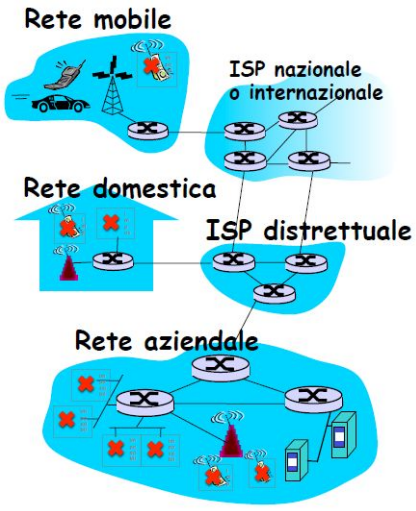
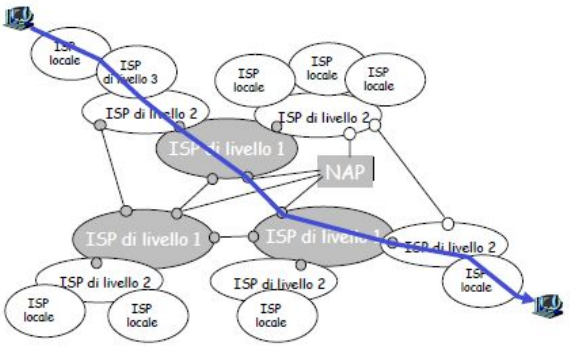
Introduzione

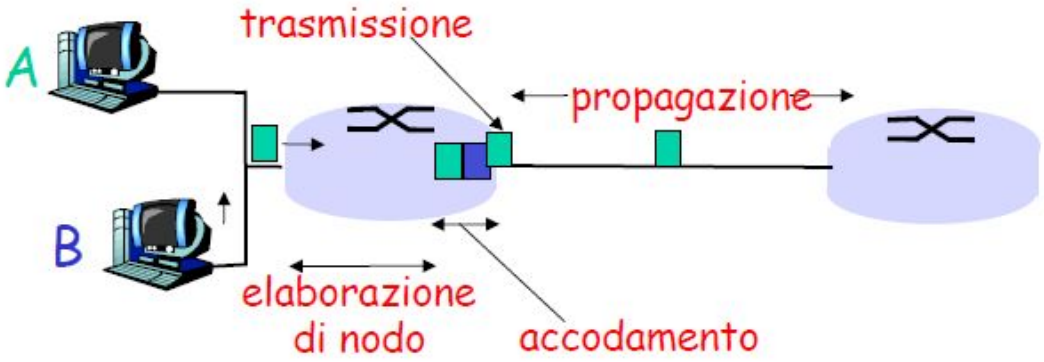


Un protocollo definisce il formato e l'ordine dei messaggi scambiati tra due o più entità in comunicazione, così come le azioni intraprese in fase di trasmissione e/o ricezione di un messaggio o di un altro evento. NON DEFINISCE INVECE L'IMPLEMENTAZIONE di tutto ciò.

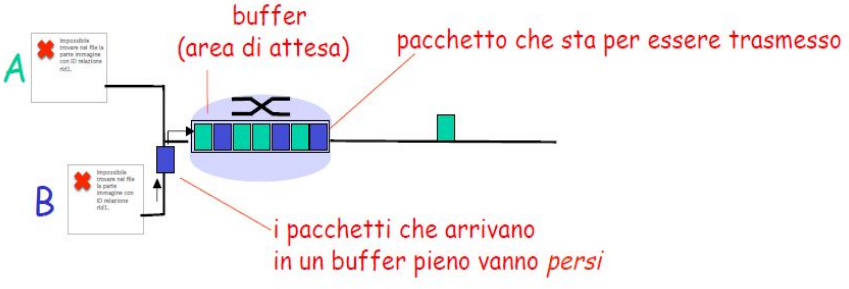


Ritardi:

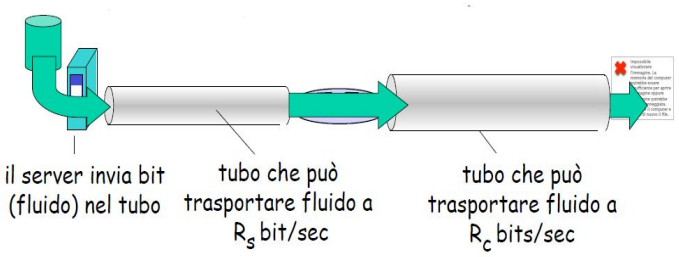
1. **Ritardo di elaborazione del nodo:**
   - controllo errori sui bit
   - determinazione del canale di uscita
2. **Ritardo di accodamento**
   - attesa di trasmissione
   - livello di congestione del router
3. **Ritardo di trasmissione (= L/R):**
   - R = frequenza di trasmissione del collegamento (in bps)
   - L = lunghezza del pacchetto (in bit)
4. **Ritardo di propagazione (= d/s)**
   - d = lunghezza del collegamento fisico
   - s = velocità di propagazione del collegamento (~2x108 m/sec)

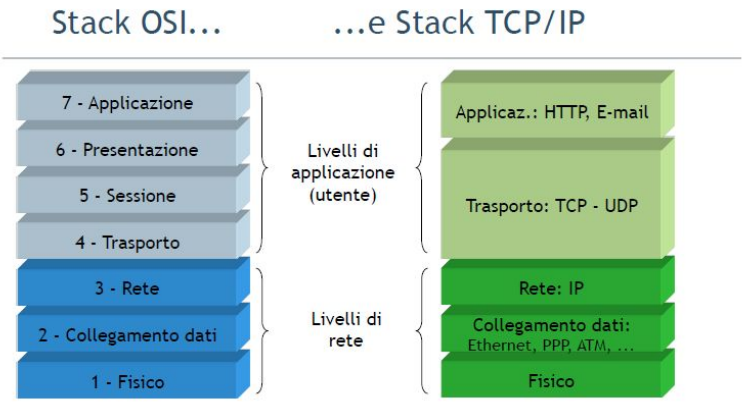


- **Perdita di pacchetti:** La coda di un router ha capacità finita, quando un pacchetto trova una coda piena, viene scartato. Esso può essere ritrasmesso dal nodo precedente oppure no.



- **Throughput:** frequenza di bit/secondo trasferiti dal mittente al destinatario
    - Collo di bottiglia: collegamento di un percorso punto-punto che vincola un throughput end-to-end





Livello 5: Applicativo

Principi delle architetture di rete:

- **Client-Server**
    - Sempre online e reperibile, in attesa di client, IP statico
    - Il Client può cambiare indirizzo
- **Peer-2-Peer (P2P)**
    - Coppie arbitrarie di peer
    - I peer non sono necessariamente sempre online, possono cambiare indirizzo
- **Architetture ibride tra P2P e Client-Server**

Concetto di "Socket" e "Processi Comunicanti":

- **Processi Comunicanti:** processo=programma in esecuzione un un host.
    - Se sullo stesso host, usano schemi di comunicazione interprocesso messi a disposizione dal SO.
    - Se su host diversi, utilizzano i socket per scambiarsi messaggi.
- **Socket:**
    - Associa un indirizzo a una porta
    - Utilizzato dai processi per inviare e ricevere messaggi, presuppone l'esistenza di un'infrastruttura esterna che trasporta il messaggio fino alla socket di destinazione.

Servizi richiesti dal livelli applicativo:

- **Perdita di dati:** tolleranza alla perdita di pacchetti.
- **Temporizzazione:** ritardo massimo nella consegna dei messaggi.
- **Throughput:** quantità minima di banda disponibile
- **Sicurezza:** cifratura, etc.

Applicazione	Tolleranza alla perdita di dati	Throughput	Sensibilità al tempo
Trasferimento file	No	Variabile	No
Posta elettronica	No	Variabile	No
Documenti Web	No	Variabile	No
Audio/video in tempo reale	Sì	Audio: da 5 Kbps a 1 Mbps Video: da 10 Kbps a 5 Mbps	Sì, centinaia di ms
Audio/video memorizzati	Sì	Come sopra	Sì, pochi secondi
Giochi interattivi	Sì	Fino a pochi Kbps	Sì, centinaia di ms
Messaggistica istantanea	No	Variabile	Sì e no

Applicazione	Protocollo a livello applicazione	Protocollo di trasporto sottostante
Posta elettronica	SMTP [RFC 2821]	TCP
Accesso a terminali remoti	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
Trasferimento file	FTP [RFC 959]	TCP
Multimedia in streaming	HTTP (es. YouTube) RTP [RFC 1889]	TCP o UDP
Telefonia Internet	SIP, RTP, proprietario (es. Skype)	Tipicamente UDP

TCP	**UDP**
+ Orientato alla connessione + Affidabile + Controllo di flusso + Controllo di congestione - Temporizzazione, sicurezza, ampiezza di banda	+ Trasferimento dati inaffidabile - Orientato alla connessione, affidabilità, flusso, congestione

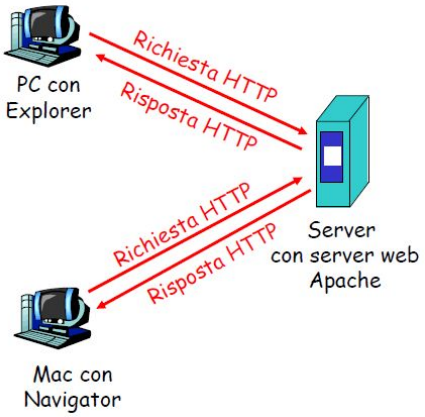
Protocollo HTTP

- **H**yper**t**ext **T**ransfer **P**rotocol
- La **porta standard** è la **80**.
- **Come funziona:** Viene aperta una connessione TCP verso il server → Il server accetta la connessione → I messaggi vengono scambiati tra client e server → Chiusura della connessione.
- **HTTP è "stateless"**, il server non mantiene traccia delle richieste che ha ricevuto in passato dai client.
- Il client e il server supportano **due metodi di connessione:**
  - **Keep-Alive, persistente:** più elementi(css, immagini, etc) vengono trasferiti utilizzando la stessa connessione TCP.
  - **Closed, non persistente:** Per ogni elemento viene aperta una connessione TCP diversa.

HTTP non persistente	**HTTP persistente**
2RTT per oggetto trasmesso	Il server lascia la connessione aperta dopo aver spedito la risposta
Sovraccarico del SO per ogni connessione TCP	I successivi messaggi vengono inviati nella stessa connessione aperta
Spesso i browser aprono connessioni TCP parallele	Il client invia una richiesta non appena incontra un oggetto referenziato
	Meno di un RTT per oggetto

- Invio dati di un form:
  - **POST:** I dati sono inviati nel campo entità e sono spediti al submit del form.
  - **GET:** I dati sono messi nell'url della richiesta.
- **Metodi disponibili HTTP/1.0:**
  - GET
  - POST
  - HEAD

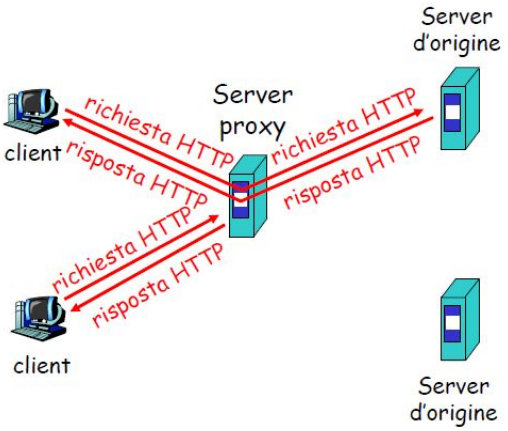
- **Metodi disponibili HTTP/1.1:**
  - GET
  - POST
  - HEAD
  - PUT (upload file nel percorso dell'URL)
  - DELETE (rimozione file nel percorso specificato nell'URL)
- Nelle risposte HTTP è indicato il response code e una piccola frase:
  - **200 OK**
  - **301 MOVED PERMANENTLY** (questa e le successive richieste andranno dirette ad un altro URI)
  - **302 MOVED TEMPORARILY**
  - **400 BAD REQUEST** (la richiesta non può essere soddisfatta a causa di errori di sintassi)
  - **403 FORBIDDEN**
  - **404 NOT FOUND**
  - **500 INTERNAL SERVER ERROR**



- **I cookie:**
  - Sono salvati lato client per memorizzare delle informazioni riguardanti l'utente.
  - Vengono utilizzati per mantenere le sessioni, per autorizzazioni e suggerimenti.
  - Alla prima connessione viene creato un id lato server in un database e il cookie viene spedito al client perchè lo possa memorizzare.
  - Vengono inviati dal client a ogni richiesta che fa al server, in modo che esso sappia riconoscerlo.

Web cache

È un proxy che ha come obbiettivo quello di fornire risposta a delle richieste HTTP senza dover interrogare il server di origine di continuo, frapponendosi tra il client e il server di destinazione.

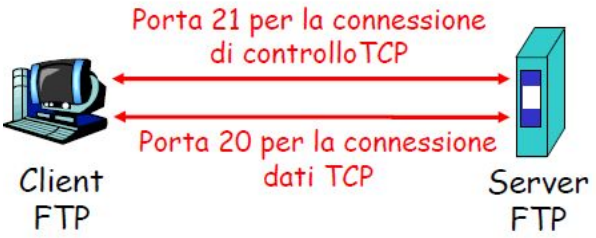


- **Tutte le richieste passano per il proxy**, che ha una **cache interna** dove memorizza il risultato delle richieste precedenti. Se il risultato di una richiesta fatta da un client(come una pagina html) **la ha già in memoria e non è scaduta**, la manderà al client, **altrimenti, se scaduta o mancante**, contatterà **il server di destinazione** e oltre a fornire il risultato della richiesta al client, **la memorizza** nel caso di altre richieste uguali.
- è **utilizzato in grosse reti**, come quelle di università, aziende, etc, dove ci sono grossi volumi di traffico verso la rete esterna.
- Serve a **ridurre i tempi di risposta** e a **ridurre il traffico** con la rete esterna
- Utilizza delle **GET condizionali** per sapere se **l'oggetto che ha in memoria è troppo vecchio** rispetto a quello del server di origine.

Conditional GET

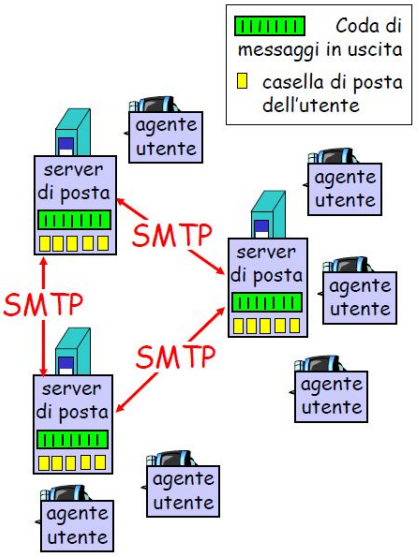
L'obiettivo è quello di **non inviare l'oggetto se la cache** ha una **versione aggiornata** del file. La data specifica dell'oggetto presente in cache viene copiata nella richiesta http e il server risponde con un **304 Not Modified** se la cache è aggiornata **o con l'oggetto** se l'oggetto in questione è stato modificato di recente sul server di origine.

Protocollo FTP



- **F**ile **T**ransfer **P**rotocol (RFC 959).
- Serve per **trasferire file da o verso un host remoto.**
- Rientra nel **modello client/server**, infatti è necessario che sia installato l'applicativo server ed è necessario usare un client.
- La **porta standard controllo** è la **21**, la **porta standard dati** è la **20**.
- Stabilisce due connessione **TCP**, una è il **canale dati (data connection)** e una **connessione di controllo fuori banda ("out of band")** per mandare comandi **(control connection).**
- Dopo il completamento del trasferimento dei file il canale dati viene chiuso lasciando aperto il canale di controllo in attesa di altri comandi.
- Include **status code** e **frasi** come il protocollo HTTP (es 331 Username OK, password required).
- I **comandi** sono inviati **come testo ASCII:**
  - STOR: archivia un file su un host remoto
  - LIST: lista di file nella dir corrente
  - RETR: riceve il file
- **Come funziona:** Una volta che il client ha aperto una connessione con il server(connessione di controllo), può iniziare a dare dei comandi in maniera interattiva, a ogni trasferimento viene aperta una connessione dati diversa per ogni file, che viene chiusa al termine del trasferimento di quel singolo file, mentre la connessione di controllo rimane aperta fino al termine della sessione.

Posta Elettronica

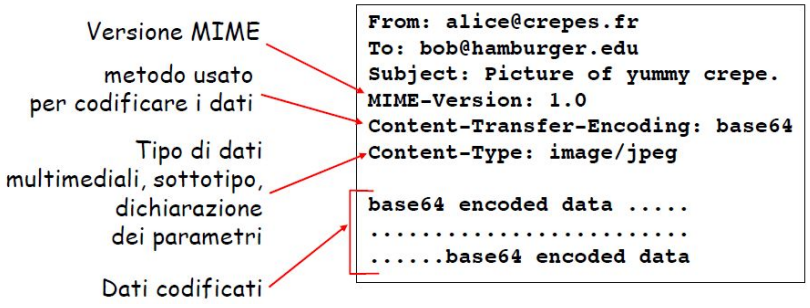


- I tre componenti principali: **User Agents, Mail servers** e **trasferimento email (SMTP)**.
- User Agents:
  - Possono creare e spedire email, leggerle, eliminarle, etc.
  - Sono programmi come Outlook, Thunderbird, Gmail, etc.
- Mail Servers:
  - **Contengono le caselle di posta** degli utenti.
  - I **messaggi da spedire** vengono **accodati.**
  - I server email per comunicare tra di loro (scambiare email) utilizzano il protocollo SMTP (Simple Mail Transfer Protocol).
  - **Sono sia server che client** in base a se stanno inviando una mail o la stanno ricevendo.
  - Alcuni software mailserver molto usati sono Dovecot, Curier, Microsoft Exchange.
- SMTP:
  - Usa **TCP** e la **porta 25** per avere una connessione affidabile.
  - **Trasferimento diretto:** i server comunicano direttamente tra di loro.
  - Il trasferimento delle email si svolge in 3 fasi:
    - Handshaking (detto greeting)
    - Trasferimento dei messaggi
    - Chiusura
  - L'iterazione tra server avviene tramite **richiesta/risposta** come il protocollo HTTP.
  - I messaggi scambiati sono in **ASCII a 7 bit.**
  - Usa una **connessione persistente,** cioè tutte le email e gli oggetti che compongono l'email viaggiano lungo la stessa connessione

- Usa una combinazione di carriage return (\r) e line feed (\n) **CRLF.CRLF.** per indicare la fine del messaggio
- I comandi utilizzati da SMTP sono:
  - HELO
  - MAIL
  - FROM
  - RCPT TO
  - DATA
  - QUIT

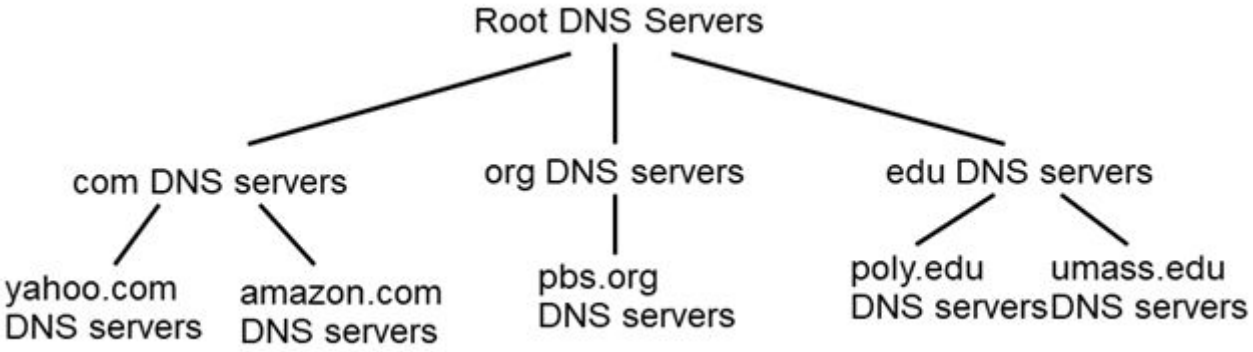
```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <rob@hamburger.edu>
S: 250 rob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like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

- Mail Access Protocols:
  - **POP:** Post Office Protocol, gestisce l'autenticazione e permette il download dei messaggi, che vengono scaricate sull'user agent svuotando la casella email sul server (**POP3,** un successivo aggiornamento di POP, permette di mantenere una copia sul server e altre funzionalità, è il protocollo di accesso ai mailserver ad oggi più usato dopo IMAP).
  - **IMAP:** Include più funzionalità rispetto a POP, come la modifica dei messaggi direttamente sul server, permette all'utente di organizzare le diverse email in cartelle, può scaricare sul'user agent i messaggi come POP.
  - **HTTP:** Utilizzato da alcuni servizi di email online come GMail, Hotmail, etc. L'applicazione web interagisce con il server email e fornisce un accesso attraverso Browser.
- **Formato dei messaggi di posta elettronica**
  - Sono formati da un'**intestazione** e un **corpo**.
  - L'intestazione è formata da **campi differenti dai comandi SMTP**:
    - To/A:
    - From/Da:
    - Subject/Oggetto:
  - Il corpo del "messaggio" è formato **solamente da caratteri ASCII**.
  - Estensioni per i file multimediali:
    - **MIME**: estensioni di messaggi di posta multimediali, RFC 2045, 2056.
    - Alcune **righe aggiuntive nell'intestazione dei messaggi** dichiarano il tipo di contenuto MIME.

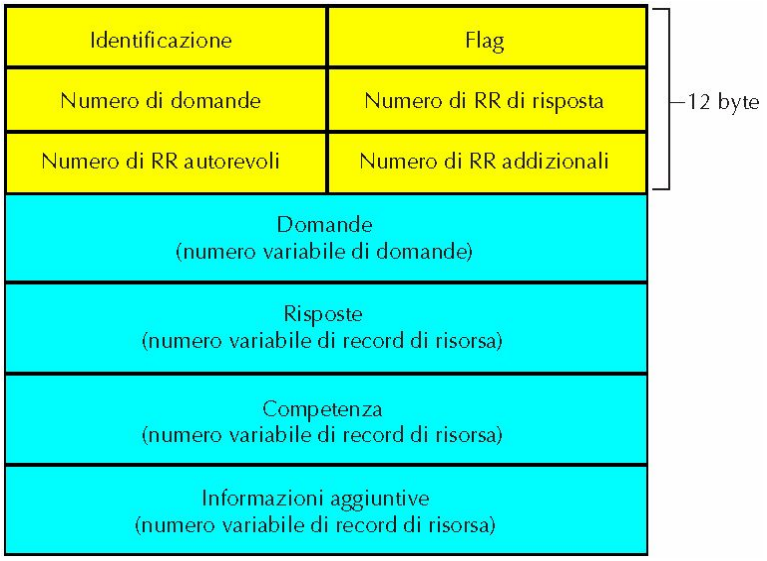


Domain Name System

- Il DNS è un **DB distribuito** (non può essere centralizzato per evitare il cedimento del nodo per troppo traffico, per la distanza DB/host e per la manutenzione) **gerarchico**.
- Il livello applicativo comunica con diversi **name servers** per risolvere un nome ed **ottenere l'IP.**
- Il DNS fornisce servizi di
  - **host aliasing** (se ho più siti sullo stesso server)
  - **mail server aliasing**
  - **load distribution** (più IP corrispondono ad un unico nome



- La **root** DNS è il **.** finale dell'url (non visibile nei browser).
- Per risolvere l'indirizzo si parte dal fondo: **. → com. → yahoo.com.**
- I domini org, net, edu, jobs.. e tutti i domini di stato sono chiamati **TLD (Top Level Domain)** e sono gestiti da enti internazionali.
- **Server DNS locale**: funge da proxy locale dei domini, tutte le query vengono mandate a lui, che poi gestirà la risoluzione in modo
  - **ricorsivo** (si affida la traduzione al DNS contattato)
  - **iterativo** (il DNS contattato risponde con il nome del prossimo server da contattare)
- Tutti i DNS non appena risolvono un dominio lo salvano in **cache** per un periodo di tempo prestabilito (solitamente vengono cachati gli IP dei TLD, per evitare di visitarli troppo)
- **Record DNS**: sono **RR (Resource Record)** del database del DNS salvate nel formato **(name, value, type, ttl)**, il tipo varia il significato di **(name → value)** in base alla funzione richiesta:
  - A: hostname → IP
  - NS: dominio → hostname del server di competenza
  - CNAME: alias del dominio → dominio
  - MX: dominio del server di posta → nome
- *Formato messaggi* (campi gialli da 16 bit)
  - **Identificazione**: numero per la domanda, la risposta alla domanda ha lo stesso numero
  - **Flags**
    - domanda o risposta
    - richiesta di ricorsione
    - ricorsione disponibile
    - risposta di competenza

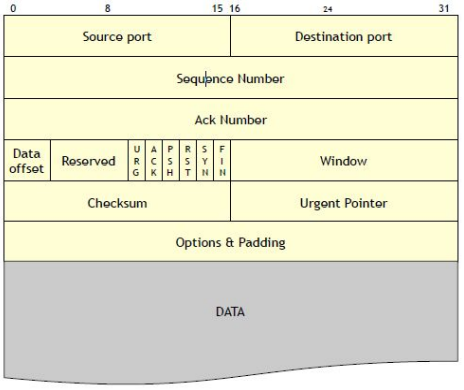


Livello 4: Trasporto

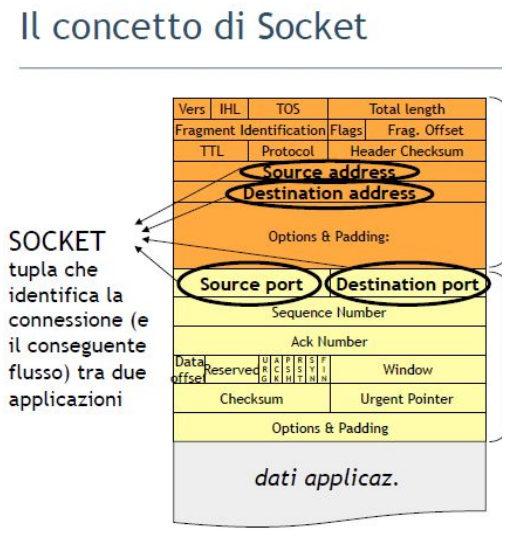
- Fornisce un canale di trasporto end-to-end privo di errori tra dei processi in esecuzione su host diversi
- Indirizza a livello applicativo permettendo di differenziare il traffico dei pacchetti introducendo il concetto di *PORTA*, un numero che identifica i processi che utilizzano il livello di trasporto
- In base al protocollo scelto (ce ne sono principalmente 2):
    - **TCP:** connection-oriented reliable e ordinato (pacchetti ritrasmessi se persi) controllo della congestione e controllo di flusso.
    - **UDP:** connectionless “best effort” (pacchetti ignorati se persi).

TCP (Transmission Control Protocol)

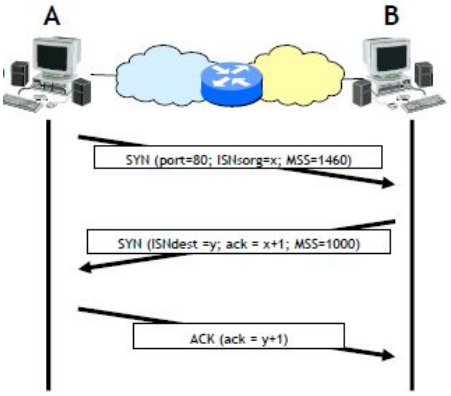
- **Header TCP**
    - **Sequence:** numero di sequenza del primo byte del payload
    - **ACK Number:** numero di sequenza del prossimo byte che si intende ricevere (ha validità se il segmento è un ACK)
    - **Offset [4 bit]:** lunghezza dell’header TCP, in multipli di 32 bit
    - **Reserved [6 bit]:** riservato per usi futuri
    - **Flags:**
        - **URG:** vale uno se vi sono dati urgenti; in questo caso il campo urgent pointer ha senso
        - **ACK:** vale uno se il segmento è un ACK valido; in questo caso l’ACK number contiene un numero valido
        - **PSH:** vale uno quando il trasmettitore intende usare il comando di PUSH;
        - **RST:** resetta la connessione senza un tear down esplicito
        - **SYN:** usato durante il setup per comunicare i numeri di sequenza iniziale
        - **FIN:** usato per la chiusura esplicita di una connessione



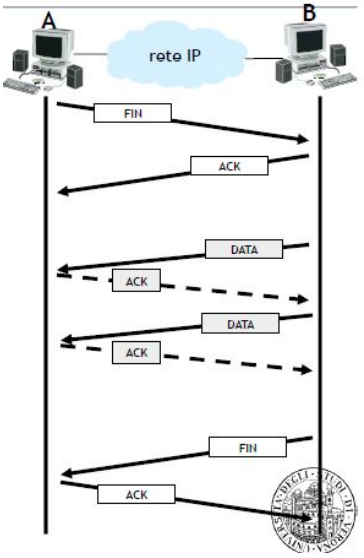
    - **Window [16 bit]:** ampiezza della finestra di ricezione (comunicato dalla destinazione alla sorgente)
    - **Checksum [16 bit]:** risultato di un calcolo che serve per sapere se il segmento corrente contiene errori nel campo dati
    - **Urgent pointer [16 bit]:** indica che il ricevente deve iniziare a leggere il campo dati a partire dal numero di byte specificato. Viene usato se si inviano comandi che danno inizio ad eventi asincroni "urgenti"
    - **Options and Padding [lunghezza variabile]:** riempimento (fino a multipli di 32 bit) e campi opzionali come ad esempio durante il setup per comunicare il MSS
    - **Data:** i dati provenienti dall’applicazione
- Utilizza le **SOCKET** per instaurare una connessione tra 2 host; viene usata sia per inviare e che per ricevere; vengono create usando la tupla $\mathbf{IP_S{:}PORT_S + IP_D{:}PORT_D}$
- Il numero di porta può essere
    - **STATICO** (well known): sono porte usate sempre e solo per delle applicazioni ben specifiche;
      [es. HTTP → 80, FTP → 20 e 21, HTTPS → 443]
    - **DINAMICO** (ephemeral): sono porte assegnate, tra quelle disponibili, dal sistema operativo quando l’applicazione ne richiede una



- Le porte non devono essere necessariamente uguali e vengono usate per mux/demux dei pacchetti ai diversi processi applicativi
- **Instaurazione della connessione**
  - "three-way handshake": SYN, SYN ACK, ACK
  - la stazione che richiede la connessione (A) invia un segmento di SYN
    - numero di porta dell'applicazione su B
    - Initial Sequence Number (ISN_A)
    - (opz.) MSS
  - la stazione che riceve la richiesta (B) risponde con un segmento SYN ACK
    - ISN_B
    - riscontro (ACK) per ISN_A
    - (opz.) MSS
  - la stazione A invia un ACK per ISN_B alla stazione B, riscontrando quindi il segmento SYN ACK della stazione B

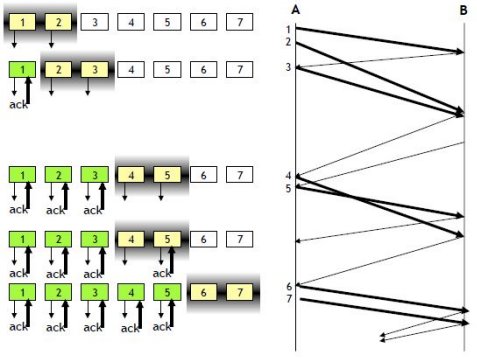


- **MSS (Maximum Segment Size)**: parametro del campo Options dell'header, indica la dimensione massima del campo Data accettabile; di default MSS = 536 byte
- **Terminazione della connessione**
  - la stazione che non ha più dati da trasmettere e decide di chiudere la connessione invia un segmento FIN
  - la stazione che riceve il segmento FIN invia un ACK e indica all'applicazione che la comunicazione è stata chiusa nella direzione entrante **(half close)**
  - nell'altra il trasferimento dati può continuare; per chiudere completamente la connessione, la procedura di half close deve avvenire anche nell'altra direzione



- I segmenti SYN, SYN ACK, FIN, FIN ACK e ACK sono segmenti vuoti (solo header) con i corrispettivi flag settati a 1
- **Ritrasmissioni** in caso di:
  - Errore individuato con il campo checksum
  - Perdita di pacchetti individuato quando manca un ACK
- **Timeout** per ritrasmissione
  - **RTT** (Round Trip Time): tempo tra l'invio di un pacchetto e l'arrivo del suo ACK;
    $\mathbf{SRTT_{now} = \alpha * SRTT_{prec} + (1 - \alpha) * RTT_{misurato}}$ (tipicamente α = 0.9), SRTT è l'RTT Stimato
  - **RTO** (Retransmission Time Out): tempo massimo che può passare dall'invio di un pacchetto all'arrivo del suo ACK, dopodiché viene considerato perso;
    dipende da:
    - distanza da sorgente a destinazione
    - condizioni di rete
    - disponibilità destinazione

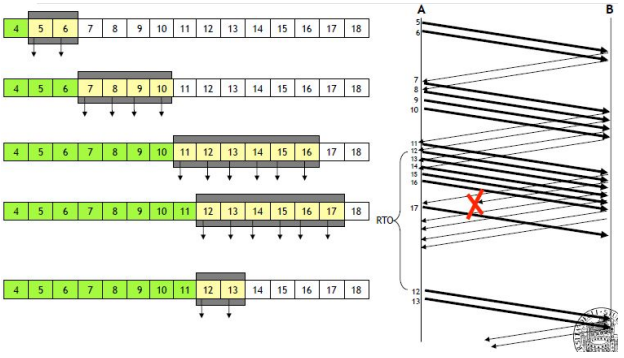
    viene calcolato in fase di instaurazione e durante la trasmissione dei dati;
    RTO = β *SRTT (tipicamente β=2) alla **prima perdita**, poi
    RTO = β * RTO in caso di ulteriori perdite dello STESSO PACCHETTO



- **Controllo di flusso:** azione preventiva che limita l'immissione di dati nella rete
  - **Sliding window** (intervallo dei pacchetti trasmessi): trasmettere N pacchetti consecutivamente senza dover aspettare l'ACK del primo per inviare i successivi. Tutti gli ACK ricevuti spostano la finestra di 1, permettendo di inviare pacchetti successivi.

- Finestra troppo piccola -> sottoutilizzo banda
- Finestra troppo grande -> tornano gli ACK prima rispetto alla fine dell'invio dei pacchetti
- I pacchetti di un host sono ordinati in base al Sequence Number a partire dal ISN di quel host
- In caso di perdita del pacchetto, allo scadere del RTO per il corrispondente ACK, il TCP lo ritrasmette con diverse logiche:
  - Go-back-N, la finestra torna indietro al pacchetto perso e si ritrasmette tutto (utile nel caso di ambienti con perdite multiple contemporanee)
  - Selective Repeat, ritrasmetto solo il segmento perso, più complesso da implementare ma più efficiente

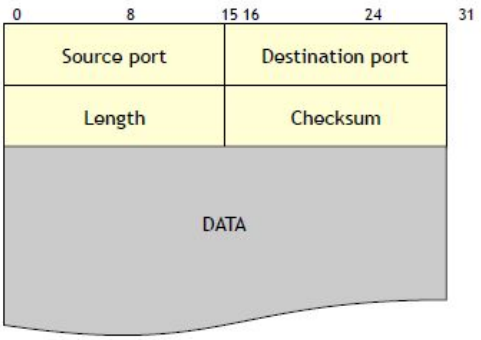
- **Controllo di congestione:** reazione alla congestione di rete
  - A causa di buffer limitati, la rete può non riuscire a mandare i messaggi a destinazione; il TCP si adatta e diminuisce il flusso dei pacchetti inviati e la dimensione della finestra di trasmissione
  - **CONGESTION WINDOW (CWND):** sliding window a dimensione variabile in base alla congestione della rete; viene modificata in base a degli algoritmi, i due fondamentali sono: Slow Start e Congestion Avoidance



  - **SLOW START**:
    - $CWND_{new}$ = CWND + #ACK, andamento esponenziale (la CWND viene duplicata)
  - **CONGESTION AVOIDANCE**:
    - $CWND_{new}$ = CWND + #ACK/CWND, andamento lineare (la CWND aumenta di 1)
  - Questi due algoritmi molto rudimentali vanno utilizzati in un algoritmo più complesso che mescola le caratteristiche dei due per ottenere un miglior controllo della congestione:
    - **Receive Window (RCVWND)**, dimensione della finestra di ricezione dichiarata dalla destinazione nel campo Data; è il massimo che la CWND può raggiungere
    - **Slow Start Threshold (SSTHRESH)**, dimensione della CWND dopo la quale si comincia ad usare il Congestion Avoidance anziché lo Slow Start
    - Poi si procede seguendo l'algoritmo:

```
CWND = 1 segmento
while (PacketsToSend != 0) {
      try {
            #ACK = sendPackets(CWND) // possono mancare ACK
            CWND_new = (CWND < SSTHRESH)
                  ? min(CWND + #ACK, SSTHRESH, RCVWND) // Slow Start
                  : min(CWND + #ACK/CWND, RCVWND) // Congestion Avoidance
            RTO = 2*RTT
            PacketsToSend -= #ACK
      } catch (MissingACK) {
            Wait(RTO)
            RTO = 2*RTO // algoritmo dato dal testo dell'esercizio
            SSTHRESH = max(CWND/2, 2) // minimo 2 segmenti
            CWND_new = 1
      }
}
```

UDP (User Datagram Protocol)



- **“Best effort”**
- **No frills** (senza fronzoli, ogni servizio non essenziale è stato rimosso per migliorare la velocità)
- **I segmenti** spediti con UDP possono essere **persi** o consegnati disordinati all’applicazione (no controllo di flusso)
- No handshake tra mittente e destinatario (connectionless)
- Ogni segmento viene **gestito indipendentemente** dagli altri
- Usato principalmente per **streaming, applicazioni multimediali, servizio DNS e SNMP** (configurazione, gestione e supervisione di dispositivi di rete).
    - Molto utilizzato per il **poco ritardo** nella connessione, per la leggerezza del suo header e per la **mancanza del controllo di congestione**
- L’integrità del pacchetto viene gestita con un checksum
    - campo che permette al destinatario di **verificare l’integrità** dei dati ricevuti. Il mittente **separa** i contenuti del segmento in **sequenze da 16 bit che somma in complemento a 1**. Dopodiché inserisce il valore ottenuto nel checksum ed invia il pacchetto. Il destinatario **riesegue** le somme e verifica se il **checksum calcolato è uguale a quello inserito nel pacchetto**. Questo metodo, tuttavia, **non garantisce totalmente l’integrità** dei dati anche se i due valori coincidono

Livello 3: Rete

Il livello di rete fornisce principalmente due funzioni:

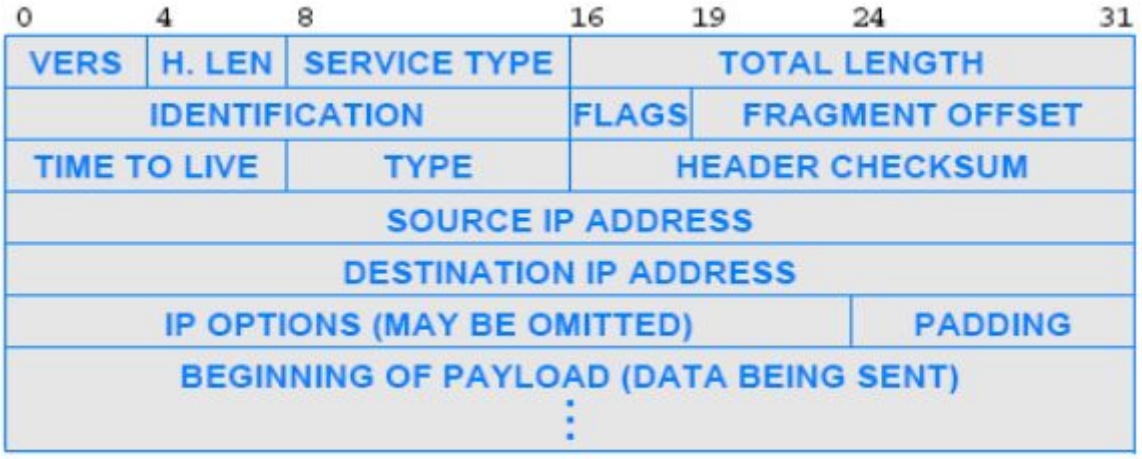
- **Routing**
    - determinare il percorso che i pacchetti devono seguire
- **Forwarding**
    - trasferimento di pacchetti dalla porta di input alla porta di output corretta

Un router deve poter implementare:

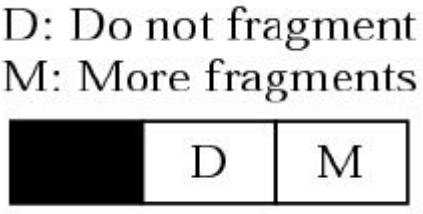
- **Buffering**
    - velocità di arrivo dei pacchetti > velocità di trasmissione
- **Scheduling**
    - usato per decidere l’ordine di trasmissione dei datagrammi nel buffer

DATAGRAMMA IP

Termine usato nello stack TCP/IP per i pacchetti, è composto da un header (20 - 60 bytes) e da un payload (1 - 64K bytes)



- **Header IP**
  - **VERS**: Versione del protocollo
  - **H.LEN**: dimensione dell'header/4 (dimensione header vuoto 20 byte / 4 = 5)
  - **SERVICE TYPE**: indica la priorità di un datagramma (usato raramente)
  - **TOTAL LENGTH**: dimensione totale in byte (header + payload)
  - **IDENTIFICATION**: id del datagramma, usato per ricostruire i pacchetti frammentati
  - **FLAGS [3 bit]**: indica se il datagramma è un frammento ed (eventualmente) se è l'ultimo:
    - **RESERVED**
    - **D** (**DON'T FRAGMENT**): se settato a 1 indica che il pacchetto non può essere frammentato. Se il pacchetto non può essere inoltrato senza frammentazione, viene scartato.
    - **M** (**MORE FRAGMENT**); se settato a 0 indica che è l'ultimo frammento di un pacchetto (o se questo non è frammentato). Se settato a 1 indica che è un frammento del pacchetto.

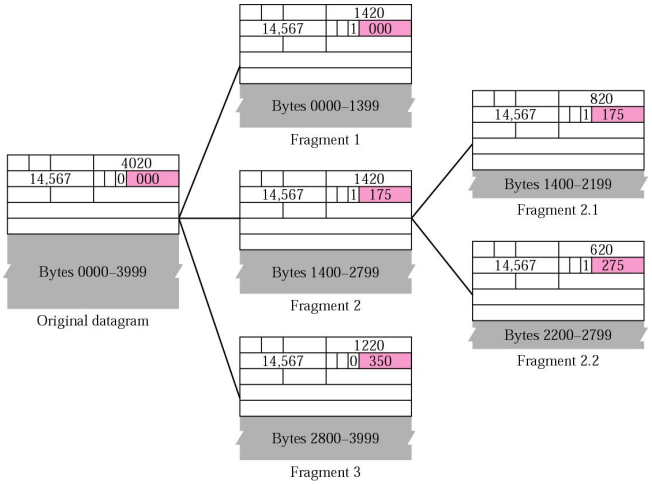


  - **FRAGMENT OFFSET**: offset del frammento rispetto al datagramma originale (val * 8)
  - **TTL**: inizializzato dalla sorgente, questo campo viene decrementato dopo ciascun hop. Quando il TTL di un pacchetto arriva a 0, il datagramma viene scartato.
  - **TYPE/PROTOCOL**: codice associato al protocollo utilizzato nel campo dati del pacchetto IP
  - **HEADER CHECKSUM**: campo utilizzato per verificare eventuali errori nell'header; ricalcolato ad ogni hop se non corrisponde, comporta lo scarto del pacchetto
  - **SOURCE IP ADDRESS**: ip sorgente
  - **DESTINATION IP ADDRESS**: ip destinazione
  - **IP OPTIONS**: campi opzionali
- **Maximum Transmission Unit**: massima **lunghezza** delle trame che la tecnologia **HW** riesce a **trasmettere**
  - Un router può essere connesso con reti aventi **MTU diverso**
  - Se un router deve mandare un datagramma ad un host e la **dimensione del datagramma** è **maggiore** della **MTU** del collegamento router-host, il dispositivo di rete deve frammentare il pacchetto.
    - **frammentazione** di un pacchetto
    - invio dei **frammenti** in modo **indipendente**
    - il pacchetto è un **datagramma IP**

Protocol	*MTU*
Hyperchannel	65,535
Token Ring (16 Mbps)	17,914
Token Ring (4 Mbps)	4,464
FDDI	4,352
Ethernet	1,500
X.25	576
PPP	296

        - header copiato quasi di peso
        - campo **FLAG** e **FRAGMENT OFFSET** settati
    - **Usato** (con la **dimensione dell'header**) per calcolare la **dimensione massima** dei dati che possono essere inviati in ciascun frammento e il **numero** di **frammenti** da inviare

- **Riassemblaggio** del datagramma a **destinazione**
    - **meno dati** da far memorizzare al **router** (non importa se è un frammento o un pacchetto intero)
    - permette di avere un **percorso dinamico** anche tra frammenti
- **Perdita di frammenti**
    - Si inizia il **riassemblaggio** solo quando **tutti i frammenti** sono presenti
        - **Buffer** perchè i frammenti probabilmente subiranno ritardi diversi
        - Tempo di storage limitato da un **timer** (attivato all'**arrivo del primo frammento**)
    - Se tutti i frammenti **arrivano** prima dello scadere del timer
        - **assemblaggio** completato
    - Se il **timer scade** e mancano ancora frammenti
        - I frammenti nel buffer vengono **scartati**
    - La **sorgente non ha idea** della eventuale **frammentazione** subita dal suo pacchetto
        - **impossibile ritrasmettere** frammenti persi
            - **o tutto o niente**
        - se la sorgente ritrasmette l'intero pacchetto, non è detto che questo venga frammentato come la volta precedente



INDIRIZZAMENTO (ADDRESSING)

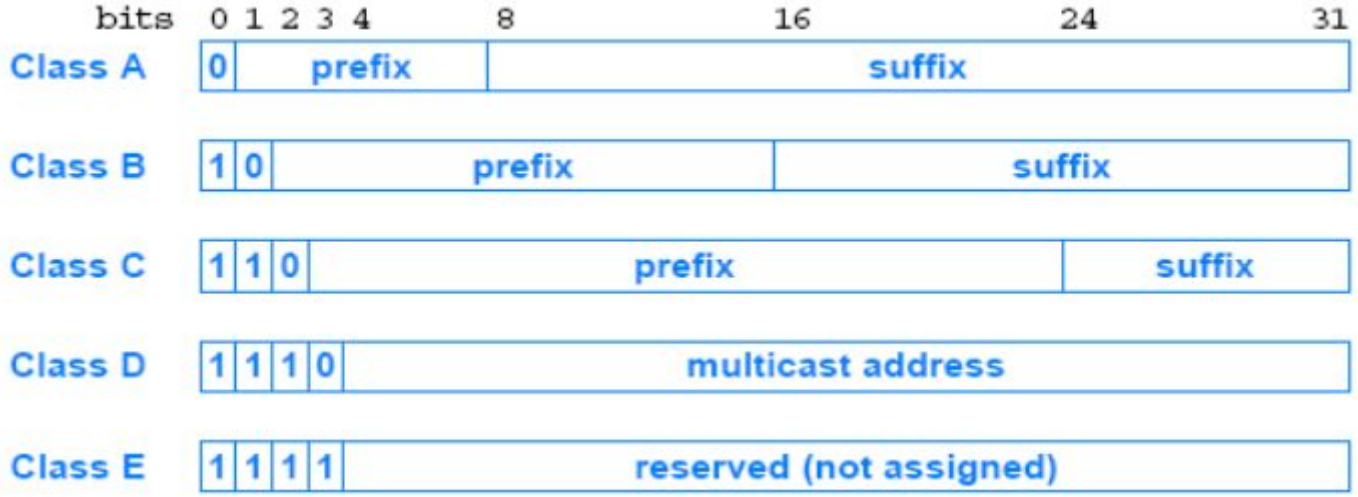
- Tutti gli host devono avere un sistema di indirizzamento **comune** necessario per **routing** e **forwarding**
- Ciascun indirizzo deve essere **unico**
- Gli **indirizzi IP** sono assegnati da un **protocollo** in SW
- **Indirizzi IP**: numero **univoco** di **32 bit**
    - rappresentato con la **dotted decimal notation**
    - dividere i 32 bit in **gruppi da 8**
    - separarli con un **punto**
    - rappresentare le sezioni con un **numero decimale**
    - da 0.0.0.0 a 255.255.255.255

  Quando un **host** vuole **comunicare** con un altro host deve inserire nel pacchetto il **suo IP** e quello della **destinazione**

  Composto da due parti:
    - **Prefisso** o **NetID**: numero identificativo e unico della rete in Internet
        - Gestita **globalmente**

          ICANN (Internet Corporation for Assigned Names and Numbers) delega dei **registrars** che forniscono un blocco di indirizzi a ogni **ISP**. Questi forniscono gli indirizzi ai loro utenti
    - **Suffisso** o **HostID**: numero identificativo e unico dell'host all'interno della rete
        - Gestita **localmente**

  Classi di indirizzi (soluzione originale, **Classful**)
    - I primi 4 bit determinavano la classe di indirizzamento di appartenenza



Address Class	Bits In Prefix	Maximum Number of Networks	Bits In Suffix	Maximum Number Of Hosts Per Network
A	7	128	24	16777216
B	14	16384	16	65536
C	21	2097152	8	256

Con l'espansione di Internet il sistema Classful **non** era più **applicabile** (alto **spreco** di indirizzi) e per questo sono stati proposti due sistemi dove **NetID** e **HostID** sono **variabili**:

- **Subnet**
- **Classless**

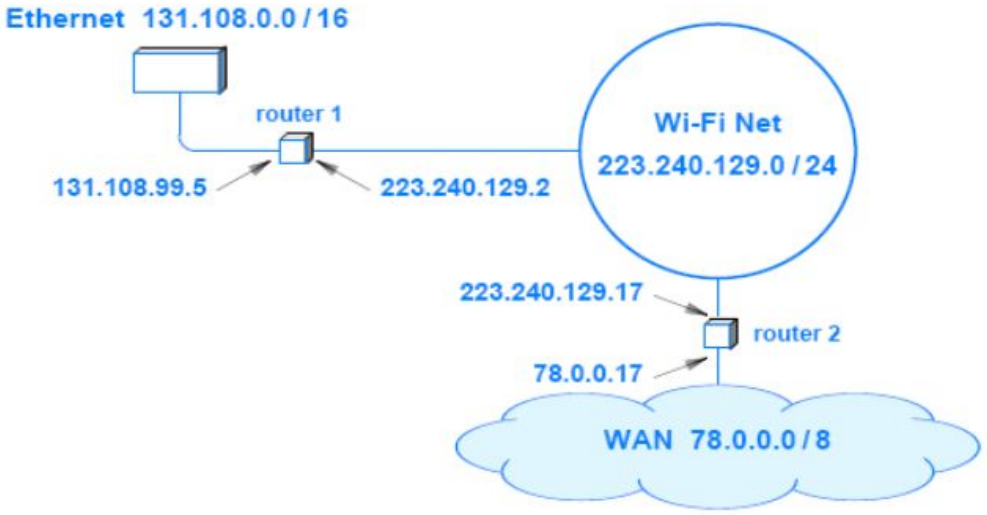
Con l'introduzione del sistema di indirizzamento **Classless** è stato aggiunto anche l'**informazione** relativa alla **separazione** tra suffisso e prefisso, chiamata **maschera d'indirizzo** o **maschera di subnet**

- Una maschera è un **valore a 32 bit** in cui sono **posti ad 1** tutti i bit necessari a raggiungere la **lunghezza** del **prefisso**
- Questo aumenta notevolmente il **processing** in quanto le decisioni di **routing** sono prese solo **analizzando** il **prefisso** tramite apposito HW
- Un router deve quindi **mantenere** in memoria i **prefissi di rete** e le rispettive **maschere**. All'arrivo di un pacchetto il router confronta l'**indirizzo di destinazione** con i **prefissi** che ha in memoria. Questo processo però non si applica a tutti i 32 bit, infatti per stabilire dove effettuare il forward viene fatta una **AND bit a bit tra maschera e IP da raggiungere**, **confrontando** poi il risultato con i **prefissi** in memoria.
- **CIDR** (Classless Inter-Domain Routing) è un sistema di **notazione** pensato per **facilitare la gestione da parte degli utenti** in cui viene specificata la dimensione del prefisso. Si presenta nella forma *d.d.d.d/m* dove *d* è il valore in **decimale** e *m* la **maschera** di rete
- Gli svantaggi principali dell'indirizzamento Classless sono il più **alto numero** di **informazioni** da salvare nei router e la **difficoltà** di **lettura** degli indirizzi.

Esistono alcuni **indirizzi IP speciali**

- L'indirizzo IP con **host address a zero** è l'**indirizzo di rete** e viene usato per **identificare** in modo univoco la **rete**

- L'indirizzo IP con **host address a uno** è l'**indirizzo di broadcast** e viene usato per **consegnare** un pacchetto a **tutti gli host** della rete di destinazione
- L'indirizzo IP **tutto a uno** è l'**indirizzo di limited broadcast** e viene usato per **consegnare** un pacchetto a **tutti gli host** della rete della sorgente. Usato durante la startup del sistema.
- L'indirizzo IP **tutto a zero** è l'**indirizzo rappresentante l'host stesso** e viene usato per **indicare** un indirizzo **non corretto**.
- L'indirizzo IP **127.0.0.0/8** è l'**indirizzo di loopback** e viene usato per **testare una comunicazione** tra applicativi di rete. I programmi, invece di venire eseguiti su host diversi, vengono **fatti girare sullo stesso host** e rispondere a indirizzi appartenenti a 127.0.0.0/8 (non è importante a che suffisso). Il livello **IP** reagisce **passando** di nuovo attraverso lo **stack dell'altra applicazione**.



PROTOCOLLI DI SUPPORTO

- **ICMP** (Internet Control Message Protocol): **complementare** al protocollo **IP**
  - IP dipende da ICMP per segnalare errori
  - ICMP usa IP per trasportare i messaggi di errore

  ICMP gestisce due tipi di messaggi:
  - messaggi per **segnalare errori**
    - Ad esempio **Time Exceeded** e **Destination Unreachable**
  - messaggi per **ottenere informazioni**
    - Ad esempio **Echo Request**/**Reply** usati da **ping**

  Quando un router deve mandare un **messaggio ICMP**, lo inserisce nel **payload** di un **datagramma** IP. Il pacchetto viene poi gestito **normalmente**, con la sola **eccezione** che se un **ICMP di errore causa un errore**, non viene inviato **nessun** messaggio.

- **DHCP** (Dynamic Host Configuration Protocol)
  La configurazione iniziale di un router deve prevedere **l'indirizzo** di ciascuna interfaccia di rete, il software di **protocollo** da usare e i valori iniziali delle **tabelle di forwarding**
  La **configurazione** degli **host** che viene eseguita **all'avvio** è nota come **bootstrapping** e viene gestita da un protocollo, il **BOOTP** (Bootstrap Protocol). Il DHCP viene usato per fornire la maggior parte delle informazioni necessarie.
  - Una volta esistevano diversi meccanismi per ottenere i parametri, tra cui **RARP** (Reverse Address Resolution Protocol) che **otteneva un IP da un server**
  - Veniva usato anche **ICMP** nelle opzioni **Address Mask Request** e **Router Discovery** per ottenere la **maschera** di rete e l'**indirizzo** di un router
  - Ora **tutti** questi meccanismi sono **implementati** nel **protocollo DHCP** e permettono un approccio chiamato "**plug-and-play networking**"

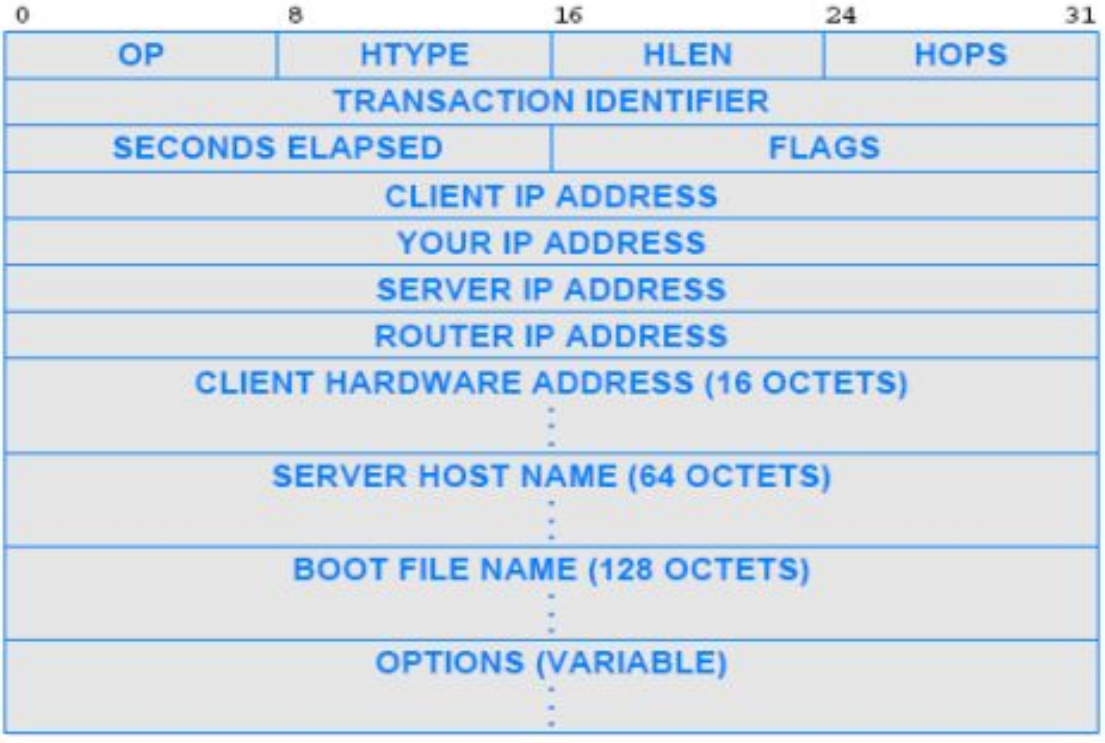
Procedura di **bootstrapping**:

- **Accensione** di un host
- Invio di una **richiesta DHCP** in **broadcast** (**DHCP DISCOVER**)
- Il server "**offre**" un **indirizzo IP** al client (**DHCP OFFER**)
  - L'indirizzo offerto può essere **permanente** o **dinamico**
  - Gli indirizzi dinamici sono assegnati solo per un **predeterminato** periodo di tempo (**lease**) per permettere al DHCP di "**tornare in possesso**" di tale indirizzo
  - Quando il lease scade il **server** può **assegnare** l'indirizzo ad un **nuovo host** ma, in media, l'**host richiede un'estensione** del periodo per poter continuare a lavorare ininterrottamente
- L'host **accetta** l'indirizzo o ne **richiede un altro**
- L'host **memorizza** l'IP del server **DHCP**

Il protocollo è stato progettato in modo che **perdite** e **duplicazioni** di pacchetti **non configurino** la rete in **modo errato**. Per questo se un **host non riceve risposta reinvia** la richiesta. Se invece un **host riceve un duplicato**, **ignora** il secondo pacchetto. Inoltre il **DHCP** usa tecniche per **evitare** la ricezione di **messaggi** in **contemporanea**

Formato dei **messaggi**:

- **OP**: indica se si tratta di una Request o di una Response
- **HTYPE**: indica il tipo di hardware della rete
- **HLEN**: lunghezza dell'indirizzo hardware
- **FLAGS**: indica se l'host può ricevere messaggi di broadcast o risposte dirette
- **HOPS**: indica a quanti server rigirare la richiesta
- **TRANSACTION IDENTIFIER**: contiene un valore per gli host per capire se la risposta è riferita alla sua richiesta
- **SECONDS ELAPSED**: indica quanti secondi sono passati dall'avvio dell'host
- **YOUR IP ADDRESS**: riempito dal serve per fornire l'IP al client
- **SERVER IP ADDRESS** e **SERVER HOST NAME**: fornisce informazioni sull'host
- **ROUTER IP ADDRESS**: fornisce l'IP del router di default



ALGORITMI DI ROUTING

- Quando un host vuole consegnare un pacchetto a un altro host
    - se i due host sono nella stessa rete abbiamo la **consegna diretta** (indirizzo fisico ottenuto attraverso ARP)
    - se i due host non sono nella stessa rete abbiamo la **consegna indiretta** (messaggio consegnato ad un router che lo inoltra fino all'host)
- Routing: processo di scoperta del cammino da una sorgente ad ogni destinazione della rete
    - Ogni router gestisce una propria tabella di routing
        - Per ogni destinazione indica su quale interfaccia girare il pacchetto
        - Basata sullo stato globale della rete
            - Questo è il problema principale
            - Troppo grande per essere gestito
            - Difficile da reperire
            - Molto dinamico
        - Per lavorare efficientemente ogni router deve
            - minimizzare la tabella di routing
            - minimizzare il numero e la frequenza dei messaggi di controllo
            - avere una tabella robusta
            - utilizzare sempre il cammino ottimo
    - Esistono diverse implementazioni tecnologiche del routing
        - Routing centralizzato o distribuito
            - centralizzato semplice ma facile che si guasti/congestioni
        - Scambio informazioni globale o locale
            - trasmettere informazioni globali è costoso
        - Routing statico o dinamico
            - statico richiede poca banda in quanto le righe della tabella di routing sono inserite manualmente e non vengono più modificate
            - Routing dinamico ottenuto tramite un protocollo di propagazione dei cammini che permette un aggiornamento dinamico della tabella di routing
        - Routing stocastico o deterministico
            - Stocastico favorisce il load balancing
        - Cammini singoli o multipli
        - State-dependent o state-indipendent
            - Modifica del percorso in base allo stato della rete
    - Dato un grafo pesato, si dice cammino minimo tra A e B il cammino con il costo più basso
        - L'obiettivo di un algoritmo di routing è quello di trovare il cammino minimo
- Distance Vector: algoritmo di routing

  PREMESSE e PREPARAZIONE
    - Vale la seguente proprietà: una porzione di cammino minimo è anche il cammino minimo tra i nodi che delimitano tale porzione
        - Per la proprietà, possiamo affermare quindi che $D(i, j) = c(i, k) + D(k, j)$
    - Dato un grafo pesato, vengono denominati:
        - $c(i, k)$: il costo del collegamento diretto tra i e k
        - $D(k, j)$: il costo del cammino minimo tra k e j
        - $D(i, j)$: il costo del cammino minimo tra i e j
    - Viene tenuta una tabella rappresentante le destinazioni, il next hop ed i costi, dove
        - **$D(i, i) = 0$**, il costo tra un host e sé stesso è 0
        - **$c(i, k)$ = costo**, il costo del collegamento
        - **$D(i,$ *altre_destinazioni_non_dirette*$) = \infty$**, visto come irraggiungibile

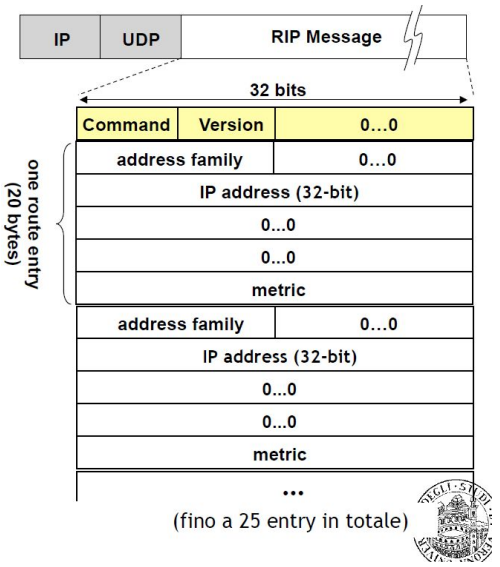
ALGORITMO

```
//INVIO
while (true) {
      if (qualche_costo_cambia){
            for (Neighbor n : this.neighbors){ //per ogni vicino
                  //invia la tua DV attuale senza la colonna interface
                  send(this.getTable().getDestinations(),
                        this.getTable().getWeight(), n)
            }
      }
}
//RICEZIONE
DistanceVector dvN = recive() //ricevi la DV del vicino n
for (RecordDV line : dvN){ //controlla la DV
      //peso minore
      if(c(this, n) + D(n, line.destination()) < D(this, line.destination())){
      //sostituisci nella tabella aggiornando il peso e hop
            replaceInDV(n, line.getWeight + c(this, n))
      }
}
```

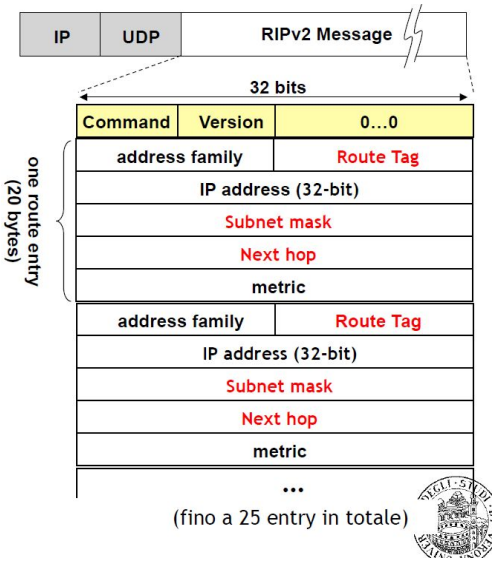
IN BREVE

- Periodicamente ogni nodo invia ai suoi vicini il DV
    - Le buone notizie viaggiano velocemente
    - Le cattive notizie viaggiano lentamente
- Quando un nodo riceve un DV aggiorna il proprio DV con il valore minimo
- Il tutto procede in maniera **asincrona**
- Tutto questo converge al cammino minimo
- Ho un problema di **counting to infinity**
    - se un router *X* si disconnette dopo un po' di tempo rischio di entrare in un circolo vizioso:
        - il router adiacente *Y* mette il collegamento a ∞ e inoltra la DV
        - il router adiacente a *Y* chiamato *Z* ha nel frattempo inoltrato la sua DV
            - *Z* raggiunge *X* attraverso *Y*
        - *Y* vede che *Z* raggiunge *X* in *n* hop ed, essendo minore di ∞ aggiorna la sua tabella e la reinvia
        - *Z* riceve la DV aggiornata e vede che il costo del collegamento tra *Y* e *X* è aumentato. Non sapendo che *Y* vede in lui il next hop e anzi, vedendo *Y* come passo per raggiungere *X*, aggiorna la sua DV col nuovo valore
            - La distanza cresce continuamente nonostante *X* non sia più raggiungibile
    - La soluzione è quella di limitare il costo del collegamento a 15 (TTL 15)
    - Split Horizon
        - semplice
            - ciascun nodo, quando invia il DV ad un vicino *k*, omette le destinazioni che hanno k come next hop
        - con poisoned reverse
            - ciascun nodo, quando invia il DV ad un vicino *k*, imposta la distanza per le destinazioni che hanno *k* come next hop a ∞

- **RIP** (Routing Information Protocol)
  - Protocollo intra - dominio semplice
    - Basato su **UDP**, porta **520**
  - Basato su **Distance Vector**
    - **Convergenza lenta**
    - Funzionamento con **reti di limitata dimensione**
    - **Semplice** da implementare
    - Semplice da gestire
    - **Uso diffuso**
    - Funziona solo su reti con la stessa maschera
  - Metrica basata sul conteggio degli hop
    - **15** considerato **come ∞**
    - Possono essere presenti hop con valore > 1
  - Vengono mantenuti **3 timer**:
    - **25-30s:** Aggiornamento periodico, usato per inviare messaggi di aggiornamento
    - **120s:** garbage collection, le entry non valide vengono marcate come infinito.
    - **180s:** se un entry non viene aggiornata entro questo timer, essa non viene considerata valida.
  - Le tabelle di routing di RIP sono gestiti da processi a livello applicativo.
  - Tipi di messaggi:
    - **Request:** generati da **router appena avviati**, **risposta diretta**.
    - **Response:** generati da router che inviano **messaggi di aggiornamento**, o **risposte a query specifiche**, viene **aggiornata** la **tabella di routing**.



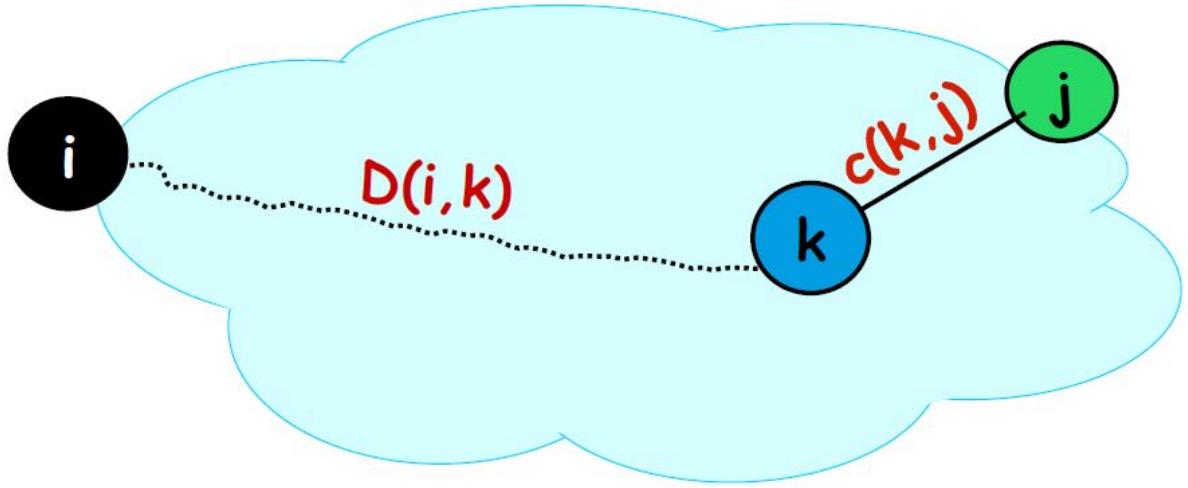
- **RIPv2**
  - Differenze rispetto al protocollo RIP:
    - **Autenticazione:** solo i messaggi dei router autorizzati vengono considerati attendibili (**autenticazione MD5 hash/password**)
    - Uso esplicito delle subnet.
    - **Utilizzo del multicast** invece del broadcast per l'invio dei messaggi.
    - **Interoperatività** con il protocollo **RIPv1**.



- **Limitazioni** di **RIPv1** e **RIPv2**:
  - Destinazioni con metriche **superiori a 15 non sono raggiungibili**.
  - Una metrica semplice comporta tabelle di routing sub-ottime.
  - Se non vi è autenticazione, i router accettano aggiornamenti RIP da chiunque: **router mal configurati possono fare danni**.

Algoritmi Link State

A differenza degli algoritmi di tipo Distance Vector, puntano a ottenere una visibilità globale della rete. Ciascun nodo ottiene tutti i link state c(*,*), e viene poi applicato l'algoritmo di Dijkstra sul grafo ottenuto.



PREMESSE e PREPARAZIONE

- Dato un nodo *i*:
  - Ad ogni iterazione l'algoritmo trova una nuova destinazione e il cammino minimo per raggiungerla
    - con una iterazione esploro i nodi di grado successivo
  - L'algoritmo di Dijkstra mantiene due insiemi:
    - insieme *N* dei nodi per cui è stato trovato il cammino minimo
    - insieme *M* che contiene tutti gli altri nodi e le relative distanze da *i*
  - Per ogni nodo *k* vengono mantenuti
    - D(*i*, *k*)
    - p(*k*): il predecessore del nodo *k* nel cammino minimo da *i*
- Devo inizializzare:
  - D(*i*,*i*) = 0, p(*i*) = *i*
    - Inizializzazione di sè stesso
  - D(*i*,*k*) = c(*i*,*k*), p(*k*) = *i*
    - $\forall$ *k* | *k* è un vicino di *i*
  - D(*i*,*s*) = ∞, p(*s*) = *UNKNOWN*
    - $\forall$ *s* | *s* non è un vicino di *i*
  - N = {i}, next - hop(*i*) = *i*
  - M = {*j* | *j* ≠ *i*}

ALGORITMO

```
//INVIO
Node i
Map<Node, Integer> M = {...} // nodo, costo da i
Map<Node, Integer> N = {...} // nodo, costo da i
Map<Node, Node> nextHop // destinazione, nodo precedente
while (true) {
      Node j = M.getMinDistance() // prendi nodo di M con la minima distanza da i
      // se ci sono più nodi con la distanza minima uguale, scegli casualmente
      N.put(j, getDistance(j, i)) // nodo e la distanza tra j e i
      M.remove(j)
      // sposto il nodo da M a N
      if(p(j) != i){
            nextHop.put(j, p(j)) // se non è un vicino
      }else{
            nextHop.put(j, j) // se p(j) = i
      }
      for(Node node : M.get(n).getNeighbors){
            if(D(i,k) < D(i,j) + c(j,k)){
                  D(i,k) = D(i,j) + c(j,k)
                  // p(k) = j
                  nextHop.put(node, n)
            }
      }
}
```

CONSIDERAZIONI

- Complessità con *n* nodi: O(n^2) ma esistono implementazioni che raggiungono O(n log n)
- Assegnazione del costo dei collegamenti
  - costo basso = alta probabilità di far parte del cammino minimo
  - Metriche statiche: non considerano il traffico
    - Numero di hop
  - Metriche dinamiche: difficili da gestire
    - Occupazione delle code o analisi del ritardo
  - Metriche quasi - statiche
    - Ricalcolo periodico delle metriche statiche per valutare il traffico generale

LINK STATE	***DISTANCE VECTOR***
● Le informazioni sulla topologia sono inviate su tutta la rete ● il miglior cammino viene calcolato da ciascun router localmente e determina il next - hop ● funziona solo se la metrica è condivisa ed uniforme ● Con *n* nodi e *E* link, O(*nE*) messaggi inviati ● Velocità di convergenza O(n log n) ● in caso di guasti ogni nodo calcola la propria tabella di routing, limitando i danni ● OSPF	● Ciascun router ha una visione limitata della topologia della rete ● Data una destinazione è possibile individuare il miglior next - hop che determina il cammino end - to - end ● Non richiede metriche uniformi tra tutti i router ● Scambio di messaggi solo tra vicini diretti ● Velocità di convergenza variabile e soggetta a routing loops e counting to infinity ● In caso di guasti si ha una propagazione degli errori in tutta la rete ● RIP

Routing gerarchico

Internet **non è una rete piatta**:

- Più di **200 milioni** di possibili destinazioni
  - Problema di **scalabilità**.
  - Non è possibile memorizzare tutte le possibili destinazioni in una tabella.
  - Scambi di tabelle così grandi saturerebbero i collegamenti.
- **Autonomia amministrativa**
  - Ogni amministratore di rete ha il **controllo del routing della propria rete.**

SOLUZIONE: Routing Gerarchico

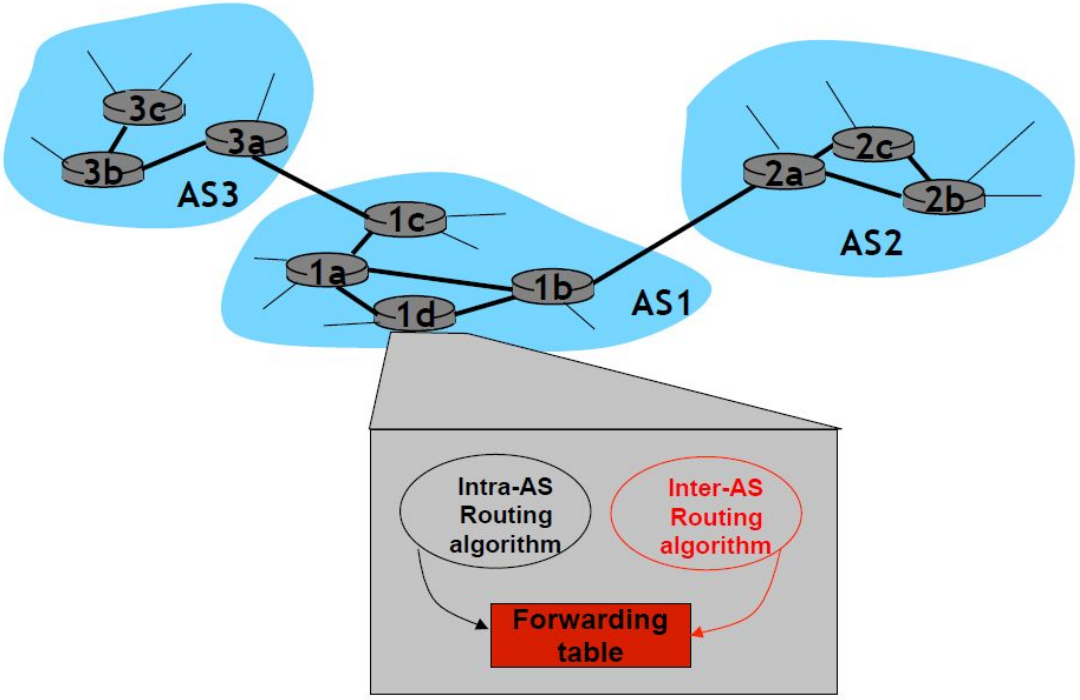
Organizzare la rete in gerarchie permette di risolvere il problema della scalabilità, riducendo la dimensione delle tabelle di indirizzamento e il traffico necessario per gli update delle stesse.

I **router** sono organizzati in **regioni** ("**A**utonomous **S**ystems" = "**AS**"):

- **I router** di uno **stesso AS** usano lo **stesso** protocollo di **routing**.
- Router di **AS differenti** possono usare **differenti protocolli**.
- Tabelle intra-AS per le destinazioni interne.

Collegate tra di loro da **router di bordo** (gateway):

- Sono i router che mettono in **collegamento diversi AS.**
- Tabelle inter-AS & intra-AS per le destinazioni esterne.



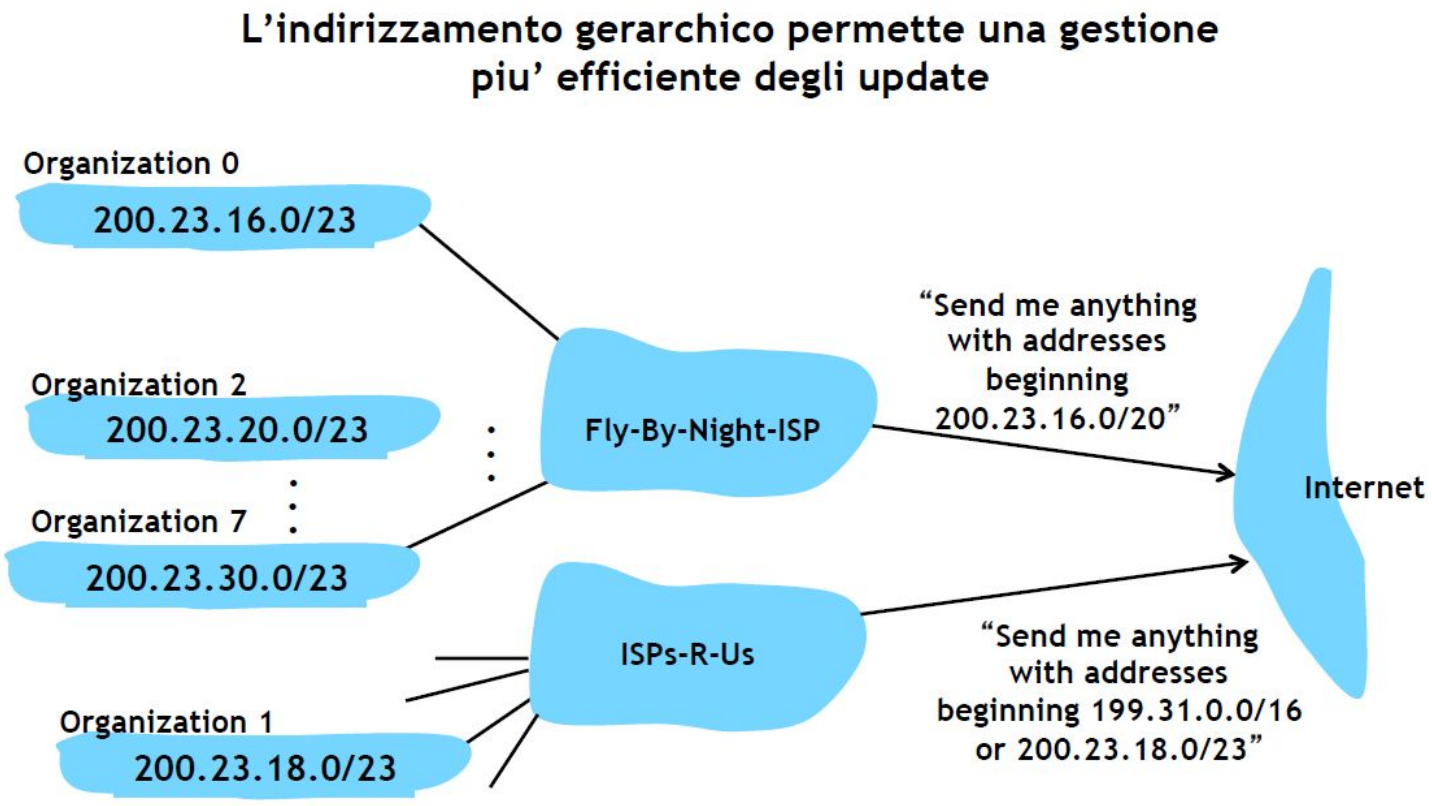
Algoritmi di routing **intra-AS**, **IGP** (**I**nterior **G**ateway **P**rotocol):

- **RIP**: Routing Information Protocol
- **OSPF**: Open Shortest Path First
- **IGRP**: Interior Gateway Routing Protocol (**Cisco**, proprietario)

L'algoritmo standard di routing **inter-AS**, è il **BGP** (**B**order **G**ateway **P**rotocol), e oltre a **rendere la rete visibile** e **indirizzabile** alle altre reti, esso mette a disposizione agli AS i mezzi per:

- Ottenere **informazioni di raggiungibilità** di altre reti dagli AS vicini.
- **Propagare le informazioni di raggiungibilità** ai router interni degli AS.
- Impostare delle **policy di indirizzamento**.
- Ottenere il **cammino migliore verso le altre reti** in base alle informazioni di raggiungibilità degli **AS vicini**.

	Intra-AS	*Inter-AS*
Policy	Singolo dominio amministrativo, policy uniforme	L'amministratore può controllare come il traffico viene instradato attraverso la propria rete
Prestazioni	Può essere focalizzato sulle prestazioni (cammini minimi)	Le policy possono essere piu' importanti delle prestazioni



Indirizzamento privato: NAT

Perchè il NAT (Network Address Translation)? Il problema della carenza di indirizzi IPv4 e del costo elevato degli archi di indirizzamento ha spinto lo sviluppo di una tecnica di indirizzamento basato sugli indirizzi privati.

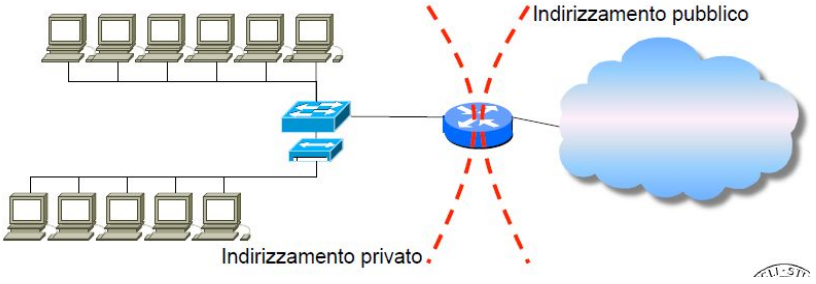
Sono stati definiti dall'IETF una alcuni range di indirizzi da utilizzare in ambito privato:

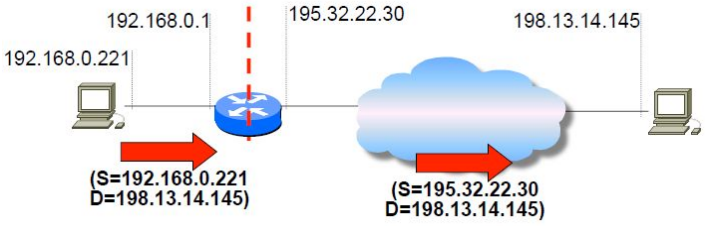
- Non sono indirizzabili pubblicamente (private addresses o non-routable addresses)
- Se un router pubblico riceve un indirizzo destinato a un indirizzo privato, viene segnalato un errore.

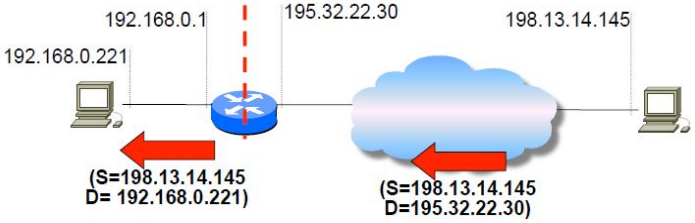
Indirizzi privati

Prefisso	Indirizzo iniziale	Indirizzo finale
10.0.0.0/8	10.0.0.0	10.255.255.255
172.16.0.0/12	172.16.0.0	172.31.255.255
192.168.0.0/16	192.168.0.0	192.168.255.255
169.254.0.0/16	169.254.0.0	169.254.255.255

- Le reti con un solo punto di accesso a internet posso sfruttare l'indirizzamento privato.
- Per poter ricevere pacchetti all'interno della rete privata, è necessario aggiungere un'ulteriore funzionalità sul router di bordo che collega la rete a Internet.
    - Al router di confine viene assegnato un indirizzo IP pubblico sull'interfaccia verso la rete esterna.
    - Il router compie un lavoro di traduzione sostituendo:
        - L'indirizzo sorgente di un pacchetto con il proprio IP pubblico.
        - L'indirizzo destinazione con l'indirizzo privato dell'host corretto.



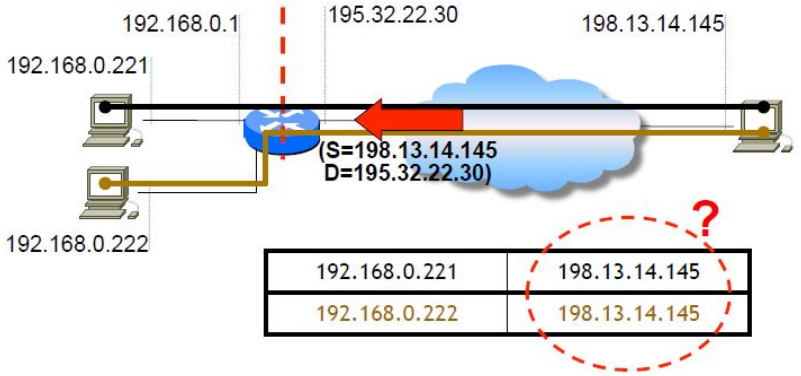




Network Address Translation Table

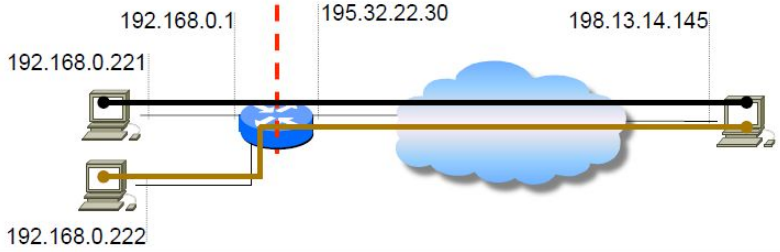
- Il router NAT mantiene al suo interno una tabella di record con il mapping tra indirizzo privato sorgente della comunicazione ed indirizzo pubblico destinazione della comunicazione.
- I record della tabella possono essere impostati manualmente, o dinamicamente in base ai datagrammi uscenti verso una determinata destinazione (cancellati tramite timeout).

- Un NAT basato unicamente sull'indirizzo non permette a diversi host della rete privata di collegarsi a uno stesso host pubblico!



Port Mapped NAT

- Il router NAT agisce da gateway di livello 4, effettuando una traduzione di IP e porta



Priv. Addr	Priv. Port	Ext. Addr	Ext. Port	NATport	Prot. 4
192.168.0.221	21023	198.13.14.145	80	14001	TCP
192.168.0.222	13444	198.13.14.145	80	14002	TCP

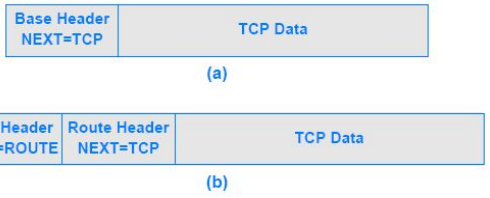
IPv6

Gli indirizzi IPv4 sono finiti a causa della crescita esponenziale della rete (2^{32} = 4.294.967.296 indirizzi pubblici, molto meno rispetto ai dispositivi connessi a internet) e l'introduzione di un nuovo protocollo di rete può anche aumentare le funzionalità (load balancing delle richieste a server con lo stesso dominio, ora gestito da DNS; multicast, supporto real-time, ecc...)

- *Formato datagrammi*
  - contiene una serie di headers, il primo è quello di base per l'IPv6, i successivi sono detti Extension Headers (possono anche non esserci); poi c'è il payload
    - gli extension header sono un trovato intelligente dato che fanno risparmiare spazio (i datagrammi usano solo gli header che sono necessari) e permettono una gestione modulare delle funzionalità (basta aggiungere un nuovo extension header anziché cambiare tutto lo standard, come sarebbe richiesto dal IPv4)
  - gli header hanno dimensioni variabili e possono anche essere più grandi del payload

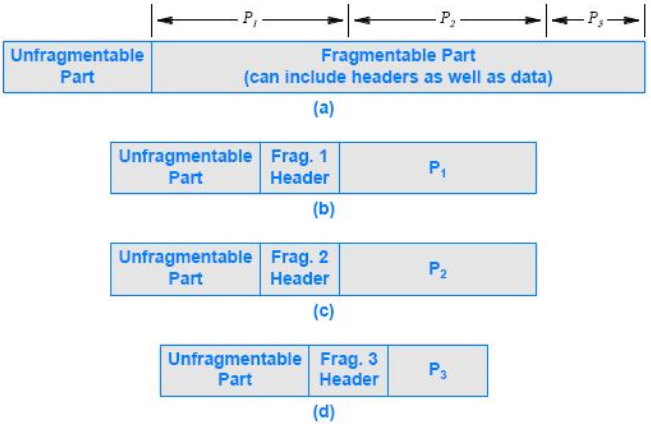
- *Formato dell'header di base (40 byte)*
  - **VERS [4 bit]** (Versione: 6)
  - **TRAFFIC CLASS [8 bit]**: specifica la classe di traffico in base al tipo di traffico; rientra nel framework "*differentiated services*" per specificare i requisiti che la rete dovrebbe soddisfare. Esempi:

    - in caso di traffico interattivo (movimenti del mouse) è la sorgente potrebbe specificare una classe con basso ritardo
    - in caso di audio real-time è la sorgente potrebbe specificare un cammino con un jitter basso
  - **FLOW LABEL [20 bit]**: usato per associare un datagramma con un cammino specifico
  - **PAYLOAD LENGTH [16 bit]**: specifica la dimensione del payload (dati trasportati dopo l'header); in IPv4 c'era un campo "total length" che invece includeva l'header
  - **NEXT HEADER [8 bit]**: campo con un doppio significato: specifica il tipo di informazione che segue l'header corrente
    - Se il datagramma include un extension header indica il tipo di extension header
    - Se non ci sono extension header specifica il tipo di dati trasportato nel payload
  - **HOP LIMIT [8 bit]**: corrisponde al campo TTL di IPv4
  - **ADDRESSES [128 bit ognuno]**: indirizzo IPv6 di sorgente e destinazione



- *Formato extension header*
  - **NEXT HEADER**: come nel Base Header
  - **HEADER LENGTH**: indica la lunghezza di tutto l'header
  - **OPTIONS**: di dimensione variabile, contiene diverse informazioni in base al tipo di header

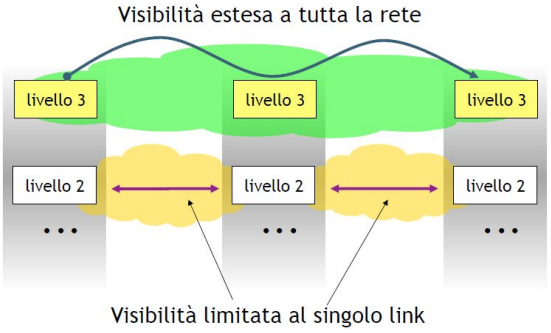
- Frammentazione, Riassemblaggio e Path MTU
  - simile a IPv4
    - viene copiato il prefisso non frammentabile (Base Header + header per il routing) in tutti i frammenti
    - la dimensione del payload viene modificata in base alla MTU della rete da attraversare
  - diversamente da IPv4
    - non ci sono campi nel Base Header per la frammentazione
    - è necessario un extension header con le informazioni sulla frammentazione
    - la presenza stessa di un "fragment header" indica che si tratta di un frammento



- Indirizzamento
  - segue la logica CIDR, dividendo prefisso e suffisso arbitrariamente con una maschera
  - introduce un concetto di gerarchia multilivello (ISP, azienda, stanza, ...)
  - indirizzi speciali
    - **unicast**: singolo host; un datagramma per questo indirizzo passa dal cammino minimo
    - **multicast**: insieme di host che può cambiare in qualunque momento; una copia del datagramma viene consegnata ad ogni membro
    - **anycast**: insieme di host con lo stesso prefisso; il datagramma viene consegnato a un qualsiasi membro dell'insieme (es. il più vicino)
  - la notazione degli indirizzi da 128 bit è esadecimale e molto difficile da gestire a mano
  - un indirizzo è diviso in massimo 8 gruppi di 16 bit separati da " : "
  - le sequenze di 0 si possono comprimere (es. FF0C:0:0:0:0:0:0:B1 → FF0C::B1)
  - il mapping degli indirizzi IPv4 esistenti in indirizzi IPv6 è fatto ponendo a zero i primi 96 bit

Livello 2: Data link

L'obiettivo del livello 2 è fornire al livello 3 di due host vicini (fisicamente connesse, sia cavo che wireless) un canale di comunicazione il più possibile affidabile. I pacchetti in questo livello vengono chiamate **trame**.



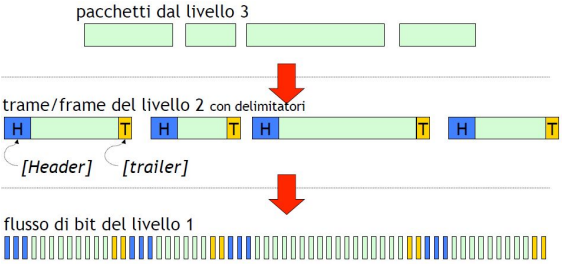
Ci sono alcune **problematiche**:

- **Errori di trasmissione** sorgente-destinatario.
- Gestione della **velocità di trasmissione**.
- **Ritardo** di propagazione nel canale.

Può offrire diverse **tipologie di servizi:**

- Connectionless senza ACK (la più usata nelle LAN).
- Connectionless con ACK.
- Connection-oriented con ACK.

Offre le seguenti **funzionalità**:



- **Framing** (delimitazione delle trame)
  - **Delimitare pacchetti** (livello 3) di lunghezza variabile **in trame** (livello 2) di lunghezza variabile.
  - Il livello 1 non sa distinguere le trame, gestisce sono i bit.
  - Sistemi non sincronizzati, **una macchina non sa quando un'altra sta trasmettendo**.
  - Le reti di telecomunicazione **non danno garanzie** sul **rispetto** delle **caratteristiche temporali** delle informazioni trasmesse e **i dati possono** essere **espansi e ridotti** nel canale generando problemi di ricezione.
  - **Modalità di framing:**
    - Character count: un campo nel header del frame indica la lunghezza del frame stesso.
    - Bit stuffing: la trama viene fatta iniziare e finire con il **byte di flag** , ovvero **01111110**
      Se la trama contiene già al suo interno questo pattern, si esegue questo algoritmo:
      - Se la sorgente incontra 5 bit "1" consecutivi, aggiunge uno "0"
        011111x ⟶ 0111110x
      - Se la destinazione incontra 5 bit "1" consecutivi, toglie uno "0"
        0111110x ⟶ 011111x
- **Rilevazione/gestione errori:**
  - Il livello 1 offre un canale **non privo di errori** (sul singolo bit, replicazione e perdita dei bit).
  - Viene usato il campo **checksum** nel header, ottenuto dal payload della trama, per il controllo degli errori, NON per la correzione.

- **Controllo di flusso** (gestione velocità di trasmissione):
  - Velocità di **invio non sempre uguale alla** velocità di **ricezione**.
  - Gestito tramite **feedback da parte destinazione**, che indica tramite flag e campi se permettere alla sorgente di continuare la trasmissione e la quantità di informazioni che la destinazione è in grado di gestire.

  Il controllo di flusso viene demandato ai livelli superiori nelle reti TCP/IP.

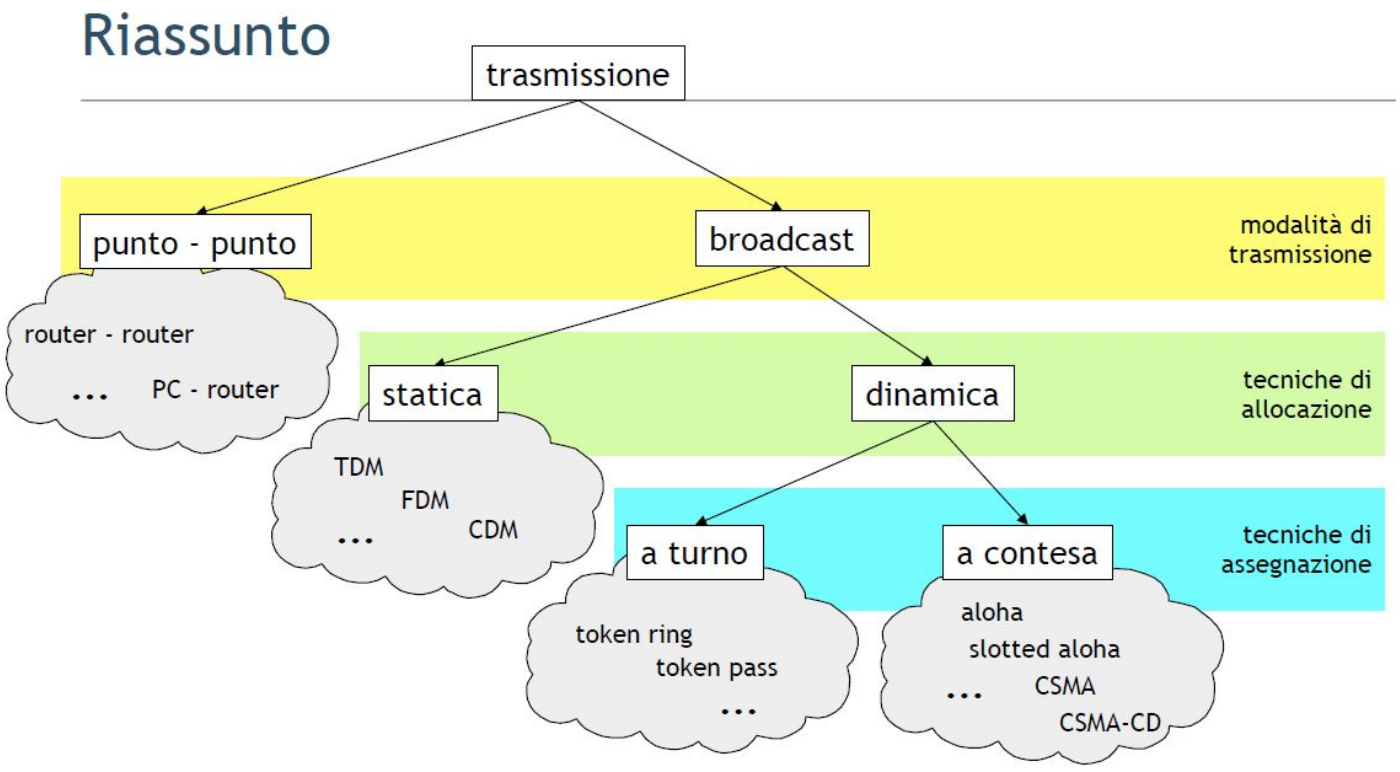
Sottolivello MAC (Medium Access Control)

Necessario poiché il livello 2 ha a che fare con problemi legati al mezzo fisico, che può essere:

- Dedicato (reti punto-punto).
- Condiviso (reti broadcast), il quale introduce problemi di accesso al mezzo e quindi di concorrenza alla trasmissione e turnazioni.

Il sottolivello MAC (livello 2 “LOW”) viene usato per gestire le politiche di accesso al mezzo condiviso, il framing e il controllo di errore, mentre il livello 2 “HIGH” gestisce in particolare il controllo di flusso.

- Allocazione del canale
  - **statica**: il mezzo viene partizionato (in base al tempo o alla frequenza d’onda di trasmissione) e ogni porzione viene distribuita alle diverse sorgenti.
    - Frequency Division Multiplexing
    - Time Division Multiplexing
    + Semplice implementazione
    - Il canale trasmissivo viene sprecato in momenti di silenzio e traffico discontinuo
    - Funzionano bene solo con pochi utenti
  - **dinamica**: il mezzo viene assegnato a chi ne fa richiesta di volta in volta (a turno o a contesa)
    - **a turno**
      viene distribuito il permesso di trasmettere
      + Nessuna ritrasmissioni a causa di collisioni
      - C’è bisogno di un sistema di gestione della turnazione (overhead di gestione)
    - **a contesa (più utilizzati)**
      ognuno prova a trasmettere e si gestiscono gli errori
      + Semplice implementazione di rete (tutto gestito dagli host)
      - Molte ritrasmissioni a causa di collisioni

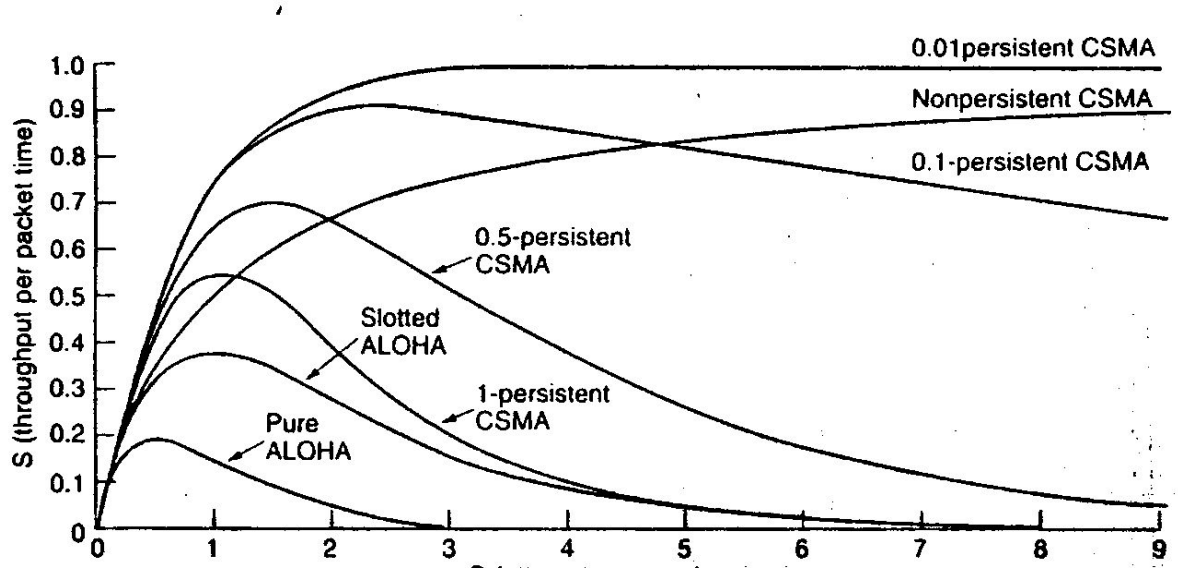


Protocolli di accesso multiplo

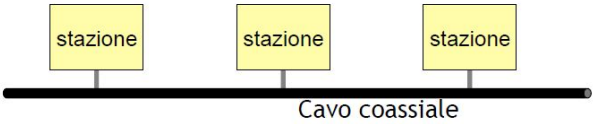
Principali algoritmi ad accesso a contesa a un mezzo di trasmissione condiviso sono:

- **ALOHA** (Definito all’università delle Hawaii)
  - Pure ALOHA:
    - La sorgente **trasmette una trama ogni qualvolta vi sono dati da inviare**, senza preoccuparsi di nient’altro (detto *continuous time*)
    - La sorgente rileva una eventuale **collisione in ricezione**, e nel caso si verifichi **si attende un tempo casuale** prima di ritentare la trasmissione.
    - Periodo di vulnerabilità: **2T**, dove **T** è il tempo necessario per trasmettere una trama, **19%** del mezzo trasmissivo sfruttato.

    - Slotted ALOHA:
        - Basato su ipotesi di slotted time (tempo suddiviso ad intervalli discreti).
        - Esattamente come ALOHA, ma la **trasmissione può iniziare solamente a intervalli discreti**, che necessita quindi di **sincronizzazione tra stazioni**.
        - Periodo di vulnerabilità ridotto a **T**, **37%** del mezzo trasmissivo sfruttato.
- **CSMA** (**C**arrier **S**ense **M**ultiple **A**ccess protocols):
    - CSMA:
        - Le **stazioni sono in grado di ascoltare il canale prima di iniziare a trasmettere**, per verificare se c'è una trasmissione in corso.
        - **Diverse varianti** disponibili in caso di **canale occupato**:
            - **persistent (1-persistent)**: nel momento in cui si libera il canale, si trasmette.
            - **non-persistent (0-persistent)**: si rimanda la trasmissione a un tempo casuale.
            - **p-persistent**: il canale viene diviso in intervalli uguali al tempo di vulnerabilità.
                a. se il canale è libero, si trasmette con probabilità p.
                b. se non si è deciso di trasmettere si attende un intervallo di tempo si torna al punto a.
                c. se si è deciso di trasmettere e c'è una collisione, si attende un intervallo di tempo e si torna al punto a.
        - In caso di **collisione** si comporta come **ALOHA**.
    - CSMA-CD (rilevazione delle collisioni)
        - **Rilevazione delle collisioni**: se viene rilevata una collisione, **si smette di trasmettere immediatamente**, quindi non si spreca tempo a trasmettere trame corrotte.
        - Per avvisare tutti della collisione, si trasmette una particolare **frequenza di jamming**.



LAN Estese



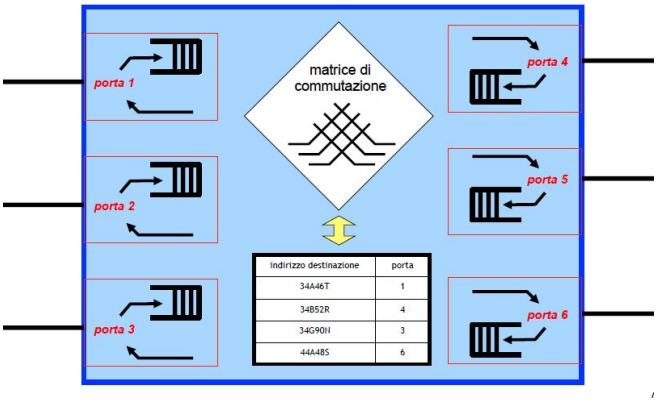
Il concetto di LAN (Local Area Network) nasce dalla necessità economica e di necessità di collegare diverse stazioni a un mezzo di comunicazione condiviso.
Il segmento di cavo coassiale non può essere troppo lungo, causa attenuazione e conformazione dell'edificio, come quindi estendere una rete locale?

Dominio di collisione: parte della rete nella quale se due stazioni trasmettono dati insieme, il segnale ricevuto dalle stazioni risulta danneggiato. Può essere spezzato e diviso in sotto-domini da apparati di rete di livello 2 come Bridge e Switch

Dominio di broadcast: parte di rete raggiunta da una trama con un indirizzo broadcast (a livello 2)

Esistono 3 tipi di apparati, in ordine crescente di complessità:

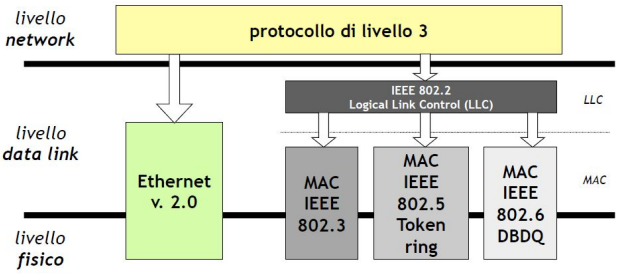
- **Hub (e Repeater) [livello 1]:** replica pari pari il segnale ricevuto su TUTTE altre porte e lo amplifica;
  - Repeater: solo 2 porte e non possono essercene più di 4 in cascata tra due stazioni
  - Hub: può avere molte più porte
  - Estende troppo il dominio di collisione (dato che si trova al livello 1)
- **Bridge [livello 2]:** usa una tabella per registrare quali stazioni fanno parte di ciascun segmento e trasmette sull'altro segmento solo se la destinazione non è nello stesso segmento, anziché mandare tutto in broadcast come un Repeater
  - Spezza il dominio di collisione
  - Supporta solo 2 segmenti
- **Switch [livello 2]:** è un Bridge multiporta; spesso ogni porta è collegata direttamente ad una stazione, così facendo:
  - Supporta conversazioni contemporanee multiple tra più segmenti
  - Ogni nodo ha un accesso dedicato (e quindi non ci sono collisioni sul segmento, dato che non esiste il segmento)

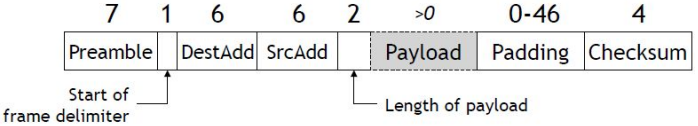


Protocolli livello 2 (incapsulamento IP)

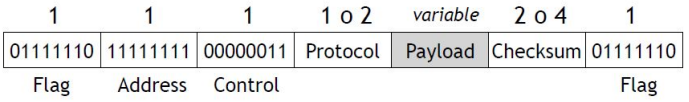
A differenza del livello 3, che vede la comunicazione come end-to-end, a livello 2 la comunicazione è vista hop-by-hop e ogni hop attraverso il quale passa il livello 3 può usare diversi protocolli livello 2; è quindi il livello 3 il primo ad unificare la visione della rete.
Esistono quindi svariate modalità di incapsulamento IP in base al contesto nel quale ci si trova:

- per l'accesso (host-router)
  - **Ethernet** (10 Mbps) e **IEEE 802.3** (famiglia di sistemi CSMA/CD 1-persistent da 1-10 Mbps)
    - usato nelle LAN
    - economico (facile installazione e manutenzione)
    - gestisce direttamente il livello 1
    - in generale poche collisioni (2-3% dei pacchetti collide una volta, 0.0001% collide più di una volta)
    - in caso di collisione, si calcola l'istante di ritrasmissione usando un algoritmo **binary exponential backoff**
      - dopo *i* collisioni, l'host attende per ritrasmettere *Rand([0, 1, ..., 2^i - 1])*
      - dopo 10 collisioni il tempo di attesa è limitato all'intervallo *Rand([0, 1, ..., 1023])*
      - dopo 16 collisioni viene riportata una *failure* al sistema operativo
    - *Formato trama*
      - **Preambolo** [7 byte]: byte "01010101" utilizzato 7 volte per sincronizzare il ricevitore
      - **Start of frame** [1 byte]: flag di inizio della trama "11010101"
      - **Addresses** [6 byte]: indirizzi MAC destinazione e sorgente della trama
      - **Length** [2 byte]: lunghezza trama in byte (0-1500); se > 1500 indica Protocol Type
      - **Payload**: informazione trasmessa
      - **Checksum** [4 byte]: codice per rilevazione di errore
    - evoluzioni
      - Fast Ethernet (100 Mbps)
      - Gigabit Ethernet (1 Gbps), offre scalabilità con le versioni precedenti





- **PPP** (con modem o con ADSL), sia per accesso che per backbone
  - character oriented
  - character stuffing per il framing
  - identificazione degli errori
  - supporta vari protocolli di livello superiore (rete)
  - negoziazione dinamica degli indirizzi IP
  - autenticazione del "chiamante"
  - *Formato della trama*
    - **Flag** (1 byte): identifica inizio e fine della trama con il byte "01111110"
    - **Address** (1 byte): utilizzato in configurazione "tutti gli host" ("11111111")
    - **Control** (1 byte): valore predefinito "00000011" → unnumbered;
      di default non fornisce un servizio affidabile: richiesta di ritrasmissione e rimozione replicazioni sono lasciate ai livelli superiori [è disponibile un'estensione per reti con alto BER (wireless) ad un servizio connection oriented (RFC1663)]
    - **Protocol** (1 o 2 byte): identifica il tipo di livello di frame (LCP, NCP, IP, IPX, ...)
    - **Payload** (>0 byte): informazione trasmessa
    - **Checksum** (2 o 4 byte): identificazione dell'errore



  - con modem
    - usa la banda telefonica per inviare i segnali; è limitato al massimo a 56 Kbps
  - con xDSL (Digital Subscriber Line)
    - famiglie di tecnologie che usano la banda del doppino telefonica
    - si distinguono in simmetrici (es. HDSL) e asimmetrici (es. ADSL)

- per le backbone (router-router)
  - Frame Relay
  - ATM
  - soluzioni SDH

Protocolli di supporto: ARP (Address Resolution Protocol)

Per il forwarding, che utilizza gli indirizzi IP, bisogna sapere il MAC del next-hop al quale mandare il pacchetto, dato che gli indirizzi IP sono solo un'astrazione.
La traduzione da indirizzo IP di un host al suo indirizzo MAC è nota come "risoluzione degli indirizzi":

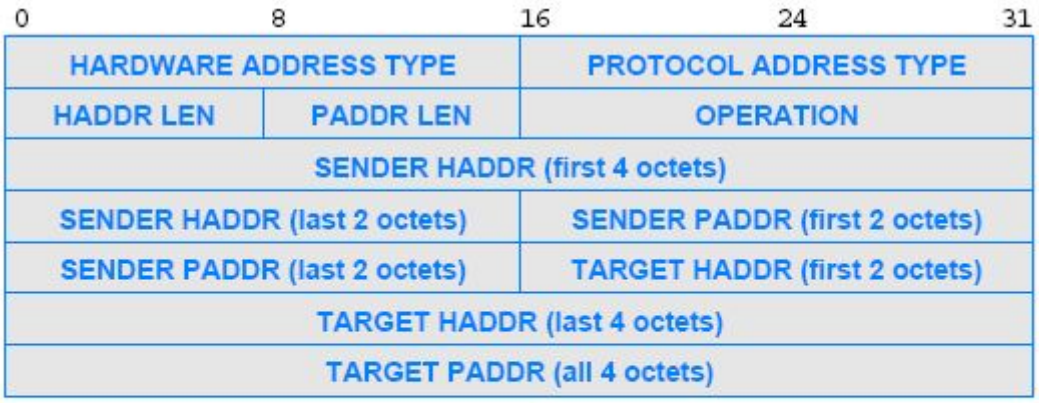
- **consegna indiretta**: l'host con l'IP dato non è nella rete; la richiesta viene fatta la router, il quale provvederà a cercare nelle altre reti
- **consegna diretta**: l'host con l'IP dato è nella rete e l'ottenimento del MAC è molto più semplice

Ci sono svariati algoritmi di risoluzione (in base al protocollo livello 3 usato e all'hardware coinvolto), in particolare per il protocollo IP su protocollo Ethernet viene usato ARP.

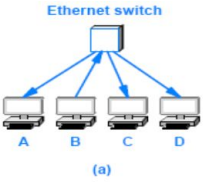
ARP

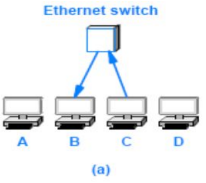
- è un protocollo generico livello 3 per la risoluzione di qualunque indirizzo livello 3 in un qualunque indirizzo livello 2 (non risolve solo gli IP in MAC), quindi non ha una dimensione di trama fissa (anche pensando alle tecnologie future)
- nella pratica, viene usato principalmente per associare indirizzi IP con indirizzi Ethernet (IEEE 802.3) o wireless LAN (IEEE 802.11)

- *Formato della trama*
  - **HARDWARE ADDRESS TYPE**: campo da 16-bit che specifica il tipo di indirizzo hardware utilizzato (in caso di Ethernet, tale valore è pari a 1)
  - **PROTOCOL ADDRESS TYPE**: campo da 16-bit che specifica il tipo di indirizzo del protocollo utilizzato (in caso di IPv4 il valore è 0x0800)
  - **HADDR LEN**: intero a 8-bit che specifica la dimensione in byte dell'indirizzo hardware (in caso di Ethernet, tale valore è pari a 6)
  - **PADDR LEN**: intero a 8-bit che specifica la dimensione in byte dell'indirizzo del protocollo (in caso di IPv4 il valore è 4)
  - **OPERATION**: campo a 16-bit che specifica se il messaggio è una "Request" (valore pari a 1) o una "Response" (valore pari a 2)
  - **SENDER HADDR**: indirizzo hardware della sorgente (lunghezza pari a HADDR LEN)
  - **SENDER PADDR**: indirizzo del protocollo della sorgente (lunghezza pari a PADDR LEN)
  - **TARGET HADDR**: indirizzo hardware del target (lunghezza pari a HADDR LEN)
  - **TARGET PADDR**: indirizzo del protocollo del target (lunghezza pari a PADDR LEN)



- Il funzionamento è il seguente:
  - Si supponga che B debba risolvere l'indirizzo IP di C
  - B invia in broadcast: "Ho bisogno dell'indirizzo MAC dell'host con il seguente indirizzo IP: C"
  - Il broadcast (con TTL) viaggia solo sulla rete locale
  - Il messaggio ARP di richiesta raggiunge tutti gli host della rete locale
  - Quando l'host C riceve la richiesta, risponde direttamente a B: "Sono l'host con indirizzo IP C e il mio MAC address è M"

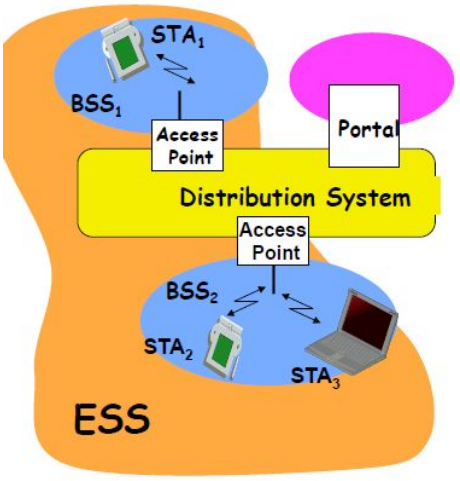




- ARP viene trasportato sul protocollo livello 2 dell'hardware in questione (es. Ethernet), il quale nel campo "type" della trama deve indicare il valore "0x806", che identifica i messaggi ARP. Così facendo, il router o il prossimo host, controllando il campo, capirà cosa farne del payload della trama, cioè lo manderà in broadcast fino a quando non verrà trovato il destinatario, il quale controllerà il campo "operation" del pacchetto ARP
- il software che implementa ARP solitamente salva in cache gli indirizzi risolti, pur non mantenendole per sempre (c'è bisogno, per esempio, di ricontrollare dopo un po', in caso l'host non sia più nella rete)

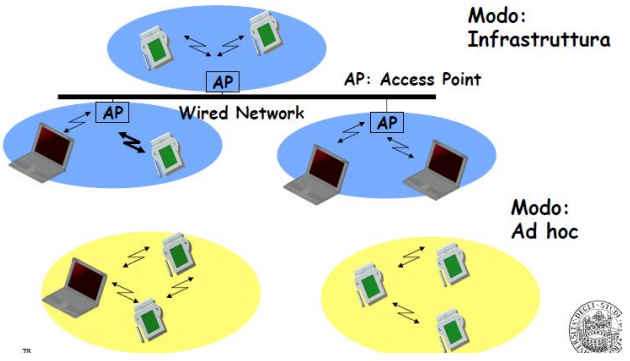
Reti 802.11 (livelli 1 e 2)

Nomenclatura



- **Station (STA)**: Terminale con capacità di accesso al mezzo wireless
- **Basic Service Set (BSS)**: Insieme di terminali che usano le stesse frequenze; isolato o collegato a un AP
- **Access Point (AP)**: Stazione integrata sia nella WLAN che nel "Distribution System"; in pratica è un bridge
- **Portal**: Bridge verso altre reti (wired)
- **Distribution System**: Rete LAN wired (backbone) per formare un'unica rete logica (ESS) partendo da diverse BSS
- **Extended Service Set (ESS)**: LAN logica formata da più BSS

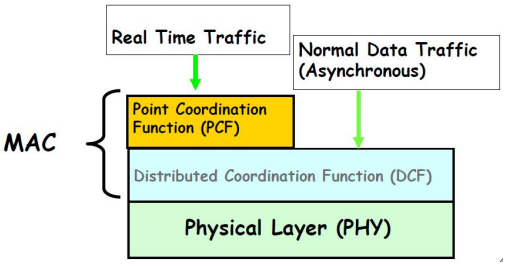
Famiglia degli standard 802.11



- 802.11a - 5GHz- Ratified in 1999
- 802.11b - 11Mb 2.4GHz- ratified in 1999
- 802.11d - Additional Regulatory Domains
- 802.11e - Quality of Service
- 802.11f - Inter-Access Point Protocol (IAPP)
- 802.11g - Higher Data rate (>20mBps) 2.4GHz
- 802.11h - Dynamic Frequency Selection and Transmit Power Control Mechanisms
- 802.11i - Authentication and Security
- 802.11n – Very High Bandwidth (10-20 times more)

Livello MAC

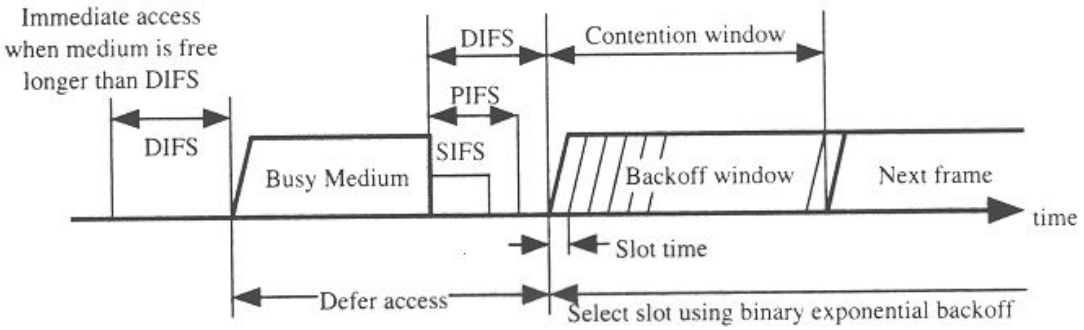
- è possibile usare il CSMA per le reti wireless applicando però alcune condizioni, dato che la banda disponibile è preziosa e le collisioni la sprecano e aumentano i ritardi. Bisogna introdurre dei concetti
  - **Distributed Coordination Function (DCF)**
    - basato su CSMA/CA
    - utilizza un algoritmo per la risoluzione delle contese per fornire accesso a tutti i tipi di traffico
    - il traffico ordinario si appoggia direttamente
  - **Point Coordination Function (PCF) [poco diffuso]**
    - supporta il traffico Real-Time
    - basato su polling controllato da un Centralized Point Coordinator
    - privo di collisioni
    - costruito su DCF, del quale sfrutta le funzionalità per fornire accesso agli utenti
  - DCF e PCF possono funzionare contemporaneamente nella stessa BSS, fornendo in alternanza periodi con e senza contesa



Time slot

- Il tempo è suddiviso in **slot**, ovvero l'unità temporale del sistema. La durata dipende da come è stato implementato il livello fisico (es. per 802.11b è 20µs)
- Le stazioni sono sincronizzate in base alla modalità, quindi il sistema è sincrono e mantenuto tale usando **trame di Beacon** (che scandiscono il tempo in broadcast)
  - "infrastructure" → con l'AP
  - "ad hoc" → tra di loro
- **Inter-frame space (IFS)**: intervallo di tempo tra la trasmissione di trame, stabilisce la priorità nell'accesso al canale; durata dipendente dall'implementazione

- SIFS: Short IFS [802.11b → 10μs]
  - separa la trasmissione di trame dello stesso “dialogo”
  - rappresenta la più alta priorità (usato per risposte immediate, es ACK, CTS, polling...)
  - durata dipendente da
    - tempo di propagazione
    - tempo di trasmissione
    - tempo di switch modalità TX e RX del trasmettitore
- PIFS: Point coordination IFS (> SIFS)
  - priorità media, per servizi real-time che usano PCF
  - usato dal controller centrale nello schema PCF durante il polling
  - PIFS = SIFS + un time-slot
- DIFS: Distributed IFS (> PIFS)
  - priorità più bassa, per il servizio di invio dati asincrono
  - usato dalle trame per l’invio asincrono con ritardo minimo nel caso di contesa del canale
  - DIFS = PIFS + 1 time slot = SIFS + 2 time slot
- EIFS: Extended IFS (> DIFS)

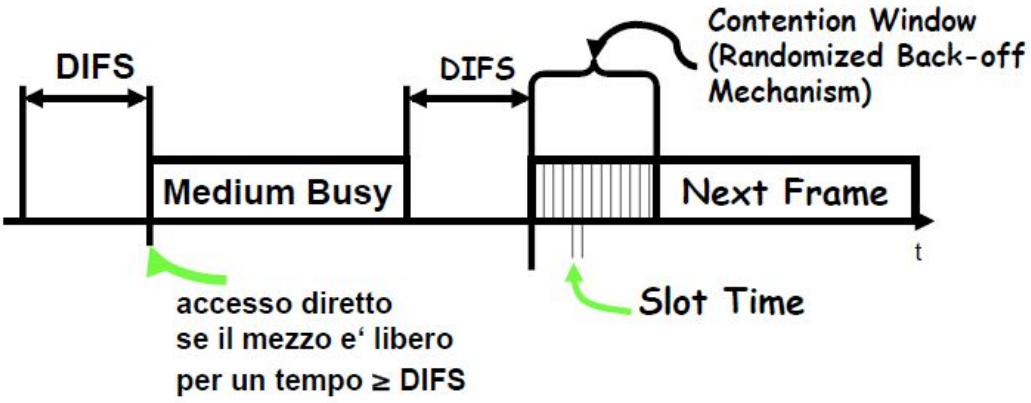


DCF con CSMA/CA

- una stazione che vuole trasmettere prima ascolta il canale
  - se il canale è **libero**, continua ad ascoltare per un tempo pari a DIFS, poi invia subito se il canale risulta ancora libero
  - se il canale è **occupato** (sia durante il DIFS che dall’inizio), continua a monitorare fino alla fine della trasmissione corrente, poi aspetta un altro DIFS
    - se il canale è ancora occupato, si ricomincia
    - se il canale è libero, estrai un numero casuale all’interno della Contention Window (**contatore di backoff**) e finché il canale è libero decrementalo:
      - se il contatore arriva a 0, trasmetti
      - se il canale diventa occupato prima dello 0, congela il contatore (che verrà usato dopo senza estrarne uno nuovo) e ricomincia
- il contatore viene estratto a caso tra i valori [0, CW - 1]
- CW = Contention Window: è un valore aggiornato ad ogni tentativo *i* di trasmissione:

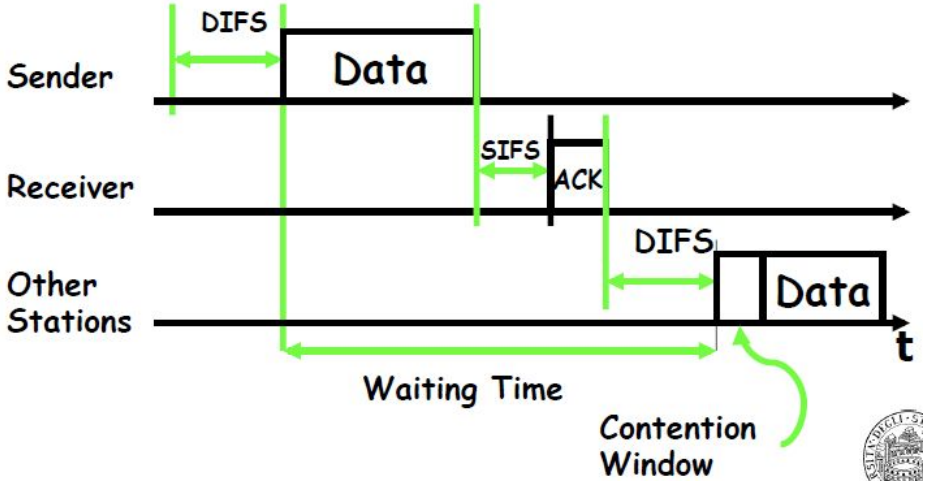
```
while (want_transmit)
      if (i = 1)
            CW_1 = CW_min
      if (i > 1)
            CW_i = 2*CW_i - 1
      if (CW_i > CW_max)
            CW_i = CW_max
```

- in definitiva
  - La prima stazione a cui scade il backoff inizia la trasmissione
  - Le altre stazioni percepiscono la trasmissione e bloccano il loro backoff, che fanno ripartire nel successivo periodo di contesa
- in caso di collisione:
  - $CW_{max} = 2 * CW_{max}$
  - aumento esponenziale del tempo di backoff in caso di ritrasmissioni multiple
  - selezione di un nuovo contatore in base al nuovo CW
  - attesa fino allo scadere del contatore di backoff
  - ricomincia l'algoritmo



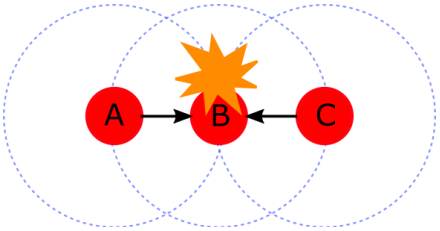
CSMA/CA con ACK

- il ricevente invia un ACK subito dopo la ricezione di una trama (aspetta un tempo pari a SIFS < DIFS) SENZA ascoltare il mezzo
- la maggior parte delle implementazioni però demandano gli invii di ACK ai livelli superiori tipo il livello 4, e non al livello 2 quindi, essendo più affidabile

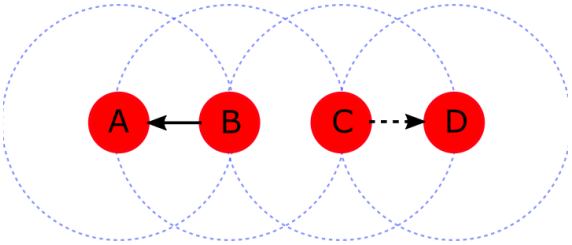


DCF con RTS/CTS (risolve il problema del terminale nascosto)

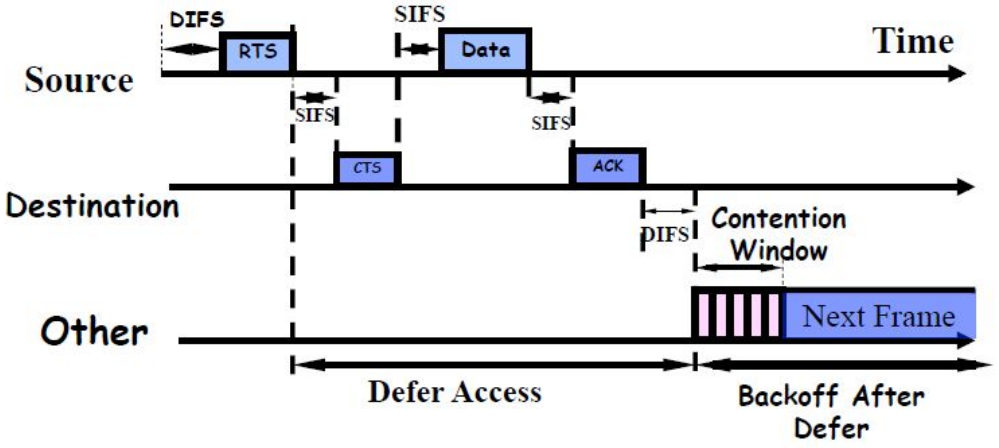
- ci sono particolari disposizioni spaziali (troppo distante dall'AP, potenza troppo debole) delle stazioni (e dell'AP) che fanno percepire la stazione solo ad un sottoinsieme di altre stazioni: si verifica il **problema del terminale nascosto**
  - A invia una trama a B e C ascolta il canale
  - C è fuori range e non riesce a ricevere il segnale di A
  - C invia a B e i segnali di A e C collidono



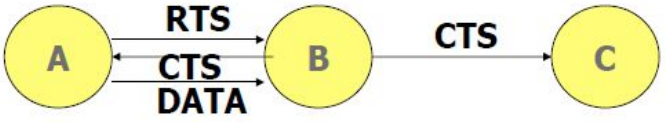
- oppure del **terminale esposto**
  - B invia trame ad A, C vuole parlare con D
  - C ascolta il canale e rileva la trasmissione di B
  - C non trasmette anche se avrebbe potuto visto che non avrebbe disturbato la trasmissione ad A



- RTS/CTS risolve questi problemi
  - la sorgente invia una trama **RTS (Requets To Send)** dopo aver percepito libero il canale per DIFS
  - il ricevente risponde con una trama **CTS (Clear To Send)** dopo SIFS
  - i dati possono essere trasmessi
- quindi RTS/CTS vengono usati per riservare il canale e sono gli unici soggetti a possibili collision



- il problema del terminale nascosto viene quindi risolto con:
  - A invia una trama RTS a B, che manda in broadcast un trama CTS
  - A e C ricevono la CTS e C blocca quindi il suo invio
  - A invia i dati a B senza problemi
  - C come capisce quanto tempo aspettare prima di ritrasmettere?
    - A include la lunghezza dei dati che vuole trasmettere nella trama RTS
    - B include la stessa informazione nella trama CTS
    - C quando riceve la CTS sa quanti dati B deve trasmettere e quindi calcola l'attesa



- il problema del terminale esposto viene risolto con:
  - B invia una RTS per A che viene percepita anche da C
  - A invia la CTS a B ma C non può riceverla perché fuori range
  - C assume che la trasmissione di A non sia raggiungibile e quindi invia senza problemi a D

Network Allocation Vector

La maggior parte delle trame 802.11 includono il campo LENGTH; le stazioni che le percepiscono impostano il NAV al tempo in cui si aspettano il mezzo libero in base al suddetto campo; se NAV > 0, il mezzo è considerato occupato