.

- si ricorda che ogni entry di una routing table ha formato <dest, netmask, oif (=output interface), metric, flag>.
- con Distance Vector **non** si conosce la topologia della rete. Si trovano dei cammini sub-ottimi (trova minimi locali).
- basso overhead.

Ogni nodo impara a conoscere i suoi vicini e si scambiano indirizzi, poi ogni router scambia tabelle col suo vicino. Le tabelle contengono metrica (distanza, tempo, etc..) e nome del nodo che consente di sapere a chi passare il pacchetto per raggiungere la destinazione (un router a distanza K da un nuovo nodo N impara path a N in K scambi). Ogni nodo conosce **solo** suoi vicini. Ogni intervallo di tempo i nodi si scambiano tabelle nell'eventualità di un cambio di topologia rete.
DV impara rapidamente nuovi path migliori ma reagisce lentamente a problemi.

Problemi: **Count-to-infinity.** Se, ad esempio, il cammino per un nodo sparisce. Se cade un nodo, non se ne accorge subito in quanto deve aspettare di venire informato dai vicini. Questa informazione si propaga per tutti i router ed alla fine tutti capiscono che il nodo non esiste più. Si spreca molto tempo.
Per risolvere il Count-to-infinity esistono due strategie:

- **Split Horizon** = Il nodo invia al vicino X le informazioni di rotta, nella propria tabella routing table, relative a destinazioni per cui *non* è usato X come next hop. Questo per evitare di tornare indietro a quelle destinazioni nel caso si perdesse il cammino.
- **Poisoned Reverse** = Strategia alternativa. Si comunica ad un nodo che il cammino che si ha verso quella destinazione è pari a infinito, in modo che il nodo penserà che la destinazione non sia raggiungibile da quel nodo. Può fallire nel caso di un loop.



In sintesi:

- DV ha problema di count-to-infinity;
- può considerare una metrica alla volta, tutti i router si passano dei vettori di distanza in cui il costo è determinato secondo la stessa metrica;
- in funzione della metrica, può convergere lentamente (es. bwth);
- può trovare path sub-ottimi;
- ogni nodo dice ai suoi vicini le informazioni su tutti.

Approccio Link State

Alternativa al DV. Questo approccio trova cammini *ottimi* e può usare più metriche alla volta.
Funziona secondo questo schema :

1. ogni nodo scopre informazioni sui propri vicini = si inizia con l'invio del pacchetto Hello (indicando il proprio IP)= su ogni linea point-to-point. Chi riceve risponde con il proprio indirizzo IP unico. Se ci sono LAN a cui sono connessi più router vengono visti come un nodo fittizio. Per completare il primo passo i nodi devono scoprire i diversi RTTP (costi), ognuno invia invio pacchetto di ping, che sono dei pacchetti di echo del protocollo ICMP.
2. ogni nodo dice a tutti le informazioni sui suoi vicini = (non ogni nodo dice ai suoi vicini le informazioni su tutte come in DV) questo avviene tramite flooding. Un nodo manda le seguenti informazioni: <**src**, **#seq** (aggiornato ogni volta che manda informazioni al proprio vicinato), **age** (quanto tempo mantenere le info), **neigh_id** e **metrica vicino**.
3. ogni nodo impara l'intera topologia.
4. ogni nodo calcola lo **shortest path** per ogni altro nodo (**Dijkstra**). È possibile che i nodi trovino delle inconsistenze mentre calcolano la topologia (loop).

Link state pkt usano strategia flooded: ogni nodo ricorda pkt ricevuti e scarta duplicati e versioni (#seq) obsolete.
Problema: mentre i pacchetti di link state c'è una vista della topologia temporaneamente inconsistente e potrebbero esserci possibili loop.
Infine, i pacchetti sono acknowledged.
Il #seq è di 32 bit e l'età (age) di 16 bit decrementa ogni secondo.

caratteristica	**distance vector**	**link state**
overhead in numero di messaggi scambiati	O(n)	$O(n^2)$
o/h computazionale	O(n)	$O(n^2)$
o/h di memoria	O(k)	O(k•n)
problemi	count-to-infinity	- oscilla a seconda della metrica usata - ci possono essere loop temporali
path	sub-ottimo	ottimo
internet	RIP BGP	OSPF

dove n=#router; k=#vicini per ogni router.

Internet structure - protocolli compagni di IP

Internet ha una struttura gerarchica divisa in 3 tiers:

1. connessione tra national network.
2. connessione tra regional network.
3. access network (LAN=local area network, ISP=internet service provider, wireless LAN,etc.).

Autonomous System= collezione di reti sotto un unico amministratore che usano lo stesso algoritmo di routing e connesse ad altri AS attraverso un router multiprotocollo.
i protocolli per la connessione tra AS sono:

- **OSPF**= Open Shortest Path First. Usa LS in quanto il numero di host all'interno di un AS è limitato quindi si può garantire un path ottimo (dijkstra).
- **BGP**= Border Gateway Protocol. Usa DV perché a fronte di milioni di AS è quello che costa meno (è una alternativa a RIP).

Servizi di gestione delle reti:

- ARP
- RARP
- BOOTP
- ICMP

sono servizi che non fanno instradamento.

ICMP - Internet Control Message Protocol

è un protocollo che affianca IP e lo aiuta a svolgere i suoi compiti, standardizzato nella rfc 1256. Si possono dedurre i tipi di servizi che svolge dai tipi di messaggi che scambia. Ci sono 6 tipi di messaggi:

1. **error reporting** →
    - **destination unreachable**= un pacchetto non può essere consegnato perché la destinazione non è raggiungibile. Questo può avvenire per diversi motivi: non c'è il cammino, il pacchetto è arrivato al destinatario ma non esiste il protocollo necessario per fare demultiplexing, la dimensione del pacchetto è troppo grande ma il bit DF(=don't fragment) non permette la frammentazione, oppure il cammino è chiuso al traffico.
    - **time exceeded**= il router si trova un pacchetto non destinato a lui perché è esaurito il TTL.
    - **parameter error**= qualche parametro dell'header non è valido (es.: checksum sbagliato).
2. **reachability testing → echo request/reply**= ping è un comando che un'applicazione utente (tramite prompt di comandi) ICMP che fa test dell'attività e raggiungibilità router.
3. **congestion control → source quench**= un router non riesce a stare dietro ai pacchetti ricevuti. Per ridurre congestione, il router invia ai router vicini un pacchetto

di source quench in cui avvisa che ha i buffer pieni. Fa backpressure. Quando si è decongestionato rimanda un source quench per notificarlo.

4. **route-change notification → redirect**= viene generato da un router A quando il suo vicino B gli invia un pacchetto, ma A capisce che B non dovrebbe essere il next hop. Allora manda un redirect per informare della situazione (controllare le tabelle di instradamento). Questo può avvenire nel LS.
5. **performance measuring → timestamp request/reply**= È una variante/OPZIONE di ping. Chiede ad ogni apparato attraversato di aggiungere il proprio timestamp. La destinazione prende tutti questi valori e li rispedisce alla sorgente.
6. **subnet addressing → address mask request/reply**= un router in una subnet annuncia ai vicini che è capace di portare pacchetti nella rete cha ha un determinato indirizzo dando l'indirizzo base e il "/k" quindi il numero di bit 1 nella parte sinistra della netmask che permette di estrarre dall'indirizzo che ho fornito l'indirizzo base della rete in cui sono. I router vicini possono mandare richieste e risposta di scambiarsi le netmask. In sostanza, serve per fare comunicare tra router le varie netmask.

I messaggi di ICMP non viaggiano da soli. Viaggiano come payload di un datagramma IP. A loro volta ICMP hanno un loro header costituito da:

- il tipo di messaggio (3 bit),
- il codice (ulteriori informazioni sul tipo, sottotipo) (8 bit),
- checksum sono 16 bit di ridondanza che validano il messaggio ICMP,
- il payload vero e proprio (informazioni su quello che ICMP sta cercando di segnalare).

DHCP - Dynamic Host Configuration Protocol

Serve per l'assegnamento di indirizzo IP in modo **dinamico**, l'end-system si configura da solo. E' un protocollo client/server e:

- possono esserci dei server DHCP interni alla LAN in cui si trova lo host,
- possono esserci anche più server DHCP all'interno della stessa LAN,
- può non esserci server all'interno della LAN. Allora c'è nella LAN un router che fa proxy: inoltra la richiesta a un server che sta in un'altra rete.

Come funziona? È un **four way handshake**:

- Prima il client manda un messaggio discover e con TTL=1. L'indirizzo suo è di tutti 0 ed etichetta il messaggio con un Transation ID unico;
- arriva una DHCP offer al client, composto da <transaction ID, proposed ID address, netmask, durata>. La durata serve perchè se non si accetta l'offer allora si libera l'IP oppure per limitare la durata del IP assegnato.
- conferma al server che il client ha preso l'indirizzo assegnato. Manda quindi una DHCP request in cui si identifica già col nuovo ip offertogli.
- Il server manda un DHCP ack.

Routing di Internet

Il routing in Internet consiste nella determinazione del path migliore tra sorgente e destinazione. Per trovare il cammino migliore bisogna trovare delle informazioni riguardo alla topologia della rete, con LS ci si procura informazioni su tutta la tipologia della rete per ogni router mentre con DV si ha una visione parziale.
Autonomous System= collezione di reti, generalmente sotto il controllo di un unico amministratore (inteso come entità), che usano lo stesso algoritmo di routing e connesse ad altri AS attraverso multiprotocol router.
Differenti hw/sw devono essere fatti interoperare.
Si fa una distinzione tra:

- **interior** gateway protocol= IGP. protocolli utilizzati all'interno degli AS → OSPF
- **exterior** gateway protocol= EGP. protocolli utilizzati tra AS → BGP

intra-dominio= è più facile raccogliere informazioni in quanto vi è un numero di apparati ridotto, quindi più facile definire topologia della rete e trovare algoritmi con cammini migliori. Una volta il protocollo interno era RIP (ha il problema del count-to-infinity perchè usa DV) ora è OSPF che usa LS.
inter-dominio= come protocollo di routing si usa BGP che usa DV (non ha il problema del count-to-infinity).

OSPF - Open Shortest Path First

E' un protocollo di Instradamento dinamico (rfc 2328) e fa un approccio Link-State (può mantenere più grafi della topologia con pesi diversi sui cammini, a seconda della metrica richiesta).

Può fare anche **instradamento multipath**: si suppone di richiedere un cammino che ha alta banda (es.: si vuole trasmettere materiale multimediale ad alta qualità) ma nessun cammino soddisfa quel requisito di banda. Con OSPF si può dividere il traffico su più cammini dato che si ha il grafo pesato con i bitrate dei link. Si calcola con dijkstra il cammino migliore. Allora si rimuove dal grafo tutti i link che fanno parte del cammino migliore. Ora lo si ricalcola e si ottiene un secondo cammino indipendente dal primo (non compete per banda con il primo cammino). Sommando le due capacità ci si avvicina alla banda richiesta. Altrimenti si ricalcola un terzo cammino e così via, finché non si raggiunge la capacità di banda richiesta. I cammini molteplici vengono usati tutti insieme alternando i pacchetti.

E' preferito per il routing intra-AS.

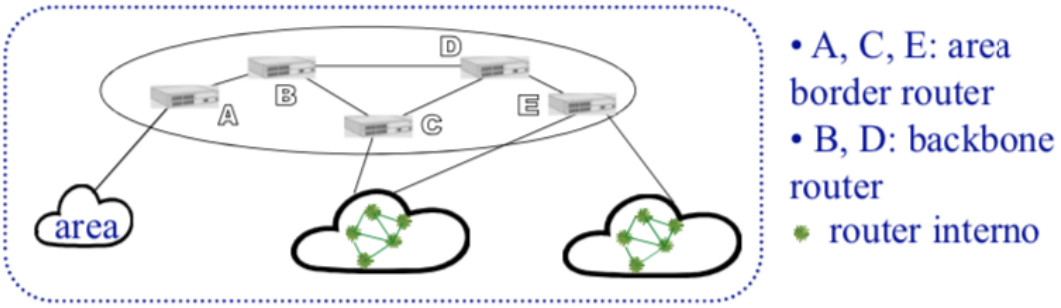
Ci sono 3 tipi di connessione che i router possono vedere:

- linee punto-punto tra 2 router (full duplex)
- LAN multiaccess (connesse a più router) + broadcast → stub link = LAN, cioè reti broadcast, connesse a più router. L'amministratore può scegliere se la LAN è di

transito quindi essere utilizzata nel calcolo dei cammini per decidere di far passare traffico che arriva da un router affacciato sulla LAN e va a un altro router oppure se la LAN è configurata come **stub link**= indica quei link che **non** sono utilizzabili per raggiungere una destinazione che **non** faccia parte di quella particolare LAN.
- WAN multiaccess + p2p = connessioni che vanno fuori dall'AS.

Se l'AS è piccolo, un amministratore può usare LS su tutto l'AS, ma se è piuttosto grande conviene semplificare l'AS usando una struttura gerarchica caratterizzata da:
- aree di router interni dove uso LS e appartengono ad una sola area
- area border router dove uso DV (router interni all'area che comunicano con quelli della dorsale o un altro router interno)
- area backbone router dove uso DV: non vengono inclusi alcuni end-system.
- AS boundary router: comunicano con i router in altri AS senza passare dalla backbone.

Si guadagna in termini di costo, ma si perde in termini di ottimalità di cammini (i cammini sono ottimi nelle aree ma non ottimi all'interno di tutti i cammini).



OSPF usa indirizzo MULTICAST di classe D **all_ routers** che indica tutti i router connessi ad una LAN. Per ogni LAN l'indirizzo all_routers è identico. Uno di questi router prende il ruolo di designated e si occupa di fare connessione della LAN con il resto di Internet (ricorda che nella LAN non ci sono router, ma ci sono sul confine).

<u>Ci sono diversi tipi di messaggi:</u>
- **hello** = per scoprire i vicini
- **link state update** = per ottenere informazioni sul vicinato
- **link state req** = serve ad un router per aggiornarsi
- **database description** = contiene l'ultimo #seq posseduto per aggiornare altri router
- **link state ack** = ack usati per fare flooding

BGP - Border Gateway Protocol

BGP usa DV ma risolvendo il problema del count-to-infinity, ricordando il path completo di ogni router (non solo del router adiacente). Serve per far comunicare più AS. E' un protocollo di livello 3 che usa il servizio di TCP (livello 4). Ogni router sa quali router deve

usare per arrivare alle varie destinazioni; nel caso cadesse un router al prossimo scambio reciproco di tabelle saprebbe da subito quanto è successo e può porvi rimedio.

BGP – tipi messaggi

- **OPEN**: per aprire comunicazione con BR adiacente
- **UPDATE**: per informazioni instradamento
  - info aggiornate su singolo path
  - cancellazione di molteplici path
- **KEEPALIVE**: per mantenere comunicazione con BR adiacente
  - come ACK a mesg OPEN
  - periodicamente per notificare liveness
- **NOTIFICATION**: per informare di condizione di errore

in sintesi:

- host sa come instradare a altri host nella stessa subnet, a router attaccati alla subnet e a access gateway
- subnet router sa come instradare ad altri subnet router nella stessa rete e a access gateway
- access gateway sa come instradare nella propria rete
- area border router sa come instradare nella propria area e verso la backbone
- backbone router sa come instradare attraverso la backbone agli area border router
- AS boundary router sa come instradare ad altri BR in Internet