Livello 2: DATA LINK

- il livello 2 lavora su un approccio punto-punto, una coppia di nodi per volta
- utilizzo del **bit stuffing** = *tecnica che consiste nell'aggiungere dei bit a zero ad un flusso di dati numerici*. In sintesi ho un flag che mi definisce la sequenza di inizio dei dati e la loro fine, ovviamente sto flag è una sequenza di numeri e sta sequenza di numeri potrebbe tranquillamente finire in mezzo ai dati anche, quindi è necessario trovare un modo per differenziare flag da una sequenza di dati comune. Bit stuffing aiuta in questo perché se ho flag 1111 e una sequenza dati pari a 01010**1111**01 sta sequenza diventa 01010**11110**1

Idle RQ

- regola la comunicazione tra due nodi adiacenti garantendo affidabilità
- ha solo valenza didattica
- considerando due nodi, A e B idle RQ opera nel seguente modo:
    - il nodo A invia per prima cosa il suo frame a B
    - B riceve il frame e segnala il ricevimento con un ACK
    - il nodo sorgente ha un buffer di dimensione 1 dove tiene copia dell'ultimo frame inviato nel caso dovesse rinviare il frame ce l'ha già pronto.
    - Quando riceve ACK svuota il buffer
    - definisce un timeout pari a **t>=RTT** cosi se non riceve un ACK ritrasmette il frame nel buffer
    - ogni volta che si riceve ACK poi il timer deve essere riavviato
- idle RQ ha un **numero di sequenza** pari a **1**
- nel caso vi fosse un errore nel timer di ritrasmissione allora ci potrebbero essere abiguita' sull'ACK dovuto alla non sincronizzazione. Conviene quindi mettere un **campo di sequenza anche per l'ACK**

    → **piggybacking** = la tecnica consiste nel mandare frame+ack contemporaneamente. Per poter adoperare questa tecnica il frame da trasmettere al mittente e l'ack dovranno essere necessariamente pronti in un tempo minore del timer di attesa della ritrasmissione del mittente. Questa possibilità ovviamente richiede che ogni unità frame contenga due numeri di sequenza distinti: uno per il frame e uno per l'ACK

Utilizzo del canale

- banalmente U è l'utilizzo totale della rete, piu il valore è altro meglio viene utilizzato il canale

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

- si ricorda inoltre che:
    - **Tp = Distanza / Vel.mezzo**
    - **Tx = Frame / Bandwidth**
    - **RTT = Tx + 2Tp**

Protocolli a finestra

- **Finestra** = la finestra di trasmissione è il numero di frame che il protocollo riesce a spedire prima di ricevere un ACK
- la dimensione della finestra rappresenta il numero di frame che riesco a mandare sfruttando il tempo di propagazione all'interno di RTT
- si può utilizzare la tecnica del NACK che permette di dire al mittente che:
    - il frame è arrivato ma non è corretto
    - il frame viene perso ma tutti gli altri sono arrivati
- IDLE rq lavora con una finestra pari ad 1, il trasmettitore però può lavorare secondo altri 2 algoritmi diversi:

1) **GO-BACK-N**
    - data una sequenza di frame N,N+1,N+2 se in tale sequenza si perde o arriva un frame corrotto a destinazione, occorre ritrasmettere l'intera sequenza a partire dal frame corrotto.
    - Per segnalare la perdita / corruzione di un frame il ricevitore trasmette un segnale NACK, cio' permette di rilevare prima il fallimento
    - appena riceve un NACK, il mittente dovra' **rimandare tutti i pacchetti ripartendo dal primo perso**
    - garantisce la corretta sequenza dei pacchetti ma richiede moltissimo tempo di ritrasmissione
    - indicando con K la finestra lato trasmissione si ha che **$K = 2^n - 1$**
    - indicando con MAX_SEQ il numero dei numeri di sequenza si ha che **$MAX\_SEQ = 2^n$**

2) **SELECTIVE-REPEAT**
    - sia trasmettitore che ricevitore hanno un buffer grande K
    - il trasmettitore invia solo il frame che non è arrivato.
    - I frame che sono stati ricevuti successivamente a quello perso, vengono conservati nel buffer
    - funziona come go-back-n ma non si ritrasmettono tutti i frame a partire dall'ultimo corretto ma solo il frame corrotto, identificato dal segnale NACK.
    - Quando entrano in **Retransmission Mode** mittente e destinatario smettono di mandare rispettivamente frame ACK

- **Retransmission Mode** = un nodo trasmettitore entra in retransmission mode quando riceve un NACK. Un nodo ricevitore entra in retransmission mode quando riceve un NACK ovvero quando arriva un frame fuori sequenza. Per ripartire ci sono due possibilità:
    - **ACK selettivo** = quando viene ricevuto un pacchetto corrotto, il nodo ricevente entra in retransmission mode. Viene scartato il pacchetto sbagliato ed inviato un NAK. I pacchetti successivi sono bufferizzati perché corretti. Quando viene ricevuto il pacchetto corretto che era stato scartato, per ogni altro pacchetto ricevuto in questo lasso di tempo, viene inviata una raffica di ack che li convalida tutti
      CONTRO: se il noto trasmettitore non riceve un segnale di ack con **selective repeat è un problema**
      PRO: non rischiano di far scattare il timer del trasmettitore

    - **ACK cumulativo** = aspetto a mandare tutti gli ACK successivi al frame perso, quando viene ricevuto il frame mancante ed è corretto mando un ACK dell'ultimo frame buono presente nel buffer, validando in questo modo anche tutti i frame precedenti. Con ACK cumulativo risparmio frame di controllo
      CONTRO: se il noto trasmettitore non riceve un segnale di ack con **selective repeat NON è un problema** in quanto vengono convalidati sempre tutti i pacchetti precedenti
      PRO: rischiano di far scattare il timer del trasmettitore

ALOHA con Carrier Sense

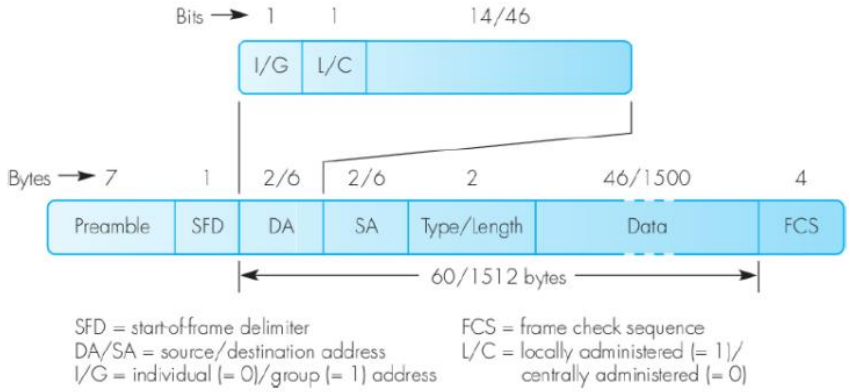
- se una stazione vuole trasmettere deve prima ascoltare il canale verificando se è libero.
- per farlo si utilizzano 3 diverse tecniche:
    - **CSMA 1-persistente** = continuo ad ascoltare il canale e appena trovo libero trasmetto
    - **CSMA 0-persistente** = non trasmetto appena il canale si libera ma aspetto un ***backoff***
    - **CSMA p-persistente** = quando si libera il canale, trasmetto con una probabilità p, mentre con probabilità 1-p aspetto un tempo
- le caratteristiche di 802.3 sono ben definite in CSMA CD:
    - **Carrier Sense** = ogni stazione sulla rete locale ascolta continuamente il mezzo trasmissivo
    - **Multiple Access** = il mezzo trasmissivo è condiviso da tutte le stazioni
    - **Collision Detection** = Se ogni nodo che trasmette è in grado anche di sentire cosa c'è nel canale, allora posso confrontare ciò che sto trasmettendo con ciò che ricevo, Se le sequenze di bit sono identiche allora sto inviando il frame correttamente e senza interferenze. Altrimenti significa che qualche altra stazione sta trasmettendo in contemporanea. Se i bit sono diversi ogni trasmissione si interrompe e ogni stazione genera un ritardo casuale definito da un range, il **Binary Exponential Backoff:**
        - ***r = rand(0 , 2^i -1)***
        - i = numero di collissioni avvenute
        - r = numero di slot di tempo che la stazione deve aspettare prima di ritentare la trasmissione
        - un singolo slot slot di BEB equivale a **51,2 microsecondi** quindi per sapere il tempo che ci mette un nodo prima di ritrasmettere bisogna fare **r*51,2 microsecondi**
        - si sceglie 51,2 microsecondi come slot perché tra uno slot di attesa e l'altro la stazione è in grado di inviare un frame

Rilevare Collisioni e Standard IEEE 802.3

- per poter verificare che i bit arrivino integri bisogna avere **tx>=2tp**
- se il tempo di trasmissione non fosse correttamente dimensionato, si potrebbero verificare casi in cui potrei aver terminato l'invio dell'intero frame prima che i bit per la verifica della collisione siano tornati indietro
- inoltre quando una stazione rileva una collisione invia una **Jam Sequence** per essere certi che tutte le stazioni coinvolte nella collisione si accorgano dell'imprevisto
- Secondo lo **STANDARD IEEE 802.3**:
    - Lunghezza massima: **2.5km**
    - utilizzo di almeno **4 ripetitori**
    - **Velocità trasmissione** 10Mb/s
    - **tx>=51.2 microsecondi**
    - **frame minimo di 64 byte**
      → in 51.2 microsecondi riesco a mandare minimo 64 byte per una lunghezza di 2.5km

Formato Frame LAN

- la **dimensione minima della frame ethernet è 512 bit, 64 byte**

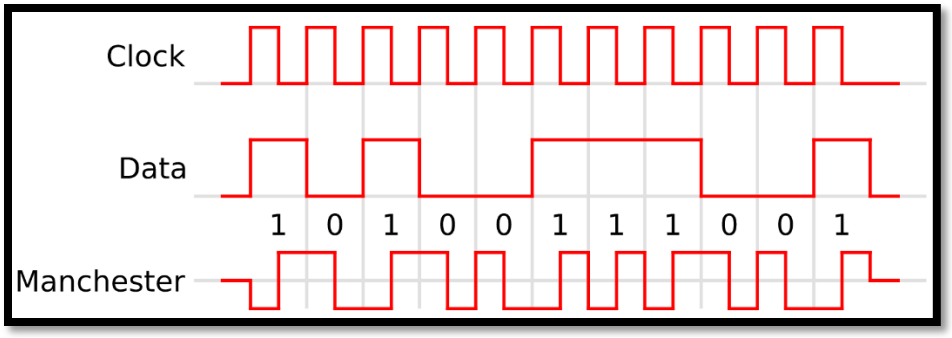


- **Preamble** = serve a svegliare gli adattatori del ricevente e a sincronizzarsi con il mittente
- **Start Frame Delimiter =** utilizzato il codice manchester
- **Type/length =** indiva la lunghezza del payload
- **payload =** da **46 a 1500 byte**. Nel caso non si raggiungano 46 byte si utilizza padding
- **CRC =** codice ridondanza ciclica

- i **MAC** address hanno tutti 6 byte, 48 bit

Codifica Manchester

- *"La codifica Manchester fornisce un modo semplice per codificare sequenze binarie arbitrarie senza mai aver lunghi periodi di tempo privi di transizioni di clock, il che permette di prevenire la perdita della sincronizzazione del clock"*
- questo aiuta anche ad evitare di confondere una sequenza di tutti zeri da una sequenza di idle
- grazie a questa codifica:
  - Ogni 1 è rappresentato da una transizione da 0 ad 1
  - Ogni 0 è rappresentato da una transizione da 1 a 0
  - IDLE è rappresentato dall'assenza di transizioni



Dispositivi di livello 2 (cenni)

HUB

- centrostella passivo, prende segnale e riinvia.
- tutti i pc collegati ad un hub si trovano nello stesso **dominio di collisione**
- un hub **propaga le collisioni**

BRIDGE

- dividono i domini di collisione
- Controlla i pacchetti corrotti che non inoltra
- il bridge utilizza una **tabella di forwarding** che mette in relazione tutti i nodi della rete con le porte del bridge a cui è connesso
- agisce nel seguente modo:
    - riceve frame
    - lo salva nel buffer
    - controlla la tabella
    - forwarding

SWITCH

- la differenza tra switch e hub è che i link che collegano i bridge agli switch non sono necessariamente link con CSMA/CD, perché lo switch gestisce le collissioni
- opera in full duplex, ha collegamenti punto punto, molto veloce e di livello 2 (a volte 3)
- esegue tutte le operazioni via hardware
- la prima tecnica di switching si chiamava **Store and forward** (come il bridge)

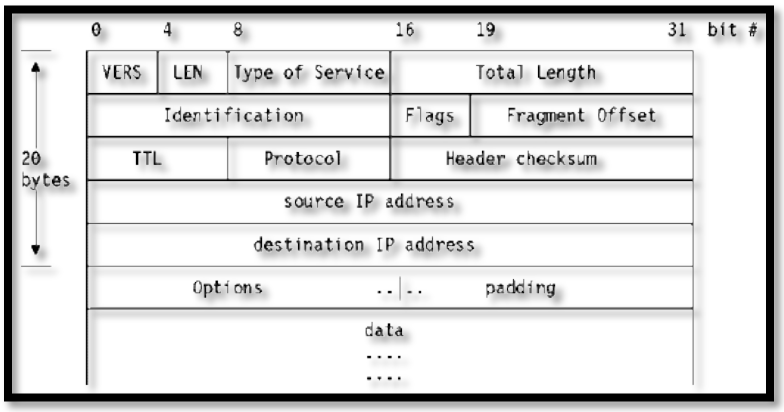
Tipi di Ethernet e loro valori standard:

- **Ethernet**:
    - **10** Mbit per secondo
    - **2,5 km**
    - 4 repeater
    - 512 bit min frame

- **Gigabit Ethernet**:
    - **1000** Mbit per secondo
    - **400m**
    - 1 repeater
    - **512 Byte** min frame

Livello 3: Network

- è il livello che ha il compito di assegnare un indirizzo agli host della rete.
- Si occupa principalmente di:
    - **Addressing:** ovvero permettere di identificare univocamente un nodo di una sottorete
    - **Routing:** ovvero definire come instradare un pacchetto tra sottoreti diverse. Per quanto riguarda questo compito si utilizzano due protocolli:
        - **OSPF (oggi MPLS) :** per instradamento inter e intra-rete
        - **RIP:** per instradamento intra-rete
- il livello 3 come il livello 2 può a sua volta essere diviso in due macro livelli:
    - **Intra-rete**: che si occupa dell'instradamento all'interno della propria rete, utile nel caso delle reti magliate
    - **Inter-rete**: che si occupa di fornire l'indirizzo IP univoco e specificare i protocolli per instradare tra sottoreti. Vengono specificate le regole e non i protocolli.
- nel layer 3 non si parla piu di frame ma di **pacchetti**.
- IP fornisce un servizio datagram
- l'header di un datagram IP è organizzato nel seguente modo:



- **Version** = Specifica la versione di IP → istruisce un gateway su come interpretare il pacchetto
- **HL =** Specifica la lunghezza dell'header, che può essere variabile. Questo campo specifica la lunghezza dell'header in multipli di 32 bit. Min 5, Max 15
- **ToS** = Fornisce una priorita' al pacchetto IP cosicche' I router possano instradarlo nel modo migliore
- **Total Length =** indica la lunghezza totale del pacchetto IP (header+payload)
- **Identification =** identificatore del pacchetto originale, uguale in ogni pacchetto del messaggio originale
- **Flags:**
    - **DF** = Don't fragment, quindi il pacchetto non e' frammentato
    - **MF** = More Fragment, quindi il pacchetto e' frammentato. Viene messo ad 1 se devono arrivare altri frammenti, oppure viene messo a zero nel caso il frammento in considerazione sia l'ultimo
- **Fragment Offset** = indica quanti bit sono stati gia' inviati dai frammenti precedenti. Vengono contati cluster di 8 byte massimizzando quindi l'uso di 13 bit (indicizza 2^{13} = 8192 gruppi di 8 byte ovvero 15536 Bytes = maximum segment size di un segmento di livello 4)
- **TTL =** Tempo massimo (numero di hop) di vita di un pacchetto nella rete. Se scaduto tale tempo e il pacchetto non e' stato consegnato esso deve essere terminato.
- **protocol** = indica il protocollo di livello 4 usato dal mittente
- **checksum** = usato per efficienza operativa, se la checksum e' sbagliata evito subito di instradare il pacchetto. La chk controllera' solo l'header per questo il livello 4 calcolera' CRC anche per contenuto IP
- **options** = campo variabile usato per specifiche opzioni.

FRAMMENTAZIONE

- *Supponiamo di dover trasferire un pacchetto di 7000 byte (7 KB) da una LAN token ring attraverso Internet fino ad un host di una LAN Ethernet. Assumiamo che la quantità massima di trasmissione (MTU) su token ring sia di 4000 byte, mentre quella Ethernet 1500 byte. In una classica Ethernet i frame contengono da un min di 64 B fino a 1500 B di dati; il livello IP aggiunge ai dati veri e propri il proprio header di 20B. Abbiamo quindi 4000 – 20=3980 byte disponibile per il payload e dobbiamo creare frammenti con payload multipli di 8 byte. Vengono inviati quindi due frammenti, che contengono, nell'header, le informazioni per la ricostruzione (indicati in figura sotto Token ring LAN). Quando questi due pacchetti giungono sulla rete di destinazione, devono essere ulteriormente riframmentati per poter viaggiare in Ethernet, ed infatti entrambi i pacchetti vengono divisi in altri 3 pacchetti, che contengono nell'header le informazioni per il destinatario (indicati in figura sotto Ethernet LAN).*

CLASSI ED INSTRADAMENTO

- Gli indirizzi IP girano su 4 classi:
  - **CLASSE A**: da 1.0.0.0 a 127.255.255.255
  - **CLASSE B**: da 128.0.0.0 a 191.255.255.255
  - **CLASSE C**: da 192.0.0.0 a 223.255.255.255
  - **CLASSE D**: usata per multicast
  - **CLASSE E**: usata per esperimenti

- Per quanto riguarda l'instradamento sono fondamentali le Tabelle di routing. Esse tengono traccia di tutte le possibili strade per fare arrivare a destinazione il pacchetto.
- il router tiene in memoria solo gli indirizzi della sua sottorete e ne lcaso arrivi un pacchetto per un'altra sottorete esso lo affida ad un router di livello superiore

NETWORK ADDRESS TRANSLATION

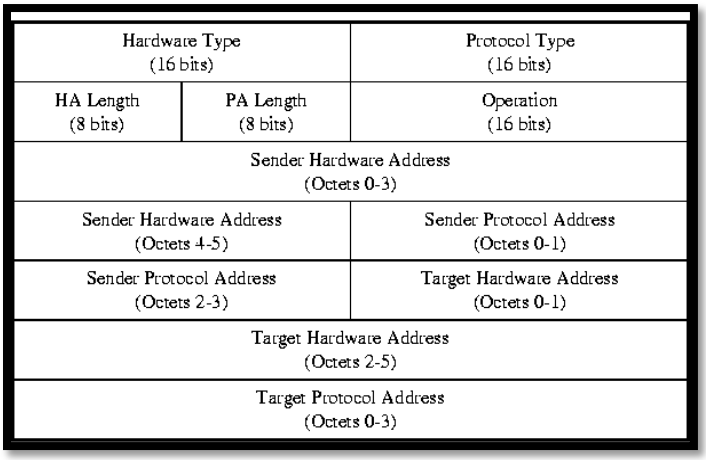
- strumento di livello 3 con funzioni di livello 4
- spesso sono i router stessi che presentano la funzionalità di NAT. Essi riescono a svolgere tale funzione grazie alle tabelle NAT o PAT, composte dai seguenti campi:
  - **Source Port** = porta dell'host
  - **Source IP** = IP dell'host
  - **External IP** = IP pubblico definito dal NAT
  - **NAT-Port** = porta assegnata dal NAT che sostituisce la Source Port se quest'ultima è già occupata
  - **Destination IP** = IP di destinazione della richiesta
  - **Destination Port** = Porta del destinatario
- permette in un certo senso di risparmiare indirizzi IP
- è un processo che consente a un indirizzo IP univoco di rappresentare un intero gruppo di computer

CLASSLESS INTER DOMAIN ROUTING

- alternative all'indirizzamento tramite classi
- viene assegnato un blocco di indirizzi contigui ad ogni richiesta considerando quindi anche il caso di possibile espansione.
- Quando bisogna capire di che sottorete è un determinato pacchetto bisogna fare un and logico tra la mscehra e il destination address per ogni associazione che ha richiesto indirizzi CIDR e quando ricavo una base di un'associazione conosciuta, inoltro il pacchetto

ADDRESS RESOLUTION PROTOCOL

- per poter comporre pacchetti di livello 2 e poter quindi comunicare con un host della stessa sottorete sono necessari: IP e MAC
- **dato un indirizzo IP, ARP fornisce il corrispondente MAC**
- ARP funziona tramite:
    - ARP Request
    - ARP Reply
- ciascun host conterrà una ARP table per massimizzare le prestazioni
- anche il gateway prende nota di tutte le coppie che passano sulla rete anche se non sono messaggi ARP per lui.



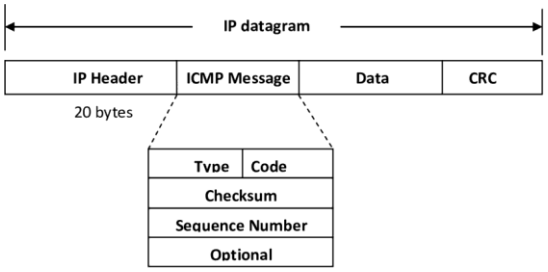
- **Hardware Type:** specifica interfaccia hw su cui utente cerca una risposta
- **Protocol Type:** indica il tipo di indirizzo ad alto livello che il mittente ha fornito
- **Hardware len e protocol len** : consentono di usare ARP su reti arbitrarie perche specificano la lunghezza dell'indirizzo MAC e dell'indirizzo del protocollo ad alto livello (IP)
- **operations:** specificano il tipo di operazioni da fare
- il pacchetto di livello 3 viaggia nel campo **dati** del frame ethernet
- il **RARP** serve per conoscere l'indirizzo IP di un host del quale si sa l'indirizzo MAC.

DYNAMIC HOST RESOLUTION PROTOCOL

- protocollo utilizzato per richiedere un indirizzo IP ad un server DHCP
- si basa su 4 messaggi nel seguente ordine
    - **client -- DHCP DISCOVER -->** : il client invia in broadcast un messaggio di discover avente come indirizzo sorgente 0.0.0.0. Questo messaggio contiene un TransactionID che ha il compito di identificare univocamente la richiesta cosicche il server possa associare le risposte inviate al client.
    - **server -- DHCP OFFER -->** : risposta del server che contiene transactionID della richiesta, indirizzo IP proposto dal server, subnet mask e il tempo di lease. in questa fase avviene anche un **ICMP check** (*ogni server sceglie un indirizzo IP da offrire ma deve verificare che non sia già in uso, questa verifica viene fatta dal server inviando un messaggio a quell'indirizzo, se si riceve una risposta, l'indirizzo dovrà essere cambiato*)
    - **client -- DHCP REQUEST -->** : il client sceglie uno tra gli indirizzi proposti dai server e manda un messaggio a tutti i server.
    - **server -- DHCP ACK -->** : il server scelto manda un ACK di conferma.
- se dopo un certo quanto di tempo l'host non riceve un OFFER allora rimanda DISCOVER perche potrebbe essersi perso.
- Se dopo il request il server ha già dato l'indirizzo IP a un altro host, e quindi non può convalidarlo, al posto dell'ACK, viene mandato un segnale di NACK. L'host, a quel punto, o sceglie altre offerte da altri server, oppure rimanda discover a tutti
- Se all'host che ha fatto request non arriva ne ACK ne NACK, dopo un certo tempo rimanda request per un po di volte. Se non ha ancora ricevuto risposta, o cambia offerta oppure ricomincia dal discover

INTERNET CONTROL MESSAGE PROTOCOL

- funzionalita' puramente di controllo
- utilizzato da hosts routers e gateways specialmente per la gestione della rete
- altre funzioni importanti:
    - rilevare errori
    - rilevare congestione dei nodi
    - verificare raggiungibilita' dei nodi
    - notifica di cambio percorso
    - misurare prestazioni dei link



- se viene rilevata la congestione di un nodo e' possibile
    - avvisare i nodi che precedono di bufferizzare in uscita per rallentare il lavoro
    - avvisare il nodo sorgente di non mandare troppi messaggi
- ICMP viene contenuto nell'IP datagram e possiede l'header IP e i campi principali sono:
    - **type**: tipo di controllo da effettuare
    - **Code**: informazioni opzionali aggiuntive

INSTRADAMENTO

- le funzionalita' principali del router sono:
    - controllo tramite ICMP
    - forwarding
- NB:
    - **ROUTING**: *funzionalità che riempie la tabella di instradamento e la tiene aggiornata*
    - **FORWARDING**: *processo che fa passare un pacchetto da una coda di ingresso ad una coda di uscita*
- entrambi sono **asincroni**
- vi sono due algoritmi per fare routing:
    - **Distance Vector**
    - **Link State Shortest Path First**
- e per realizzare questi algoritmi sono stati implementati:
    - **RIP**
    - **OSPF**

DISTANCE VECTOR

- è un algoritmo distribuito che consente ad ogni router di costruire una routing table che contiene il costo di ogni percorso per raggiungere tutti gli altri
- inizialmente ogni router si costruisce la propria tabella valutando il costo dei link che lo connettono ai propri vicini.
- funziona nel seguente modo
    - PASSO 1
        - ogni nodo all'inizio conosce solo se stesso e i nodi a lui adiacenti
        - ogni nodo salva le informazioni sui costi dei link nella tabella delle adiacenze
    - PASSO 2
        - le informazioni riguardo la distanza da un nodo all'altro viaggiano da nodo a nodo attraverso pacchetti vector
        - chi riceve questi pacchetti scopre cosi nuovi percorsi ai link e quindi aggiorna la tabella di routing se conviene
- Criticità:
    - altissimo overhead su reti ampie dovuto allo scambio di vettori
    - bouncing effect → puo' degenerare nel count to infinity

- la tecnica **split horizon** si candida per risolvere bouncing effect / count to infinity. Ogni nodo propaga il distance vector ma invia sempre infinito come costo di raggiungimento di stazioni indirette che raggiungerebbe tramite la stazione a cui sta inviando il vettore ovvero il vicino. Il problema di split horizon è che come ogni cosa, il messaggio potrebbe perdersi, cosi facendo quindi si potrebbe arrivare alla non convergenza delle tabelle. Una possibile soluzione è quella di utilizzare la funzionalità trigger update ma per ogni entry dopo 6 mancati update si aggiorna ad infinito.
- **Trigger Update** = appena rilevato un imprevisto, la stazione che ha rilevato il problema spedisce subito il suo vettore senza aspettare lo scadere del timer (inizio dello scambio dei vettori).
- **RIP** implementa la tecnica Distance Vector
    - Possiede le seguenti proprietà
        - per ogni entry esiste un timer. Se per 6 tempi di update non ci sono risposte allora viene messa la entry ad infinity
        - **Trigger Update**: esiste un timer di propagazione del DV per ogni nodo, Se un nodo su un suo link rileva un fallimento, spedisce subito il DV ai vicini anche se i timer non sono ancora scaduti
        - il costo di un link si indica con il numero di hop
        - **Update storm**: puo capitare che i timer scadano tutti insieme, quindi viene generato tantissimo traffico in rete. Ogni nodo genera quindi il proprio update con un ritardo randomico (0-5 secondi)

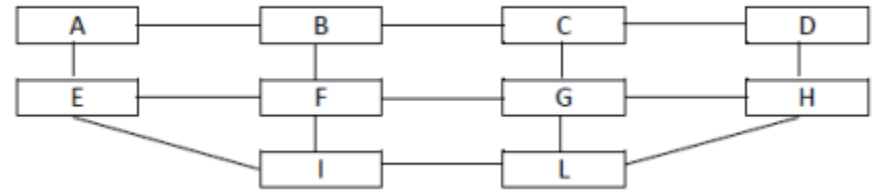
LINK STATE ROUTING

- invece di scambiare i vettori delle distanze si scambiano i vettori di stato ovvero tutti i nodi si trasmettono i costi dei link a loro connessi
- i vettori di stato vengono inviati in flooding a tutti gli altri nodi
- ogni nodo quindi ha tutte le informazioni per avere una visione completa della rete
- Un vettore di stato conterrà il costo di raggiungimento di un nodo ai suoi vicini e sarà sempre cosi. Questo dato viene mandato a tutti gli altri nodi consentendo a tutti i nodi di mappare tutta la sottorete
- il pacchetto link state ha le seguenti caratteristiche:
    - **numero di sequenza**: serve ai nodi per droppare i pacchetti già ricevuti
    - **fattore di aging**: usato per eliminare pacchetti che continuano a girare in rete, serve ad evitare che un pacchetto rimanga troppo in rete
- Ogni router al boot eseguirà
    - determinazione address nodi vicini
    - determinazione costi dei nodi vicini
    - costruzione di una tabella locale
    - comunicazione della tabella in flooding a tutti gli altri nodi
    - calcolo dello shortest Path per tutti i nodi (algoritmo di Dijkstra)
    - Scaduto un timer torna al punto 2
- link state è piu piccolo di un DV perché contiene solo le informazioni dei suoi vicini
- richiede $O(n^2)$ messaggi in rete per fare routing. Come soluzione si decide un **designated router**: tutti gli stati vengono inviati a questo router ed è proprio quest'ultimo a calcolarae i cammini minimi e a mandarli a tutti i suoi nodi ogni periodo di tempo (in genere solo se cambiano)
- **Open Shortest Path First (OSPF)** è il protocollo di internet che gestisce la gran parte dei problemi di routing con tecnica Link State.
    - OSPF opera in:
        - **Autonomous Systems:** sistemi autonomi composti da diverse sottoreti collegate ad un'area zero
        - **Backbone areas:** aree a piu alto livello gerarchico, vi sono collegati speciali routers che fungono da gateway tra sottorete degli host e internet backbone
            - → ogni AS è composto da una backbone area (area 0) che connette le varie sottoreti e gli altri AS. I router che permettono questi collegamentei sono i bourder router (utilizzano BGP). L'area 0 è composta solo da router, dato che funziona con OSPF per evitare i problemi legati al flooding si inserisce un **designated router** che non è altro che un router che concentra su di se tutte le update fatte su ogni nodo. Le informazioni raccolte da ogni nodo vengono spedite al designated router. Solo il DR costruirà tabelle di routing e le invierà a tutti i nodi connessi periodicamente.
    - invia i seguenti tipi di messaggi:
        - **Hello**: usato da un router per scoprire nuove reti/router adiacenti
        - **Link State Update**: scopo di istruire un router sullo stato e il costo corrente di ogni link della topologia. Come per il RIP contiene un numero di sequenza.
        - **Link State ACK**: ogni router valida un link state update con questo messaggio
        - **Database Description**: usato per informare il ricevente del messaggio se è disponibile o meno un aggiornamento
        - **Link State Request**: usato per richiedere un link state update da ogni router adiacente

BORDER GATEWAY PROTOCOL

- utilizzato da ogni **Autonomous System Boundary Router** *(Un ASBR è un router connesso a più di un Autonomous system (Internet) (AS), che scambia informazioni di routing con router in altri AS.)*
  → permette la comunicazione tra router che compongono area zero
- messaggi inviati tramite TCP
- utilizza approccio distance vector modificato, chiamato quindi **Path Vector**
  → viene indicato esplicitamente tutto il percorso da fare per raggiungere un AS boundary router. Al posto di propagare le distanze si propagano i cammini.
- Path Vector:
  - partendo dai distance vector, il path vector viene costruito usando le tabelle dei propri vicini
  - sceglie il cammino minimo che porta alla destinazione

Esempio



Vediamo quale può essere il percorso migliore da F a D: il percorso migliore è FGCD
Ora supponiamo che si rompe il link FG
Il nodo F cerca quindi altri possibili link per raggiungere D, e lo fa attraverso i messaggi provenienti dai nodi vicini
Al nodo F arrivano i PV :

- Da B arriva BCD con costo 8 (link buono, tengo questo)
- Da E arriva EFGCD con costo 3 (ma F si accorge che è presente il link che si è rotto, dunque lo scarta)
- Da I arriva IFGCD con costo 10 (ma F si accorge che è presente il link che si è rotto, dunque lo scarta)

- utilizza quattro tipi di messaggi:
  - **OPEN**: aprire una relazione con un AS boundary router adiacente
  - **UPDATE**: usato per trasmettere informazioni sull'instradamento per un singolo percorso che per comunicare una lista di percorsi che devono essere rimossi
  - **KEEP ALIVE**: utilizzato per validare una Open e mantenerla attiva, confermandola periodicamente
  - **NOTIFICATION**: usato per informare che si e' verificata una condizione di errore

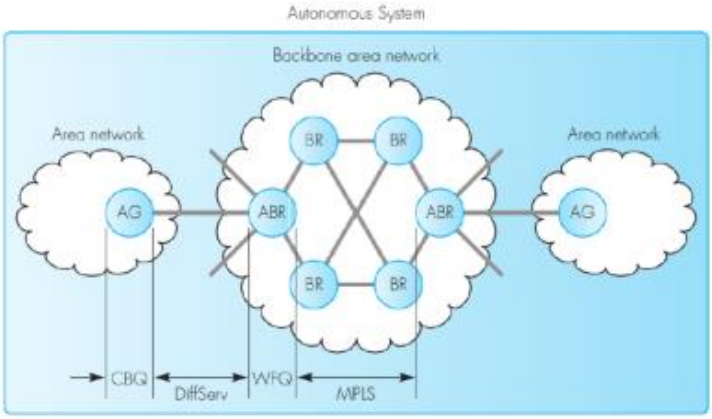
- **BGT** = **Border Gateway Tunnelling** = Quando un pacchetto IP viaggia attraverso sottoreti non e' detto che tutte usino il protocollo IP. I Border Gateways delle reti che non utilizzano IP provvedono quindi ad incapsulare il pacchetto IP in un altro pacchetto conforme al protocollo usato nella sottorete da attraversare.

NOMENCLATURA:

- **Subnet Router**: contengono informazioni sufficienti per poter instradare pacchetti agli altri subnet router e al default gateway
- **Gateway**: mantengono informazioni sufficienti per poter instradare pacchetti alla loro sottorete.
- **Area Border Router**: contengono informaizoni sufficienti per poter instradare pacchetti ad altri router di bordo area nello stesso o differente AS
- **Border AS routers**: possono instradare pacchetti ad altri AS boundary attraverso la internet backbone

MULTI PROTOCOL LABEL SWITCHING (MPLS)

- il fattore temp è fondamentale quando si parla di real time
- OSPF oggi è stato sostituito da MPLS
- ogni pacchetto inviato ha anche un campo ToS che viene gestito da un **packet classifier** all'interno del router
- Quest'ultimo ha il compito di scegliere le regole di scheduling che devono essere applicate al pacchetto:
    - controlla il ToS del pacchetto
    - assegna un'etichetta al pacchetto
    - aggiunge nuovo header
    - inserisce il pacchetto nella coda di uscita piu appropriata
- **DiffServ** = quando il traffico viene diviso in base al tipo e non al singolo flusso dati
- MPLS aggrega il traffico di un flusso ed effettua un bilanciamento tra tutti i flussi.
    → questo comportamento avviene nell'area di backbone a cui ogni AS e' collegato



AG = Access Gateway

WFQ = Weighted for queuing

CBQ = Class-Based Queing

BR = Backbone Router

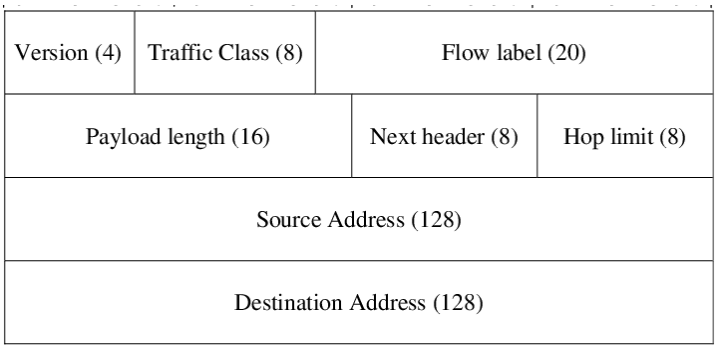
ABR = Area Border Router

- **AG** = effettua scheduling in base alla classe, dopo separa ogni flusso di dati necessario per permettere l'interpretazione corretta dei pacchetti di ogni flusso da parte degli area bourder router.
- **ABR** = provvedono ad aggiungere/rimuovere header MPLS dalla testa del pacchetto IP. L'header MPLS ha senso solo ad ogni hop e non in tutto il percorso, questo perche' quando vengono usati sempre i percorsi minimi alcuni ABR si sovraccaricano e formano degli **hotspot.** Per evitare cio' viene effettuato un **load balancing** attraverso **label switching**

- l'obiettivo di MPLS è quello di gestire al meglio il traffico. Per faro:
    - si determina un Label Switched Path attraverso l'area zero.
    - alla ricezione di un pacchetto IP l'indirizzo di destinazione viene utilizzato per determinare la porta di uscita
    - valore di DiffServ viene analizzato dal modulo LSP.
- con le informazioni sopra riportate viene formato **l'header MPLS per il singolo hop.**

IPv6

- nasce per sopperire alla mancanza di ipv4 e presenta diverse caratteristiche:
    - spazio di indirizzi aumentato da 32 bit a 128 divisi in 8 word da 2 byte
    - indirizzi gerarchici per ridurre le dimensioni delle routing table
    - introduzione di sistemi di sicurezza come autenticazione e criptazione
    - funzione di autoconfigurazione per consentire ad un host di ottenere un IP tramite la rete senza intervento umano
    - compatibilita' con protocolli IPv4
    - header semplificato per aumentare velocita' di elaborazione dei router (40 Byte):



        - **Version** = versione del datagramma IP
        - **Traffic Class** = (successore di ToS)
        - **Flow Label** = usata dal mittende per etichettare una sequenza di pacchetti come se fossero nello stesso flusso. (0=best effort)
        - **Payload Length** = dimensione del payload (numero di byte di tutto quello che viene dopo header)MIN
        - **Next Header =** indica che tipo di intestazione segue l'header di base di IPv6. (simile a campo protocol) Se non vi è estensione del header allora Next Header conterrà puntatore all'header del livello superiore (TCP), se vi è estensione allora Next Header punterà ad un Extension Header che conterrà a sua volta un Next Header etc etc finchè non si arriva ad un NextHeader che punta a TCP
        - **Hop Limit =** limite di salti consentito (equivale al TTL)
        - **Source Addr =** 16 B per indirizzo sorgente
        - **Dest Addr =** 16 B per indirizzo destinazione

- le opzioni vengono messe in NextHeader ovvero che l'header puo diventare piu grande per contenere informazioni aggiuntive. Tra le piu importanti vi è il header **ROUTING:**
        - se viene messo 43 nei bit meno significativi del next header allora si usa ROUTING.
        - Serve per implementare source routing. L'header contiene I seguenti campi importanti:
            - **HEADER EXTENSION LENGTH**= al massimo 24 indirizzi da cui si vuole passare
            - **STRICT/LOOST BIT MAP** = tanti bit quanti i possibili router fissati. Se c'è uno voglio un collegamento diretto verso il router (routing esatto) altrimenti se c'è zero non mi interessa come arrivo a quel router ma è sufficiente che ci passi. Questo campo mi dice quindi se il prossimo router è indicizzato dal precedente in modo diretto o meno

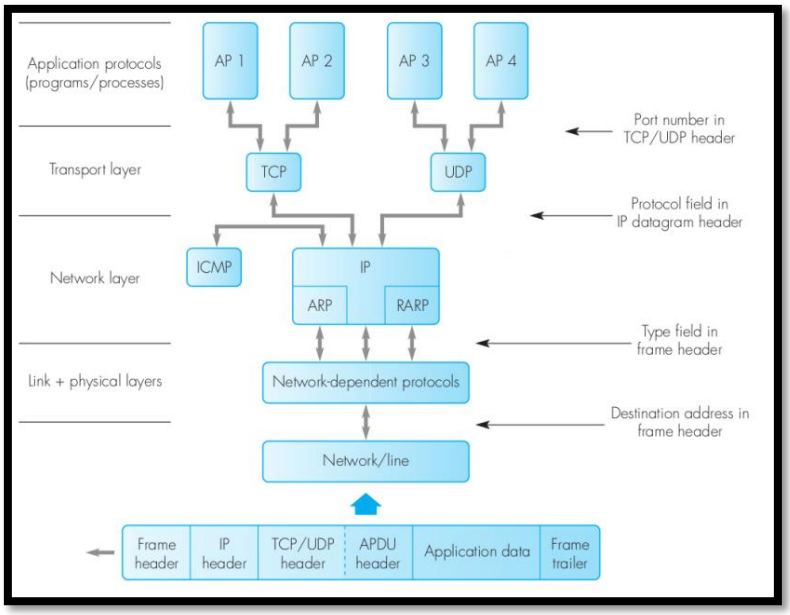
- ha il seguente formato: 8 blocchi divisi da ":" scritti in esadecimale. Inoltre raddoppiando il separatore si può scrivere una sequenza di tutti zero
    - es: FDEC : BA98 : 7654 : 3210 :: 0089 in notazione contratta
    - es: FDEC : BA98 : 7654 : 3210 : 0000 : 0000 : 0000 : 0089 in notazione completa

Livello 4: TRANSPORT

- non si parla piu di nodi ma di **host**
- si parla di comunicazione di processi all'interno dell'host stesso
- I servizi a questo livello possono essere
    - AFFIDABILI = TCP
    - NON AFFIDABILI = UDP

NAMING

- problema che consiste nel riuscire a determinare univocamente una entità consentendone cosi la comunicazione
- ogni layer ha un protocollo destinato alla soluzione di questo problema, a livello 4 è necessario:
    - associare un indirizzo IP ad un nome simbolico di un servizio → si utilizza il **DNS**
    - legare una singola applicazione e quindi un processo ad un indirizzo IP →si utilizza la **PORTA**
- la porta è quindi un **descrittore di sessione di livello 4** che lega in modo univoco una istanza di livello 4 con un processo di livello 7
- una combinazione **<IP,PORTA>** viene definita **SOCKET.**
- Ogni segmento TCP/UDP utilizzera' quindi
    - porta sorgente → **Ephemeral Port**
    - porta destinazione → **Well Known Port**
- si hanno 2^{16} porte
    - da 0 a 1023 servizi standard
    - da 1024 a 49151 programmi famosi
    - 49152 - * porte libere
-

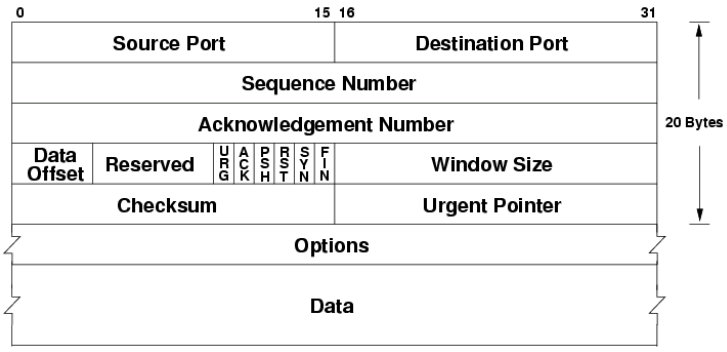


ARCHITETTURA CLIENT E SERVER

- architettura fondamentale dei servizi internet
- si basa sul concetto di
    - client = Componente sempre attiva
    - server = Componente sempre passiva
- i servizi offerti da un server sono in ascolto su alcune porte predefinite mentre i client effettuano connessioni da porte con numero dinamico

Transmission Control Protocol TCP

- fornisce un servizio che garantisce affidabilità alla comunicazione best-effort di IP
- una connessione TCP è bidirezionale poiché apre due connessioni
    - Client → server
    - Server → client
- è un protocollo a finestra e le unità dati si chiamano **segmenti**
- header TCP formato da minimo 20 Byte ed ha la seguente struttura



- in TCP i ruoli client e server devono essere ben distinti:
    - CLIENT:
        - crea socket
        - crea connessione dove indica IP destinazione e porta
    - SERVER
        - Crea socket
        - Bind: unisce indirizzo IP del processo alla porta interessata
        - Listen: dichiara il numero di connessioni massime che potrà ricevere
        - Accept: attende che qualcuno si connetta
        - Fork: crea un figlio con una nuova porta ed assegna a lui l'incarico di gestire la richiesta del client

- la connessione iniziale di TCP avviene tramite il 3WH (Three Way Handshake)

1) client **-- SYN -->** server
    - ogni entità TCP sceglie un numero di sequenza (Initial Sequence Number)
    - ogni ISN cambia per ogni connessione per garantire che un segmento di una connessione non interferisca con il segmento di un'altra connessione
    - client invia un segmento con Code Bits il bit SYN=1 e il numero di sequenza = ISN scelto

2) client **<-- SYN+ACK --** server
    - viene effettuata una fork che genera un processo figlio che gestisce la connessione con quell client
    - il figlio forkato deve trovare il suo ISN e spedirlo al client insieme all'ACK del segmento successivo

3) client **-- ACK -->** server
    - il client risponde convalidando il segment del server

IMPORTANTE: Gli ISN vengono validati attraverso i due ACK, inoltre il massimo numero di ISN è 2^{32}

Controllo di flusso

- per evitare che il mittente non invii dati se il destinatario non è in grado di ricevere si utilizza il campo **Window Size** nel segmento TCP il quale indica il numero di byte che si è in grado di ricevere (Wr) ed è ovviamente determinato dallo spazio nel **buffer di ricezione**
- Si adotta il seguente schema:
    - Lato **Mittente**: si mantiene una variabile **Ws** che contiene il valore della dimensione del buffer di lato ricezione (Num di Byte)
    - Lato **Ricezione**: si mantiene una variabile **Wr** definita precedentemente
    - finchè Ws>=0 e ci sono dati da inviare:
        - il mittente può spedire segmenti decrementando di volta in volta il valore di Ws sottraendo il numero di byte inviati.
        - Man mano che il buffer in lato ricezione si riempie Wr viene decrementato
        - Man mano che i dati vengono letti dal buffer ricevitore dal livello superiore il valore della varaibile Wr viene incrementato
        - ogni volta che il valore di Wr viene modificato allora viene inviata una window **UPDATE**
- nel caso un ACK venisse perso vi sono due strategie:
    - **PERSISTENT TIMER**
        - ogni volta che viene ricevuto un segmento con Win_size = 0 il sender viene bloccato e viene fatto partire un timer
        - Se un segmento contenente un aggiornamento della finestra non viene prima della scadenza del timer, TCP client tramite il segmento **window probe** richiede esplicitamente al server l'ACK perduto
        - Dopo un numero finito di tentativi se non ha risposta la connessione viene chiusa
        - Questa soluzione evita che si formino DEADLOCK

    - **KEEP ALIVE TIMER**
        - durante la connessione allo scadere di un timer il server periodicamente spedisce un keep alive probe che controlla che il client sia ancora attivo
        - dopo un numero predefinito di non risposte il server chiude la connessione evitando di tenere quindi aperte connessioni inesistenti

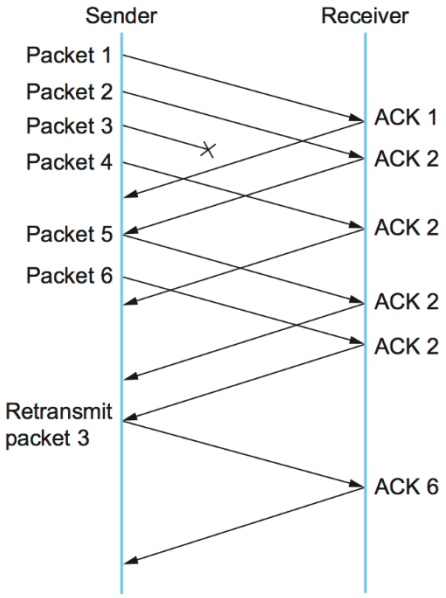
Ottimizzazioni TCP Data Transfer (Nagle, Clark)

- **Silly Window Syndrome** = *Un processo di scrittura molto lento da parte del mittente nel buffer di trasmissione (o di lettura da parte del ricevente) porta infatti all'invio di segmenti di dati molto piccoli, aumentando così il rapporto tra header e dati con un conseguente uso inefficiente del canale.*
- per gestire questo problema esistono le seguenti tecniche:
    - **Delayed Acknowledgements**
        - consiste nel fare attendere il server un tempo prefissato dopo la ricezione del dato
        - in questo modo gli eventuali dati che in questo tempo arrivano sul buffer di invio del mittente potranno essere raggruppati e inviati insieme all'ack del dato precedente correttamente ricevuto.
    - **Algoritmo di Nagle**
        - si invia un piccolo segmento alla volta
        - mentre l'entità TCP lato server aspetta la ricezione dell'ACK il buffer di invio si riempie cosi che quando sarà nuovamente possibile inviare piu dati utili potranno essere inviati in un solo segmento
    - **Algoritmo di Clark**
        - consiste nel compattare i window update lato server
        - si ritarda la spedizione del window update finchè il buffer del server ha raggiunto la MSS oppure la metà della memoria disponibile

Controllo degli errori

- TCP utilizza ACK cumulativi e non usa NACK, Ogni segmento è validato da un ACK e per ciascuno di essi esiste un timer che viene resettato ogni volta che riceve un ACK
- Tecnica **FAST RETRANSMIT:**
    - Presupposto: In internet i segmenti possono arrivare fuori sequenza
    - Quando al destinatario arrivano segmenti fuori sequenza esso si limita a memorizzarlo temporaneamente e ad inviare un ACK indicando il segmento che si aspetta. Normalmente il segmento mancante arriva entro un breve intervallo di tempo e a quel punto il ricevente risponderà con un ACK che validerà tutti i segimenti ricevuti fino ad esso. → ACK CUMULATIVO
    - Lato sender il problema è che riceve un ACK che indica la perdita di un segmento seguito dopo da un piccolo intervallo di tempo e un ACK che valida tutti i segmenti inviati.
    - Sarebbe bello quindi che la ritrasmissione del pacchetto non avvenga alla ricezione del primo ACK fuori sequenza

        → **si aspetta la ricezione di tre ACK fuori sequenza prima di reinviare il segment mancante**
    - **fast retransmit** richiede l'utilizzo di un **timer di ritrasmissione RTO associato ad ogni segmento inviato lato sender**
    - al termine di RTO se non si e' ancora ricevuto ACK il segmento mancante viene ritrasmesso.



RTO: Timer TCP

- a livello 4 non si possono conoscere esattamente i tempi di percorrenza della rete. è necessario quindi trovare un modo per stimarli
- RTO è un timer che serve a definire un lasso di tempo necessario prima di rimandare un segmento nel caso non si fosse ricevuto un ACK
- RTO varia in funziona di RTT e di un offset di tempo, quindi RTO = RTT + offset
- Essendo che RTT è dinamico allora anche RTO deve essere calcolato dinamicamente e variare durante la connessione:
    - si parte con un RTO elevato, il primo RTO quindi è di circa **3 secondi**
    - il secondo RTO si attiva alla ricezione del primo ACK e si calcola nel seguente modo:
        - ***RTO = RTT + K(D) con K=4 e D=RTT/2***
    - una volta superati i primi 2 ack della trasmissione, ogni volta che un ACK viene ricevuto prima che il timer scada, verrà effettuata una nuova misura dell'RTT, di D e dell' RTO:
        - ***RTTnew = a*RTTold + (1- a)M***
        - ***Dnew = B*Dold+(1 - B)*|RTTold - M|***
        - ***RTO = RTTnew+4(Dnew)***
        - dove:
            - *M = misura RTT appena effettuata (tempo in cui ricevo ACK per il segmento inviato)*
            - *a,B = fattore di smorzamento, circa 0,9 o 7/8*

Retransmission Ambiguity Problem

- se un segmento viene perso subito all'inizio, i calcoli di RTO non sono affidabili, stesso vale se RTO scada prima che arrivi l'ack per colpa della rete congestionata o altro.
- Per questa ragione si introduce il **timer di Kam** che dice che se si verifica un errore o un timeout allora:

RTOnew = 2*RTOold

Opzione TimeStamp

- il calcolo dell'RTT non avviene sempre per ogni segmento inviato, ma circa ogni 2/3 segmenti trasmessi.
- Se si vogliono avere stime piu frequenti, bisogna dichiararlo in fase di apertura della connessione specificando opzione TimeStamp nell'header.
- ogni entità TCP possiede un time out a 32 bit incrementato ad intervalli.
    - Per ogni segmento inviato il mittente legge il valore del timer e lo inserisce nel segmento in corrispondenza del campo time-stamp value
    - quando il destinatario riceve il messaggio risponderà con il rispettivo ACK e includerà nel campo time-stasmp-echo-reply il valore letto nel campo time-stamp value cosi che il mittente possa misurare RTT sottraendo al valore corrente il valore ricevuto

Selective ACK

- ack selettivo invece che cumulativo
- permette di fare ack dell'ultimo pacchetto ricevuto in sequenza e l'ultimo pacchetto che potrebbe essere fuori sequenza. Appena ricevuto un ack fuori sequenza viene spedito il pacchetto corretto.
- nel caso TCP volesse utilizzare un ACK selettivo allora bisognerebbe aggiungere tale opzione nel campo options all'inizio dell'handshake TCP

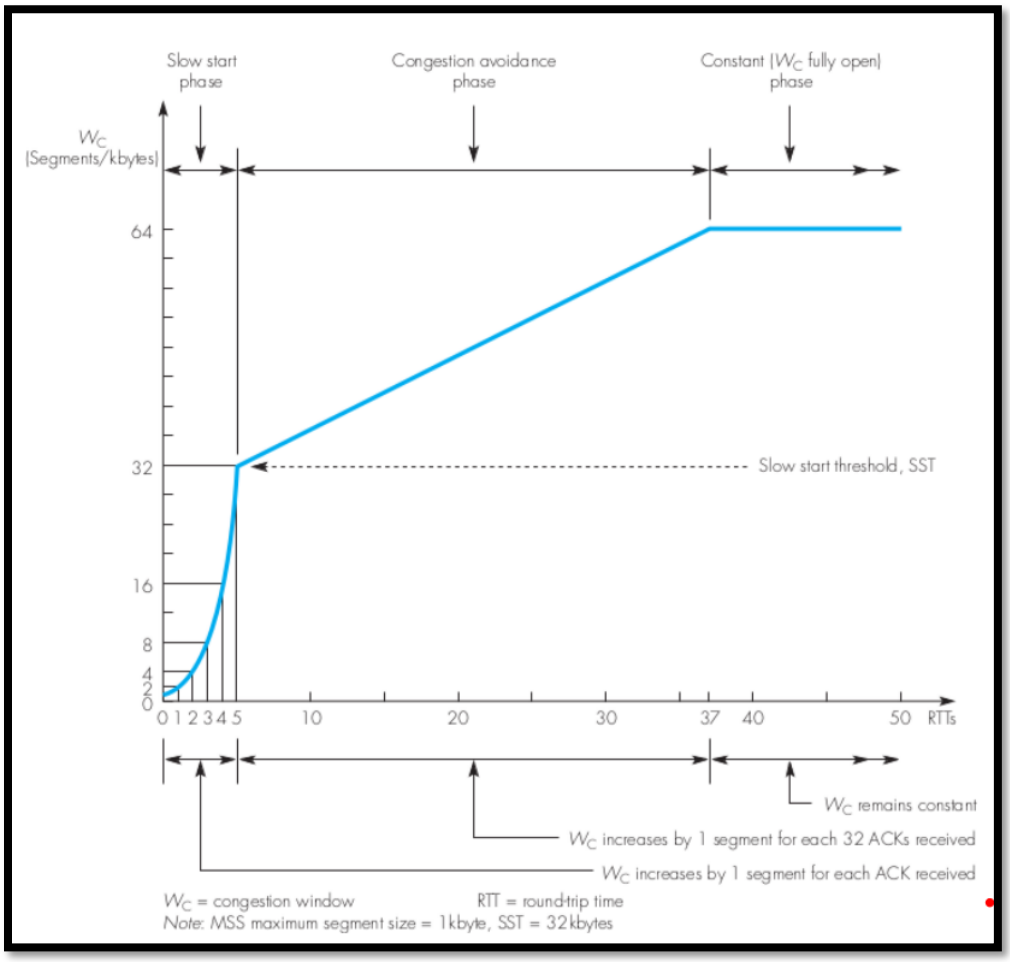
A.R.B 2022

Controllo della congestione

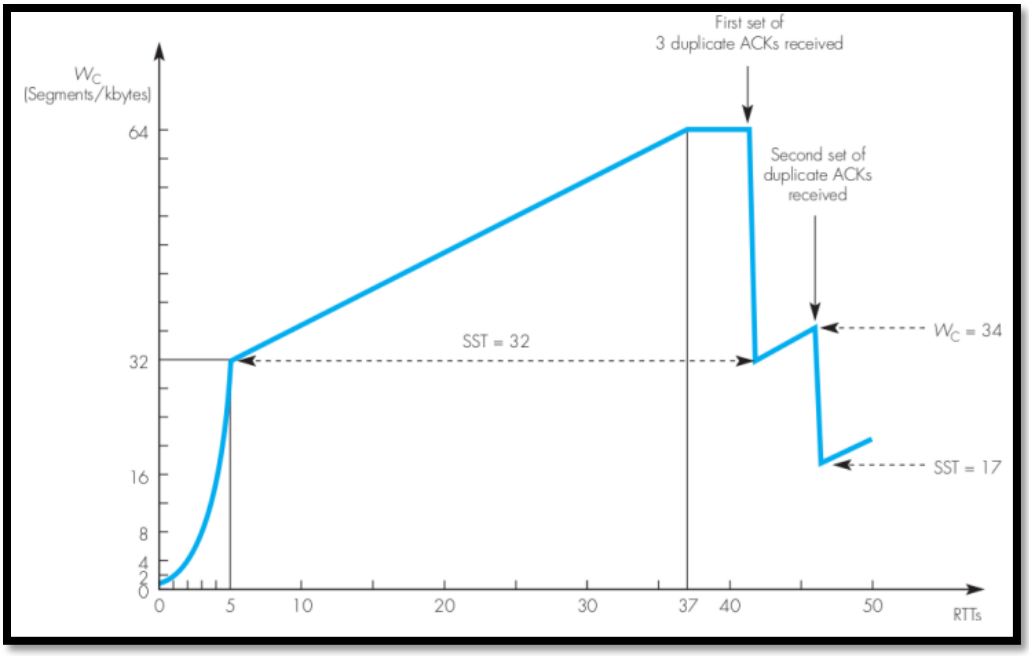
- TCP quando si avvia parte con una finestra piccola e man mano aumenta fino al limite imposto dal ricevitore.
- Quando avviene un errore si azzera tutto e si parte ancora con la piu piccola dimensione possibile
- nel caso non si verifichino errori, TCP divide il trasferimento in tre fasi:

1) **Slow Start**
   - ogni entità TCP mantiene una variabile **Wc** detta **congestion window variable**
   - all'apertura della connessione il mittente non conosce la capacità e setta quindi Wc = MSS
   - Il mittente invia un solo segmento e farà ripartire un timer di ritrasmissione
     - Se il timer è scaduto e il mittente NON ha ricevuto ACK, allora il segmento viene inviato di nuovo.
     - Se il timer non è scaduto e si riceve ACK, allora **$Wc = 2^{\#trasmissione} * MSS$**
     - la variabile Wc viene quindi incrementata esponenzialmente
   - la fase di slow start continua finchè
     - non si riceve un ack duplicato oppure
     - non scade un timer di ritrasmissione oppure
     - non si raggiunde la **SS Treshold**

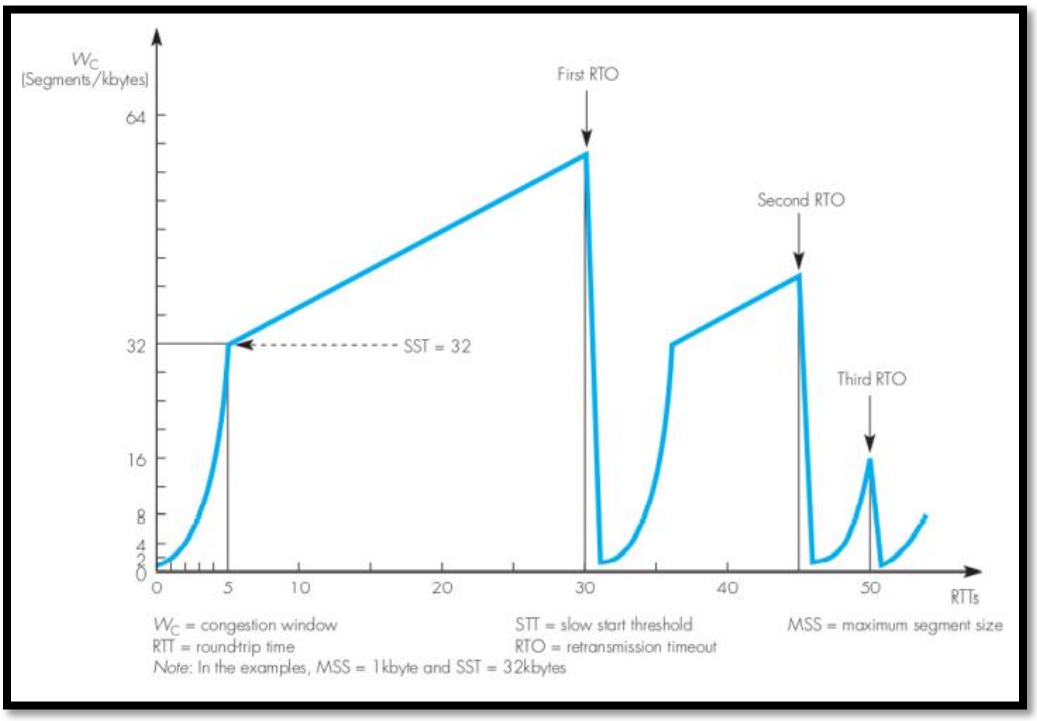
2) **Congestion Avoidance**
   - la variabile Wc viene incrementata linearmente aggiungendo MSS per ogni ACK ricevuto
   - questa fase continua fino a che non viene raggiunta la dimensione massima prevista in fase di setup della connessione oppure se raggiungo la dimensione del buffer receiver (ovvero Wc = Wr)
3) **Constant**
   - durante questa fase si spediscono finestre di congestione della stessa dimensione pari al massimo possibile, stabilito in fase di apertura della connessione



- come definito nel fast retransmit, nel controllo di flusso il destinatario manda 3 ACK duplicati. Ciò non è un problema e il livello di congestione viene considerato leggero e quindi alla ricezione del terzo ACK:
    - **SST = (Wc)/2**
    - **Wc = min(new SST, old SST)**
    - si procede con congestion avoidance



- un errore grave è l'**RTO Recovery.** Viene considerato grave poiché avviene quando si ipotizza che la rete sia congestionata. Quello che si fa quindi è
    - **SST = (Wc)/2**
    - **Wc = 1MSS**



Chiusura della connessione TCP

- TCP permette diversi tipi di chiusura:

1) NORMALE

- Client: Chiusura attiva
- Server: Chiusura passiva
- quando il client esegue la **close**:
  - Client manda un messaggio con bit FIN = 1 al server
  - Server riceve EOF ed effettua close validando il client
  - Server manda al client un segmento FIN = 1
  - Client valida con ACK il FIN del server
  - Client attende un timer dopo la ricevuta del FIN cosicchè se l'ack mandato dal client è stato perso un altro FIN arrivi al client dal server

2) SIMULTANEOUS CLOSE

- Avviene quando si parla di peer to peer:
  - Entrambi gli attori mandano FIN
  - Entrambi mandano ACK per il FIN ricevuto
  - Entrambi attendono per essere sicuri che l'ack arrivi

3) HALF CLOSE

- Solo una parte finisce l'invio dei dati mentre l'altra ha ancora segmenti pendenti
- Consideriamo come se il client richiedesse la chiusura e il server dovesse ancora finire di inviare dati:
  - dopo invio della FIN del client essa viene validata dal Server
  - prima di chiudere, il server aspetta che tutti i pacchetti pendenti siano inviati
  - Al termine viene effettuata la chiusura attiva della parte destra (FIN)
  - Client valida con ACK la chiusura del server

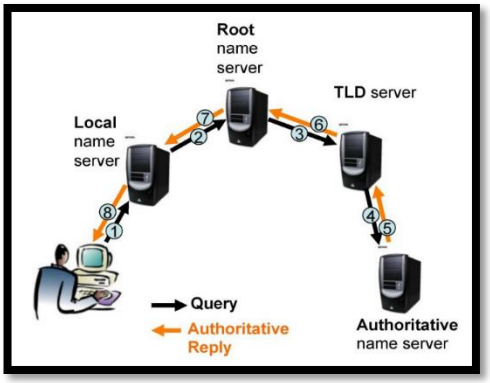
Livello 7: APPLICATION

Domain Name System

- il DNS permette ad ogni entità sulla rete di possedere un **nome simbolico**
- struttura DNS:
    - i dati utilizzati dal DNS si chiamano Domain Space Name e sono indicizzati per nome. La struttura è gerarchica.
    - un **Dominio** è uno spazio di nomi organizzato in modo gerarchico
    - un **Servizio** è il servizio offerto da quel dominio
- l'adozione di una struttura gerarchica facilita la risoluzione dei nomi che nella maggior parte dei casi vengono risolti in loco.
- Per ottenere questa gerarchica
    - lo spazio di dominio dei nomi è stato diviso in **zone**
    - ogni zona viene amministrata da un'autorità separata che fornisce uno o piu name server per zona.
- **FQDN** – **Fully Qualified Domain Name** = *"the complete domain name for a specific computer, or host, on the internet."*
- vi sono due modi di risolvere i nomi:

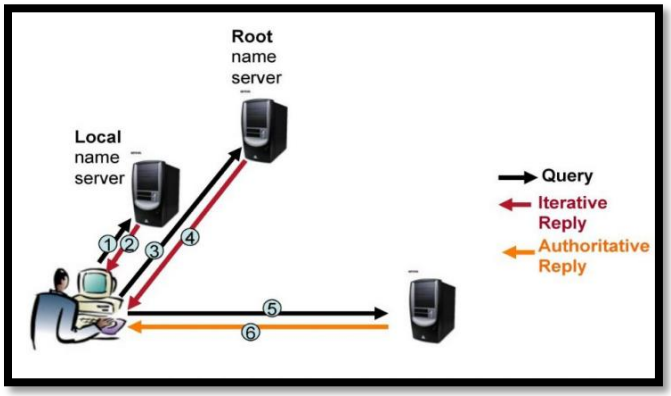
1) RICORSIVA

- *Nella Risoluzione DNS di tipo Ricorsiva, il client aspetta dal server DNS contattato la risposta alla sua richiesta. Il server DNS se è responsabile del dominio, risolve l'indirizzo altrimenti trasmette la richiesta ad un server DNS di livello superiore e aspetta la risposta per il client*
- viene utilizzato UDP
- tutti gli host si riferiscono al DNS locale per risolvere un indirizzo



2) ITERATIVA

- *il client invia una query al Local Name Server, esso verifica se il nome può essere convertito rispondendo al client con l'indirizzo IP corrispondente, altrimenti si limita a comunicargli il nome del server che secondo lui è in grado di risolvere il nome. Successivamente, il client ripete la procedura con il server DNS fornitogli.*



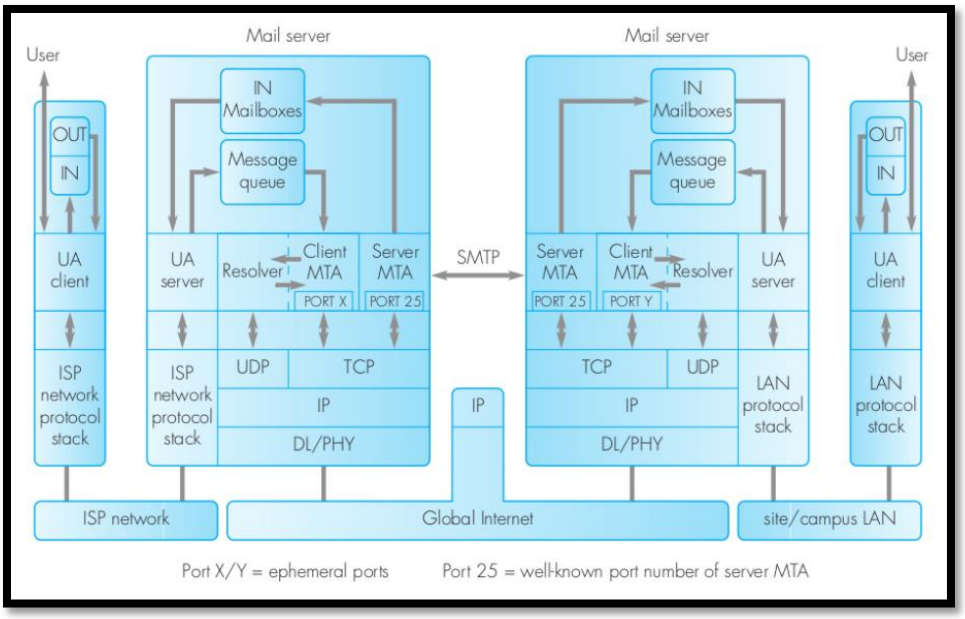
File Transfer Protocol

- in FTP i dati viaggiano separati dal controllo, e utilizzano due connessioni TCP distinte
    - Porta **21**: **Controllo** → aperta dal client
    - Porta **20**: **Dati** → aperta dal server

Posta Elettronica

- Le componenti funzionali per un mittente di posta elettronica sono:
    - **2 User Agent** = risiedono a livello 7 rispettivamente nella macchina client e l'altra nel server di posta.
        - **UA Client** gestise la scrittura/lettura per la gestione delle mail, consentendone la trasmissione in rete attraverso TCP. (I software di posta elettronica hanno il loro UA)
        - **UA Server** risiede fisicamente nel server di posta e permette di interagire con gli UA del client

    - **2 Message Transger Agent** = risiedono entrambi nel livello 7 del server di posta
        - **MTA Client** sincronizza i dati con i client mantenendo per ognuno di essi una IN mailbox nota comem essage store
        - **MTA Server** si occupa di cercare il serevr di posta elettronica del destinatario interrogando il DNS locale



- tutti i dati viaggiano in codifica ASCII
- **MIME - Multi Protocol Internet Mail Extension** permette l'invio di tipi di dati diversi attraverso la posta elettronica come dati binari, audio video ma mantenendo lo stesso modo di trasferimento dei messaggi. Esso introduce ovviamente nuovi campi nell'header della mailù

Codifica BASE 64

- per essere trasferiti, tutti i tipi non ASCII vengono convertiti in Base64:
    - Ogni dato viene diviso a blocchi di 24 bit
    - Ogni blocco viene diviso in 4 sottoblocchi da 6 bit ciascuno
    - i 3 sottoblocchi sono interpretati come ASCII
- esempio:

    si vuole codificare 1001111000101100.

    raggruppo in blocchi di 24 bit, se ce ne sono di meno si mette un uguale alla fine (in binario, 0011 1101)

    *la sequenza diventa quindi: 100111100010110***00111101**

    si divide il blocco in 4 sottoblocchi da 6 bit: 100111 - 100010 - 110000 - 111101

    si codifica ogni blocco utilizzando la tabella base64 e quindi in questo caso diventa: ***niw9***