Reti di calcolatori

- Livelli ISO/OSI
  - Livello 2 "Datalink"
    - Incapsulamento frame
      - Character stuffing
      - Bit stuffing
    - Tecniche per l'affidabilità
      - Idle RQ
      - Go-back-N ARQ
      - Selective Repeat ARQ
    - Protocolli affidabili
      - HDLC
      - PPP
    - DataLink nelle reti broadcast
      - Protocolli di accesso al canale condiviso
        - Token Ring
        - ALOHA
        - ALOHA con Carrier Sense (CS)
        - CSMA/CD
      - Formato frame LAN
        - Codifica Manchester
    - Sottolivelli del livello 2
      - LLC
    - Domini di collisione
      - Hub
      - Bridge
      - Switch
    - Approfondimenti su Ethernet
    - Approfondimenti su VLAN
  - Livello 3 "Network"
    - Livelli di accesso a Internet
    - Protocolli
      - IP
        - Frammentazione e ricostruzione pacchetto IP
        - Classi di indirizzamento
        - Subnetting
        - NAT (Network Address Translation)
        - CIDR (Classless Inter Domain Routing)
      - ARP
        - RARP

Introduzione

Tipologie di rete

- LAN (Local Area Network)
- MAN (Metropolitan Area Network)
- WAN (Wide Area Network)

Vi possono poi essere reti interconnesse, ovvero reti composte da elementi utilizzanti tecnologie potenzialmente diverse.

Componenti di una rete

- macchine utente
- sottorete di comunicazione
- gateway (ponte d'accesso LAN -> Esterno)
- router (instradatore da rete a rete)

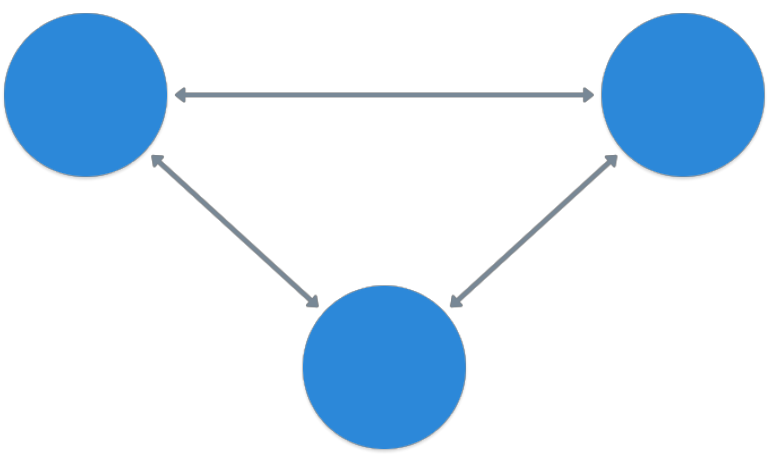
Topologie di rete

- reti magliate (punto a punto)
  - parzialmente connesse
  - completamente connesse
- reti broadcast
  - centro stella
  - ethernet
  - token ring

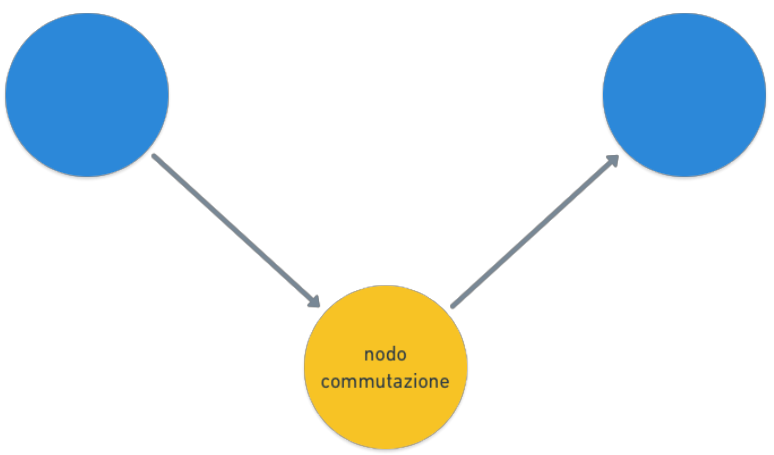
Reti magliate

Ogni nodo può comunicare con un altro nodo, ma potenzialmente non in modo diretto. Esistono nodi di commutazione per inoltrare un messaggio al nodo destinatario.

Durante l'inoltro di un messaggio si tiene traccia di quanti nodi di comunicazione il messaggio deve attraversare prima di arrivare a destinazione (**hop count**)



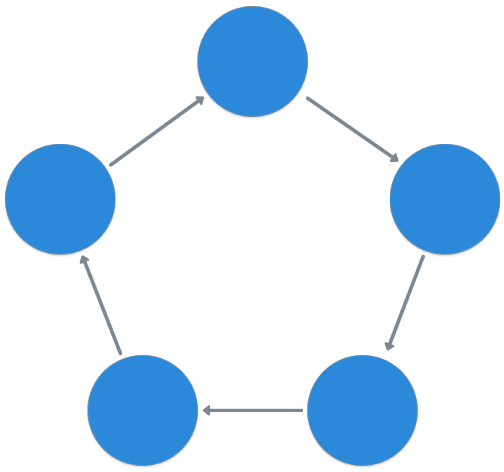
completamente connessa



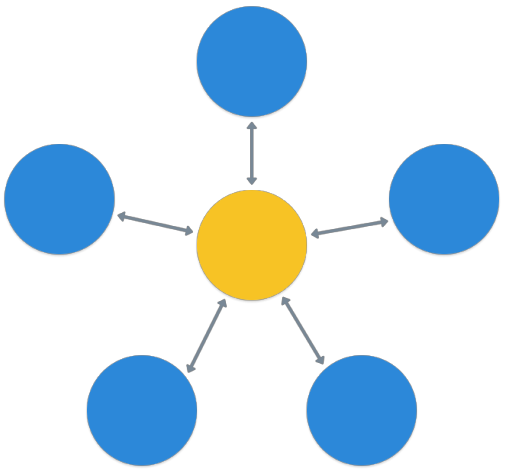
parzialmente connessa

Reti broadcast

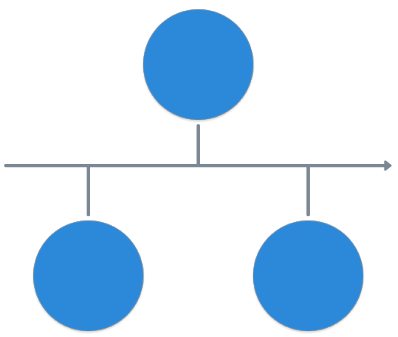
Ogni nodo riceve i messaggi di tutti gli altri.



token ring



centro stella



ethernet

Commutazione

- commutazione di messaggio
- commutazione di pacchetto

La commutazione è la scelta di un percorso (linea di ingresso - linea d'uscita) che serve per poter spedire un messaggio da un mittente ad un destinatario.

I dati da trasmettere vengono decorati con:

- un'intestazione iniziale (che identifica missente e destinatario)
- una coda che funge da codice rilevatore di errore
- possibili altre informazioni, tipo il protocollo usato, etc.

Intestazione	Dati	Coda

I messaggi spediti in rete possono corrompersi (rovinarsi).

Packet error rate: $P_{er} = 1 - (1 - b_{er})^n \cong \ n \cdot b_{er}$, dove b_{er} è la probabilità di errore sul singolo bit e n il numero di bit trasmessi.

Commutazione di messaggio

Il messaggio viene inviato così com'è (intero) nella rete.

Svantaggi:

- messaggio indivisibile può essere di grandi dimensioni
- difficile gestione delle cose nei nodi di commutazione (precedenza ai messaggi

piccoli, grandi, come fare?)
- difficile gestire ed allocare memoria per i messaggi nei nodi (dimensioni variabili)
- se corrotto, necessario reinviare tutto

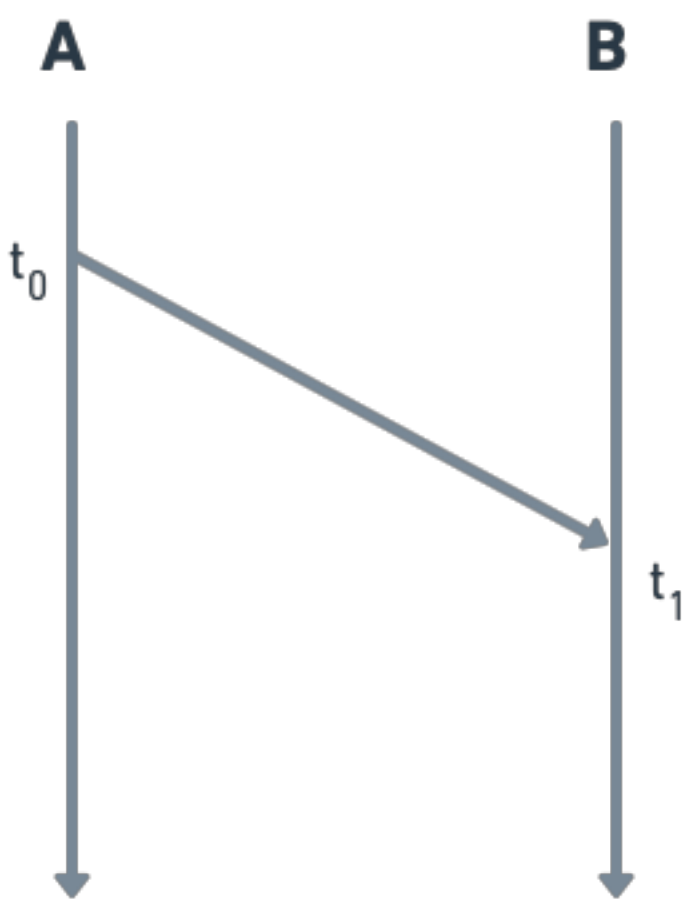
Commutazione di pacchetto

Il messaggio viene suddiviso in n pacchetti di dimensione standard. Risolve gli svantaggi della commutazione di messaggio.

Svantaggi:

- necessaria decomposizione/ricomposizione del messaggio
- ogni pacchetto ripete la stessa intestazione
- tempo di arrivo e ordine dei pacchetti non predicibile

Tempi di invio e ricezione pacchetti



Il tempo di latenza T è il tempo necessario per spedire un pacchetto, dalla presa in carico del pacchetto alla sua effettiva ricezione.

$$T = t_{coda} + t_{elaborazione} + t_{trasmissione} + t_{propagazione}$$

- t_{coda} è il tempo di permanenza del pacchetto nella coda del nodo
- $t_{elaborazione}$ è il tempo che serve al nodo per decidere su quale strada inoltrare il

pacchetto

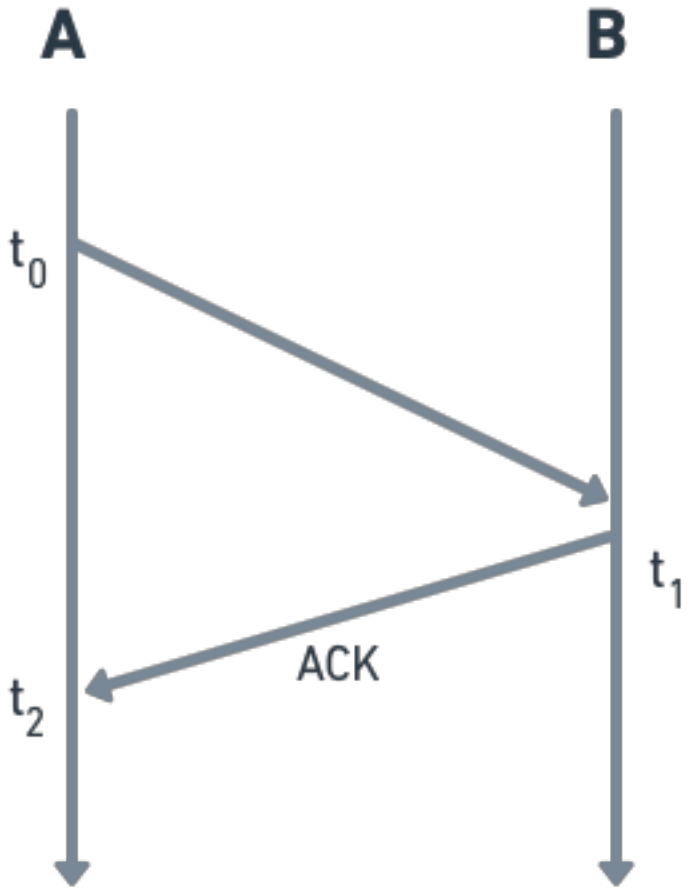
- $t_{trasmissione}$ è il tempo necessario per trasferire i dati dal nood sul mezzo fisico (ad esempio 1 bit per ogni colpo di clock)
- $t_{propagazione}$ è il tempo che impiega il pacchetto una volta sul mezzo trasmissivo a raggiungere il destinatario

Il t_{coda} e il $t_{elaborazione}$ sono generalmente trascurabili, mentre:

- $t_{trasmissione} = \frac{dimensione\ pacchetto}{bitrate}$
- $t_{propagazione} = \frac{distanza}{velocità\ mezzo}$

Nel tempo di trasmissione il $bitrate$ è dato dalla quantità di bit al secondo che il nodo può immettere nel mezzo trasmissivo, e varia da mezzo a mezzo, così come la velocità dello stesso.

ACK, RTT e RTD



All'arrivo del pacchetto, il destinatario invia un pacchetto particolare chiamato ACK, che informa il mittente che il pacchetto è giunto a destinazione.

Il tempo $t_2 - t_0$ è chiamato **Round Trip Time (RTT)**, detto anche tempo di andata e ritorno (o PING), ed è composto generalmente da $t_x + 2t_p$ (ovvero sia tempo di trasmissione che di propagazione, assumendo $t_x = 0$ per l'ACK).

Il solo tempo di propagazione andata/ritorno ($2t_p$) invece è chiamato **Round Trip Delay (RTD)**

Allora $RTT \div 2$ sarà circa il tempo di latenza T della rete in quel momento.

Utilizzo del canale

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

U simboleggia l'utilizzo del canale trasmissivo. Più è alto e meglio viene usato il canale. Se abbiamo U del 33% ad esempio, possiamo inviare più pacchetti prima di aver ricevuto un ACK sui primi inviati.
Precisamente potremmo inviare k pacchetti, per far si che U raggiunga il 100%:

$100\% = k \cdot U$

> $100\% = 3 \cdot 33$, dove $U = 33$

Traffico in rete e Jitter

Il traffico in rete può essere *real time* o *non real time*. Ad esempio si verifica traffico real time per l'invio della voce in una chiamata internet (importante che i pacchetti vengano ricevuti con costanza) mentre non real time per l'invio di mail o file (tempi di invio e ricezione non dipendenti).

Nella trasmissione real time i pacchetti corrotti vengono eliminati (*package drop*), in quanto la perdita di un singolo pacchetto non comporta effetti collaterali all'utente finale (esempio di quando si sente la voce smorzata in chiamata).

È impossibiile che internet garantisca un tempo di latenza costante. La varianza tra i tempi di latenza dei vari pacchetti inviati si chiama **Jitter**.

Se ad esempio il primo pacchetto è arrivato nel tempo t_1 e il secondo nel tempo t_2, nella formula $t_1 = t_2 + a$, il jitter sarà a.

Per compensare eventuali ritardi nella trasmissione, il destinatario avrà un buffer *play-out* dove immagazzinerà alcuni pacchetti.

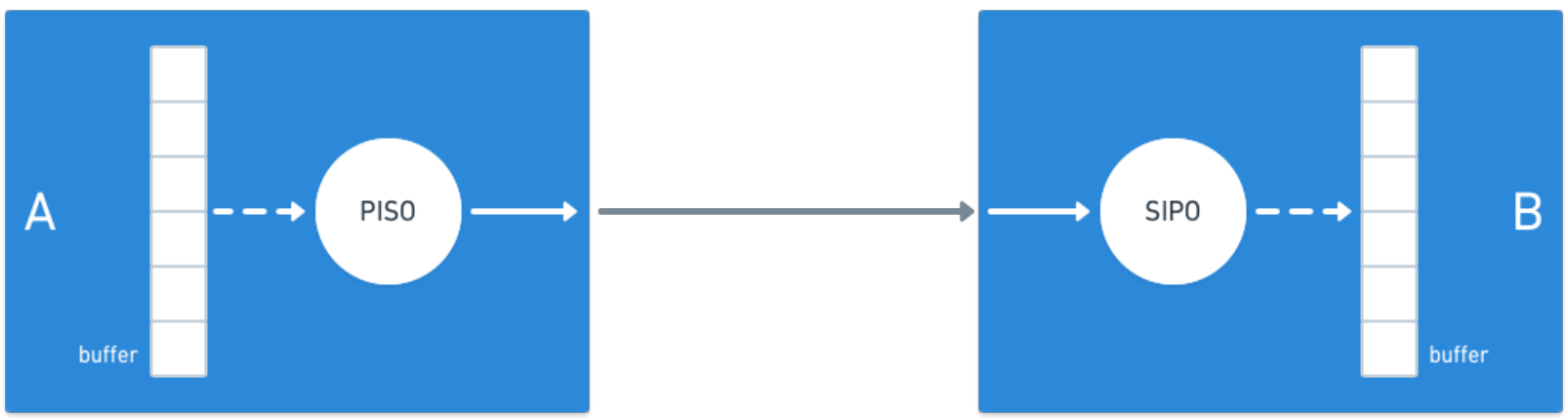
Modalità trasmissive

Constant e Variable Bit Rate

Le sorgenti di informazione possono emettere dati ad un ritmo binario costante (**CBR**) o a ritmo binario variabile (**VBR**).

Quelle costanti operano sempre al loro picco di efficienza, mentre quelle variabili possono attenuare la loro efficienza in base all'esigenza. Ad esempio se vi è un video digitale in streaming con immagini a basso movimento e colore uniforme, si può inviare meno informazione in un lasso di tempo fissato, potendo quindi abbassare il bitrate, per poi riaumentarlo in presenza di immagini ad altro movimento o colore vario. Opera generalmente in CBR invece il traffico realtime.

Invio e ricezione sul mezzo fisico



Un nodo della rete deve poter inviare una sequenza di bit su di un mezzo fisico. Questo avviene attraverso la serializzazione e la deserializzazione del messaggio da inviare e ricevere. Vi è un'unità PISO (*Parallel Input Serial Output*) in invio e un'unitò SIPO (*Serial Input Parallel Output*) in ricezione, che comunicano entrambe con un buffer che contiene i dati in modo "parallelo" (tutti insieme).

Comunicazione con e senza connessione

Comunicazione con connessione (orientata alla connesione) significa che viene usato un protocollo di comunicazione per stabilire una connessione logica o fisica *end-to-end* tra gli agenti della comunicazione, prima di trasmettere i dati.

> Es: in una stanza piena di gente, una persona A prima avverte la persona B che sta per parlare, poi parla. Di per se il parlare di A non specifica chi sia il ricevente, ma B lo sa perchè è avvenuta una preparazione alla comunicazione attraverso un protocollo (avvertire, etc).

Comunicazione senza connessione (non orientata alla connessione) significa che si inviano i pacchetti tra gli agenti della comunicazione senza mettersi prima "d'accordo", facendo si che ogni pacchetto di per se abbia le informazioni necessarie affinchè la

comunicazione avvenga tra gli agenti.

> Es: in una stanza piena di gente, una persona A mette una lettera chiusa sul tavolo con sopra scritto "lettera per B". B non sa che A vuole parlare, ma leggendo (così come potenzialmente anche altre persone) il sopra della lettera, capisce di essere il destinatario e legge la lettera.

##

##

ATTENZIONE: PARTI MANCANTI

- pagine da 14 a 17
- argomenti riguardanti:
  - trasmissione digitale
  - sincronizzazione
  - PCM (Pulse Code Modulation)
  - Time Division Multiplexing

##

##

Sottoreti di comunicazione

In una sottorete abbiamo:

- host
- router/nodo

Ma chi si deve preoccupare della correttezza della trasmissione dei pacchetti, ovvero che arrivino tutti alla destinazione, non duplicati e in ordine?

Sottorete affidabile

In una sottorete affidabile sono i nodi/router interni che gestiscono la correttezza della trasmissione dei pacchetti.
Questo significa che i nodi dovranno portarsi dietro informazioni aggiuntive circa il pacchetto che stanno trasmettendo.
Questo metodo era usato in passato a causa dei collegamenti tra nodi che potevano avere tasso di errore alto.

Sottorete non affidabile

In una sottorete non affidabile (anche detta *datagram* o *best-effort*) sono gli host che devono verificare la correttezza dei pacchetti inviati e ricevuti, mentre i nodi interni servono solo all'instradamento.

L'affidabilità è implementata al livello di trasporto (4°) degli host, attraverso il protocollo TCP (viene usato UDP invece quando non è necessaria affidabilità).

Livelli ISO/OSI nella sottorete

Livelli gestiti dai nodi

I nodi della sotterete lavorano solo con i primi 3 livelli del modello ISO/OSI:

- livello 1 (fisico)
- livello 2 (datalink)
- livello 3 (network)

I livelli fisico e datalink lavorano solamente su di uno specifico collegamento nodo-nodo, e non hanno visione globale della sotterete.

Livello 1 (fisico)

Si occupa solamente di inviare i bit nel mezzo trasmissivo, in modo diverso in base alle tecnologie usate da quest'ultimo.

Livello 2 (datalink)

Gestisce i dati come *frame*, invia al livello fisico una *stringa di bit* e qualora sia necessaria una sottorete affidabile controllerà anche la trasmissione nello specifico collegamento nodo-nodo

Livello 3 (network)

È l'unico ad avere una visione globale della sottorete ed è diviso in due parti:

- sottolivello 3A che gestisce l'instradamento intranet (visione globale della sottorete)
- sottolivello 3B che gestisce l'instradamento internet (visione globale della rete)

Livelli gestiti dagli host

- i primi 3 livelli visti per i nodi e in più...
- livello 4 (trasporto)
- ~~livello 5 (sessione)~~ *deprecato*
- ~~livello 6 (presentazione)~~ *deprecato*
- livello 7 (applicazione)

Livello 4 (trasporto)

Viene usato per l'affidabilità in caso di rete non affidabile (protocollo TCP) altrimenti è possibile non fare controlli sull'affidabilità (protocollo UDP).
Il protocollo UDP è utile anche nei casi in cui non è troppo importante un errore ed essendo molto veloce, la ritrasmissione di un pacchetto corrotto non è costosa.

Livello 7 (applicazione)

Contiene protocolli che forniscono supporto di comunicazione all'utente finale (es. mail, web, ftp, dns, ...). Non vengono qui descritte le applicazioni vere e proprie ma i loro protocolli di comunicazione.

La riduzione dei livelli ISO/OSI da 7 a 5 significa adottare lo stack di protocolli TCP/IP. Inoltre il livello 3 (network) negli host è fortemente semplificato, poichè non è presente il sottolivello 3A per l'instradamento nella sottrete, in quanto l'host comunica solo con il nodo più vicino.

Livelli ISO/OSI

Livello 2 "Datalink"

- Si occupa solo del collegamento tra due nodi

- Non sa quali altri nodi ci siano e non ha quindi informazioni su mittente o destinatario
- Limita il frame con un'intestazione e una coda

Incapsulamento frame

Normalmente al frame viene inserita un'intestazione `STX` (*Start Of Text*) e una coda `ETX` (*End Of Text*).

Siccome `STX` e `ETX` possono facilmente trovarsi nella parte dati del frame, illudendo quindi il mittente o il ricevitore del frame, ci sono dei metodi per essere sicuri di quali siano intestazione e coda:

- character stuffing
- bit stuffing

Character stuffing

Ai normali `STX` e `ETX` si sostituiscono `DELSTX` e `DELEXT`, dopodichè si duplicano ogni `DEL` trovati nella parte dati del frame.
In ricezione si eliminano i `DEL` duplicati nel body. Se quando si legge un `DEL` i caratteri successivi sono `ETX` vuol dire che sono arrivato in fondo (altrimenti troverei un altro `DEL`).

Bit stuffing

Stesso procedimento del character stuffing ma non si usano i caratteri ma i bit.
L'intestazione (e la coda) conterrà una sequenza di n `1` (es: `01111110`).
Ogni volta che nella parte dei dati si troverà $n-1$ `1` consecutivi gli si aggiungerà uno `0`.
Così solo l'intestazione (e la coda) avrà una sequenza di n `1` consecutivi, permettendo quindi di distinguerle dalla parte dati.

Tecniche per l'affidabilità

Anche se non sono più usati, ci sono alcuni protocolli per garantire l'affidabilità a livello 2, che usano il metodo **ARQ** (Automatic Repeat Request), che prevede la ritrasmissione del pacchetto dopp un tempo di *timeout* fino a quando il destinatario non ne abbia

confermato la ricezione (con un ACK):

- Idle RQ (stop-and-wait ARQ)
- Continuous RQ:
  - go-back-N ARQ
  - selective repeat ARQ

Idle RQ

Se dopo un tempo $t_x > RTT$, l'ACK non è ancora arrivato, il mittente reinvia il frame.
Ogni frame ha un numero progressivo (*numero di sequenza*) partendo da 0, così il ricevente può sapere se ha già ricevuto quel frame.
Il ricevente in ogni caso risponde con l'ACK, anche se ha già il frame in arrivo.

Siccome il mittente potrebbe aspettare troppo poco per l'ACK, anche gli ACK vengono numerati, così che il mittente possa capire quale ACK si riferisce a quale frame.

Se il ricevente deve mandare a sua volta dei frame e un ACK al mittente può mandarli insieme, stando attento che il tempo di preparazione del suo frame non vada oltre il tempo di *timeout* del mittente (**piggybacking**).

Se il RTT è dominato dal tempo di trasmissione, il mittente non può migliorare di molto la velocità totale.
Se invece domina il tempo di propagazione, il mittente può inviare k frame e ascoltare poi per gli ACK senza aspettare il primo ACK del ricevente per inviare un secondo frame.
Così facendo migliora l'utilizzo del canale e si parlerà di **protocollo a finestra di dimensione k**

Go-back-N ARQ

È un protocollo a finestra di dimensione k dove si invia una sequenza di frame k_0, k_1, ..., k_n.
Se in tale sequenza di perde o arriva un frame corrotto al ricevente, si ritrasmette tutta la sequenza a partire dal frame che ha avuto problemi.
Il ricevente, per segnalare il frame errato, trasmette un segnale *NAK* (negative o non ack) che rileva il fallimento prima dello scadere del timer del mittente.
Con questa politica si rileva in anticipo un fallimento. Bisogna anche aggiustare i numeri di sequenza dei frame in modo tale che coprano tutti i k frame.

È vero che si spreca tempo reinviando tutti i frame (per garantire l'ordine di arrivo), ma il ricevente non deve allocare memoria in quanto lavora con l'ultimo pacchetto arrivato e

basta.
Perciò si utilizza in genere quando la memoria del ricevente è poca, quando ho finestre di piccole dimensioni o di dimensioni grandi ma con tasso di errore basso.

Selective Repeat ARQ

È come go-back-N ARQ ma non si reinviano tutti i frame ma solo quello errato.
Serve quindi che il ricevitore abbia un buffer almeno di k elementi così da gestire gli altri frame in attesa del reinvio di quello errato.

Quando il frame perso viene ritrasmesso al ricevitore si hanno due strategie per validare il frame:

- ACK selettivo, dove convalido ogni frame da quello errato in poi con un ACK ciascuno
- ACK cumulativo, dove aspetto a inviare gli ACK dei frame successivi a quello corrotto e invio un ACK singolo che vale per tutti quelli da quello corrotto a quello indicato nell'ACK (l'attesa rischia di far scattare il timer del trasmettitore)

Quando il ricevente invia un NAK e quando il trasmettitore riceve un NAK si dice che si entra in *retransmission mode*.

Qualora il trasmettitore non riceva un segnale di ACK, con le due strategie si avrà:

- ACK selettivo, potrebbe essere un problema
- ACK cumulativo, nessun problema, dato che vengono convalidati tutti i pacchetti precedenti

Data una finestra di dimensione k, si avrà bisogno di n numeri di sequenza ($0, 1, ..., n-1$), dove n è:

- $k+1$ per Idle RQ e Go-back-N ARQ
- $2k-1$ per Selective Repeat ARQ

per evitare che un'eventuale ritrasmissione della finestra (o frame) a causa di errori venga interpretata come la finestra (o frame) successiva.

Protocolli affidabili

- HDLC (High-Level Data Link Control)
- PPP (Point-to-Point Protocol)

HDLC

Utilizza il bit stuffing per incapsulare il frame.
I frame inviati dal mittente vengono chiamati *commands* e quelli ricevuti dal mittente *responses*.

Formato frame (il campo *controllo* è quello dei numeri di sequenza):

Flag	Indirizzo	Controllo	Dati	FCS	Flag
8 bit	8 bit	8 o 16 bit	lunghezza variabile, 0 o più bit multipli di 8	16 o 32 bit	8 bit

PPP

Fornisce metodo standard per trasmettere sullo stesso canale pacchetti generati da diversi protocolli di livello superiore (supporto multiprotocollo) e nasce per evitare una proliferazione eccessiva di protocolli di livello 2.

È diviso in due parti:

- LCP (Link Control Protocol) che si occupa di stabilire un link tra le due parti con le opzioni richieste
- NCP (Network Control Protocol) che si occupa di selezionare e configurare un protocollo di rete comune tra sorgente e destinazione

Formato frame:

Flag	Indirizzo	Controllo	Protocollo	Dati	Check	Flag
8 bit	8 bit	8 bit	8 o 16 bit	lunghezza variabile, 0 o più bit multipli di 8	16 o 32 bit	8 bit

dove:

- Indirizzo = `11111111` (sempre)
- Control = `00000011` (sempre)
- Protocol serve a negoziare il protocollo di livello 3 disponibile sulla macchina e indica il protocollo usato per generare i dati
- Check è il codice CRC (Cyclic Redundancy Code) per error detection

DataLink nelle reti broadcast

Nelle reti broadcast il frame viene inviato a tutte le stazioni della LAN.
I ricevitori non destinatario dovranno *droppare* il frame.
In casi broadcast il frame conterrà info sul sorgente e destinatario.
Siccome il canale è condiviso, servono dei protocolli di *mutua esclusione* o di *accesso unico al canale*.

Esistono diversi protocolli di accesso al canale condiviso:

- deterministici
- casuali

Protocolli di accesso al canale condiviso

- Token Ring
- ALOHA
- ALOHA con CS
- CSMA/CD

Token Ring

Si ha un token, viene passato in modo circolare.
Solo chi ha il token in un determinato momento può inviare frame.

Svantaggi:

- c'è comunque il traffico del token nella rete
- se ho tanti nodi, ognuno deve aspettare che il token faccia il giro da tutti
- se si perde il token? Qualcuno a controllare e poi si rigenera? Chi lo rigenera?
- modifica dei nodi complicata, devo riconfigurare il giro del token

Alternative:

- metto un centro stella attivo, che ha una coda di messaggi da rispedire
- temporizzo i nodi, ogni nodo può parlare per un determinato tempo

Con controllo centralizzato diventa improponibile se ho molti nodi.
Con controllo distribuito sono i nodi a controllare.

ALOHA

Controllo distribuito sui nodi.
Ogni nodo invia quando vuole (affindandosi alla probabilità) e gli errori/corruzioni vengono risolti ritentando (sfruttamento rete al 18%, ma garantisce fairness).

ALOHA con Carrier Sense (CS)

Un nodo verifica che il canale sia libero prima di inviare (con un polling).
Se due (o più) polling si verificano simultaneamente il canale risulterà occupato.
Il polling consiste nell'inviare un segnale X che tornerà anche al mittente. Se il segnale in ricezione è lo stesso inviato, il canale è libero (no disturbi).
Quindi si aspetta un tempo casuale per riascoltare il canale e riprovare

3 modalità:

- CSMA 1 persistente: continuo ad ascoltare il canale, appena trovo libero trasmetto
- CSMA non-persistente: non trasmetto appena il canale si libera ma aspetto un tempo casuale (*backoff*)
- CSMA p-persistente: quando si libera il canale trasmetto con una probabilità p, mentre con probabilità $(1-p)$ aspetto un tempo casuale

> Ethernet (802.3) è di tipo CSMA/CD (CSMA 1 persistente, con attesa casuale dopo collisione).
> CSMA/CD sta per Carrier Sense Multiple Access / Collision Detection

CSMA/CD

L'attesa casuale è in un range Binary Exponential Backoff (BEB):
si prende un numero random tra 0 e $2^i - 1$ e lo si moltiplica per $51.2\mu s$ che è lo slot BEB di tempo minimo con cui si è sicuri di rilevare una possibile collisione, dove i è il numero di collisioni avvenute.

Per poter rilevare una collisione devo avere $t_x \geq 2t_p$, così ho il tempo necessario per controllare che i bit che sto inviando si sono o meno corrotti e nel caso fermarne l'invio. Altrimenti non saprei se i bit che ho trasmesso sono bit corrotti a causa di una collisione e se integri se sono i miei, magari proveniente da un'altra stazione che nel frattempo ha trasmesso.

Quando una stazione rileva una collisione invia altri 32 bit detti **jam sequence** per essere certi che tutte le stazioni coinvole nella collisione se ne accorgano.

Ethernet impone che $t_x \geq 51.2\mu s$ che include, oltre a $2t_p$, il ritardo dei ripetitori e una maggiorazione "per stare tranquilli".
È calcolato pensando alla massima lunghezza di una rete Ethernet, cioè 2.5Km.
Siccome in $51.2\mu s$ riesco a mandare (minimo) 64 Byte, tutti i frame Ethernet devono avere almeno questa lunghezza (altrimenti non avrò più $t_x \geq 2t_p$).

Formato frame LAN

Preamble	SFD	DA	SA	Type/Length	Dati	FCS
7 Byte	1 Byte	2:6 Byte	2:6 Byte	2 Byte	46:1500 Byte	4 Byte

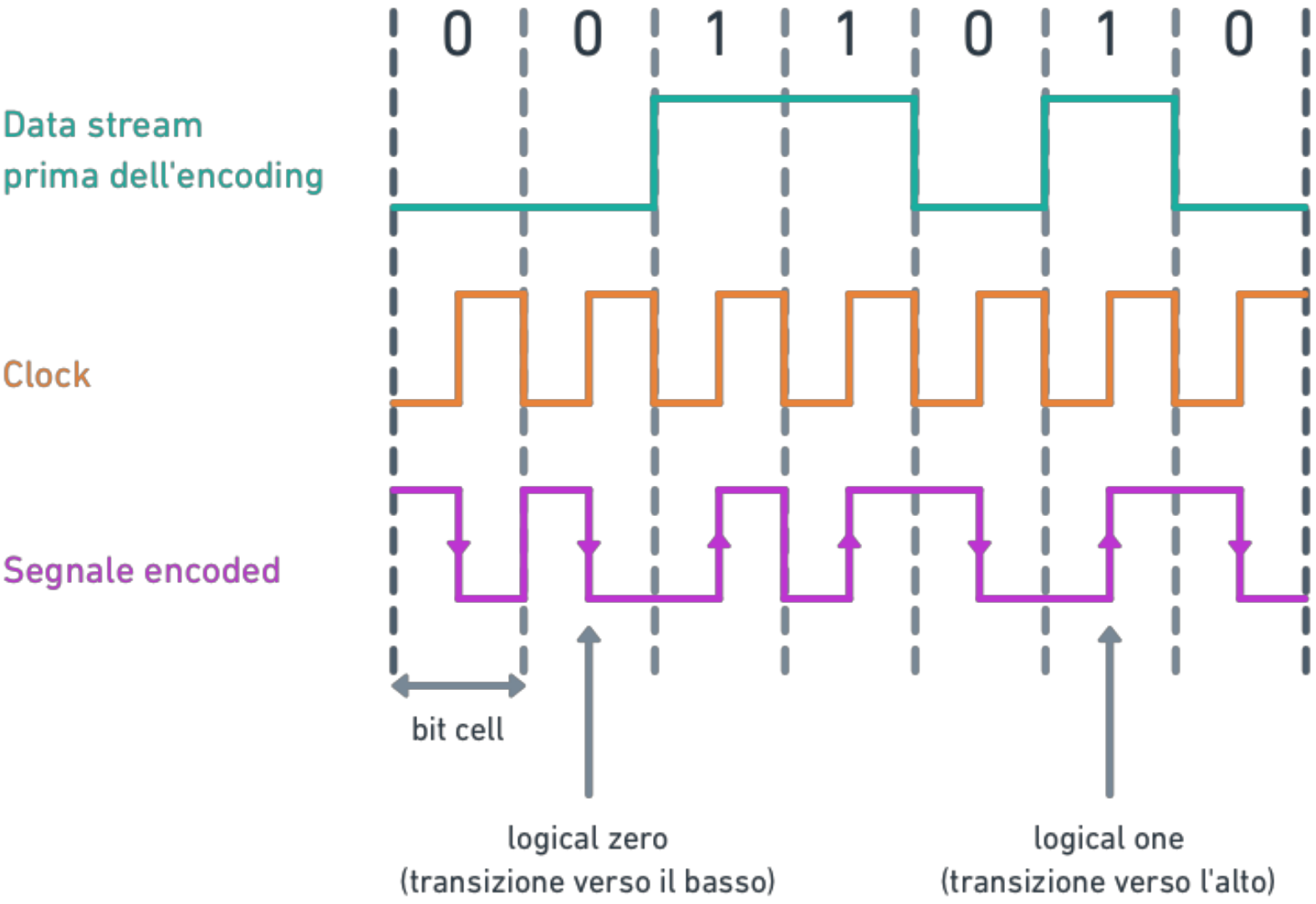
- Preamble (preambolo) ha valore `10101010` e servono a "svegliare" gli adattatori del ricevente e a sincronizzare gli oscillatori con quelli del mittente
- SFD (Start Frame Delimiter) ha valore `10101011` (i due bit a 1 consecutivi indicano che il messaggio è importante) e svolte la stessa funzione del campo flag dell'HDLC
- DA (Destination MAC Address)
- SA (Source MAC Address)
- Type/Length indica la lunghezza del campo dati (veniva usato da PPP per scegliere il protocollo di livello 3)
- Dati che vengono paddati a 46 Byte qualora ne dovessero essere spediti di meno (per il discorso delle collisioni)
- FCS (Frame Check Sequence) o *controllo a ridondanza ciclica* (CRC) che è il risultato di un operaazione matematica sui bit contenuti nel frame a cui viene accodato, e viene ricalcolato dal destinatario che così compara i due per verificare eventuali errori

Ogni nodo collegato ha quindi un indirizzo MAC a 6 Byte (48 bit di indirizzamento) univoco.

Codifica Manchester

Per poter far capire ai nodi la differenza tra assenza di segnale e la trasmissione dello 0 si utilizza la codifica Manchester:

- ogni 1 è rappresentato da una transizione da 0 a 1
- ogni 0 è rappresentato da una transizione da 1 a 0
- l'assenza di segnale è rappresentata dall'assenza di transizioni

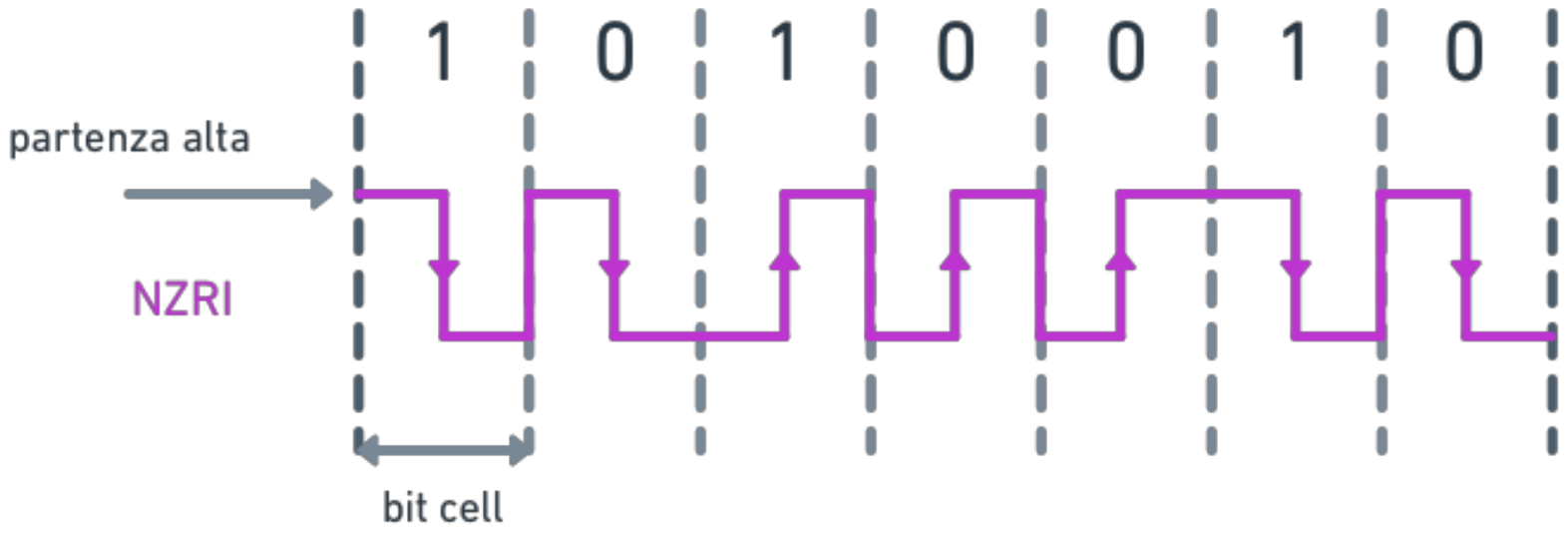


Il ricevitore, per capire quando arriva il dato, userà il preambolo per "allinearsi".

Vi è anche la variante del Manchester ma con transizioni invertite (basso/alto per `0` e alto/basso per `1`).

La codifica Manchester permette anche la sincronizzazione degli "orologi" di mittente e destinatario.

Vi è poi la codifica **NRZI** (Non Return to Zero Inverted) che in corrispondenza dello `0` mantiene il segnale così come lo aveva trovato e in corrispondenza dell'`1` lo cambia (ha bisogno di definire il punto di inizio del segnale, alto o basso).
Ad ogni bit successivo si effettua quindi la transizione fatta precedentemente (se `0`) o si cambia la transizione (se `1`)



Sottolivelli del livello 2

Il livello 2 si divide in:

- **LLC** (Logical Link Control), la parte "alta"
- **MAC** (Media Access Control), la parte bassa

La parte alta è comune a tutti gli standard della famiglia IEEE 802, mentre la parte bassa ha diverse implementazioni (una tra queste il CSMA/CD).

Siccome i problemi riguardanti la condivisione del canale e l'univocità delle macchine (indirizzo MAC) sono risolte dalla parte bassa MAC, l'LLC fornisce l'affidabilità (se richiesta) e le funzionalità punto-punto (PPP).

LLC

A seconda delle implementazioni, l'LLC può fornire al livello superiore diverse modalià di servizio:

1. **Logical Data Link**, che è non affidabile e non orientato alla connessione. Può inviare a un altro punto, a più punti o broadcast e non si interessa dell'ordine dei pacchetti, della loro correttezza o della loro effettiva ricezione.
2. **Data Link Connection**, che è affidabile e orientato alla connessione. Sono presenti meccanismi di correzione degli errori e di garanzia dell'ordine dei pacchetti (praticamente l'HDLC).

Domini di collisione

Usando ethernet, se abbiamo un solo cavo abbiamo un solo *dominio di collisione*. Ovverò una sola realtà fisica collegata dove possono avvenire collisioni.

Ci sono alcuni componenti che possiamo usare al posto del semplice cavo:

- hub
- bridge
- switch

Hub

Un hub è un centro stella passivo di livello fisico che semplicemente riceve e ritrasmette il segnale (funziona da ripetitore). Non separa/crea domini di collisione. L'hub non interviene a livello 2.

Bridge

Il Bridge è un hub che lavora anche a livello 2, separando quindi i domini di collisione (a cui è collegato). Possiede una scheda di rete con relativo indirizzo MAC, indirizza sull'uscita corretta i pacchetti e controlla che non siano corrotti, prima di inoltrarli.

Per instradare pacchetti da un dominio a un altro il bridge possiede una **tabella di forwarding**, che opera in questo modo:

1. all'inizio la tabella è vuota
2. al bridge arriva un frame da A
3. il bridge salva nella tabella da dove è arrivato il frame di A (da quale porta del bridge)
4. il bridge instrada il frame verso B a seconda che:
    - la porta di B non è nota: il frame viene inoltrato a tutte le porte tranne quella di A (*fooding*)
    - la porta di B è nota ed è la stessa di A: il frame non viene inoltrato (gli sarà già arrivato siccome è nello stesso dominio di collisione)
    - la porta di B è nota e non appartiene allo stesso dominio di collisione di quella di A: il frame viene inoltrato nella porta del dominio di B
5. nel caso B rispondesse, si riparte dal punto 2

Dopo un certo periodo la tabella viene resettata e ricostruita (per tenerla aggiornata).

Vi sono due tecniche aggiuntive/alternative per i bridge:

- **Spanning Tree Algorithm** che serve per evitare che si vengano a creare circoli viziosi per cui un frame continui a girare eternamente mantenendo occupata la rete (mandato nella porta X, ritorna nella porta Y). Per farlo si crea un albero dei percorsi seguendo un algoritmo spanning-tree (dal più corto al più lungo) e si inoltrano i frame sempre verso il percorso più breve. Se il percorso scelto venisse a

mancare si passa al secondo, etc.
- **Source Routing**, tecnica tipica dell'instradamento in reti token ring, per cui il frame in arrivo al bridge ha già le informazioni su quale percorso prendere, semplificando quindi il lavoro del bridge stesso.

Switch

È come un bridge ma non usa CSMA/CD perchè opera con collegamenti punto a punto (non più broadcast), può lavorare contemporaneamente più frame (ed è più veloce).

In fase di creazione della tabella di forwarding (quando non sa ancora dove inoltrare il destinatario) rispedisce il frame in tutte le porte.

Utile ad esempio per collegare i bridge e un server, in modo da funzionare da buffer per il server, che così non è in un dominio broadcast di collisione.

Lo switch è così veloce perchè lavora a livello hardware e non software. Può lavorare in due modalità:

- *store-and-forward*, il frame arriva, si legge il destinatario e si incanala a livello hardware verso la porta giusta, ma si fanno anche i controlli CRC e ci si accerta di riceverlo per intero (bisogna quindi aspettare tutto il frame prima di reinviarlo)
- *cut-through*, il frame arriva, si legge il destinatario e si incanala a livello hardware verso la porta giusta e lo si comincia a spedire senza effettuare alcun controllo (ancor prima di riceverlo tutto)

Approfondimenti su Ethernet

Prevede esclusivamente trasmissioni via cavo in banda base a velocità di 10, 100 e 1000 Mbps. Per differenziare le varie tecnologie usate viene usato l'acronimo *NBaseA*, dove *N* è la velocità di connessione, *Base* indica che è in banda base e *A* è una sigla legata al tipo di cavo utilizzato ed altre caratteristiche.

Da 100Mpbs si chiama *Fast Ethernet*.

Vi sono:

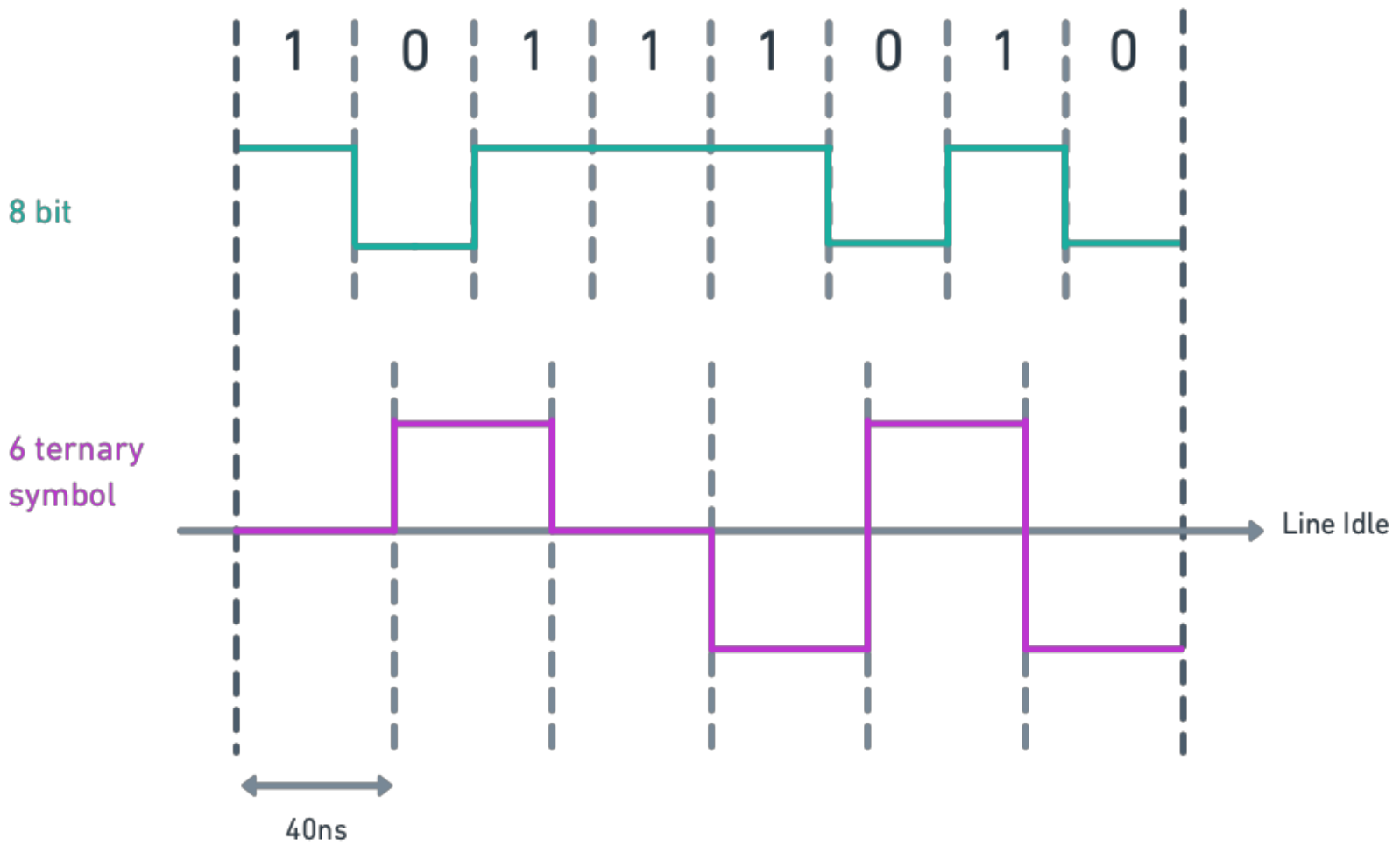
- **10Base5**, cavo coassiale (fili spessi), lunghezza massima dei segmenti 500m
- **10Base2**, cavo coassiale (fili sottili), lunghezza massima dei segmenti 185m
- **10BaseT**, topologia hub con una coppia di cavi intrecciati, lungheza massima dei segmenti 100m
- **10BaseF**, topologia hub con fibra ottica, lunghezza massima dei segmenti 2Km

- **100BaseFX**, fibra ottica, lunghezza massima dei segmenti 308m
- **100BaseT4**, doppino di categoria 3 (poco pregiato) ritorto a 4 coppie
- **100BaseTX** doppino di categoria 5 (più pregiato) ed è l'unica full-duplex

Attualmente vengono installate (tra le più diffuse) le varianti a 100Mbps e superiori, come la **100BaseT**.

> La "T" di "*NBase*T" sta per "Twisted Pair", doppino ritorto, mentre "T4" ad esempio, significa ritorto a 4 coppie.

Il raggiungimento della velocità di 100Mpbs ha però reso impossibile usare la codifica Manchester. Si usa quindi la codifica **8B6T**, che converte 8 bit in 6 simboli in base 3, basati sulla fase del segnale:



Mentre per la 100BaseTX si usa la codifica **4B5B**.

Per l'ethernet a 1000Mpbs si ha una lunghezza massima del cavo di 10m. Ma si ha anche una grandezza minima del frame da 512 Byte. Per ovviare al problema di mandare frame così grandi ci sono due alternative:

- *Carrier Extension*, ovvero padding con bit inutili che verrano poi rimossi
- *Clustering*, dove metto insieme più frame della grandezza standard di 64 Byte e mando tutto insieme (se necessario paddando anche dopo un intervallo di tempo).

Approfondimenti su VLAN

Le VLAN sono delle LAN che non sono tali a causa di dove sono fisicamente i nodi, ma per fini organizzativi. Si vuole logicamente dividere i nodi in gruppi funzionali, che potrebbero anche differire per il cablaggio usato.

Come facciamo però ad impedire alcune comunicazioni e permetterne altre?

Nelle schede MAC dei bridge viene aggiunta l'informazione riguardo al gruppo di appartenenza del nodo, un identificatore VLAN (detto *colore*) e memorizzo quest'informazione nella tabella di forwarding, che diventa:

Nodo	Porta	Vlan
A	1	1
B	1	1
C	1	2
D	2	1
E	2	2
F	2	1

Anche il frame dovrà contenere quest'informazione. Infatti nello standard **802.1Q** il frame risulta 4 Byte più lungo. In questi 4 Byte ci sono:

- PRI, un campo sulla priorità
- CFI, selettore formato del frame per compatibilità con token ring
- VLAN ID, identificatore della VLAN, è un valore maggiore di 1500 così non rischio di scambiarlo per la possibile lunghezza del dato (dello standard 802.3)

Livello 3 "Network"

Si occupa di:

- indirizzamento, identificare univocamente un nodo di una sottorete
- instradamento, come instradare un pacchetto tra sottoreti diverse

Come il livello 2 è diviso in due parti:

- 3A, Rete (intra-rete), instrada i pacchetti all'interno della sottorete. Se la sottorete è

broadcast non serve
- 3B, Inter-rete (IP - Internet Protocol), fornisce l'indirizzo IP che è univoco e specifica i protocolli per instradare tra sottoreti e le regole perchè il tutto funzioni con questi indirizzi

Livelli di accesso a Internet

Esistono diversi livelli tramite i quali Internet è raggiungibile:

- Tier 3 (Livello di accesso), ovvero tutte le sottoreti locali (aziende, dipartimento universitario, etc)
- Tier 2, ovvero gli ISP nazionali o regionali che uniscono le diverse sottoreti del Tier 3
- Tier 1, ovvero gli ISP internazionali che uniscono gli ISP nazionali (ce ne sono una ventina al mondo), collegando quindi continenti e nazioni diverse

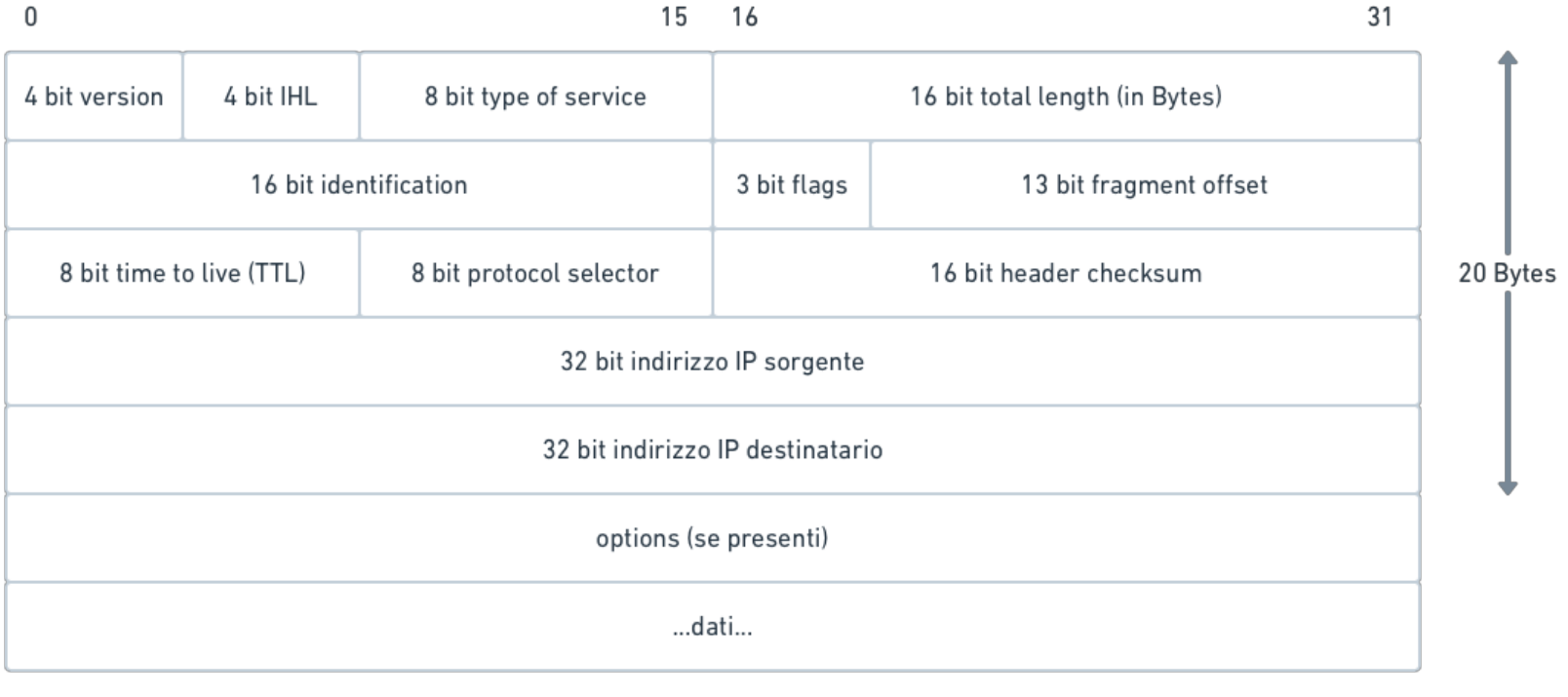
Tra il Tier 1 e il 2 ci sono i **NAP**, switch a 10 Gb che smistano tutto il traffico nazionale.

Protocolli

- IP
- ICMP
- ARP

IP

Non ci sono più frame ma si chiamano *pacchetti* e fornisce un servizio non affidabile. L'intestazione del pacchetto è organizzata in 5 parole da 32bit e un campo opzionale:



- **Version**, specifica la versione di IP (in questo caso 4) e istruisce quindi gateway,

router e host su come interpretare il pacchetto

- **IHL (Internet Header Length)**, la lunghezza dell'header può essere variabile (campo opzionale), in multipli di 32 bit. Il minimo è 5, il massimo è 15
- **ToS (Type of Service)**, fornisce una priorità al pacchetto IP, cosicchè i router possano instradarlo nel modo migliore
- **Total Length**, indica la lunghezza totale del pacchetto, compresa intestazione e dati. La dimensione massima è $2^{16} - 1 = 65535$
- **Parola di 32 bit per la frammentazione dei pacchetti**
  - **Identification**, identificatore del pacchetto originale, uguale per ogni pacchetto di un singolo messaggio
  - **3 bit Flags**, composti da un bit riservato, uno *DF (Don't Fragment)* per indicare di farlo passare in una rete che lo può spostare intero e uno *MF (More Fragment)* a `1` quando devono essere inviati ancora dei pacchetti, a `0` quando è l'ultimo
  - **Fragment offset**, indica quanti bit sono già stati inviati dai frammenti precedenti; non vengono contati i bit ma gruppi di 8 Byte, massimizzando così l'uso di questi 13 bit (non esiste quindi un frammento più piccolo di 8 Byte). Indicizza $2^{13} = 8192$ gruppi di 8 Byte, quindi 65536 Byte (che è la maximum segment size di un segmento di livello 4)
- **Parola di 32 bit per controllo**
  - **Time To Live (TTL)**, tempo massimo di vita di un pacchetto nella rete; se, scaduto questo tempo, il pacchetto non è stato consegnato, va eliminato. Di solito indicato come numero di *hop*
  - **Protocol Selector**, indica il protocollo di livello 4 usato dal mittente, cosicchè il destinatario possa passarlo allo stesso protocollo
  - **Header Checksum**, usato per efficienza operativa, poichè se la checksum è sbagliata evito subito di instradare il pacchetto (verifica integrità solo per la parte dell'intestazione, per questo il livello 4 ricalcolerà la CRC anche per il contenuto IP)
- **Options**, campo opzioni variabile, nel quale vengono specificate opzioni su: sicurezza, source routing, ...

L'indirizzamento è a 32 bit, quindi si hanno 2^{32} indirizzi possibili (il numero non è più sufficiente, problema risolto con IPv6). Ogni macchina connessa a internet è indirizzabile da un indirizzo IP univoco (non esistono due macchine con stesso IP connesse contemporaneamente).

Frammentazione e ricostruzione pacchetto IP

Supponiamo di dover trasferire un pacchetto di 7000 Byte da una LAN token ring attraverso internet fino ad un host di una LAN ethernet. Assumiamo che la quantità massima di trasmissione (MTU) sulla rete token ring sia di 4000 Byte, mentre quella ethernet di 1500 Byte. IP aggiunge altri 20 Byte di intestazione quindi:

- abbiamo a disposizione $4000 - 20 = 3980\ Byte$ per l'invio
- dobbiamo creare frammenti con payload multipli di 8 Byte per l'invio
- vengono inviati due frammenti (3976 e 3024 Byte)
- passano attraverso internet
- arrivano alla rete ethernet dove devono essere frammentati di più perchè la quantità massima di trasmissione qui è di 1500 Byte ($1500 - 20 = 1480\ Byte$)
- ogni pacchetto che arriva viene frammentato in altri 3 pacchetti (1480 - 1480 - 1016 Byte il primo, 1480 - 1480 - 64 Byte il secondo)

Classi di indirizzamento

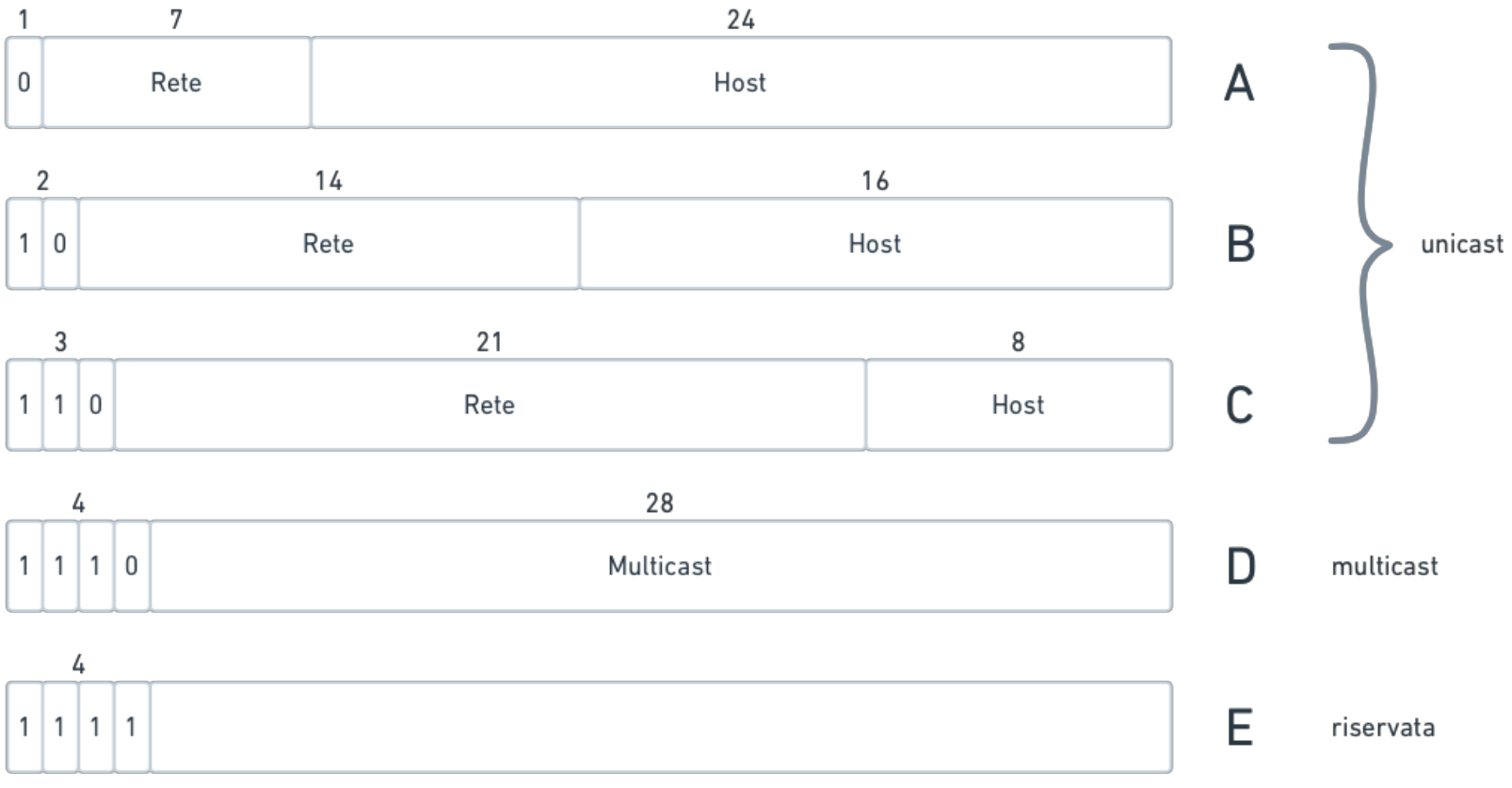
Gli indirizzi IP erano, inizialmente, divisi in 4 classi:

Classe	Bit rete	Bit host	Indirizzi	Note
A	8	24	1.0.0.0 - 127.255.255.255	non tutti e 8 i bit per la rete sono utilizzati, quello più significativo è messo a `0`
B	16	16	128.0.0.0 - 191.255.255.255	non tutti e 16 i bit per la rete sono utilizzati, i 2 più significativi sono messi a `10`
C	24	8	192.0.0.0 - 223.255.255.255	non tutti e 24 i bit per la rete sono utilizzati, i 3 più significativi sono messi a `110`
D	-	-	-	è utilizzata per il multicast
E	-	-	-	è utilizzata per gli esperimenti

I primi bit sono i selettori della classe:

- se il primo bit è 0 `=>` classe A
- se il primo bit è a 1
  - se il secondo bit è 0 `=>` classe B
  - se il secondo bit è a 1

- se il terzo bit è a 0 `=>` classe C
- se il terzo bit è a 1 `=>` le altre



Subnetting

I router non possono avere una tabella di routing contenente tutti gli host, date le classi di indirizzamento (sono troppi).

Quindi, ogni router di una sottorete conosce gli indirizzi degli host a lui collegati e quelli dei router a livello superiore.

I router dei livelli superiori non tengono traccia di tutti gli indirizzi degli host delle sottoreti collegate, bensì si suddivide il net id, creando un subnet id (oltre il net id) che permette di verificare se un indirizzo fa parte di una delle sottoreti senza conoscerne gli host. In pratica:

- il router controlla che il net id del destinatario sia uguale a quello delle sue sottoreti
  - se si, si controlla a quale sottorete inoltrare il pacchetto in base al subnet id
  - se no, il pacchetto non viene inoltrato nelle sottoreti collegate al router
- il router della subnet a cui arriva il pacchetto inoltrerà il pacchetto all'host corretto

Il subnet id è sempre ricavato da una parte della parte host dell'indirizzo. Come capire quanti bit sono stati assegnati al subnet id? Viene utilizzata una **subnet mask**.

Per capire a quale sottorete deve essere indirizzato il pacchetto, viene fatto un AND logico con la subnet mask, formata da tutti 1 nella porzione net id e subnet id e 0 altrove.

Con la terminologia `Indirizzo/Numero` (ad esempio 130.50.12.0/22) si indica che ad un indirizzo viene associata una subnet mask con un numero di `1` pari a `Numero` (iniziali, nell'esempio 22).

NAT (Network Address Translation)

Non si dà un indirizzo IP diverso agli host della sottorete, perchè non ci sarebbero abbastanza indirizzi per tutte le sottoreti del mondo.

Per questo si fa vedere esternamente il router della sottorete con un indirizzo IP singolo e ogni host avrà un indirizzo "privato" univoco solo all'interno della sottorete.

Quando un host vuole comunicare con l'esterno, il servizio NAT del router rimpiazza il suo indirizzo privato con quello pubblico della sottorete, e salva questa associazione in una tabella. Siccome più host possono comunicare con l'esterno, non basta associare all'host interno l'indirizzo pubblico della sottorete ma serve differenziarlo anche per numero di porta usata per la comunicazione. Siccome più host potrebbero usare la stessa porta, il NAT rimpiazza le porte duplici con delle porte diverse, scelte dallo stesso NAT.

Nella tabella di indirizzamento vengono quindi salvati:

- ip interno
- porta interna
- ip esterno
- porta esterna
- ip destinazione
- porta associata dal NAT (potenzialmente diversa da quella usata internamente per evitare duplcicità)

Se un host interno (ad esempio un server) volesse essere visto esternamente con un proprio e univoco indirizzo IP, alla sottorete dovrebbero essere associati due indirizzi pubblici, e il NAT provvederebbe a cambiare solo l'indirizzo privato di quell'host con quello pubblico a lui riservato (solitamente l'host è collegato in DMZ con il NAT).

CIDR (Classless Inter Domain Routing)

CIDR propone un altro metodo di indirizzamento. Ad un'azienda non si assegna una porzione di classe, bensì una quantità di indirizzi IP che servono, considerando anche eventuali espansioni aziendali.

Non c'è più il concetto di classi e si usa il subnetting.

ARP

Siccome per comunicare a livello 2 mi servirà l'indirizzo MAC dell'host destinatario, serve un protocollo che dato un indirizzo IP mi dia il corrispondente indirizzo MAC.

Sono previsti due tipi di messaggi:

- *ARP Request*
- *ARP Reply*

Quando un host vuole comunicare con un altro, invi a tutti nella rete un'ARP Request contenente:

- il proprio MAC
- il proprio indirizzo IP
- l'indirizzo IP del destinatario

Quando l'host detinatario riceverà l'ARP Request risponderà con un'ARP Reply, destinato al MAC dell'host sorgente, contenente il proprio MAC.

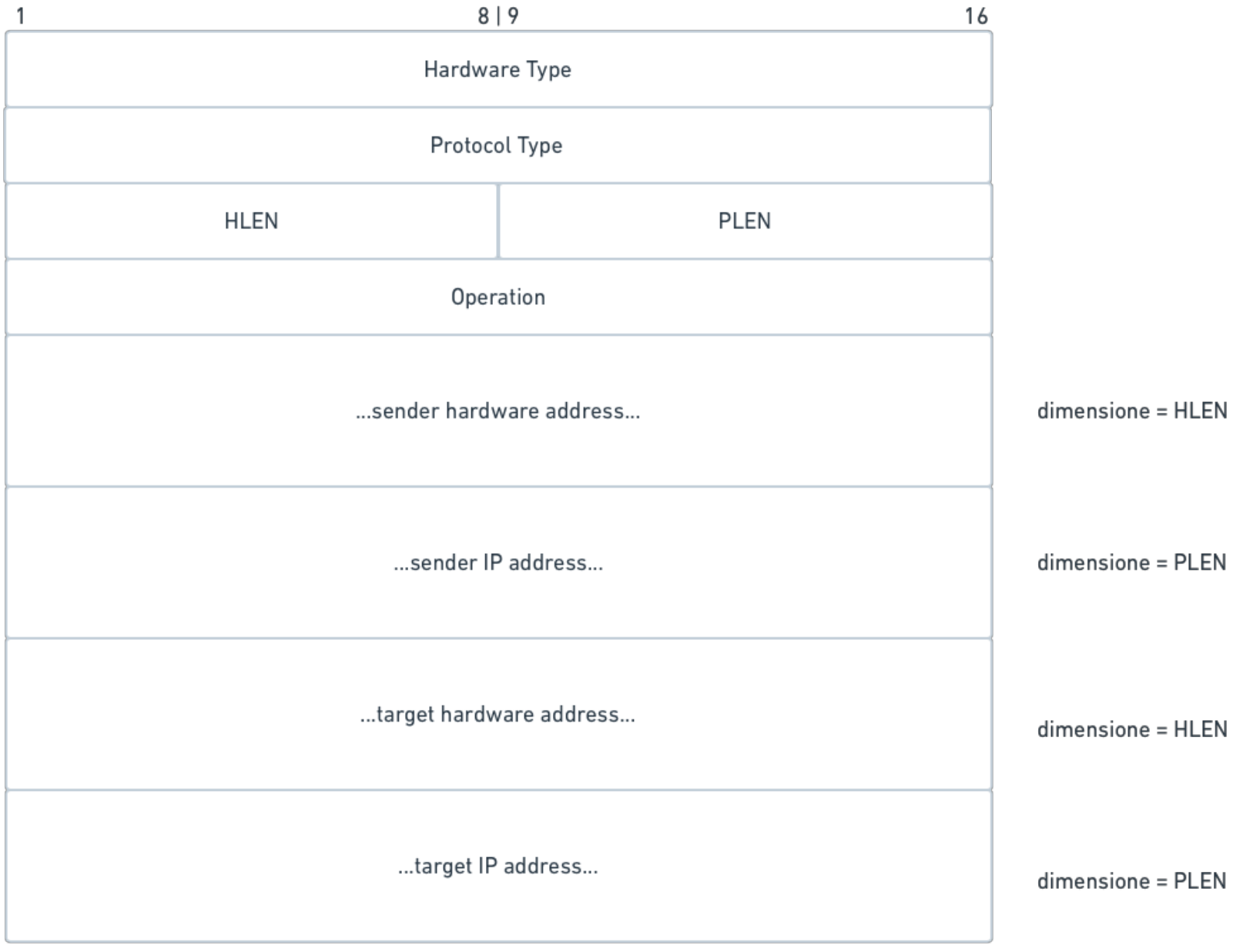
L'associazione tra indirizzo IP e MAC viene poi salvata nella tabella ARP (*ARP Cache*) degli host così da non dover mandare di continuo pacchetti ARP.

Nel caso A voglia parlare con un nodo presente in un'altra rete, il gateway, dotato di proxy ARP, si sostituisce a B (A crederà di parlare con B).

RARP

È una variante di ARP che viene usata quando un host conosce il suo MAC ma non il suo indirizzo IP (caso di nodi diskless, senza drive fisico da cui fare il boot e che quindi chiedono info ad un server). Viene predisposto un server RARP che possiede una tabella contenente tutte le coppie MAC-IP degli host.

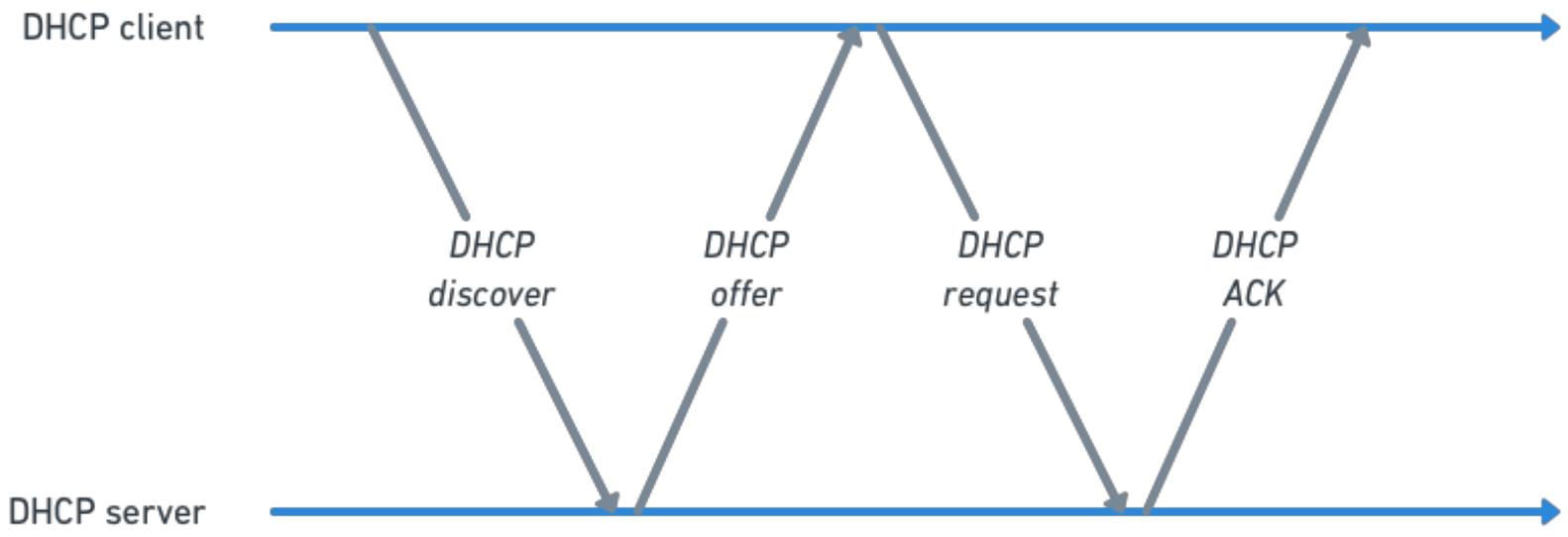
Formato pacchetto ARP



- **hardware type**, specifica il tipo di interfaccia hardware su cui l'host cerca una risposta (in ethernet si setta a `1`)
- **protocol type**, indica il tipo di dinriizzo ad alto livello che il mittente ha fornito (per IP si setta a `0x0800`)
- **HLEN (Hardware Length)**, specifica la lunghezza dell'indirizzo hardware (MAC)
- **PLEN (Protocol Length)**, specifica la lunghezza dell'indirizzo del protocollo ad alto livello (IP)
- **Operation**, specifica se si tratta di una ARP Request (valore `1`), ARP Reply (valore `2`), RARP Request (valore `3`) o RARP Reply (valore `4`)

DHCP

Utilizzato per richiedere ad un server DHCP un indirizzo IP dinamico.



- **DHCP Discover**, il client invia a tutti (indirizzo broadcast 255.2555.255.255) una richiesta per ottenere un IP dinamico, e come IP sorgente mette 0.0.0.0 (che identifica la propria macchina). Questo messaggio contiene un *transaction ID* che ha il compito di identificare univocamente la richiesta (generato a partire dal clock e dal MAC del sorgente)
- **DHCP Offer**, è la risposta del server che contiene:
  - lo stesso transaction ID della discover del client
  - l'indirizzo IP proposto dal server per il client (scelto verificando che non sia già associato a nessuno, provando a inviare un messaggio a quell'indirizzo e verificando l'assenza di risposta)
  - la subnet mask
  - il tempo durante il quale il client potrà utilizzare questo indirizzo (*tempo di lease*)
- **DHCP Request**, il client accetta o meno l'indirizzo IP proposto dal server
- **DHCP ACK**, il server manda un ACK di conferma

Potrebbe capitare che il client manda la DHCP Request al server, che però ha già dato l'indirizzo IP offerto a qualche altro host, così manda un DHCP NACK. Qualora il client invece non ricevi risposta, egli rimanda un pò di volte la request e al limite rifà DHCP discover per altro server.

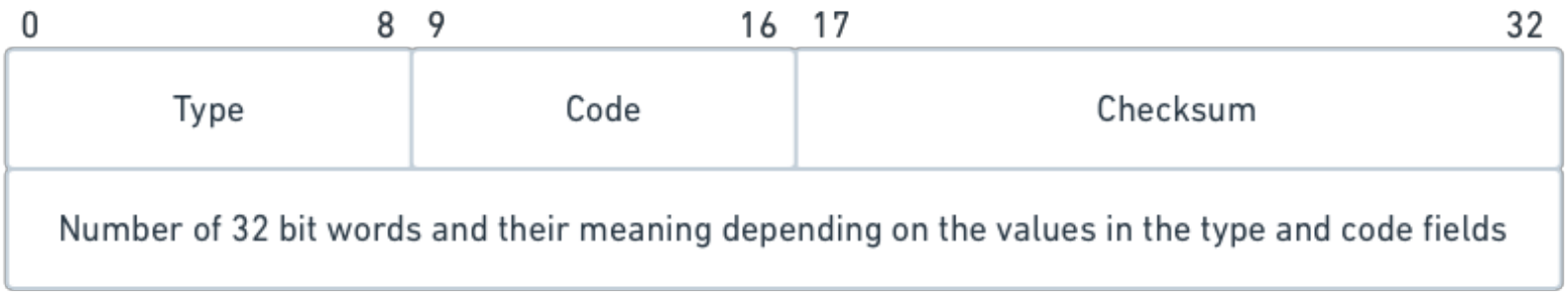
ICMP (Internet Control Message Protocol)

Funzionalità puramente di controllo, le principali sono:

- rilevare errori
- rilevare congestione nodi
- verificare raggiungibilità nodi
- notificare cambio percorso per raggiungere host
- fornire subnet mask della sottorete

- misurare prestazioni dei link

Formato pacchetto ICMP



- **Type**, indica il tipo di controllo che si vuole effettuare, che può essere:
  - *destination unreachable*, datagram scartato, causa specificata nel messaggio
  - *time exceeded*, datagram scartato, causa TTL esaurito
  - *parameters error*, datagram scartato, causa un parametro dell'intestazione è "irriconoscibile"
- **Code**, indica le informazioni aggiuntive opzionali, per esempio il motivo per il quale una stazione risulta irraggiungibile

Instradamento

Se ne occupano i router che hanno le funzionalità di:

- controllo (ICMP)
- forwarding (inoltro con tabella di instradamento)

Tabella di instradamento: associazione tra indirizzo IP di un host e la porta di uscit dal router per raggiungerlo (si tiene nota solo degli IP vicini, gli altri sono gestiti da altri router)

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 cosa di uscita

Ci sono due modi (algoritmi) di routing:

- distance vector
- link-state shortest path first

Distance Vector

Si tiene una tabella di instradamento che contiene il costo di ogni percorso per raggiungere tutti gli altri host. Inizialmente ogni router si costruisce la sua, valutando il

costo dei link con cui è connesso. Poi i router si scambiano le informazioni (vettori distanza) sulle proprie tabelle a periodicità definita da un time così da integrare le informazioni e avere tutti i percorsi minimi.

Criticità:

- altissimo overhead su reti ampie, dovuto allo scambio delle informazioni sulle tabelle (vettori)
- *bouncing effect*

Bouncing effect e Trigger Update

Il bouncing effect è il caso in cui un collegamento si guasta, il router più vicino se ne accorge subito e nella sua tabella mette ∞ al costo per i nodi aldilà di quel collegamento. Poi però un altro router gli invia la sua tabella, dove c'è ancora l'informazione per raggiungere quel nodo con il collegamento guasto e siccome il costo sarà minore di ∞ il primo router aggiorna la propria tabella erroneamente.

Una possibile soluzione è il **trigger update**, ovvero quando un router modifica la propria tabella a seguito di un imprevisto, invia a tutti l'aggiornamento senza aspettare il timer. Questa soluzione però non elimina del tutto il problema, perchè il pacchetto di trigger update potrebbe non arrivare a destinazione.

Un altro problema, generato dal bouncing effect è il *count-to-infinity*.

Count-to-infinity e Split Horizon

In un collegamento lineare tra A-B-C-D... se si guasta il collegamento tra A e B, B mette costo infinito ma C poi gli dice che può arrivarci passando tramite lui. Questo propagarsi dell'informazione sbagliata porta a stime di distanza (costo) sempre maggiori.

Una possibile soluzione a bouncing effect e count-to-infinity è lo **split horizon**, in cui ogni router propaga sempre ∞ come costo di raggiungimento di stazioni indirette che raggiungerebbe tramite la stazione a cui sta inviando l'informazione. Questa soluzione non elimina del tutto il problema, perchè i pacchetti possono comunque perdersi.

La soluzione per entrambi i problemi è usare il protocollo *RIP*.

RIP

Funzionamento:

- per ogni entry della tabella di instradamento del router esiste un time. Se per 6

tempi di update (circa 30 secondi l'uno) non ho risposta, allora viene messa la entry a infinity
- si usa trigger update (c'è sempre il problema count-to-infinity)
- il costo di un link si indica con il numero di hop nell'intervallo 0:15, con $16 = \infty$
- *update storm* (tempesta di update): può capitare che i timer scadano tutti insieme, quindi viene generato tantissimo traffico in rete, perciò ogni nodo genera il proprio update con un ritardo (0 - 5 secondi)

Link State

Invece di scambiare i vettori distanza, si scambiano i vettori di stato.

Tutti i nodi trasmettono i costi dei link a loro connessi (link state) non solo ai vicini ma in *flooding* a tutti gli altri. Un nodo diventa quindi indipendente dagli altri perchè tutti hanno le informazioni dell'intera rete. Ci sono un sacco di nodi in più nel traffico di rete, ma questo metodo seleziona il percorso più veloce.

Una volta determinata la tipologia della rete, ogni nodo calcola i cammini minimi verso ogni possibile destinazione con l'algoritmo di Dijkstra (Shortest Path First, SPF).

I pacchetti sono caratterizzati da:

- **numero di sequenza**, in una topologia magliata, lo stesso link state può giungere da nodi diversi. Si ha quindi bisogno di un meccanismo per droppare i link state già ricevuti
- **fattore di aging**, usato per eliminare pacchetti che continuano a girare in rete (e accorgersi di eventuali loop). Può capitare che un pacchetto permanga troppo nella rete (TTL)

e sono più piccoli dei vettori distanza, perchè contengono le informazioni dei soli nodi vicini, non di tutta la tabella di un router.

Siccome vengono generati $O(n^2)$ messaggi nella rete per costruire le tabelle, si designa un router predefinito (*designated router*), a cui vengono inviati tutti gli stati e che poi calcolerà i camminimi minimi e li manderà in flooding a tutti i suoi nodi ogni periodo di tempo (di solito solo se cambiano).

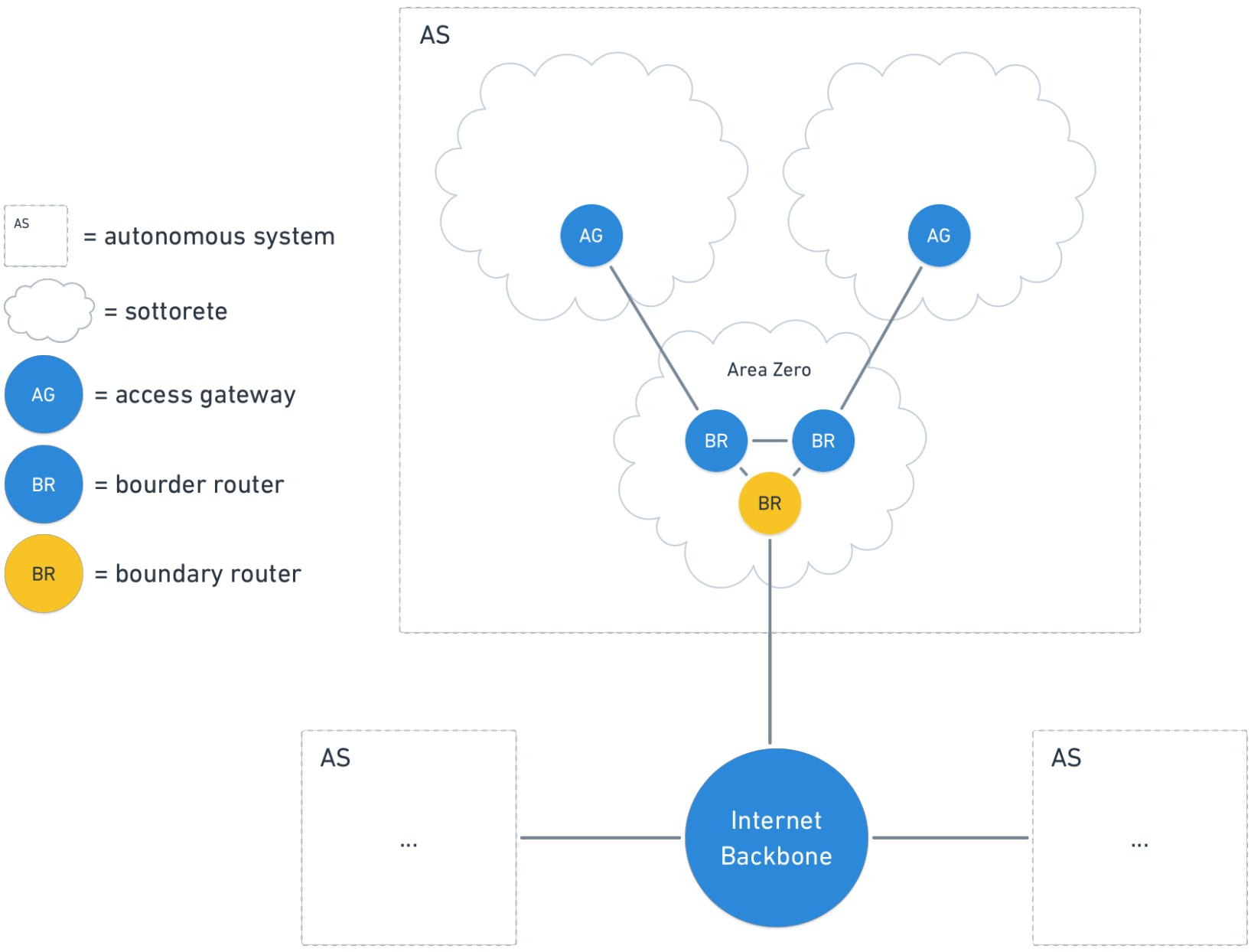
OSPF (Open Shortest Path First)

È il protocollo di internet definito nell'RFC 2328 che gestisce la gran parte dei problemi di routing con tecnica Link State. Opera:

- negli ***autonomous system*** (sistemi autonomi), composti da diverse sottoreti

collegate a un'area zero.

- tra ***aree backbone*** (aree zero), aree più alte a livello gerarchico, che possono essere composte sia da router che da sottoreti. Nell'area zero ci sono speciali routers, chiamati **AS boundary routers**, che fungono da gateway tra la sottorete degli host e la internet backbone (dorsale internet), mentre i router che collegano le sottoreti all'area zero si chiamano **bourder routers**. Nell'aree zero, per evitare problemi relativi al flooding, vi è un designated router che concentra su di se gli aggiornamenti per le tabelle di routing e distribuisce poi le informazioni complete ai nodi vicini (vedi link state).



I tipi di messaggi che vengono scambiati durante OSPF sono:

- **Hello**, usato da un router per scoprire nuove reti/router adiacenti, scoprendo così il costo di un link
- **Link State Update**, inviato dal designated router ad intervalli periodici (o quando necessario, tipo cambiamento costi link, etc) che porta con se le informazioni riguardanti le tabelle di routing, da far propagare agli altri router
- **Link State ACK**, ogni router valida un Link State Update con questo messaggio
- **Database Description**, viene usato per informare il ricevente del messaggio se è

disponibile o meno un aggiornamento
- **Link State Request**, viene usato per richiedere un Link State Update da ogni router adiacente

OSPF offre inoltre:

- *Multiple Paths to Dest*, mantiene cioè in memoria tutti i possibili percorsi con uguale costo
- *Source Routing Possible*, uno specifico percorso può essere configurato a mano (tipico caso per raggiungere il designated router). Nell'intestazione del pacchetto verranno indicati tutti gli hop da percorrere (*waypoint*), in modo statico. In IPv4 viene indicato nel campo Options
- *Multiple Metrics*, i link possono essere considerati medianete diversi parametri, come bit-rate, distanza fisica, etc.

La comunicazione tra Autonomous System avviene attraverso il protocollo Border Gateway Protocol.

BGP (Border Gateway Protocol)

Viene usato dagli AS Boundary Routers.
Usa un approccio come Distance Vector, ma modificato. Chiamato **Path Vector**, indica esplicitamente tutto il percorso da fare per raggiungere un AS Boundary Router. Al posto di propagare le distanze, si propagano cammini.

Vengono usati 4 tipi di messaggio:

- **Open**, usato per aprire una relazione con un AS Boundary Router adiacente
- **Update**, usato sia per trasmettere informazioni sull'instradamento per un singolo percorso che per comunicare una lista di percorsi che devono essere rimossi
- **Keep Alive**, usato per validare una Open e per mantenerla attiva, confermandola periodicamente
- **Notification**, usato per informare che si è verificata una condizione di errore

Border Gateway Tunneling: quando un pacchetto IP viaggia attraverso sottoreti, non è detto che tutte usino il protocollo IP. I router di frontiera delle reti che non usano IP provvedono quindi a incapsulare il pacchetto IP in un altro pacchetto conforme al protocollo usato nella sottorete da attraversare, che verrà gestito nella sottorete non-IP e rimosso dall'ultimo router quando si ritorna sul protocollo IP.

MPLS (Multi Protocol Label Switching)

Al giorno d'oggi l'OSPF è stato sostituito dall'MPLS. Questo è un protocollo di livello 2 che non instrada ma inoltra (punto-punto). Aggiunge ai pacchetti di livello 3 una sorta di etichetta (*label*) che identifica il tipo di priorità di velocità che il pacchetto deve avere, anche per venire incontro alle esigenze di traffico real-time, non real-time, etc.

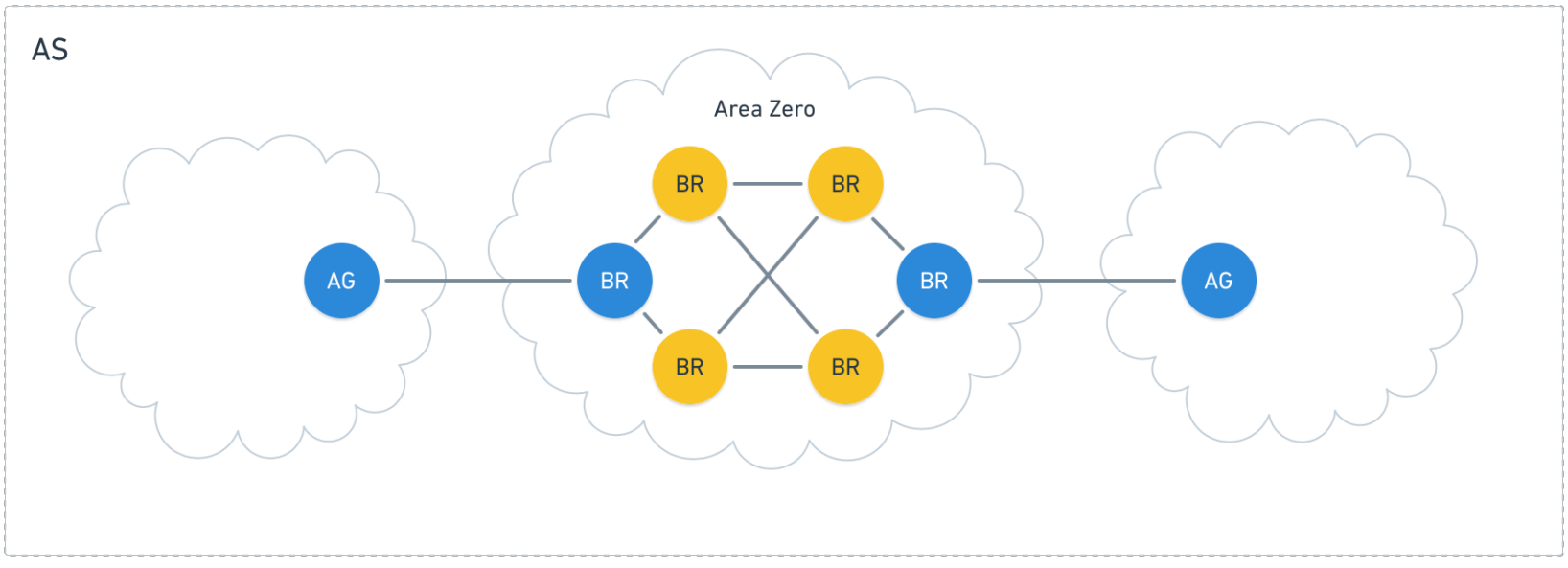
In base all'etichetta, si attuerà una diversa politica di scheduling nelle code di uscita.

Funzionamento etichettamento

Alla ricezione di un pacchetto da una delle interfacce di ingresso, questo conterrà un campo ToS (Type of Service), che viene gestito da un **packet classifier**, che dovrà:

- controllare nell'intestazione del pacchetto IP il ToS
- assegnare una etichetta al pacchetto
- aggiungere una nuova intestazione
- inserire il pacchetto nella coda di uscita appropriata

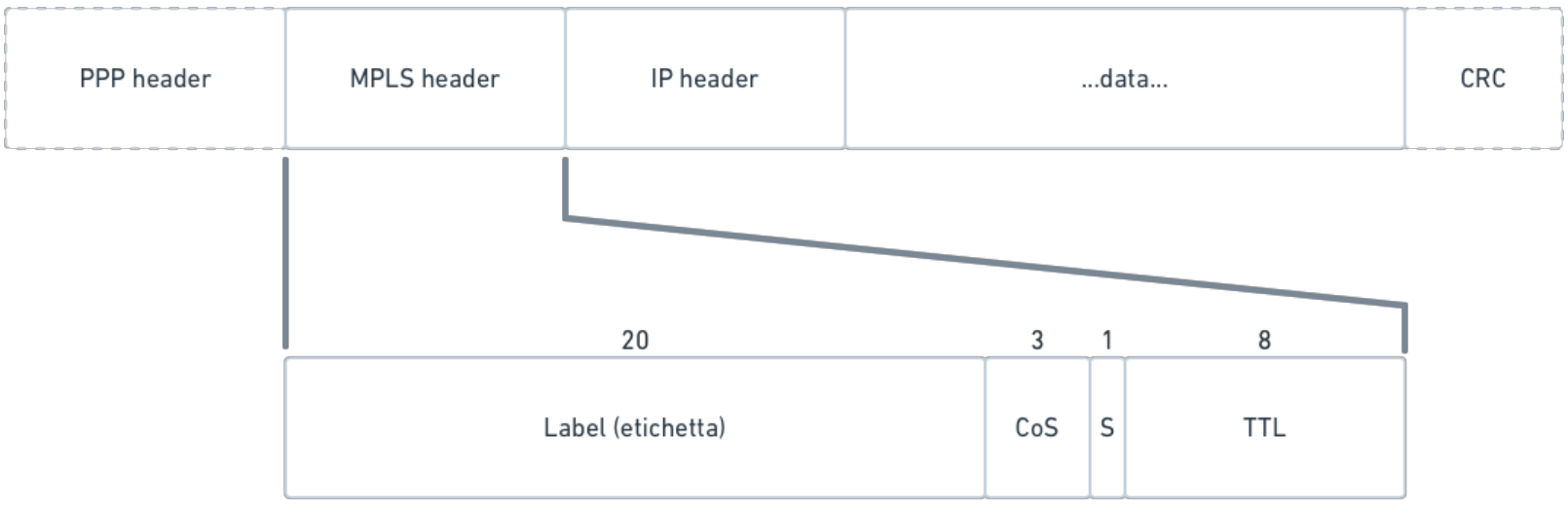
In genere il traffico viene diviso in base al tipo, mentre MPLS tenta di aggregare il traffico di un flusso ed effettuare bilanciamenti tra i flussi. Questo viene applicato nell'area di backbone a cui ogni AS è collegato:



- **access gateway (AG)**, effettua lo scheduling in base alla *classe* dei pacchetti, perchè deve distinguere tra un largo numero di applicazioni e utenti. Separa poi ogni flusso di dati (**shaping**), necessario per permettere la corretta interpretazione di ogni flusso da parte dei bourder router, che separano i pacchetti in code separate (es: real-time, non real-time, etc) (**grooming**)
- **bourder router (BR, blu)**, anche noto come LER (*Label Edge Router*), provvede ad aggiungere/rimuovere l'intestazione MPLS dalla testa dei pacchetti IP, poichè MPLS ha senso solo ad ogni hop. Questo perchè quando vengono usati sempre i percorsi minimi, alcuni bourder router potrebbero sovraccaricarsi formando un **hotspot**. Per evitarlo, viene cambiata intestazione MPLS e instradato il pacchetto

verso un altro percorso. Viene effettuato quindi un **load balancing** attraverso **label switching**. Per gestire il traffico viene determinato un **LSP (Label Switched Path)** attraverso l'area zero. Quindi alla ricezione di un pacchetto IP, l'indirizzo destinazione viene usato per determinare la porta d'uscita, mentre il valore *DiffServ (Differentiated Services)* contenuto in DS (primi 6 bit del campo ToS) viene analizzato dal modulo LSP che crea quindi un'intestazione MPLS per l'hop successivo e sceglie quindi le regole di scheduling del pacchetto.

Formato intestazione MPLS



- **Label**, identifica la politica per questo hop, e contiene l'identificatore per la porta d'uscita
- **CoS (Class of Service)**, stabiliscce le regole di scheduling

IPv6

Nato per sopperire alla mancanza di indirizzi Iv4, aggiunge nuove funzionalità. Le principali sono:

- spazio degli indirizzi aumentato da 32 bit a 128 bit (da 2^{32} a 2^{128}) divisi in 8 word da 2 byte
- indirizzi gerarchici, per ridurre le dimensioni delle tabelle di instradamento associate ai router della internet backbone
- un header semplificato, per aumentare la velocità di processing dei router (la fase di instradamento rimane invariata)
- una funzione di autoconfigurazione, per consentire ad un host di ottenere un indirizzo IP tramite la rete senza l'intervento umano
- compatibilità con i protocolli IPv4

Formato pacchetto IPv6

- **8 Byte di parametri**
  - **version (4 bit)**, indica la versione del datagramma IP: per IPv6 è settato a `6`
  - **traffic class (8 bit)**, si traduce come "classe di traffico", successore di ToS, classifica il pacchetto favorendo regole di routing e lo smistamento assegnando una classe di priorità, rispetto ad altri pacchetti provenienti dalla stessa sorgente
  - **flow label (20 bit)**, usata dal mittente per etichettare una sequenza di pacchetti come se fossero nello stesso flusso. Aiuta nella gestione del QoS (Quality of Service), nel controllo di flusso, anche se ancora in fase sperimentale. Per best-effort è settato a `0`.
  - **payload length (16 bit)**, è la dimensione del payload, ovvero il numero di byte di tutto ciò che viene dopo l'header. Eventuali estensioni dell'header sono considerate payload. Max a 64 KB, minimo a 536 B (perchè 536 B + 40 B di header = 576 B, minimo per far si che il pacchetto non venga frammentato)
  - **next header (8 bit)**, indica quale tipo di intestazione segue l'header di base IPv6. Innovazione dell'IPv6 è che consente di usare più spazio per l'header. Se si vuole estendere l'header, next header punterà ad un HE (Header Extension), che conterrà anch'esso un next header, e così via. Se non c'è estensione, punterà all'header del livello successivo (TCP)
  - **hop limit (8 bit)**, è il limite di salti consentito, praticamente è il TTL per IPv6, con default a 256
- **32 Byte per indirizzi**
  - **source address**, 16 Byte per indirizzo sorgente
  - **destination address**, 16 Byte per indirizzo destinazione

Extension header

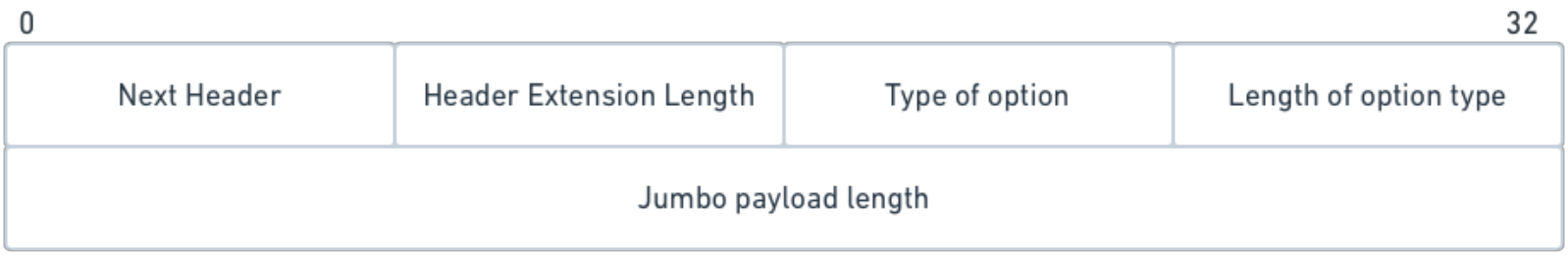
In IPv6 solo la sorgente può frammentare. Quindi occorre conoscere la MTU (Maximum Transfer Unit) di ogni sottorete che il pacchetto dovrà attraversare (si usa ICMP per trovare taglia minima del pacchetto).

Gli extension header contengono le opzioni dell'IPv6:

- **opzione hop by hop**
- **Routing**
- **frammentazione**
- **autenticazione**
- **sicurezza**
- **opzioni per la destinazione**

Hop by hop

Se c'è `00` nei bit meno significativi del Next Header, allora si usa hop by hop. Serve per gestire pacchetti più grandi di 64 KB. Header extension sarà così:



- *next header*, rappresenta il puntatore all'header successivo
- *header extension length*, esclude i primi 8 Byte, quindi in questo caso è 0
- *type of option*, `194` per hop by hop
- *length of option type*, lunghezza del jumbo payload length
- *jumbo payload length*, lunghezza del payload

Routing

Se c'è `43` nei bit meno significativi del Next Header, allora si usa routing. Serve per implementare il source routing. Header extension sarà così:

- *header extension length*, al massimo 24 indirizzi da cui si vuole passare
- *routing type*, non è ancora ben definito...
- *segments left*, viene inizializzato al numero di step intermedi che partecipano. Viene decrementato ad ogni hop prefissato
- *reserved*
- *strict/loose bit map*, tanti bit quanti i possibili router fissati (max 24). Se c'è 1 voglio un collegamento diretto verso quel router (routing esatto), se c'è 0 non mi ineteressa come arrivo a quel router ma è sufficiente che ci si passi (routing approssimato).

Frammentazione

Come IPv4 abbiamo l'identificazione del pacchetto, il fragment offset e il More Fragment Bit (M). Ma l'unico che può frammentare è la sorgente, non come in IPv4 dove frammenta ogni gateway per la sua sottorete. Quindi la sorgente userà source routing per sapere da dove passerà il pacchetto, e ICMP per capire dove intradarlo.

Formato indirizzo IPv6

Un indirizzo è rappresentato comunemente da 8 blocchi di 16 bit divisi a ":", che per comodità vengono scritti in esadecimale. Blocchi contigui di 0 possono essere omessi, sostituendoli con "::".

> Es:
> 16bit : 16bit : 16bit : 16bit : 16bit : 16bit : 16bit : 16bit
> FDEC : BA98 : 7654 : 3210 : 0000 : 0000 : 0000 : 0089 (notazione completa)
> FDEC : BA98 : 7654 : 3210 :: 0089 (notazione contratta)

Per facilitare il processo di transizione da IPv4 a IPv6, sono state introdotte 3 tipologie di indirizzi IPv6, pensate per contenere al loro interno indirizzi IPv4:

- *IPv4-compatible IPv6*, impiegati in alcune situazioni di protocol tunneling, sono formati da 96 bit posti a 0 seguiti dall'indirizzo IPv4 (es. ::193.206.71.151)
- *IPv4-mapped IPv6*, sono invece formati da 80 bit a 0, seguiti da una porzione di 16 bit a 1 e infine l'indirizzo IPv4 (es: ::FFFF:193.206.71.151)
- *IPv4-translated IPv6*, sono invece formati da 64 bit a 0, seguiti da 16 bit a 1 e altri 16 a 0 e infine l'indirizzo IPv4 (es: ::FFFF:0:193.206.71.151)

Le ultime due tipologie vengono usate in quelle situazioni in cui è necessario far comunicare nodi IPv4-only con nodi IPv6-only.

IPv6 attua una organizzazioen gerarchica basata sulla grandezza del servizio:

- Tier 1, organizzazioni a livello continentale. Con Registry e TLA si identifica una singola organizzazione di livello 1
- Tier 2, organizzazioni a livello nazionale. Campo NLA
- Tier 3, organizzazione a livello locale. Campo SLA

Questa organizzazione ha un grande impatto sulla dimensione delle tabelle di routing, visto che si trasforma in una divisione dei 128 bit tramite **PF (Prefix Format)** e quindi tutti i pacchetti con stesso prefisso verranno instradati nella stessa direzione. Si effettua quindi un'aggregazione degli indirizzi (**address aggregation**).

I primi bit dell'indirizzo si chiamano prefisso. Un prefisso `010` indica una struttura gerarchica così composta:

- `010` indica la comunicazione punto-punto

- **Registry** indica il continente
- **TLA** indica la nazione
- **NLA** indica una zona intra-nazionale
- **SLA** indica una zona locale
- **Interface ID** è così composta:
    - 16 bit per la sottorete (net id, sono usati per il subnetting)
    - 48 bit corrispondente all'indirizzo MAC dell'host (host id)

Difficile interoperabilità con IPv4

IPv6 non può affermarsi di colpo in bianco al posto di IPv4. Lo andrà a sostituire ma al momento nelle reti coesistono host con IPv4 e host con IPv6.

Come permettere questa interoperabilità?

Piano piano si stanno costruendo router e switch che lavorano con entrambi, ma è complicato sostituire tutto l'hardware di internet. Perciò si lavora via software (ad esempio i sistemi operativi ora supportano entrambi). Ci sono 3 soluzioni principali:

- dual-stack
- NAT-PT
- tunneling

Dual-stack

Prevede l'utilizzo di entrambi i protocolli a livello 3. Questo permette di interpretare entrambe le versioni del protocollo e quindi smistare a livelli superiori il contenuto del pacchetto senza che questi sappiano da quale protocollo IP derivi. L'identificazione IPv4 e IPv6 viene fatta leggendo l'*EtherType* associato al pacchetto, aiutando a smistare il pacchetto senza dover accedere al campo *version* del pacchetto di livello 3.

Svantaggi:

- aumenta complessità della rete
- non risolve problema diminuzione degli indirizzi IPv4, poichè serve comunque un'interfaccia dotata di due indirizzi, uno IPv4 e l'altro IPv6. Inoltre entrambi gli indirizzi devono essere "annunciati" nella rete, complicando il routing

Un miglioramento del dual stack è dato dall'ALG (Application Level Gateway).

È una soluzione che permette ad una rete totalemente IPv6 di dialogare con postazioni IPv4 al di fuori della rete. Solo il gateway è dotato di dual stack. Ad oggi però i gateway ALG operano solamente su alcuni dei servizi di rete (ad esempio HTTP), e se ne fossero

necessari di più, ne servirebbero altri.

NAT-PT (Network Address Translator - Protocol Translator)

Opera come il NAT vero e proprio ma converte l'indiriizzo IPv6 in IPv4 e viceversa, secondo alcune tecniche. Introduce molti limiti (infatti alcuni servizi non funzionano, a meno che non si introducano gateway ALG appositi). Viene usata come gateway tra una rete IPv6 e l'esterno, se all'esterno ci sono reti IPv4, in attesa che diventino IPv6.

- IPv6 vuole comunicare con IPv4: l'indirizzo IPv4 viene convertito con *IPv4-mapped to IPv6*
- IPv4 vuole comunicare con IPv6: all'host IPv6 viene assegnato un indirizzo "esterno" IPv4 da un pool di indirizzi dal gateway.

Tunneling

Prevede il principio del tunneling, per cui si stabilisce un collegamento punto-punto tra due host e i pacchetti IPv6 vengono incapsulati dall'host sorgente in pacchetti IPv4, inviati nel tunnel e, una volta giunti a destinazione, l'host destinatario li decapsula e li tratta come se fossero comunissimi pacchetti IP.

È difficile applicarlo alle reti globali, per cui il suo utilizzo è limitato a reti locali più o meno grosse.

Livello 4 "Trasporto"

Il come inviare i messaggi è risolto nei livelli 1-3. Ora si tratta di come mettere in comunicazione applicazioni da un host A a un host B (collegamento *end-to-end*). A livello 4 si spediscono e si ricevono **segmenti**.

Naming e porta

Il problema del naming consiste nel riuscire a determinare univocamente un'entità, consentendo così la comunicazione. Tra livello 2 e livello 3 ci sono ARP/RARP.

Una volta che il pacchettp arriva al livello 4 si ha necessità di:

1. associare un indirizzo IP ad un nome simbolico di un servizio (indirizzo di un sito web, di posta elettronica, etc)
2. legare una singola applicazione (e quindi uno specifico processo) ad un indirizzo IP

Per risolvere il punto 1 si usa il protocollo DNS (si vedrà più avanti). Per risolvere il punto 2, si introduce un nuovo tipo di astrazione: la **porta**. Ad ogni applicazione che

comunica in rete viene associata una porta, cioè un valore numerico.

La coppia <IP,porta> (chiamata **socket**) determina in modo univoco un processo in internet. Quindi in ogni segmento (sia UDP che TCP) ci saranno la porta sorgente e la porta destinazione.

In totale si hanno 2^16 possibili porte:

- dalla 0 alla 1023, chiamate ***well known ports***, sono associate a servizi standard (mail, web, etc.)
- dalla 1024 alla 49151 sono associate a "programmi famosi" (aziende private)
- le porte successive sono lasciate libere

Architettura client/server

Tutti i servizi internet seguono il paradigma client/server:

- client: sempre attivi, effettuano richieste
- server: sempre passivi, rispondono alle richieste

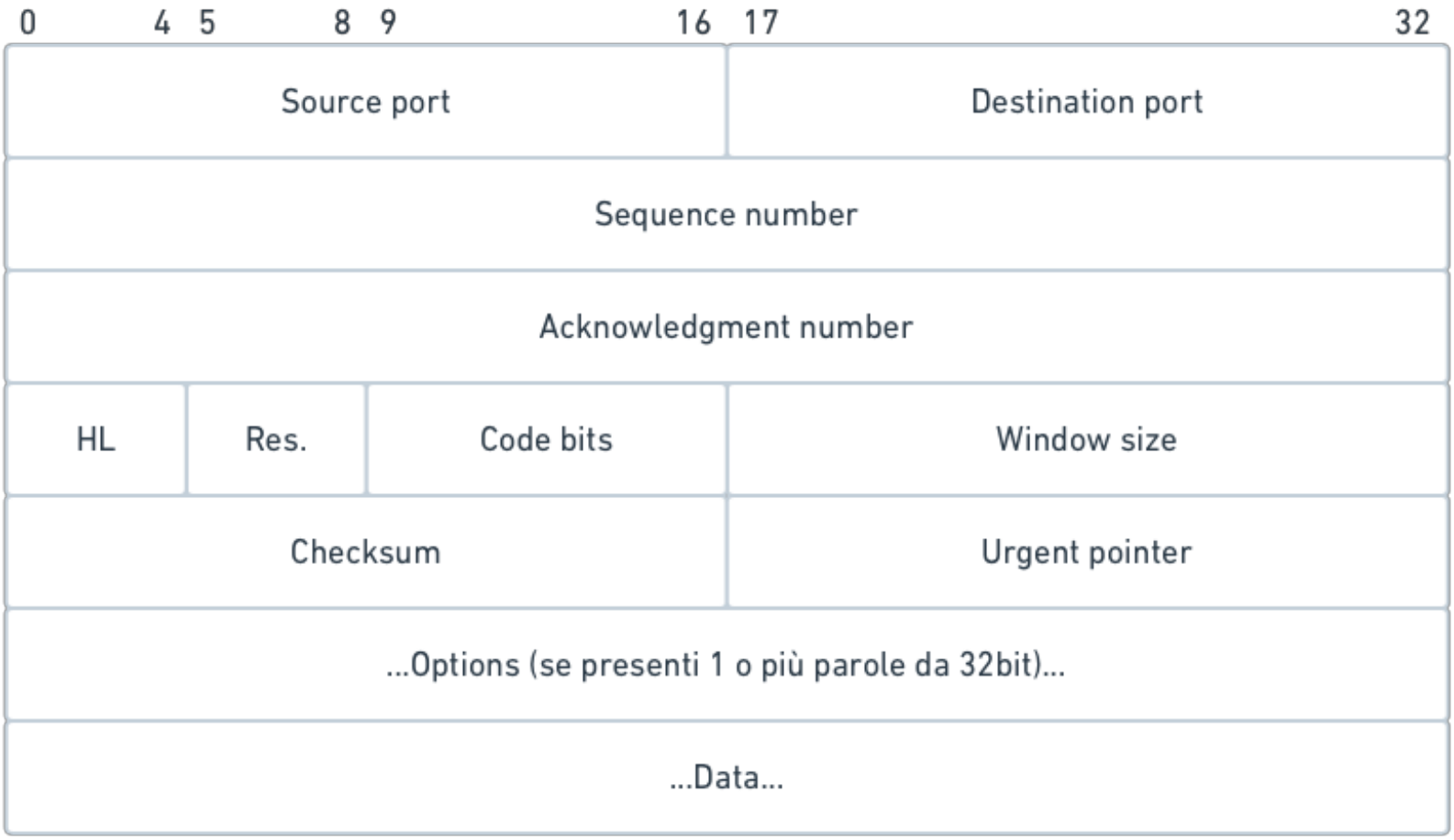
TCP (Transmission Control Protocol)

Fornisce un servizio che garantisce affidabilità alla comunicazione "best-effort" di IP. È bidirezionale, cioè apre due connessioni, una client-server e l'altra server-client.

Svolge 3 diverse operazioni:

1. configura una connessione logica tra due socket precedentemente creati (***three way handshake***)
2. trasferisce in modo affidabile i blocchi di dati su questa connessione
3. chiude la connessione logica

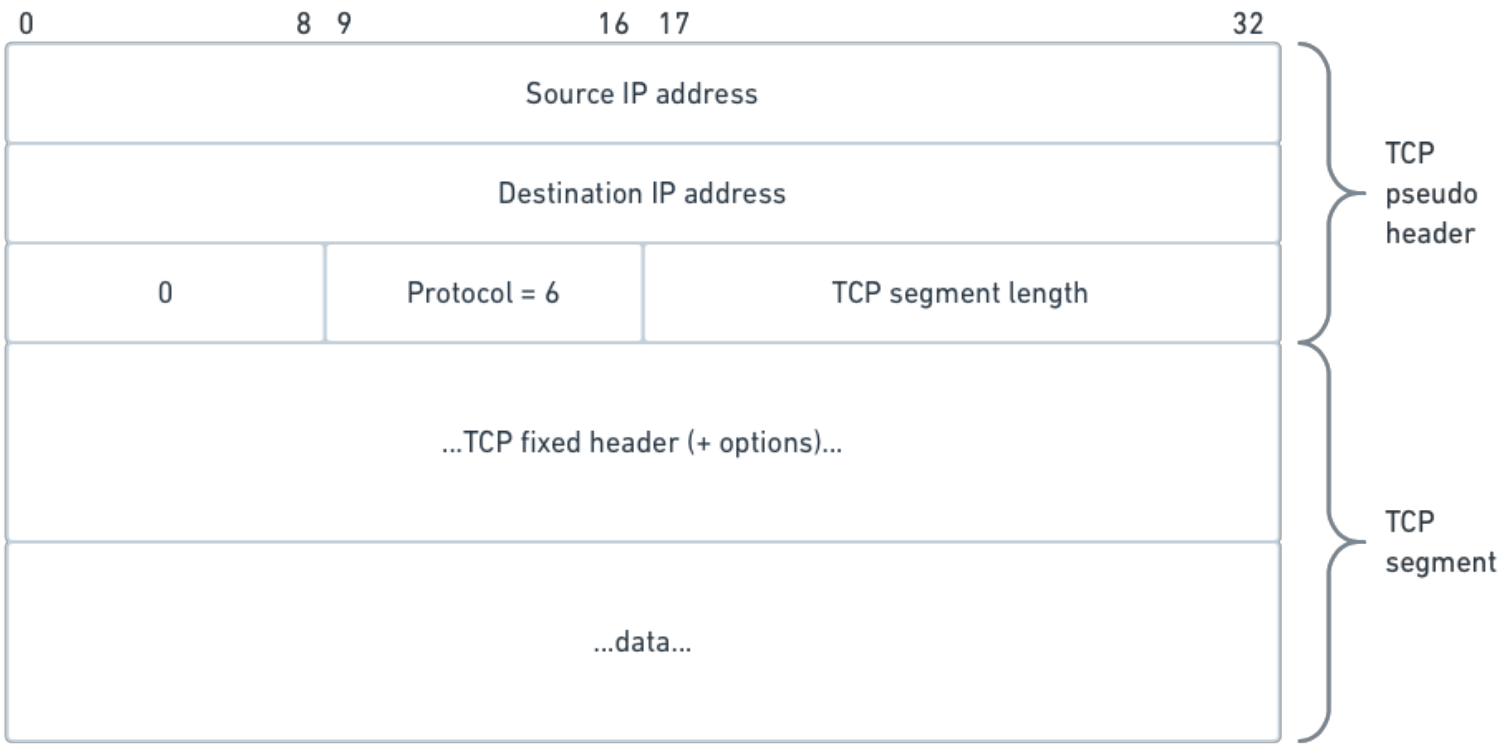
È un protocollo a finestra. Un header TCP è formato da minimo 20 Byte. Un segmento è così composto:



- **source port (16 bit)**, identifica il numero di porta sull'host mittente associato alla connessione TCP
- **destination port (16 bit)**, identifica il numero di porta sull'host destinatario associato alla connessione TCP
- **sequence number (32 bit)**, numero del segmento del dato da trasferire. Numera i Byte, perchè TCP considera payload come stringhe di Byte. È puntatore al primo Byte nel campo dati
- **acknowledgment number (32 bit)**, ha significato solo se il campo ACK è impostato a 1. È usato solo dalla destinazione, che per convenzione indica il numero del segmento che si aspetta di ricevere (quindi successivo all'ultimo convalidato/arrivato).
- **HL (Header Length o Data Offset)**, numero di parole da 32 bit presenti nel campo Options (header ha lunghezza variabile)
- **Res. (Reserved) (4 bit)**, bit non utilizzati e predisposti per sviluppi futuri del protocollo
- **code bits (6 bit)**, maschera di 6 bit che identifica il tipo di segmento. Più bit possono essere settati a 1, aggiungendo più informazioni in un singolo segmento. I possibili valori sono:
  - *SYN*: stabilisce una connessione
  - *ACK*: conferma della validità del campo
  - *FIN*: chiude la connessione
  - *PHS*: invio dei dati alla ricezione di questo segmento
  - *URG*: urgente

    - *RST*: rifiuta una richiesta di connessione o ripristina i valori iniziali del numero di sequenza
- **window size (16 bit)**, anche "Advertise Window", è la larghezza della finestra per spedire i frammenti. La dimensione è deisa a priori, prima di iniziare la comunicazione, e scelta in base alla dimensione massina di trasferimento del canale in uso e del buffer lato destinazione. Si usa per evitare la frammentazione
- **checksum**, contrariamente al checksum del pacchetto IP (che riguardava solo l'header), questo riguarda tutto il segmetno, cioè header + payload. Inoltre alcuni campi del pacchetto IP sono inclusi nella generazione di questo checksum e formano il cosiddetto ***TCP pseudo header***
- **urgent pointer (16 bit)**, denota il numero di Byte nel campo dati che deve essere subito consegnato all'applicazione che sta comunicando. Ha significato solo se il code bit "URG" è settato a 1 e indica lo scostamento in Byte a partire dal numero di sequenza dei dati urgenti del flusso
- **options**, usato per decisedere, ad esempio, la dimensione massima di un segmento (MSS - Maximum Segment Size). Se non specificata viene scelta di default a 536 Byte (poichè sommando header TCP e IP ottengo i famosi 576 Byte)
- **payload**, rappresenta i dati da trasmettere

TCP Pseudo header



Questi sono i campi che vengono presi dall'header IP e dal segmento per creare il checksum. Il TCP pseudo header viene creato 1 volta e memorizzato localmente con i parametri che arrivano nel primo segmento, per la composizione dei segmenti successivi e per evitare attacchi MITM (Man In The Middle). Lo pseudo header non viene spedito, ma viene costruito localmente da ogni macchina.

Fasi comunicazione TCP

Apertura della connessione

1. il client manda un segmento con bit SYN attivo e un numero di sequenza. Ogni entità TCP deve scegliere un numero di sequenza iniziale (*ISN, Initial Sequence Number*). Essi sono entrambi diversi da 0 e cambiano in ogni sessione, garantendo così che un segmento relativo a una specifica connessione non interferisca con un'altra. Il numero di sequenza viene scelto in base ai bit meno significativi del clock, e sono tutti diversi per un massimo di 2^{32} numeri di sequenza.
2. il server risponde con SYN+ACK, cioè accetta il SYN del mittente. Il server si mette poi in listen per segmenti del client. Se il server non è in listen mentre il client invia segmenti, a quest'ultimo viene comunicato un segmento con RST arrivo e la connessione viene interrotta. Il server quindi fa una fork, ovvero genera un processo per gestire la connessione con il client, spedice il SYN+ACK al mittente con lo stesso numero di sequenza ricevuto.
3. il client manda un segmento con bit ACK attivo, per validare il SYN+ACK del server. La connessione è ora stabilita.

Controllo di flusso

Il campo Window Size nel segmento TCP serve a regolare quanti Byte inviare/poter ricevere.

Il destinatario fa sapere quanti Byte può ricevere (se ha buffer di ricezione vuoto saranno uguali a quanto è grande il buffer). Man mano che il mittente invia blocchi di N Byte, il destinatario occuperà il buffer con quei dati, e risponderà indicando la nuova dimensione (diminuita) del buffer di ricezione, cosicchè il mittente sappia quanti Byte mandare al massimo nel prossimo segmento che dovrà inviare.

Se il server non può più ricevere, comunica al client che la sua window size è 0, e il client si metterà in attesa di una nuova comunicazione del server con window size > 0.

In caso di perdita dell'ACK che aggiorna dal server al client la window size disponibile (*window update*) ci sono due strategie di recovery:

- **Persistent Timer** (salvaguardia il client), ogni volta che viene ricevuto un segmento contenente window size = 0, il client si blocca e viene fatto partire un timer. Se non viene ricevuto un aggiornamento sul window size del server prima che il timer si esaurisca, il client (tramite il segmento *window probe*) richiede esplicitamente al server l'ACK perduto. In caso di N risposte mancanti, il client chiude la connessione.

- **Keep Alive Timer** (salvaguardia il server), il server ha un timer, allo scadere del quale invia un segmento *keep alive probe*, che controlla che il client sia ancora attivo. In caso di N risposte mancanti, il server chiude la connessione.

Ottimizzazione data transfer

Se si vogliono mandare segmenti piccoli, il traffico è oneroso, perchè ogni (per esempio inviando 1 carattere) segmento inviato genera un ACK di risposta (che può anche informare del buffer in ricezione pieno, quindi bisogna attendere il window update, etc). Oppure quando abbiamo un buffer di ricezione incredibilmente piccolo, sono più i Byte inviat per la gestione del flusso TCP che per i dati. Questo problema è chiamato *silly window sindrome*.

Ci sono tre soluzioni:

- **Delayed acknowledgements**, che consiste nel far attendere il server un tempo prefissato (200ms) dopo la ricezione di un segmento, e aspettare eventuali altri dati, così da inviare un unico ACK complessivo e ridurre quindi il numero di segmenti in rete. In caso di applicazioni utente che devono essere responsive (vedi telnet, server di echo), si potrebbe creare un delay inaccettabile per l'utente.
- **algoritmo di Nagle**, che è una versione modificata del delayed acknowledgements, per cui un'entità TCP può avere un solo "segmento piccolo" alla volta, in attesa di ACK. In questo modo il client invia un segmento piccolo e mentre aspetta la ricezione dell'ACK del server, il suo buffer di invio si riempie, così che quando sarà nuovamente possibile inviare (alla ricezione dell'ACK spedito dal server) riesca ad inviare più informazioni in un unico segmento.
- **algoritmo di clark**, consiste nel compattare i window update lato server, cioè ritardare la spedizione del window update (dopo il window update 0) finchè il buffer di ricezione del server abbia raggiunto la Maximum Segment Size oppure se raggiunge la metà della memoria disponibile. In questo modo il server non informerà mai il client se la dimensione della finestra è piccola, e quest'ultimo non invierà mai segmenti contenenti una quantità ridotta di dati.

Controllo degli errori

Il controllo degli errori di TCP è molto simile a quello di HDLC. La differenza più grande è che in HDLC i numeri di sequenza in invio e ricezione si riferivano a blocchi di dati, mentre in TCP si riferiscono a Byte dello stream totale.

Inoltre, dato che il RTT su internet è molto più grande che su singolo link dove HDLC opera, la dimensione della finestra non è derivabile dai numeri di sequenza. Esiste

invece un campo specifico nell'header TCP (window size) che indica ad ogni passo il massimo numero di Byte che possono essere ricevuti.

TCP usa ACK cumulativi e non usa NACK. Ogni segmento è validato da un ACK e per ciasuno di essi esiste un timer che viene resettato ogni volta che si riceve un ACK.

Fast Retransmit

I segmenti viaggiano su internet e non ho modo di sapere che arrivino nell'ordine corretto. Quindi ogni volta che al server arriva un segmento non in ordine (ne manca uno in mezzo) manda un ACK specifico avvertendo che si è "fuori sequenza". Quando poi al server arriverà/arriveranno i segmenti mancanti manderà un ACK avvertendo che tutto è ritornato ok (ACK cumulativo).

Lato client quindi non si rimanda subito il/i segmento/i fuori sequenza, ma si aspetta 3 ACK del server che avvertono del "fuori sequenza" prima di reinviare.

Questa tecnica comprende l'utilizzo di un timer di ritrasmissione (RTO) associato ad ogni segmento inviato lato client. Allo scadere dell'RTO, se non si è ancora ricevuto l'ACK, quel segmento "mancante" viene ritrasmesso. Il timer è necessario anche nel caso in cui il client non ha più segmenti da inviare, poichè non si accorgerebbe del segmento mancante perchè non riceverebbe più ACK.

RTO

Siccome siamo su internet, a quanto si dimensiona il timer RTO? Se troppo grande spreco tempo!

RTO dipende dall'RTT, che può variare durante una connessione, quindi dovràà variare anche RTO.

Inizialmente si parte con un valore di RTO relativamente elevato:

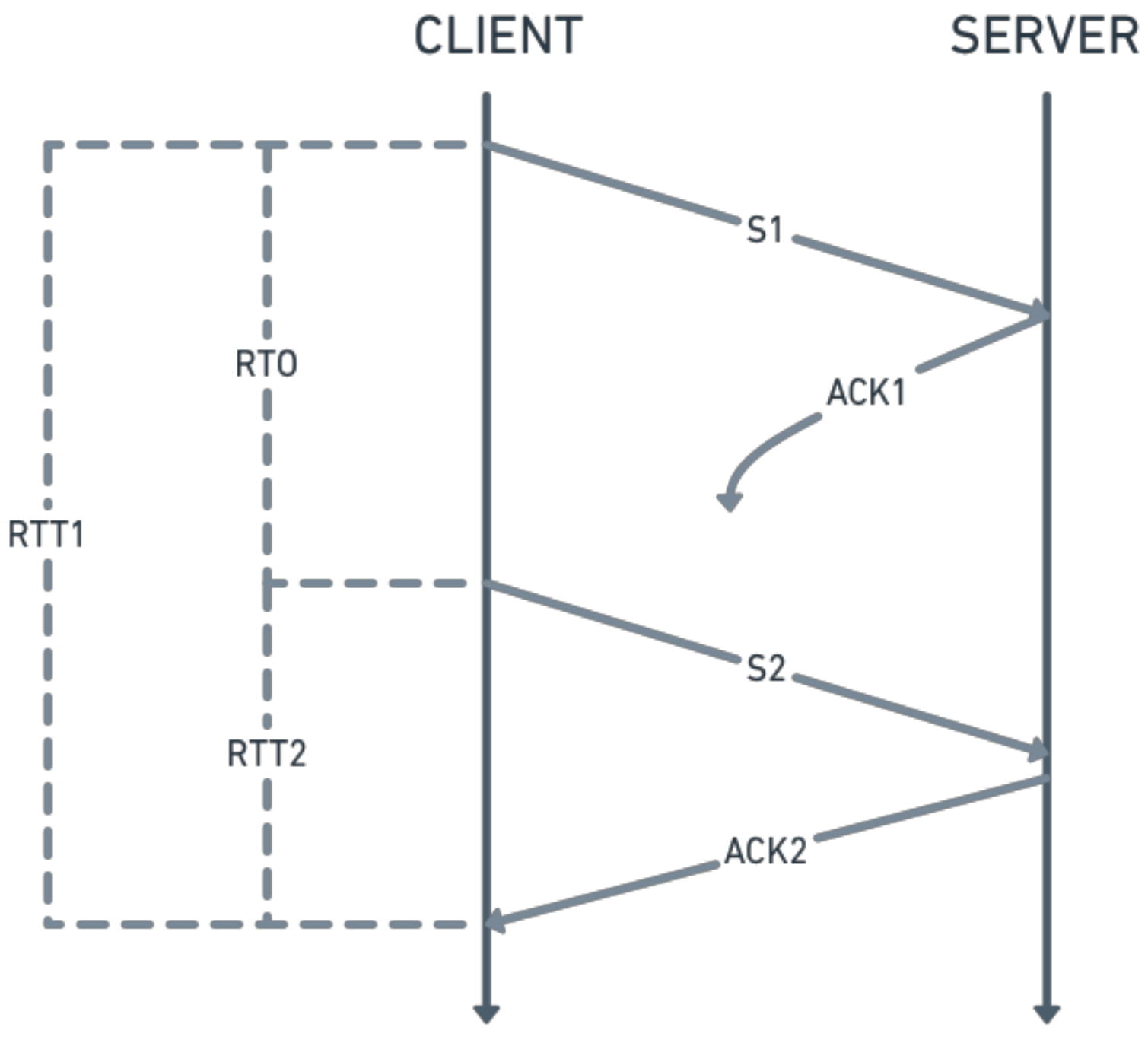
- primo RTO di circa 3 secondi
- secondo RTO si attiva alla ricezione del primo ACK data la formula: $RTO = RTT + 4 \cdot D,\ D = \frac{RTT}{2}$, dove D è l'approsimazione della deviazione standard sull'RTT
- RTO successivi, calcolati dopo che ricevo un nuovo ACK prima che il timer scada, aggiornando RTT e D della formula precedente come (con α e β sono circa 0.9, fattori di "smorzamento" ottenuti sperimentalmente e RTT_{now} è l'RTT misurato per il nuovo ACK):
  - $RTT_{new} = \alpha \cdot RTT_{old} + (1 - \alpha)RTT_{now}$
  - $D_{new} = \beta \cdot D_{old} + (1 - \beta)|RTT - RTT_{now}|$

- $RTO_{new} = RTT_{new} + 4 \cdot D_{new}$

> Per ogni RTO minore di 1 secondo, RTO viene fissato ad 1 secondo (quindi RTO non può mai essere minore di 1 secondo).
> Questa tecnica funziona bene solo se non ci sono errori.

Retransmission ambiguity problem

I calcoli dell'RTO visti non sono più attendibili se un segmento viene perso o, comunque, in caso di errore. Questo perchè se il mittente non dovesse ricevere ACK, dopo aver aspettato l'RTO calcolato lo rimanda. Ma come calcolare il nuovo RTO? Bisogna considerare l'RTT del primo segmento (che è ancora in corso non avendo avuto ACK di risposta), o di quello rimandato?



Quale RTT scelgo? RTT1 o RTT2? Entrambi non vanno bene, non sarebbero corretti rispetto allo stato della rete.

Per non avere problemi si adotta la **politica Karn**: se si verifica un errore, il nuovo RTO

è il doppio di quello vecchio. Dopodichè riparto come se avessi ricevuto il primo ACK.

Opzione timestamp

Nella realtà per finestre piccole (connessioni brevi) TCP fa una sola stima iniziale per il calcolo dell'RTT. Per finestre grandi invece tale stima viene fatta ogni 2 o 3 segmenti trasmessi. Se voglio avere strime più frequenti, va dichiarato in fase di apertura della connessione, specificando l'opzione **timestamp** nell'header TCP.

Ogni entità TCP possiede un time-out a 32 bit incrementato ad intervalli compresi tra 1ms e 1s. Per ogni segmento inviato, il mittente legge il valore del timer e lo inserisce nel segmento in corrispondenza del campo *timestamp value*.

Quando il destinatario riceve il messaggio, risponderà con il rispettivo ACK e includerà nel campo *timestamp echo reply* il valore letto nel campo timestamp value del segmento ricevuto. Così facendo, quando l'ACK è ricevuto dal mittente, questi può fare la differenza tra il valore attuale del suo time-out e quello scritto nell'ACK, per stimare l'RTT.

S-ACK (Selective ACK)

Invece che gli ACK cumulativi, si invia un ACK contenente le informazioni sull'ultimo segmento ricevuto in sequenza e l'ultimo segmento che potrebbe essere fuori sequenza. Quando il client riceve un ACK che indica un "fuori sequenza", viene spedito il segmento corretto.

Il S-ACK viene dichiarato in fase di apertura della connessione.

Controllo della congestione

Siccome TCP può avere più connessioni simultanee e non sa la capacità del canale, ma lavora solo sulla grandezza della finestra di invio, concordata con il ricevitore, adotta un approccio cauto.

Inizia scegliendo la finestra più piccola possibile, per poi aumentarla fino al limite imposto dal ricevitore. Quando avviene un errore si azzera tutto e si riparte.

Nel caso non si verifichino errori (o perdita ACK), TCP divide il trasferimento dati in 3 fasi:

1. **Slow start**, si trasmettono segmenti piccoli partendo dalla dimensione minima dei segmenti stabilita all'apertura della connessione e con dimensione finestra a 1. Ad ogni ACK ricevuto correttamente la dimensione della finestra raddoppia. Questa fase viene iterata fino a che non si raggiunge la soglia SST (Slow Start Threshold)

(pari alla metà della dimensione massima della finestra).

2. **Congestione Avoidance**. All'inizio di questa fase la finestra di congestione ha come dimensione SST. Da questo punto in poi (sempre se non si verificano errori) TCP trasmette finestre aumentandone la dimensione linearmente (aggiungo 1 segmento alla finestra dopo aver ricevuto tutti gli ACK per la scorsa finestra). Raggiunta la soglia massima trasferibile, si entra nella fase Constant.
3. **Constant**, durante questa fase si spediscono finestre della stessa dimensione, pari al massimo possibile (stabilito durante l'apertura della connessione).

Fast Recovery / Fast Retransmit

Il fast retransmit prevedeva che il client aspettasse 3 ACK "fuori sequenza" prima di reinviare un pacchetto perso. Per il controllo della congestione questo non è considerato un errore grave, in quanto il server ha comunque ricevuto quasi tutti i pacchetti.

Però alla ricezione del terzo ACK "fuori sequenza" si imposta un nuovo valore per SST pari all'attuale finestra dimezzata e si modifica comunque la grandezza della finestra di invio portandola al minimo tra l'SST iniziale e quello appena calcolato.

RTO Recovery

Se c'è un errore per cui scade il timer della trasmissione (RTO) allora viene considerato errore grave, perchè si ipotizza che la rete sia congestionata.

Si ricomincia con la fase di *slow start* e si imposta un nuovo SST pari alla metà della dimensione della finestra di invio nel momento dell'errore.

Chiusura della connessione TCP

- il client manda un segmento con bit FIN attivo
- il server riceve il segemnto e manda anch'esso un segmento con bit FIN attivo
- il client valida con ACK il FIN del server

Il client, dopo aver mandato l'ultimo ACK, aspetta un tempo regolato da un timer, pari a 2MSL (dove MSL è il Maximum Segment Lifetime, ovvero il massimo "tempo" nel quale un segmento può viaggiare in internet prima di essere droppato). Il client, aspettando, fa si che, qualora il suo ACK fosse andato perso, il server gli reinvii il suo FIN, cosicchè il client possa reinviare l'ACK.

Simultaneous close

Se siamo in una situazione peer-to-peer:

- entrambi gli attori mandano la FIN
- entrambi mandano l'ACK per il FIN ricevuto
- entrambi vanno in "timed-wait" per essere sicuri che l'ACK arrivi

Half close

È possibile che solo una parte abbia finito l'invio dei dati, mentre l'altra abbia ancora dei segmenti pendenti:

- dopo l'invio della FIN di uno dei due attori (A), essa viene validata dall'altro (B), e A manda l'ACK finale
- prima di chiudere però, B deve aspettare che tutti i suoi pacchetti pendenti siano inviati
- al termine degli invii viene effettuata la FIN da B verso A e ci si comporta come una normale chiusura

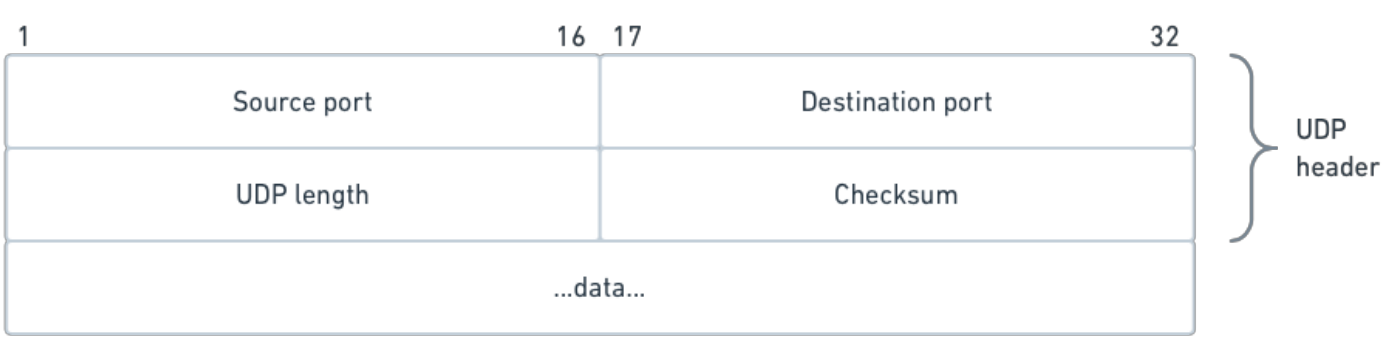
UDP (User Datagram Protocol)

- no connessioni
- nessuna aggiunta al servizio del livello IP
- appena pronto un segmento, questo viene inviato
- la chiusura è locale (non si informa l'altra parte)
- non ci sono ACK
- nessun controllo
- molto più veloce di TCP
- non viaggia frammentato, ma in un'unica entità (se necessaria frammentazione viene fatto a livello applicazione)

È solitamente usato per l'invio di informazioni di piccole dimensioni. Eventuali errori vengono gestiti a livello applicazione, che si occuperà eventualmente di ritrasmettere i segmenti.

Formato segmenti UDP

L'header UDP è di 8 Byte.



- **UDP length**, in teoria la dimensione massima è di 65535 Byte, ma poichè non affidabile, si spediscono sempre segmenti con lunghezza massima 8192 Byte
- **checksum**, è opzionale e formato come quello di TCP. Se non viene utilizzato ha tutti zeri

Performance e ottimizzazioni

Ci sono alcuni aspetti per cui la vera implementazione differisce da quella studiata in astrazione:

- **i livelli 3,4,7 si scambiano solo puntatori alla struttura dati**, per cui nella costruzione dei vari oggetti (pacchetto, segmento, etc) non si passa al livello sottostante l'intero header aggiunto dal livello, ma solo puntatori a dove risiede effettivamente l'header creato.
- **gli header sono formati attraverso la modifica di header preconfezionati (Template Header)**
- **timer virtuale (o logico)**, tecnica per gestire la coda dei timer, in quanto avere n timer quanti sono i segmenti inviati sarebbe troppo oneroso. Si ricorre dunque a liste puntate contenenti in ogni elemento il valore di tempo da aggiungere a quello precedente e un "descrittore di segmento" per riuscire ad associargli il segmento corretto. Ad ogni colpo di clock viene decrementato il valore del primo elemento della lista finchè non raggiunge 0 (e viene tolto). Un elemento viene tolto anche quando arriva l'ACK corrispondente.

Livello 7 "Application"

Tutti i livelli dall'1 al 4 hanno una funzionalità ben precisa. Il livello 7 invece fonde in se funzionalità diverse, tante quanti sono i servizi offerti all'utente finale:

- SMTP (posta elettronica)
- FTP (file transfer)
- DNS (risolutore di nomi logici in indirizzi IP)
- HTTP (web protocol)
- DHCP, implementato a livello 7 ma offre funzionalità di livello 3

DNS (Domain Name System)

Permette la mappatura da nome simbolico a indirizzo IP e viceversa.

> Es: www.unimi.it
> www -> identificatore del servizio (web)

> unimi.it -> nome del dominio -> 10.11.128.2

Struttura DNS

Tutti i dati che vengono usati dal DNS sono chiamati *domain space name* (spazio dei nomi di dominio) e sono indicizzati per nome. La struttura è gerarchica in quanto:

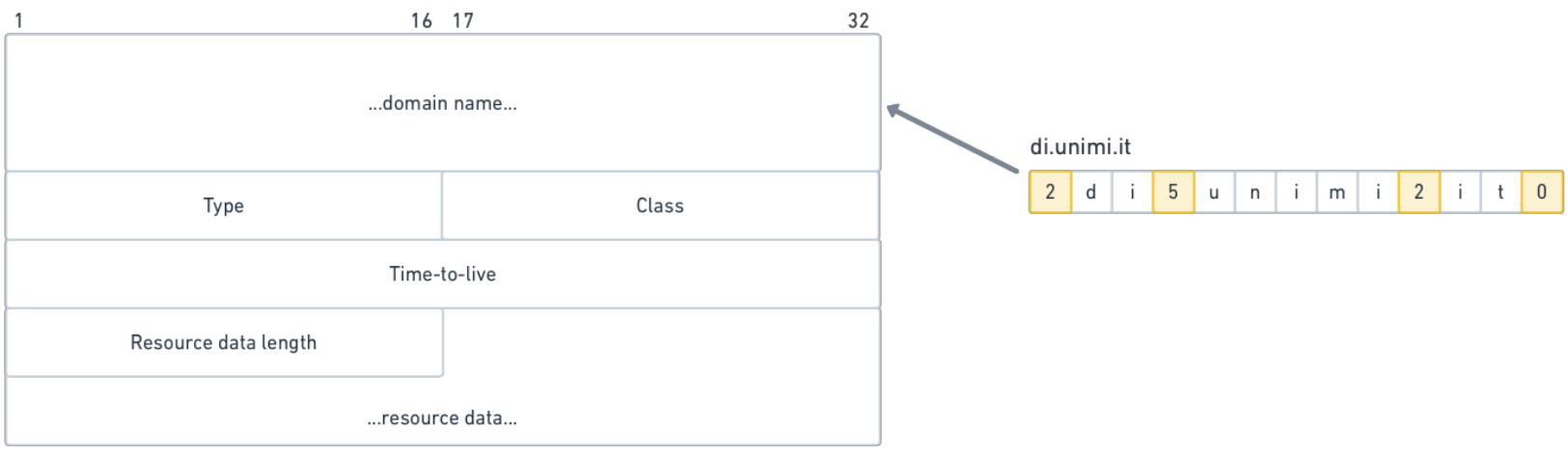
- è amministrabile in modo distribuito
- si sceglie di assegnare i nomi in relazione alla locazione geografica, in modo che molte delle richieste di risoluzione di un nome possano essere risolte più velocemente e senza introdurre traffico in zone non interessate (vale il principio di località spaziale)

I domini sono divisi in:

- **generic domains**, sono domini che esistevano quando internet era ancora relativamente piccola e quindi bastavano per identificare una particolare organizzazione (nati in america)
- **country domains**, sono domini nati con la crescita di internet, introducono domini specifici per ogni nazione

Resource records

Ogni dominio può avere informazioni aggiuntive al solo indirizzo IP. Esse sono memorizzate in uno o più records, tutti indicizzati dal nome di dominio relativo. Un resource record è così formato:

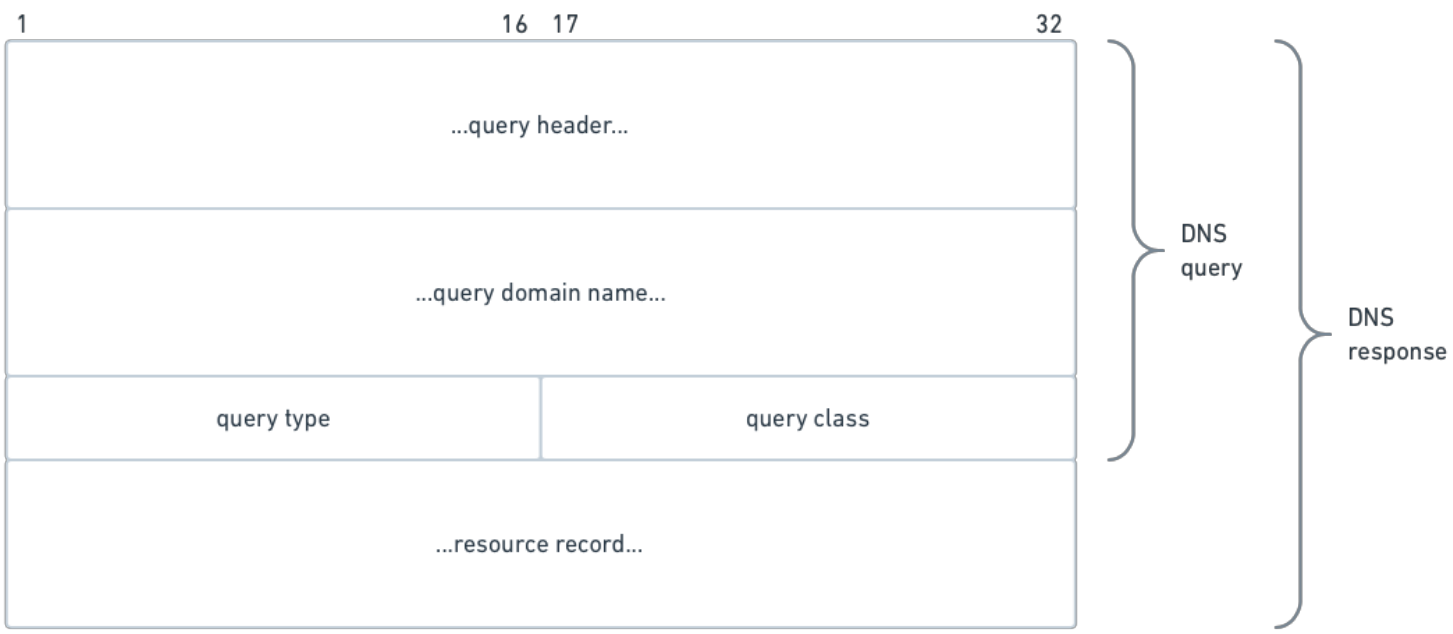


- **domain name**, è il nome di dominio a cui il record è correlato. Prevede un counter (casella in giallo) prima di ogni "etichetta", che ne determina la lunghezza. L'ultimo carattere è sempre lo 0, che rappresenta la root
- **type**, indica il tipo di record che può essere:
  - A (1), indirizzo IP in forma binaria
  - NS (2), name server
  - PTR (12), indirizzo IP in notazione decimale puntata

    - HINFO (13), indica la presenza di informazioni sul tipo dell'host e sul sistema operativo
    - MX (15), contiene un email gateway che si può utilizzare per instradare messaggi di posta
    - CNAME (Canonical Name Record), indica un'associazione canonica (alias)
- **time-to-live**, indica il tempo in secondi entro i quali l'informazione contenuta nel record è valida (necessaria per motivi di caching, il valore tipico è 2 giorni)

DNS query messages

Per operare su database di indirizzi dei DNS, i messaggi DNS sono "query". Sono di piccola dimensione e viaggiano attraverso UDP (se si perde un messaggio lo rimando, anche perchè sarebbe più costoso aprire la connessione che il trasferimento della query). Il formato è il seguente:



- **query header**, è una stringa standard da 12 Byte e contiene 16 bit di identificazione dove viene memorizzata l'identificativo per associare richiesta con risposta, e 16 bit di flags, 1 per capire se è una richiesta o una risposta e 4 per capire il tipo della ricerca
- **query domain name**, contiene il nome di dominio a cui la richiesta è associata, nello stesso formato visto all'interno dei resource records

Risoluzione nomi

Lo spazio del dominio dei nomi è suddiviso in zone, ognuna delle quali mantiene e gestisce solo una parte dei domini. Ogni zona è poi amministrata da un'autorità separata, che è anche responsabile di formire uno o più server dei nomi (**name server**) per la zona.

Associati ad ogni zona troviamo quindi un primary name server, ma possibilmente

anche uno o più secondary name server. Il primary contiene effettivamente i records relativi alla sua porzione del domain name space, mentre i secondimantengono una cache dei dati contenuti nel primo. Il *caching* avviene quando, verificando che il record richiesto non è presente nel proprio database, i secondary o primary server indirizzano la richiesta al server di livello superiore e, alla ricezione dell'indirizzo IP richiest, mantengono una copia volatile della risposta per il TTL previsto (già visto che è 2 giorni).

Un nome completo di dominio (es. di.unimi.it) viene chiamato **FQDN (Fully Qualified Domain Name**.

Recursive name resolution

Se ogni primary potesse parlare con tutti gli altri primary ci sarebbe un overhead altissimo. Si è quindi deciso di non far conoscere ai primary server l'indirizzo IP degli altri primary server, ma di permetterne il contatto solo con un insieme ristretto di top-level root name servers, che mantengono le coppie nome-indirizzoIP di tutti i primary server e, a ogni richieta, comunicano quale primary server interpellare. Questo metodo di risoluzione dei nomi è chiamato **recursive name resolution**.

In pratica, avendo un local name server che vuole conoscere l'indirizzo IP di "unimi.it" si ha:

- un local name server richiede la risoluzione del nome "unimi.it" a uno dei top-level root name servers
- il top-level root name server gli risponde con l'indirizzo IP di un altro name server, quello che si occupa di ".it"
- il local name server allora chiede a questo nuovo name server
- il name server che si occupa di ".it" risponde con l'indirizzo IP di un altro name server, quello che si occupa di "unimi.it"
- il local name server allora chiede a questo nuovo name server
- il name server che si occupa di "unimi.it" risponde con l'effettivo indirizzo IP richiesto di "unimi.it"

Qualora il local name server avesse già in cache l'informazione, la restituisce senza interpellare i name server di livello superiore.

Iterative name resolution

Quest'altra tecnica propone invece di interpellare sempre il server di livello superiore più vicino (ovvero non si interpella il name server di root, ma quello del dominio più alto, come .it nel nome unimi.it). Questo implica che i primary server non debbano

conoscere gli indirizzi IP dei top-level root servers ma del server del livello subitamente superiore.

In pratica, avendo un local name server che vuole conoscere l'indirizzo IP di "unimi.it" si ha:

- il local name server chiede al name server che si occupa di ".it"
- il name server che si occupa di ".it" risponde con l'indirizzo IP del name server che si occupa di "unimi.it"
- il local name server chiede al name server che si occupa di "unimi.it"
- il name server che si occupa di "unimi.it" risponde con l'indirizzo IP richiesto in origine, ovvero quello di "unimi.it"

Posta elettronica

Vi sono due macchine client che vogliono scambiarsi messaggi. La comunicazione avviene tramite due macchine server di posta elettronica, una per il client e una per il server. Ogni client comunica col proprio server di posta elettronica.

Gli attori sono i seguenti (per ciascuno, sia mittente che destinatario):

- 2 **User Agent (UA)** che risiedono rispettivamente a livello 7 del client e del server di posta del client.
  - UA del client permette di gestire e inviare al UA server le mail
  - UA del server permette di interagire con l'UA client
- 2 **Message Transfer Agent (MTA)** che risiedono entrambi nel livello 7 del server di posta del client.
  - MTA "client" si occupa di sincronizzare i dati con i client, mantenendo per ognuno di essi una IN mailbox nota come *message store* (usa porta *x* e protocollo SMTP)
  - MTA "server" si occupa di cercare il server di posta elettronica del destinatario, interrogando il DNS locale (usa *well-known-port* 25 e TCP)

Fasi di comunicazione:

1. il mittente prepare la mail compilando i campi necessari e invia il messaggio
   - destinatario: utente@sottodominio.dominio
   - per l'invio si usa protocollo POP3 o IMAP (POP3 in disuso, IMAP nel caso dev'essere fatta copia del messaggio inviato sia su server che su client)
2. il server di posta del mittente elabora e invia il messaggio al server di posta del destinatario

    - simile al server TCP
    - si usa coda FIFO per i messaggi da inviare
    - si risolve l'indirizzo IP del server di posta del destinatario (es. utente@sottodominio.dominio -> utente@10.11.172.2)
3. i server di posta comunicano fra loro
    - si usa protocollo SMTP
    - il destinatario non deve necessariamente essere presente al momento dell'invio (comunicazione asincrona)
    - il client destinatario si collegherà al suo server di posta per verificare i messaggi arrivati

Struttura messaggi mail

Si ha un header e un body:

- **Header** (tutti i campi hanno un formato standard)
    - Usati dal sistema di trasferimento messaggi
        - **From**
        - **To**
        - **Cc**
        - **Received**
        - **Return-Path**
    - Usati dall'UA del client
        - **Sender**
        - **Date**
        - **Message-Id**
        - **Reply-to**
        - **Subject**
    - Definiti dall'utente (esempi)
        - **X-PhoneNumber**
        - **X-FaxNumber**
- **Body** è il messaggio vero e proprio

Tutti i dati che viaggiano via mail sono nella codifica ASCII (128 caratteri US, 2^7 caratteri su 7 bit). In genere, se i dati sono stringhe di caratteri, tutti i campi dell'header e del body sono convertiti in NVT (Networ Virtual Terminal) ASCII, codifica in cui ogni carattere asci da 7 bit è prefissato con uno 0 (NUT).

Il contenuto del messaggio può essere composto solo da linee con al massimo 1000 caratteri NVT ASCII. Ogni riga termina con la sequenza "\r\n" (Carriage Return CR,)

Line Feed LF) e l'ultima riga del messaggio finisce con il carattere speciale punto ".".

MIME (Multipurpose Internet Mail Extensions)

Permette l'invio di tipi di dati diversi attraverso la posta elettronica, come dati binari, audio, video, mantenendo lo stesso sistema di trasferimento dei messaggi.

Con MIME vengono introdotti opportuni campi aggiuntivi all'header della mail:

- **MIME-Version**, definisce la versione MIME in uso
- **Content-description**, descrizione breve testuale del contenuto del messaggio
- **Content-Id**, identificatore univoco assegnato dal UA
- **Content-Transfer-Encoding**, la sintassi di trasferimento usata
- **Content-length**, il numero di Byte nel body del messaggio

Quindi se **MIME-Version** è presente, informa il destinatario che il contenuto della mail non è normale testo NVT ASCII.

Alcuni valori possibili di **Content-Type** sono: *Text, Image, Audio, Video, Application, Message*.

Codifica BASE64

Per essere trasferiti, i contenuti non ASCII, vengono convertiti in BASE64, una codifica nella quale:

- ogni dato viene diviso a blocchi di 24 bit
- ogni blocco viene diviso in 3 sottoblocchi da 6 bit ciascuno
- ogni sottoblocco viene interpretato come ASCII attraverso una tabella che mappa i valori da 0 a 63 (quelli possibili in 6 bit) in un carattere ASCII

Se dovessi non avere almeno 24 bit di blocco, aggiungo del padding di caratteri "=".

FTP (File Transfer Protocol)

Usato per trasferimento file. I dati viaggiano separati dal controllo, in due canali (connessioni TCP) distinte (protocollo out-of-band):

- porta 21 per il controllo
- porta 20 per i dati

Il client richiede comunicazione FTP sulla porta 20 del server, indicando la porta del client nella richiesta. Verrà quindi allacciata la comuicazione anche per il canale di contorllo sulla porta 21 e sulla porta usata dal client per i dati + 1.

Nella comunicazione FTP ci sono diversi comandi/operazioni disponibili e diversi codici di controllo/errore ("FTP server ready", "password required for < username >", etc).

HTTP (HyperText Transfer Protocol)

Protocollo in-band (una sola comunicazione TCP per dati e controllo). Si tratta di un file transfer (FTP) mascherato dove usa la porta 80 lato server.

All'inizio si apriva una nuova connessione per ogni trasferimento (troppo lento causa "slow start" TCP ad ogni riapertura connessione). Ora le connessioni sono "persistent" per un quanto di tempo, con la versione 1.1 del protocollo. I dati vengono codificati in NVT ASCII.

Passi principali:

1. determinazione URL
2. interrogazione DNS per risoluzione nome
3. attesa risposta DNS
4. connessione TCP sulla *well-known-port* 80 dell'IP risultato della query DNS
5. GET della pagina richiesta
6. attesa risposta server
7. rilascio della comunicazione TCP
8. visualizzazione immagini
9. fetch e visualizzazione di specifiche parti delle pagine (immagini, video, etc.)

Possibile utilizzo di "proxy" (intermediari) che cachano le risorse (o le recuperano dal server se modificate dall'ultima volta). Il web è molto cachato.

SIP (Session Initiation Protocol) per Voice Over IP

Usato per stabilire, gestire e terminare comunicazioni di telefonia IP e VoIP. Adatto per trasferimento di audio, video, messaggistica testuale, etc). Inoltre favorisce un'architeturra scalabile e modulare ed è, ad oggi, il più usato.

Usa UDP perchè non serve ritrasmettere un informazione (non voglio sentire la voce di N secondi fa) e perchè è più veloce.