Università degli Studi di Verona

Reti di Calcolatori

Dispense del Corso

Mattia Zorzan

14 agosto 2020

Indice

1 Introduzione

La presente è una dispensa riguardante il corso di **Reti di Calcolatori**. Per la stesura di questa dispensa si è fatta fede al materiale didattico fornito direttamente dal professore nell'Anno Accademico 2018/2019. Eventuali variazioni al programma successive al suddetto anno non saranno quindi incluse.
Insieme a questo documento in formato PDF viene fornito anche il codice LaTeXcon cui è stato generato, reperibile all'indirizzo `https://github.com/davbianchi/dispense-info-univr.git`.

2 Una breve introduzione alle Reti

2.1 Protocollo e Dispositivi

Per prima cosa diamo la definizione di **protocollo**

Protocollo Definisce un formato e l'ordine dei messaggi scambiati tra due o più entità in comunicazione, così come le azioni intraprese in fase di trasmissione/ricezione di un messaggio.

Quelli a cui noi siamo abituati sono i *protocolli umani*, in questo caso invece faremo riferimento ai *protocolli di rete*. In questi protocolli le entità sono dispositivi HW e SW. I protagonisti possono essere

- **Sistemi Terminali (Hosts)**: Fanno girare programmi ed applicativi, sono posti alle estremità di Internet.
- **Clients/Servers**: Il *client* (host) richiede e riceve un servizio da un programma *server* in esecuzione su un altro terminale.

2.2 Il nucleo della Rete

Internet possiede una struttura fondamentalmente gerarchica. Al centro di tutti troviamo gli **ISP (Internet Service Provider)** di *livello 1* (Nazionali o Internazionali). I loro diretti sottoposti sono gli ISP di *livello 2* (Nazionali o Distrettuali) che dialogano solo con alcuni ISP di livello 1 e tra loro. Ultimi sono gli ISP di *livello 3* e gli ISP *locali*. Queste vengono definite "*Last Hop Networks*", ossia le più vicine ai sistemi terminali.

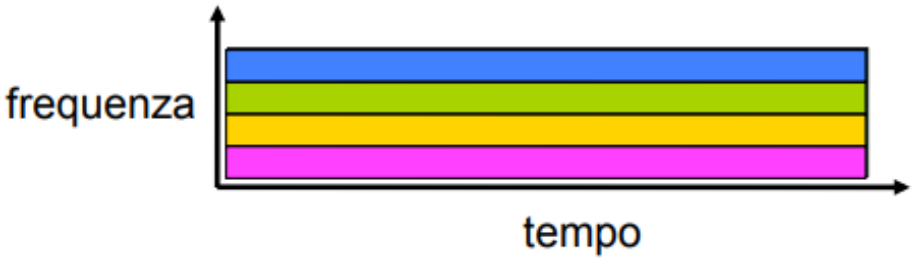
La vera domanda che dovrebbe sorgere spontanea è, come vengono trasferiti i dati attraverso la rete? Vi sono fondamentalmente due modi

- **Commutazione di Circuito**
- **Commutazione di Pacchetto**

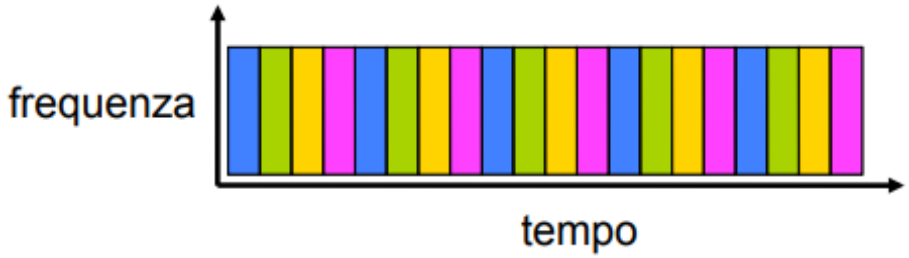
2.2.1 Commutazione di Circuito

Nella *Commutazione di Circuito* per l'intera durata della sessione si hanno risorse riservate per consentire la comunicazione tra sistemi periferici, utilizzabili in modo esclusivo. Le risorse sono infatti usate "*alla chiamata*", quindi potrebbe essere necessario un accodamento per l'utilizzo di queste.
L'ampiezza della banda su cui è possibile trasmettere è infatti la capacità del commutatore, queste risorse vengono divise in pezzi ed ogni pezzo viene allocato ai vari collegamenti.
La naturale conseguenza dell'impossibilità nel condividere risorse è l'inutilizzo di queste.
La banda può essere suddivisa in base a due criteri

- **Divisione di Frequenza**



- **Divisione di Tempo**



Tutti gli esempi precedenti vengono proposti prendendo in considerazione una situazione con 4 utenti

2.2.2 Commutazione di Pacchetto

In questa metodologia il flusso dei dati viene suddiviso in "*pacchetti*", questi viaggiano attraveso i collegamenti e la loro velocità è pari alla *velocità totale di trasmissione* del collegamento stesso. Se invio quindi un pacchetto di L bit su un canale con valocità R *bps*, il tempo totale srà di L/R *secondi*.
I commutatori di pacchetto utilizzano la **trasmissione store-and-forward**, quindi prima di poter inviare in uscita il primo bit di un pacchetto questo dovrà essere arrivato interamente nel buffer.

Può succedere, a causa dell'impossibilità del buffer di fare lo storage di altri pacchetti, che alcuni di questi vengano scartati. Quando un pacchetto viene scartato può essere ritrasmesso come no.

L'arrivo dei pacchetti può avvenire in ritardo, distinguiamo 4 tipi di ritardo

- **Ritardo di Elaborazione del Nodo**
- **Ritardo di Accodamento**
- **Ritardo di Trasmissione**
- **Ritardo di Propagazione**

2.2.2.1 Ritardo di Elaborazione del Nodo

Può essere causato dal controllo degli errori sul bit o dalla determinazione del canale di uscita

2.2.2.2 Ritardo di Accodamento

Può essere causato dall'attesa di trasmissione o da un eccessiva congestione del router

2.2.2.3 Ritardo di Trasmissione

Si calcola come L/R, con R=*Frequenza di trasmissione del collegamento (in bps)* e L= *Lunghezza del pacchetto*

2.2.2.4 Ritardo di Propagazione

Si calcola come d/s, con d= *Lunghezza del collegamento fisico* e s= *Velocità di propagazione del collegamento*

3 Il Modello a Strati

La problematica legata alla comunicazione tra sistemi è costituita da due sottoproblemi

1. Il linguaggio utilizzato
2. La modalità di trasmissione delle informazioni

Creiamo quindi un modello che descriva le caratteristiche della comunicazione. In questo chiameremo il linguaggio come **Comunicazione Logica** ed le modalità di scambio come **Cominicazione Fisica**.
Con il fine di semplificare il tutto, viene adottato un approccio a strati basato sul "*divide et impera*". Vi sono però delle regole

- Ogni livello comunica solo con i due livelli *adiacenti*. Riceve il messaggio da quello superiore, lo elabora e lo spedisce al livello inferiore (Comunicazione Fisica)
- Il messaggio viene rimandato con informazioni aggiuntive rispetto a quando viene ricevuto, grazie all'attività di elaborazione
- Ogni livello colloquia con il suo omologo di una altra macchina (Comunicazione Logica)

Tra ogni coppia di livelli adiacenti esiste un **interfaccia** che definisce i *servizi* offerti dal livello sottostante a quello superiore. Per espletare i suoi servizi, ogni livello compie una serie di funzioni che verranno poi sfruttate attraverso richieste (*primitive*).
Queste primitive possono essere di 4 tipi

- **Request**: Richiesta di un servizio
- **Indication**: Indicazione di un evento
- **Response**: Risposta ad un indicazione
- **Confirm**: Conferma di una richiesta

Per ogni *Request* esiste un *Indication* di entità pari.

Abbiamo invece due tipi di **servizi**

- **Connection Oriented**: Simulano una connessione punto-punto (gestiscono consegna sequenziale)
- **Connectionless**: Semplice passaggio dell'informazione senza preoccuparsi si instaurare una connessione

Per ogni tipo di servizio è possibile associare una **Quality of Service**

- Affidabile: Non ci sono perdite perchè il ricevente informa sempre dell'avvenuta ricezione
- Non Affidabile: Sono possibili perdite di dati

Definiamo ora lo stack dei livelli Internet, o **Stack Protocollare**, e vediamo come questi sono disposti

5. Applicazione
4. Trasporto
3. Rete
2. Collegamento
1. Fisico

3.1 Applicazione

Include protocolli quali *HTTP*, *SMTP* e *FTP*. Tramite *DNS* avviene anche la traduzione dei nomi degli host in indirizzi di rete a 32 bit.
In questo livello i pacchetti scambiati prendono il nome di **messaggi**.

3.2 Trasporto

Troviamo due protocolli a livello di trasporto: *TCP* e *UDP*. TCP fornisce un servizio *connection-oriented*, garantendo la consegna dei messaggi e controllo di flusso. Altra caratteristica è la possibilità di spezzare segmenti troppo lungi in segementi più piccoli e fornisce un meccanismo di controllo della congestione.
UDP invece fornisce un servizio *non connection-oriented*, senza garantire la consegna dei messaggi, senza controllo di flusso e congestione.
I pacchetti a livello Trasporto prendono il nome di **segmenti**.

3.3 Rete

Si occupa di trasferire i pacchetti da un host all'altro. Il protocollo a livello di Trasporto passa al livello di Rete un *segmento* e *un indirizzo di destinazione*.
Il livello di Rete comprende il protocollo *IP*, che definisce i campi dei pacchetti e come sistemi periferici e router agiscono su questi.
Contiene anche vari prtocolli di instradamento che determinano i percorsi che il pacchetto deve seguire da sorgente a destinazione.
I pacchetti prendono il nome di **datagrammi**.

3.4 Collegamento

Si occupa del vero e proprio trasferimento dei datagrammi da nodo a nodo. Possono esserci diversi protocoli di livello 2 tra un nodo e l'altro, il livello di Rete dovrà quindi dattarsi di conseguenza.
A livello Collegamento i pacchetti prendono il nome di **trame** o **frames**.

4 Livello Applicazione

4.1 Architettura Client-Server

Server Il *server* è un host sempre attivo con indirizzo IP fisso.

Client Il *client* comunica con il server in qualunque momento esso voglia, può avere indirizzo IP dinamico. Non comunica direttamente con gli altri client.

Sugli host vengono eseguiti dei programmi, detti **processi**, che comunicano tra di loro utilizzando *schemi interprocesso* definiti dal Sistema Operativo. Processi su host differenti comunicano attraverso *messaggi*.
Distinguiamo due tipologie di processi

- **Processi Client**: Processo che da inizio alla comunicazione
- **Processi Server**: Processo che attende di essere contattato

Socket Un processo invia e riceve messaggi tramite la sua *Socket*. Il concetto è analogo a quello di *porta* Un processo che vuole inviare un messaggio lo fa uscire dalla propria socket. Questo presuppone l'esistenza di un'infrastruttura esterna che trasporterà il messaggio attraverso la rete fino alla porta del processo di destinazione.
Notare il fatto che non basti essere a conoscenza dell'IP della destinazione, sullo stesso host possono girare più processi, è per questo importante indicare anche la porta associata al processo in questione.

4.2 Servizi dei protocolli di trasporto Internet

4.2.1 Servizio di TCP

- Connection oriented: Richiede un setup tra i processi client e server
- Controllo di Flusso: il mittente non vuole sovraccaricare il destinatario
- Controllo della Congestione: Riduce il processo di invio quando la rete è sovraccaricata
- NON OFFRE: Temporizzazione, garanzia su ampiezza di banda minima, sicurezza

4.2.2 Servizio di UDP

- Trasferimento dati inaffidabile tra processi d'invio e di ricezione
- NON OFFRE: Setup della connessione, affidabilità, controllo di flusso, controllo di congestione, temporizzazione, ampiezza di banda minima e sicurezza

4.3 Web e HTTP

Una pagina web è costituita da oggetti, questi possono essere varie cose, come immagini, filmati, file HTML, ecc.
Ogni pagina web è formata da un file HTML di base che include i diversi oggetti referenziati da un **URL**.

Il protocollo che gestisce le applicazioni web è l'**HTTP (HyperText Transfer Protocol)** e si basa sul modello *client/server*

- Il *client* richiede, riceve e "visualizza" gli oggetti web
- Il *server* invia questi oggetti in risposta ad una richiesta

Questo protocollo usa *TCP*, prima il client instaura una connessione TCP (ovvero crea una *socket*) con il server (*Porta 80*), il server accetta la connessione TCP dal client in modo da poter cominciare lo scambio tra i due, alla fine la connessione viene chiusa.
HTTP è un protocollo **stateless**, ovvero il *server* non tiene traccia delle richieste del client.
Posso avere 2 tipi di connessioni HTTP

- **Connessioni non persistenti**: Almeno un oggetto viene trasmesso su una connessione TCP
- **Connessioni persistenti**: Più oggetti possono essere trasmessi su una singola connessione TCP

4.3.1 Connessioni non persistenti

Dopo che l'utente ha inserito l'URL

1. Il client HTTP comincia una connessione TCP con il server HTTP sulla porta 80
2. Il server HTTP, in attesa di una connessione TCP alla porta 80 "accetta" la connessione e avvisa il client
3. Il client trasmette un *messaggio di richiesta* nella socket della connessione TCP indicando quale oggetto vuole
4. Il server HTTP riceve il messaggio di richiesta, forma un *messaggio di risposta* che contiene l'oggetto richiesto, lo invia poi nella sua socket
5. Il server HTTP chiude la connessione TCP
6. Il client HTTP riceve il messaggio contenente la risposta, esaminando poi il file HTML ripeterà n volte questi passi per ogni oggetto aggiuntivo che gli servirà

In base a quanto appena visto definiamo ora il concetto di **Round Trip Time (RTT)**, ossia il tempo di risposta. Esso viene indicato come il tempo che ci mette un pacchetto per andare da client al server HTTP e ritornare.
Tenuto conto che ci dovrà essere un pacchetto per iniziare la connessione, una richiesta e l'effettivo trasferimento dei dati, è intuitivo dire che in totale ci vorrà un tempo di *2RTT + tempo di trasmissione*.

4.3.2 Connessioni persistenti

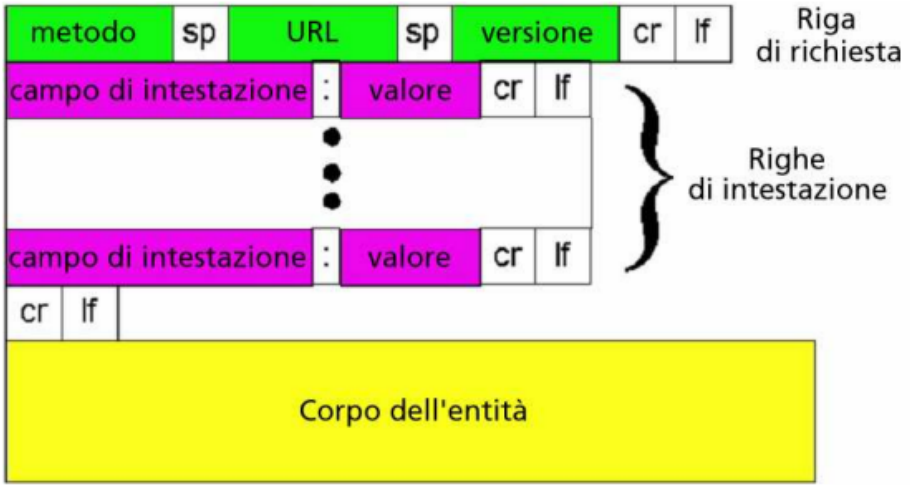
In questo caso il server lascia la connessione aperta dopo l'invio di una risposta in modo che i successivi messaggi tra client e server HTTP sfrutteranno questa connessione già esistente. Il client invia le richieste non appena incontra un oggetto referenziato nel file HTML principale, mettendoci un RTT per tutti gli oggetti

4.3.3 Messaggi HTTP

Si possono avere 2 tipi di messaggi

- Messaggi di **Richiesta**
- Messaggi di **Risposta**

4.3.3.1 Messaggi di Richiesta



Per eseguire *l'upload dell'input di un form* posso usare due metodi

- **Metodo Post**: La pagina web spesso include form per input dall'utente, questo input viene incorporato nel messaggio al server

- **Metodo URL**: Usa il metodo *GET*, l'input arriva nel campo URL

Definiamo quindi i tipi di metodi

- **HTTP/1.0**
  - GET
  - POST
  - HEAD (Esclusione di un oggetto nella risposta)
- **HTTP/1.1**
  - GET, POST, HEAD (Come 1.0)
  - PUT (Include il file nel corpo dell'oggetto e lo invia al percorso specificato nell'URL)
  - DELETE (Cancella il file specificato nell'URL)

4.3.3.2 Messaggi di Risposta

Il formato della risposta HTTP è molto simile a quello della richiesta, vengono introdotti dei codici di stato nella prima riga. Questi possono avere vari significati.

- **200 OK**: La richiesta ha avuto successo
- **301 Mooved Permanently**: Oggetto trasferito, nuova posizione nel campo *Location* della risposta
- **400 Bad Request**: Messaggio non compreso dal server HTTP
- **404 Not Found**: Il documento richiesto non si trova sul server
- **500 HTTP Version not Supported**: Il server non ha la versione del protocollo HTTP

4.3.4 Cookie

Sono formati da quattro componenti

1. Riga di intestazione nella risposta HTTP
2. Riga di intestazione nella richiesta HTTP
3. File mantenuto nel terminale dell'utente e gestito dal browser
4. Un database sul sito

Ma cosa possono contenere i cookie? Nello specifico

- Autorizzazione
- Carta per Acquisti
- Raccomandazioni
- Stato della Sessione (E-Mail)

4.3.5 Cache Web

L'obiettivo è quello di soddisfare la richiesta dell'utente senza contattare il server d'origine. In questo modo il browser invia tutte le richieste HTTP al *Server Proxy*. Se l'oggetto richiesto è nella cache allora viene ritornato, altrimenti il server proxy si fa carico della richiesta al server d'origine, ritornando poi l'oggetto al client. Questo metodo è stato implementato al fine di rendere più veloce la risposta del server al client, riducendo di conseguenza il traffico.

4.3.5.1 GET Condizionale

L'obiettivo è quello di non inviare un oggetto se la *cache* ne ha già una copia aggiornata. Distinguiamo un messaggio *conditional GET* dal fatto che nella sua intestazione è presente il campo `If-modified-since:` seguito da un timestamp, che rappresenta la data di modifica del file, se questo è stato modificato dopo il timestamp il server provvederà ad inviarne una copia aggiornata, ignorerà la richiesta altrimenti.

4.4 FTP

Il **File Transfer Protocol** serve per il trasferimento di file da/a un host remoto.
Il *client* è il lato che inizia il trasferimento, il *server* è un host remoto. Utilizza la *porta 21*.

Il client FTP contatta il server alla porta 21, specificando TCP come protocollo di trasporto. Il client ottiene l'autorizzazione sulla connessione di controllo, tramite comandi su questa cambia poi la *directory remota*.
Quando il server riceve un comando per trasferire un file viene aperta una connessione TCP verso il client, questa verrà poi chiusa alla fine del trasferimento.

Ecco una lista di *comandi*

- **USER username**
- **PASS password**
- **LIST**: Elenca i file della directory
- **RETR filename**: GET su un file della directory
- **STOR filename**: PUT su un file della directory

E *risposte* comuni

- **331 username OK, password required**
- **125 data connection already open; transfer starting**

- **425 Can't open data connection**
- **452 Error writing file**

4.5 Posta Elettronica

Vi sono 3 requisiti fondamentali per l'utilizzo della *Posta Elettronica*, il primo, banalmente, è un utente, servono poi un server di posta e **SMTP (Simple Mail Transfer Protocol)**.
Penso sia scontato spiegare quale sia il ruolo dell'utente, passiamo dunque al server di posta. Ogni mail server è composto da una *casella di posta (mailbox)* che contiene i messaggi in arrivo. Abbiamo poi una *coda di messaggi* da trasmettere e il protocollo SMTP che gestisce l'invio dei messaggi da un server di posta all'altro.

4.5.1 SMTP

Il *Simple Mail Transfer Protocol* usa TCP per inviare messaggi dal client al server, *porta 25*. Il trasferimento avviene in modo diretto tra due server, questo avviene in 3 fasi

- *Handshaking*
- *Trasferimento del messaggio*
- *Chiusura*

Alcuni dettagli da ricordare sono che SMTP usa una connessione persistente ed il messaggio sia nel formato *ASCII a 7 bit*.
Possiamo paragonare SMTP e HTTP molto facilmente, infatti se HTTP esegue un'azione di *pull* SMTP ne esegue una di *push*, la differenza sta nel come vengono inviati gli oggetti. HTTP incapsula ogni oggetto in un proprio messaggio di risposta mentre SMTP può inviare più oggetti in un singolo messaggio.

I messaggi hanno un proprio formato, naturalmente, sono infatti composti da *intestazione* e *corpo*. L'intestazione contiene *from*, *to* e *subject*, banale capire cosa sono, mentre il corpo è l'effettivo messaggio. Si può invece usare **MIME** per estendere il messaggio. Con questo viene aggiunto il supporto a

- L'impiego di codifiche diverse da *ASCII*
- L'aggregazione di messaggi
- La codifica di messaggi (o di una loro parte) non testuali

4.5.2 Protocolli di Accesso

Tra questi troviamo

- **POP3**: Autorizzazione tra utente e server e download

- **IMAP**: Manipolazione di messaggi
- **HTTP**

4.5.2.1 POP3

Fase di Autorizzazione

- *Comandi lato client*
  - **user**: Dichiara il nome utente
  - **pass**: Password
- *Risposte del server*
  - **+OK**
  - **-ERR**

Nelle transazione (lato client)

- **list**: Elenca i numeri dei messaggi
- **retr**: Ritorna i messaggi in base al numero
- **dele**: Cancella un messaggio
- **quit**

POP3 è un protocollo stateless tra le varie sessioni

4.5.2.2 IMAP

Con *IMAP* i messaggi vengono mantenuti sul server, offrendo inoltre la possibilità di organizzarli in cartelle. A differenza di POP3 lo stato dell'utente viene mantenuto tra le varie sessioni.

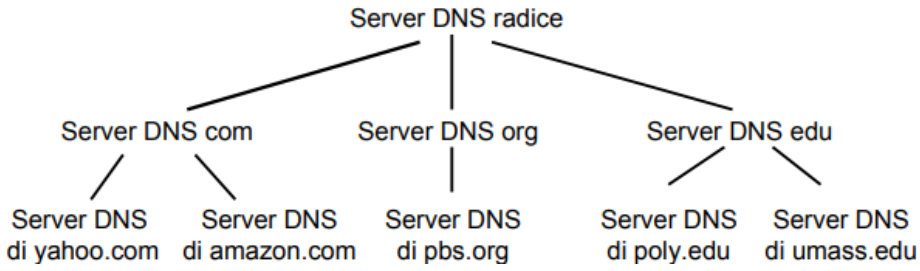
4.5.2.3 HTTP

Alcuni server di posta utilizzano *HTTP* per l'interazione con i messaggi di posta in entrata, tra questi *Gmail, Yahoo! Mail* e *Hotmail*, per citarne alcuni.

4.6 DNS

Le persone come noi hanno molti modi per identificarsi, come il Nome o il Codice Fiscale. Gli Host (o i Router) invece vengono tanto identificati con un codice a *32 bit* detto *Indirizzo IP* quanto da un nome, si pensi al fatto che per connettersi a *Google* non si digita un IP ma *www.google.it*.
Per l'associazione tra nome e IP si una il **DNS (Domain Name System)**.

DNS è un *database distribuito*, implementato come dei *server DNS* che si rapportano l'uno con l'altro gerarchicamente.



Il server *Radice* viene contattato da un server DNS locale che non può risolvere un nome, se anch'esso non conosce la mappatura contatta un server DNS autorizzato, ottenendola. La ritorna poi al server DNS locale.

Oltre alla traduzione degli hostname in indirizzi IP, DNS offre altri servizi, quali:

- **Host Aliasing**: Fornisce degli alias per i nomi più complessi, in questo caso il nome originale viene detto *hostname canonico*.
- **Mail Server Aliasing**: Stesso concetto, applicato però ai domini degli indirzzi di posta elettronica.
- **Load Distribution**: Redistribuisce il carico tra server replicati, ad esempio quelli di siti molto trafficati.

4.6.1 Server TLD

I **Top-Level Domain** si occupano dei nomi *com, org, edu, ecc.* e tutti quelli locali di alto livello (*uk, fr, ecc.*).
Ogni organizzazione dotata di host Internet pubblici deve fornire i record DNS che mappano i nomi di tali host agli indirizzi IP.

4.6.2 Server DNS Locali

Non appartengono strettamente alla gerarchia dei server, ogni ISP ne ha uno. Quando un host fa una richiesta DNS questa viene inviata al suo server DNS locale (che farà da proxy inoltrandola alla gerarchia di server DNS)

4.6.3 Caching e aggiornamento dei record

Quando un server apprende una mappatura la mette in *cache*, dove però quest'informazione sparisce dopo un certo periodo di tempo. Tipicamente, un server DNS locale memorizza nella cache gli IP dei server TLD

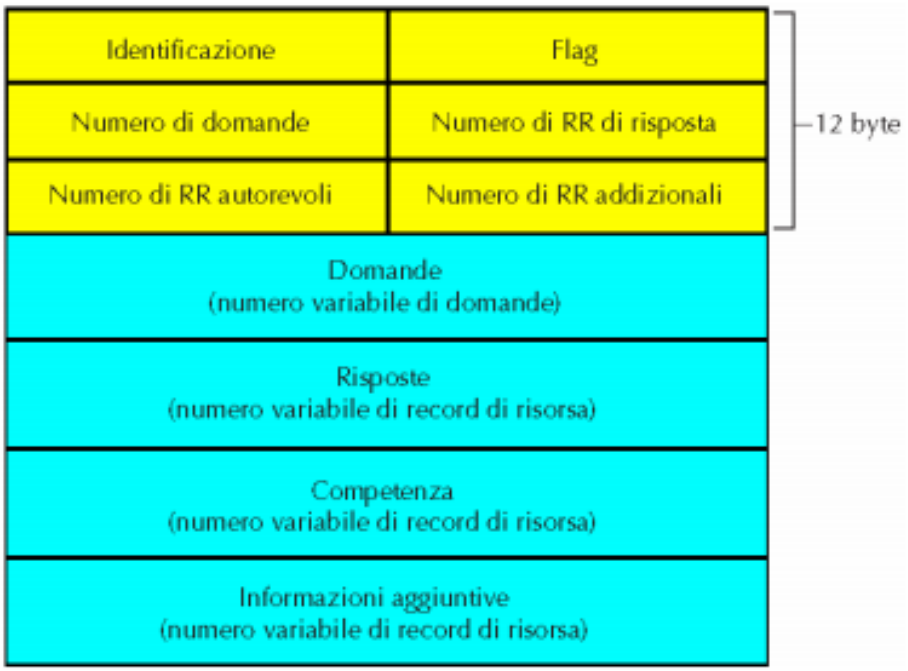
4.6.3.1 Record DNS

Il formato dei **Record di Risorsa (RR)** è il seguente: *(name, value, type, tll)*

- **Type=A**: *name* è il nome dell'host mentre *value* è l'indirizzo IP
- **Type=NS**: *name* è il dominio (Esempio: .com) e *value* è il nome dell'host del server di competenza
- **Type= CNAME**: *name* è l'alias di qualche nome canonico (*www.ibm.com* è in realtà *servereast.backup2.ibm.com*). *value* è il nome canonico
- **Type=MX**: *value* è il nome del server di posta associato a *name*

4.6.3.2 Messaggio DNS

Le domande (*query*) ed i messaggi di risposta condividono lo stesso formato



Dove

- **Identificazione**: Numero di *16 bit* per la domanda, la risposta usa lo stesso numero
- **Flag**:
  - Domanda o risposta

- Richiesta di ricorsione
- Ricorsione disponibile
- Risposta di competenza

4.7 Programmazione delle Socket

La socket è *l'interfaccia* di un host locale, creata dalle applicazioni e gestita dal Sistema Operativo. Su questa il processo di un applicazione può inviare e ricevere messaggi al/dal processo di un altra applicazione. Più semplicemente è il punto in cui un processo accede al canale di comunicazione attraverso una porta.
La **Socket API**, introdotta nel 1981 sfrutta il paradigma client/server. Vi sono fondamentalmente due tipi di servizio tramite Socket API

- Datagram Socket (*utilizza UDP*, inaffidabile)
- Stream Socket (*utilizza TCP*, affidabile e orientata ai Byte)

4.7.1 Programmazione con TCP

Il client, per prima cosa deve contattare il server e quest'ultimo deve essere in esecuzione ed aver creato una socket che da il "benvenuto" al client.
Il client, a sua volta, per contattare il server crea una socket TCP specificando IP e numero di porta del processo server. Creata questa il client TCP stabilisce la connessione con il server.
Al momento del contatto da parte del client il server crea la socket per il suo processo, permettendo così la comunicazione con il processo client. È possibile quindi che lo stesso server comunichi con più client contemporaneamente.
Passiamo ora ad un po' di terminologia

- **Flusso (Strem)**: Sequenza di caratteri che fluisce da/verso un processo
  - **Flusso d'Ingresso (Input Stream)**: Collegato ad un origine d'ingresso (tastiera o socket, per esempio)
  - **Flusso d'Uscita (Output Stream)**: Collegato ad un uscita per il processo (monitor o socket, ad esempio)

4.7.2 Programmazione con UDP

Essendo gestita tramite UDP non vi è una "connessione" tra client e server. Manca infatti la fase di *handshaking*. Per l'invio di ogni pacchetto viene allegatala porta di destinazione e l'IP del destinatario.
Il server dovrà quindi estrarre IP e porta direttamente dal pacchetto arrivatogli. I pacchetti possono andare persi oppure arrivare in ordine diverso rispetto a quello di invio.

5 Livello di Trasporto

Il **Livello di Trasporto** si occupa di fornire un canale end-to-end ideale e privo di errori. Come tutti i livelli offre dei servizi al livello superiore e esegue delle funzioni.
Fornisce essenzialmente due servizi

- **Connection-oriented affidabile**: TCP
- **Connectionless non affidabile**: UDP

Le funzioni svolte sono

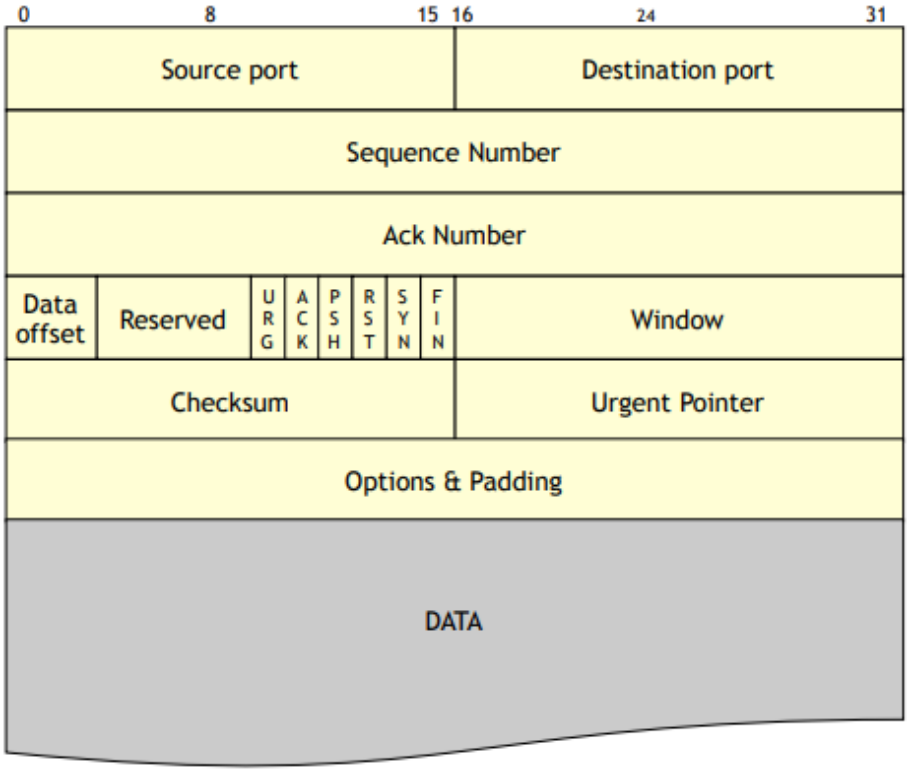
- Indirizzamento a livello applicazione, multiplexing e demultiplexing
  - *Demultiplexing*: Trasporto dei dati dai segmenti alla giusta socket
  - *Multiplexing*: Azione per radunare dati dalle varie socket per incapsularli in segmenti (ognuno con una propria intestazione) che verranno poi passati al livello Rete
- Instaurazione e gestione delle connessioni
- Recupero degli errori
- Consegna ordinata dei segmenti
- Controllo di flusso
- Controllo della congestione

È importante ricordare che tutti questi servizi sono serviti da *TCP* ma solo il primo viene offerto a *UDP*

5.1 TCP

Il **TCP (Transmission Control Protocol)** si occupa del controllo di trasmissione rendendo *affidabile* la comunicazione tra mittente e destinatario.

5.1.1 Header TCP



- **Source/Destination Port** (16 bit ciascuna)
  Indirizzi della porta sorgente e destinazione (rispettivamente)
- **Sequence Number** (32 bit)
  Numero di sequenza del primo byte del payload (DATA nell'immagine)
- **Acknowledge Number** (32 bit)
  Numero di sequenza del prossimo byte che si intende ricevere (solo se questo segmento è *ACK*)
- **Offset** (4 bit)
  Lunghezza dell'header, in multipli di 32
- **Resereved** (6 bit)
  Riservato per usi futuri
- **Window** (16 bit)
  Ampiezza della *Reciving Window* (solo se questo segmento è *ACK*)

- **Checksum** (16 bit)
  Risultato di un calcolo che serve per comprendere se il segmento contiene errori

- **Urgent Pointer** (16 bit)
  Indica che il ricevente deve iniziare a leggere il campo DATA dal numero di byte specificato. Usato se si inviano comandi che generano eventi *asincroni* urgenti

Vi sono anche 6 **flag**

- **URG**: 1 se urgente, in questo caso *Urgent Pointer* ha significato
- **ACK**: 1 se ACK, in questo caso *Acknowledge Number* è un numero valido
- **PSH**: 1 se il trasmettitore usa il comando *PUSH*
- **RST**: Reset della connessione senza chiusura esplicita
- **SYN**: Utilizzato durante il setup della connessione per comunicare il numero di sequenza iniziale (*ISN*)
- **FIN**: Usato per chiusura esplicita della connessione

5.1.2 Indirizzamento

Come posso distinguere diverse applicazioni al livello di trasporto? Si utilizzano le **porte**, ossia un codice che identifica un'applicazione. Conoscendo questo ora è possibile definire le *socket* come tuple che identificano la connessione tra due applicazioni. Il numero di porta può essere

- **Statico (Porte Note)**: Identificativo ad applicazioni largamente utilizzati. Sono stabilite dall'**IANA** (*Internet Assigned Number Authority*) e sono quelle inferiori al 1024
- **Dinamico**: Identificativi assegnati direttamente dal Sistema Operativo al momento dell'apertura della connessione

L'utilizzo delle porte, insieme agli indirizzi IP sorgente e destinazione (*socket* quindi) servono per multiplexing/demultiplexing dati da parte di TCP. I socket identificano univocamente una connessione (su cui passa un unico flusso di informazioni).

5.1.3 Gestione delle connessioni

5.1.3.1 Instaurazione della Connessione

Il TCP è un protocollo *connection-oriented*, ossia ci deve essere una connessione tra mittente e destinatario prima di inviare dati.
Quest'instaurazione avviene attraverso un **Three-Way Handshake**

- La stazione A (richiedente della connessione) invia il segmento di SYN specificando
  - Numero di porta dell'applicazione cui si vuole accedere
  - ISN (Initial Sequence Number)
- La stazione B (ricevente della richiesta di connessione) risponde con un segmento SYN specificando
  - ISN
  - ACK al segmento SYN di A
- La stazione A riscontra il segmento SYN di B con un ACK

Questa procedura apre due canali monodirezionali, che visti come un'unica entità risultano un unico canale bidirezionale.
I segmenti SYN, ACK, dati, ecc. sono particolari, i loro campi dell'header assumono valori particolari

- Un segmento SYN è un segmento vuoto, possiede solo l'header ed il suo flag SYN è posto a 1
- Un segmento ACK e un segmento vuoto, possiede solo l'header ed il suo flag ACK è posto a 1
- Un segmento SYN ACK e un segmento vuoto, possiede solo l'header ed il suoi flag SYN e ACK sono posti a 1
- Un segmento dati è un segmento contiene i dati dell'applicazione in cui i falg SYN e ACK sono a 0

MSS (Maximum Segment Size) Nel campo **Options & Padding** dell'header è possibile inserire l'MSS, questo esprime la dimensione massima del campo dati dei segmenti che verranno inviati (in Byte, default 536 Byte)

5.1.3.2 Terminazione della Connessione

Procedura di terminazione

- La stazione che non ha più dati da trasmettere e vuole terminare la connessione invia un segmento FIN
- La stazione che riceve il segmento FIN riscontra questo con un ACK e indica all'applicazione che la connessione entrante è stata chiusa

Se il processo di chiusura termina in questo modo siamo davanti ad una chiusura parziale della comunicazione (si parla di *half-close*). Nell'altro verso la comunicazione può continuare, per chiudere completamente la connessione anche l'altra stazione deve eseguire l'half-close.

5.1.4 Gestione di Errori e Perdite

Il TCP è un protocollo **affidabile**, garantisce infatti la corretta ed ordinata consegna dei segmenti.
Deve quindi poter gestire

- Gli errori (individuati tramite *Checksum*)
- Perdita di segmenti (individuata tramite gli ACK, se non avviene il riscontro di un segmento questo viene considerato perso)

Nel caso in cui ci si trovi in una di queste due situazioni TCP ritrasmette i segmenti. Questa tecnica prende il nome di " *"positive acknowledgement with retransmission"*.
Nella versione base di questa tecnica, la sorgente non ritrasmette il segmento successivo fino al riscontro di quello correntemente inviato.

Ma per quanto devo aspettare prima di considerare un segmento perso? Viene quindi introdotto il concetto di **RTO (Retransmission Timeout)**. Come si può facilmente immaginare, indica la quantità di tempo entro la quale la sorgente si aspetta di ricevere un ACK prima di considerare il segmento perso.
Naturalmente questo non è un valore statico, viene bensì ricalcolato di volta in volta, per primo durante l'instaurazione della connessione, poi durante la trasmissione dei dati. Questo calcolo si basa sull'**RTT (Round Trip Time)** (tempo tra l'invio di un pacchetto ed il suo riscontro)

5.1.4.1 Stima dell'RTT

Il calcolo è molto semplice

$$SRTT = (\alpha * SRTT_{precedente}) + ((1 - \alpha) * RTT_{istantaneo})$$

con $\alpha \in [0, 1]$.
Il valore di α indica

- Stabilità, SRTT non viene influenzato da singoli segmenti aventi RTT molto diversi ($\alpha = 1$)
- Instabilità, SRTT dipende molto dalla misura puntuale dei singoli RTT istantanei

5.1.4.2 Stime dell'RTO

$$RTO = \beta * SRTT$$

con β tipicamente 2 (*Delay Variance Factor*)
In caso di ritrasmissione, l'RTO viene ricalcolato in base ad un processo di *exponential backoff*.
Infatti se è scaduto l'RTO molto probabilmente c'è congestione

$$RTO_{new} = 2RTO$$

5.1.5 Controllo di Flusso

Anche se possono sembrare due concetti coincidenti, *Controllo di Flusso* e *Controllo di Congestione* esprimono cose diverse

- **Controllo di Flusso**: È un'azione preventiva finalizzata a limitare l'immissione di dati in base alla capacità della rete
- **Controllo di Congestione**: Reazione all'eccessiva congestione della rete

Con la tecnica di postive acknoledgement with retransmission abbiamo visto che una stazione invia un nuovo segmento solo se l'ultimo è stato riscontrato, questa tecnica è di suo un azione di *Controllo di Flusso*, anche se poco efficiente.

5.1.5.1 Controllo di Flusso a Finestra

Per aumentare l'efficienza è possibile trasmettere più segmenti consecutivi senza attendere ogni singolo riscontro. Alla ricezione dei riscontri dei segmenti iniziali della sequenza, la finestra scorre a destra permettendo la trasmissione dei nuovi segmenti. Questa tecnica è detta della **Sliding Window**.
Consideriamo ora di avere una finestra di dimensione troppo piccola, in questo caso vi è un sottoutilizzo della banda. Al contrario invece, se la finestra è troppo grande i riscontri potrebbero arrivare prima che tutti i segmenti vengano inviati.

Come comportarsi invece in caso di perdita di segmenti? Abbiamo detto che, in caso di perdita, passato un RTO la sorgente provvede a reinviarlo. E se il segmento facesse parte di una finestra (multipli pacchetti inviati)?
Possibili soluzioni

- **Go-back-N**: La finestra torna indietro e vengono ritrasmessi tutti i segmenti
- **Selective Repeat**: Viene ritrasmesso solo il segmento perso

5.1.6 Controllo di Congestione

A causa di un eccessiva congestione della rete alcuni segmenti potrebbero venire persi, la sorgente quindi dovrebbe diminuire il tasso di immissione dei nuovi segmenti. Questo è detto **Controllo di Congestione**

5.1.6.1 Congestion Window

In caso di congestione il controllo di flusso a finestra protegge implicitamente anche la rete, tuttavia questo potrebbe non essere sufficiente. Difatti va ancora stabilita la dimensione ottimale della finestra.
La soluzione sta nello stabilire dinamicamente la dimensione della finestra, questa prende quindi il nome di **Congestion Window (CWND)**.

5.1.6.2 Algoritmi per il controllo di congestione

Esistono due algoritmi che regolano la dimensione della CWND

- **Slow Start**: Per ciascun riscontro ricevuto la CWND aumenta di dimensione, quindi la CWND ha un andamento esponenziale
- **Congestion Avoidance**: Per ciascun riscontro ricevuto la CWND aumenta di 1/CWND, quindi la CWND ha un andamento lineare

Ci sono alcune variabili da tenere in considerazione

- **CWND**: Dimensione della finestra di trasmissione
- **RECWND (Recieve Window)**: Dimensione della finestra di ricezione, limite massimo della CWND (annunciata dalla *destinazione*)
- **SSTHRESH (Slow Start Threshold)**: Dimensione della CWND raggiunta la quale avviene il passaggio da *Slow Start* a *Congestion Avoidance*
- **RTT**: Tempo trascorso tra l'invio di un segmento e la ricezione del riscontro
- **RTO**: Tempo atteso dalla sorgente prima di ritrasmettere un pacchetto di cui non ha ricevuto riscontro

L'algoritmo che regola la CWND è il seguente

- All'inizio della trasmissione si pone
  - CWND = 1 segmento
  - SSTHRESH = RECWND o RECWND/2
- La CWND evolve con *Slow Start* fino a SSTHRESH
- Raggiunto SSTHRESH si passa a *Congestion Avoidance*
- La CWND cresce fino a RECWND

Cosa fare nel caso vi sia perdita di pacchetti o errori? La finestra non si muove e si aspetta RTO, allo scadere del quale si pone

- SSTHRESH = CWND/2
- CWND = 1

Si riprende quindi a trasmettere con la tecnica *Go-Back-N*, si prosegue quindi come illustrato prima.
Se gli errori/perdite fossero consecutivi? Dal secondo in poi si ricalcola RTO con *Exponential Backoff*, la CWND rimane a 1 ma SSTHRESH viene impostato a 2 segmenti.

5.1.6.3 Fast Retransmit - Fast Recovery

I segmenti vanno persi o per congestione o per errori di trasmissione. Invece che utilizzare metodi eccessivi, magari per la perdita di un solo segmento, sono stati implementati due algoritmi

- **Fast Retransmit**: Ritrasmette subito il segmento considerato perso
- **Fast Recovery**: La CWND non viene chiusa

5.1.6.4 ACK Duplicati

Negli ACK il campo Ack Number contiene il successivo numero di sequenza che ci si aspetta arrivi. Se un segmento arriva fuori sequenza, la destinazione invia un ACK indicando il numero di sequenza del segmento che non e' ancora arrivato.
La ricezione di un numero sufficientemente alto di ACK duplicati può essere interpretata come forte di indicazione che e' avvenuta una perdita.

Se arrivano 3 ACK duplicati

- SSTHRESH = CWND/2
- Si usa *Fast Retransmit*
- CWND = SSTHRESH + 3 (*Fast Recovery*)
- Per ogni successivo ACK duplicato la CWND aumenta di 1

Quando la sorgente riceve l'ACK che conferma la ricezione del segmento ritrasmesso

- CWND = SSTHRESH
- Si prosegue utilizzando *Congestion Avoidance*

Se il segmento ritrasmesso viene perso, si attende lo scadere del RTO, la CWND torna a 1 e si riparte in Slow Start.
Se viene perso piu' di un segmento: l'algoritmo cerca di recuperare il primo, anche se ce la fa, arrivano gli ACK duplicati dei successivi segmenti persi che l'algoritmo non sa trattare, per cui scade il RTO.

5.2 UDP

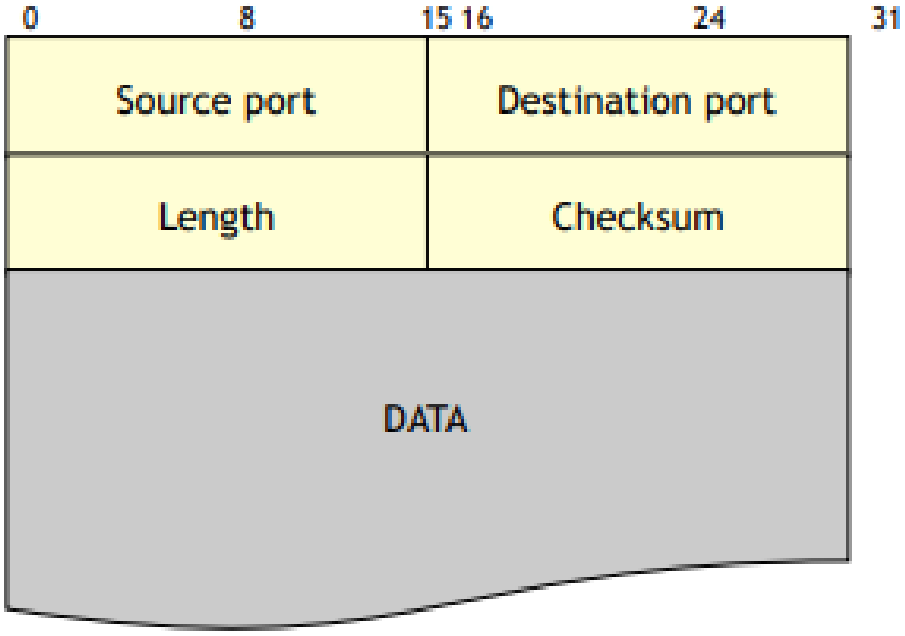
Il protocollo **UDP (User Datagram Protocol)** è un protocollo *connectionless non affidabile*. Esso svolge solo funzione di indirizzamento delle porte, infatti non gestisce

- Connessioni
- Controllo di Flusso

- Recupero errori (li *rileva* e basta)
- Controllo della Congestione
- Riordinamento dei pacchetti

Per tutte queste funzioni si affida ai livelli superiori.

5.2.1 Header UDP



- **Source/Destination Port** (16 bit)
  Identificano i processi sorgente e destinazione dei dati
- **Length** (16 bit)
  Lunghezza in Byte, header compreso
- **Checksum** (16 bit)
  Campo di controllo per il rilevamento errori nel campo DATA

6 Livello di Rete

La funzione chiave del **Livello di Rete** è il **routing**, ovvero la determinazione del percorso che dovranno seguire i datagrammi dalla sorgente alla destinazione. Altra funzione fondamentale è il **forwarding** che trasferisce i pacchetti da una porta di input ad un'appropriata porta di output. Entrambe queste funzioni si basano sull'*addressing*.
Tutte queste operazioni si basano sul funzionamento del **router**, le cui principali sono

- Eseguire algoritmi e protocolli di routing (RIP, OSPF, BGP)
- Trasferire datagrammi dalla porta di input o alla porta di output corretta

6.1 Porte di Input

6.1.1 Switching decentralizzato

Data una destinazione, viene determinata la porta di output in base alle tabelle di forwarding in memoria locale. Se i datagrammi arrivano ad una velocità maggiore di quella di commutazione si utilizza il *queuing*, che ne permette l'accodamento in un buffer.

6.1.2 Commutazione basata su memorie

Era utilizzata dalla prima generazione di router. All'epoca la commutazione era controllata dalla CPU, il pacchetto veniva copiato nella memoria di sistema, quindi la velocità era "*limitata*" dalla velocità della memoria.

6.1.3 Commutazione via Bus

La commutazione tra porte avveniva attraverso un bus condiviso. La velocità di commutazione era data dalla velocità del bus.

6.1.4 Commutazione con rete interconnessa

Sfrutta le *Reti di Banyan*, ossia reti di interconnessione sviluppate per connettere *processori in architetture multiprocessore*. I datagrammi vengono frammentati in celle di lunghezza fissa, commutate successivamente in matrici ottimizzate.
Consiste in $2n$ bus che collegano n porte di input a n porte di output.

6.2 Porte di Output

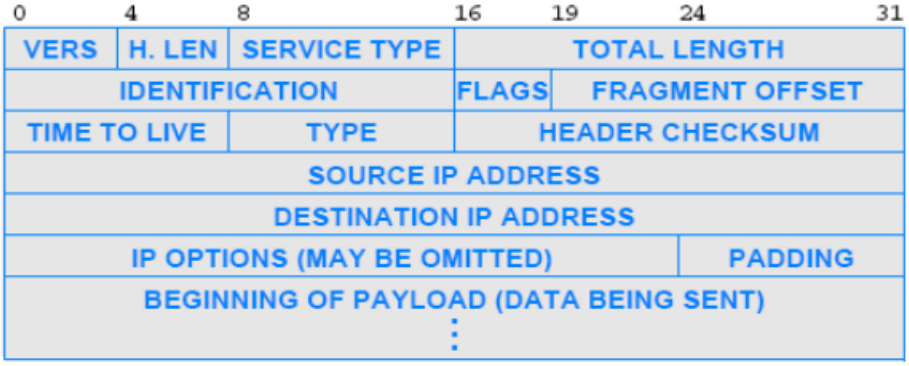
Due concetti

- **Buffering**: Necessario quando la velocità di arrivo dei datagrammi è superiore a quella di trasmissione
- **Scheduling**: Utilizzato per determinare l'ordine di trasmissione dei datragrammi nella queue del buffer

6.3 IP

Nello stack *TCP/IP* si utilizza il termine **datagramma IP** per riferirsi ad un pacchetto, come per il TCP questo è costituito da un *header* e da un *payload*.

6.3.1 Header IP



- **VERS** (4 bit)
  Indica la versione del protocollo
- **H.LEN** (4 bit)
  Specifica la dimensione dell'header ($tot\_byte\_header/4$)
- **SERVICE TYPE** (8 bit)
  Classe di servizio del datagramma
- **TOTAL LENGTH** (16 bit)
  Numero totale di byte del datagramma
- **IDENTIFICATION** (16 bit)
  Numero assegnato al datagramma, usato in caso di frammentazione
- **FLAGS** (3 bit)
  Sono in realtà 3 flag da 1 bit ciascuno. Il primo (*Reserved*) è sempre settato a 0, il secondo (*Don't Fragment*) indica se il datagramma può essere frammentato o meno ed il terzo (*More Fragments*) indica se questo datagramma è l'ultimo di una serie di frammenti
- **FRAGMENT OFFSET** (13 bit)
  Offset del frammento rispetto al datagramma originale (va moltiplicato per 8 per ottenere il vero offset)
- **TIME TO LIVE** (8 bit)
  Viene decrementato da ogni router attraverso cui il datagramma passa, se raggiunge 0 questo viene scartato

- **TYPE** (8 bit)
  Specificano il tipo di dati trasportati dal payload
- **HEADER CHECKSUM** (16 bit)
  Checksum dell'header
- **SOURCE IP ADDRESS** (32 bit)
  IP della sorgente
- **DESTINATION IP ADDRESS** (32 bit)
  IP della destinazione
- **IP OPTIONS**
  Campi opzionali con informazioni aggiuntive
- **PADDING**
  Se ho un "IP OPTIONS" valido che non arriva a 32 bit si aggiungono 0 fino a raggiungere quella dimensione

6.3.2 Frammentazione IP

A seconda della tecnologia HW, i diversi tratti di rete possono avere **MTU (Maximum Transmission Unit)** diverso. Quando un datagramma ha dimensione maggiore dell'MTU del tratto di rete in cui deve essere inviato il router lo divide in *frammenti* che vengono inviati indipendentemente.
Il router usa quindi il valore dell'MTU per calcolare la dimensione massima di ogni frammento ed il numero effettivo di questi. L'header di ogni frammento è pressochè uguale a quello del datagramma originale (a parte ovvie differenze come *More Fragments* o *Fragment Offset*), ogni frammento porterà nel suo payload una porzione dei dati del datagramma originale.

6.3.2.1 Riassemblaggio

Il datagramma verrà riassemblato a destinazione, riducendo così la quantità di dati da memorizzare nel router (*forwarding* non ha bisogno di sapere se un datagramma è frammento o no).
Ogni frammento viaggia in modo dinamico, se ci fosse un vincolo di passaggio per il riassemblaggio tutti i frammenti dovrebbero passare da quel router, aumentando il traffico.

6.3.2.2 Perdita di Frammenti

I datagrammi vengono riassemblati solo quando tutti i frammenti sono arrivati a destinazione. Fino a quel momento questi sono salvati in un buffer, IP specifica un tempo massimo per cui i frammenti vanno conservati (parte all'arrivo del primo frammento).
Non esistono meccanismi per i quali la destinazione comunichi i frammenti arrivati e se la sorgente ritrasmette il datagramma i suoi frammenti potrebbero seguire percorsi diversi.

6.3.3 Indirizzamento a Livello di Rete

L'indirizzamento è una componente fondamentale per Internet, infatti vi è uno schema di indirizzamnto comune per tutti gli host.
A questo scopo non possono essere usati i MAC-address del *sotto-livello MAC* in quanto Internet coinvolge vari tipi di tecnologie con (possibilmente) varie tipologie di MAC-address. Per ovviare a questo problema vengono utilizzati gli **Indirizzi IP**.
A ciascun host viene assegnato un indirizzo univoco a 32 bit, questo dovrà essere specificato quando un host vorrà inviare un datagramma in Internet (*Source IP Address* e *Destination IP Address*, come visto prima nell header).
Gli IP vengono espressi in **Notazione Decimale Puntata**, gli indirizzi sono divisi in 4 sezioni da *8 bit* ciascuna, delimitate da un punto, con *valore minimo 0* e *valore massimo 255*.
Tra i vari indirizzi IP c'è una gerarchia, infatti gli indirizzi sono divisi in *prefisso* e *suffisso*

- **Prefisso**: Identifica la rete fisica a cui un host è connesso
- **Suffisso**: Identifica un host specifico all'interno di una rete

6.3.3.1 Indirizzamento Classful

Sorge spontanea una domanda, quanti bit devo assegnare al prefisso? E quanti al suffisso?
Il prefisso deve essere un numero sufficientemente grande per identificare tutte le reti fisiche in Internet, il suffisso deve essere un numero abbastanza grande da esprimere tutti gli host in una rete.
La soluzione originale al problema prevedeva la divisione dello spazio di indirizzamento in *3 classi primarie*, ciascuna con dimensione di prefisso/suffisso differente. I primi 4 bit di un indirizzo indicavano la classe di appartenenza.
Metà degli indirizzi esistenti appartenevano alla classe A, con limite a 128 reti, in modo da consentire ai principali ISP di creare la loro rete gigante. Con lo stesso concetto la classe C era pensata per permettere a piccole organizzazioni di creare la loro LAN.
Venne creata l'**ICANN (Internet Corporation for Assigned Names and Numbers)** per l'assegnamento degli indirizzi e la soluzione a dispute riguardanti loro. Questa corporation non assegna direttamente i prefissi ma autorizza un insieme di *registrars* a farlo. I registrars assegnano i blocchi agli ISP che li assegnano a loro volta agli utenti.

6.3.3.2 Indirizzamneto Classless

Tutti hanno cominciato a chiedere indirizzai di classe A o B, in modo da avere molti indirizzi al fine di espandersi. Ci si ritrovò in una situazione in cui vi era un forte sottoutilizzo degli indirizzi di ogni classe.
Vennero proposte due soluzioni

- Subnet
- Indirizzamento Classless

Si consideri un ISP ed un cliente che chiede un prefisso che può contenere 55 host. Con *classfull*

- Indirizzo di classe C
- Sarebbero bastati 6 bit per rappresentarli tutti (190 indirizzi sprecati)

Con *classless* l'ISP può assegnare arbitrariamente la dimensione del prefisso, in questo caso *26 bit di prefisso* e *6 bit di suffisso*.
Come può un router conoscere la dimensione di un prefisso essendo questo arbitrario di rete in rete? Invece di aggiungere la dimensione del prefisso esplicitamente venne introdotto il concetto di **Maschera**

Maschera Valore a *32 bit* in cui sono posti a 1 tutti i bit fino a raggiungere la dimensione del prefisso.

Un router tiene in memoria

- I prefissi di rete delle destinazioni
- Le corrispondenti maschere

Quando arriva un datagramma con IP generico il router confronta l'indirizzo di destinazione con gli indirizzi che ha in memoria e fa il forward in base alla destinazione.
Questo confronto non avviene su tutti e 32 i bit, viene infatti considerata la maschera per ogni prefisso con cui fine fatto l'*AND bit-a-bit* nei confronti dell'indirizzo IP. Il risultato viene confrontato con il prefisso in memoria, se sono uguali è stata trovata la destinazione del pacchetto.

6.3.3.3 Notazione CIDR

Il **CIDR (Classless Inter-Domain Routing)** viene espressa nella forma $ddd.ddd.ddd.ddd/m$, dove

- ddd è un valore decimale in funzione della notazione decimale puntata
- m è in numero di bit del prefisso

Dopo aver ricevuto il *prefisso CIDR da un ISP* si possono assegnare liberamente gli indirizzi host ai propri utenti.

6.3.3.4 Indirizzi Speciali

Il protocollo IP definisce una serie di indirizzi riservati detti **Indirizzi Speciali**

- **Indirizzo di Rete (Suffisso a 0)**: Indica il solo prefisso di una rete, identifica quindi le rete nel complesso. Non deve mai comparire come indirizzo di destinazione di un datagramma.
- **Directed Broadcast (Suffisso a 1)**: Semplifica il broadcasting, il datagramma con questo indirizzo di destinazione viaggia su Internet fino a raggiungere la rete indicata dal prefisso per poi essere consegnato a tutti gli host della stessa.
- **Limited Broadcast (Tutti i bit a 1)**: Il datagramma con questo indirizzo come destinazione verrà consegnato a tutti gli host della rete dell'host che ha generato il datagramma. Utilizzato durante lo *startup del sistema*, quando l'host non conosce ancora l'indirizzo di rete.
- **This Host (Tutti i bit a 0)**: Durante il boot un host non può ancora indicare un indirizzo sorgente corretto in quanto non lo possiede ancora. A tale scopo questo indirizzo indica "This Host", ossia indica lo stesso host che ha generato il datagramma. Non valido come "*Destination IP Address*" di un datagramma. Utilizzato in fase di startup per dialogare con i *server DHCP*.
- **Loopback (127.0.0.0/8)** Utilizzato per testing, invece che far girare due programmi su host separati il programmatore utilizza un solo host per entrambi ed utilizza quest'indirizzo per farli comunicare. Il suffisso è irrilevante (solitamente si utilizza 127.0.0.1). Di fatto non viene trasmesso alcun datagramma in quanto come viene inviato esso torna a se stesso.

6.3.3.5 Principi di Indirizzamento

A ciascun router vengono *assegnati 2 o più* IP (uno per ogni rete a cui esso è connesso). Un route infatti ha connessioni verso una o più reti fisiche, il cui indirizzo è specificato in ogni prefisso.
Quindi un IP non identifica un host specifico ma la connessione tra esso e la rete, quindi un host con più IP avrà un indirizzo per ogni connessione.

6.4 ICMP

L'**ICMP (Internet Control Message Protocol)** è un protocollo associato ad IP utilizzato principalmente per inviare messaggi di errore alla sorgente in caso di problemi. Questi due protocolli dipendono uno dall'altro

- IP dipende da ICMP per l'invio di messaggi d'errore
- ICMP dipende da IP per trasportare i messaggi di errore

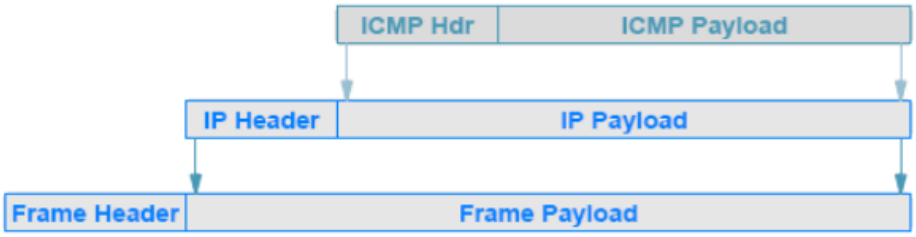
Ecco alcuni messaggi

Number	Type	Purpose
0	Echo Reply	Used by the ping program
3	Dest. Unreachable	Datagram could not be delivered
5	Redirect	Host must change a route
8	Echo	Used by the ping program
11	Time Exceeded	TTL expired or fragments timed out
12	Parameter Problem	IP header is incorrect
30	Traceroute	Used by the traceroute program

Abbiamo due tipologie di messaggi

- Messaggi per *segnalazione errori*
- Messaggi per *ottenere informazioni*

Come detto prima ICMP usa IP per trasportare i propri messaggi, mettendo il messaggio ICMP nel payload del datagramma IP che verrà inviato come ogni altro datagramma. Se un messaggio ICMP genera un errore non verrà inviato alcun errore per evitare "l'effetto valanga".



6.5 DHCP

Quando un host o un router vengono accesi il Sistema Operativo ed il SW che gestisce il protocollo di rete vengono inizializzati.
Chi gestisce un router deve specificare dei valori iniziali, quali

- L'indirizzo per ciascuna connessione di rete (interfaccia)
- Quale SW di protocollo utilizzare
- I valori iniziali delle tabelle di forwarding

Questa configurazione viene salvata e caricata dal router durante lo startup.
La configurazione degli host segue un processo noto come **bootstrapping**. È stato ideato un protocollo che permette ad un host di ottenere una serie di parametri con una singola richiesta, chiamato **BOOTP (Bootstrap Protocol)**. Attualmente è il DHCP che viene usato per fornire la maggioranza delle informazioni di configurazione.

Il **DHCP (Dynamic Host Configuration Protocol)** viene utilizzato per permettere ad un host di una rete di ricevere automaticamente, ad ogni richiesta d'accesso, la configurazione IP necessaria per stabilire una connessione e inter-operare con tutte le altre sotto-reti, purchè anch'esse integrate nello stesso modo con IP.
Sono stati creati vari meccanismi per permettere ad un host di ottenere i diversi parametri

- **RARP (Reverse Address Resolution Protocol)** permette di ottenere un indirizzo IP da un server
- **ICMP** ha i messaggi "*Address Mask Request*" e "*Router Discovery*" utilizzati, rispettivamente, per ottenere la maschera della rete e l'indirizzo del router

DHCP utilizza un approccio "*Plug-and-Play Networking*", infatti permette di ottenere molte di queste informazioni e quelle sopracitate automaticamente.

Quando un host si accende

- Invia in *broadcast* una **DHCP discover** utilizzando UDP attraverso la *porta 67*. In questo messaggio l'indirizzo sorgente sarà *0.0.0.0*
- Il server DHCP risponde con una **DHCP offer** in *broadcast* su tutta la sottorete. Dato che il discover è stato mandato in broadcast, è molto probabile che l'host sorgente riceva più *offers*.
- Il client, dopo aver scelto "*l'offerta più conveniente*", risponderà al server DHCP con una **DHCP request**, contenete i parametri di configurazione.
- Il server risponde con un **DHCP ACK** confermando i parametri.

Si può configurare DHCP per fornire due tipi di indirizzi

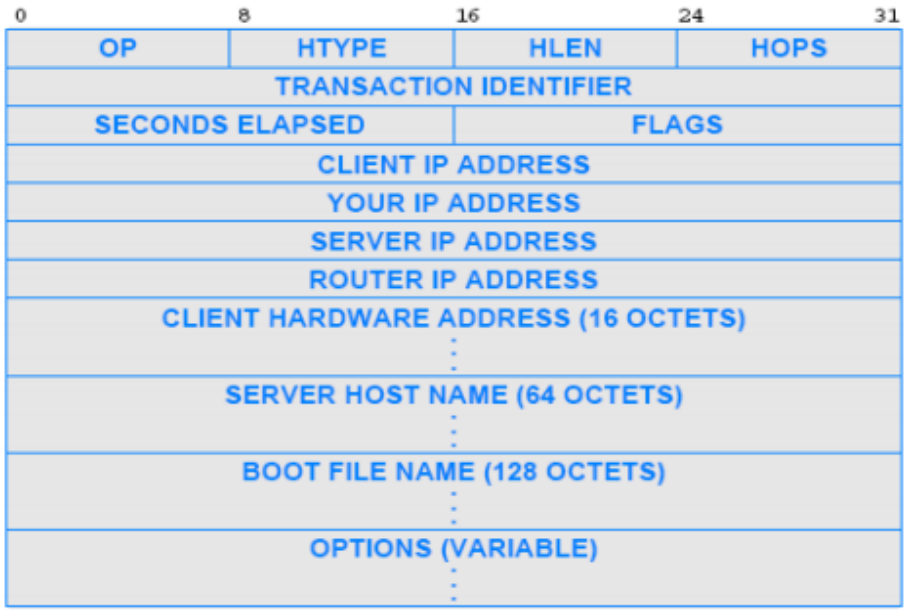
- Indirizzi assegnati permanentemente (come **BOOTP**)
- Indirizzi dinamici scelti da un insieme allocato appositamente

Come regola generale, gli indirizzi permanenti si assegnano a server mentre quelli dinamici ad host generici (assegnati solo per un tempo predeterminato, "**lease**").
Quando il periodo di lease scade il server considera come libero quell'indirizzo, l'host può quindi rinunciarvi o rinegoziare con il server un'estensione del periodo

di lease (solitamente approvate). Questo non vuol dire che DHCP non possa essere configurato per *negare* sempre le suddette richieste.

In caso di perdita o duplicazione di pacchetti DHCP non vi è alcun errore, se un host non riceve *response* re-invia la *request*, se invece riceve duplicati li ignora.
Quando un host trova un server DHCP lo memorizza per utilizzi futuri. Non è possibile che due *DHCP request* arrivino in contemporanea.

6.5.1 Formato dei messaggi



Versione leggermente modificata dei messaggi BOOTP

- **OP** (8 bit)
  Indica se il messaggio è *Request* o *Response*
- **HTYPE** (8 bit)
  Indica il tipo HW della rete
- **HLEN** (8 bit)
  Indica la lunghezza dell'indirizzo HW
- **FLAGS** (16 bit)
  Indica se l'host può ricevere *messaggi broadcast* o *risposte dirette*

- **HOPS** (8 bit)
  Indica a quanti server rigirare la richiesta
- **TRANSACTION IDENTIFIER** (32 bit)
  Indica il valore, usato da un host, per capire se una risposta si riferisce ad una sua richiesta
- **SECONDS ELAPSED** (16 bit)
  Indica quanti secondi sono passati dall'avvio dell'host

I campi finali si usano per trasportare le informazioni di risposta verso la sorgente

- **YOUR IP ADDRESS** (32 bit)
  Se un host non conosce il proprio IP, il server usa questo campo per indicarglielo
- **SERVER IP ADDRESS** (32 bit)
  Fornisce l'IP del server DHCP contattato
- **SERVER HOST NAME** (512 bit)
  Fornisce l'Host Name del server DHCP contattato
- **ROUTER IP ADDRESS** (32 bit)
  Indirizzo IP del router di default

6.6 Carenza di Indirizzi IP

Il protocollo IP usa 32 bit per l'indirizzamento, all'epoca della sua introduzione sembravano più che sufficienti, ora invece la crescita esponenziale di Internet ha posto il problema della loro carenza.

6.6.1 NAT

Lo **IETF (Internet Engeneering Task Force)** ha definito alcuni range di indirizzi da utilizzare solo in ambito privato, ogni volta che un pacchetto con *IP privato* viene consegnato ad un *router pubblico* viene segnalato un errore.
Gli **Indirizzi IP privati** vengono utilizzati in reti con un solo (o pochi) punto d'accesso ad Internet. Viene per questo introdotto il **NAT (Network Address Translation)** con il fine di risolvere i problemi di instradamento tra *reti ad indirizzamento pubblico* e *reti ad indirizzamento privato.*

Datagrammi Uscenti NAT sostituisce *l'indirizzo sorgente* dei datagrammi uscenti con il proprio *IP pubblico*

Datagrammi Entranti NAT sostituisce *l'indirizzo di destinazione* dei datagrammi entranti con l'*IP privato* dell'host corretto

6.6.1.1 NAT Table

Il router abilitato NAT (router di confine) mantiene al suo interno una tabella in cui tiene traccia dei mapping tra *IP privato* delle sorgenti ed *IP pubblico* delle destinazioni. Questa tabella può essere aggiornata in due modi

- **Manualmente**: Il gestore della rete configura staticamente i record della NAT Table
- **Dinamicamente**: Ogni volta che un datagramma uscente passa per il router NAT viene creato un record della tabella. Questi vengono cancellati con un meccanismo di *timeout*

Il NAT basato unicamente su indirizzo non permette a diversi host privati di connettersi contemporaneamente allo stesso host pubblico

6.6.2 NAPT

Il **NAPT (Network Addres and Port Translation)** è un evoluzione nel NAT che permette al router di agire da *gateway di livello 4*, traducendo sia IP che numero di Porta. Il funzionamento è analogo a quello di NAT.

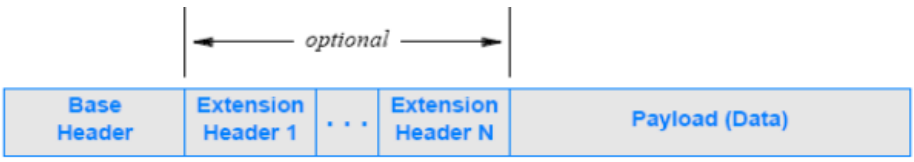
6.6.3 IPv6

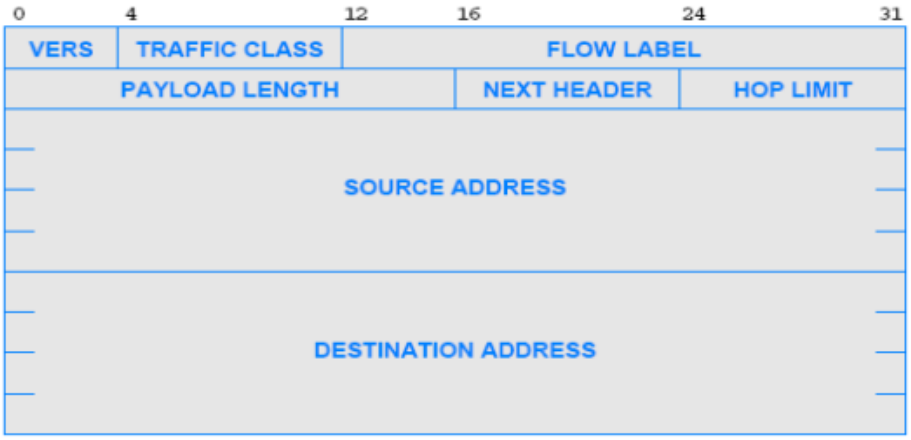
Mantiene molte caratteristiche di IPv4 (attuale implementazione), tra cui

- Connectionless
- Hop massimi per ogni datagramma

Molte funzionalità, però, sono cambiate.
Gli indirizzi IPv6 sono formati da **128 bit**, hanno un formato dell'header diverso e introduce il concetto di "**Extension Header**", un datagramma IPv6 quindi è formato da un *header*, *extension header* (facoltativo) ed i dati inviati.
Questa versione del protocollo introduce il supporto del traffico Real-Time, ossia viene introdotto un meccanismo che permette di creare un cammino tra sorgente e destinazione, potendo associare i datagrammi a tale cammino (per applicazioni che richiedono maggiore qualità del servizio).

6.6.3.1 Formato dei Datagrammi





Header di Base

- **VERS** (4 bit)
  Versione 6
- **TRAFFIC CLASS** (8 bit)
  Classe di traffico in base al tipo di traffico, specifica i requisiti che la rete fovrebbe soddisfare
- **PAYLOAD LENGHT** (16 bit)
  Dimensione del payload
- **HOP LIMIT** (8 bit)
  Corrisponde al TIME-TO-LIVE di IPv4
- **FLOW LABEL** (20 bit)
  Associa un datagramma ad un cammino specifico
- **NEXT HEADER** (8 bit)
  Specifica il tipo di informazione che segue l'header corrente, se ho *Extension Header* ne indica il tipo, se non ne ho indica il tipo di dati nel payload
- **SOURCE/DESTINATION ADDRESS** (128+128 bit)
  Indirizzo sorgente e destinazioen del datagramma

Extension Header

- **NEXT HEADER** (8 bit)
  Esiste un valore unico per ogni header possibile, indica il valore del prossimo header
- **HEADER LEN** (8 bit)
  Dimensione dell'Extension Header, può variare di header in header

Perchè inserire header multipli? In quanto si risparmia spazio (vengono utilizzate sempre solo un sottinsieme delle funzionalità), si possono inoltre aggiungere funzionalità senza che ogni header abbia informazioni al riguardo pur non utilizzandole.

6.6.3.2 Frammentazione

Simile alla frammentazione IPv4.
Come in IPv4

- Il prefisso del datagramma originale è copiato in ogni frammento
- La dimensione varia in base alla MTU della rete

Diversamente da IPv4

- Non esistono campi dell'header riguardanti la frammentazione, queste vanno infatti aggiunte in un extension header di tipo "*frammentazione*"

6.6.3.3 Indirizzamento

Come con CIDR, la divisione tra *prefisso* e *suffisso* è arbitraria. IPv6 introduce il concetto di **Gerarchia Multilivello**

- Il livello più alto è per gli ISP
- Livello successivo per organizzazioni
- Livello successivo per siti specifici
- E così via fino agli host locali

Indirizzi Speciali

- **Unicast**: Corrisponde ad un singolo host. Datagramma instradato verso tale indirizzo segue il cammino minimo
- **Multicast**: Corrisponde ad un insieme di host che possono variare in qualsiasi momento. Viene consegnata una copia del datagramma ad ogni membro dell'insieme
- **Anycast**: Corrisponde ad un insieme di host che condividono lo stesso prefisso. Il datagramma viene consegnato ad uno qualsiasi degli host dell'insieme

Notazione Esadecimale Nell'indirizzo IPv6 ciascun gruppo di *16 bit* viene separato dal carattere ":". Le sequenze di zeri si possono comprimere

Esempio: FF0C:0:0:0:0:0:0:B1 $\rightarrow$ FF0C : : B1

Questo facilita la transizione, infatti per trasporre gli indirizzi IPv4 esistenti basta porre a 0 i primi *96 bit*

7 Livello Data-Link

L'obiettivo del livello **Data-Link** è quello di fornire al livello di Rete una connessione il più possibile affidabile tra due macchine adiacenti.
Per *Macchine Adiacenti* si intende connesse da un canale di comunicazione, sia esso fisico o wireless. Condizione necessaria è però che questo canale sia quella che possiamo definire una "*pipe digitale*", ossia che l'ordine dei bit venga rispettato e quindi essi arrivino a destinazione come sono stati inviati.

7.1 Tipologia di servizi offerti al livello superiore

- **Servizio Connectionless senza Acknowledge**
  Nessuna connessione viene attivata e non si attende alcun *feedback* dalla destinazione. Se una trama viene persa non ci sono tentativi per recuperala, questo compito viene lasciato ai livelli superiori.
- **Servizio Connectionless con Acknowledge**
  Nessuna connessione viene attivata ed ogni trama inviata viene riscontrata in maniera individuale.
- **Servizio Connection-Oriented con Acknowledge**
  Viene attivata una connessione e, al termine del trasferimento, essa viene abbattuta. Anche in questo caso ogni trama viene riscontrata in maniera individuale.

7.2 Funzionalità del Livello Data-Link

- **Framing**
  Delimitazione delle trame
- **Rilevazione/Gestione Errori**
  Viene effettuato un controllo sul contenuto della trama, in caso vi fossero errori gestisce il recupero della stessa
- **Controllo di Flusso**
  Gestisce la velocità di trasmissione

7.2.1 Framing

Bisogna tenere conto del fatto che la lunghezza dei pacchetti ricevuti del *Livello di Rete* e delle corrispondenti trame di *Livello Data-Link* può variare. Inoltre, dato che il *Livello Fisico* tratta solo bit non è possibile distinguere dove termina una trama e dove comincia la successiva.
Il **framing** si occupa proprio di rendere distinguibile una trama dall'altra attraverso l'utilizzo di opportuni codici all'inizio o alla fine di ogni *frame*.

7.2.1.1 Modalità di Framing

Ho più modi per implementare il *framing*:

- Posso inserire intervalli temporali tra trame consecutive
  - Problema: Per natura delle reti non viene data alcuna garanzia sul rispetto delle caratteristiche temporali, quindi l'intervallo inter-trama potrebbe essere esteso o ridotto generando problemi di ricezione.
- Posso marcare l'inizio e la fine di ogni trama
  - Character Count
  - Bit Stuffing

Character Count Un campo nell'header del frame indica il numero di caratteri nel frame stesso

Bit Stuffing Ogni trama ha un numero arbitrario di bit. In questa modalità di *framing* ogni trama inizia con un particolare pattern (*01111110*) chiamato *Byte di Flag*.
Sorge spontaneo un dubbio, come mi comporto se la trama contiene già la sequenza di bit 01111110? In questo caso, ossia se vengono trovati cinque 1 consecutivi, viene aggiunto uno 0. Questo 0 aggiuntivo verrà poi tolto al momento della ricezione.

7.2.2 Rilevazione degli errori

Posso avere diverse tipologie di errori, essi possono essere:

- Errori sul singolo bit
- Replicazione di bit
- Perdita di bit

Al fine di rilevare questi errori è stato inserito un campo dell'header detto **checksum**. Esso è il risultato di un calcolo fatto utilizzando i bit della trama. La destinazione, una volta ricevuta la trama ripete il calcolo, se il valore ottenuto coincide con quello nel campo checksum allora la trama è corretta.

7.2.3 Gestione del Flusso

Può capitare che la sorgente invii trame ad una velocità superiore a quella con cui la destinazione può riceverle. La conseguenza diretta di questo fatto è una congestione del nodo destinazione. Il *Controllo di Flusso* è implementato in modo da controllare la velocità di trasmissione della sorgente in base al feedback inviatigli dalla destinazione. Questo feedback può dare istruzione di:

- Bloccare la trasmissione fino al comando successivo

- Limitare la quantità di informazione inviata in base a quanta ne è ancora in grado di gestire la destinazione

Nelle reti **TCP/IP** il controllo di flusso e di recupero errori è affidato ai livelli superiori

8 Il sotto-livello MAC

Il mezzo fisico, con cui il *Livello Data-Link* si interfaccia, è collegato anch'esso tramite un mezzo fisico. Questo può essere:

- *Dedicato* (reti punto-punto)
- *Condiviso* (reti broadcast)

Se il mezzo fisico è condiviso nascono delle problematiche legate all'accesso, bisognerà infatti selezionare l'host che ha diritto di trasmettere sul mezzo condiviso, creando una situazione di contesa per la risorsa trasmissiva.
Viene per questo introdotto un sotto-livello al Data-Link che gestisce queste problematiche: il **sotto-livello MAC (Medium Access Control)**.

Per mezzo condiviso si intende che un singolo canale trasmissivo può essere utilizzato da più sorgenti. Naturale quindi comprendere il fatto che se ogni partecipante alla comunicazione inviasse informazioni contemporaneamente nessuna di queste sarebbe distinguibile dalle altre.
Necessitiamo quindi una serie di regole per poter utilizzare il mezzo, dette *Tecniche di Allocazione del Canale*.

8.1 Tecniche di Allocazione

Esistono due categorie di tecniche:

- **Allocazione Statica**
  Il mezzo trasmissivo viene "*partizionato*" ed ogni porzione viene data alle diverse sorgenti. Questo può avvenire in base:
  - al tempo: Ogni sorgente ha a disposizione il mezzo per un dato periodo di tempo
  - alla frequenza: Ogni sorgente ha a disposizione una determinata frequenza
- **Allocazione Dinamica**
  Il canale viene allocato di volta in volta a chi ne fa richiesta e può essere utilizzato da un'altra sorgente solo dopo che la prima finisce di utilizzarlo e lo libera.

8.1.1 Allocazione Statica

Vale la pena usare questa tecnica di allocazione nel caso un cui ci si trovi con pochi utenti e questi abbiano grossi carichi di informazioni da inviare che però sono costanti nel tempo. Se le risorse vengono date ad utenti che non le utilizzano, quindi non inviano niente, queste sono effettivamente *perse*.

8.1.2 Allocazione Dinamica

Il canale trasmissivo può essere assegnato:

- **A turno**: Viene distribuito il "*permesso*" di trasmettere, la durata viene decisa dalla sorgente
- **A contesa**: Ogni sorgente prova a trasmettere indipendentemente dalle altre

Nel promo caso si sottintende la presenza di un *Overhead di Gestione*, ossia di un meccanismo per l'assegnazione del permesso per trasmettere. Nel secondo caso, invece, non c'è nessun meccanismo di assegnazione al fine di mantenere sorgente e destinazione i più semplici possibile. La *Trasmissione a Contesa* è generalmente la più utilizzata.

8.2 Protocolli di Accesso Multiplo

Principali algoritmi usati dai protocolli:

- **ALOHA**
  - Pure ALOHA
  - Slotted ALOHA
- **Carrier Sense Multiple Access Protocols**
  - CSMA
  - CSMA-CD (Collision Detection)

8.2.1 Pure ALOHA

Definito nel 1970 dall'Università delle Hawaii
Algoritmo:

- Una sorgente può trasmettere una trama ogniqualvolta vi sono dati da inviare (*continuous time*)
- La sorgente rileva, ascoltando il canale, l'eventuale collisione
- Se avviene una collisione, la sorgente aspetta un tempo casuale per poi ritrasmettere la trama

Il **Periodo di Vulnerabilità** per il Pure ALOHA, che è l'intervallo di tempo in cui possono avvenire delle collisioni, e 2T (con T = *Tempo di Trama*, ossia il tempo necessario per trasmettere una trama).

8.2.2 Slotted ALOHA

Proposto nel 1972 da Roberts per duplicare la capacità di Pure ALOHA.
L'algoritmo si basa sulla divisione del tempo in intervalli discreti. La differenza sta proprio in questo, infatti esso si comporta come il Pure ALOHA ma una trama può essere inviata solo ad intervalli di tempo definiti, è dunque necessario che vi sia sincronizzazione tra due stazioni.
Nella pratica segue quest'algoritmo:

- Quando un nodo ha una trama da spedire aspetta l'inizio dell'intervallo di tempo successivo, per poi inviarla per intero
- Se non avvengono collisioni, l'operazione ha avuto successo
- Se invece si verifica una collisione, il nodo la rileva prima della fine dello slot e ritrrasmette con probabilità p negli slot successivi fino a che non avrà successo, con $p \in [0, 1]$

Di conseguenza il **Periodio di Vulnerabilità** è T (*Tempo di Trama*).

8.2.3 Carrier Sense Multiple Access (CSMA)

Ci troviamo ora nell'ambito delle LAN, dove le stazioni possono monitorare lo stato del canale di trasmissione. Le stazioni ascoltano il canale *prima* di iniziare a trasmettere per verificare se sta avvenendo una trasmissione.
Algoritmo:

- Se il canale è libero si trasmette
- Se esso è occupato, sono possibili diverse varianti
  - **Non-persistent**: Rimanda la trasmissione ad un nuovo istante, scelto in maiera casuale
  - **Persistent**: Nel momento in cui il canale torna libero la stazione inizia a trasmettere
- Se avviene una collisione si comporta come ALOHA

Vi sono alcune varanti del CSMA, che verranno illustrate nelle sezioni a seguire

8.2.4 CSMA P-Persistent

Il tempo viene suddiviso in intervalli, la misura degli intervalli coincide con il periodo di vulnerabilità, ossia 2τ.
Algoritmo:

1. Ascolta il canale
   - Se il canale è libero:

        - Si trasmette con probabilità P
        - Se si è deciso di trasmettere, si passa al punto 2
        - Se si è deciso di non trasmettere si attende un intervallo di tempo e si torna al punto 1
    - Se è occupato si attende un intervallo di tempo e si torna al punto 1
2. Se c'è collisione
    - Si attende un tempo casuale e si torna al punto 1

Il periodo di vulnerabilità è legato al *Ritado di Propagazione* (τ). Infatti, se una stazione ha iniziato a trasmettere ma il suo segnale non è ancora arrivato a tutte le stazioni, qualcun altro potrebbe iniziare la trasmissione. In genere il *Ritardo di Propagazione* è molto inferiore al *Tempo di Trama* (T)

8.2.5 CSMA con Collision Detection (CSMA-CD)

Rispetto al P-Persistent esso migliora in quanto riesce a rilevare le collisioni e bloccare la trasmissione immediatamente. In questo modo non vi è spreco di tempo. Dopo aver interrotto la trasmissione invia una sequenza, detta *Sequenza di Jamming*, per segnalare alle altre stazioni l'avvenuta collisione.

8.3 LAN Estese

La rappresentazione tipica di una LAN è una serie di stazioni (PC) connesse ad un segmento di cavo (Bus). Poichè questo segmento di cavo non può essere troppo lungo, nel tempo, è nato il problema di estendere le LAN. Per fare questo si sfruttano tre tipi di apparati:

- Repeater o Hub
- Bridge
- Switch

8.3.1 Dominio di Collisione

Parti di rete per cui, se due stazioni trasmettono contemporaneamente, il segnale ricevuto da queste risulta danneggiato.

8.3.2 Dominio di Broadcast (Segmento Data-Link)

Parte di rete raggiunta da una trama con indirizzo di broadcast, se due stazioni appartengono alla stessa rete di Livello Data-Link, due stazioni hanno lo stesso dominio di broadcast.

8.3.3 Repeater o Hub

Apparato che interviene solo a Livello Fisico. Esso replica le trame in arrivo da un segmento ad un altro, amplificando il segnale.
Se un Repeater collega *due o più segmenti* si parla di **Hub**, che copia la trama ricevuta e la ripete su tutte le porte.

8.3.4 Bridge

Collega due segmenti di rete. A differenza delle Repeater, il Bridge è un apparato intelligente.
Seleziona, basandosi su una tabella da lui stesso mantenuta in cui si tiene traccia del segmento di rete di appartenenza per ciascuna rete, se ripetere una trama generata da un segmento di rete su un altro segmento di rete. Grazie al Bridge si può spezzare il dominio di collisione.

8.3.5 Switch

Può essere visto come un Bridge multiporta. Spesso ad ogni porta è associata una ed una sola stazione. Riesce quindi a realizzare un accesso dedicato per ogni nodo, eliminando le collisioni potendo quindi sopportare conversazioni contemporanee multiple.

8.4 Incapsulamento IP

Il Livello Data-Link svolge una serie di funzionalità che permettono il trasferimento *hop-by-hop*. Ognuno di questi hop può avere un protocollo di Livello 2 diverso da quello dell'hop successivo.
Esistono diverse modalità di incapsulamento IP:

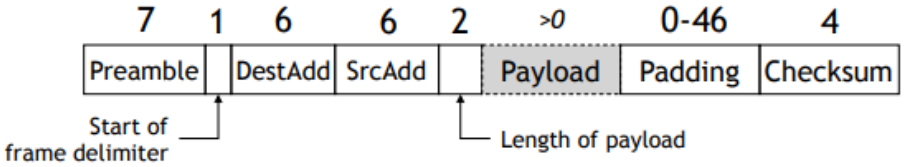
- Soluzioni per l'accesso
  - Ethernet e 802.3
  - PPP
    - PPP con Modem
    - PPP con ADSL
- Soluzioni per il backbone
  - Frame Relay
  - ATM
  - Soluzioni su SDH

8.4.1 Ethernet e 802.3

Si interfaccia direttamente con il Livello Fisico sopportando un carico del 30% (3 Mb/s) con picchi del 60% (6 Mb/s). A livello statistico, sotto carico medio, il 2/3% collide una volta e solo qualche pacchetto su 10000 ha collisioni multiple. Gli standard Ethernet e 802.3 implementano un livello MAC di tipo *CSMA/CD 1-persistent*. In caso di collisione, l'istante in cui ritrasmettere viene calcolato tramite un algoritmo di *Binary Exponential Backoff*

- Dopo le collisioni, l'host attende un intervallo di tempo casuale prima di ri-trasmettere, estraendo un numero casuale nell'intervallo [0, 1, ..., $2^i - 1$]
- Vincoli
  - Dopo 10 collisioni, tempo d'attesa limitato tra [0, 1, ..., 1023]
  - Dopo 16 collisioni, viene riportata una **failure** al Sistema Operativo

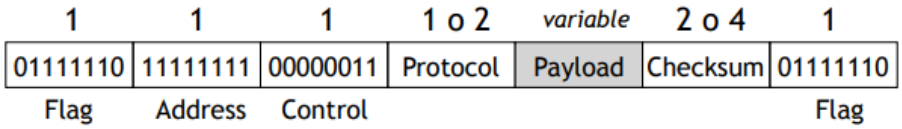
Il formato della trama è il seguente



- **Preamble** (7 Byte)
  Si usa per sincronizzare il ricevitore
- **Start of Frame** (1 Byte)
  Flag di inizio trama
- **Destination Address** (6 Byte)
  Indirizzo di destinazione della trama
- **Source Address** (6 Byte)
  Indirizzo sorgente della trama
- **Length** (2 Byte)
  Lunghezza in Byte della trama, tra 0 e 1500 (se > 1500 indica *Protocol Type*)
- **Payload**
  Informazione Trasmessa
- **Checksum**
  Codice per rilevamento errori

8.4.2 PPP

PPP viene utilizzato sia nell'accesso che nel backbone. Esso è un protocollo *Charachter Oriented* e utilizza il *Bit Stuffing* per il framing.
Il formato della trama è il seguente



- **Flag** (1 Byte)
  Identifica inizio e fine della trama (*Bit Stuffing*)
- **Address** (1 Byte)
  Utilizzato in configurazione "*Tutti gli Host*"
- **Control** (1 Byte)
  - Valore predefinito 00000011 con il significato di *unnumbered*
  - Di default non fornisce un sevizio affidabile in quanto le richieste di ritrasmissione e rimozione duplicati sono lasciate ai livelli superiori
- **Protocol** (1 o 2 Byte)
  Identifica il tipo di livello della trama
- **Payload**
  Informazione trasmessa
- **Checksum** (2 o 4 Byte)
  Identificazione errori

L'accesso può avvenire con Modem o xDSL

- Modem: Utilizza la banda telefonica, ha un limite superiore di *56k*
- xDSL (Digital Subscriber Line): Famiglie di tecnologie che permettono di utilizzare la banda disponibile del doppino telefonico. Possono esserci sistemi *Simmetrici e Asimmetrici*.
  Per esempio, *ADSL* e un Sistema Simmetrico su Singola Coppia, è *Rate Adaptive*, va dai 640 agli 8200 Kb/s in upstream e fino a 512 Kb/s in upstream.

8.5 Address Resolution Protocol

Ogni scheda di rete ha un indirizzo univoco associato chiamato indirizzo MAC. Questo viene creato e assegnato al momento della produzione della scheda di rete. Necessitiamo dell'indirizzo MAC per la *consegna diretta/indiretta*.

8.5.1 Consegna Diretta

Sorgente e destinazione appartengono alla stessa rete e quindi il pacchetto può essere inviato direttamente alla destinazione

8.5.2 Consegna Indiretta

Sorgente e destinazione non appartengono alla stessa rete, la sorgente manda le trame al router che si farà carico della consegna. Per consegnare le trame al router si utilizza l'indirizzo MAC.
Il router riceve una trama che contiene come indirizzo MAC di destinazione quello del router. Il Livello Data-Link estrae il payload e passa al Livello di Rete.
Il router controlla l'IP di destinazione, se non è il suo indirizzo IP allora va consegnato al router successivo. Per fare questo controlla le tabelle di routing per trovare il *next-hop*. Incapsula quindi il pacchetto in una nuova trama con indirizzo MAC di destinazione quello del prossimo router.

8.5.3 Risoluzione degli indirizzi

Dato un indirizzo IP come posso risalire al suo MAC? Utilizzo **ARP (Address Resolution Protocol)**.
Un host può risolvere l'indirizzo di un altro host solo se entrambi sono collegati alla stessa rete fisica.
Supponiamo che un host A voglia risolvere l'indirizzo IP di un host B:

- A invia una richiesta in broadcast che chiedendo l'indirizzo MAC dell'host con indirizzo IP:B
- Il broadcast (Limitato) viaggia solo sulla rete locale, quindi raggiungerà tutti gli host della rete
- Quando l'host B riceverà la richiesta risponderà direttamente ad A (quindi non in broadcast) dandogli il proprio indirizzo MAC

8.5.4 ARP Caching e Processing dei Messaggi

Inviare una richiesta ARP per ogni datagramma sarebbe inefficiente in quanto avremmo tre trame che viaggiano per ogni datagramma: una richiesta ARP, una risposta ARP e la trama con i dati.
Per ridurre il traffico nella rete, ARP estrae e salva le informazioni delle risposte ARP, queste verranno mantenute in memoria tramite una tabella. Questa viene gestita come una cache, ogni associazione IP $\rightarrow$ MAC viene aggiornata quando si riceve una risposta, se la tabella è piena si rimuove l'entry più vecchia. Se invece una entry non viene aggiornata per molto tempo essa viene rimossa.
Prima di inviare una richiesta ARP, ogni host controlla che non ci sia già una entry per l'associazione che sta cercando nella tabella ARP, se l'informazione è presente l'associazione viene utilizzata senza inviare alcuna richiesta, altrimenti

viene inviata la richiesta ed al momento della risposta la cache viene aggiornata e si utilizza l'informazione appena ottenuta per inviare la trama.

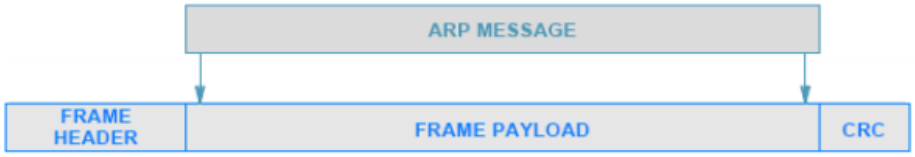
8.5.5 Formato dei messaggi

ARP non risolve solo IP e MAC, lo standard è generale e specifica messaggi diversi in base ai protocolli coinvolti.

- **HARDWARE ADDRESS TYPE** (16 bit)
  Tipo di indirizzo HW utilizzato (= 1 se Ethernet)
- **PROTOCOL ADDRESS TYPE** (16 bit)
  Tipo di indirizzo del protocollo utilizzato (0x0800 se IPv4)
- **HADDR LEN** (8 bit)
  Dimensione in Byte dell'indirizzo HW (= 6 se Ethernet)
- **PADDR LEN** (8 bit)
  Dimensione in Byte del protocollo (= 4 se IPv4)
- **OPERATION** (16 bit)
  Specifica se il messaggio è una *Request* (1) o una *Response* (2)
- **SENDER HADDR**
  Indirizzo HW della sorgente
- **SENDER PADDR**
  Indirizzo del protocollo della sorgente
- **TARGET HADDR**
  Indirizzo HW del target
- **TARGET PADDR**
  Indirizzo del protocollo del target

Quando viene inviata una richiesta il campo TARGET HADDR è composto da zeri in quanto la sorgente non conosce quello della destinazione.
Quando viene inviata una risposta il target si riferisce all'host che aveva originato la richiesta e quindi non serve a niente.

8.5.6 Trasporto dei messaggi



Quando viaggiano sulla rete fisica i messaggi ARP vengono rinchiusi in una trama, quindi il protocollo ARP viene considerato come dei dati trasportati dal Livello Data-Link.
Nell'header della trama esiste un campo "*type*" che indica il tipo di trama (per Ethernet il valore 0x806 denota i messaggi ARP). Il campo avrà lo stesso valore sia per le richieste che le risposte, andrà dunque controllato il valore del campo *OPERATION* nel messaggio ARP.

8.6 Le Reti 802.11

8.6.1 Architettura di riferimento delle WLAN

Con *Station* (STA) si indica un terminale con capacità di accesso al messo wireless.
Il *Basic Service Set* (BSS) è l'insieme dei terminali che usano le stesse frequenze.
Per *Access Point* si intende una stazione integrata sia nella WLAN che nel "*Distribution System*".
Il *Portal* è un Bridge verso le altre reti.
Il *Distribution System*, infine, è una rete di interconnessione per formare una unica rete logica partendo da diverse BSS.

Nello specifico il BSS è formato da un insieme di terminali con lo stesso protocollo MAC che competono per l'accesso allo stesso mezzo condiviso. Questo può essere isolato o no, in questo caso esso sarebbe collegato ad un *Distribution System* attraverso un *Access Point* che funziona come un Bridge.

8.6.2 Il Livello MAC nelle reti 802.11

L'obiettivo è quello di modificare il CSMA al fine di evitare le collisioni, per questo è stato creato il **CSMA/CA** (Collision Avoidance).

Distributed Coordination Function (DCF) Basata sul CSMA/CA, utilizza un algoritmo per la risoluzioni delle contese, tramite questo è possibile dare accesso a tipi diversi di traffico. Il traffico ordinario si appoggia direttamente su DCF.

Point Coordination Function (PCF) Supporta il traffico Real-Time. Si basa sul "*polling*" coordinato da un *Centralized Point Coordinator*. PCF è costruito su DCF e sfrutta le sua funzionalità per dare accesso agli utenti.

Entrambi possono funzionare contemporaneamente sulla stessa BSS, fornendo alternatamente periodi con e senza contesa. Dei due PCF è il meno usato.

8.6.3 Time Slots

Il tempo viene diviso in intervalli, chiamati *Slot*, essi sono le unità di tempo del sistema e variano in base all'implementazione del Livello Fisico.
Le stazioni si sincronizzano

- nella modalità "*infrastructure*, con l'*Access Point*
- nella modalità "*ad hoc*, tra di loro

8.6.3.1 Inter-Frame Space

Intervallo di tempo tra la trasmissione di trame, viene usato per stabilire dei livelli di priorità nell'accedere al canale. La durata dipende dall'implementazione fisica.

Esistono 4 tipi di *IFS*:

- **SIFS**: Short IFS
- **PIFS**: Point coordination IFS (> SIFS)
- **DIFS**: Distributed IFS (> PIFS)
- **EIFS**: Extended IFS (> DIFS)

Point coordination IFS : Priorità media per servizi Real-Time che usano PCF. Il valore è dato da *SIFS + 1 time-slot*. Usato dal controller centrale nel PCF durante il polling.

Distributed IFS : Priorità più bassa per il servizio di invio asincrono. Valore dato da *SIFS + 2 time-slot*. Usato dalle trame per l'invio asincrono con ritardo minimo nel caso di contesa del canale.

8.6.4 DCF con CSMA/CA

Una stazione con dei dati da trasmettere ascolta in canale

1. Se il canale è libero
   - Continua ad ascoltare per capire se il mezzo rimane libero per un tempo pari a DIFS. I questo caso ai può trasmettere subito.
2. Se il canale è occupato
   - La stazione continua a monitorare il mezzo fino a quando la trasmissione corrente è finita. Questo avviene sia perchè il mezzo risulta occupato fin dall'inizio sia perchè diventa occupato durante il DIFS.
3. Quando la trasmissione corrente è finita, la stazione aspetta un altro DIFS
   - Se, dopo il DIFS, il canale è nuovamente occupato, si torna al punto 2
4. Se il mezzo rimane libero per un DIFS
   - La stazione estrae un numero casuale di slot uniformemente distribuito all'interno di una **Contention Window** [0, CW-1]
     - *Contatore di Backoff*
   - Finchè il canale resta libero, la stazione decrementa il contatore man mano che il tempo passa. Se questo arriva a 0 la stazione trasmette.
5. Se il canale torna ad essere occupato prima che il contatore arrivi a 0
   - Il contatore viene congelato e si torna al punto 2. Al punto 4, però, non verrà estratto un nuovo contatore ma si continuerà con il valore congelato.

Per quanto concerne il *Contatore di Backoff*, esso è un intero che corrisponde al numero di time-slot.
Di fatto è una variabile uniformemente distribuita nell'intervallo [0, CW-1], dove CW è il valore della **Contention Window**, esso viene aggiornato ad ogni tentativo di trasmissione

- $i = 1$: $CW_1 = CW_{min}$
- $i > 1$: $CW_i = 2CW_{i-1}$ (per >1 si intende numero di tentativi consecutivi di trasmissione)
- $\forall i$: $CW_i \leq CW_{max}$

Ricapitolando, la prima stazione a cui scade il Contatore di Backoff inizia la trasmissione, le stazioni che percepiscono la trasmissione congelano il loro Backoff, che verrà fatto ripartire nel successivo periodo di contesa.

In caso di collisione, la CW_{max} raddoppia, quindi la lunghezza del tempo di Backoff aumenta esponenzialmente in caso di collisioni consecutive.
Si seleziona quindi un nuovo valore della CW randomicamente nell'intervallo [0, CW_{max}]

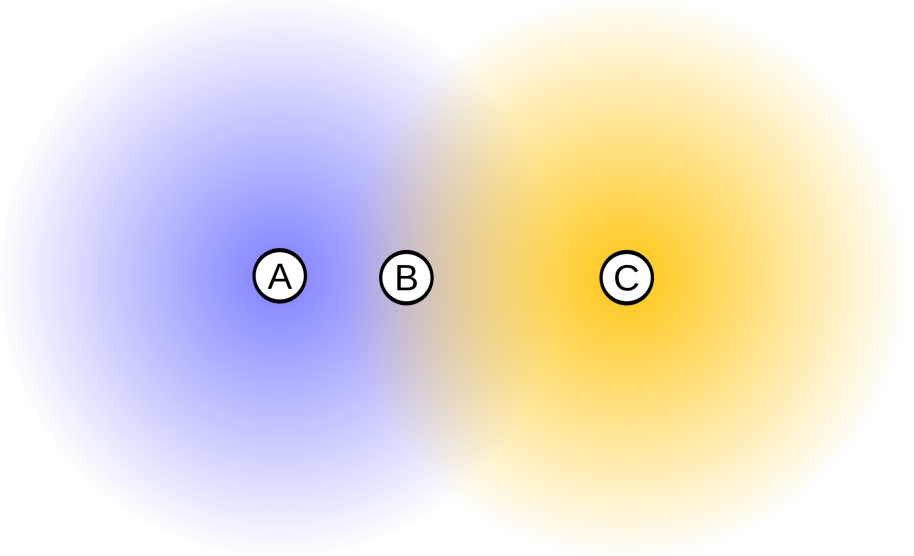
La stazione che riceve le trame manda un ACK immediatamente dopo la ricezione, senza ascoltare prima il mezzo (nella maggior parte delle implementazioni l'invio degli ACK è la sciato ai livelli superiori)

8.6.5 DCF con RTS/CTS

Introduciamo una serie di problematiche risolvibili con la metodologia *RTS/CTS*

8.6.5.1 Problema del Terminale Nascosto

Il segnale generato dalle stazioni (o dall'Access Point) è percepibile solo fino ad una certa distanza, questa dipende dalla potenza di emissione del segnale.
A causa di particolari disposizioni spaziali il segnale emesso da una stazione può essere percepito solo da un sottinsieme di stazioni. da qui nasce il *Problema del Terminale Nascosto*.



Descriviamo ora cosa succede precisamente

- A invia una trama a B
- C ascolta il canale, non rilevando però la trasmissione di A (essendo fuori dal suo range, *giallo* nel grafico)

- Pensando che il canale sia libero, C invia una trama a B, causando una collisione

8.6.5.2 Problema del Terminale Esposto

Sostanzialmente si tratta del problema opposto, una stazione non trasmette in quanto rileva una trasmissione tra altre due stazioni, che però non influenzerebbero la la sua.
Questo problema è dato dal posizionamento troppo ravvicinato delle stazioni nella sottorete.

Più precisamente

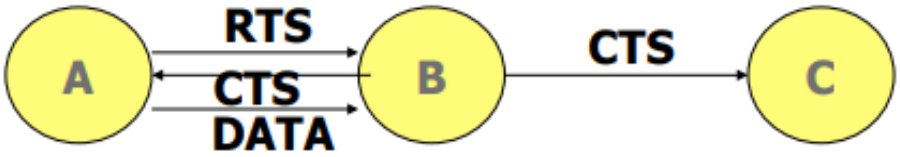
- B invia delle trame ad A e C vuole inviarne a D
- C ascolta il canale e rileva la trasmissione da B ad A
- C non trasmette nonostante egli possa farlo, dato che non disturberebbe in alcun modo la trasmissione

8.6.5.3 RTS/CTS

Con lo scopo di risolvere questi problemi sono stati introdotti *RTS* e *CTS*. La sorgente, dopo aver percepito il canale libero per un intervallo di tempo pari a DIFS, invia in broadcast una trama *RTS*, la stazione che si vedrà come destinatario risponderà con una trama *CTS* dopo un intervallo SIFS. A questo punto i dati possono essere trasmessi. Vediamo dunque, dopo questa introduzione, come vengono risolti i problemi del Terminale Nascosto ed Esposto

Problema del Terminale Nascosto

- A invia una trama RTS in broadcast
- B invia, come risposta, una trama CTS, a sua volta in broadcast
- A e C ricevono la trama CTS
- C, sapendo che verranno inviati dei dati a B, evita di trasmettere
- A può inviare dati a B senza preoccuparsi di possibili collisioni



Al fine di mettere a conoscenza le altre stazioni del tempo per cui intende trasmettere, A include la lunghezza della trasmissione nella trama RTS. Quest'informazione verrà poi inclusa anche nella trama CTS da B, in modo che tutte le stazioni che riceveranno la CTS sapranno per quanto dovranno attendere. La trama *DATA* verrà quindi inviata solo dopo aver "prenotato" il canale.

Problema del Terminale Esposto

- B invia una trama RTS in broadcast, questa verrà dunque percepita da A e da C
- A risponde a B con una trama CTS, che non verrà però sentita da da C, in quanto fuori range
- C assume quindi di non essere raggiungibile, quindi procede con la trasmissione a D

8.7 Network Allocation Vector

Nell'802.11 l'ascolto del canale è sia *fisico* che "*virtuale*". Se anche una sola delle due indica il canale come occupato, allora 802.11 lo considera tale.
L'ascolto *virtuale* è fornito dal **Network Allocation Vector (NAV)**.
La maggior parte delle trame 802.11 includono un campo *length* con la lunghezza della trama, i nodi che le percepiscono impostano quindi il NAV al tempo in cui si aspettano che il mezzo sia libero. Avendo NAV >0 il mezzo sarà considerato occupato.
Non dovendo restare costantemente in ascolto le WLAN possono così risparmiare energia.