RETI DI ELABORATORI

– Appunti –

UNIVERSITÀ
DEGLI STUDI
DI TORINO

Ramiro Calò

8 Gennaio 2023

Indice

Capitolo 1

Premessa e Disclaimer

Il presente documento è da intendersi esclusivamente a scopo didattico. Sottoposto a licenza Creative Commons con le specifiche di "Attribution - NonCommercial - NoDerivs", la violazione della quale comporterà l'avvio di procedure legali nei confronti di chi ha violato tale licenza. Si deve tenere presente che, sebbene il documento sia stato sviluppato tramite testi, persone, siti web e corsi di attendibilità e affidabilità non discutibile, il documento è stato sviluppato come appunti personali dell'autore e il contenuto potrebbe non essere totalmente corretto. Si coglie l'occasione per invogliare il lettore a comunicare eventuali errori, che verranno presi in considerazione e verificati, inviando una mail all'indirizzo: *ramiro.calo[at]edu.unito.it* o *rc.ramirocalo@gmail.com*.
L'obiettivo della pubblicazione e la distribuzione libera di questo documento è la diffusione di conoscenza libera e un piccolo tassello per aiutare a comprendere al meglio l'Informatica per una consapevolezza maggiore di quello che è oramai il nostro quotidiano utilizzo.
Nel caso questo documento ti sia stato condiviso tramite gruppi Telegram, Whatsapp o privatamente, ti invito a dare un'occhiata alla **Repository Github** a cui questo documento appartiene.
Si augura una buona lettura e si ringrazia il lettore per la fiducia nell'aver scelto questo documento.

Buon Lavoro e Grazie!

Capitolo 2

Introduzione

Il corso di Reti di Elaboratori studia gli elementi fondamentali delle tecnologie di trasmissione del livello data link, dei protocolli di accesso a mezzi condivisi e dei protocolli di trasmissione wireless, la suite di protocolli TCP/IP e i principi che guidano la strutturazione e la progettazione di applicazioni distribuite.

2.1 Cos'è Internet

Internet (INTER-NETWORKING) è il sistema ingegnerizzato più grande e complesso che l'uomo abbia mai creato. Internet può essere descritto su più livelli per essere descritto nel migliore dei modi, quindi avremo:

- Descrizione di componenti hardware e software che lo compongono
- Descrizione delle infrastrutture di rete che forniscono servizi ad applicazioni distribuite

Prendendo internet come esempio e caso studio si possono derivare tutti i concetti che caratterizzano una qualsiasi rete di calcolatori.

2.1.1 Internet e Componenti Base

Internet è una rete di reti di calcolatori, interconnette miliardi di dispositivi di calcolo in tutto il mondo. Connette talmente tanti dispositivi in aggiunta ai normali calcolatori che tutti utilizziamo, come auto, elettrodomestici ecc. da ciò si delinea una nuova definizione di internet: IoT - Internet of Things. Monitorare quanti dispositivi e come si connettono ad internet è una buona prassi da informatico, per questo lasciamo delle risorse:

- internetworldstats
- internetlivestats

I dispositivi connessi a internet prendono il nome di **Host** oppure **End-System** poichè sono i punti finali e più periferici di internet, vedremo come e perchè più avanti. Gli host sono connessi tra loro da una rete di collegamenti e apparati chiamati **commutatori di pacchetti o router**. I collegamenti possono essere effettuati per mezzi di vario tipo:

- Cavi Coassiali
- Fili di Rame
- Fibre Ottiche
- Onde Elettromagnetiche
- Intere sottoreti che utilizzano una tecnologia differente da internet

Dipendentemente dalla tecnologia di collegamento utilizzata si hanno differenti caratteristiche e velocità (TRANSMISSION RATE), le velocità sono misurate in bps (bit/secondo), mentre i ritardi vengono misurati in millisecondi. Internet inteso come rete di reti si descrive da sè come una rete (e una serie di collegamenti) tra end-system ovvero sistemi periferici, infatti ognuno di questi end-system è un'isola nella rete e nel momento in cui si vuole inviare dati da un host ad un altro il sistema suddivide i dati e aggiunge ad ognuna parte una intestazione, il risultato è uno strumento denominato **pacchetto**. Il sistema che ha generato i pacchetti li invia attraverso la rete per recapitarli all'host di destinazione e riassemblarli per avere il dato finale. Nel mezzo di questo viaggio vengono utilizzati due tipologie di commutatori:

- Router: generalmente nel cuore della rete
- Link-Layer Switch (o semplicemente Switch): questi vengono utilizzati all'interno delle reti di accesso

Dal sistema di invio a quello di ricezione i pacchetti compiono un viaggio denominato **cammino o percorso** (route o path).
Gli end-system accedono ad internet tramite gli ISP ovvero gli Internet Service Provider che altro non sono che una complessa struttura di router e switch ad alta velocità e di collegamenti fisici, spiegati precedentemente. Gli ISP di livello più basso sono connessi a quelli nazionali tramite collegamenti che arrivano fino al fruitore di contenuti mentre gli ISP di alto livello sono

formati da router ad alta velocità connessi da fibra ottica. Sia che sia di alto o basso livello gli ISP fanno uso di protocolli IP:

- **TCP**: Transmission Controll Protocol
- **IP**: Internet Protocol

L'insieme dei protocolli standard utilizzati in internet si chiamano TCP/IP, gli standard di internet sono molto importanti per il funzionamento del sistema e vengono pubblicati (aggiornati e mantenuti) da **IETF** (Internet Engineering Task Force) tramite gli **RFC** (Request For Comment)ovvero le pubblicazioni che spiegano gli standard fa adottare e come fare.

2.1.2 Internet come Servizi

Internet offre servizio ad applicazioni distribuite, ovvero offre l'infrastruttura necessaria alle applicazioni distribuite. Queste applicazioni sono applicazioni che coinvolgono più sistemi periferici, la parte più importante è che tali applicazioni vengono eseguite sugli host e non sui commutatori. Una tipologia di applicazioni distribuite è la tipologia client-server come può essere un browser. Per fare ciò le applicazioni dialogano con la rete attraverso le API (Application Programming Interface), le API sono delle regole che bisogna seguire per recapitare i dati in maniera corretta. Questo ci porta a capire che l'applicazione e il destinatario devono essere progettati per **cooperare**.

2.1.3 Protocolli di Rete

Come due esseri umani utilizzano un protocollo per comunicare e cooperare per un obiettivo, come ad esempio una persona chiede ad un'altra l'ora e riceve una risposta, in maniera del tutto simile avviene con le macchine. Un host esegue una richiesta e riceve una risposta, nello specifico:

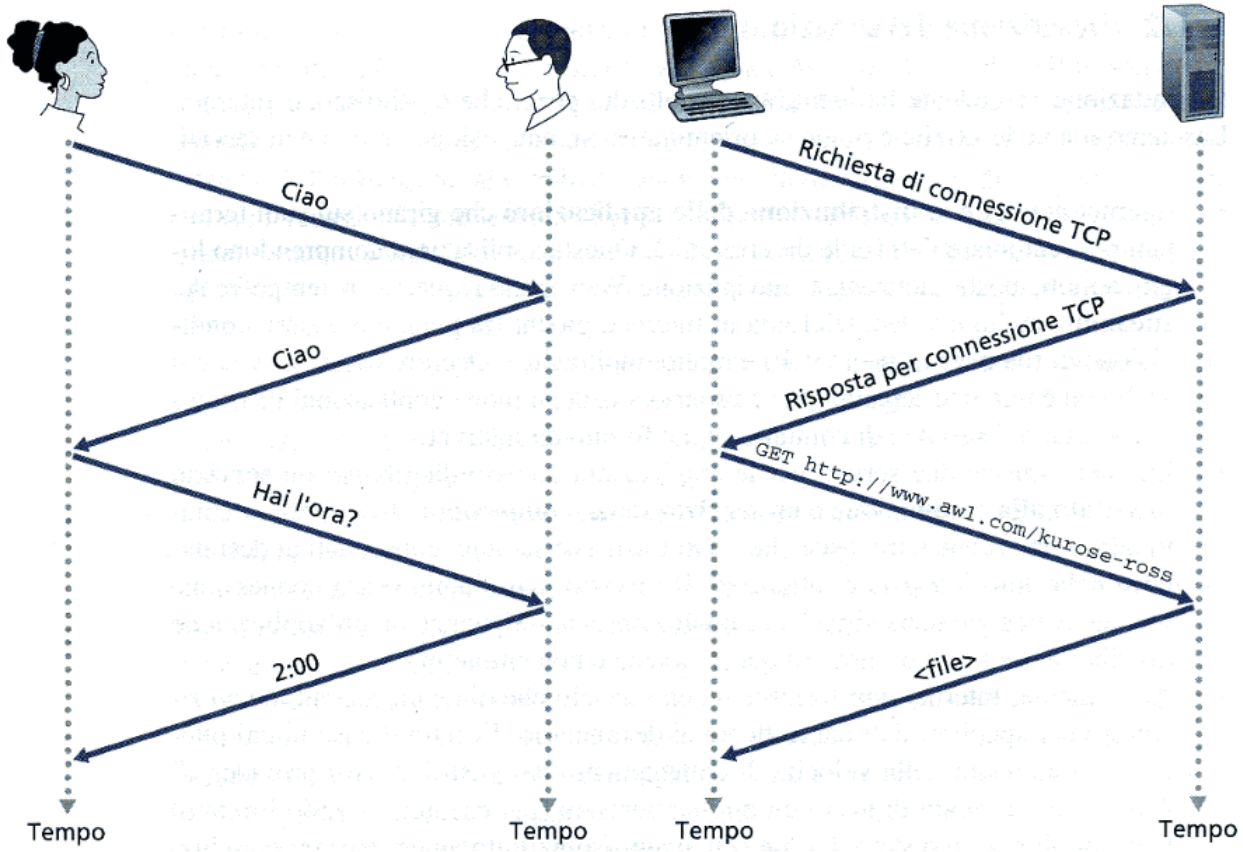


Figura 2.1: Il copyright dell'immagine è da attribuire a [1] in bibliografia

Un protocollo definisce il formato e l'ordine dei messaggi scambiati tra due o più host, così come le azioni intraprese in fase di trasmissione e/o ricezione di un messaggio da un altro evento.
Un protocollo di rete è quindi in primo luogo una specifica che contiene:

- Il formato dei messaggi scambiati dalla entità di protocollo
- L'ordine in cui questi messaggi vengono scambiati
- Le azioni che una entità deve compiere

2.1.4 Le tre componenti di Internet

- Network-Edge (end-system, clients, server, spesso organizzati in grossi Data Center)

- Reti di accesso, Mezzi fisici (cavi, fibre ottiche, wireless, ..)
- Network Core (router interconnessi da collegamenti, sotto-reti, rete di reti)

2.2 I confini della Rete

I dispositivi connessi ad internet sono i sistemi periferici chiamati anche host, questi si distinguono tra client e server, sebbene parlando di client e server ci si riferisce a calcolatori non è una definizione corretta, poichè il client e il server si riferiscono all'applicativo che sta eseguendo la connessione e che rispettivamente è eseguito su un computer che fruisce dei contenuti (client) e che fornisce contenuti (server).

2.2.1 Reti di Accesso

Introduciamo il concetto di **Edge-Router** (router di confine) che è il primo router sul percorso dal sistema di origine a un qualsiasi altro sistema di destinazione. Una rete di accesso è una rete che connette fisicamente un sistema al suo edge-router. Vi sono differenti tipi di rete di accesso:

- Residenziali
- Istituzionali
- Reti di accesso mobile

Reti di accesso residenziali

Di seguito spieghiamo come è formata e come funziona una rete di accesso residenziale:

- DSL: Accesso fornito dalla compagnia telefonica che è anche una ISP
- Il modem DSL usa la linea telefonica esistente per scambiare dati
- Il modem DSL converte i dati digitali in toni ad alta frequenza che possono essere trasmessi sul doppino telefonico fino alla centrale locale
- I segnali analogici vengono convertiti in digitali nel DSLAM (Digital Subscriber Line Access Multiplexer)

- Le linee trasportano dati e segnali telefonici codificandoli in tre bande di frequenza NON sovrapposte:

  Canale Downstream (Verso l'abitazione)

  Canale Upstream (Verso il DSLAM)

  Canale telefonico a due vie

La ADSL è l'acronimo di Asimmetric Digital Subscriber Line ovvero la downstream e l'upstream hanno velocità diverse.
Negli USA data la spaventosa diffusione del televisore si è scelto come mezzo trasmissivo dei dati e l'accesso a internet, oltre alle immagini televisive. Tale soluzione necessita di un Cable Modem, la controparte del DSLAM invece viene svolta dal CMTS (Cable Modem Terminator System). Il cavo utilizzato è il cavo coassiale. Negli USA vi è anche un'altra tipologia chiamata HFC (Hybrid Fiber Coax) un mix tra fibra ottica e cavo coassiale.

Reti di accesso istituzionali/aziendali

Nelle università e all'interno di aziende per connettere i sistemi periferici al router si utilizza la rete locale, altresì chiamata **LAN - local area network**. Esistono diverse tipologie di reti LAN ma la tecnologia che si preferisce è la **Ethernet** che utilizza un doppino di rame intrecciato per connettere numerosi sistemi periferici tra loro e ad uno switch Ethernet, che a sua volta viene collegato al router e quindi a Internet. L'accesso tramite questa tecnologia permette di raggiungere velocità che vanno da 100 Mbps fino anche a 1 o 10 Gbps. Una rete LAN viene anche implementata come tipologia Wireless, ovvero gli utenti trasmettono e ricevono pacchetti entro un raggio di qualche decina di metri da e verso un access point wireless, il quale è connesso a una rete Ethernet cablata, la tecnologia alla base della tipologia Wireless è il Wi-Fi conosciuto in termini tecnici come IEEE 802.11, che raggiunge una velocità intorno ai 100 Mbps.

Accesso su scala Geografica: Fibra, 3G, 4G

I dispositivi iPhone e Android utilizzano una infrastruttura wireless, la medesima utilizzata dalla telefonia cellulare per inviare/ricevere pacchetti tramite una stazione base gestita da un fornitore di telefonia cellulare. Differentemente dalla tecnologia Wi-Fi, un utente può trovarsi a chilometri dalla stazione base. Gli ISP, che in Italia corrispondono quasi esclusivamente alle compagnie di telecomunicazioni, hanno investito enormi somme nelle reti wireless di terza e quarta generazione (3G, 4G), che consentono accesso

wireless a Internet con commutazione a pacchetto e su scala geografica a velocità che superano 1 Mbps. La tecnologia LTE fonda le sue basi nella tecnologia 3G che può raggiungere velocità superiori a 10 Mbps. Nell'ultimo periodo l'evoluzione di questa tecnologia è arrivata alla generazione quinta o 5G, con velocità che raggiungono 1 Gbps ma che teoricamente dovrebbero potersi spingere fino a 10 Gbps.

2.2.2 Mezzi Trasmissivi

I mezzi trasmissivi possono presentarsi in differenti modi e dimensioni, e ogni coppia di trasmettitore-ricevitore lungo il percorso del pacchetto, può essere collegata da mezzi trasmissivi diversi. Tra i vari mezzi trasmissivi possiamo annoverare la fibra ottica, il cavo coassiale, il doppino intrecciato e lo spettro radio sia terrestre che satellitare. La distinzione più importante da fare nei mezzi trasmissivi è:

- **Mezzi Vincolati:** Le onde vengono contenute all'interno di un mezzo fisico come la fibra o il coassiale o il doppino
- **Mezzi Non Vincolati:** Le onde si propagano nell'atmosfera e nello spazio esterno

2.3 Il Nucleo della Rete - Core Network

Il nucleo della rete può essere visto come una fitta maglia di commutatori di pacchetti e collegamenti che interconnettono i sistemi periferici di cui abbiamo parlato.

2.3.1 Commutazione di Pacchetto

Le applicazioni distribuite si scambiano messaggi secondo un determinato protocollo. I messaggi possono avere una funzione di controllo o contenere dati. La sorgente suddivide i messaggi in pacchetti, che attraverso collegamenti e commutatori di pacchetti svolgono un viaggio da sorgente a destinazione. I pacchetti vengono trasmessi su collegamenti dei quali sfruttano la velocità totale di trasmissione del mezzo, quindi se un sistema periferico o un commutatore invia un pacchetto di L bit su un canale che ha velocità R bps, il tempo di trasmissione sarà pari a $\frac{L}{R}$ secondi.

Trasmissione Store-And-Forward

Quasi tutti i commutatori di pacchetto utilizzano la trasmissione **Store And Forward**. Questo significa che il router o il generico commutatore deve ricevere l'intero pacchetto prima di iniziare l'inoltro. Un router è solitamente dotato di molti collegamenti, infatti la sua funzione è proprio quella di instradare un pacchetto in entrata in un collegamento in uscita. Poniamoci in una situazione molto semplice dove il router o il commutatore generico abbia un solo collegamento in entrata e un solo collegamento in uscita, quindi il nostro commutatore solo dopo aver ricevuto tutti i bit del pacchetto può iniziare a trasmettere sul pacchetto in uscita. Ora che sappiamo l'infrastruttura su cui stiamo ponendo l'attenzione proviamo a calcolarci il tempo che intercorre da quando la trasmissione inizia dalla sorgente a quando il destinatario ha ricevuto in toto il pacchetto.
La sorgente ovviamente inizia a trasmettere a tempo 0, trascurando i ritardi di propagazione al tempo $\frac{L}{R}$ il pacchetto sarà stato ricevuto e memorizzato dal router. Quindi non appena si raggiunge questo stadio inizia l'inoltro sul collegamento in uscita, di conseguenza al tempo $2\frac{L}{R}$ avremo l'intero pacchetto ricevuto dal destinatario. Se il router non dovesse trattenere i bit che arrivano fino al completamento del pacchetto prima di eseguire l'inoltro allora avremo il tempo totale di $\frac{L}{R}$, il router però necessita di trattenere il router perchè deve eseguire una manipolazione prima di eseguire l'inoltro.
Generalizzando possiamo dire: avendo N collegamenti tra sorgente e destinatario (e di conseguenza abbiamo $N-1$ router tra sorgente e destinatario), con ogni collegamento che ha una velocità di R allora possiamo dire che un solo pacchetto di dimensione L avrà un tempo (o un ritardo) di propagazione pari a $d_{end\_to\_end} = N\frac{L}{R}$.

Ritardi di Accodamento e Perdita di Pacchetti

Un concetto importante all'interno di un commutatore di pacchetti è il **buffer di output** (o coda di output). Ovvero, un pacchetto che arriva al commutatore oltre a subire un ritardo di **store and forward** subisce un ritardo di accodamento, il pacchetto deve spendere una quantità di tempo all'interno del buffer di output affinchè i pacchetti che sono arrivati precedentemente vengano processati. I ritardi di questo tipo sono variabili perchè dipendono dal livello di traffico della rete.

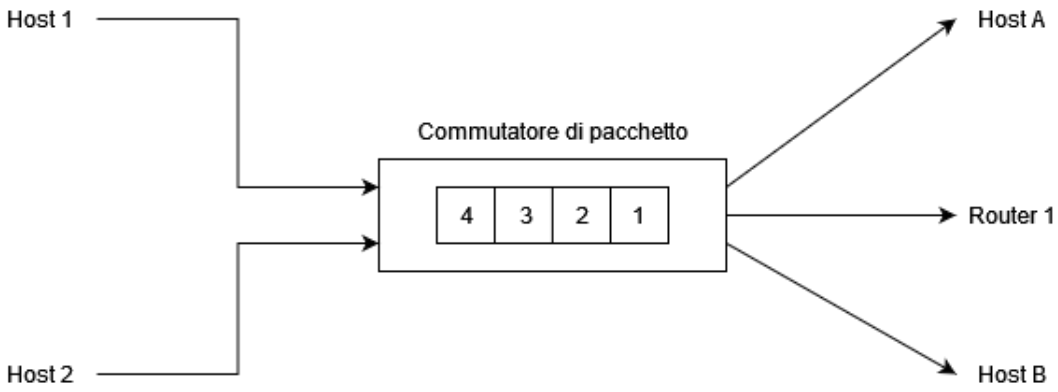


Figura 2.2: La figura rappresenta la coda di output all'interno di un commutatore di pacchetto, i numeri rappresentano un pacchetto e la sua posizione all'interno della coda

A questo punto ci si accorge che è possibile avere il buffer pieno poiché è di dimensione finita, di conseguenza appena giunge un pacchetto al commutatore che ha la coda piena, viene scartato o un pacchetto già presente all'interno della coda per fare spazio a quello appena arrivato oppure viene scartato proprio quello che è appena arrivato, questa situazione fa verificare l'evento chiamato **perdita di pacchetto** (o packet loss). Per comprendere meglio questa situazione facciamo un esempio reale: **congestione**.
La congestione è un evento che si verifica nel momento in cui un commutatore non riesce a liberare la coda di output con una velocità che sia pari o superiore a quella con cui i pacchetti arrivano in coda. Poniamo il caso che il collegamento in entrata abbia una velocità di 100 Mbps mentre il collegamento in uscita abbia una velocità di 15 Mbps, i pacchetti arrivano a una velocità superiore rispetto a quella con la quale il commutatore li invia e di conseguenza la coda andrà a riempirsi fino a saturazione, in quel momento il commutatore dovrà decidere quale pacchetto scartare e si verificherà una perdita di pacchetti.

Tabelle di Inoltro e Protocolli di Instradamento

Come abbiamo già accennato il router possiede molti collegamenti, il pacchetto arriva al router da un collegamento e il router decide su quale collegamento in uscita immettere il pacchetto. Questa decisione viene presa in maniera diversa dipendentemente dal tipo di rete nella quale ci troviamo. Nello specifico di Internet, ogni sistema periferico possiede un indirizzo IP

che è una sorta di indirizzo gerarchico, possiamo paragonarlo ad un indirizzo fisico di una abitazione dove abbiamo: Italia, Puglia, Lecce, Castro, 73030, Piazza Dante, 9.
Ogni router controlla una parte dell'indirizzo IP e tramite una **tabella di inoltro** lo immette in un collegamento. Questo processo avviene per ogni router della rete nel quale il pacchetto transita. Queste tabelle vengono aggiornate (e talvolta create) in maniera automatica dal router, partendo e seguendo i **protocolli di instradamento**.

Commutazione di Circuito

Diversamente da come abbiamo visto nella commutazione di pacchetto, nella commutazione di circuito le risorse sono riservate. Le reti telefoniche sono un perfetto esempio, per prima cosa si stabilisce una connessione tra due sistemi periferici tramite un collegamento a "regola d'arte", una volta effettuata questa connessione la si mantiene attiva per tutta la durata della sessione. Questa connessione garantisce larghezza di banda e velocità e viene chiamata connessione end-to-end (o punto a punto).
Questa tipologia di connessione può essere implementata in due modi differenti:

- Multiplexing a Divisione di Frequenza (FDM)
- Multiplexing a Divisione di Tempo (TDM)

Il FDM suddivide lo spettro di frequenze tra connessioni stabilite tramite il collegamento. Ogni connessione ha una banda di frequenza su ogni collegamento per l'intera durata della connessione. Nelle reti telefoniche la larghezza di banda è circa 4 KHz. La larghezza di banda viene detta **bandwidth** (o ampiezza di banda).
Il TDM invece suddivide la connessione in slot temporali ma ogni slot utilizza interamente la larghezza di banda dello spettro.
Nella commutazione di circuito vi sono alcune problematiche per quanto riguarda i **periodi di silenzio**. Durante una telefonata per esempio se una persona smette di parlare si sta sprecando la risorsa, poichè i circuiti dedicati sono inattivi, questo è il punto fondamentali per cui i commutatori di pacchetto sono una strategia migliore.

2.3.2 Internet: una Rete di Reti

Abbiamo visto come è possibile che i sistemi periferici si connettono ad internet tramite gli ISP di accesso, ora bisogna capire come i vari ISP siano

connessi tra loro. Capire come gli ISP si interconnettono è fondamentale per comprendere come si configura la rete di reti che è internet.
Alla base di tutto abbiamo una gerarchia tra gli ISP, ve ne sono di primo livello (o ISP Tier-1), di secondo, terzo e così via che formano quelle che andremo a chiamare strutture di rete. Ogni struttura di rete va ad aggiungere un livello di ISP oppure alcune tecnologie utilizzate, attualmente possiamo delineare una struttura di rete 5.
Eviteremo di spiegare le strutture di rete dalla 1 alla 4 poichè fungono da ponte per comprendere come e perchè si è arrivari alla struttura 5, noi invece ci concentreremo a capire il funzionamento di quest'ultima.
Partiamo da uno schema.

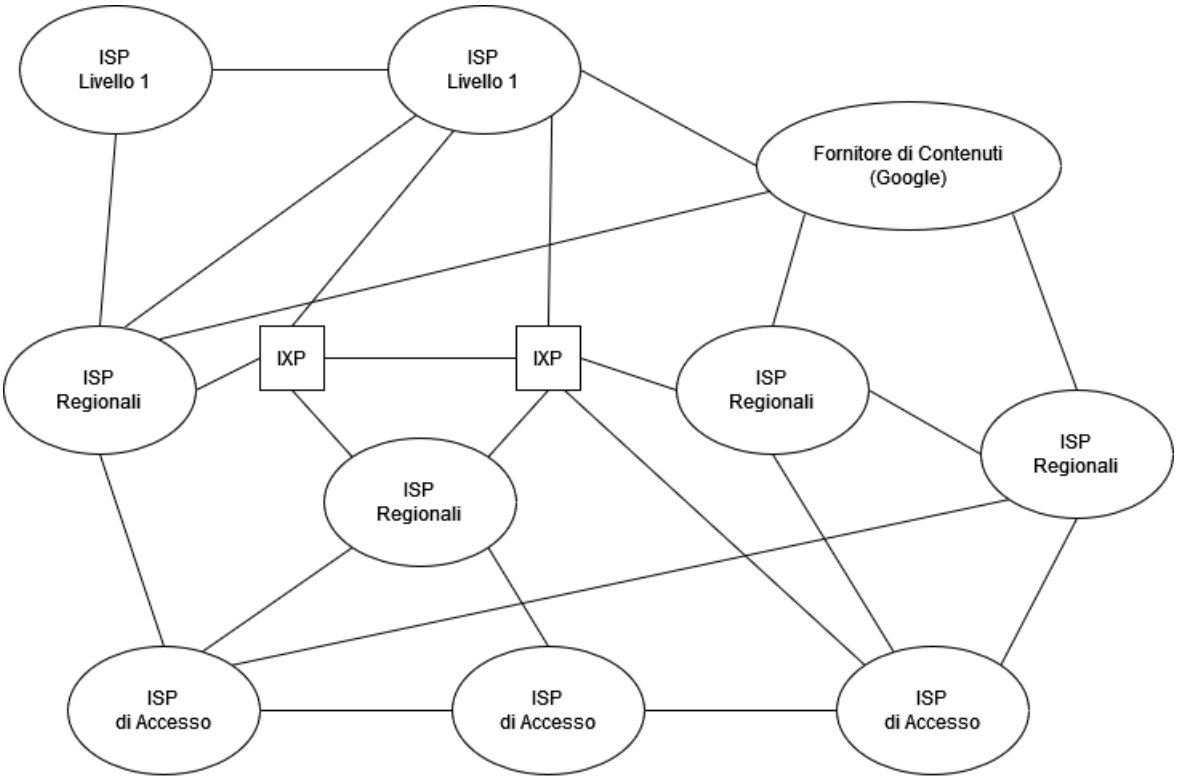


Figura 2.3: La figura rappresenta uno schema generale di una possibile "Internet"

Analizzando l'immagine poniamo l'attenzione per prima cosa sugli ISP di primo livello, sono in cima alla gerarchia e hanno una portata globale, infatti, diversamente da ISP di più "basso" libello, come ISP regionali o di

accesso, non devono pagare per il traffico internet che si muove da una area geografica ad un'altra come ad esempio tra USA e Europa (questo compone la Struttura di Rete 1).
Poi possiamo notare gli ISP Regionali (o di secondo livello) sono qegli ISP che connettono gli ISP di primo libello, di cui sono clienti, agli ISP di accesso (quindi di terzo livello), di cui sono invece fornitori. Questi ISP hanno per l'appunto portata regionale ovvero la loro copertura si estende fino a coprire un'intera nazione.
Abbiamo infine gli ISP di terzo livello (o di accesso) che sono gli ISP clienti degli ISP regionali e fornitori invece degli utenti, infatti sono gli ISP più vicini all'utente finale e lavorano maggiormente su modelli di consumo. A questa struttura bisogna aggiungere:

- PoP
- Multi-Homing
- Peering
- IXP

I **Point of Presence** sono dei punti della rete dove i dispositivi condividono una connessione e possono comunicare. Possiamo definirli come i punti dove "termina" la rete della compagnia del nostro fornitore e "inizia" la rete privata. L'hardware messo a disposizione in questi PoP, ovvero router molto vicini tra loro, ha il compito di aiutare le connessioni tra sistemi periferici tramite la grande velocità di cui dispongono. I PoP sono inoltre uno dei modi per valutare l'estensione di un ISP, più PoP un ISP possiede più è considerato grande e quindi di livello maggiore.
Il **multi-homing** è una tecnica di connessione di una rete (o di una singola macchina) a più di una rete al fine di incrementare affidabilità e performance. L'affidabilità è aumentata dal fatto che nel caso in cui un fornitore subisca un guasto si continua ad avere traffico tramite l'altra (o le altre) rete a cui si è collegati. Le performance sono invece aumentate dal fatto che avendo più connessioni si possono utilizzare simultaneamente, inoltre si può effettuare una connessione rendendola più efficiente scegliendo la rete tramite quale stabilirla in base a velocità o destinazione.
Il **peering** è una interconnessione tra ISP che permette di scambiare traffico tra le loro reti a quelle degli utenti. Tramite questa tecnica un utente che si connette a un ISP Regionale che vuole raggiungere un'altra macchina che è connessa a un "vicino" ISP Regionale differente, ovvero non c'è la necessità di arrivare ad un ISP di primo livello ma l'ISP Regionale farà peering

con l'altro ISP Regionale. In questo modo si aumentano le performance abbattendo i costi, tale tecnica può essere eseguita da qualsiasi ISP verso qualunque ISP dello stesso livello.
Un **IXP** è un luogo fisico, come un datacenter, dove ISP, ma anche altre tipologie di fornitori di servizi, si connettono tra loro, questi luoghi rappresentano i "confini" di due o più reti. Consentono agli ISP di condividere il transito al di fuori della propria rete, inoltre riducono la latenza e i costi.
Ci manca un ultimo passaggio per arrivare alla Struttura di Rete 5, ovvero la struttura che rappresenta nella maniera migliore l'internet odierno.
Parliamo ora di CPN e CDN, ovvero di Content Provider Networks e Content Delivery Networks, le reti di fornitura e distribuzione contenuti. La differenza è minima ma sostanziale, grandi aziende come Google ad esempio possono essere entrambe, per comodità continueremo a chiamarle fornitori di contenuti.
Mantenendo l'esempio di Google, possiamo stimare più di 100 datacenter distribuiti globalmente, tutti interconnessi da una propria rete privata, di fatto essendo al pari livelli di un ISP Tier-1. In questa posizione riesce ad aggirare i costi degli ISP di livello 1 e fare peering direttamente con ISP di livello più basso tramite IXP. Sottolineiamo come tali reti devono comunque effettuare degli accessi a ISP di primo livello per raggiungere determinati servizi o sistemi periferici.
Riassumento:

- ISP di Primo Livello
- ISP Regionali
- ISP di Accesso
- PoP, Multi-Homing, Peering, IXP
- CDN e CPN

NB: Se gli ISP di primo livello sono poco più di una dozzina, gli ISP Regionali e quelli di Accesso insieme invece sono centinaia di migliaia.

2.4 Perdite di Pacchetti, Ritardi e Throughput

Sebbene l'obiettivo di una rete come internet sia quella di far comunicare sistemi periferici in tempo zero, spostando grosse quantità di dati istantaneamente, bisogna però scontrarsi con la fisica e con le tecniche di cui internet si avvale che purtroppo introducono ritardi o addirittura la perdita

di alcuni dati.
Ormai sappiamo che un pacchetto che va dalla sorgente al destinatario subisce dei ritardi all'interno di ogni nodo (e commutatore di pacchetto). Tali ritardi sono:

- Ritardo di Elaborazione
- Ritardi di Accodamento
- Ritardo di Trasmissione
- Ritardo di Propagazione

Ritardo di Elaborazione - Processing Delay Questo ritardo è accomulato per diversi fattori, bisogna controllare l'intestazione del pacchetto, comprendere dove dirigerlo ed eventualmente controllare se vi sono errori di bit dovuti alla trasmissione del nodo precedente. Solitamente nei router ad alta velocità si parla di un ritardo dell'ordine dei microsecondi.

Ritardo di Accodamento - Queuing Delay Quando il pacchetto ha subito l'elaborazione attenderà un tempo, dell'ordine dei microsecondi o dei millesecondi, all'interno della coda di output. Ovvero nella coda che detiene i pacchetti che devono essere inviati su quel determinato collegamento. Se la coda è vuota il ritardo di accodamento è nullo perciò comprendiamo come questo ritardo si possa esprimere in funzione del traffico in ingresso.

Ritardo di Trasmissione - Transmission Delay Ora assumiamo che il nostro pacchetto abbia raggiunto la priorità per essere inviato, soltamente la coda di output ha una politica decisionale FCFS. Ponendo L come la lunghezza in bit del pacchetto e R la velocità in bps di trasmissione del collegamento allora si dice Ritardo di Trasmissione la quatità di tempo che occorre per immettere dal primo fino all'ultimo bit sul collegamento, e si esprime come $\frac{L}{R}$.
L'ordine di grandezza varia dai microsecondi ai millesecondi.

Ritardo di Propagazione - Propagation Delay Il Ritardo di Propagazione è il tempo che un bit impiega a percorrere il collegamento fino al router di destinazione. La velocità a cui il bit viaggia dipende dal materiale di cui è fatto il mezzo, possiamo dare un intervallo che è dai $2 \times 10^8 \frac{m}{s}$ fino ai $3 \times 10^8 \frac{m}{s}$ ovvero la velocità della luce nella fibra ottica. Sempre dell'ordine

dei millesecondi, il ritardo di propagazione è dato dalla distanza tra i due router fratto la velocità di propagazione del mezzo $\frac{d}{v}$.

Ritardo Totale Il ritardo totale del nodo è dato dalla somma dei ritardi appena enunciati:

$$d_{nodo} = d_{elab} + d_{acc} + d_{trasm} + d_{prop}$$

2.4.1 Ritardo di Accodamento e Perdita di Pacchetti

Diversamente dagli altri ritardi enunciati, il ritardo di accodamento può variare da pacchetto a pacchetto. Questo si può notare facilmente perchè se avessimo una coda di output vuoto nella quale arrivano contemporaneamente dei pacchetti il primo avrebbe un ritardo di accodamento nullo mentre l'ultimo in coda deve aspettare la trasmissione di tutti quelli che sono arrivati prima di lui. Data questa grande variabilità la misurazione di tale ritardo viene effettuata tramite strumenti probabilistici e statistici. Per capire quando il ritardo è rilevante o trascurabile ci si deve informare sulla velocità di arrivo del traffico in coda, sulla velocità di trasmissione del collegamento e sulla natura del traffico, se arriva seguendo un periodo oppure a raffiche.
Supponiamo di avere pacchetti grandi sempre e solo L bit e diamo sempre a R la caratterizzazione di velocità di trasmissione aggiungiamo infine α ovvero la velocità media di arrivo dei pacchetti in coda, da questo possiamo dire che $L\alpha\frac{bit}{s}$ è la velocità di arrivo dei bit in coda. Assumendo di avere una coda infinita possiamo dire che il rapporto $\frac{L\alpha}{R}$ è l'INTENSITA' DI TRAFFICO.
Da qui possiamo evidenziare due situazioni:

1. $\frac{L\alpha}{R} > 1$, la velocità media di arrivo dei bit nella coda supera la velocità trasmissiva e quindi la coda andrà a saturarsi.

2. $\frac{L\alpha}{R} < 1$, in questo caso bisogna controllare la natura del traffico. Se il traffico ha natura periodica allora si avrà che ogni pacchetto troverà la coda vuota e non ci saranno ritardi di accodamento, mentre se la natura del traffico è composta da raffiche periodiche si potranno verificare significativi ritardi di accodamento. Infatti supponendo un arrivo di N pacchetti il generico $n - esimo$ pacchetto in coda subirà $(n-1)\frac{L}{R}$ secondi di ritardo.

Abbiamo paralto di tale ritardo di accodamento assumendo che la coda sia infinita, purtroppo il buffer di un commutatore in realtà è finito. Di conseguenza quando si approssima a 1 l'intesità di traffico un pacchetto può trovare la coda satura. In tale situazione è impossibile per il router memorizzare il pacchetto e di conseguenza lo eliminerà generando una perdita di pacchetti, questa situazione prende anche il nome di buffer overflow. Le prestazioni di un nodo vengono misurate tenendo conto di questi due fattori:

- Ritardo di Accodamento Medio
- Probabilità di Perdita di Pacchetti

2.4.2 Ritardo End-To-End

Per comprendere il ritardo end-to-end (o end-to-end delay) supponiamo una rete non congestionata, una sorgente e una destinazione tra cui poniamo $N-1$ router. L'accumulo dei ritardi di ciascun nodo è definito come end-to-end delay e si esprime come:

$$d_{end-to-end} = N(d_{elab} + d_{trasm} + dprop) \text{ dove } d_{trasm} = \frac{L}{R}$$

2.4.3 Ritardo di Pacchettizzazione (e altri)

Un ritardo che si può verificare oltre a quelli già visti è un ritardo dei sistemi periferici che eseguono un determinato protocollo, ovvero un protocollo può ritardare volontariamente la trasmissione dei pacchetti per i motivi più vari, per esempio per la condivisione del mezzo con altri sistemi periferici, esploreremo meglio questo punto quando parleremo del protocollo TCP. Tra i più rilevanti abbiamo il ritardo di trasposizione di pacchetti (o Pacchettizzazione). Avviene in differenti contesti, ad esempio in un flusso multimediale come la telefonia IP (o VoIP, Voice over IP). In questo contesto il mittente deve immettere nel pacchetto la comunicazione digitale, codificarlo e poi iviarlo, questo ritardo è il ritardo di pacchettizzazione, ovvero il tempo di "costruzione" del pacchetto che può essere quasi nulle oppure a volte significativo.

2.4.4 Throughput nelle Reti

Supponiamo che S invii un file di grandi dimensioni a D attraverso una condivisione P2P, il throughput istantaneo è la velocità istante per istante alla quale D sta ricevendo il file. Ponendo il file di F bit e un tempo T necessario al fine di avere tutti gli F bit recapitati a D allora $\frac{F}{T}$ bps

viene denominato throughput medio. Ora che abbiamo le definizioni basilari possiamo porci in alcuni esempi più complessi di una condivisione P2P.

Figura 2.4: Situazione d'esempio

Ponendoci nella situazione mostrata in figura possiao capire che avremmo due possibili sviluppi:

- Se $R_S < R_C$ allora i bit immessi dal server arriveranno senza nessun problema al client e il throughput del sistema sarà appunto R_S bit
- Se $R_S > R_C$ allora il router non riuscirà a trasmettere i bit ad una velocità pari o maggiore a quella con cui entrano e di conseguenza si avrà una situazione poco gradevole dove l'accumulo dei bit all'interno del router andrà ad aumentare.

Per tali ragioni, ovvero per evitare il formarsi di un collo di bottiglia, il thorughput di una rete come quella in figura sarebbe il $min(R_S, R_C)$ e nello specifico di un file $\frac{F}{min(R_S,R_C)}$. Il medesimo ragionamento l'avremmo fatto se la rete fosse costituita da N router il throughput sarebbe stato $\frac{F}{min(R_1,R_2,...,R_N)}$. In un esempio reale il fattore limitante è sempre la rete di accesso. Se avessimo un traffico diverso come può essere una quantità di client che attingono bit da un server affrontando tutti percorsi diversi meno un collegamento, allora quel collegamento che ha una velocità X distribuirà uniformemente tale velocità concedendo $\frac{X}{C}$ dove C è il numero di connessioni che stanno sfruttando quel collegamento. Riassumendo possiamo dire che il throughput dipende dalla velocità di trasmissione dei collegamenti e dal traffico sulla rete.

2.5 Architettura di Internet

Una architettura di rete a livelli o a strati caratterizza una organizzazione che descrive il complesso di funzionalità logiche della rete. Ciascun livello fornisce al livello superiore o inferiore i servizi o le funzionalità richieste, in tali livelli si include anche il livello fisico ovvero l'infrastruttura concreta che determina la topologia della rete.

I livelli sono sede di determinati protocolli che possono essere implementati tramite software o hardware, o combinarli. Generalmente i protocolli di livello più alto sono software e scendendo si arriva all'implementazione hardware passando per una combinazione dei due. Una architettura di questo tipo comporta dei vantaggi in ambito manutentivo e di aggiornamento che sono resi semplici dalla modularità dettata dai livelli stessi, ma comporta svantaggi in quanto si rischia di duplicare i servizi tra i livelli e di non condividere le informazioni tra i livelli in maniera opportuna, di fatto distruggendo il principio stesso dei livelli.
La pila dei protocolli chiamata anche STACK ha due possibili configurazioni:

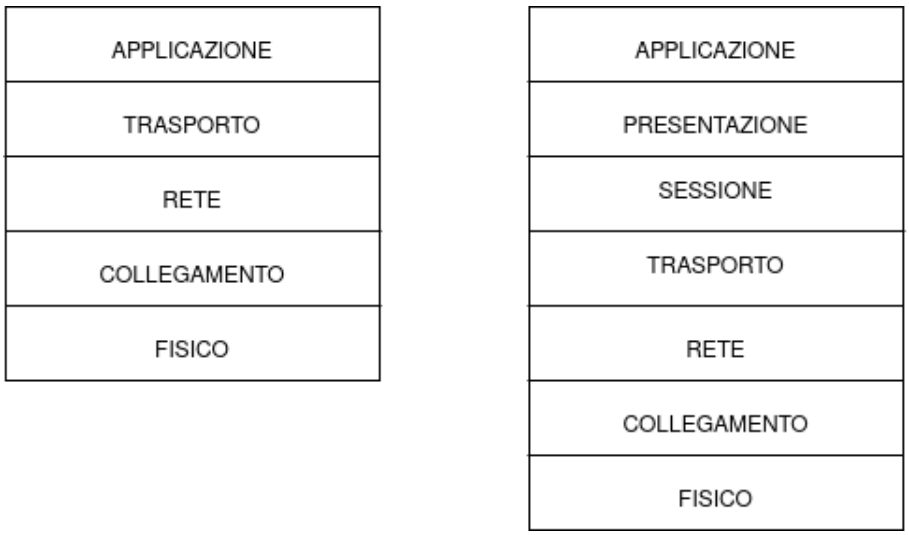


Figura 2.5: Confronto tra Stack TCP/IP e Modello ISO/OSI

La seconda pila fa riferimento al modello OSI che non trattiamo dettagliatamente mentre vedremo il modello a cinque livelli.

2.5.1 Livello di Applicazione

Il livello di applicazione è la sede delle applicazioni di rete e dei relativi protocolli. I protocolli di questo livello riguardano:

- HTTP che consente richiesta e trasferimento di pagine web
- SMTP che consente il trasferimento di messaggi di posta elettronica
- FTP che consente il trasferimento di file tra due sistemi remoti

- DNS che consente la traduzione di nomi di host in indirizzi da 32 bit

Il livello di applicazione è distribuito sui sistemi periferici come applicazione sul sistema periferico stesso che consentono lo scambio di pacchetti, tali pacchetti li chiameremo **messaggi**.

2.5.2 Livello di Trasporto

Trasferisce i messaggi del livello di applicazione tra punti periferici gestiti dalle applicazioni. Abbiamo due protocolli a questo livello:

- **TCP:** fornisce un servizio orientato alle connessioni, include la funzione di garanzia di consegna e il frazionamento di messaggi lunghi in messaggi più piccoli
- **UDP:** Nessuna garanzia, nessuna affidabilità o controllo di flusso e congestione.

I pacchetti a questo livello si denominano **segmenti**, alcune volte anche **datagrammi** ma solo per il protocollo UDP.

2.5.3 Livello di Rete

Tale livello si occupa di trasferire i pacchetti a livello di rete, i pacchetti prendono il nome di **datagrammi**, da un host ad un altro. Il livello di trasporto passa al livello di rete un segmento e un indirizzo di destinazione, tramite il protocollo IP questo livello definisce i campo dei datagrammi e il percorso, infatti vi sono altri protoccolli, tutti della famiglia IP, che gestiscono l'instradamento.

2.5.4 Livello di Collegamento

Il livello di collegamento è il livello a cui si affida il livello di rete per trasferire il pacchetto da un nodo ad un altro, nello specifico ogni nodo passa il datagramma dal livello di rete al livello di collegamento che lo condice al nodo successivo dove il livello di collegamento passerà nuovamente il pacchetto al livello di rete del nuovo nodo. I pacchetti a questo livello vengono chiamati **frame**.

2.5.5 Livello Fisico

Il livello fisico si occupa di spostare fisicamente i bit che compongono il pacchetto (Frame) da un nodo ad un altro, i protocolli di questo livello sono

dipendenti dal livello di collegamento ma ancor più dal mezzo fisico che si utilizza.

2.5.6 Modello ISO/OSI

ISO (Intenational Organization for Standardation) propose il modello a sette livelli, il modello OSI - Open Systems Interconnection. Poneva il livello di Presentazione e di Sessione tra quello Applicativo e quello di Trasporto. Il ruolo del livello di presentazione è quello di consentire una interpretazione del significato dei dati. Il ruolo del livello di sessione è quello di delimitazione e sincronizzazione dei dati coprendendo i mezzi per la ricostruzione dei pacchetti.

2.5.7 Incapsulamento

I router e i commutatori a livello di collegamento sono tutti commutatori di pacchetto. In linea con la complessità di internet, sebbene i commutatori gestiscano l'hardware esattamente come i sistemi periferici, non gestiscono tutti i livelli dello stack. I router riescono a interpretare dal livello fisico fino al livello di rete mentre i commutatori di collegamento solo il livello fisico e di collegamento. Questo fa da premessa al concetto di incapsulamento: un host mittente genera un messaggi a livello di applicazione che viene passato al livello trasporto che aggiunge le proprie informazioni incapsulando il tutto all'interno di un segmento, il segmento subisce il medesimo trattamento al livello di rete e l'incapsulamento genera un datagramma che subirà un'ulteriore aggiunta al livello di collegamento e verrà denominato frame. Ad ogni livello il pacchetto ha sempre due campi:

1. Intestazione (o Header)
2. Payload (o body) che è tipicamente il pacchetto proveniente dal livello superiore

Il processo di incapsulamento è più complesso di come è stato enunciato perchè un messaggio, ad esempio, potrebbe essere troppo lungo e quindi andrebbe prima frammentato e poi ricostruito a destinazione, capiremo meglio più avanti i dettagli.

Capitolo 3

Applications Layer

Le applicazioni sono il motore che spinse il successo di internet, le applicazioni, come il World Wide Web, la posta elettronica e l'e-commerce, sono state ciò che entrando nella quotidianità degli utenti si sono resi essenziali e non sostituibili. A questo livello troviamo i programmi eseguiti sui sistemi periferici che comunicano tramite la rete, ad esempio in applicazioni web esistono due programmi:

1. Il browser che viene eseguito sull'host dell'utente
2. Il server web che si trova sull'host che viene interrogato dal browser

3.1 Architettura

Lo sviluppatore di una applicazione di rete deve capire quale architettura adottare per l'organizzazione dei vari sistemi periferici, le architetture sono principalmente di due tipologie:

- **Architettura Client - Server:** dove abbiamo un host, denominato Server, che è sempre attivo rispondendo sempre alle richieste degli host denominati Client. In questa architettura il client e il server non comunicano direttamente tra loro, inoltre ricordiamo che il server ha un indirizzo IP fisso (o meglio detto statico), mentre il client può avere un indirizzo IP variabile. Quindi i client inviano una richiesta al server che risponde con un pacchetto contenente le informazioni che il client voleva. Questo ci porta a capire come per grossi servizi un singolo server non può soddisfare tutte le richieste per tanto si fa capo a un datacenter che ospitando diversi host creano un grosso server virtuale.

- **Architettura P2P - (peer-to-peer):** in questa architettura la struttura del server (datacenter) è minima o del tutto assente. Le applicazioni che utilizzano questa architettura comunicano direttamente e hanno solitamente una struttura simile o identica, si annoverano in queste applicazioni: condivisioni di file, telefonia, videoconferenze ecc. Se esiste un server in questa architettura è solitamente un server di log per tenere traccia ad esempio degli indirizzi IP dei peer. Il punto strategico di questa architettura è la scalabilità poichè ogni peer oltre ad essere carico di lavoro aggiunge potenza al sistema stesso.

3.2 Comunicazione tra Processi

A livello di singolo host i processi comunicano tramite il sistema operatico che mette a disposizione un approccio interprocesso. Se invece i processi sono su host differenti per comunicare sfruttando messaggi che si scambiano sulla rete. La coppia di processi che comunicano tramite la rete indipendentemente dall'architettura vengono etichettati come client e come server. Per chiarire questa affermazione nell'architettura P2P se un peer sta richiedendo un file ad un altro peer in questo momento il primo viene definito client mentre quello che risponde con il file viene chiamato server, mentre per quanto riguarda l'architettura client-server la spiegazione è superfluo. Possiamo generalizzare dicendo:

> Nel contesto di una sessione di comunicazione tra una coppia di processo quello che avvia la comunicazione è indicato come client mentre quello che attende di essere contattato è detto server.

Una applicazione che invia messaggi tramite la rete sfrutta una interfaccia denominata **socket**. Possiamo definire la socket come il punto in cui il codice applicativo di un processo accede al canale di comunicazione tramite una porta, ottenendo una comunicazione tra processo che lavorano su macchine fisicamente separato. Gli host vengono identificati tramite indirizzi IP, il mittente oltre a conoscere tale indirizzo deve conoscere anche il numero di porta legata all'applicazione che vuole raggiungere poichè su una macchina ci possono essere differenti applicazioni che utilizzano la rete e a ognuna vi è assegnata un numero di porta. Alcuni servizi specifici e più utilizzati hanno alcune porte dedicate in maniera standard, i web server ad esempio hanno la porta 80, ssh la porta 22 e FTP la porta 21.

3.3 Servizi di Trasporto

Come abbiamo detto la socket è una interfaccia tra il livello applicativo e il livello di trasporto, quindi il client utilizza la socket e il livello di trasporto, quindi il client utilizza la socket per inviare un messaggio e la consegna alla socket dell'host ricevente. Le reti mettono a disposizione varie tipologie di protocollo di trasporto, in maniera che uno sviluppatore possa scegliere il protocollo migliore per la propria applicazione. In maniera sommaria possiamo scegliere quattro tipologie:

- **Trasferimento Dati Affidabile:** Come abbiamo già visto i pacchetti possono essere persi nella rete, alcune applicazioni però non possono permettersi di perdere informazioni poichè potrebbero avere gravi conseguenze. Se un protocollo fornisce un servizio dove garantisce che i dati arrivino e siano completi, tale servizio si chiama **trasferimento dati affidabile**. Questa tipologia di servizio garantisce senza la minima ombra di dubbio che i dati verrano recapitati senza errori al ricevente. Vi sono poi altre tipologie di applicazioni ovvero **applicazioni che tollero le perdite**, in particolare le applicazioni multimediali audio/video a uso personale possono tollerare una certa quantità di dati perduti.

- **Throughput:** Come abbiamo già visto il throughput è il tasso al quale il processo mittente può inviare i bit al processo ricevente. Dal momento che la suddivisione della banda viene fatta tra tutte le sessioni che vengono create in maniera dinamica allora il throughput può essere molto variabile nel tempo. Allora il protocollo a livello di trasporto può offrire il servizio di throughput garantito ovvero se il protocollo lo permette l'applicazione può richiedere un throughput di r bps. Se il protocollo a livello di trasporto non può fornire questo servizio, l'applicazione dovrà codificare i dati a un livello inferirore o rinunciare. Le applicazioni che hanno requisiti di throughput vengono dette **applicazioni sensibili alla banda**, altrimenti vengono dette **applicazioni elastiche**. Quest'ultime sono applicazioni si cui non si ha interesse su un throughput alto o basso.

- **Temporizzazione:** Questa tipologia di servizio interessa applicazioni di campo real-time, infatti un servizio di questo tipo garantisce che una socket riceva i pacchetti entro un lasso di tempo ben preciso. Infatti si può immaginare come in una sessione di telefonia se i pacchetti non

venissero recapitati entro un lasso di tempo ben preciso si creerebbero pause innaturali durante la conversazione.

- **Sicurezza:** Un protocollo di trasporto può includere livelli di sicurezza come la riservatezza, l'integrità dei dati e l'autenticazione. Per esempio sulla riservatezza possiamo dire che il protocollo cifra i dati che invia il mittente e lo decifra per il ricevente.

3.4 Servizi di Trasporto in Internet

In internet troviamo due procolli di trasporto messi a disposizione delle applicazioni (e quindi a livello applicativo):

- TCP - Transmission Control Protocol
- UDP - User Datagram Protocol

Questi due protcolli offrono modelli di servizio molto diversi tra loro.

3.4.1 Servizi di TCP

Un'applicazione che adotta il protocollo TCP si avvale di due servizi:

- **Servizio Orientato alla Connessione:** Client e Server si scambiano informazioni prima che i messaggi tra l'applicazione e il server fluiscano. Questa procedura e l'Handshaking. Dopo l'handshaking si dice che esiste una connessione TCP tra le socket client e la socket server, la connessione è del tipo Full-Duplex, i due processi possono scambiarsi contemporaneamente messaggi sulla stessa connessione.
- **Servizio di Trasferimento Affidabile:** TCP permette all'applicazione di avere un trasporto di byte senza errori, duplicazioni e nell'ordine giusto. Il servizio garantisce la consegna dei dati senza perdita.

Il protocollo include un meccanismo di controllo della congestione, questo non è un servizio orientato alle applicazioni ma punta ad una stabilità di internet, essenzialmente TCP esegue una strozzatura del processo di invio se il traffico in rete appare eccessivo rispetto all'hardware messo a disposizione.

3.4.2 Servizi di UDP

Il modello di servizi che il protocollo UDP mette a disposizione è semplice e leggero. Viene definito senza connessione perchè non viene eseguito l'handshake e non garantisce un trasferimento affidabile. Il protocollo UDP non garantisce che i dati arrivino, quindi è possibile che vi siano errori, duplicati, perdite e nemmeno l'ordine dei pacchetti è assicurato. La congestione non è controllata. A causa, o per merito, di questo modello UDP può spingere i dati al livello sosttostante a velocità più elevate.

3.5 Protocolli a Livello di Applicazione

Un protocollo applicativo (o a livello applicazione o di applicazione) stabilisce gli standard di comunicazione tra due processi che sono in esecuzione su sistemi periferici differenti, ovvero stabilisce la modalità con cui i processi si scambiano messaggi. Tali protocolli stabiliscono:

- I tipi di messaggio
- La sintassi dei vari tipi di messaggio
- La semantica dei campi, il significato delle informazioni dei messaggi
- Le regole per determinare quando e come i processi inviano e ricevono messaggi

Le **RFC - Requests for Comments** sono le specifiche di pubblico dominio che descrivono i protocolli, inclusi quelli a livello di applicazione. Sottolineiamo il fatto che un protocollo a livello applicativo è solo una parte della applicazione e non l'applicazione in sè. Concentreremo la nostra dissertazione su sei applicazioni e relativi protocolli:

1. Web con il protcollo HTTP
2. Posta elettronica con il protcollo SMTP
3. Protocollo FTP
4. Servizio di Directory con DNS
5. Streaming Video
6. Applicazioni P2P

3.6 WEB e HTTP

Il web o meglio l'applicazione del World Wide Web è stata l'applicazione che ha cambiato per sempre l'uso di internet. Ciò che attira l'attenzione è che il web opera su richiesta, quindi si può avere ciò che si vuole quando lo si vuole. Fino ad allora questo non era possibile perchè televisione e radio stabilivano quando il contenuto poteva essere fruibile.
HTTP (HyperText Transfer Protocol) definisce sia la struttura dei messaggi sia la modalità in cui client e server si scambiano messaggi. Definiamo alcuni termini prima di addentrarci in HTTP:

- Pagine Web: detta anche documento è costituita da oggetti
- Oggetto: è un file di qualsiasi tipo HTML, immagini, applet Java, ecc.
- File HTML Principale: è la parte principale delle pagini web e referenzia altri oggetti tramite URL
- Browser Web: applicazione che implementa il lato client di HTTP
- Web Server: applicazione che implementa il lato server di HTTP e ospita gli oggetti web indirizzabili tramite URL che un client può volere

L'idea di fondo del protcollo HTTP è composta dal lato server, in attesa di una richiesta (nello specifico un messaggio di richiesta HTTP) e provvede ad una risposta (nello specifico un messaggio di richiesta HTTP), e dal lato client che invia il messaggio di richiesta e attende la risposta, che contiene gli oggetti richiesti.

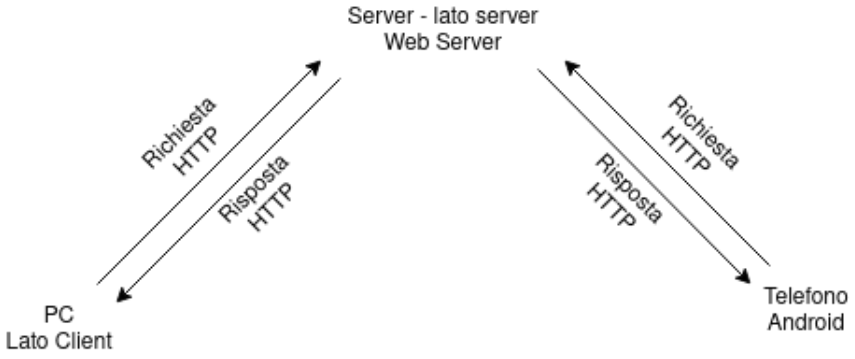


Figura 3.1: Schematizzazione Richieste e Risposta HTTP

HTTP utilizza il protocollo di trasporto TCP, quindi il client HTTP stabilisce per prima cosa una connessione TCP con il server, di conseguenza vengono utilizzate le socket di client e server per effettuare l'accesso a TCP. Come abbiamo già visto abbiamo una struttura a livelli ben progettata ha i suoi vantaggi, ne vediamo uno istantaneamente poichè il protocollo HTTP non si preoccupa di dati smarriti, recupero di dati o riordini di essi poichè di questi problemi se ne occupa il protocollo al livello inferiore ovvero TCP. Per completare una panoramica del funzionamento di HTTP bisogna specificare che il protocollo HTTP lato server viene definito come **Stateless Protocol** ovvero protocollo senza memoria di stato, ciò significa che il server non mantiene informazioni sul client da cui ha ricevuto una richiesta, di conseguenza se la medesima richiesta da parte del client viene eseguita più volte a distanza di pochi secondi il server risponderà nuovamente inviando il messaggio senza avere "memoria" dell'invio precedente.

3.7 Connessioni Persistenti e Non-Persistenti

HTTP utilizza TCP in due modalità:

- Connessione Persistente
- Connessione Non-Persistente

La versione HTTP/1.0 usa la connessione non persistente che consente di trasferire un singolo file per ogni connessione TCP. Le versioni HTTP/1.1 e HTTP/2 usano connessioni persistenti di default ovvero consentono il trasferimento di più file con un'unica connessione TCP, rimane però la possibilità di configurare client e server per l'utilizzo di connessioni non-persistenti.

3.7.1 Connessione Non-Persistente

Per comprendere come una connessione non persistente avvenga facciamo un esempio. Supponiamo di avere un client che effettua una richiesta per una pagina web contente un semplice file html che indirizza 10 immagini per un totale di 11 file. Le transiazioni HTTP avvengono quindi come segue:

1. Il browser instaura una connessione TCP col server sulla porta 80
2. Il browser invia un messaggio di richiesta HTTP
3. Il server HTTP rivece il messaggio di richiesta, cerca il file richiesto e se lo trova, lo inserisce in un messaggio di risposta HTTP che invia al client

4. Il server web chiude la connessione TCP instaurta con il client
5. Il client riceve la risposta, interpreta il file html e trova i riferimeti a 10 oggetti immagine, la connessione si chiude.
6. Vengono ora ripetuti i passi 1 fino al 4 per ogni oggetto immagine che il client deve ricevere

Quindi avremo undici connessioni totali. Dieci di queste connessioni saranno quelle relative alla richiesta delle immagini e possono essere eseguite in serie oppure in parallelo. Le connessioni in parallelo solitamente previa congigurazione possono essere da 5 a 10, ovviamente le connessioni in parallelo riducono i tempi di risposta. Come abbiamo detto il server all'arrivo di una richiesta invia il file html mediante TCP. Il tempo che intercorre tra l'inizio della richiesta fino all'ottenimento della risposta viene definito come RTT (Round Trip Time) o tempo di andata e ritorno. Nello specifico RTT definisce quell'intervallo di tempo che passa da quando un poccolo messaggio parte dal client e arriva al server, fino a quando il client non riceve la risposta completa del server.

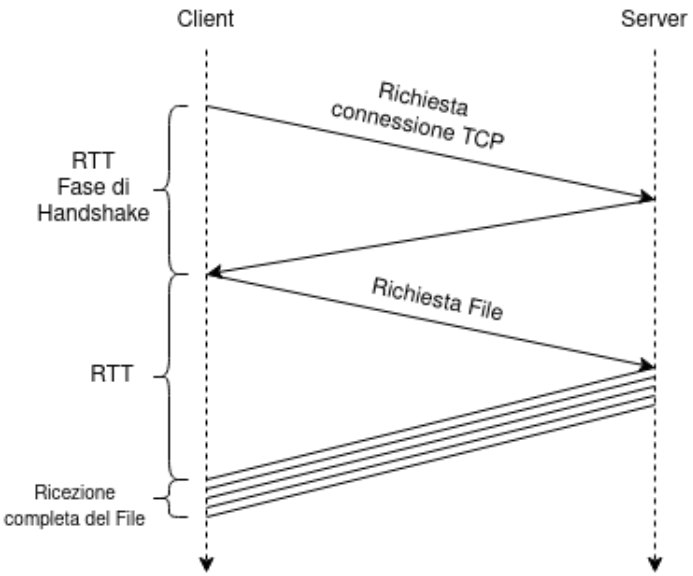


Figura 3.2: Schematizzazione per esempio di Connessione Non Persistente - Questo schema viene ripetuto per ogni file delle undici connessioni che abbiamo citato

Svantaggi:

- Per ogni file richiesto è necessario instaurare una nuova connessione TCP che però richiede buffer e variabili sia sul client sia sul server. Se il server riceve centinaia e centinaia di richieste allora si avrà un sovraccarico.
- Prima dell'inizio del trasferimento di ogni singolo gile devono trascorrere due RTT

3.7.2 Connessione Persistente

La connessione persistente consente di trasferire più file con un'unica connessione TCP. Nell'esempio precedente avremo un'unica connessione che comprende il file html e le 10 immagini. Il server chiude la connessione dopo un lasso di tempo in cui la connessione non è utilizzata, questo lasso di tempo detto **intervallo di timeout**. La connessione persistente ha due modalità di funzionamento:

- Senza Parallelismo
- Con Parallelismo

La connessione persistente senza parallelismo consente al client di inviare una nuova richiesta solo quando la risposta precedente è stata ricevuta. In questo caso prima che nizi il trasferimento di ciascuna immagine delle 10 che il file html referenzia bisogna attendere un RTT.

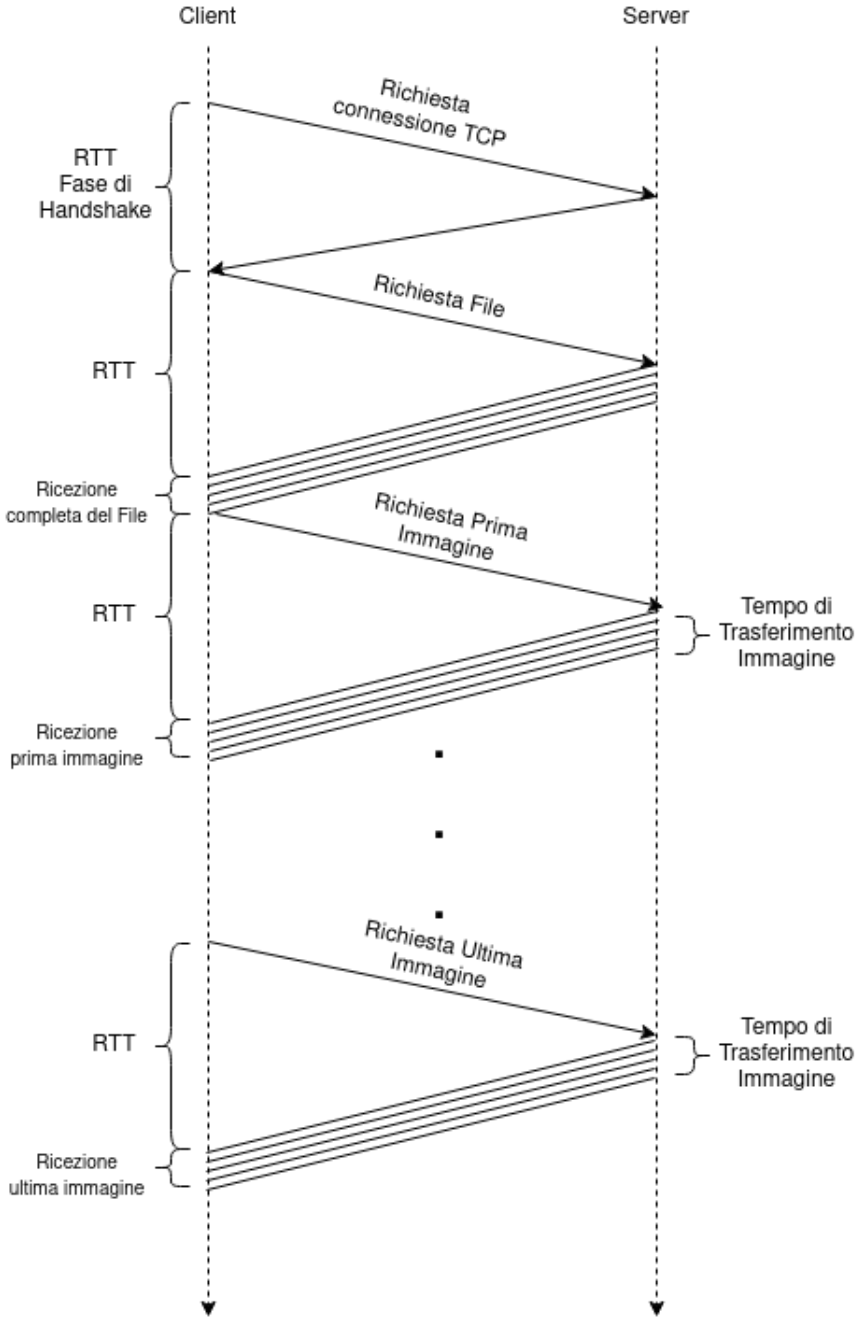


Figura 3.3: Schematizzazione per esempio di Connessione Persistente

La connessione persistente con parallelismo consente al client di inviare una richiesta appena trova nelle pagine html i riferimenti agli oggetti senza attendere la risposta alla precedente richiesta. Il server quando riceve le richieste inizia ad inviare gli oggetti uno dopo l'altro. Di conseguenza avremo il tempo di attesa di un solo RTT per tutti gli oggetti e non più un RTT per ogni oggetto. La connessione TCP inoltre rimane attiva per un tempo ingeriore rispetto all'assenza di parallelismo. Il parallelismo è la modalità di default di funzionamento della versione HTTP/1.1 .

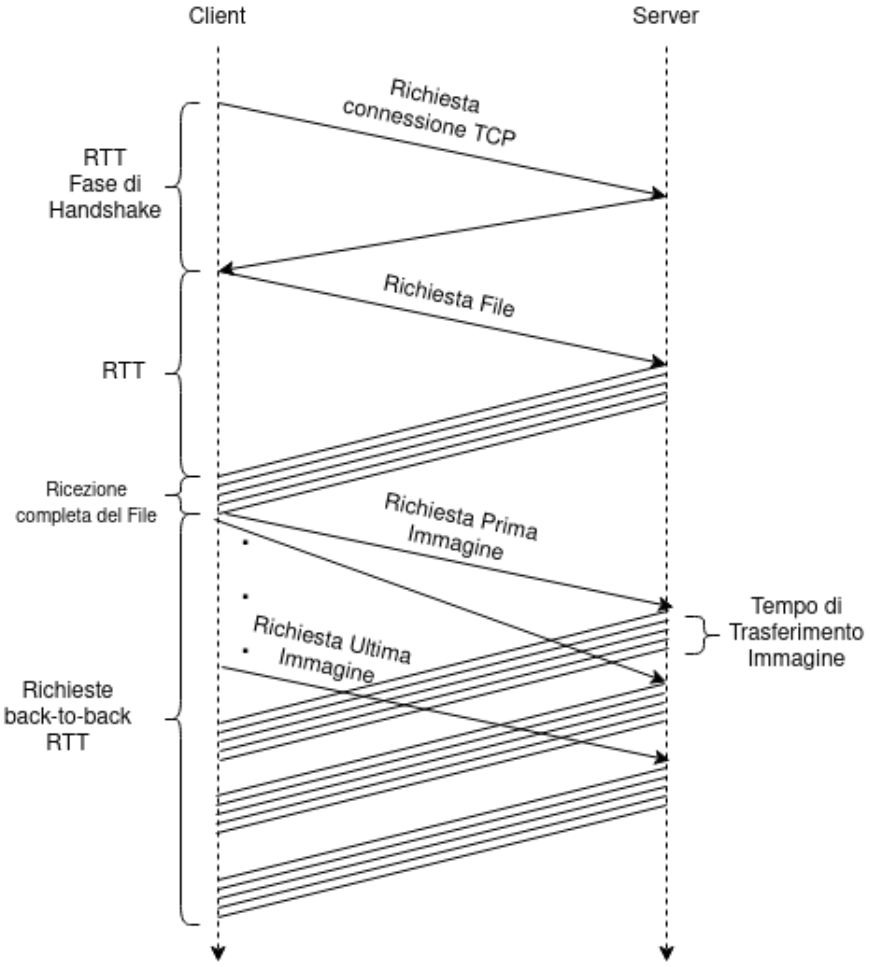


Figura 3.4: Schematizzazione per esempio di Connessione Persistente con Parallelismo

3.8 Messaggi HTTP

Un messaggio di richiesta HTTP è composto dalle seguenti parti:

[method][URL][version] [header] [body]

Un esempio di messaggio è:

```
GET /cartella/pagina.html HTTP/1.1
Host: www.blog.it
Connection: close
User-Agent: Mozilla/5.0
Accept-language: it
```

In generale un messaggio è costituito da un numero indefinito di righe, la prima è la **riga di richiesta** seguita dalle **righe di intestazione**. La riga di richiesta è formata dal metodo che può essere:

- GET: Recupera una risorsa del server
- POST: Invia una risorsa al server
- DELET: Cancella una risorsa dal server
- PUT: Memorizza una risorsa sul server
- HEAD: Recupera solo l'header della risposta senza la risorsa

Dopo il metodo abbiamo il codice identificativo unico della risorsa e infine la versione del protocollo HTTP che stiamo usando.
Le righe di intestazione:

- Host: specifica l'host su cui risiede l'oggetto
- Connection: il browser comunica al server la metodologia di connessione, se ha parametro close la connessione è non persistente altrimenti il parametro è keep-alive
- User-Agent: specifica la tipologia di browser che sta eseguendo la richiesta ed è utile poichè il server può inviare diverse versioni dello stesso oggetto in base al parametro di questa opzione
- Accept-language: specifica la versione della lingua che l'utente preferisce

Quest'ultima opzione non è essenziale ma è stata inserita di proposito per far capire come in realtà le opzioni e quindi le stesse righe di intestazioni possono essere innumerevoli, se si vuole approfondire si consiglia una lettura di:

`developer.mozilla.org/es-US/docs/Web/HTTP`

Nello specifico la sezione HTTP Messages, purtroppo tale risorsa è fruibile solo in lingua inglese.
Nell'esempio che abbiamo esplicato vi era il metodo GET di conseguenza il "corpo", il body del messaggio era vuoto ma non è così per tutti i metodi. Abbiamo analizzato l'header del messaggio ma con metodi come POST la parte del body sarebbe composto dai dati che il client sta inviando al server, per fare un esempio avremo l'header come l'abbiamo visto ma col metodo POST e le credenziali nel blog quando eseguiamo l'accesso a un sito.
Proseguendo sull'esempio iniziale vediamo un messaggio di risposta che un server HTTP potrebbe inviare:

```
HTTP/1.1   200 OK
Connection: close
Date: Thu, 18 Aug 2022 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2022 15:11:03 GMT
Content-Lenght: 6821
[dati...dati...dati...]
```

Quest'ultima riga tra parentesi sta a rappresentare il body della risposta con l'oggetto richiesto dal client. L'header invece è formato da una **riga di stato** e da sei **righe di intestazione** che precedono il body. Nella riga di stato abbiamo la versione del protocollo HTTP seguita dal **codice di stato** e dalla **reason**, questa coppia, che chiameremo codice di risposta o semplicemente codice di stato, può variare a seconda della risposta:

- 200 OK : Richiesta andata a buon fine
- 301 Moved Permanently : Risorsa spostata in un nuovo URI
- 404 Not Found : Risorsa non trovata
- 500 Internal Server Error : Configurazione del server errata
- 505 HTTP Version Not Supported : Il server non dispone della versione HTTP richiesta

Poi abbiamo:

- **Connection: close** → Il server dichiara che chiuderà la connessione TCP dopo l'invio dell'oggetto
- **Date** → Indica ora e data della crazione e invio della risposta
- **Server** → è analoga a User-Agent, specifica che applicazione funge da web server
- **Last-Modified** → Indica il momento di creazione o ultima modifica dell'oggetto in questione
- **Content-Lenght** → Determina il numero di byte dell'oggetto
- **Content-Type** → Indica la tipologia dell'oggetto inviato, nel nostro caso `text/html`

Anche in questo caso le righe di intestazione possono essere molto più prospere rispetto a quelle che abbiamo proposto e rinnoviamo l'invito alla consultazione del sito precedentemente citato.

3.8.1 Cookie

Spesso i web server hanno il bisogno di autenticare i propri utenti per fornire specifici contenuti in base all'identità. La tecnologia dei cookie ci consente di tenere traccia degli utenti e presenta quattro componenti essenziali:

1. Una riga di intestazione nel messaggio di risposta HTTP, ad esempio "Set-Cookie: [valore]"
2. Una riga di intestazione nel messaggio di richiesta HTTP
3. Un file mantenuto sul sistema dell'utente e gestito dal browser
4. Un database sul server per il sito

Schematizziamo un esempio per rendere più comprensibile quanto appena detto:

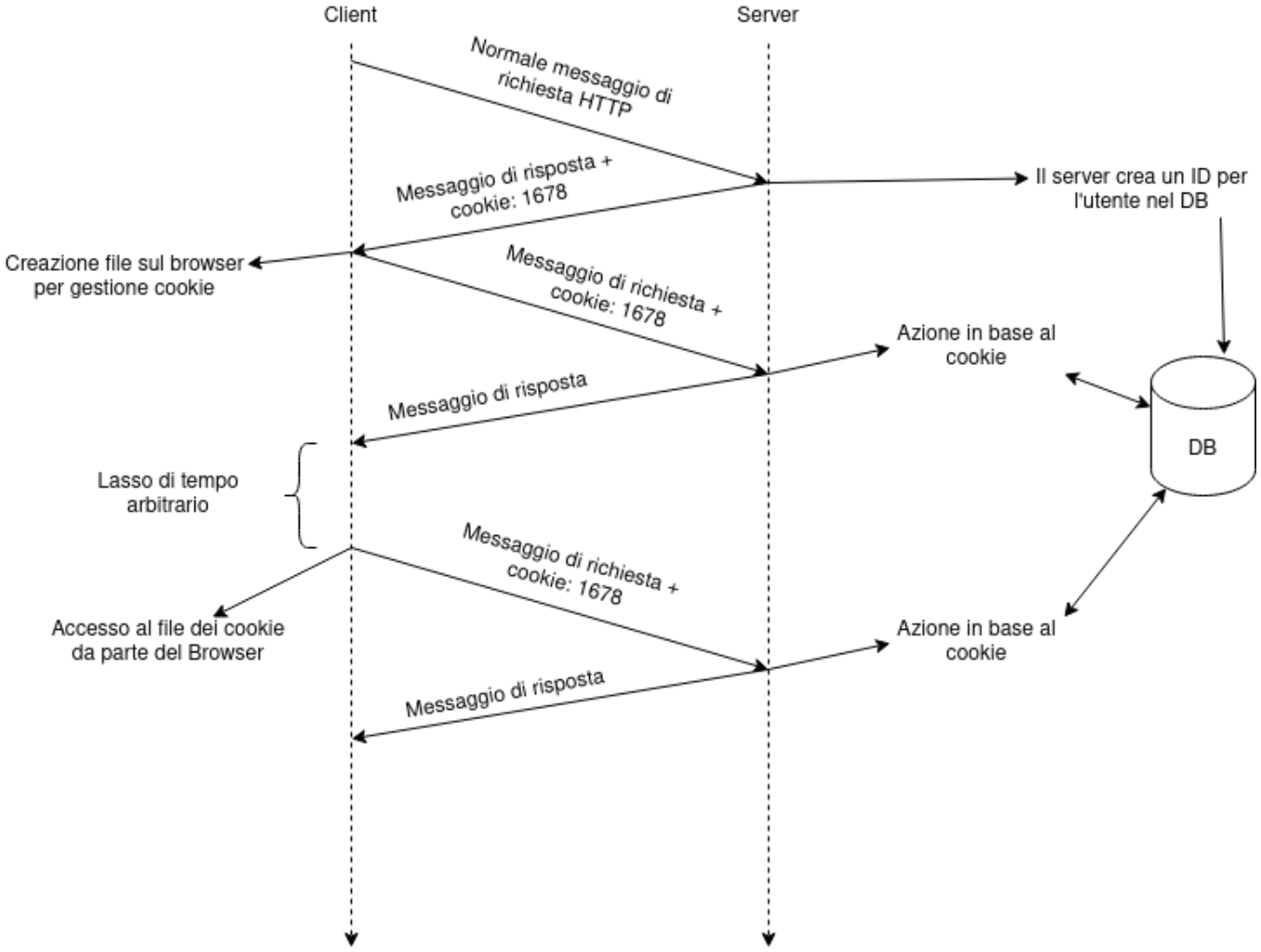


Figura 3.5: Esempio di utilizzo dei cookie

Se in un secondo momento l'utente del client si registrerà allora il proprietario del server potrà associare il proprietario del server potrà associare all'identificativo il nome, cognome e tutte le informazioni richieste nella registrazione all'identificativo già salvato nel DB. Usando però una combinazione di cookie e informazioni comportamentali che l'utente ha sul sito, come prodotti cercati o altro, si può sfociare in controversie visitando

`www.cookiecentral.com`

3.8.2 Web Caching

Una web cache viene detta anche proxy server, risponde alle richieste HTTP dei client al posto del server effettivo. Un client può essere configurato in modo tale che faccia la richiesta prima al proxy, possiamo fare una scaletta di ciò che abbiene:

1. Il browser invia una richiesta HTTP per un oggetto specifico stabilendo una connessione TCP col server proxy
2. Una volta ricevuta la richiesta controlla se l'oggetto richiesto è memorizzato localmente, se esiste viene inviato nel messaggio HTTP di risposta dal proxy stesso
3. Ovviamente esiste il caso nel quale la cache non possiede l'oggetto e di conseguenza invia la richiesta al server originario, arrivata la risposta l'oggetto viene salvato nella cache e poi viene copiato nella risposta al client

Possiamo quindi stabilire che la cache è sia un server che un client. Il motivo principale all'esistenza dei proxy è che possono ridurre notevolmente i tempi di risposta alle richieste dei client, questo perchè l'ampiezza di banda che identifica il collo di bottiglia nel percorso client-server è di gran lunga inferiore rispetto all'ampiezza di banda minima tra client e proxy. I proxy quindi riescono ad alleggerire di molto il traffico web di internet e di conseguenza riescono ad aumentare le prestazioni di tutte le applicazioni.

3.8.3 GET Condizionale

Abbiamo visto quali sono i benefici della web cache ma sorge un nuovo problema:

> L'oggetto ospitato nel web server potrebbe essere modificato rispetto alla copia nel client (proxy browser).

HTTP permette di verificare se gli oggetti sono aggiornati tramite un meccaniscmo chiamato **GET Condizionale**. Il get condizionale utilizza il metodo GET e include una riga di intestazione denominata **"If-modified-sice:"** . Per prima cosa un proxy invia al server originario per conto del client:

```
GET   /roba/roba.gif   HTTP/1.1
Host: www.robadiroba.com
[altre righe di intestazione poco interessanti]
[data...data...data...]
```

Il server invia la risposta e l'oggetto:

```
HTTP/1.1   200 OK
Date: Sat, 3 Oct 2022 16:39:02
Server: nginx
Last-Modified: Wed, 9 Sep 2022  09:50:04
Content-Type: image/gif
[data...data...data...]
```

A questo punto l'ggetto risiede nel proxy ma c'è la possibilità che l'oggetto sia modificato quindi il proxy invia un controllo GET condizionale:

```
GET   /roba/roba.gif   HTTP/1.1
Host: www.robadiroba.com
If-modified-since: Wed, 9 Sep 2022  09:50:04
```

Data che la riga "If-modified-since" è uguale a "Last-Modified" allora il server risponderà con:

```
HTTP/1.1   304 Not Modified
Date: Sat, 10 Oct 2022  15:39:29
Server: nginx
[Body Vuoto]
```

La riga di stato 304 Not Modified comunica al proxy che non è stato modificato e che quindi il proxy può inviarlo al client senza problemi.

3.9 Posta Elettronica

Il sistema postale di internet ha tre componenti principali:

- User Agent
- Server di Posta
- Protocollo SMTP

Quando un mittente completa il messaggio lo user agent lo invia al server di posta che viene posto nella coda dei messaggi in uscita. Il destinatario che vuole leggere il messaggio lo recupera tramite lo user agent che lo recupera dalla casella di posta nel suo mail server. Ogni destinatario ha una casella di posta collocata in un server di posta, per accedere al server bisogna essere autenticato con credenziali. Il server del mittente gestisce anche i provlemi

del server del destinatario, se non riesce a consegnarlo trattiene il messaggio nella coda e tenta di inviarlo ogni 30 minuti. SMTP è il principale protocollo a livello applicativo per la posta elettronica che sfrutta un trasferimento dati affidabile come TCP. SMTP ha un lato client che viene sfruttato nel server del mittente e un lato server che viene sfruttato nel server del destinatario.

3.9.1 SMTP - Simple Mail Transfer Protocol

SMTP è un protocollo che è stato sviluppato agli albori di internet per questo si porta degli strascichi, bisogna codificare tutto il corpo i ASCII a 7 bit, andava bene quando i messaggi erano solo testo, ora però si hanno file multimediali che non è il massimo codificare in ASCII 7 bit. Facciamo un esempio:

1. Il mittente invoca il proprio user agent a cui gornisce l'indirizzo del destinatario e il messaggio
2. Lo user agent del mittente invia al server mail del mittente il messaggio che verrà posto in coda di messaggi
3. Il server del mittente userà SMTP lato client per inviare al server dal destinatario il messaaggio che utilizzerà il lato server di SMTP
   (a) Il server mittente esegue un handshake SMTP col server destinatario e invia tramite connessione TCP il messaggio
   (b) Il server destinatario riceve il messaggio e lo pone nella cartella del destinatario
4. Il destinatario quando lo ritiene opportuno convoca il proprio user-agent per leggere il messaggio

Vediamo un esempio di messaggi scambiati tra lato client e lato server dal protocollo SMTP. Il client è pollo.fr e il server è gallina.edu:

```
S: 220 gallina.edu
C: HELO pollo.fr
S: 250 Hello gallina.edu, pleased to meet you
C: MAIL FROM: <mittente@pollo.fr>
S: 250 mittente@pollo.fr ... Sender OK
C: RCPT TO: <destinatario@gallina.edu>
S: 250 destinatario@gallina.edu ... Recipient OK
C: DATA
```

```
S: 354 Enter mail, end with "." on a line by itself
C: Come stai?
C: Tutto bene in Francia?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 gallina.edu closing connection
```

Il client dialoga tramite comandi: HELO, MAIL FROM, RCPT TO, DATA e QUIT che sono autoesplicativi mentre il server risponde sempre coi codici di stato. Il protocollo SMTP differisce per diversi fattori dal protocollo HTTP che abbiamo visto, il punto più importante è che il protocollo HTTP è un protocollo di PULL ovvero il client attira i dati a sè mentre SMTP è un protocollo di PUSH i dati vengono spinti verso un server. I messaggi inviati tramite SMTP hanno al loro interno le riche di intestazione che sono:

```
From: mittente@pollo.fr
To: destinatario@gallina.edu
Subject: Vediamo come stai
```

Facendo un riassunto abbiamo lo user agent del mittente che usa SMTP verso il proprio server mail, quest'ultimo utilizzerà ancora SMTP per inviare al server mail del destinatario il messaggio avendo una situazione di questo tipo:

Figura 3.6: Schematizzazione Client-Server SMTP

Lo user-agent mittente ha bisogno del proprio server mail perchè non riuscirebbe a gestire i problemi, eventuali, che il server mail destinatario potrebbe avere. Ora però bisogna capire come lo user agent destinatario prende i messaggi dal proprio server di posta. SMTP non è un protocollo adatto poichè è di push e non di pull, qui entrano in campo due opzioni:

- **POP3 - Post Office Protocol**
- **IMAP - Internet Mail Access Protocol**

Un'ulteriore opzione potrebbe essere HTTP ma non lo prendiamo studieremo.

3.9.2 POP3 - Post Office Protocol

POP3 è un protocollo di accesso molto semplice.
Il client apre una connessione TCP sulla porta 110, una volta stabilita POP3 procede in tre fasi:

- Autorizzazione
- Transazione
- Aggiornamento

Durante l'autorizzazione lo user agent invia nome utente e password in chiaro al server per autenticarlo. Durante la transazione lo user-agent recupera i messaggi e può mancare i messaggi per la cancellazione, rimuovere i marcatori e avere le statistiche sulla posta. Durante l'aggiornamento il server rimuove i messaggi che sono stati marcati per la cancellazione, tale fase avviene solo dopo che lo user-agent ha inviato il comando di quit e chiusa la sessione POP3. Nella fase di autorizzazione abbiamo due comando principali che sono:

- **user <nome-utente>**
- **pass <password>**

Se l'autorizzazione non ha successo uno dei due comandi avrà come risposta `-ERR` dal server altrimenti `+OK` seguito da alcune precisazione. POP3 può essere impostato dall'utente in due modalità:

- Scarica e Cancella
- Scarica e Mantieni

Nella prima opzione abbiamo che il server una volta scaricati i messaggi nel client autorizzato cancellarà in fase di aggiornamento tutto ciò che ha scaricato, altrimenti nell'opzione due manterrà in memoria i messaggi. Per esempio nella prima modalità si useranno i comandi:

```
list
retr
dele
```

Lo user agent chiede al server di elencare la dimensione dei messaggi memorizzati tramite `list`, poi recupera il singolo messaggio tramite `retr` e infine con `dele` li marca per la cancellazione in fase di aggiornamento quindi sopo che lo user agent ha inviato il comando `quit`.

3.9.3 IMAP - Internet Mail Access Protocol

POP3 non fornisce all'utente la possibilità di creare cartelle remote e assegnare i messaggi. Per superare i limiti di POP3 si è sviluppato il protocollo di posta IMAP, che proprio per i limiti che vuole supeare è ben più complesso. IMAP associa di dafault i messaggi in arrivo nella cartella INBOX in maniera che l'utente possa leggerlo, cancellarlo e così via. Inoltre abbiamo i comandi per creare e cancellare cartelle e associare i messaggi a tali cartelle. IMAP conserva informazioni sullo sessione dell'utente per tenere traccia di nomi delle cartelle e associazioni tra messaggi e cartelle. Un punto di forza di IMAP è la componente che consente di ottenere parti di messaggi.

3.10 FTP - File Transfer Protocol

Il protocollo FTP consente il trasferimento di file da e verso un host remoto. Si basa su un modello client-server, il client è la parte che inizia la richiesta di trasferimento mentre il server è in attesa di una connessione sulla porta 21. In realtà non è l'unica porta a disposizione per FTP, infatti il protocollo utilizza due porte per due connessioni. Il protocollo FTP lato client e lateo server ha in comune il fatto che interagisce con il filesystem dei due host: Locale e Remoto.
Un tipico utilizzo di FTP:

1. Il client FTP contatta il server FTP usando TCP sulla porta 21, questa connessione è utilizzata come connessione di controllo
2. Il client invia le credenziali per l'autorizzazione tramite la connessione di controllo
3. Il client sfrutta la connessione di controllo per richiedere una lista dei file remoti e inviare comandi
4. Quando il server riceve un comando per un trasferimento di un file, il server apre una connessione chiamata connessione dati sulla porta 20
5. Terminato il trasferimento il server chiude la connessione

Sulla connessione dati necessitiamo di fare qualche precisazione:

- Al giorno d'oggi la connessione non è iniziata dal server, ma dal client per alcuni casi in cui è presente il NAT (lo vedremo più avanti)
- Per ogni file che richiede il trasferimento si apre una connessione dati

La connessione di controllo utilizza, in alcune situazioni, il meccanismo OUT-OF-BAND, questo meccanismo consente l'accesso a macchine pur avendo il Sistema Operativo non attivo o addirittura spento. Il server FTP mantiene uno stato, quindi mantiene directory corrente e altro.
Comandi e Risposte:

- USER username
- PASS password
- RETR filename (ottenere un file)
- STOR filename (memorizzare un file)
- 331 username OK; password required
- 125 data connection already open;
- 425 can't open data connection;
- error writing file

3.11 DNS - Domain Name System

Il DNS ha vari utilizzi ed è un sistema complesso, infatti il DNS è un database distribuito su una vasta gamma di DNS Server ed inoltre un protocollo a livello applicazione che consente l'interrogazione di tale database. Talvolta viene considerato un protocollo di livello più basso perchè è utilizzato come protocollo di supporto per altri protcolli dall'applicazione (come il browser). Il database e il protocollo offrono un servizio di traduzione, gli host e i router per essere identificati utilizzano 32 bit, ovvero quattro numeri decimali da 0 a 255 separati da un punto come: 75.84.132.2 ovvero un indirizzo IP, ogni numero è un byte. Per una persona è semplice comprendere sia molto complicato memorizzare numeri e numeri, per questa ragione gli host e i router possono essere configurati con un **hostname**, quindi all'indirizzo viene associano un nome. Capito questo punto possiamo capire lo scopo di DNS ovvero la traduzione, o in linguaggio tecnico **risoluzione**, dal nome all'indirizzo IP, questo sistema viene implementato perchè le macchine lavorano meglio con i numeri ma gli esseri umani invece riescono a ricordare meglio i nomi.
I servizi forniti dal DNS sono molteplici, il principale rimane la **traduzione** da nome ad indirizzo e viceversa, vediamo come funziona:

1. Un host esegue il lato client del DNS (o resolver)
2. Il browser estrae il nome di un host, ad esempio www.google.com, dall'URL e lo passa al resolver

3. Il client DNS esegue una query contentente l'hostname a un DNS server
4. Il client DNS riceverà dal DNS server una risposta contenente l'indirizzo IP della macchina a cui deve connettersi
5. Il browser a quel punto avvierà una connessione TCP con protocollo HTTP sulla porta 80 verso l'indirizzo IP fornito dal DNS server

Introduciamo alcuni termini:

- **Host Aliasing:** Un host può avere più nomi associati poichè ogni nome può avere accesso ad un determinato servizio dell'host oppure l'hostname canonico, ovvero il nome "principale" della macchina, è molto complicato da ricordare, di conseguenza un sinonimo riceverà lo stesso indirizzo IP.
- **Mail Server Aliasing:** I server di posta hanno solitamente dei nomi complicati come smtp.gmail.com, grazie a questo servizio dei DNS possiamo semplicemente utilizzare `username@gmail.com` e verrà risolto dal server DNS senza che l'utente sappia eggettivamente il nome canonico della macchina
- **Distribuzione del Carico di Rete:** Vi sono siti, e più in generale macchine, che quotidianamente sopportano traffici intensi, per questo spesso un sono hostname canonico fornisce un insieme di indirizzi IP perchè una sola macchina non riuscerebbe a sopportare il carico altrimenti. I DNS riescono a distribuire il carico tra i vari host, quando il DNS server restituisce una lista di Indirizzi IP esegue una rotazione ponendo come prima, per ogni richiesta fatta, un diverso indirizzo IP dalla richiesta precedente, in maniera da distribuire il carico in maniera equa, questo meccanismo è attuabile dato che i client solitamente si connettono al primo indirizzo IP della lista.

All'inizio della nostra trattazione abbiamo definito il DNS come un database distribuito, l'architettura più semplice per un database sarebbe uno schema centralizzato ma avremmo alcuni svantaggi:

- **Un Solo Punto di Fallimento:** Se si guasta il DNS server tutta internet ne soffre
- **Volume di Traffico:** Impensabile che un solo server regga tutto il traffico di internet

- **Database Distante:** Se è centralizzato significa che il server esiste in un punto specifico del pianeta che non può essere equidistante da tutti gli host quindi per alcuni vi saranno ritardi maggiori che per altri
- **Manutenzione:** Il singolo DNS server avrebbe un database enorme che è difficile da gestitre e aggiornare

Oltre a decentralizzare il database si è notato che conveniva decentralizzare il controllo sull'unicità dei nomi arrivando ad avere un sistema gerarchico. In questo modo si distribuisce delegando l'autorità dei nome e la responsabilità del mapping fra nomi.
Il mapping implica un partizionamento dello spazio dei nomi in domini, tale partizionamento rende più chiara la delega di autorità e responsabilità.

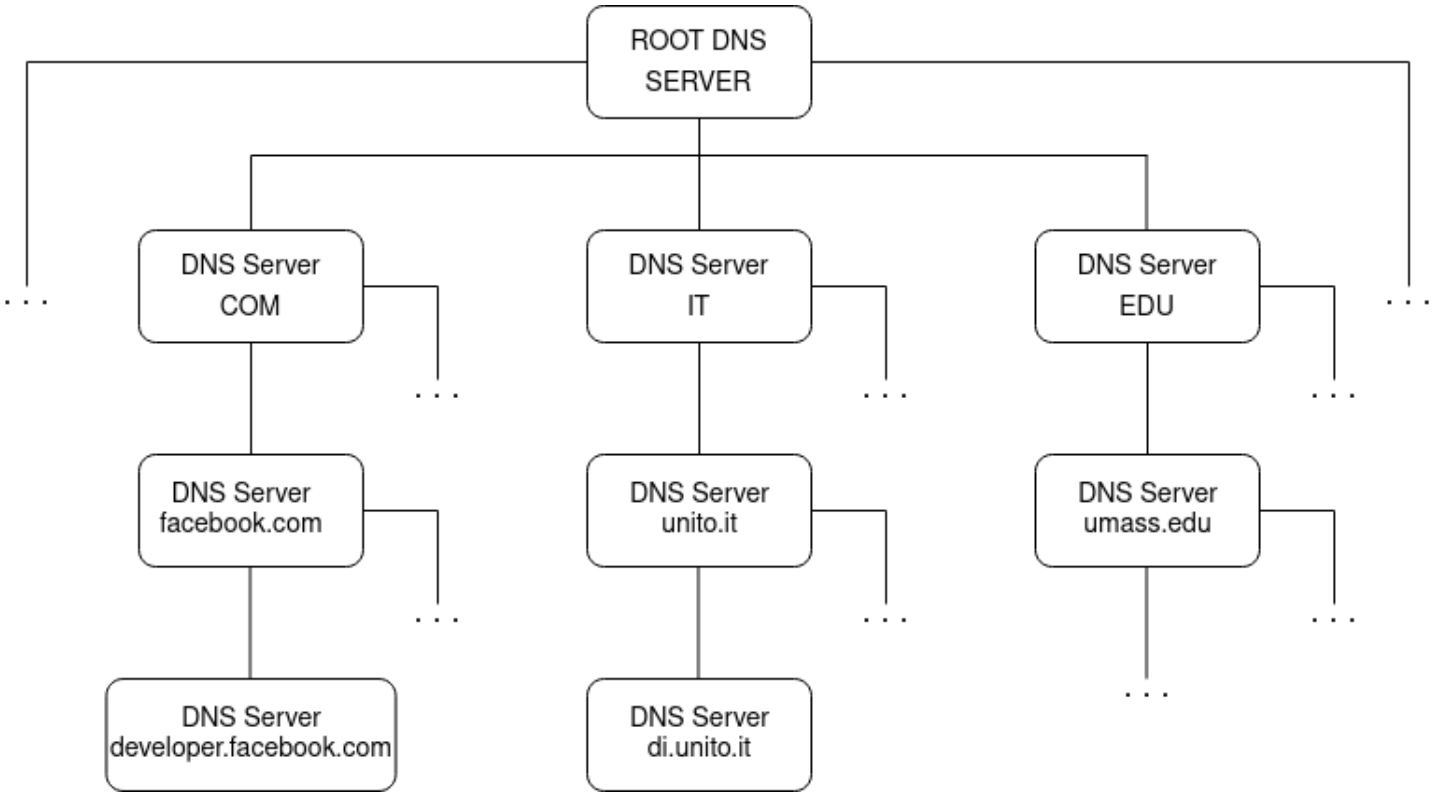


Figura 3.7: Schematizzazione della Gerarchia DNS - Mapping per Autorità e Responsabilità

Da questo albero possiamo distinguere tre grandi famiglie di DNS server:

- **ROOT SERVER:** Esistono circa 400 in tutto il mondo forniscono gli indirizzi dei server TLD
- **TLD SERVER - Top Level Domain:** Server che si occupano dell'autorità e responsabilità di risoluzione di domini di primo livello come COM, IT, JP, EDU, GOV, ...
  Forniscono gli IP dei server autoritativi.
- **DNS SERVER AUTORITATIVI:** Ogni organizzazione che ha host pubblicamente accessibili tramite internet deve fornire record DNS pubblicamente accessibili che associano i nomi degli host agli indirizzi IP. Ovviamente può scegliere se avere un proprio DNS oppure se pagare un'azienda che fornisce tale servizio.

I DNS server autoritativi sono gestiti dalle aziende che a loro volta suddividono il sito o il sistema perchè troppo grande quindi si potrebbe avere una situazione del tipo:

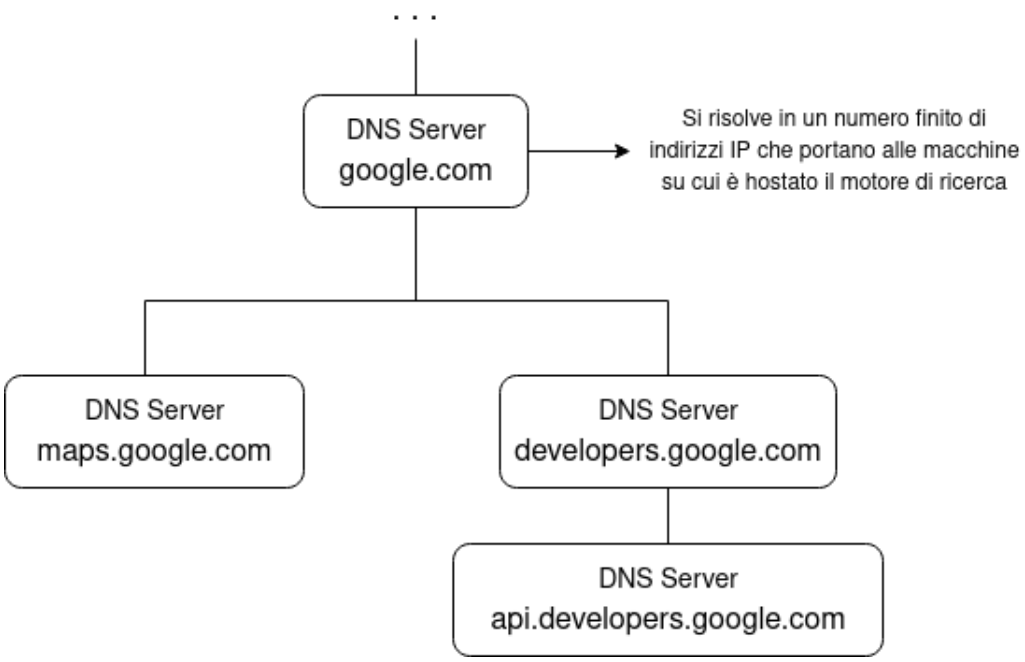


Figura 3.8: Esempio di DNS Autirtativi e propri sotto domini

La figura mostra solo una ristretta situazione ipotetica, la realtà è più grande e complessa. Questo sistema però viene denominato sistema di **Sot-**

toDomini dove il dominio di primo livello è il `.it, .com, .gov, .edu, .org, ecc...`
I server che implementano il database distrivuito di DNS comprendono tutti i **record di risposta**. Un record di risorse (o resource record) contiene:

```
(Name, Value, Type, TTL)
```

Il TTL è il Time To Live ovvero un campo che stabilisce quando una risorsa deve essere eliminata dalla cache.
Name e Value dipendono invece dal campo Type:

- Se `Type = A` allora il Name è il nome dell'host e Value è l'indirizzo IP associato
- Se `Type = NS` allora Name è un dominio (roba.com) e Value è l'hostname del DNS server autoritativo che sa come ottenere gli indirizzi IP degli host del dominio
- Se `Type = CNAME` allora Value indica il nome canonico dell'host per il sononimo in Name
- Se `Type = MX` allora Value è il nome canonico di un mail server che ha il sinonimo in Name

Prima di presentare un esempio di utilizzo del DNS facciamo presente che fuori dalla gerarchia esiste un altro tipo di server DNS detto DNS server locale, serve per avere un server DNS vicino agli host infatti ogni ISP ne possiede diversi e fungono da proxy per risolvere l'intera gerarchia.
L'esempio che facciamo ora è suddiviso in due modalità, la risoluzione dei nomi viene eseguita in due modi:

1. Esecuzione di Query in modalità Iterativa: poniamo il caso che `cis.poly.edu` vuole contattare l'host `gaia.cs.umass.edu`, allora la risoluzione avverrà:
   (a) `cis.poly.edu` contatterà il proprio DNS server locale per chiederne l'indirizzo IP
   (b) Il DNS server locale chiederà al root server l'indirizzo IP in questione
   (c) Il root server risponderà con l'indirizzo del server TLD DNS
   (d) Il server DNS locale allora contatterà il TLD DNS server
   (e) Il TLD DNS server risponderà con l'indirizzo del server DNS autoritativo

(f) Il DNS server chiede l'IP di `gaia.cs.umass.edu` al server DNS autoritativo

(g) Il DNS autoritativo allora risponderà con l'indirizzo IP richiesto

(h) Il DNS server locale fornisce l'IP di `gaia.cs.umass.edu` a `cis.poly.edu`

2. Esecuzione di Query in modalità ricorsiva: poniamo il caso che `cis.poly.edu` vuole contattare l'host `gaia.cs.umass.edu`, allora la risoluzione avverrà:

   (a) `cis.poly.edu` chiede al proprio DNS server locale di risolvere `gaia.cs.umass.edu`

   (b) Il DNS server locale chiede al root DNS server se conosce l'indirizzo IP di `gaia.cs.umass.edu`

   (c) Il root server chiede l'indirizzo interessato al TLD DNS server

   (d) Il TLD DNS server chiede al server autoritativo l'indirizzo IP richiesto

Il server autoritativo risponderà con l'indirizzo IP e a cascata verranno eseguite le risposta all'indietro fino al client che voleva conoscere l'indirizzo.
Questo metodo non viene utilizzato perchè sovraccarica di lavoro il root server.
Le metodologie che abbiamo visto sebbene l'iterativa è la modalità più usata abbiamo una tecnologia che ci aiuta a diminuire ancora di più il carico di lavoro ovvero il **caching**. Ciò che avviene è che il server DNS locale memorizza dei record in maniera tale che non si debba ripetere la query iterativa ogni volta. Ognuno di questi record ha un TTL per poi essere aggiornato. Bisogna prestare attenzione al fatto che questi record posso essere **out-of-date**, cioè che necessitano di un aggiornamento prima che il TTL sia completo. I record sono i medesimi di quelli discussi precedentemente.
Il formato dei messaggi DNS è il seguente:

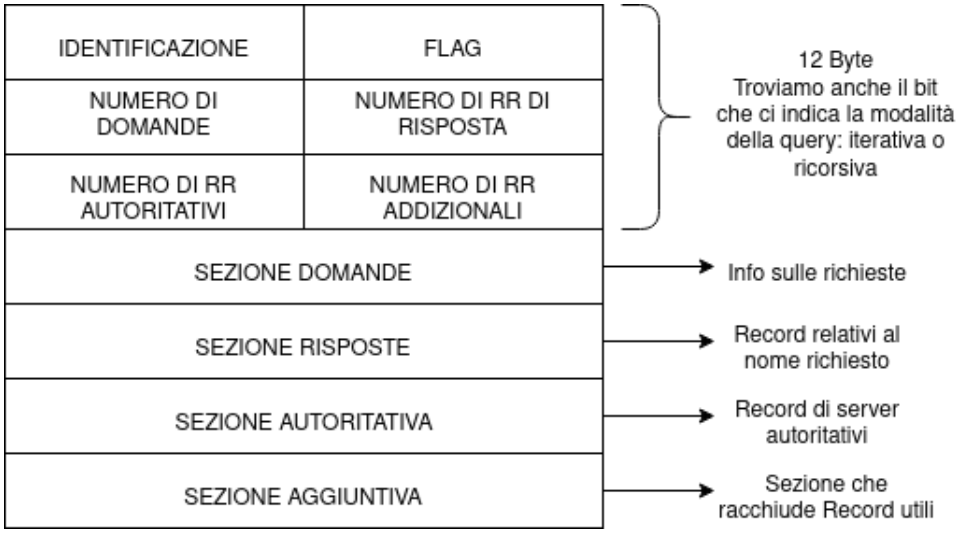


Figura 3.9: Schematizzazione del formato dei messaggi DNS

Una questione importante da analizzare è il **mapping inverso**:
Solitamente il mapping che si vuole utilizzare è da **nome di dominio** a **resource record**, il mapping inverso ci consente di fare l'opposto, nello specifico vogliamo una **inverse query** quindi da indirizzo IP vogliamo risalire al nome. Il formato dei messaggi DNS consente di esprimere una query di questo tipo anche se il protocollo non le implementa. Infatti l'implementazione di tale query risulterebbe molro complessa, per questo motivo sono state create delle query dirette ma particolari: **Pointer Query**.
Le pointer query sono delle interrogazioni che chiedono un RR particolare con `type = PTR` (in realtà identificato dall'intero 12), se si vuole giocare si veda `dns.google`, che utilizzano un nome particolare, la domanda deve contenere un indirizzo IP. Un indirizzo IP però non ha la struttura di un nome di dominio, allora si sono implementati dei nomi di dominio speciali, facciamo un esempio `dns.google` ha come indirizzo IP `8.8.4.4` mentre nel record PTR il nome di dominio sarà: `4.4.8.8.in-addr.arpa`, il dominio `.in-addr.arpa` è un dominio riservato a questo scopo per IPv4.

3.11.1 Server Secondari

Se un server autoritativo è per qualsiasi ragione down allora si ha una situazione di single points of failuer, quindi nessuno può ottenre informazioni. Bisogna però considerare che un server autoritativo può essere molto usato

ed essere sovracaricato creando un collo di bottiglia, in questo caso diventa essenziale dividere il carico in diversi server DNS. Questi due problemi vengono risolti tramite server secondari ovvero server DNS che prendono informazioni su una Zona di Autorità, il protocollo zone transfer permette il trasferimento delle informazioni sul server primario. Non si utilizza UDP per questo protocollo sebbene DNS utilizza sempre questo protocollo sulla porta 53.

3.11.2 Conclusione

Concludendo possiamo dire che DNS ha due aspetti importanti:

- **Astratto:** specifica la sintassi dei nomi, le informazioni associabili ai nomi e le regole per delegare autorità. Se fosse solo questo però la traduzione sarebbe impossibile da coordinare in maneira efficiente.
- **Concreto:** Specifica l'implementazione di un sistema di calcolo distribuito che tradice i nomi negli attributi collegati ad esse in modo efficiente

3.12 Applicazioni P2P

In una architettura client-server abbiamo una dipendenza da una infrastruttura server sempre attivi, in una architettura peer-to-peer (P2P) questa dipendenza è minima o del tutto assente. Infatti i peer, che sono host come computer fissi e portatili controllati dagli utenti, sono interconnessi tra loro e comunicano direttamente l'uno con l'altro, tali peer non sono di proprietà di fornitori di contenuti o di servizi. Una particolarià di questa archiettura è la forte **scalabilità** rispetto ad una architettura client-server- Vediamo il perchè analizzando una applicazione esplicitamente adottata per il P2P: la distribiuzione di un file voluminoso.

3.12.1 Scalabilità

Vediamo con un esempio perchè una architettura cliet-server (cs) è meno perfermante rispoetto ad una P2P, facciamo delle considerazioni iniziali:

- u_s la banda di upload del collegamento di accesso del server
- u_i la banda di upload del collegamento di accesso dell'i-esimo peer
- F la dimensione del file da distribuire, in bit

- N il numero di peer che vuole una copia del file
- D_{cs} il tempo di distribuzione dell'architettura client-server

Definiamo il tempo di distribuzione come il tempo necessario affinchè tutti gli N peer ottengono una copia del file. Per effettuare una analisi pura supponiamo che non vi siano colli di bottiglia nella rete e che i client, i server e i peer non stiamo partecipando a nessuna applicazione di rete aldilà della distribuzione del file.

Architettura Client-Server Osservazioni:

1. Il server deve distribuire F bit a N peer quindi in totale NF bit, dato che la banda di upload del server è u_s, di conseguenza il tempo di distribuzione sarà almeno $\frac{NF}{u_s}$
2. d_{min} è la banda di download del peer che ha il valore più basso quindi $d_{min} = min\{d_1, d_2, ...d_N\}$ allora il peer con d_{min} non potrà che ricevere il file almeno in $\frac{F}{d_{min}}$

Se mettiamo insieme queste affermazioni abbiamo che:

$$D_{cs} \geq max\{\frac{NF}{u_s}, \frac{F}{d_{min}}\}$$

Questo sancisce un limite inferiore al tempo di distribuzione minimo per questa architettura, inoltre si può programmare il server affinchè:

$$D_{cs} = max\{\frac{NF}{u_s}, \frac{F}{d_{min}}\}$$

Ponendoci in questa situazione per N sufficientemente grande il tempo di distribuzione è dato da $\frac{NF}{u_s}$ quindi aumenta linearmente con l'aumentare del numero di peer.

Architettura P2P Osservazioni:

1. Come per l'architettura client-server il tempo di distribuzione minimo dipende dalla velocità di download più bassa quindi è sempre almeno $\frac{F}{d_{min}}$
2. Bisogna consegnare NF bit ovvero F bit a N peer, sappiamo che la velocità di upload complessivo del seistema è $u_{tot} = u_s + (u_1 + u_2 + ... + u_N)$ ovvero quella del server più quella di ciascun peer dato che comunicano tra loro. Quindi il tempo di distribuzione minimo è: $\frac{NF}{(u_s+(u_1+u_2+...+u_N))}$

3. Al principio il file è posseduto esclusivamente dal server che per trametterlo all'interno della comunità dei peer deve inviare ciascun bit del file almeno una volta nel collegamento quindi avremo un tempo di distribuzione minimo di $\frac{F}{u_s}$ perchè poi i peer possono distribuirlo nuovamente tra loro.

Se mettiamo insieme queste informazioni sapremo che:

$$D_{P2P} \geq max\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{(u_s+\sum_{i=1}^{N} u_i)}\}$$

Come per l'architettura client-server questo determina un limite inferiore al tempo di distribuzione minimo nel P2P, tramite una apposita programmazione si può decretare che:

$$D_{P2P} = max\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{(u_s+\sum_{i=1}^{N} u_i)}\}$$

Supponendo che la velocità di upload di tutti i peer è un u possiamo dire che il tempo di distribuzione nel P2P è sempre minore dell'architettura client-server e inoltre ci si accorge di come aumentando il numero di peer nel P2P avremo un tempo di distribuzione che tende asintoticamente a $\frac{F}{u}$ invece che crescere linearmente nel client-server.

3.12.2 Ricerca di Informazioni

L'architettura P2P nasce con lo scopo di distribuire informazioni tra i peer, quindi hanno preso piede principalmente due tipologie di applicazioni:

- **File Sharing:** L'indice tiene traccia dinamicamente della porzione dei file che i peer condividono. I peer comunicano all'indice ciò che posseggono e interrogano l'indice per sapere dove trovare i file.
- **Messaggistica Istantanea:** L'indice crea la corrispondenza tra utenti e posizione, l'indice conosce la posizione dell'utente quando questo aprendo l'applicativo lo informa e interrogano l'indice per conoscere la posizione (indirizzo IP) degli altri utenti.

Questi applicativi hanno una serie di problemi a causa della directory centralizzata:

1. Il trasferimento file è distribuito, ma il processo di localizzazione è fortemente centralizzato
2. Unico punto di guasto
3. Collo di bottiglia
4. Violazione del diritto d'autore

3.12.3 Query Flooding

Un nuovo protocollo P2P risolveva questi problemi di centralizzazione sfruttando il meccaniscmo query flooding. Non vi era nessun server centralizzato ma esclusivamente i peer tra cui il sistema manteneva una connessione TCP, costituendo un grafo tramite cui le richieste venivano propagate per tutto il sistema. Ogni peer ovviamente è connesso a un determinato numero di peer così detti **vicini** che a loro volta avevano altri vicini.
Schematizzando:

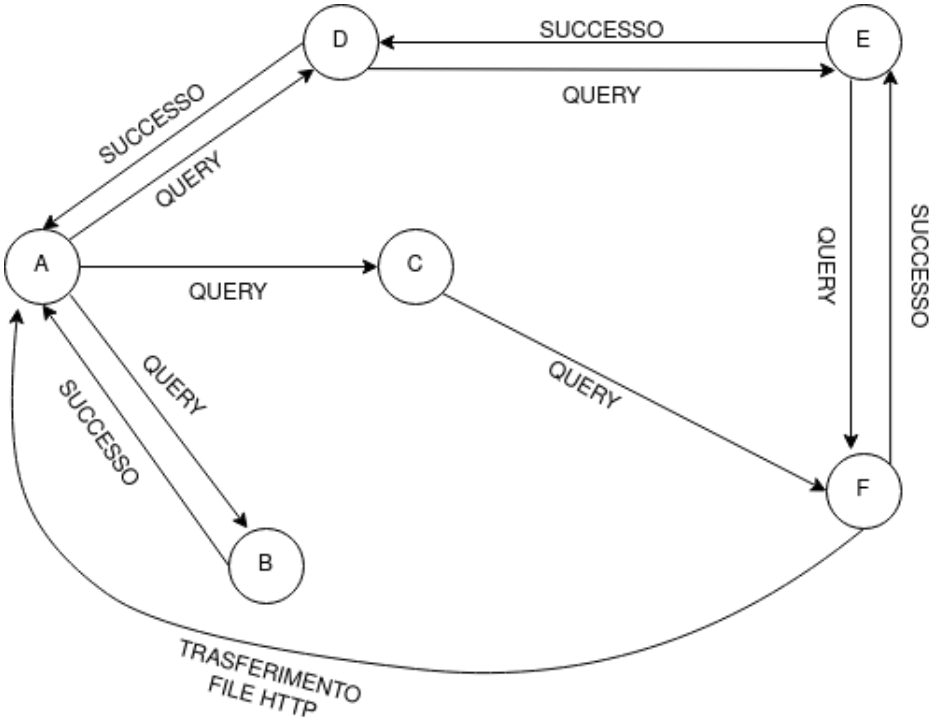


Figura 3.10: Schematizzazione di una Query Flooding

Le query sono identificate da un ID e come si vede in figura alcuni nodi possono essere in comune come per F, infatti F risponde esclusivamente a una query per poi iniziare il trasferimento del file. La scalabilità risiede nel fatto che la query flooding è a raggio limitato.

3.12.4 BitTorrent

BitTorrent è un protocollo P2P rivoluzionario per il suo algoritmo di trading, tale algoritmo implementa un meccanismo che incentiva gli scambi, viene chiamato **Tit-For-Tat**. Questo algoritmo è una parziale rivoluzione all'egoistica scelta di chiudere il programma P2P dopo aver scaricato il file di interesse. Andando con ordine però partiamo dall'inizio capendo come funziona, in maniera superficiale, il protocollo. L'insieme di tutti i peer che partecipano alla distribuzione di un file è chiamato **torrent**. I peer che partecipano ad un torrent scaricano **chunk** di uguale dimensione (solitamente 256 kbyte). Quando un nuovo peer si unisce ad un torrent non ha alcun chunk da condividere ma col passare del tempo ne accumula e mentre scarica invia le parti scaricate ad altri peer. Ciascun torrent ha un nodo denominato **tracker**, ogni peer che si unisce al torrent si registra al tracker a cui periodicamente si connette per informarlo che è attivo. Appena un nuovo peer comunica al tracker la sua unione al torrent il tracker estrae 50 indirizzi IP di peer che già partecipano al torrent, ma può essere un numero qualunque, e li invia al nuovo peer. Il nuovo peer stabilisce connessioni TCP contemporaneamente con tutti i 50 peer, tali peer vengono definiti **vicini**, tale lista nel corso del tempo può cambiare. Il nuovo peer periodicamente chiede ai vicini la lista dei chunk che hanno in maniera da richiedere quelli che gli mancano. Quindi stabilisce quali chunk richiedere e quali invece inviare, per stabilire quali chunk richiedere si adotta il **rarest first** ovvero i chunk più rari tra quelli che ancora mancano, cioè i chunk tra quelli che mancano ma con il minor numero di copie tra i suoi vicini. In questo modo BitTorrent cerca di avere sempre un numero di copie uguali per ciascun chunk. Per stabilire quali richieste il peer deve rispondere si adotta un algoritmo di trading. Alla base dell'algoritmo abbiamo il nostro peer1 che attribuisce priorità ai propri vicini in base alla velocità più alta con i quali condividono i chunk. Per ciascun vicino, peer1 misura continuamente la velocità, di questi ne sceglie 4 con la velocità più alta e contraccambia inviando chunk a sua volta. Ogni tot secondi peer 1 ricalcola la velocità e modifica l'insieme di conseguenza, questi peer vengono detti **unchonked**. Ogni x secondi invece peer1 sceglie un nuovo peer casuale da includere nell'insieme, questo peer viene detto **optimistically unchoked**. Se il nuovo peer, peer2, ha una velocità abbastanza alta può rimuovere nell'insieme. Questo succede anche per peer2 e per tutti gli altri, l'effetto è quello di creare insiemi di peer che si trovano nella velocità e nella mancanza di chunk. La selezione casuale in più stabilisce che i nuovi peer abbiamo chunk da condividere. I **choked** sono invece tutti i nodi che non danno parete degli unchoked. Questo

algoritmo è il Tit-For-Tat.

3.13 Video Streaming e CDN

I siti che forniscono streaming audio e video sono migliaia, come YouTube, Netflix, Amazon, ecc. Il traffico di questi siti rappresenta il 37% del traffico di internet. Prima di addentrarci capiamo brevemente cos'è un video:

> Un video è una sequenza di immagini, visualizzate con un tasso di 24 $\rightarrow$ 30 immagini (frame) al secondo. Una immagine senza compressione e codificata digitalmente consiste in un array di pixel ognuno codificato con un numero di bit che rappresentano le caratteristiche del pixel: luminosità, colore, ecc. I video vengono compressi per raggiungere un compromesso tra qualità e il tasso di bit con cui invitare i bit sulla rete (bit rate). La compressione consente di creare versioni multiple dello stesso video per diversificare la qualità che avreanno bit rate diverso:
>
> 300 kbps, 1 Mbps, 3 Mbps
>
> Così chi ha una banda che lo permette può visualizzare il video a qualità superiori.

3.13.1 Streaming HTTP e DASH

Nello streaming HTTP ogni video risiede su un server con un URL specifico. Il client non deve fare altro che stabilire la connessione TCP e inviare la richiesta GET HTTP, inizierà allora a ricevere il video di risposta HTTP i messaggi vengono salvati in un buffer che quando raggiunge una certa soglia allora inizia la riproduzione visualizzandoli. In questo modo i client riceveranno una sola versione anche se la banda subisse dei cambiamenti. Il provlema è risolto dallo streaming dinamico adattivo su HTTP (DASH), questo protocollo permette a client con accessi diversi a Internet di fare streaming dei video con codifiche diverse e inoltre consente di adottare la scelta della versione in base all'andamento della banda. Ogni versione ha un proprio URL specifico pertanto il server fornisce il file **manifest** che contiene tutti gli URL dei differenti bit rate del video.

3.13.2 CDN - Content Delivery Network

Come ormai sarà chiaro non conviene avere un solo server o un solo data center con tutti i contenuti per un servizio del genere, poichè avremo un

singolo punto di fallimento, potremmo avere colli di bottiglia nella rete e il video ne risentirebbe così come l'azienda. Per superare tutto questo vi sono le CDN, che gestisce server distribuiti in molti punti diversi, conserva copie dei video e altri contenuti e dirige l'utente verso la macchina che può offrire il servizio migliore. Le politiche per la distribuzione dei server da parte delle CDN sono essenzialmente:

1. **Enter Deep:** Pone gruppi di server (cluster) negli ISP di accesso sparsi per il mondo
2. **Bring Home:** I cluster vengono posti in punti chiave come vicini ai PoP e connettendoli attraverso una rete privata

I contenuti sono copiati strategicamente, i contenuti più visti in un determinato paese ad esempio, se un video non è in un cluster allora viene copiato dal cluster più vicino in maniera da abbattere i tempi.

Funzionamento I CDN utilizzano il DNS come un algoritmo di routing:

- Un client fa richiesta di `www.netcinema.com`, dopodichè sceglie il contenuto che URL `video.netcinema.com/6Y7B23V`
- Il client chiede al DNS Locale (LDNS) di risolvere l'URL
- Il LDNS allora chiede al DNS Autoritativo di NetCinema che risponde con l'indirizzo del DNS Autoritativo del CDN con cui ha un contratto
- Il DNS Autoritativo del CDN viene interrogato dal LDNS che restituisce l'indirizzo IP della macchina dove risiede il contenuto
- Il LDNS invia l'IP al client
- Il client si connette all'indirizzo e inizia la riproduzione, se è implementato il DASH il server invia un file descrittivo con tutti gli URL delle varie versioni

Strategie di Selezione Cluster La parte importante delle CDN è la strategia con cui scegliere il cluster a cui indirizzare il client. Abbiamo due approcci:

- **Geografica:** Banalmente il cluster che in linea d'aria è più vicino al client che ha fatto richiesta

- **Traffico:** Si determina il cluster in base alle condizioni del traffico effettuando misure in tempo reale facendo eseguire dei ping verso i LDNS da tutti i cluster

Capitolo 4

Transport Layer

Posto tra il livello applicativo e quello di rete (networking), il livello di trasporto costituisce una parte fondamentale dell'architettura delle reti. Vediamo quali sono i principi alla base del livello di trasporto e l'implementazione nei protocolli utilizzati, TCP e UDP. Il livello di trasporto vedremo come estende il servizio tra due sistemi periferici proprio del livello di rete a un servizio di trasporto tra due processi che lavorano a livello applicativo. Vedremo come si implementa il servizio tramite UDP (protocollo non orientato alla connessione) e tramite TCP (orientato alla connessione). Vediamo poi come tale livello controlla la velocità trasmissiva per evitare problemi di congestione.

4.1 Introduzione

Un protocollo di trasporto mette a disposizione una comunicazione logica tra processi di host differenti, ciò significa che eseguono processi come se fossero direttamente connessi, quindi scambiano messaggi senza preoccuparsi dell'infrastruttura fisica sottostante. Il livello di trasporto è implementato nei sistemi periferici ma non nei router della rete. Il protocollo di trasporto coverte i messaggi in pacchetti noti come **segmenti**, in realtà se il protocollo è UDP i pacchetti vengono chiamati anche **datagrammi**.
Se il messaggio è troppo grande viene spezzettato e a ciascuno parte viene aggiunta una intestazione e viene creato un segmento. Il segmento viene passato al livello di rete sottostante dove viene incapsulato in un pacchetto (datagramma a livello di rete) e viene inviato e fino a destinazione il segmento non viene elaborato dalla rete. Arrivato a destinazione ed ese-

guito lo spacchettamento fino al segmento il livello di trasporto (tramite il protocollo) elabora il segmento e passa al livello applicativo il messaggio.

4.1.1 Livello di Trasporto e di Rete

È possibile non capire bene la differenza che intercorre tra il servizio che offre il livello di trasporto e quello che fornisce il livello di rete. Iniziamo col capire che il trasporto fornisce un metodo di comunicazione logica tra **processi**, mentre il livello di rete si pone come soluzione di comunicazione logica tra host. Per capire questa sottile differenza facciamo una analogia con il sistema postale:

- I messaggi della nostra applicazione sono le lettere
- I processi sono le persone che scrivono le lettere
- Gli host sono dei condomini

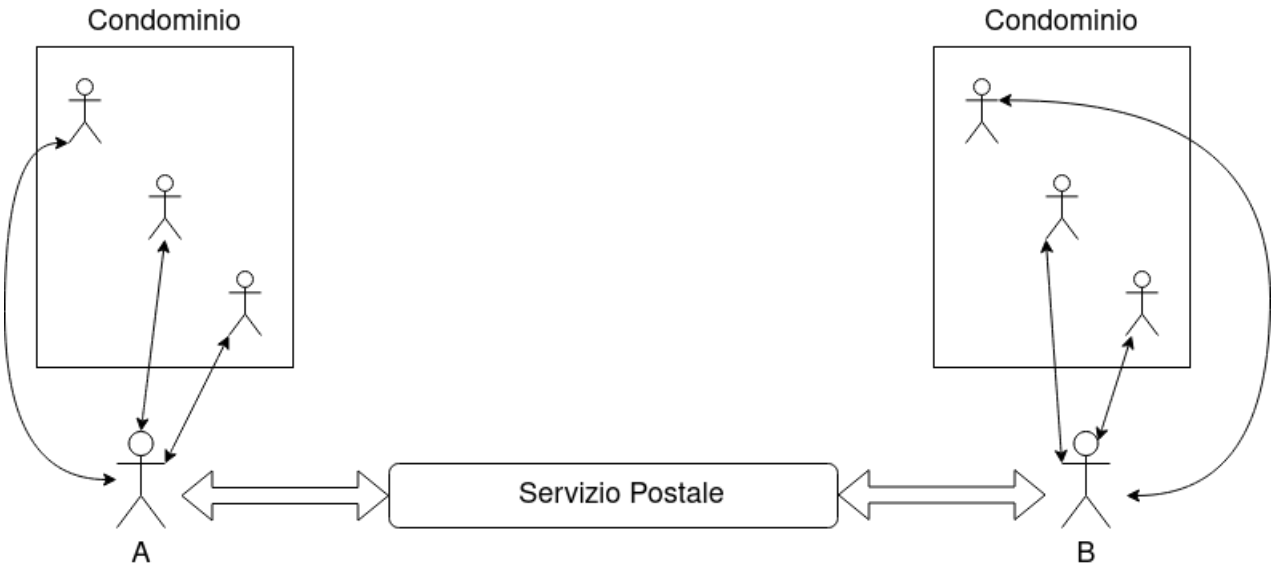


Figura 4.1: Comparazione Livello di Trasporto con un esempio di vita reale nel servizio postale

Quello che abbiamo in figura è A e B che raccolgono le lettere delle persone all'interno di un condominio e le affidano al servizio postare, inoltre si occuopano di smistarle attraverso tutte le persone all'interno del condominio. A e

B però non si occupano di far arrivare le lettere da un condominio all'altro, questo è compito del servizio postale.
In questo caso possiamo dire che:

- A e B offrono il servizio del livello di trasporto
- Il servizio postale invece si occupa di offrire il servizio del livello di rete

4.1.2 Protocolli e Livello di Trasporto di Internet

In Internet, o in una rete TCP/IP, si hanno due protocolli di trasporto: UDP (User Datagram Protocol) che fornisce alle applicazioni un servizio **non affidabile** e **non orientato** alla connessione, TCP (Trasmission Control Protocol) fornisce invece un servizio **affidabile** e **orientato** alla connessione. Lo sviluppatore durante la programmazione dell'applicazione sceglierà quale dei due protocolli utilizzare, e di conseguenza indicare la socjet da creare in base al protocollo. Vediamo brevemenete il livello di rete per capire quello di trasporto. Il protocollo a livello di rete si chiama IP (Internet Protocol), come abbiamo già detto il protocollo oggre una comunicazione logica tra host e il modello su cui è progettato prende il nome di **Best Effort Delivery Service**. Questo modello come dice il nome stesso farà del suo meglio per consegnare i pacchetti del livello di trasporto agli host che vogliono comunicare ma **non offre garanzie**. IP non assicura che i pacchetti saranno consegnati, che l'ordine sarà corretto o che il pacchetto sia integro quindi lo definiamo **non affidabile**, ricordiamo che ogni host ha un **indirizzo IP**. Possiamo definire UDP e TCP come una estensione del servizio offerto da IP col compito di commutare la consegna dei pacchetti "da host a host" a "da processo a processo". Tale passaggio viene fatto sfruttando il meccanismo di **Multiplexing** e **Demultiplexing**. Partendo da UDP, che è il più semplice, quello che offre è essenzialmente la consegna dati da processo a processo e il controllo degli errori, infatti UDP viene definito un servizio non affidabile perchè non garantisce che i dati vengano recapitati. TCP, d'altro canto è più complesso, ma è definito un servizio di trasferimento dati affidabile, infatti grazie a una serie di tecniche TCP riesce ad assicurare la consegna dei dati e l'ordine convertendo di fatto il protocollo non affidabile IP in uno affidabile. In particolare il servizio migliore offerto è il **controllo della congestione** ovvero evita che le connessioni intasino il traffico, di fatto offrendo un servizio a Internet stesso, può raggiungere questo obiettivo grazie alla possibilità di regolare la velocità con la quale il mittente immette traffico nella rete.

4.1.3 Multiplexing e Demultiplexing

Supponiamo che una macchina stia eseguendo due sessioni Telnet e FTP, quindi abbiamo quattro applicativi: due Telnet, uno FTP e uno HTTP. Il livello di trasporto quando riceve i dati dal livello di rete deve indirizzarli a uno di questi quattro processi. Un singolo processo può implementare e gestire una o più socket attraverso cui i dati fluiscono dalla rete al processo e viceversa. Il livello di trasporto trasferisce i dati alla socket e non al processo, quindi ogni socket ha un identificatore il cui formato dipende dal protocollo che si implementa. Ogni pacchetto a licello di trasporto ha dei campi specifici per l'identificatore e vengono esaminati dal protocollo adottato per poi decidere la socket su cui redirigere i dati. Possiamo dire allora che il compito di trasportare i dati dei pacchetti a livello di trasporto verso la socket corretta e detto **Demultiplexing**, il compito di radunare i frammenti da diverse socket, incapsulare per creare i pacchetti e passarli al livello di rete, è detto **Multiplexing**. Parlando dell'identificatore sono il numero della porta di origine e il numero della porta di destinazione. Ogni porta è un numeri di 16 bit e vanno da 0 a 65535, da 0 a 1023 sono le **porte note** riservati a protocolli noti HTTP (80), FTP (21), SSH (22), ecc. Quindi ogni socket nell'host deve avere un numero di porta, e quando un pacchetto arriva all'host il livello di trasporto esamina il numero della porta di destinazione e dirige il pacchetto verso la socket corrispondente. L'identificatore visto a un livello di rete invece è una quadrupla composta da:

- Indirizzo IP **sorgente**
- Indirizzo IP **destinatario**
- Numero di Porta **sorgente**
- Numero di Porta **destinatario**

4.1.4 Multiplexing e Demultiplexing Non Orientati alla Connessione - UDP

Ogni **datagram UDP** (pacchetto UDP) arriva all'host, UDP esamina il numero di porta di destinazione e dirige il datagram verso la porta corrispondente. Quando il livello applicativo richeide la creazione di una socket UDP avremo due situazioni:

1. Lato Client il livello di trasporto assegna alla socket un numero di porta tra 1024 e 65535

2. Lato Server invece è il programma a stabilire un numero di porta determinato poichè sarà la porta su cui dovrà ricevere i datagrams

Quindi una socket UDP viene completamente identificata tramite la coppia (IP address, Numero di Porta) e può essere usata sia come sorgente che come destinazione. Di conseguenza si dice che UDP non è orientato alle connessioni perchè se due segmenti UDP presentano indirizzi IP sorgenti diversi e numeri di porta di origine diversi ma hanno il medesimo indirizzo IP di destinazione e la medesima porta di destinazione allora UDP consegnerà entrambi i segmenti alla stessa socket e quindi allo stesso processo.

4.1.5 Multiplexing e Demultiplexing Orientati alla Connessione - TCP

La gestione di **segmenti TCP** (pacchetto TCP) nel multiplexing e demultiplexing è diversa da UDP, poichè TCP è un protocollo più complesso. Le socket vengono identificate da qiattro parametri:

- IP Sorgente
- IP Destinatario
- Numero di Porta Sorgente
- Numero di Porta Destinatario

La singola socket è identificata dalla coppia di paramentri (IP, numero di porta) che costituiscono un **end-point**, quindi avremo **end-point sorgente** e **end-point destinazione**. Contrariamente a quello che accade utilizzando UDP, in TCP abbiamo che se due segmenti hanno la medesima coppia di parametri IP e porta saranno sempre diretti a **socket differenti**. In questo modello però vi è una eccezione che riguarda i segmenti che trasportano le richieste per stabilire una connessione.
Quindi una applicazione server su un host A si presenta alla rete con una socket di benvenuto che si pone in attesa da parte di richieste di conessione TCP su un determinato IP e un certo numero di porta. Il client che vuole contattare A crea una socket che sia connessa a (IP-A, porta-A). Un segmento di richiesta connessione altro non è che un segmento con porta di destinazione porta-A e una intestazione con un bit speciale settato a 1. Quando il processo server riceve la richiesta crea una nuova socket apposita per quella connessione e rimane in attesa. Quindi il trasporto lato server registra una porta sorgente, un IP sorgente, IP del server, porta del server e con questi valori identifica la socket.

4.2 Protocollo UDP

Il protocollo UDP aggiunge ben poco a IP:

- funzione di multiplexing e demultiplexing
- controllo superficiale degli errori

UDP viene preferito a volte a TCP perchè:

- Si ha un controllo più fine sui dati inviati a livello applicativo
- Nessuna connessione stabilita significa nessun ritardo sull'invio di dati
- Nessuno stato di connessione significa bassa complessità
- Intestazioni brevi si concretizzano in un miglior throughput
- Se implementato correttamente è un protocollo leggero

Un tipico caso di utilizzo di UDP è DNS, poichè evita attese inutili per stabilire una connessione ed è tollerante a perdita di messaggi. Un altro caso è una applicazione multimediale, audio/video, dove si tollera, in maniera contenuta, la perdita di pacchetti. Se l'affidabilità la si vuole ottenere utilizzando UDP la si deve implementare a livello applicativo.
Formato del **datagram UDP**:

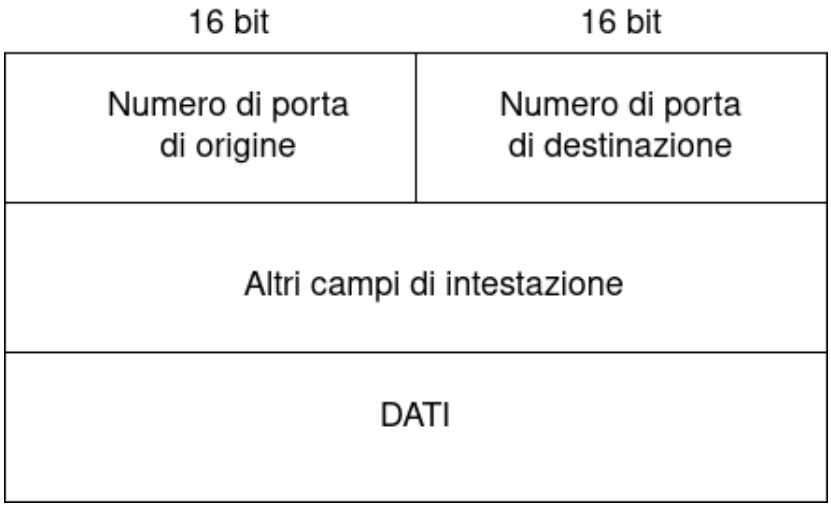


Figura 4.2: Formato datagrammi UDP

Per UDP viene generato uno pseudo-header che però non viene spedito ma serve a calcolare la **checksum** ovvero un metodo di controllo degli errori. Lo pseudo-header ha il formato che segue:

Figura 4.3: Formato Pseudo-Header

Proto sta per protocol e nel caso di UDP è il numero 17, Lunghezza è la lunghezza del datagram UDP escluso lo pseudo-header. Questa piccola complessità è data dal fatto che UDP vuole sapere se IP ha consegnato all'host e se è corretto, non vengono esplicitati nel datagram poichè si ottimizza lo spazio ma ovviamente la socket è (IP, porta) quindi non possiamo calcolare correttamente l'operazione solo con le porte.

4.2.1 Checksum

Nella figura che porta un esempio del formato dei datagrammi UDP, nello spazio "Altri campi di intestazione" vi sono 16 bit dedicati alla lunghezza dell'intero datagramma e 16 bit dedicati al checksum. Come abbiamo già detto il checksum è il meccanismo che ci consente di verificare se vi sono errori o meno.
Funzionamento:

- Lato Mittente:

1. Tratta il pacchetto come sequenze di interi da 16 bit
2. Vengono sommate tutte le sequenze
3. Si fa il complemento a 1 della somma
4. Mette il checksum all'interno del campo specificato

- Lato Destinatario:
  1. Calcola checksum, ogni segmento viene sommato come per il mittente
  2. La somma del precedente punto viene sommato alla checksum del mittente
  3. Verfica se è presente uno 0 allora è stato rilevato un errore e il pacchetto viene scartato

Il checksum viene utilizzato poichè non vi è garanzia che tutti i collegamenti lo facciano e non vi è garanzia che all'interno di un router non vi siano accidentali modifiche. UDP quindi si dice che metta a disposizione un controllo end-to-end. Se incontra un errore dipendetemente dalle implementazioni o viene scortato o reinviato al client con un avvertimento.

4.3 Principi di Trasferimento Dati Affidabile

Uno dei primi problemi che vogliamo risolvere è come effettuare un trasferimento dati affidabile sfruttando il livello di rete che offre un protocollo di trasferimento (IP) non affidabile. Le caratteristiche del canale non affidabile sanciranno la complessità del protocollo di trasferimento dati affidabile: **Reliable Data Transfer o rdt**. Quindi vogliamo creare una situazione del tipo:

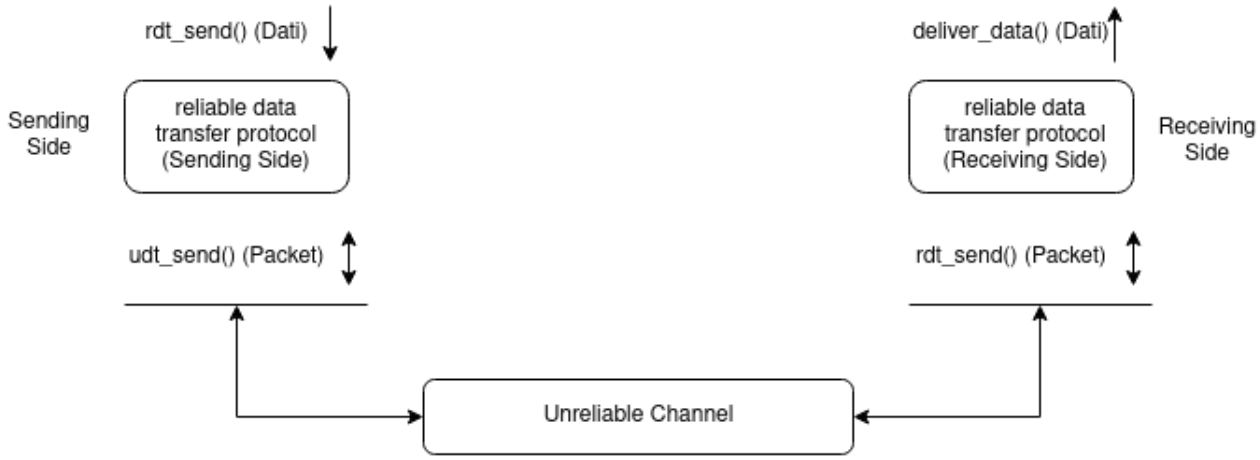


Figura 4.4: Ciò a cui si vuole arrivare come protocollo di trasferimento dati affidabile

Per capire bene come questo sia possibile utilizzeremo Automi a Stati Finiti, se non si conoscono questi strumenti si riprenda un corso di Linguaggi Formali e Traduttori, o Algoritmi e Strutture Dati o un corso ancora più specializzato nel campo matematico.

4.3.1 Protocollo rdt 1.0

Vediamo un trasferimento dati ideale quindi perfettamente affidabile (rdt 1.0):

- Nessun Errore
- Nessuna Perdita

LATO MITTENTE rdt_1.0:

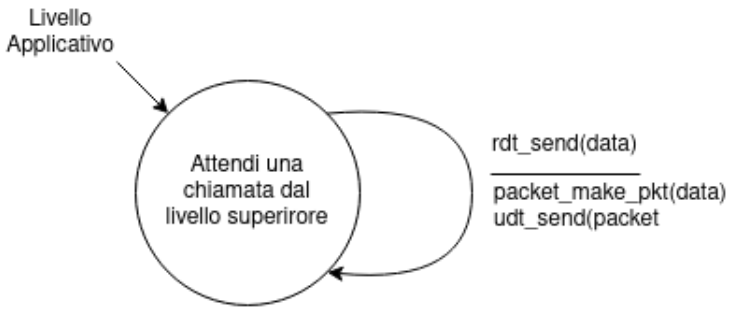


Figura 4.5: Lato Mittente del protocollo rdt 1.0

LATO DESTINATARIO rdt_1.0:

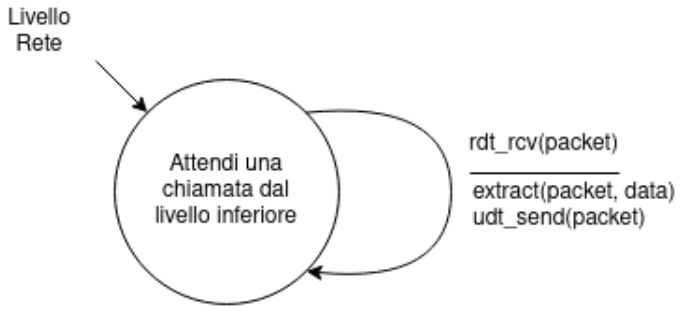


Figura 4.6: Lato Destinatario del protocollo rdt 1.0

Non vi sono informazioni di controllo e i dati vengono spediti a un tasso qualsiasi.

4.3.2 Protocollo rdt 2.0

Ora supponiamo che il livello sottostante possa:

- Modificare Bit del pacchetto
- Non possa perdere pacchetti
- Non possa cambiare l'ordine dei pacchetti

Supponiamo che la checksum permette di scoprire errori sui bit ma vogliamo anche rimediare a una situazione di errore. Introduciamo allora le notifiche

(ACKs) che permettono di comunicare esplicitamente che il pacchetto ricevuto è OK e le notifiche negative (NAKs) che comunicano invece che il pacchetto presenta errori. Se il mittente riceve un NAK ritrasmette il pacchetto.
Quindi arriviamo a rdt_2.0 con:

- Error Detection
- FeedBack

I protocolli di trasferimento dati affidabile basati su ritrasmissioni si chiamano ARQ (Automatic Repeat Request):

- Rilevamento dell'errore
- Feedback del destinatario
- Ritrasmissione

LATO MITTENTE rdt_2.0:

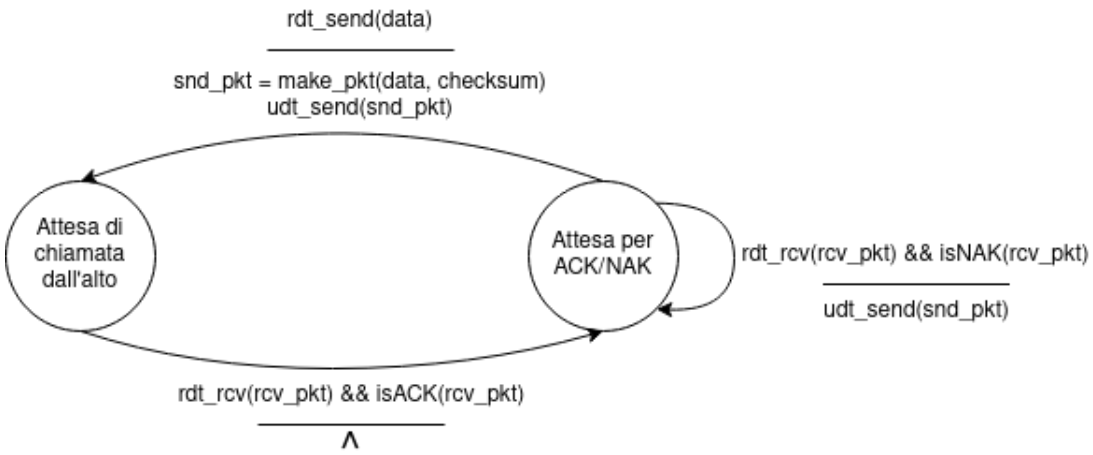


Figura 4.7: Lato Mittente del protocollo rdt 2.0

LATO DESTINATARIO rdt_2.0:

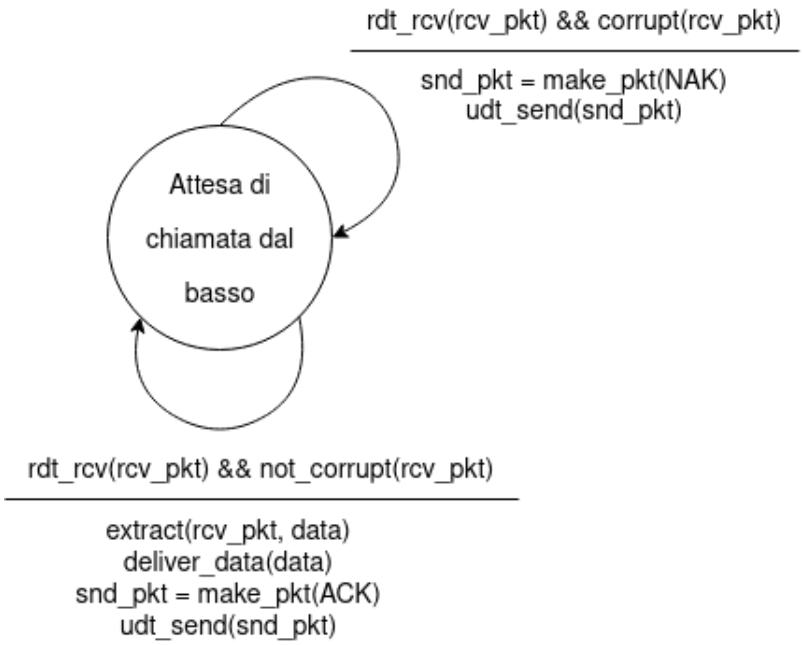


Figura 4.8: Lato Destinatario del protocollo rdt 2.0

Sorge un problema: Il pacchetto ACK/NAK viene corrotto e il mittente non può semplicemente ritrasmettere perchè non sapendo cosa sia accaduto potrebbe generare duplicati.

4.3.3 Protocollo rdt 2.1

Decidiamo che il mittente debba ritrasmettere il pacchetto ma aggiungiamo un nuovo meccanismo **Sequence Number (o Numero di Sequenza)**, numeriamo i pacchetti, così il destinatario riesce a scartare i duplicati.

LATO MITTENTE rdt_2.1:

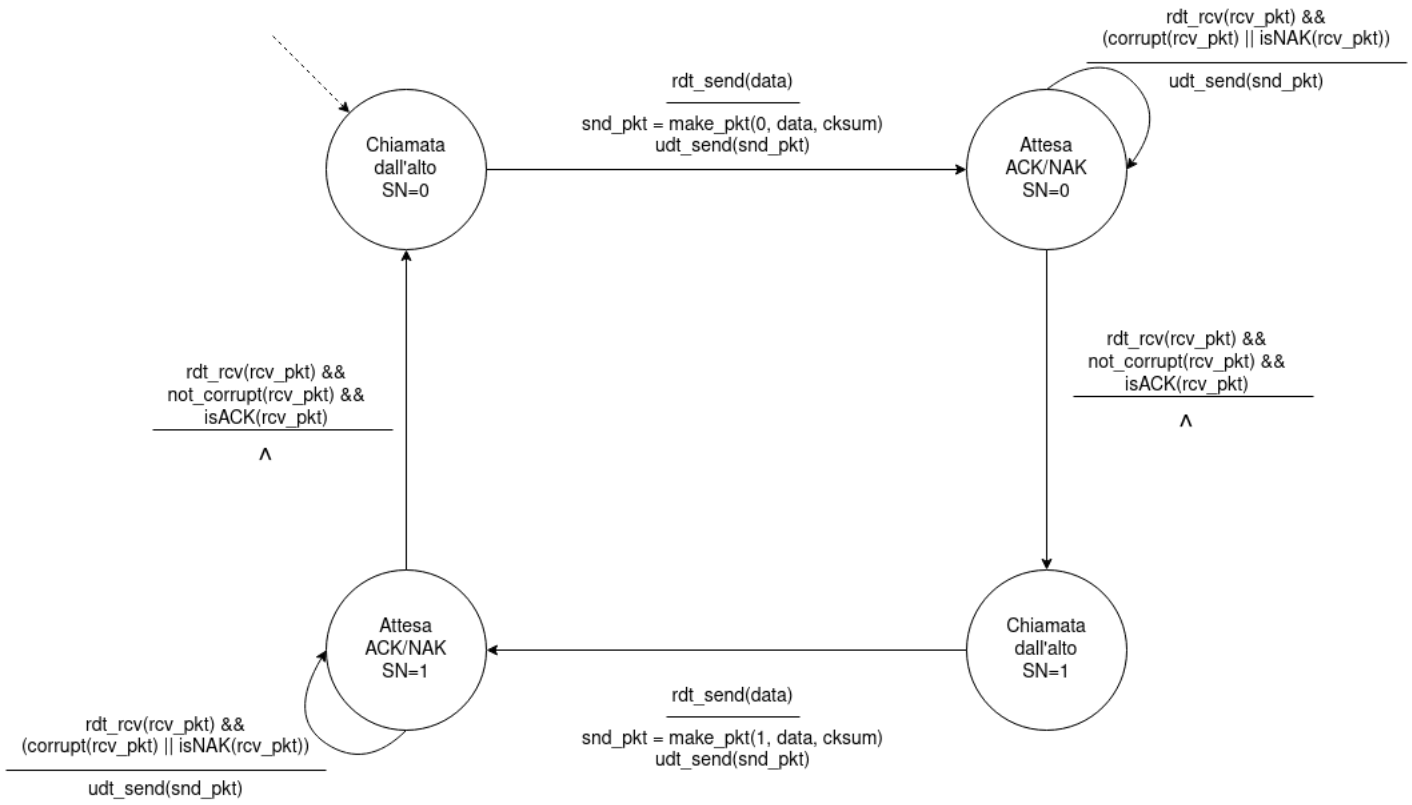


Figura 4.9: Lato Mittente del protocollo rdt 2.1

LATO DESTINATARIO rdt_2.1:

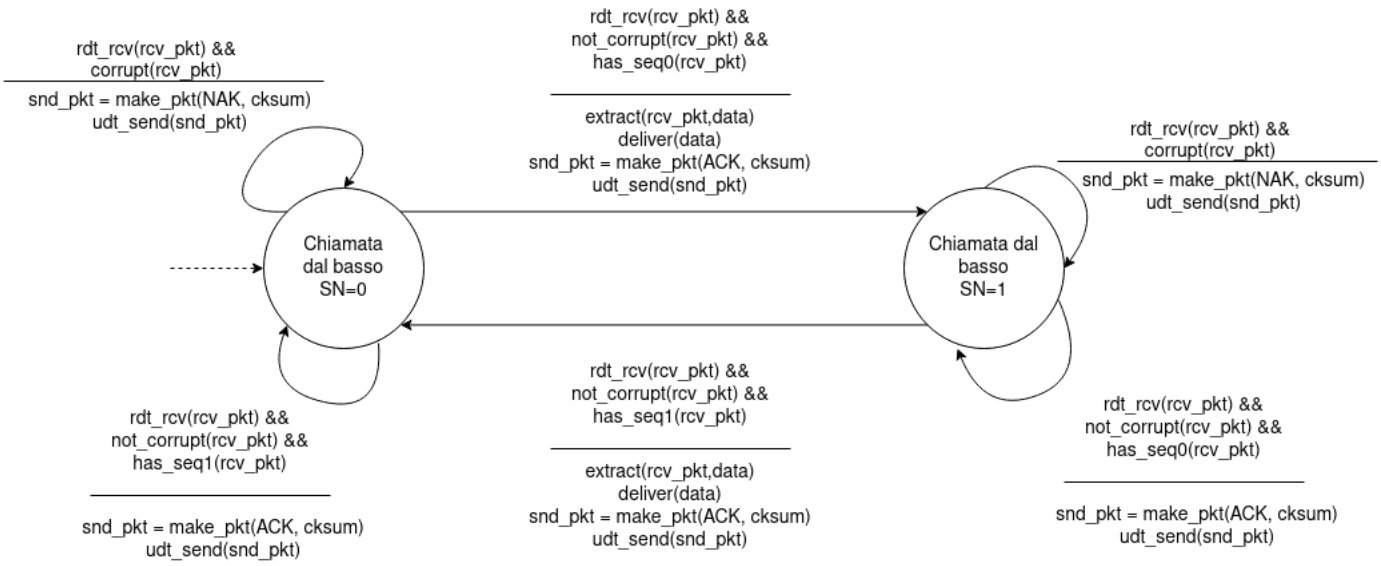


Figura 4.10: Lato Destinatario del protocollo rdt 2.1

Questi due automi spiegano già come mittente e destinatario funzionano però possiamo fare un paio di osservazioni per chiarire meglio come si comporta rdt 2.1:

- Il mittente per far capire al destinatario che i pacchetti sono sequenziali e non ritrasmessi invia alternati 0 e 1 come Sequence Number, la ritrasmissione viene invece effettuata dal mittente avviene se riceve 2 ACK per il penultimo pacchetto inviato, ciò fa capire che l'ultimo non è stato ricevuto

- Quindi per quello detto sopra abbiamo che il destinatario invia un ACK Duplicato nel momento in cui riceve una interruzione nella sequenza quindi o un doppio 0 o un doppio 1, in maniera più formale si può dire che se il destinatario è in uno stato dove attente uno 0 e riceve un 1 (e vale anche il viceversa) allora attua il meccanismo di ACK duplicati

Arrivati a questo punto abbiamo risolto il problema di pacchetti corrotti e il problema dei pacchetti duplicati causato dalla risoluzione del primo.

4.3.4 Protocollo rdt 2.2

Volendo rendere rdt 2.1 più snello e leggero introduciamo una variazione denominata rdt 2.2, questo protocollo è un protocollo che rimane affidabile nel trasferimento dati ma è privo di NAK. Il destinatario ora deve includere il numero di sequenza del pacchetto di cui sta inviando l'ACK, viene fatto includendo rispettivamente ACK0 o ACK1, all'interno della funzione `make_pkt()` il mittente allora dovrà effettuare un controllo nella funzione `isACK()` includendo un argomento 0 o 1. Questo ci consentirà di avere una base affidabile di partenza per trovare una soluzione al problema di smarrimento dei pacchetti. **LATO MITTENTE rdt_2.2:**

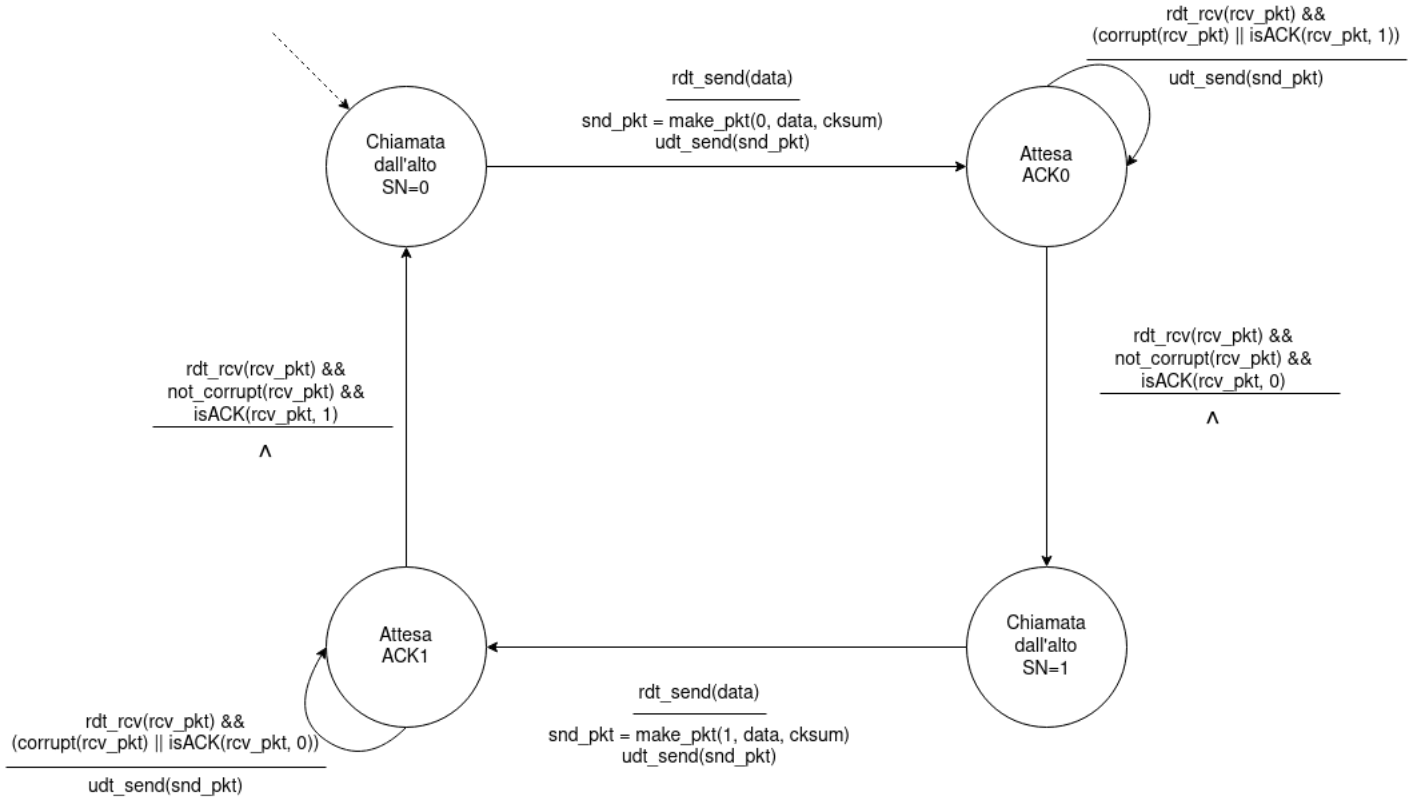


Figura 4.11: Lato Mittente del protocollo rdt 2.2

LATO DESTINATARIO rdt_2.2:

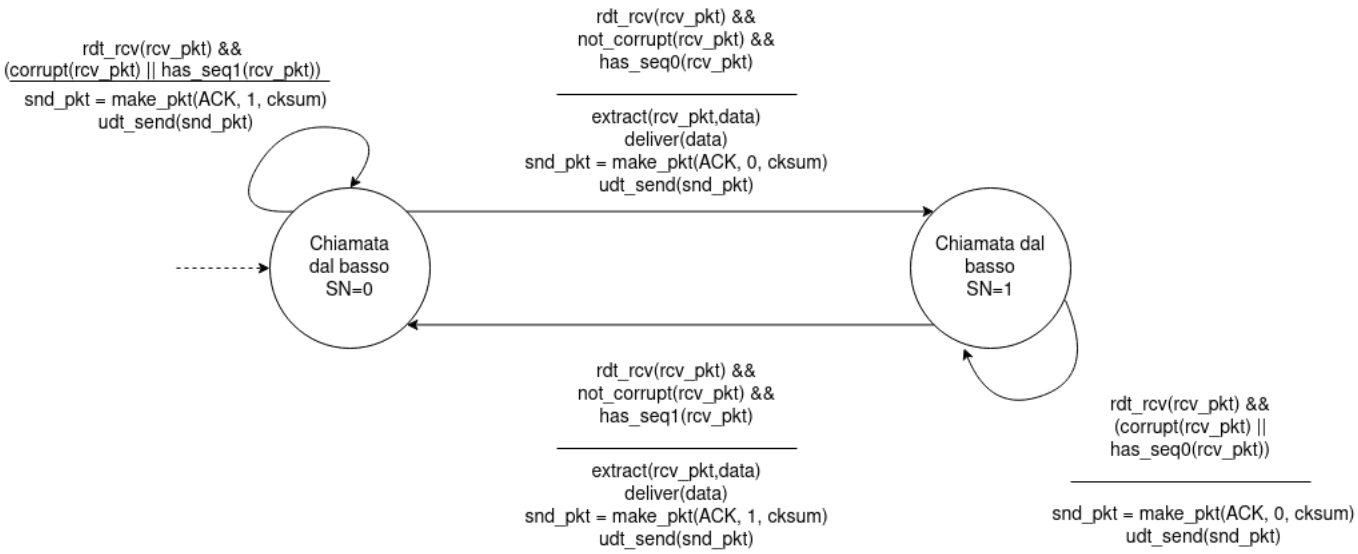


Figura 4.12: Lato Destinatario del protocollo rdt 2.2

Ora passiamo alla risoluzione del problema dello smarrimento dei pacchetti, l'evoluzione di rdt fino alla versione 2.2 è indispensabile per mettere in atto una soluzione ma non è sufficiente quindi abbiamo la necessità di introdurre un nuovo meccanismo: **il Timer**.

4.3.5 Protocollo rdt 3.0

Vediamo l'approccio: il mittente attende un ACK per un intervallo di tempo che possa essere ragionevole, infatti non può essere una quantità di tempo certa perchè anche la stima di un tempo, a causa dei vari ritardi già discussi, è quasi impossibile da fare. Quindi se non riceve un ACK durante questo intervallo il mittente ritrasmette il pacchetto, nel caso in cui l'ACK o il pacchetto dati non fosse perso ma semplicemente in ritardo non abbiamo problemi di sorta perchè implementiamo il numero sequenziale che risolve il problema lato mittente e lato destinatario invece avremo l'ACK con il numero di sequenza del pacchetto a cui fa riferimento. Dicevamo quindi che dobbiamo implementare un **Timer**, con tutte le funzioni necessarie:

- Inizializzazione del Timer
- Risposta con azione appropriata all'internet del Timer

- Fermare il Timer

LATO MITTENTE rdt_3.0:

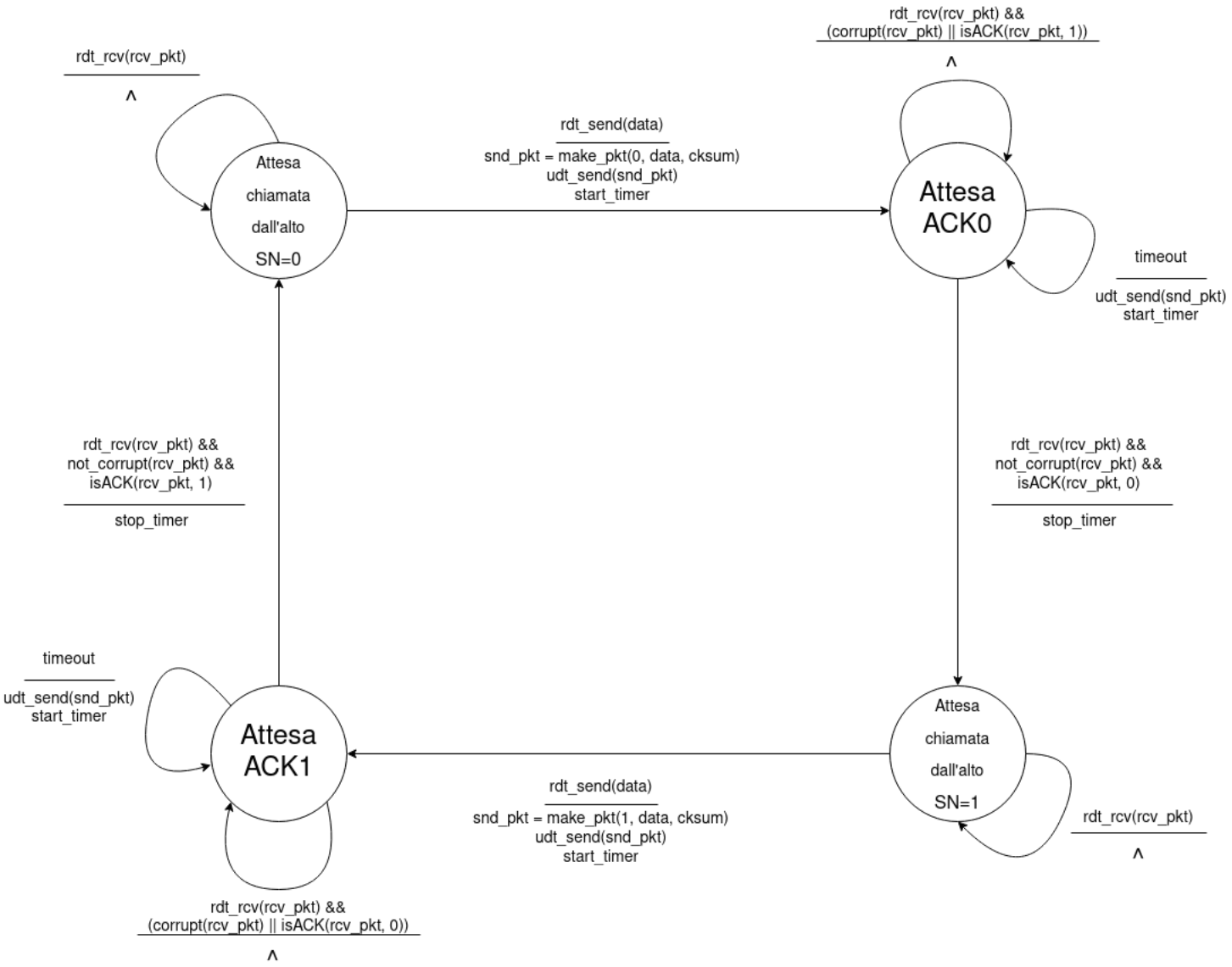


Figura 4.13: Lato Mittente del protocollo rdt 3.0

LATO DESTINATARIO rdt_3.0:

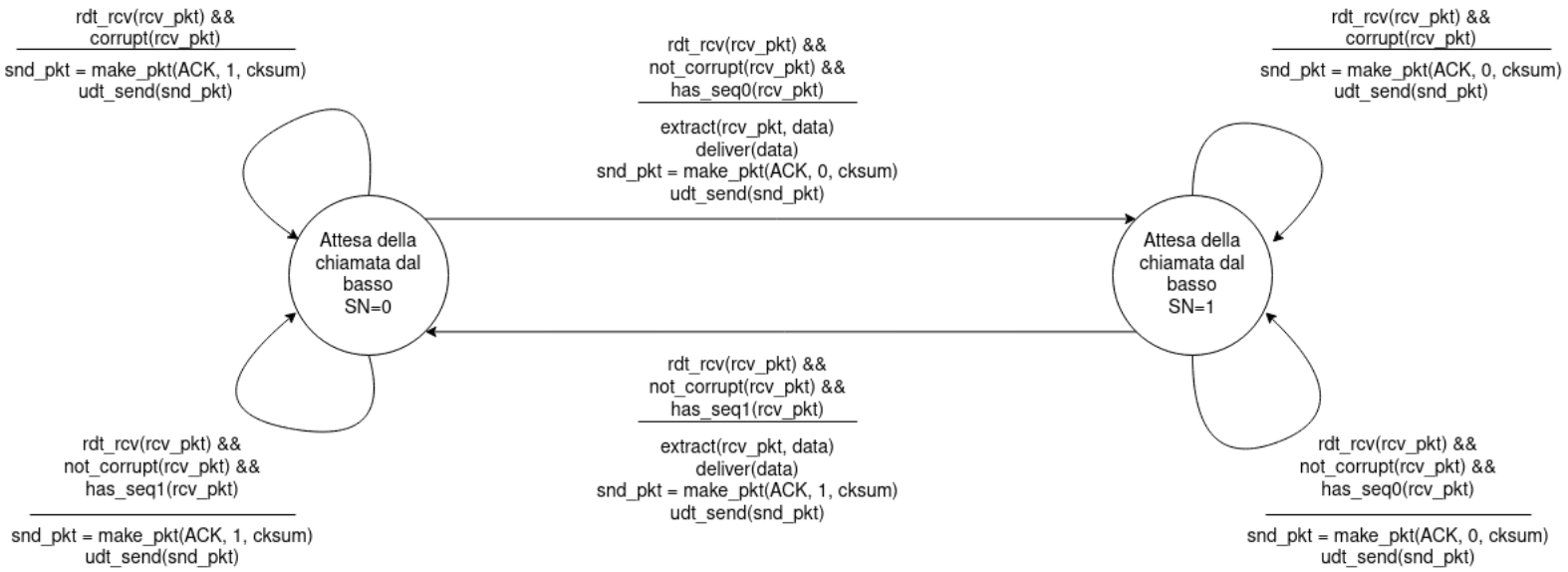


Figura 4.14: Lato Destinatario del protocollo rdt 3.0

4.3.6 Prestazioni rdt 3.0

Immaginiamo di avere due host posizionati sulla costa est e over degli USA. Calcoliamo che il ritardo di propagazione One-Way è 15 millesecondi quindi un RTT è di 30 ms. Il tasso di trasmissione (R) è di 10^9 bit al secondo. La dimensione di un pacchetto (L) è di 1000 byte. Il tempo di trasmissione è pari quindi a:

$$d_t = \frac{L}{R} = \frac{8000bit}{10^9 bps} = 8microsecondi$$

Quindi l'ultimo bit del pacchetto viene immesso nel canale al tempo 8 microsecondi e quindi arriva a destinazione a tempo:

$$\frac{RTT}{2} + \frac{L}{R} = 15,008ms$$

L'ACK sarà di ritorno a tempo:

$$\frac{RTT}{2} + \frac{L}{R} + \frac{RTT}{2} = 30,008ms$$

Quindi l'utilizzo del canale sarà:

$$U_{mit} = \frac{\frac{L}{R}}{RTT + \frac{L}{R}} = 0,00027$$

Pari a 2,7 centesimi dell'1% del tempo. Il throughtput è di 267 kbps su un collegamento che mette a disposizione 1Gbps. Questo è dato dal meccanismo **Stop-And-Wait** del protocollo, poichè deve attendere per ogni pacchetto inviato l'ACK prima di inviare il pacchetto successivo. In alcuni casi tale meccanismo è d'aiuto perchè si ha un RTT basso oppure un $\frac{L}{R}$ alto. Nell'internet odierno però la maggior parte delle situazioni sfavorisce questo approccio e ciò ha portato a sviluppare l'idea del meccanismo di **Pipelining**. Tale meccanismo consente al mittente di inviare molti pacchetti prima che gli ACKs corrispettivi siano arrivati, bisogna però:

- Aumentare l'intervallo dei numeri di sequenza
- Creare bufferizzazione sia lato mittente e sia lato destinatario

Per far ciò si sono sviluppati due protocolli:

- **GO-BACK-N (GBN)**
- **SELECTIVE REPEAT (SR)**

4.4 Go-Back-N

Questo protocollo ci permette di inviare più pacchetti senza dover attendere un ACK il massimo numero di pacchetti lo chiameremo N se disponibile, N quindi è il numero di pacchetti che possiamo inviare in attesa di ACK. Il protocollo prevede tre puntatori ai pacchetti per avere controllo sul flusso:

- **base:** Il numero di sequenza più piccolo che non ha ricevuto ACK. I pacchetti con SN tra 0 e base-1 hanno ricevuto tutti gli ACKs.
- **nextseqnum:** Il più piccolo SN inutilizzato ovvero il primo pacchetto da inviare. I pacchetti nell'intervallo [nextseqnum, base+N-1] sono pacchetti non ancora inviati.

I pacchetti con $SN \geq (base + N)$ non possono ancora essere inviati. Di conseguenza N genera una "finestra" di pacchetti inviabili e scorre in avanti man mano che gli ACK vengono ricevuti. La finestra è limitata ad N poichè vi sono la bufferizzazione e il controllo di flusso che sono problemi da gestire. Nell'intestazione del pacchetto viene inserito il SN che a priori sarà deciso essere lungo k bit e di conseguenza avremo che i SN sono posti in un intervallo $[0, 2^k - 1]$ quindi le operazioni che coinvolgono i SN devono essere in modulo a 2^k.
Il mittente nel protocollo GBN deve affrontare tre casistiche:

- **Invocazione Dall'Alto:** Nel momento in cui il livello superiore invoca `rdt_send()` controlla se la finestra è piena, ossia se vi siano già N pacchetti senza ACK, Se la finestra non è piena viene fenerato un pacchetto e inviato, i puntatori aggiornati. Se la finestra è piena il mittente restituisce al livello superiore i dati e verranno trasmessi in un altro momento.

- **Ricezione di un ACK:** Nel protocollo GBN, l'ACK del pacchetto con il numero di sequenza n verrà considerato un ACK cumulativo, ovvero i pacchetti con un numero di sequenza minore di n sono correttamente arrivati al destinatario.

- **Evento di Timeout:** Quando si verifica un timeout il mittente ritrasmette tutti i pacchetti spediti ma che non hanno ricevuto un ACK. Se su ruceve un ACK ma vi soo ancora pacchetti non riscontrati, quindi senza ACK di ritorno, il timer viene fatto ripartire, se non ci sono pacchetti in attesa di ACK allora il timer viene fermato.

LATO MITTENTE GBN

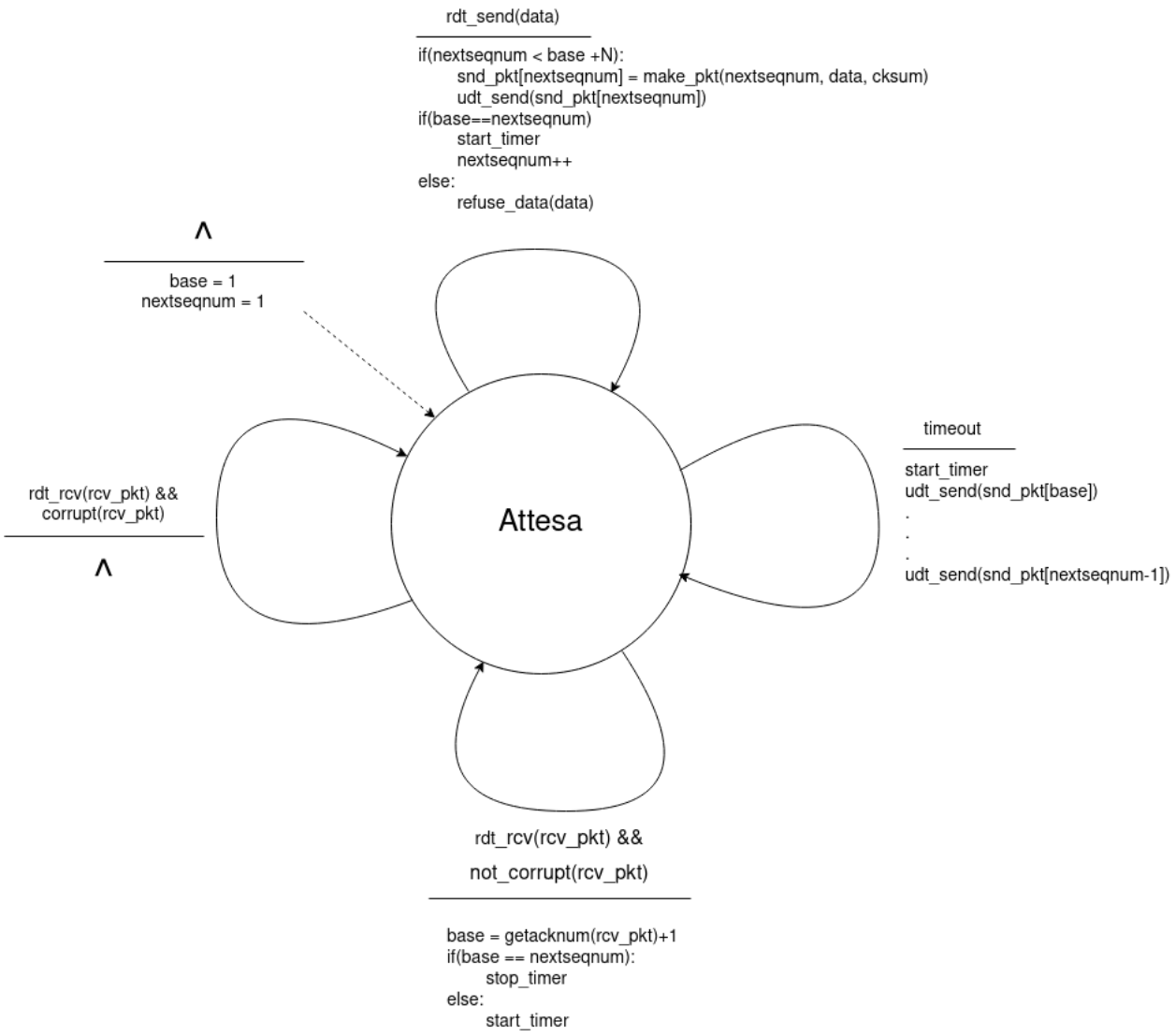


Figura 4.15: Lato Mittente del protocollo GBN

Il destinatario riceve un pacchetto con numero di sequenza n in ordine e corretto, invia un ACK per quel pacchetto e consegna i dati al livello superiore. In tutti gli altri casi il destinatario scarta i pacchetti e rimanda un ACK per il pacchetto in ordine ricevuto più di recente. Se il pacchetto k è stato

ricevuto allora tutti i pacchetti con SN inferiore a k sono stati consegnati poichè il destinatario consegna al livello superiore un pacchetto alla volta. Dato che il mittente deve ritrasmettere tutti i pacchetti dall'ultimo ricevuto correttamente dal destinatario è inutile che il destinatario memorizzi i pacchetti ricevuti correttamente ma non in ordina. L'unico volere da memorizzare è `expectedseqnum` avendo così una semplicità sempre vantaggiosa.

LATO DESTINATARIO GBN

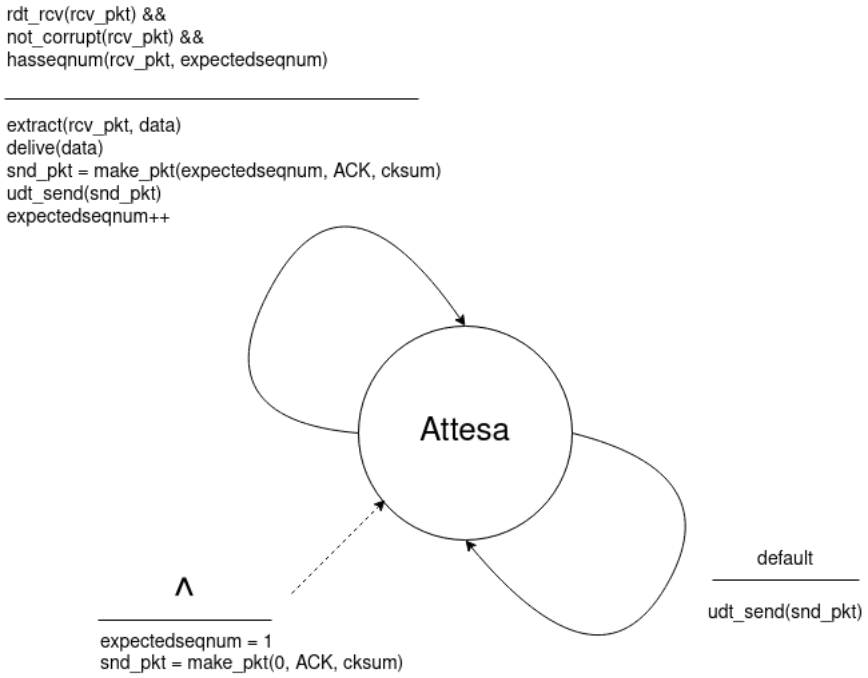


Figura 4.16: Lato Destinatario del protocollo GBN

Per concludere notiamo come in una programmazione basata su eventi, le diverse procedure sono invocate da altre a formare una pila di protocolli.

4.5 Selective Repeat

Come abbiamo potuto notare parlando di **GBN**, siamo riusciti a risolvere il problema delle scarse prestazioni a livello di utilizzo del collegamento. In alcuni casi però questo approccio non è vantaggioso perchè se la finestra è

molto ampia e il prodotto tra larghezza di banda e ritardo è grande allora si troveranno numerosi pacchetti nella pipeline, da questo consegue che un errore su un pacchetto può provocare la ritrasmissione di molteplici pacchetti, questa ritrasmissione cresce al crescere della probabilità di errore.
Il protocollo **Selective Repeat - SR** pone una soluzione a questo problema. I protocolli SR evitano le ritrasmissioni non necessarie e permettono al mittente di ritrasmettere solo i pacchetti per cui si sono verificati dagli errori:

- Il mittente può spedire fino a N pacchetti nella pipeline
- Il destinatario invia ACK individuali per ogni pacchetto
- Il mittente inizializza un timer per ogni pacchetto non ancora riscontrato, al timeout viene ritrasmesso solo il pacchetto associato a quel timer

Come abbiamo detto gli ACK sono individuali e vengono inviati dal destinatario sia per i pacchetti in sequenza che per quelli non in sequenza che vengono immassi in un buffer in attesa dell'arrivo dei mancanti. Il mittente ha le stesse caratteristiche di base del GBN quindi una finestra che prevede al massimo N pacchetti e che limita i pacchetti non riscontrati.Questo modo di comunicare ci porta ad avere, non più un singolo puntatore nel destinatario (`expectedseqnum`), ma un qualcosa di molto simile al mittente avremo dunque una finestra di ricezione impostata sempre su una dimensione di N pacchetti che però slitta in maniera asincrona rispetto a quella del mittente perchè lo slittamento è dettato dall'invio degli ACK individuali.
Vediamo ora le azioni corrispondenti agli eventi che accadono lato Mittente:

- **Dati Ricevuti dal Livello Superiore:** Il mittente controlla il numero di sequenza successivo che se rientra all'interno della finestra allora i dati vengono impacchettati (incapsulamento) e inviati. Altrimenti come accadeva in GBN vengono o salvati in un buffer e restituiti al livello superiore
- **Timeout:** Vengono usati ancora i timer per proteggersi dalla perdita di pacchetti, la particolarità risiede nel fatto che ogni pacchetto ha un timer dato che in caso di timeout solo quel pacchetto verrà ritrasmesso. L'implementazione può essere fatta da un solo contatore hardware. L'implementazione può essere fatta da un solo contatore hardware da cui ne si ricavano molteplici timer logici.

- **ACK Ricevuto:** Quando un ACK viene ricevuto allora il mittente etichetta il pacchetto come riscontrato purchè si trovi nella finestra. Se il numero di sequenza combacia con la base allora la finestra non ha ancora ricevuto un ACK, se in questo caso vi sono ancora pacchetti non trasmessi (magari perchè appena entrati nella finestra) allora vengono trasmessi.

Vediamo ora la controparte, Lato Destinatario:

- **Il Pacchetto con Numero di Sequenza nell'intervallo [rcv_base, rcv_base + N -1] viene ricevuto correttamente:** Al mittente viene inviato un ACK selettivo. Se il pacchetto non era ancora stato ricevuto biene posto nel buffer, se il numero di sequenza è uguale a rcv_base allora viene consegnato al livello superiore insieme a tutti i pacchetti che hanno numeri consecutivi e sono presenti nel buffer.
- **Viene ricevuto un Pacchetto con SN nell'intervallo [rcv_base - N, rcv_base - 1]:** Il pacchetto non viene ignorato ma genera nuovamente un ACK sebbene sia stato già riscontrato (**le finestre non sono allineate!**)
- **In tutti i casi restanti il pacchetto è ignorato**

  IMPORTANTE: Nel Selective Repeat le finestre non sempre coincidono!

Questo può comportare dei problemi, ovvero sono problemi legati alla dimensione della finestra. Dato che le operazioni vengono fatte in modulo 2^k con k numero di bit utilizzati per scrivere il numero di sequenza potrebbe accadere che per il mittente che non ha ricevuto l'ACK relativo a un pacchetto lo ritrasmetta dopo il timeout ma la finestra del destinatario ha operato uno slittamento che fa rientrare il numero di sequenza in modulo e che quindi il pacchetto ritrasmesso venga accettato come nuovo dal destinatario. Questo problema si risolve dando un periodo del SN che sia maggiore della finestra, nello specifico il periodo deve essere $2N$.

4.6 Riepilogo Meccanismi per Trasferimento Dati Affidabile

- **CheckSum:** Utilizzato per rilevare errori sui bit in un pacchetto trasmesso

- **Timer:** Serve a far scadere un pacchetto e ritrasmetterlo, si usa sia per pacchetti smarriti sia nel caso di ritardi che superano il timeout, in questi casi c'è la possibilità che il destinatario abbia duplicati
- **Numero di Sequenza:** Usato per numerare sequenzalmente i pacchetti di dati che fluiscono tra mittente e destinatario
- **Acknowledgment (ACK):** Usato dal destinatario per comunicare al mittente che un pacchetto è stato ricevuto correttemente
- **NAK (Negative ACK):** Usasto dal destinatario per comunicare al mittente che un pacchetto è stato ricevuto con errori
- **Finestra e Pipeling:** Il mittente può essere forzato a inviare solo pacchetti con un numero di sequenza in un determinato intervallo

4.7 Protocollo TCP - Protocollo Connection-Oriented

Il protocollo TCP è un protocollo di trasferimento dati **Stream-Oriented** ovvero non esiste più il concetto di messaggi ma si parla di stream di byte. È inoltre un protocollo orientato alla connessione quindi avremo una fase di handshake iniziale tramite cui, per mezzo di alcuni scambi preliminari, le due parti si accordano sui parametri della connessione che avverrà. Una peculiarità di TCP è il controllo di flusso quindi il mittente non sovraccaricherà il destinatario di pacchetti.

Struttura dei Segmenti TCP:

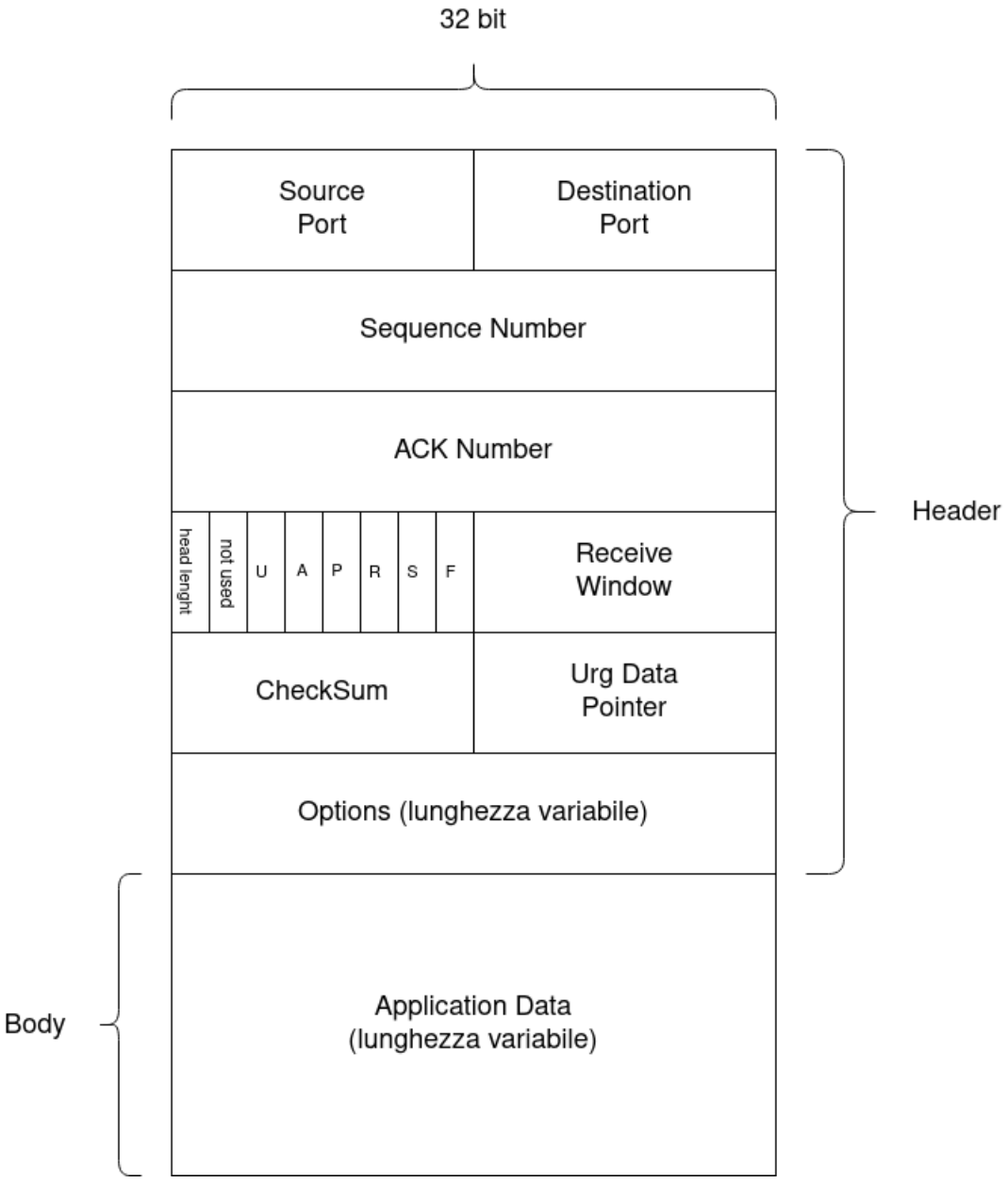


Figura 4.17: Formato dei Segmenti TCP

Ogni segmento inizia con una intestazione (header) che ha come grandezza minima 20 byte poichè sicuramente avremo 5 parole da 32 bit ciascuna.

In più è possibile avere il campo `options` che ha lunghezza variabile e quindi anche non essere presente. Tralasciamo la prima parloa formata da due campi relativi alle porte dato che abbiamo già parlato a cosa servono precedentemente.
Iniziamo parlando dei numeri di sequenza e dei numeri di ACK. Bisogna prima di tutto capire come i numeri di sequenza non si riferiscano al segmento bensì al flusso di byte che compone il segmento:

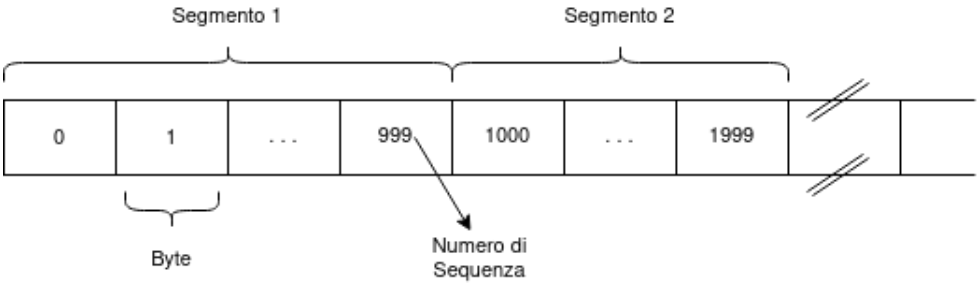


Figura 4.18: Formato dello stream di byte

Quindi possiamo capire come TCP non veda i byte in maniera strutturata, cioè non vede la suddivisione tra i vari campi ma il flusso è ordinato. Il numero di sequenza in TCP esprime lo scostamento (in bytes) dell'inizio del segmento all'interno del flusso. Il numero di ACK ha significato, cioè è valido, solo se il bit A (ACK) in figura è settato a 1, il numero in sè specifica il valore del prossimo numero di sequenza che il mittente si aspetta di ricevere dal destinatario. Questo ci fa comprendere che il numero di ACK fa riferimento ad uno stream differente! TCP utilizza il meccaniscmo di ACK cumulativo. I segmenti fuori ordine non hanno una gestione standard ma vengono gestiti in maniera differenti in base all'implementazione di TCP che si utilizza. Nella struttura troviamo poi:

- **Head Lenght:** Ovvero 4 bit che ci indicano la lunghezza dell'header, la lunghezza deve essere data a causa del campo options che ha lunghezza variabile
- **Not Used or Reserved:** Sono 4 bit impostati a 0 che vengono lasciati per futuri sviluppi di TCP

Nella figura abbiamo poi inserito 6 bit a formare 6 campi da 1 bit ciascuno, nella realtà quei campi sono 8 poichè vi sono anche i bit `CWR` e `ECE` che vengono usati per il controllo della congestione e quindi li vedremo più avanti.

Dei 6 bit presentati abbiamo visto il bit A o ACK per la validazione del campo $ACKNumber$ e ora vedremo i bit P o PSH che sta per Push e il bit U o URG che sta per Urgent. Il bit Push introduce una variazione del comportamento standard di TCP, se viene settato a 1. Il comportamento di default di TCP è quello di ritardare entro certi limiti la consegna dei dati al destinatario per una questione di ottimizzazione, se la Maximum Segment Size (MSS) è di un certo valore allora TCP tenterà di utilizzarla quanto più possibile al massimo anche se questo comporta il ritardo dell'invio del segmento, magari perchè i dati stanno per essere prodotti, così facendo TCP diminuisce l'**overhead**, tale approccio viene implementato sia dal mittente che dal destinatario. Questo approccio però può essere dannoso per alcune applicazioni dove non si possono tollerare ritardi. Allora TCP mette a disposizione una primitiva PUSH che settano a 1 il bit PSH informa il ricevente che il segmento è stato forzato e che bisogna subito consegnare all'applicazione ricevente i dati non appena vengono richiesti. Quindi di fatti PSH=1 impedisce al TCP ricevente di trattare i dati, una cosa importante da ricordare però è che PSH=1 non forza l'applicazione ricevente a consumare i dati. Nelle applicazioni distribuite è necessario avere la possibilità di inviare dati urgenti che almeno funzionalmente sembri che viaggino su un canale diverso (Out of Band) bisogna quindi avere la possibilità di segnalare che ci sono dati urgenti al ricevente in maniera che smetta di fare ciò che sta facendo e legge da TCP i dati urgenti. Dato che TCP non ha un canale dedicato ai dati urgenti utilizza il campo **Urgent Pointer** o **Urg Data Pointer** e il bit U o URG.
Urgent Pointer se significativo indica l'ultimo byte degli urgenti quindi il ricevente deve leggere fino a quel vyte per avere la certezza di aver letto tutti i dati urgenti, il puntatore è da intendersi come **offset** a partire dal numero di sequenza corrente.
Il bit R o RST viene usato per reinizializzare la connessione nel caso di problemi. Il bit S o SYN viene usato con ACK per stabilire una connessione:

- **SYN = 1** e **ACK = 0** $\rightarrow$ Richiesta di Connessione
- **SYN = 1** e **ACK = 1** $\rightarrow$ Connessione Accettata

Il bit F o FIN dichiara che la connessione può terminare perchè il mittentenon ha più dati da inviare.
Il campo **Window Size** o **Receive Window** indica quanti byte possono essere inviati a partire da quello che ha ricevuto nel campo **ACK Number**. Se il campo è impostato a 0 significa che i byte fino a $ACK\_NUMBER - 1$ sono stati ricevuti ma il ricevente non ho avuto modo di consimarli e che

quindi non ha altro spazio nel buffer. Il ricevente autorizzerà di nuovo l'invio di dati con lo stesso $ACKNumber$ ma $WindowSize > 0$.

4.7.1 Timeout e Stima del RTT in TCP

Per capire un valore plausibile di timeout TCP utilizza un meccanismo chiamato **Exponential Weighted Moving Average (EWMA)**. Definiamo RTT (Round Trip Time) come il tempo che intercorre tra l'invio di dati e la ricezione della risposta del destinatario. Quindi l'intervallo di tempo tra l'istante t_0 in cui il segmento inizia ad essere trasmesso dal mittente e l'istante di tempo t_1 in cui il mittente riceve il messaggio di ACK.
Allora:

$$RTT = t_1 - t_0$$

Ciascun segmento avrà un proprio RTT poichè per tutto quello visto finora sappiamo che nella rete ci possono essere ritardi TCP però non calcola la RTT in caso di Ritrasmissioni. Ogni RTT calcolato prende il nome di **Sample RTT**. Il meccanismo EWMA pesa la storia, ovvero ciò che è stato finora la stima dell'RTT e il campione corrente. Quindi indicano con **EstimatedRTT** il valore medio di RTT abbiamo che:

$$EstimatedRTT = (1 - \alpha)EstimatedRTT + \alpha SampleRTT$$

α è un valore di correzione posto solitamente a $0,125$. Ogni SampleRTT viene incluso per effettuare un ricalcolo di EstimatedRTT.
Il **Re-Trasmission Time Out (RTO)** è l'intervallo di tempo massimo che può trascorrere tra l'istante di trasmissione di un segmento e l'istante di ricezione dell'ACK prima che TCP sorgente ritrasmetta il segmento. Valori piccoli di RTO possono provocare numerose ritrasmissioni spesso non necessarie mentre valori grandi introducono tempo troppo elevati per la ritrasmissione di segmenti persi. La valutazione migliore è quello di impostare RTO al valore di EstimatedRTT più un margine che dipende dalla fluttuazione dei valori di SampleRTT.
Quindi:

$RTO = EstimatedRTT + 4 * DevRTT$
Dove DevRTT è la stima della varianza di RTT e si calcola:
$DevRTT = (1-\beta)DevRTT + \beta|SampleRTT - EstimatedRTT|$
$\beta = 0,25$

4.7.2 Trasferimento Dati Affidabile - TCP

Dato che TCP è un protocollo affidabile sappiamo che di dati in arrivo è esattamente quello spedito per fare ciò si utilizza:

- Pipeling
- ACK Cumulativo
- Timer per la ritrasmissione
  - Timeout
  - ACK duplicati ricevuti

Eventi Lato Mittente Per il momento ignoriamo ACK duplicati e il controllo di flusso e congestione:

- **Dati Ricevuti dal Livello Applicativo (tramite Socket)**
  - Crea un segmento con seqnum che è il numero del primo byte nello stream di dati nel segmento
  - Fa partire un timer se non era già attivo
    - Il timer si riferisce al più vecchio segmento spedito che non ha ricevuto un ACK
    - Timeout Interval è l'intervallo di timeout
- **Timeout**
  - Ritrasmette il segmento che ha causato lo scatto del timeout
  - Fa ripartire il timer
- **Ricezione ACK**
  - Se l'ACK si riferisce ad un segmento non ancora riscontrato si aggiornano i segmenti riscontrati e si fa ripartire il timer se ci sono ancora segmenti da riscontrare

Raddoppio del RTO Il valore dell'RTO viene ricavato tramite le stime viste precedentemente nei casi in cui si riceve un ACK oppure si ricevono dati dall'applicazione. Se però RTO deve essere reimpostato a causa dello scatto del timeout viene (entro certi limiti) raddoppiato, non venendo quindi ricavato dalle stime. Quindi il valore di RTO cresce esponenzialmente agli

scatti del timeout, poichè il mittente reinvia il segmento con il SN più piccolo non ancora riscontrato e il raddoppio dell'RTO cerca di porre soluzione. Ovviamente se cresce troppo viene annullato poichè ci potrebbero essere provlemi più gravi di connessione. Questo è un primo meccanismo, molto limitato, di controllo della congestione noto come **Exponential BackOff**.

Fast Retransmit Come abbiamo bisto RTO può essere eccessivamente lungo e quindi prima di ritrasmettere un pacchetto può passare un tempo troppo grande e quindi inutile. Grazie alla tecnica di Fast Retransmit tramite **Triple Duplicate ACKs** possiamo rendere più efficiente il meccanismo di ritrasmissione. Poniamo il caso in cui il mittente invia S segmenti di cui il secondo viene perso, riceverà allora il primo ACK per il secondo segmento come risposta al primo, poi invierò il secondo che viene perso e alla ricezione del destinatario dei restanti tre saranno inviati tre ACK per il secondo pacchetto mai ricevuto. Allora il mittente anzichè attendere lo scadere del timeout invierà nuovamente il secondo segmento poichè capirà che è stato perso. Il numero di ACK necessari è tre per una questione sperimentale, TCP inoltre ricordiamo che dei segmenti fuori ordine non ha una gestione standard ma varia da implementazione a implementazione.

4.7.3 Generazioni di ACK in Base ad Eventi

EVENTO	AZIONE DEL RICEVENTE TCP
Arrivo ordinato di segmento con SN atteso. I precedenti segmento sono riscontrati.	ACK ritardato. Attende fino a 500 millisecondi per l'arrivo ordinato di un altro segmento. Se non arriva il successivo invia l'ACK
Arrivo ordinato si segmento con SN atteso. Un altro segmento ordinato è in attesa di trasmissione ACK.	Invia un ACK cumulativo immediatamente riscontrando entrambi i segmenti ordinati.
Arrivo non ordinato con SN superiore a quello atteso. VIene rilevato un buco.	Invia un ACK duplicato indicando il numero di sequenza del prossimo byte atteso, praticamente l'estremità inferiore del buco.
Arrivo di segmento che colma parzialmente o completamente il buco.	Invia un ACK immediatamente se il segmento ricevuto comincia dall'estremità inferiore del buco.

Figura 4.19: Tabella riassuntiva di generazioni di ACK in base ad Eventi

4.7.4 Prima Conclusione TCP

Avendo visto i protocolli GBN e SR ci viene spontaneo domandarci se TCP è nella categoria GBN oppure nella categoria SR. La risposta è come sempre nel mezzo! TCP è un protocollo ibrido tra GBN e SR. Infatti TCP ha delle caratteristiche GBN come l'utilizzo di ACK cumulativi e ha delle caratteristiche SR come le possibilità di ritrasmissione di segmenti singoli per compensare perdite e smarrimenti.

4.7.5 Controllo di Flusso

Il controllo di flusso ci consente di regolare la velocità del mittente in base alla velocità del destinatario. Questo viene implementato poichè il destinatario come sappiamo è provvisto di un buffer che viene riempito del mittente,

se l'applicazione del destinatario ha una velocità bassa di consumazione dei dati all'interno del buffer rispetto alla velocità con la quale il mittente immette i dati nel buffer, questo causa un overflow del buffer. Questo problema si risolve implementando un meccanismo di controllo del flusso basato su una finestra di trasmissione e una finestra di ricezione le quali dimensioni possono variare nel tempo.

Lato Destinatario Lato Destinatario abbiamo due variabili che vengono aggiornate:

- `Last_Byte_Rcvd:` Il numero d'ordine dell'ultimo byte che il destinatario ha ricevuto dalla connessione
- `Last_Byte_Read:` Il numero d'ordine del byte più recente che l'applicativo ha prelevato dal buffer

Per non avere un caso di overflow si deve verificare:

$$Last\_Byte\_Rcvd - Last\_Byte\_Read \leq RcvBuffer$$

Inoltre il destinatario tiene aggiornato la variabile `rwnd` ovvero il numero di byte che sono liberi nel buffer:

$$rwnd = RcvBuffer - (Last\_Byte\_Rcvd - Last\_Byte\_Read)$$

Tale variabile viene inviata al mittente per far sapere quanti byte sono ancora disponibili per la memorizzazione di nuovi dati.

Lato Mittente Il mittente mantiene due variabili:

- `Last_Byte_Sent:` Il numero d'ordine dell'ultimo byte che il mittente ha inviato nella connessione
- `Last_Byte_Acked:` Il numero d'ordine dell'ultimo byte di cui il mittente ha ricevuto un ACK dal destinatario

Di conseguenza:

```
Last_Byte_Sent - Last_Byte_Acked
```

Sancisce il numero di byte che non sono stati ancora riscontrati ma sono stati trasmessi. All'arrivo del segmento da parte del destinatario aggiorna `Last_Byte_Sent` e poi legge il campo `rwnd`, con queste variabili il mittente può garantire che non vi sia overflow dal buffer del destinatario soddisfatto:

$$Last\_Byte\_Sent \leq Last\_Byte\_Acked + rwnd$$

4.7.6 Gestione della Connessione

Mittente e destinatario devono eseguire una fase di handshake per mettersi d'accordo sulla connessione e devono accordarsi su alcuni paramentri della connessione. In generale la vita della connessione è divisa in tre fasi:

- Fase di Handshake
- Fase di Connessione Aperta
- Fase di Chiusura della Connessione

4.7.7 Stabilire la Connessione Three-Way Handshake

La fase è composta da 3 passi:

- **Passo 1:** Il processo client invia un segmento TCP al processo server. Il segmento non contiene dati nel suo campo payload. Il segmento ha:
  - il bit $SYN = 1$
  - il campo sequence number contiene il numero di sequenza x che il client ha scelto per numerare i segmenti
- **Passo 2:** il processo server che ha ricevuto il segmento con $SYN = 1$, inizializza il buffer e alloca le variabili necessarie, infine un segmento di riscontro detto $SYNACK$ viene inviato al client, tale segmento continua ad avere il payload vuoto. Il segmento $SYNACK$ ha:
  - $SYN = 1$
  - $ACKNumber = (x + 1)$
  - $SequenceNumber$ è posto a y dove y è il numero iniziale di sequenza che il processo server usa per numerare i segmenti che tale processo invierà al client
- **Passo 3:** Quando il client ha ricevuto il $SYNACK$ alloca il buffer di trasmissione e inizializza le variabili necessarie per gestire la connessione. Il client invia al server un segmento che notifica al server che ha ricevuto il $SYNACK$, il segmento ha:
  - $SYN = 0$
  - $SequenceNumber = (x + 1)$
  - $ACKNumber = (y + 1)$

Terminate queste fasi può iniziare lo scambio di messaggi tramite la connessione istanziata.

4.7.8 Chiusura della Connessione

La connessione TCP può essere rilasciata indipendetemente da ognuno dei lati poichè è una connessione **Full-Duplex**. Quando il mittente non ha più dati da inviare invia un segmento con bit $FIN = 1$, questo non significa che non ci sia più scambio ma che la connessione non funge da trasporto dati, ma solo per informazioni di controllo. Prima di inviare il segmento con $FIN = 1$ però si attendono tutti i riscontri, a questo punto i dati possono transitare solo dal destinatario al mittente (cioè verso la sorgente) quando questa operazione è conclusa anche il server invia il segmento con $FIN = 1$ e al riscontro entrambi distruggono il descrittore della connessione e quindi la connessione stessa. Il client che ha inviato l'ACK di riscontro al FIN del server aspetta il doppio di $MaxSegmentLifetime$ e poi dealloca. Elenchiamo i 5 passi necessari per chiudere una connessione:

- Il processo client invia al processo server un segmento con bit $FIN = 1$
- Dopo aver ricevuto il segmento il processo server invia un segmento con bit $ACK = 1$ e dealloca buffer e variabili
- Il processo server invia un segmento con $FIN = 1$
- Alla ricezione il client invia un segmento con $ACK = 1$ e attende che si propaghi con un tempo solitamente di 30 secondi
- Alla ricezione di $ACK = 1$ dal client il server chiude la connessione

4.7.9 Gestione della Congestione

La congestione viene definita come una condizione di fortissimo ritardo e perdita di datagrammi causata dal sovraccarico delle interfacce di uno o più router:

- Più interfacce di un router mandano più pacchetti sulla stessa interfaccia in uscita
- Quando il tasso di pacchetti in uscita supera la velocità di assorbimento della interfaccia, il router comincia ad accodare i pacchetti in uscita
- Se il sovraccarico continua la coda cresce
- Quando il router finisce il buffer scarta i pacchetti
- Il router non ha velocità sufficiente a trattare i pacchetti le code di ingresso si allungano

- Se finisce la memoria per la coda in ingresso non legge i pacchetti in entrata

Scenario 1 Poniamo un primo scenario dove abbiamo due host A e B, una connessione che condivide un solo router intermedio. Definiamo che A stia inviando dati ad una frequenza media λ_{in} byte/s. I dati sono inviati senza porre rimedio ad errori, controllo di flusso e gestione. Ignorando le informazioni delle varie intestazioni, il tasso al quale host A invia il traffico al router è λ_{in} byte/s. Host B opera in maniera simile e quindi trasmette a λ_{in} byte/s. Il collegamento ha capacità R e il router ha un buffer che gli consente di bufferizzare i pacchetti entranti quando la loro velocità supera la capacità del collegamento uscente. Finchè il throughput per connessione non supera il valore $\frac{R}{2}$ tutto ciò che trasmette il mittente viene ricevuto dal destinatario. Se però il tasso di invio supera $\frac{R}{2}$ il throughput non supera $\frac{R}{2}$. Questo perchè il collegamento non è in grado di consegnare pacchetti a un tasso superiore a $\frac{R}{2}$, quindi per quanto A e B inviino velocemente i dati non avranno mai un throughput superiore a $\frac{R}{2}$. Quando la velocità di invio tende a $\frac{R}{2}$ il ritardo cresce esponenzialmente, quindi per quanto possa sembrare utile avere una velocità che arriva ad R per sfruttare il collegamento al massimo in realtà si stanno causando problemi, nello specifico lunghi ritardi di accodamento.

Scenario 2 Ora partendo dalla base vista nel precedente scenario imponiamo una grandezza finita al buffer del router. Possiamo subito intuire che se la coda è piena il router scarterà i pacchetti che arrivano e di conseguenza avremo che il mittente prima o poi dovrà ritrasmettere i pacchetti che sono andati perduti. Definiamo un'ulteriore diversificazione dello scenario precedente denominando il tasso di invio come il tasso di trasmissione verso la socket indicandolo con λ_{in} byte/s mentre λ'_{in} byte/s è il **carico offerto**, ovvero il tasso al quale il livello di trasporto invia i segmenti nella rete, indipendentemente se siano trasmessi o ritrasmessi. Da come si effettua la ritrasmissione cambiano le prestazioni, prendiamo in analisi un caso non reale dove l'host A riesce a determinare se il router abbia spazio disponibile nel proprio buffer e di conseguenza trasmette un pacchetto solo quando è libero il buffer. Non avremmo bisogno di ritrasmissione perchè non vi sarebbero perdite di pacchetti poichè avremo $\lambda'_{in} = \lambda_{in}$ con throughput pari a λ_{in}. È un caso ideale poichè tutto ciò che viene inviato viene anche ricevuto e la velocità media dell'host che invia non supera $\frac{R}{2}$. Analizziamo un caso più reale dove la ritrasmissione avviene solo quando si è certi della perdita

del pacchetto, per fare ciò il mittente imposta un timer con un valore sufficientemente grande da essere certo che il pacchetto per cui non si è ricevuto un ACK sia andato perso. Le prestazioni in questo caso sarebbero $\lambda'_{in} = \frac{R}{2}$ mentre λ_{out}, il tasso di ricezione del destinatario cala a $\frac{R}{3}$. Quindi il mittente deve effettuare ritrasmissioni a causa del buffer overflow del router e questo è un altro costo della congestione. Un ulteriore caso di costo della congestione è dovuto alle ritrasmissioni non necessarie, se il timeout è prematuro si può verificare che il pacchetto non è stato perso ma ha semplicemente acquisito ritardo di accodamento, in questo caso però il mittente invierà comunque nuovamente il pacchetto. Il pacchetto ritrasmesso sarà quindi instradato dal router e il destinatario riceverà il pacchetto originale e la copia, scartandone uno, di conseguenza il router ha utilizzato risorse per nulla, inviando una copia non necessaria. In termini pratici avremo il throughput pari a $\frac{R}{4}$ e il carico offerto $\frac{R}{2}$.

Extra Il principio che è stato delineato in questi semplici scenari genera problemi maggiori quando la rete diventa più complessa, in reti più complesse si insinua anche il problema della competizione dello spazio nel buffer dei router perchè magari il router deve servire due o più connessioni contemporaneamente.

4.7.10 Generalità di Controllo della Congestione

Si possono distinguere due approcci per il controllo della congestione:

- **Controllo di Congestione End-To-End:** Il livello di rete non offre alcun supporto al livello di trasporto per il controllo della congestione e quindi bisogna comprendere per il controllo della congestione e quindi bisogna comprendere la situazione dall'osservazione del comportamento della rete. TCP deve utilizzare questo approccio perchè il livello IP non offre feedback sulla congestione.
- **Controllo della Congestione Assistita dalla Rete:** I router forniscono un esplicito feedback sulla situazione della congestione della rete al mittente. Questo può consistere in un avviso con un bit che indica traffico sul collegamento o più sofisticato informando la frequenza trasmissiva che il router può sopportare su un collegamento uscente. Queste due modalità vengono concretizzate dall'invio di un pacchetto **chokepacket** che indica che il router è congestionato oppure il router marca un pacchetto che fluisce dal mittente al destinatario e quindi

poi il destinatario notificherà la congestione al mittente. Tutto ciò necessità di un RTT almeno.

4.7.11 TCP e il Controllo della Congestione

Come abbiamo visto a causa del mancato feedback sulle condizioni di traffico da parte del livello IP, TCP deve implementare un approccio end-to-end. Il mittente allora dovrà imporre un limite alla velocità di invio in base alla condizione di congestione che si troverà ad affrontare. Di conseguenza modulerà la propria velocità di invio aumentandola o diminuendola. Ci troviamo ad affrontare allora tre problemi principali:

1. Limitazione della velocità da parte del mittente TCP
2. Percezione della congestione sul percorso lungo la destinazione
3. Algoritmo da usare per variare la velocità di invio

1. Abbiamo visto come il mittente TCP gestisca buffer e variabili tra cui `LastByteSent` e `rwnd`. Aggiungiamo una nuova variabile chiamandola `cwnd` che specifica la grandezza in byte della finestra di congestione e ci impone un vincolo di velocità di immissione infatti:

   $$LastByteSent - LastByteAcked \leq min\{cwnd, rwnd\}$$

   Cioè la quantità di dati non riscontrati non può eccedere il minimo tra i valori `cwnd` e `rwnd`. Tralasciando `rwnd` abbiamo che il mittente all'inizio di ogni RTT può inviare `cwnd` byte e alla fine dell'RTT il mittente riceve gli ACK relativi di conseguenza possiamo dire che il mittente ha una velocità di $\frac{cwnd}{RTT}$ $\frac{byte}{s}$ e regolando il valore di `cwnd` si può regolare la velocità.

2. Il mittente si trova nella situazione di **Evento Perdita**, ogni qual volta si ha un timeout o una ricezione di tre ACK duplicati, quindi se si verifica una situazione del genere un datagramma è stato scartato a causa di un overflow del buffer del router. Se consideriamo un caso dove non abbiamo smarrimenti notiamo che TCP allargherà la finestra di congestione con rapidità diversa in base alla frequenza di arrivo degli ACK, quindi più velocemente arrivano gli ACK più la finestra verrà allargata rapidamente. Per questo TCP viene definito **auto-temporizzato**, ovvero gestisce autonomamente gli incrementi dell'ampiezza di `cwnd` in base agli ACK.

3. Per comprendere al meglio l'algoritmo di controllo di congestione di TCP ci prendiamo la libertà (tanto so i miei appunti) di aprire una nuova sezione. Qui ci limitiamo a dire che si suddivide in tre fasi:
   - **Slow Start**
   - **Congestion Avoidance**
   - **Fast Recovery**

4.7.12 Algoritmo di Controllo Congestione

Iniziamo col sottolineare che l'algoritmo è implementato solo dal TCP sorgente. Negli anni vi sono state molte modifiche le principali le troviamo nel 1988 con **TCP TAHOE** e successivamente nel 1990 con **TCP RENO**, ci soffermeremo per lo più su quest'ultimo. L'algoritmo ha lo scopo di controllare la finestr `cwnd` tramite due stati principali:

- Congestion Avoidance (CA)
- Slow Start (SS)

Questi due stati sono "obbligatori" nelle implementazioni TCP dalla versione Reno si implementa una terza fase detta **Fast Recovery**.

4.7.13 Congestion Avoidance

Il TCP sorgente in questo stato modifica la dimensione di `cwnd` al verificarsi di due eventi:

1. Ricezione del singolo ACK
2. RIcezione di tre ACK duplicati

Ovviamente la dimensione viene aggiornata in maniera diversa:

1. Per ogni singolo segmento riscontrato (in maniera cumulativa) per la prima volta si ha un così detto **INCREMENTO ADDITIVO**:

   $$cwnd = cwnd + MSS \cdot \frac{MSS}{cwnd}$$

   Questo aumento è lineare.

2. Se si riceve invece l'evento di tre ACK duplicati allora in questo stato si ha un **DECREMENTO MOLTIPLICATIVO**, tenendo sempre presente che non si può scendere sotto il minimo stabilito da MSS:

$$cwnd = max\{\frac{cwnd}{2}, MSS\}$$

Sul punto [1] possiamo osservare che se non vi sono perdite o errori allora ogni ACK cumulativamente riconosce tutti i byte di `cwnd`, infatti il tempo che intercorre tra la trasmissione di tutti i byte nella `cwnd` e la ricezione dell'ACK che li riscontra è di RTT, inoltre il numero di segmenti all'interno di `cwnd` è pari a $\frac{cwnd}{MSS}$, ricordiamo che TCP è un flusso di byte. Osservando questo possiamo intuire che l'incremento additivo è di $1MSS$ ogni RTT, se tutto fila liscio.
L'approccio di Incremento Additivo viene utilizzato come probing ovvero TCP cerca di "capire" fino a che punto la banda è utilizzabile e non appena si verifica una perdita è un errore si attiva il Decremento Moltiplicativo. Ne consegue che il tasso di invio (rate) è approssimabile a

$$rate \approx \frac{cwnd}{RTT} \quad (\frac{bytes}{s})$$

4.7.14 Slow Start

L'algoritmo Slow Start differentemente da come potrebbe far intuire il nome è una fase di TCP dove si ha una crescita esponenziale della velocità di invio, il nome deriva dal fatto che gli albori di TCP la velocità iniziale del sorgente era posto al massimo possibile, nel 1988 questa metodologia è stata sostituita da SS.
L'algoritmo viene impiegato in due situazioni:

1. Quando la connessione inizia
2. Quando si verifica un timeout

In questo stato TCP pone al principio della connessione $cwnd = 1MSS$ e lo incrementa di $1MSS$ per ogni ACK che riceve per la prima volta quindi non più cumulativamente come nello stato CA:

$$cwnd = cwnd + MSS$$

Tale fase viene portata avanti finchè non si incorre in uno di questi tre eventi:

1. `cwnd` arriva alla grandezza pari alla metà di quello che aveva prima di entrare in SS. In questo caso TCP passa nello stato CA.

2. Si ricevono tre ACK duplicati, `cwnd` allora si dimezza e TCP passa nello stato CA. (Se implementa solo gli stati SS e CA, se vi è anche lo stato FR - Fast Recovery la situazione cambia leggermente)

3. Si verifica un timeout e si rinizia da capo lo stato SS.

L'implementazione dell'algoritmo nella sua totalità, cioè raggruppando tutte e tre le fasi, è possibile tramite una variabile denominata `ssthresh` che indca il valore a cui si trova `cwnd` quando SS termina e inizia CA. Tale variabile all'inizio della connessione viene posta ad infinito in maniera da non influire sull'evoluzione dell'algoritmo, quando si verifica un timeout $ssthresh = \frac{cwnd}{2}$ e $cwnd = 1$ in maniera da riniziare lo stato SS. Se poi `cwnd` arriva al, o supera il, valore di `ssthresh` allora si passa in CA.

DISCLAIMER Qui capiamo le basi di SS e CA, successivamente anche di FR, una spiegazione visuale tramite un Automa a Stati Finiti viene data alla fine. QUesto è dovuto al fatto che le due importanti variazioni (Tahoe e Reno) sono simili ma comunque hanno importanti differeze, finora abbiamo parlato dell'algoritmo come se vi fossero solo gli stati SS e CA quindi sottolineaiamo l'importanza dell'automa infondo

4.7.15 Fast Recovery

Quando si ricevono tre ACK duplicati secondo Reno non vi è motivo di ritornare in SS o un CA perchè è un evento puntuale. Di conseguenza viene immediatamente ritrasmesso il segmento con ACK number successivo agli ACK duplicati ricevuti. A questo punto si sfrutta il fatto che TCP memorizza i segmento fuori sequenza:

1. $sshthresh = \frac{cwnd}{2}$

2. $cwnd = ssthresh + 3$ Si contano che almeno tre segmenti successivi al mancante siano stati ricevuti

3. Si continua a trasmettere rispettando `cwnd` e aumentandola linearmente

4. Quando si riceve l'ACK per il pacchetto perduto allora si passa in CA con $cwnd = ssthresh$

4.7.16 Automa a Stati Finiti dell'Algoritmo di Controllo di Congestione

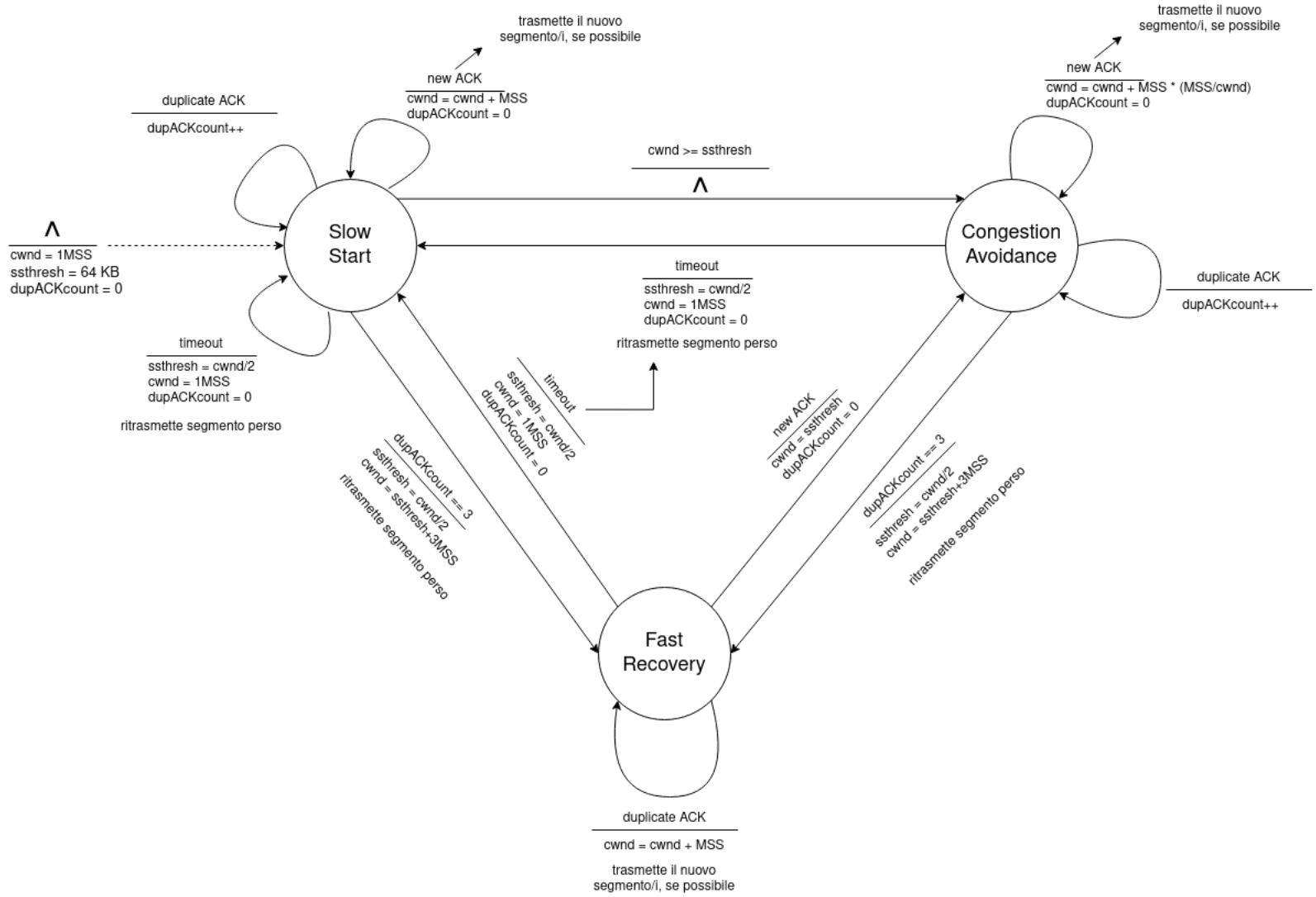


Figura 4.20: Automa a Stati Finiti per l'algoritmo di controllo di congestione

4.7.17 Riassunto

TCP è un protocollo di trasporto orientato alla connessione affidabile che svolge le funzioni:

- Indirizzamento a livello applicativo / Multiplexing / Duplexing → utilizza le porta per indirizzare le diverse applicazioni
- Instarazione, gestione e rilascio connessione → utilizza le socket per identificare la connessione, gestisce la connessione scambiando informazioni necessarie per il canale di comunicazione
- Recupero degli errori → segmenti persi ed errori vengono gestiti tramite timeout di ritrasmissione, RTT e RTO vengono calcolati dinamicamente; recuperati gli errori il TCP può consegnare ordinatamente i segmenti all'applicazione
- Controllo di flusso → controlla il flusso tramite una finestra scorrevole, con dimensione che varia, per controllare il tasso di immissione in rete
- Controllo della congestione → gestisce la dimensione della finestra di congestione che varia dinamicamente

4.7.18 Fairness

Al netto di tutte le ottimizzazioni che abbiamo enunciato possiamo, brutalmente, ridurre l'algoritmo a AIMD (Additive Increase Multiplicative Decrease), ovvero quello che ci produce un grafico a dente di sega relativo alla dimensione della finestra di congestione nel tempo. Prendiamo in considerazione allora k connessioni TCP che hanno sistemi periferici sorgenti differenti così come i destinatari, in comune hanno l'attraversamento di un collegamento di capacità R bps. Ognuna connessione sta trasferendo un file di grosse dimensioni e vi sono solo queste k connessioni sul collegamento in questione. Un meccanismo di controllo di congestione si dice **FAIR** se la velocità trasmissiva media di ogni connessione è $\frac{R}{K}$. Ciò a cui vogliamo arrivare è quindi capire se AIMD è fair. Semplificando usiamo solo due connessioni TCP con gli stessi valori MSS e RTT. Delineiamo un grafico che ci indichi il rapporto tra il throughput delle connessioni:

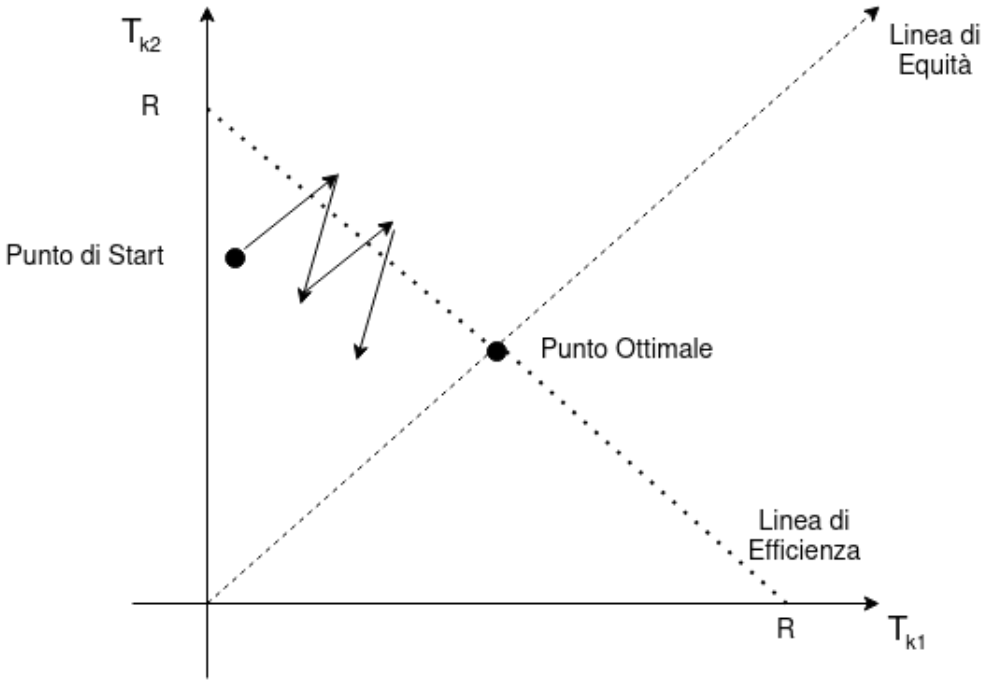


Figura 4.21: Grafico del rapporto tra throughput delle connessioni k_1 e k_2

Analizzando il grafico, che è il peggiore mai fatto ma anche questo è un record, abbiamo tracciato una linea di **Equità**, ci indica dove T_{k_1} e T_{k_2} sarebbero equamente ripartite, la linea di **Efficienza** segna invece il limite superato il quale si riceve un feedback di congestione, il **punto ottimale** è la posizione ideale dove ci sarebbe efficienza massima e massima equità. Senza giri di parole possiamo intuitivamente dire che l'oscillazione dei due T_k mostrati nel gradico, fatti malissimo, ci daranno in media una equità di ripartizione di banda. Per fare questa dissertazione abbiamo fatto molte assunzioni quindi abbiamo anche eliminato molti fattori che rendono la situazione più complicata, bisogna quindi sottolineare che TCP AIMD è **IN MEDIA** fair ma non sempre e di conseguenza anche non sempre efficiente al 100%.

Capitolo 5

Network Layer - Piano Dati

Riassumento malamente il livello di trasporto possiamo dire che è un livello della pila che consente la comunicazione da processo a processo, il livello di rete d'altro canto consente la comunicazione tra host. Si consideri una rete, semplice, dove si ha una comunicazione tra H_1 e H_2, hosts, permessa dai router R_1 , vicino a H_1, e R_2, vicino a H_2. Quindi il livello di rete sorgente incapsula i segmenti del livello di trasporto in datagramma, arrivato a destinazione il livello di rete destinatario avrà il compito di spacchettare il datagramma e inviare i segmenti al livello di trasporto. Questo percorso viene sviluppato in due "fasi" che astraiamo per maggiore chiarezza:

- **Piano dei Dati:** Ha il compito di inoltrare il datagramma dal collegamento (link) entrante al collegamento uscente
- **Piano di Controllo:** Le azioni di inoltro della fase precedente vengono coordinate da quest fase in modo da avere il trasferimento dei datagrammi end-to-end attraverso i percorsi router tra mittente e destinatario

Ricordiamo che i router hanno una pila di livelli "ristretta" ovvero i router non eseguono protocolli a livello di trasporto e di applicazione, quindi non esegue nulla a livelli superiore a quello di rete.

5.1 Separazione Piano Dati - Piano Controllo

Per comprendere meglio i due piani li specifichiamo tramite le loro principali funzioni:

- **INOLTRO (Forwarding):** Quando un router riceve un pacchetto deve immetterlo nel collegmento appropriato, questa funzione non è l'unica del piano dei dati sebbene sia la più importante, in questa funzione vi è anche la possibilità di bloccare il pacchetto o anche di immetterlo su più link in uscita.
- **INSTRADAMENTO (Routing):** Il piano di controllo determina il percorso da H_1 a H_2 a livello globale e non locale del router tramite questa funzione, il metodo utilizzato per tale funzione è l'algoritmo di instradamento (o algoritmo di routing)

Solitamente i termini Inoltro e Instradamento vengono trattati come sinonimi, nello specifico però sono due funzioni ben distinte e le tratteremo di conseguenza:

- Inoltro: Azione locale tramite cui il router trasferisce da una interfaccia in entrata ad una interfaccia in uscita i pacchetti. Azione di pochi nanosecondi, solitamente implementata a livello hardware.
- Instradamento: Implementazione software che indica il percorso globale di rete che determina come i pacchetti partano dal mittente e arrivino al destinatario.

Quindi l'instradamento indica dove il pacchetto debba passare ma è l'inoltre a scegliere poi la strada da intraprendere per arrivare in quel punto. L'inoltro dei pacchetti viene effettuato tramite lo strumento delle **Tabelle di Inoltro**, il router estrae dal pacchetto alcune informazioni che fungono da indice per la tabella e di conseguenza partorire l'interfaccia uscente per il pacchetto stesso.

5.2 Piano di Controllo

Visto che i due piano sono intrinsecamente connessi tra loro poichè fanno parte dello stesso livello, vediamo l'approccio che il piano di controllo ha in maniera da comprendere meglio il piano dati. Infatti il piano di controllo tramite l'algoritmo di instradamento fornisce i valori per le tabelle di inoltro che appunto la funzione di inoltro sfrutta, per fare ciò possiamo sfruttare due approcci: il primo, quello tradizionale, e il secondo detto **Software-Defined** (SDN):

- **APPROCCIO TRADIZIONALE:** L'algoritmo di instradamento è implementato in ogni router, quindi instradamento e inoltro vengono fatte interamente al router, per avere una visione globale le funzioni di instradamento comunicato tra loro per determinare i valori da inserire nelle tabelle, tramite i messaggi del protocollo di instradamento. Se le tabelle fossero configurate da operatori di rete non vi sarebbe bisogno di alcun bisogno di protocolli di instradamento. Il problema è che vi sarebbe necessità di un tempo enorme per configurare le tabelle ogni volta e aumenterebbe in maniera esponenziale gli errori dovuti agli esseri umani.
- **APPROCCIO SDN:** Un approccio alternativo è l'implementazione di un controllo remoto, fisicamente lontano dai router che calcola e comunica i valori a tutti i router. Tale controller potrebbe essere in un data center e gestito da un ISP, la parte importante è che abbia elebata affidabilità e ridondanza, comunica con messaggi contenenti tabelle e altre informazioni di instradamento. Questo approccio del pinao di controllo viene chiamato Software-Defined Networking proprio perchè il controller è un software che calcola e invia i valori necessari al routing.

5.3 Servizi di Rete

Il livello di rete può offrire diversi tipi di servizi, i servizi dipendono dal modello di servizi che si applica, alcuno servizi possono essere:

- **Consegna Garantita:** assicura che il pacchetto giunga prima o poi a destinazione
- **Consegna Garantita a Ritardo Limitato:** Questo servizio garantisce la consegna rispettando un limite di ritardo noto
- **Consegna Ordinata:** Garantisce che l'ordine dei pacchetti che arrivano è corretto
- **Banda Minima Garantita:** Questo servizio a livello di rete emila il comportamento trasmissivo a bit rate specificato anche se il pacchetto può attraversare collegamenti fisici diversi, finchè l'host trasmette a bit rate minore non si verificano perdite

- **Servizi di Sicurezza:** Il livello di rete sorgente può cifrare i datagrammi inviati e sarà compito del destinatario decifrarli

Il modello più noto e utilizzato è il **Best-Effort** ossia "col massimo impegno", non garantisce consegna, ordine o banda minima. Nonostante ciò si è notato che con una adeguata larghezza di banda, come quelle odierne, rimane uno dei migliori modelli.

5.4 Architettura dei Router

Generalizziamo l'architettura dei router per comprendere i componenti essenziali all'interno di queste macchine:

- **Porte di Ingresso:** Svolgono le funzioni di terminazioni fisiche di un collegamento in ingresso. All'interno del router svolgono inoltre una funzione di collegamento in maniera da inoltrare il pacchetto correttamente nella struttura di commutazione affinchè sia indirizzato verso le porte d'uscita corrette. Bisogna sottolineare che le porte di cui stiamo parlando sono interfacce fisiche e non softwre come quelle discusse quando abbiamo parlato di socket.
- **Struttura di Commutazione:** Questa componente è letteralmente una rete all'interno dei router di rete, infatti è una struttura di collegamento fisico tra porte di ingresso e porte di uscita.
- **Porte di Uscita:** Le porte di uscita sono componenti che operano funzioni necessarie al collegamento fisico nei quali bisogna immettere i pacchetti che vengono memorizzate dalle porte in uscita prima di immetterli. Le porte in uscita sono solitamente accoppiate alle porte d'entrata poichè entrambe le tipologie di porta sono bidirezionali, questa condizione sulla scheda di collegamento è detto **line card**.
- **Processore di Instradamento:** La maggiorparte delle funzioni del piano di controllo, il processore in base all'approccio esegue protocolli di instradamento gestendo le tabelle di inoltro dei collegamenti attivi ed elaborando quelle nuove. Nei router SDN, il processore ha il compito di interfacciarsi con il controllo remoto per ricevere i valori delle tabelle e installare le tabelle nelle porte di ingresso.

I componenti implementati a livello hardaware sono le porte di ingresso, le porte in uscita e la struttura di commutazione questo perchè vi sono solitamente N porte su una line card e di conseguenza la pipeline dovrebbe elaborare i datagrammi N volte più velocemente, impensabile per una implementazione software. Questa implementazione hardware diventa invece possibile nel momento in cui i chip che vengono utilizzati sono molto specializzati. Per queste ragioni il piano dati riesce ad operare sulle grandezze dei nanosecondi. Il piano di controllo invece ha ordini di grandezza dei millisecondi ai secondi, infatti il piano di controllo è implementato lato software. Analizziamo ora due modalità di inoltro facendo uso di una analogia stradale. Supponiamo che l'entrata e l'uscita sia uno svincolo costituito da una rotonda e ogni auto prima di entrare abbia bisogno di una piccola elaborazione, detto questo abbiamo:

- **Inoltro basato sulla destinazione:** Se una auto si ferma al casello di entrata indica la sua destinazione e un addetto cerca la destinazione e determina l'uscita dalla rotonda quindi indica al guidatore l'uscita dalla rotonda.
- **Inoltro generalizzato:** È possibili che la rampa d'uscita non venga determinata dalla destinazione. L'addetto può indirizzare la rampa d'uscita in base alla marca dell'auto, lo stato di provenienza, il modello, la targa e in base a uno o più di questi fattori determinano la rampa. D'altra parte una auto che non risulta idonea potrebbe essere bloccato. Quindi un inoltro potrebbe essere deciso su un numero di fattori arbitrario.

5.4.1 Porte di Ingresso e Inoltro per Destinazione

L'elaborazione effettuata dalle porte di ingresso è centrale per la funzionalità dei router. In questo momento infatti vengono utilizzate le tabelle di inoltro per determinare la porta di uscita, infatti ogni porta ha memorizzata una copia della tabella senza dover interrogare il processore di instradamento ogni volta.
Schematizziamo le porte di ingresso:

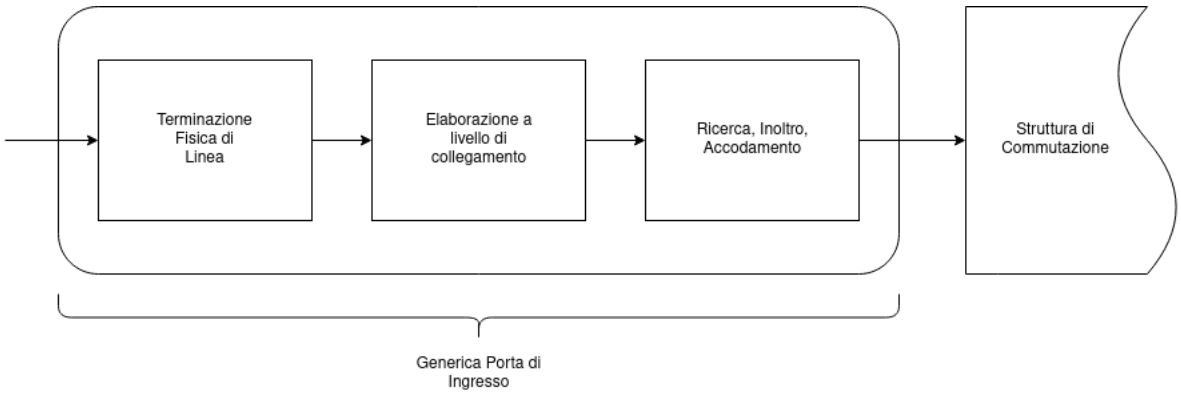


Figura 5.1: Schematizzazione delle funzioni delle porte di ingresso.

Scegliamo ora un caso semplice di inoltro per comprendere il funzionamento, quindi vediamo il caso di inoltro basato sulla destinazione. Permettiamo che gli indirizzi di destinazione sono tutti a 32 bit e che il router ha quattro collegamenti numerati da 0 a 3, l'inoltro viene determinato tramite intervallo degli indirizzi di destinazione. Ad ogni intervallo corrisponde una interfaccia di uscita:

Intervallo degli indirizzi di destinazione	Interfaccia
Da 11001000000101110001000 00000000 A 11001000000101110001011 11111111	0
Da 11001000000101110001100 00000000 A 11001000000101110001100 11111111	1
Da 11001000000101110001101 00000000 A 11001000000101110001111 11111111	2
ALTRIMENTI	3

Figura 5.2: Schematizzazione delle funzioni delle porte di ingresso.

Soluzione più conveniente è quella di costituire una tabella dove al posto di un intervallo di indirizzo si ha un prefisso dell'indirizzo, i

prefissi possono avere anche lunghezze differenti in maniera da disambiguare indirizzi che hanno uguali prefissi, fino a una certa lunghezza, che hanno diverse interfacce d'uscita. Questa regola viene chiamata: **Regola di Corrispondenza a Prefisso Più Lungo**

Corrispondenza di Prefisso	Interfaccia
11001000 00010111 00010	0
11001000 00010111 00011 000	1
11001000 00010111 00011	2
ALTRIMENTI	3

Figura 5.3: Tabella di inoltro basata sui prefissi

Quindi la ricerca hardware viene effettuata in nanosecondi, di conseguenza necessita tecniche che superino i limiti di una ricerca lineare su una grande tabella. Non tratteremo gli algoritmi che eseguono queste operazioni. Facciamo una piccolissima digressione sulle memorie usate può essere fatta citando memorie DRAM integrate e SRAM veloci per eseguire accessi alla memoria in maniera efficiente. TCAM poi può essere una delle memorie migliori poichè consente di restituire il contenuto della tupla nella tabella di inoltro in un tempo approssimativamente costante. Una volta compresa la porta di uscita allora il pacchetto viene passato alla struttura di commutazione, se però è impegnata il pacchetto verrà bloccato e accodato nella porta di ingresso. Vi sono poi altre azioni che fanno le porte di entrate:

- Elaborazione a livello fisico e di collegamento
- Il numero di versione del pacchetto, cheksum e altri campi vengono controllati e a volte riscritti

- I contatori usati per la gestione di rete vengono aggiornati

5.4.2 Struttura di Commutazione

Il core di un router è questo componente tramite cui i pacchetti vengono commutati dalla porta di ingresso alla porta di uscita. La commutazione può essere eseguita in vari modi:

- **Commutazione in Memoria:** Precedentemente i router erano semplici calcolatori, di conseguenza la commutazione veniva effettuata dando il controllo alla CPU. Le porte erano dei semplici dispositi i I/O, la porta di ingresso segnalava l'arrivo di un pacchetto con interrupt e veniva copiato dentro la memoria del processore, che provvedeva all'estrazione dell'intestazione e dell'indirizzo di destinazione. In questo caso abbiamo delle limitazioni per quanto riguarda la memoria, un limite di spazio, e un limite riguardante le operazioni poichè il bus di sistema permette una operazione di scrittura/lettura per volta. Tuttavia la commutazione in memoria viene effettuata tuttora con alcune migliorie: la ricerca e la memorizzazione viene eseguita sulle line card di ingresso.
- **Commutazione tramite Bus:** Le porte di ingresso trasferiscono un pacchetto direttamente alle porte di uscita tramite un bus condiviso senza far entrare il processore in azione. Si aggiunge una etichetta al pacchetto che indica la porta di uscita. Il pacchetto viene quindi ricevuto da tutte le porte di output ma solo quella corrispondente all'etichetta lo trasmetterà sul collegamento fisico. Le limitazioni di questo approccio sono ovviamente dovute a situazioni in cui più pacchetti arrivano contemporaneamente al router, in questo caso uno solo verrà servito mentre gli altri attenderanno, il bus può trasferire un pacchetto per volta. Se la banda del bus è occupata da un pacchetto è interamente utilizzata.
- **Commutazione attraverso Rete di Interconnessione:** Una architettura che supera alcune limitazioni è l'architettura di rete di interconnessioni utilizzata in passato dalle arhitetture multiprocessore. Quello che ci si presenta è una matrice di commutazione di $2n$ bus che collegano n porte di ingresso a n porte di uscita. Ogni bus verticale interseca ogni bus orizzontale, il controller della struttura di commutazione ci consente di aprire o chiudere ogni

incrocio. Se un pacchetto arriva alla porta A e deve giungere alla porta Y allora si chiuderà l'incrocio AY e solo la porta Y riceverà il pacchetto. Se allo stesso tempo un altro pacchetto giunge ad un'altra porta di ingresso e ha la porta di uscira diversa da Y allora può essere consegnato contemporaneamente. Ovviamentese avesse la medesima porta di uscita invece dovrebbe attendere che il precedente abbia fatto il suo corso. Questa commutazione viene definita **Non-Blocking**. Si amplia la capacità di commutazione aumentando le strutture di commutazione e facendole operare in parallelo, in questo modo il pacchetto può essere spezzettato dalla porta di ingresso e riassemblato dalla porta di uscita.

5.4.3 Porte di Uscita

L'elaborazione delle porte di uscita consiste nel recupero dei pacchetti dalla memoria e trasmetterli sui link di uscita, quindi selezionare i pacchetti dalla coda, apportare le elaborazioni del livello di collegamento e del livello fisico.

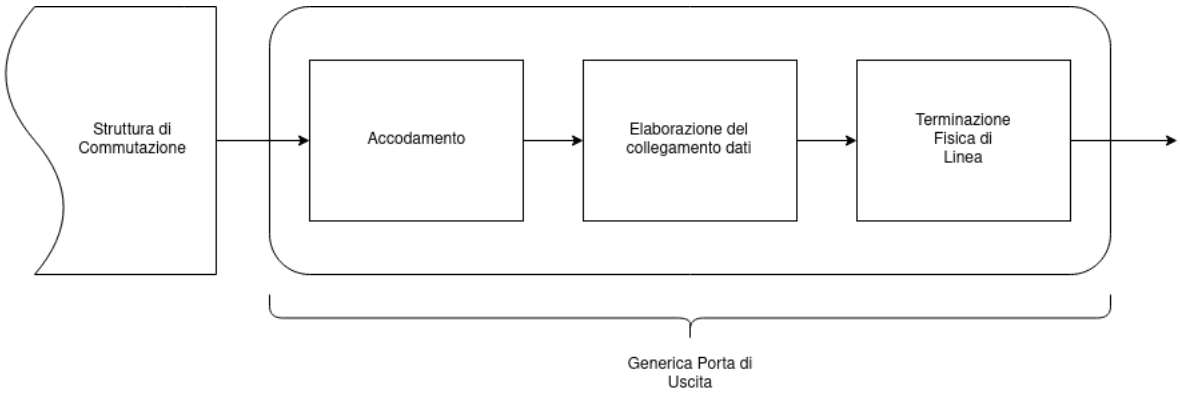


Figura 5.4: Schematizzazione della generica porta di uscita

5.5 Accodamenti

Come abbiamo visto parlando di porte di ingresso e porte di uscita, le code si creano proprio in questi frangenti. Dipendentemente dalla quantità di traffico di rete, dalla velocità della struttura di commutazione e dalla linea. Di conseguenza le code che si vengono a creare

di per sè non sono un problema, costituiscono un problema al crescere poichè la memoria andrà ad esaurirsi e di conseguenza si verifica una peridita di pacchetti. Definiamo con R_{line} (pacchetti al secondo) il tasso di trasmissione della velocità della linea di ingresso e di uscita, inoltre abbiamo N porte di ingresso e N porte di uscita. Semplifichiamo l'esempio dando la stessa lunghezza e che arrivino alle porte di ingresso nello stesso tempo che occorre per ritrasmettere un pacchetto sullo stesso collegamento. R_{switch} sarà invece il tasso al quale i pacchetti vengono inoltrati dalla porta di ingresso a quella di uscita. Una situazione dove non avremo problemi di accodamento nelle porte di ingresso si ha quando R_{switch} è N volte più veloce di R_{line}, l'accodamento sarebbe trascurabile poichè se ci trovassimo nella situazione peggiore cioè che arrivino pacchetti a tutte le N porte di ingresso e hanno tutti le necessità di uscire dalla singola porta di uscita non vi sarebbero provlemi poichè ogni gruppo di N pacchetti può essere elaborato dalla struttura di commutazione prima dell'arrivo del successivo. Abbiamo quindi due casistiche in cui abbiamo problemi:

- **Accodamento nelle porte di ingresso:** Quando la struttura di commutazione ha una velocità inferiore a quello della porta di ingresso.
- **Accodamento nelle porte di uscita:** La struttura di commutazione ha un rate superiore alla port di uscita. Troppi pacchetto in arrivo sulla stessa porta di uscita.

5.5.1 Accodamenti in Ingresso

Supponiamo alcuni punti:

- Velocità di collegamenti uguali
- Un pacchetto viene trasferito dalla porta di ingresso alla porta di uscita nello stesso intervallo di tempo che si necessita a far arrivare un pacchetto alla linea di ingresso
- Il trasferimento avviene secondo algoritmo FCFS

Se due pacchetti in due diverse code di ingresso sono destinati alla medesima coda di uscita allora uno dei due pacchetti sarà bloccato e dovrà attendere poichè la struttura di commutazione puà trasferire un

solo pacchetto a una certa porta di uscita. Oltre a questo provlema possiamo avere un'altra causa dell'accodamento, il fenomeno **Blocco in Testa alla Coda (HOL)**. Ovvero se si verifica che un pacchetto arriva in una coda di ingresso dove vi è un altro pacchetto che sta attendendo il suo turno per quanto detto prima, allora anche il secondo arrivato dovrà attendere anche se la propria coda di destinazione è libera. Questi problemi portano alla saturazione dei buffer delle code di ingressi e di conseguenza causare perdita di pacchetti.

5.5.2 Accodamenti in Uscita

Se R_{switch} ha comunque un rate N volte superiore a R_{line} e che i pacchetti che arrivano a N porte di ingresso ma sono diretti alla medesima porta di uscita (e non abbiamo le premesse poste all'inizio della sezione) si potrebbero verificare un crescere della coda in uscita poichè la porta di uscita può inviare un solo pacchetto per volta. Da qui avremo code lunghe e buffer saturi, se capitasse una situazione del genere si portebbe adottare una politica **Drop-Tail** ovvero scartarlo oppure decidere se scartare più di uno dalla coda, qualche volta è utile utilizzare questa politica per ad esempio poter inviare un pacchetto di allerta di congestione. Le politiche di eleminazione sono riassunte sotto gli algoritmi **AQM** (active queue managment). Il maggiormente usato è **RED (random early detection)**. Quando una coda di usita si riempie bisogna scegliere l'ordine di trasmissione e di conseguenza ci sarà bisogno di uno **Scedulatore Di Pacchetti**.
Le raccomandazioni odierne guidano a una sufficientemente corretta grandezza dei buffer:

$B = \frac{RTT \cdot C}{\sqrt{N}}$
con C = Capacità del collegamento

5.5.3 Scheduling di Pacchetti

Arrivati a questo punto bisogna avere degli algoritmi di scelta dei pacchetti che sono in coda da trasmettere, quindi capire quali politiche adottare per dare priorità ai pacchetti in coda.
Vediamo i più importanti algoritmi di scheduling:

1. **FCFS:** Una coda gestita da algoritmo FCFS (First Come First Served), come dice il nome, imposta la priorità dei pacchetti in

base al loro arrivo. Il primo arrivato è i primo ad essere servito a tale algoritmo deve essere affiancato da una memoria di buffer per la coda molto importante, la coda viene definita FIFO (First In First Out).

2. **Code con Priorità:** Le code con priorità adottano una strategia di classificazione pacchetti, ad esempio si possono classificare in base alla loro natura dividendo i pacchetti in due code: per i pacchetti dedicati alla gestione della rete e per gli utenti, dando alla prima una priorità alta e alla seconda una priorità più bassa. La strategia ha due modalità di gestione che sono:

   - Preemptive
   - Non-Preemptive

   Ovvero nella prima si ha che i pacchetti della coda con priorità bassa vengono trasmessi fintanto che non vi sono pacchetti nella coda con priorità alta, appena un pacchetto viene immesso nella coda con priorità alta allora si smette di inviare i pacchetti nella coda a bassa priorità e viene iniziata la trasmissione dei pacchetti con priorità alta. Ricordiamo che per fare questo bisogna introdurre alcuni controlli poichè i pacchetti sono entità discrete e non si può interrompere la trasmissione se il pacchetto non è stato totalmente inviato. Nella seconda semplicemente non si può fermare l'invio e quindi bisogna svuotare la coda prima di poter trasmettere i pacchetti dell'altra.

3. **Round Robin e Accodamento Equo-Ponderato:** Lo scheduling Round Robin divide i pacchetti in code di priorità ma la priorità non è così rigido come la precedente strategia. La forma semplice di round robin prevede l'alternanza della trasmissione quindi avendo classe 1 e 2 prima si invia il pacchetto della classe 1 e poi della classe 2 per poi tornare alla classe 1 e così via. La modalità work-conserving round robin non lascia mai il collegamento inattivo, fintanto che abbiamo pacchetti di qualsiasi classe, quindi se la classe ha la coda vuota si passa alla successiva. Una strategia molto utilizzata è WFQ - Weighted Fair Queuing (Accodamento Ponderato Equo), anche qui si ha una ciclicità, se abbiamo tre categorie si servirà la 1, poi la 2 e infine la 3 per poi ricominciare. È inoltre conservativa perchè se la coda si svuota

si passa alla successiva immediatamente. La peculiarità di WFQ è che le varie classi si possono servire in maniera differenziata. A ciascuna classe i si assegna un peso w_i, WFQ garantisce che i pacchetti ricevono una frazione di servizio $\frac{w_i}{\sum w_j}$ dove il denominatore è la somma effettuata su tutte le classi che hanno pacchetti da trasmettere, quindi garantisce una frazione di banda. Su un collegamento con capacità R la classe i avrà un rendimento pari a $R\frac{w_i}{\sum w_j}$.

5.6 Internet Protocol - IPv4, IPv6, Indirizzamento

Il protocollo del livello di rete è IP (Internet Protocol), questo protocollo ha due versioni principali: IPv4 e IPv6. La versione 4 è la più utilizzata mentre la versione 6 è proposta come sostituzione della v4 per problemi che poi vedremo più avanti. Trattando questo protocollo vedremo anche l'indirizzamento, ovvero la funzione di instradamendo del protocollo.

5.6.1 Formato dei Datagrammi IPv4

Abbiamo già accennato che il pacchetto del livello di rete è denominato **DATAGRAMMA**, l'analisi del formato di tali datagramma può sembrare superfluo ma bisogna conoscerlo affondo per comprendere come funziona internet.
I campi principali sono:

- **VERSIONE:** Sono 4 bit che specificano la versione del protocollo, infatti il formato del datagramma cambia in funzione della versione, per far comprendere al router come interpretare il pacchetto.
- **LUNGHEZZA INTESTAZIONE:** Sono 4 bit che indicano la lunghezza dell'intestazione, infatti l'intestazione può avere differenti opzioni, sebbene solitamente la lunghezza è sempre 20 byte. Nel caso in cui si utilizzino opzioni solitamente tralasciate questo campo indica l'inizio dei dati nel pacchetto e la fine dell'intestazione.

- **TIPO DI SERVIZIO (TOS):** Questi bit consentono di distinguere diversi tipi di datagrammi. Il router talvolta necessita di distinguere tra il traffico come i pacchetti della telefonia dai pacchetti FTP. Per distinguere vengono utilizzati anche i requisiti di QoS (Quality of Service).
- **LUNGHEZZA DATAGRAMMA:** Rappresenta in byte la lunghezza totale del datagramma (intestazione + dati). Il campo è di 16 bit e quindi un datagramma ha lunghezza massima di 65535 byte.
- **IDENTIFICAZIONE:** 16 bit utilizzati per identificare in maniera univoca i singoli frammenti in cui viene spezzettato un pacchetto IP.
- **FLAG:** Sono 3 bit utilizzati per il controllo della frammentazione dei datagrammi.Il primo bit $RESERVED$ è sempre settato a 0, il secondo DF (Don't Fragment) se settato a 1 indica che il pacchetto non deve essere spezzettato, il terzo MF (More Fragments) se settato a 0 indica che il pacchetto è l'ultimo frammento o il solo frammento del pacchetto originario, di conseguenza tutti gli altri frammenti hanno questo bit a 1.
- **OFFSET:** Sono 13 bit che identificano lo spiazzamento dei dati nel caso essi siano un frammento di un datagramma frammentato, l'offset è espresso in multipli di 8 byte e fa riferimento alla posizione dei dati nel pacchetto originario.
- **TEMPO DI VITA (TTL):** Il Time-To-Live è un campo che serve a non far rimanere in giro per la rete i datagrammi. Ogni elaborazione di un router decrementa di 1 questo campo fino ad arrivare a zero, momento in cui viene scartato dal router.
- **PROTOCOLLO:** Questo campo occorre quando il pacchetto arriva a destinazione, infatti specifica il protocollo di trasporto a cui passare il pacchetto. TCP utilizza il valore 6 e UDP il valore 17. Questo campo è l'anello di collegamento tra livello di rete e livello di trasporto.
- **CHECKSUM DI INTESTAZIONE:** È il campo che consente di verificare se ci sono errori nell'intestazione del datagram-

ma, ogni router lo calcola e controlla che sia identico al campo. Se si identifica un errore il datagramma viene scartato. Ovviamente ogni elaborazione del router aggiorna alcuni campi e quindi bisogna ricalcolare anche il checksum. Il checksum di TCP/UDP è diverso da questo campo e viene calcolato poichè IP non necessariamente trasporta dati di un protocollo di trasporto TCP/UDP.

- **INDIRIZZI IP - SORGENTE E DESTINAZIONE:** Alla creazione di un datagramma il sorgente inserisce il proprio IP nel datagramma e attraverso il DNS recupera l'IP di destinazione e lo inserisce nello specifico campo.
- **OPZIONI:** Estendono i campi dell'intestazione e possono avere lunghezza variabile, per questo motivo non si può determinare dove il campo dei dati inzierà. I datagrammi possono richiedere l'elaborazione di queste opzioni e questo può far variare significativamente il tempo di elaborazione dei router.
- **DATI:** Nella stragrande maggioranza dei casi, il campo detiene pacchetti TCP/UDP ma non necessariamente può essere costituito da messaggi ICMP.

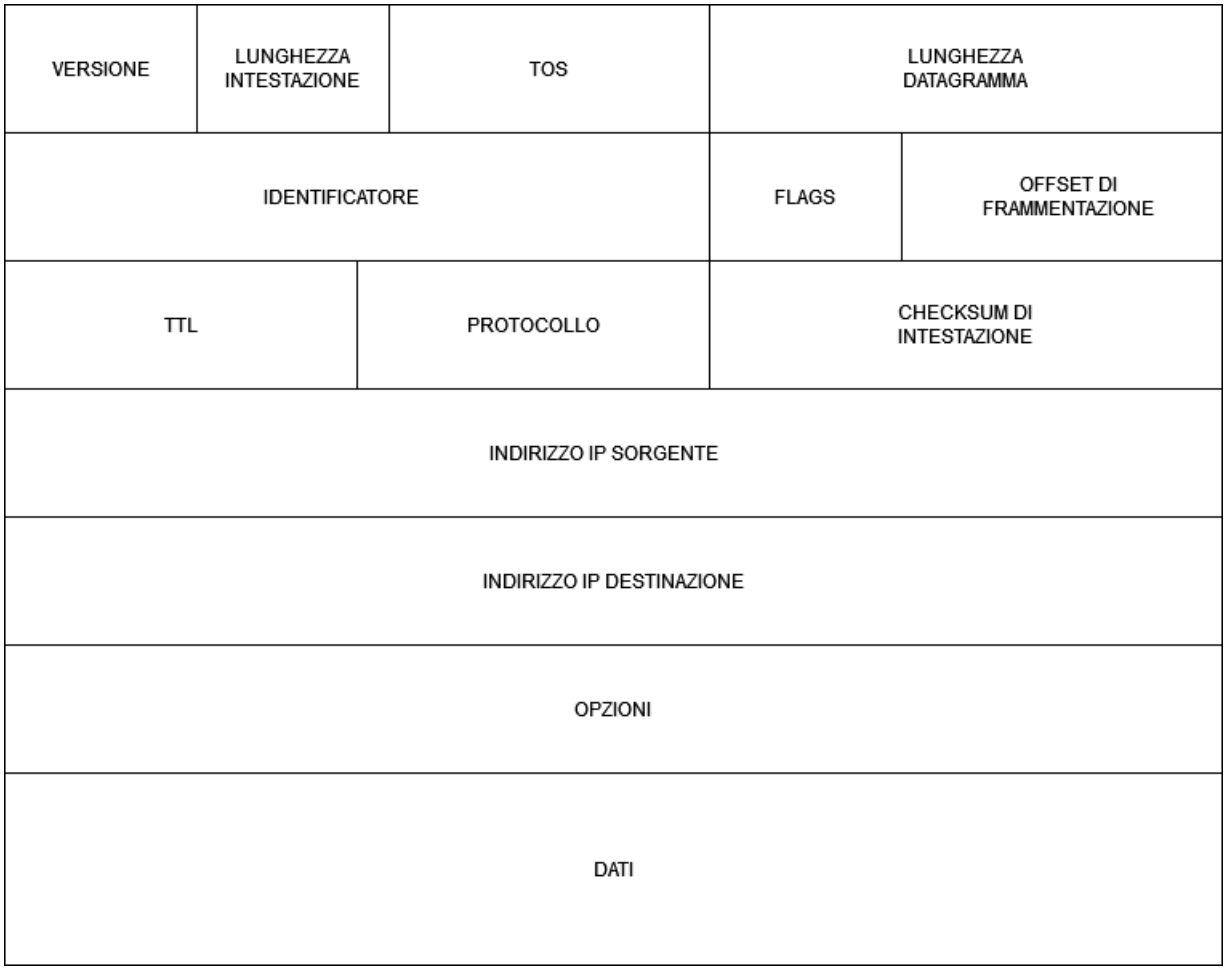


Figura 5.5: Formato Datagramma IP

5.6.2 Frammentazione dei Datagrammi IPv4

MTU (maximum transmission unit) o unità massima di trasmissione è la quantità massima di dati che un frame a livello di collegamento può trasportare. Come sappiamo l'incapsulamento prevede che i datagrammi IP vengano posti all'interno di un frame a cui però viene posto un limite di grandezza non modificabile, MTU appunto. Il fatto di avere una quantità massima di dati trasmissibili non crea un problema, ma vi sono complicazioni derivate dal fatto che non tutti i collegamenti sono uguali o di conseguenza MTU varia in base al collegamento che

stiamo utilizzando. Quindi possiamo immaginare una situazione dove abbiamo utilizzato collegamenti che hanno MTU abbastanza grande da poter contenere il nostro datagramma ma ad un certo punto, lungo il nostro percorso, troviamo un collegamento con MTU inferiore alla grandezza del nostro datagramma. La soluzione a questo problema, nella versione IPv4, è la frammentazione.
Il router che si accorge della differenza di dimensioni, divide il datagramma in frammenti più piccoli e li trasmette lungo il collegamento d'uscita stabilito. Tali frammenti devono essere ricostituiti a formare il datagramma originario prima di trasmettere il payload al livello di trasporto poichè TCP/UDP si aspetto un segmento integro. Vi è inoltre una difficoltà per quanto riguarda le prestazioni, infatti una volta frammentato il datagramma non verrà più ricostituito dal livello di rete dei router ma esclusivamente da quello dell'host di destinazione, in tale modo non vi sono cali di prestazione nell'elaborazione dei router.
Ora occorre capire come avviene questa frammentazione!
Il router preleva la porzione dati del datagramma IP e lo spezza in più porzioni, in modo che ciascuna (con l'aggiunta dell'header) stia in un frame, bisogna inoltre tenere di conto che ogni frammento, tranne l'ultimo, ha dimensione multipla di 8 byte, perchè in questo modo è definito il campo offset dell'header IP. L'ultimo frammento in generale è il più corto e verrà identificato come ultimo tramite il campo More Fragment (MF) che sarà settato a zero. Tutti gli altri frammenti avranno questo campo impostato a 1. Il protocollo IP usa sempre tre campi dell'header per controllare il meccanismo di frammentazione e permettere il riassemblaggio. Come abbiamo già citato questi campi sono: **Identificatore**, **Offset di Frammentazione**, **More Fragment (MF)**. L'header del datagramma originale verrà copiato interamente nei frammenti, con opportune modifiche al campo **Opzioni**, e verrà cambiato il campo Fragment Offset che indica il punto dell'area dati del datagrammi originale dove comincia la porzione dei dati trasportata dal frammento. Come abbiamo già sottolineato questo è un multiplo di 8 byte. Se l'offset è di 185, il frammento porta la porzione di dati che inizia nella posizione $185 \cdot 8 = 1480$ byte. Bisogna specificare a cosa serve il campo Identificatore, tutti i frammenti sono caratterizzati dall'avere dello stesso identificatore del datagramma originale, viene assegnato univocamente dal trasmettitore, che ha un contatore dei datagrammi IP trasmessi, e la coppia (IP provenienza, Identificatore) che

rende univocamente identificabile un certo datagramma IP e tutti i suoi frammenti.
Vediamo ora come avviene invece il riassemblaggio di questi frammenti!
Eseguita la frammentazione ogni frammento viaggia in maniera autonoma fino a destinazione. Solo alla fine del viaggio avrà luogo il riassemblaggio per ricostruire il datagramma originale. Il ricevitore riconosce di avere ricevuto un frammento e non un datagramma intero in due modi:

- il pacchetto IP ha un offset uguale a zero, ma ha il flag MF settato ad 1 - questo significa che è il primo frammento
- il pacchetto Ip ricevuto ha un offset diverso da zero - significa che è un frammento successivo al primo e se MF è zero significa che è l'ultimo frammento

Il protocollo IP del ricevente identifica univocamente i frammenti di uno stesso datagramma mediante la coppia (IP trasmettitore, Identificatore). Il ricevente non conosce la dimensione del datagramma originale perchè ogni frammento ha il campo Lunghezza Totale settato per identificare la lunghezza del frammento stesso e non quella del datagramma originale. Nel momento in cui riceverà il frammento con il campo MF a zero si potrà capire la dimensione totale del datagramma sommando all'offset dell'ultimo frammento la lunghezza dei dati trasportati dall'ultimo frammento. Nel momento in cui un frammento viene perso diventa impossibile ricostruire il datagramma originale. Per evitare sprechi all'arrivo del primo frammento (inteso cronologicamente, il frammento non deve essere per forza il primo nel senso di inizio del datagramma originale) il ricevente inizializza un timer, se il timer scade prima che tutti i frammenti siano giunti a destinazione allora scarta tutti i frammenti ricevuti fino a quel momento.

5.6.3 Indirizzamento IPv4

Iniziamo con lo specificare che solitamente un host ha un solo collegamento verso la rete e lo utilizza per inviare (o ricevere) datagrammi. Il confine tra host e collegamento viene detto interfaccia (che banalmente è una scheda di rete). Dato che il router ha il compto di inoltro, da un collegamento in entrata ad uno in uscita, deve per forza di cose avere

più collegamenti a cui corrispondono più interfacce, data la possibilità di inviare e ricevere datagrammi tramite ogni interfaccia (e collegamento), ogni interfaccia detiene un indirizzo IP che sono quindi associati all'interfaccia e non all'host o al router.

Composizione Indirizzo IPv4

Ogni indirizzo IP (versione 4) è formato da 32 bit (4 byte) per un totale di 2^32 indirizzi, circa 4 miliardi. Per una migliore lettura viene espresso in **Dotted-Decimal Notation**, ognuno dei 4 byte viene convertito e separato dal successivo tramite un punto, quindi ognuno dei 4 decimali ha un range da 0 a 255, per esempio:

193.32.216.9

Ovviamente dalle macchine viene visto in sistema binario. Ogni indirizzo IP è univoco a livello globale e di conseguenza non può essere scelto in modo casuale. Una parte dell'indirizzo di un'interfacciaè determinata dalla sottorete cui è collegata. La sottorete (o subnet) è una parte della suddivisione di una singola rete IP, la suddivisione è ovviamente visibile solo nella parte logica della rete. Il processo di **Subnetting** è la divisione di una singola rete in gruppi di macchine che hanno in comune in ciascun indirizzo IP un determinato **prefisso**.
Facciamo una schematizzazione d'esempio:

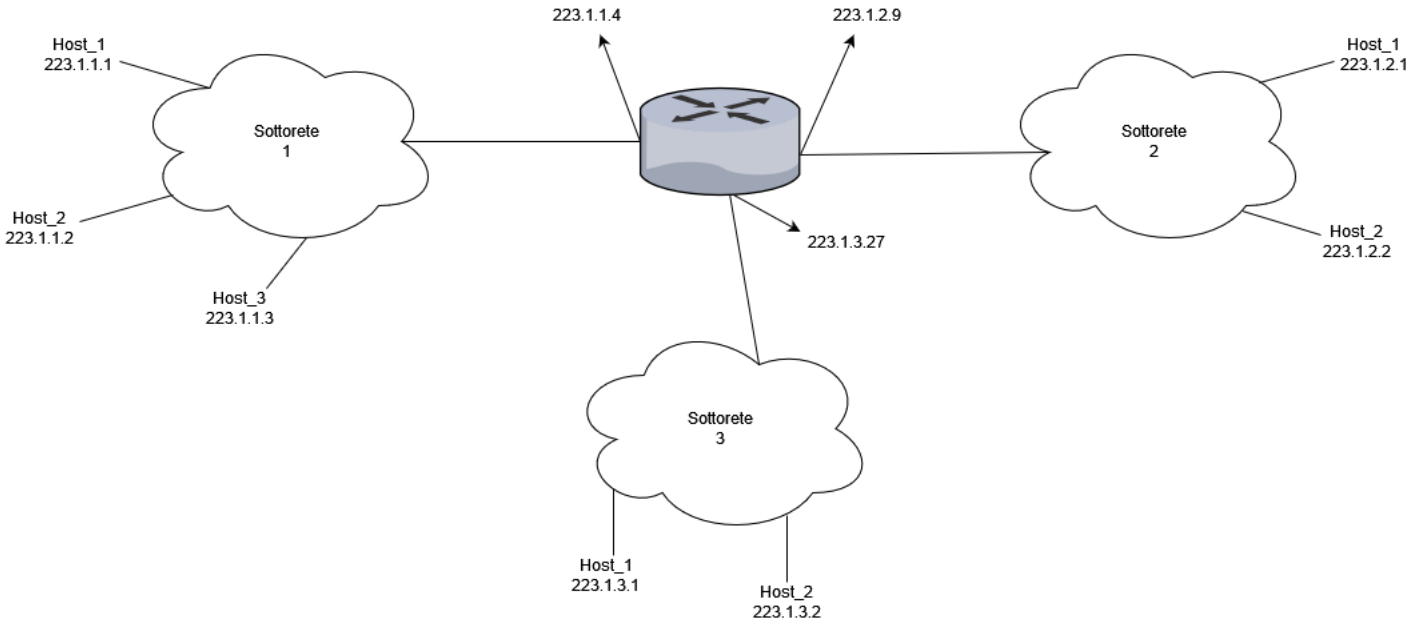


Figura 5.6: Schematizzazione di sottoreti

Ora che abbiamo compreso le reti e il subnetting possiamo scrivere il nostro indirizzo Ipv4 con una nuova notazione:

193.32.216.9/24

/24 sta ad indicare la maschera di sottorete (o subnet mask) questo significa che un host connesso a questa sottorete avrà sicuramente come prefisso i primi 24 bit, quindi in maniera generica:

193.32.216.xxx/24

La maschera viene espressa anche in dotted-decimal notation ponendo a 1 i bit che vengono identificati come prefisso, di conseguenza, continuando l'esempio la subnet mask sarà:

255.255.255.0

che in binario sarebbe:

11111111.11111111.11111111.00000000

In soldoni diciamo che la maschera di rete serve in generale a "bloccare" gli slot del prefisso, ogni bit a 1 è un bit bloccato mentre a 0 possiamo considerarli come variabili, sono quelli occupabili.
Avendo similitudini molto grandi bisogna sottolineare che quanto appena detto non si deve confondere con l'Indirizzo IP Broadcast che è sempre 255.255.255.255, quando un host invia un datagramma con indirizzo di destinazione questo appena citato, viene ricevuto da tutti gli host nella stessa sottorete del sorgente.
La strategia di assegnazione degli indirizzi IP fino al 1993 è detto **Classful Addressing**, classi di indirizzi IP, ovvero una formalità per dividere lo spazio di indirizzamento in v4. Secondo questo sistema di indirizzamento basato sulla classe prevede che dai primi bit di un indirizzo IP si possa determinare la classe e di conseguenza la maschera di sottorete. Le classi erano A, B e C con indirizzi di sottorete da 8, 16, 24 bit, quindi una sottorete di classe C (/24) poteva ospitare $2^8 - 2 = 254$ host, non 256 perchè 2 indirizzi sono riservati ad usi specifici. Il problema sorse quando 254 host erano pochi per le organizzazioni ma eseguendo un cambio di classe e salendo alla classe B (/16) erano sovradimensionate, poichè si avevano 65534 host disponibili che per aziende da 2000 host erano troppi, avendo di conseguenza uno spreco di indirizzi che ricordiamo essere univoci. Per questi motivi nel 1993 si è passati alla strategia **Classless Interdomain Routing (CIDR)**. Questo nuovo schema consente una migliore gestione degli indirizzi e migliora le prestazioni di instradamento tramite una più efficiente organizzazione delle tabelle di inoltro. Tale sistema permette infatti di definire il prefisso e la parte host dell'indirizzo in maniera "continua" senza ricorrere alle classi, la notazione è:

$$a.b.c.d/x$$

dove x stabilisce il numero di bit che compongono il prefisso, di conseguenza $y = 32 - x$ bit sono i bit per gli hosts pari a $(2^y - 2)$. Quindi x poteva ora assumere valori anche al di fuori dei soli 8, 16 e 24 diventando, quando serve, /21, /11, ecc. In questo modo si sono limititati finora gli sprechi di indirizzi IPv4 in un sempre più crescente internet.

Approfondimento su Indirizzamento Basato su Classi Come abbiamo già detto l'indirizzamento è una funzionalità essenziale di qua-

lunque sistema di comunicazione, tanto più per un sistema che vuole essere universale, cioè in grado di connettere due qualunque calcolatori al mondo. In TCP/IP gli indirizzi sono numeri binari, compatti, facili da trasmettere e a partire dai quali il software di rete può calcolare una route o instradamento.
Se chi legge ha già dei rudimenti di Reti di Calcolatori, potrebbe pensare che si possono utilizzare indirizzi fisici, di reti fisiche (anche detti Indirizzi MAC, li vedremo più avanti), questi indirizzi però sono di tipo diverso e alcune reti fisiche potrebbero non avere indirizzi universali.
Quindi non possono esistere due macchine con due indirizzi IP uguali. Una volta garantita l'univocità degli indirizzi, non ci sono problemi se una macchina detiene più di un indirizzo IP ciò che conta è che entrambi siano univoci. Il router basa l'instradamento sulla sola conoscenza della sottorete di destinazione, allora si necessita di una struttura dell'indirizzo.

Un indirizzo IP è una coppia (netid, hostid), dove:

- netid = identifica la rete fisica a cui appartiene la macchina = indirizzo di rete
- hostid = identifica la macchina sulla rete identificata da netid = indirizzo di host relativo alla rete netid

Questo dovrebbe farci comprendere l'importanza delle maschere di rete spiegate prima!
Si nota immediatamente quale sia la sottorete di destinazione e l'instradamento può essere efficiente riducento la quantità di informazione da considerare e rende più semplice e veloce il processo di decisione. Il netid deve essere univoco, infatti ad ogni netid corrisponde un'unica rete fisica, e ogni rete fisica può avere più di un netid basta che ognuno di essi sia univoco.
Non abbiamo ancora visto l'aspetto dell'instradamento però prendiamo come assioma che la decisione sulla strada da seguire per raggiungere la destinazione è basata solo sulla conoscenza della rete di destinazione e non sulla conoscenza di tutti i singoli host che ci sono nel mezzo. Abbiamo l'esigenza quindi di ridurre l'informazione da esaminare per l'instradamento, l'indirizzo dell'host quindi consiste anche dell'indirizzo fisico della rete in cui risiede. Bisogna precisare che l'indirizzo dell'host è in realtà l'indirizzo della scheda di rete che esso usa per connettersi

all'internet, ma la conoscenza dell'indirizzo della rete fisica aiuta non solo l'instradamento per il router ma anche per gli host nella "consegna diretta", la vedremo più avanti quando parleremo del protocollo ARP.
A questo punto bisogna comprendere come si suddivide il suffisso dal prefisso, come penso sia ormai chiaro un prefisso molto grande consente di indirizzare molte reti ma ognunca con pochi host perchè di conseguenza il suffisso è piccolo, d'altro canto con un suffisso grande si hanno reti con molti host ma ci possono pochi di queste reti. La soluzione in passato è stato come abbiamo già detto l'indirizzamento basato su classi.
Ricordiamo che questo metodo non è più utilizzato nell'identificare gli indirizzi moderni.
Lo schema di auto-identificazione comprendeva i primi 3 bit che identificavano una specifica classe: avevamo la **classe A** con la possibilità di avere **126 reti** e **più di 16 milioni di host**, la **classe B** con la possibilità di avere **più di 16 mila reti** e **più di 65 mila host**, la **classe C** con la possibilità di avere **più di 2 milioni di reti** e **254 di host**. Oltre a queste tre classi che erano le principali utilizzate si avevano le classi D ed E rispettivamente per il multicast e riservata per usi futuri, per queste due ultime classi si utilizzavano 4 bit per l'identificazione.

Indirizzi di Rete e di Broadcast Diretto Ora che abbiamo compreso come e perchè venivano identificate le reti e i loro host precisiamo che un hostid pari a zero non è un indirizzo di host legare come indirizzo di destinazione, poichè sta ad indicare l'intera sottorete per convenzione. Analogamente per convenzione si può esprimere un indirizzo di broadcast diretto cioè che si riferisce a tutte le macchine della sottorete, tale indirizzo è identificato da un hostid con tutti i bit posti a 1. Sottolineiamo che il protocollo IP non supporta il broadcast a livello di netid, ovvero non si ha un indirizzo per spedire a tutte le reti un determinato messaggio, si può fare solo per le sottoreti conoscendo a priori il netid.

Broadcast Limitato, lo Zero indica "Questo" e Vantaggi dell'Indirizzamento IP Un indirizzo che si compone di tutti i 32 bit posti a 1 è un indirizzo speciale che significa broadcast sulla sottorete in cui appare. Prende il nome di **broadcast, broadcast limitato o indirizzo broadcast della rete locale**. Questo indirizzo di broadcast non può uscire dalla rete su cui viene inviato, una macchina può inviare un pac-

chetto indirizzato al broadcast anche senza conoscere l'indirizzo della rete su cui si trova. Può essere usato quando non si conosce nemmeno il proprio indirizzo, vedremo un caso a breve parlando del protocollo DHCP.
Un indirizzo IP che si compone di 32 bit posti a 1 ha un suo opposto: un indirizzo IP con 32 bit settati a 0. Essendo l'opposto se l'indirizzo IP broadcast significa "tutti", quest'altro indirizzo IP indica "questo", più precisamente questa sottorete e questo specifico host. Questa convenzione dipende dal contesto. Se una macchina non conosce il proprio netid, potrebbe inviare pacchetti in cui il proprio hostid è valorizzato, mentre il netid è tutto a zero.
I vantaggi dell'indirizzamento IP sono molteplici:

- Permette di avere una integrazione di reti di grosse, medie e piccole dimensioni
- Lunghezza fissa rende più semplice l'elaborazione: i primi bit specificano le dimensioni di prefisso e suffisso, e si rende efficiente l'estrapolazione di netid e hostid
- Netid facilita la consegna diretta, lo capiremo quando parleremo di consegna diretta e bisognerà aspettare la digressione sul protocollo ARP
- Netid facilita l'amministrazione degli indirizzi: la centralizzazione si basa solo sui netid perchè gli hostid sono assegnati e gestiti dall'amministratore locale

  N.B. : Netid e Maschera di Rete sono sinonimi, il primo termine viene utilizzato durante tutto il corso e quindi ho voluto introdurlo per non creare ambiguità durante lo studio.

Indirizzo di Loopback Un indirizzo molto speciali è rappresentato dall'indirizzo di Loopback: `127.0.0.1`, in realtà gli indirizzi di loopback possono essere anche uno qualunque di `127.x.x.x` su specifici sistemi operativi, su alcuni os invece sono tutti ammessi ad eccezione di `127.0.0.0`. La destinazione che indica questo indirizzo IP è la macchina stessa, quindi il pacchetto non viene nemmeno spedito sulla rete. Viene usato principalmente per testing degli strati superiori di protocollo e come IPC, poichè le applicazioni si parlano come se fossero su macchine differenti.

Due macchine stesso IP

Blocchi di Indirizzi IP

Se si vuole ottenere un blocco di indirizzi IP bisogna rivolgersi al proprio ISP che attinge ad un blocco più grande e ne riserva una porzione al richiedente. Per esempio supponiamo che un provider abbia allocato un blocco di indirizzi che corrisponde a:

> 192.168.16.0/20
> Si traduce in:
> 11000000.10101000.00010000.00000000
> Quindi dato /20 sappiamo che la maschera è:
> 255.255.240.0

Questo provider decide di dividerlo in 8 blocchi:

> Blocco ISP 192.168.16.0/20 → **11000000.10101000.0001**0000.00000000
> Organizzazione 1 192.168.16.0/23 → **11000000.10101000.0001000**0.00000000
> Organizzazione 1 192.168.18.0/23 → **11000000.10101000.0001001**0.00000000
> FINO A
> Organizzazione 7 192.168.30.0/23 → **11000000.10101000.0001111**0.00000000

L'ISP non è l'unico ente che però distribuisce e alloca indirizzi IP, esiste l'organizzazione noProfit ICANN (Internet Corporation for Assigned Names and Numbers) di cui mi riservo di indirizzarvi verso una informazione completa su tale organizzazione per le controverse situazioni dove è stata coinvolta, un solito punto di partenza è `icannwatch.org` Oltre ad allocare indirizzi IP gestisce root server DNS e amministra i registri regionali degli indirizzi che si occupano dell'allocazione all'interno delle regioni rispettive.

DHCP - Server DHCP e Protocollo DHCP

Gli indirizzi IP relativi alle interfacce dei router vengono configurati manualmente, anche perchè devono essere statitici, ovvero non cambiare nel tempo.

Anche gli host possono essere configurati manualmente ma non è la prima opzione e nemmeno la migliore, per fare questo si utilizza il **DHCP - Dynamic Host Configuration Protocol**. Il DHCP consente ad un host di ottenere l'indirizzo IP in maniera automatica, inoltre informa l'host appena entrato nella sottorete di informazioni quali:

1. L'indirizzo del gateway (router, o programma nel router, per uscire dalla sottorete)
2. DNS server locale
3. La maschera di sottorete

L'amministratore può configurare il DHCP in maniera tale che un indirizzo IP venga riservato ad una macchina specifica in maniera tale che l'host in questione abbia sempre lo stesso indirizzo ogni volta che entra nella sottorete. Nella stragrande maggioranza dei casi il DHCP assegna un indirizzo temporaneo che cambierà ogni volta che l'host entra nella sottorete. Il protocollo DHCP viene detto **Plug-And-Play** o anche **Zero-Conf** per la peculiarità di automatizzare la connessione tra host e rete, se non ci fosse questo protocollo l'amministratore di rete dovrebbe configurare manualmente gli indirizzi IP di ogni host che entra nella sottorete, ogni volta che si connette.
Questo protocollo è largamente utilizzato nelle reti residenziali e nelle LAN wireless, dove gli host entrano ed escono dalla rete in maniera molto frequente. Il protocollo è un protocollo client-server, il client è un host che appena entrato nella sottorete vuole ottenere informazioni sulla configurazione della rete. Nel caso più semplice il router svolge anche la funzione di server DHCP, può succedere, in situazioni più complesse, che il server DHCP sia connesso alla sottorete ma che sia una macchina fisicamente separata dal router.

Il protocollo DHCP si applica in quattro passi:

1. **Individuazione del Server DHCP:** L'host che si collega alla sottorete vuole identificare il DHCP per ottenere un indirizzo IP, questa operazione viene definita tramite un messaggio **DHCP DISCOVER**, il pacchetto è trasportato tramite UDP sulla porta 67. Il datagramma IP che contiene il messaggio DHCP con l'indirizzo IP di destinazione pari a `255.255.255.255` ovvero viene

inviato in **broadcast** a tutti i nodi della sottorete, dato che l'host non ha un vero IP sorgente pone il proprio indirizzo a `0.0.0.0` che è una specifica di "questo host".

2. **Offerta del Server DHCP:** Il server DHCP invia in broadcast un messaggio **DHCP OFFER**. Viene chiamato "offerta" perchè si possono avere differenti server DHCP che rivecono il discover e propongono una "offerta" al client. Tale DHCP OFFER contiene l'ID della transazione, l'indirizzo IP proposto al client, la maschera di sottorete e il LEASE TIME dell'indirizzo, ovver la durata della connessione con quel determinato indirizzo. Questo tempo è dell'ordine delle ore o dei giorni.

3. **Richiesta DHCP:** Il messaggio **DHCP REQUEST** verrà inviato dal client al server di cui ha scelto l'offerta, contenendo i parametro di configurazione.

4. **Conferma DHCP:** Il server risponde al client confermando i parametri richiesti tramite il messaggio **DHCP ACK**.

Vi sono comunque delle tecniche che consentono ai client di rinnovare il lease time, una volta scaduto, sempre tramite DHCP.

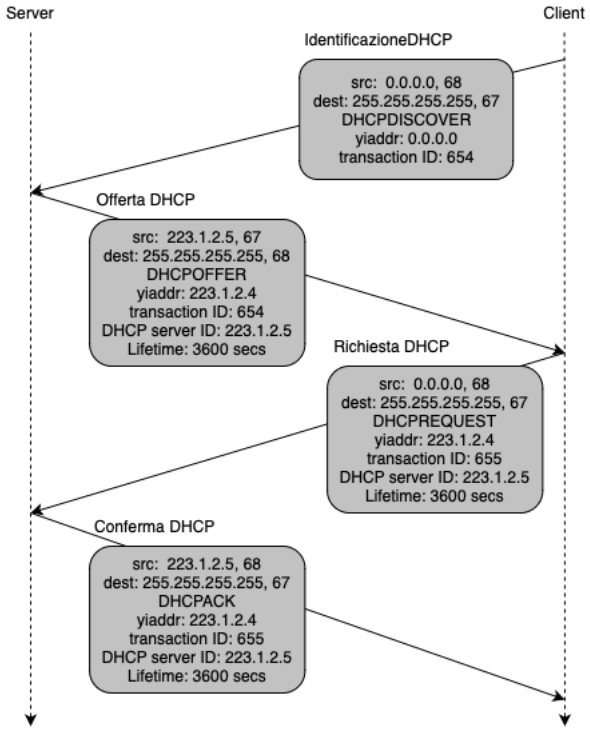


Figura 5.7: Schematizzazione del formato dei messaggi e dello scambio tra server e client

NAT - Network Address Translation

Il NAT nasce come soluzione al crescere di internet, come abbiamo visto ogni macchina deve avere un indirizzo IP per essere conosciuto in rete e poter eseguire connessioni. Il problema nasce dal fatto che gli indirizzi IP sono un insieme finito poichè espressi su 32 bit, di conseguenza più macchine sono presenti nella rete meno indirizzi restano disponibili, fino a saturazione. Una soluzione alla saturazione è proprio il NAT che si comporta come un **unico** dispositivo con un **unico** indirizzo IP sebbene sia di fatto un router e non un host. Il NAT si avvale del precedentemente spiegato DHCP per dare ad ogni host nella propria sottorete un indirizzo IP che non è "pubblico" e che serve solo all'interno della propria sottorete. Di fatto il NAT comunica con l'esterno della sottorete, per comunque riuscire a far comunicare gli host con l'esterno si avvale di uno strumento denominato **TABELLA DI**

TRADUZIONE NAT.
Vediamo un esempio per comprendere come il NAT svolge la propria funzione.
L'host `10.0.0.1` vuole otterene tramite la propria porta `3345` una connessione con l'host `128.119.40.186` su porta `80`. La richieta passa attraverso il router, il nostro NAT, che salva nella tabella di traduzione il proprio indirizzo, `138.76.29.7`, con una porta arbitraria, ad esempio `5001`, e l'indirizzo dell'host sorgente con la porta `3345`. Il datagramma di `10.0.0.1` viene modificato e l'indirizzo sorgente e la porta vengono poste identiche a quelle del NAT, ovvero `138.76.29.7:5001`. L'host a cui è rivolta la richiesta allora risponderà al NAT sulla porta `5001`, a questo punto il NAT tramite la tabella conosce l'indirizzo dell'host e la porta a cui la porta `5001` è associata e indirizzerà i dati ricevuti da `129.119.40.186` all'host che li ha richiesti ovvero `10.0.0.1:3345`.
Questa operazione viene effettuata per tutto il traffico della sottorete, gli indirizzi IP all'interno della sottorete sono definiti in base all'utilizzo. Una buona lettura di approfondimento è: `https://it.wikipedia.org/wiki/Indirizzo_IP_privato`.
Oltre a reti private è possibile, sebbene sconsigliato, porre server pubblici "dietro" un NAT, bisogna configurare opportunamente il NAT e in questi casi è possibile che un unico indirizzo IP per il NAT non basti più.

5.6.4 Internet Protocol Versione 6 - IPv6

Nei primi anni '90, l'Internet Engineering Task Force diede l'inizio allo sviluppo del successore di IPv4. Questo perchè l'indirizzamento a `32 bit` iniziava ad essere piccolo davanti alla crescente richiesta di blocchi IP. Allora IETF ha colto l'occasione per rivedere l'intero protocollo e riadattarlo per renderlo più flessibile ma anche per implementare nuove funzionalità.

Formato Datagrammi IPv6 Di conseguenza alla nuova implementazione il formato del datagramma ha subito delle mutazioni:

- **VERSIONE:** `4 bit` che identificano la versione del protocollo di rete

- **CLASSE DI TRAFFICO:** `8 bit` utilizzati per determinare la priorità del pacchetto, si possono quindi categorizzare pacchetti di specifiche applicazioni piuttosto che quelli di altri servizi
- **ETICHETTA DI FLUSSO:** `20 bit` per identificare un flusso di datagrammi
- **LUNGHEZZA DEL PAYLOAD:** `16 bit` (intero senza segno) indicano il numero di byte che seguono i `40 byte` di intestazione
- **INTESTAZIONE SUCCESSIVA:** Identifica il protocollo a cui verranno consegnati i dati del datagrammi
- **LIMITE DI HOP:** Il contenuto viene decrementato di 1 fino al raggiungimento dello 0, momento in cui viene eliminato il datagramma
- **INDIRIZZI SORGENTE/DESTINAZIONE:** Gli indirizzi IPv6 a `128 bit`
- **DATI:** Payload che viene consegnato al protocollo specificato nel campo di intestazione successiva

Più e meno - IPv6 VS IPv4

Vediamo prima ciò che IPv6 ha introdotto/modificato:

- **INDIRIZZAMENTO ESTESO:** L'indirizzo passa da `32 bit` a `128 bit`, in questa maniera lo spazio degli indirizzi diventa praticamente inesauribile. Inoltre IPv6 implementa **ANYCAST** (oltre a broadcast e unicast), ovvero è possibile consegnare un datagramma a uno specifico gruppo di host.
- **INTESTAZIONE OTTIMIZZATA A `40 Byte`:** IPv6 implementa una intestazione fissa a `40 byte`, avere questa dimensione fissa consente una elaborazione più rapida dei datagrammi IP.
- **ETICHETTA DEI FLUSSI:** IPv6 si differenzia in maniera sostaziale per una definizione non propriamente esaustiva di FLUSSO (FLOW). L'RFC relativo (2460) ci dice che tale etichettatura consente di gestire in maniera specifica alcuni flussi di pacchetti che magari esula dalla gestione di default. Sebbene questo flusso

non venga esplicitato in maniera chiara si vede come sia necessario differenziare il traffico in tipologie diverse tra loro ma con una specificità di gestione.

Vediamo ora cosa è stato eliminato e perchè:

- **FRAMMENTAZIONE/RIASSEMBLAGGIO:** Frammentazione e riassemblaggio sono operazioni che richiedono molto tempo per essere eseguite, soprattutto dai router. Di conseguenza IPv6 gestisce queste operazioni sui sistemi periferici in maniera esclusiva, in questo modo le operazioni riescono ad essere molto più rapide. In un caso in cui i datagrammi sono troppo grandi per un collegamento viene inviato al mittente un messaggio di errore ICMP, e il mittente invia datagrammi di grandezza inferiore.
- **CHECKSUM:** Questa operazione è stata eliminata dato che è sembrato molto ridondante poichè i protocolli di trasporto integrano già questa funzione. Inoltre era una operazione molto onerosa dato che in IPv4 ogni router, data l'elaborazione che apportava, aveva la necessitò di ricalcolarla.
- **OPZIONI:** Viene eliminata e di conseguenza si riesce ad avere una intestazione di dimensione fissa a `40 byte`.

Transiziona da IPv4 a IPv6

Abbiamo compreso perchè c'è la necessità di passate da IPv4 a IPv6, quello che dobbiamo capire ora è come eseguire questa transizione, è impensabile che si abbia una giornata in cui tutto internet si spegne e si aggiornano tutte le macchine al fine di per implementare la nuova versione. Di conseguenza si è pensato ad un approccio più tranquillo e migliore denominato **TUNNELING** (da non confondere con tunnelling in ambito VPN).
Prima di spiegare come funziona tale approccio precisiamo che i sistemi IPv6 sono retrocompatibili ma non è possibile il viceversa.
Questo implica che si possono, in maniera lenta, aggiornare i sistemi alla nuova versione.
Supponiamo di avere due nodi IPv6 che sono interconnessi tra loro da diversi nodi IPv4, questi nodi IPv4 sono il nostro tunnel. Il primo nodo IPv6 prima di un nodo IPv4 incapsula il proprio datagramma versione 6 in un datagramma versione 4 e lo spedisce al nodo successivo.

Quando il primo nodo versione 6 estrarrà il datagramma e avremo il nostro datagramma versione 6 come se non fosse mai stato trasportato da nodi versione 4. Di conseguenza avremo la capacità di aggiornare un po' per volta senza accusare ritardi o blocchi all'intera internet.

Inoltro Generalizzato e SDN

Abbiamo precedentemente visto come viene eseguito un inoltro all'interno di un router, man mano che venivano introdotte nuove tecnologie per aumentare l'efficienza dell'instradamento si è giunti al paradigma **MATCH-ACTION** intrinsecamente connesso con l'approccio **SDN**. Precedentemente, queando abbiamo parlato di inoltro generalizzato ci siamo concentrati maggiormente sull'inoltro basato sulla destinazione, ora possiamo espandere il concetto spiegando il paradigma match-action e le tabelle che ne derivano. Il lato "MATCH" può essere eseguito su più campi dell'intestazione e non solo sulla destinazione, e il lato "ACTION" può produrre significati maggiori: inoltro su più porte, servizio di load-balancing, riscrittura di valori come nei NAT, bloccare o eliminare come nei Firewall, e tanto altro.
Parlando di SDN ci accorgiamo di come definizioni come router o switch perdano di significato e di conseguenza introduciamo un più generico packet switch, poichè alla fine bisogna necessariamente dialogare con il controller remoto. Come abbiamo detto parlando di SDN il controller si occupa di creare e distribuire le tabelle di inoltro, se parliamo del paradigma match-action le tabelle vengono denominate tabelle di flusso e lo standard per la creazione è **OPENFLOW**, ora vedremo cosa contiene ogni occorrenza di una generica tabella di flusso:

1. **INSIEME DI VALORI DEI CAMPI DELL'INTESTAZIONI**, tali valori sono i confronti con i quali il pacchetto deve essere confrontato. Un confronto rapido è dettato dall'hardware e dalle memorie TCAM con milioni occorrenze.

2. **INSIEME DI CONTATORI**, vengono aggiornati quando un pacchetto viene associato a una occorrenza nella tabella, sono importanti valori perchè alcuni riguardano ad esempio informazioni temporali sugli ultimi aggiornamenti.

3. **AZIONI**, quando un pacchetto ha associata una occorrenza si determina un insieme di azioni necessarie da intraprendere per il pacchetto.

Match

La sezione che corrisponde al lato "match" corrisponde all'insieme dei campi dell'intestazione. Questo insieme è detto anche RULES. I campi sono in totale 12, il primo campo è la porta di ingresso che fondamentalmente non serve al lato match. Abbiamo poi un insieme di campi dedicati al livello di collegamento: MAC source, MAC destination, Eth Type, VLAN ID, VLAN priority. Un insieme di campi al livello di rete: IP source, IP destination, IP TOS, IP protocol. Infine abbiamo il livello di trasporto con TCP/UDP source port, TCP/UDP destination port.
Se un pacchetto ha più match allora si risponderà al confronto con l'occorrenza che ha la priorità più elevata.

Action

Elenchiamo le azioni principali dopo il match:

- **INOLTRO (Forwarding):** Un pacchetto può essere inoltrato ovunque tranne che nella porta dove è entrato, può essere inoltrato anche in multicast. L'inoltro può essere verso il controller che applica una elaborazione per cui può essere inoltrato in base alle regole aggiornate.
- **SCARTO (Dropping):** Occorrenza della tabella dei flussi senza azioni, indica che il pacchetto deve essere scartato.
- **MODIFICA (Modify-Field):** Prima che si verifichi l'inoltro bisogna modificare alcuni campi, tutti possono essere modificati tranne il protocollo IP.

Capitolo 6

Network Layer - Piano di Controllo

Finora ci siamo soffermati a capire come funzionassero le varie tabelle di inoltro e di flusso, questo perchè sono il punto di congiunzione tra il piano di dati e quella di controllo. Ora quello che ci poniamo di comprendere è come queste tabelle vengono calcolate, mantenute e installate.
Ricordiamo che abbiamo due approcci:

- **Controllo Locale:** Dove inoltro e instradamento vengono eseguiti interamente al router, ognuno ha poi un componente di instradamento che comunica con gli altri router per comprendere come riformulare la prossima tabella. Un approccio di questo tipo era il più blasonato fino a qualche decennio fa, protocollo OSPF e BGP sono basati su questo approccio.
- **Controllo Logicamente Centralizzato o Controllo Remoto:** Un controller remoto, quindi al di fuori dei router, calcola e distribuisce le tabelle che devono essere utilizzate da ogni router. Il controller interagisce con un agente di controllo in ogni router tramite un protocollo che gestisce le tabelle, l'agente di controllo ha funzionalità minime solitamente esegue gli ordini del controller remoto, inoltre non interagisce direttamente con gli altri agenti di controllo. Questo è il punto chiave che ci consente di separare un controllo locale e un controllo remoto.

Questi due approcci possono funzionare grazie agli algoritmi di instradamento che ora iniziano a vedere.

6.1 Algoritmi di Instradamento - Categorizzazione

Lo scopo di questi algoritmi è quello di determinare i percorsi tra le sorgenti e i destinatari attraverso la fitta rete di router. Il percorso migliore potrebbe essere certamente quello più breve, ma vedremo che i fattori in gioco sono molteplici come i costi di un tratto di un percorso o le policy scandite dai router. Quindi gli algoritmi forniscono un punto centrale del networking risolvendo un problema tutt'altro che banale. Per affrontare questo studio introduciamo i grafi in maniera del tutto superficiale poichè non è scopo del corso, se si vuole approfondire rimandiamo ad un corso di Algoritmi e Strutture Dati.
Quindi un grafo, che descriviamo come $G = (N, E)$, una coppia formata da un insieme di nodi N e un insieme di archi E, ogni arco congiunge due dei nodi nell'insieme N. Nel nostro caso assumiamo che i router sono i nodi e che gli archi interpretano i collegamenti fisici tra i router. Ad ogni arco inoltre si associa un costo, di "percorrenza", che nel nostro caso può assumere significato di lunghezza del collegamento, velocità del collegamento, o altri significati ancora.
Per maggiore omogeneità della trattazione non daremo un significato specifico, l'importante è ricordare che in base alla tipologia di percorso che vogliamo avere come risultato cambia il costo, per intenderci se vogliamo calcolare un percorso migliore in termini di distanza allora il costo è determinato dalla vicinanza di due router, ricordando che bisogna poi avere lo stesso significato per tutti i costi degli archi. Gli archi vengono denotati dalla coppia di nodi che congiungono, quindi l'arco che congiunge il nodo x a y è (x, y) e $c(x, y)$ è il costo dell'arco, se (x, y) non appartiene a E allora $c(x, y) = +\infty$. Il grafo che consideriamo, è **non orientato** questo si concretizza nel fatto che l'arco (x, y) è uguale a (y, x) e $c(x, y) = c(y, x)$, inoltre se (x, y) è in E allora x e y si dicono **vicini**.
Denotiamo che un **percorso** in un grafo è una sequenza di nodi tali che ogni coppia sia un arco appartenente a E e il costo del percorso è la somma dei costi degli archi.
Dati due generici nodi del grafo esistono più di un singolo percorso

per congiungerli, uno di questi percorsi (o anche più) è sicuramente il **percorso a costo minimo**. Il percorso a costo minimo è anche il percorso più breve se ogni arco ha il medesimo costo.
Questo ci presenta la domanda a cui rispondiamo gli algoritmi di instradamento: individuazione dei percorsi meno costosi tra sorgente e destinazione.
Gli algoritmi di instradamento si classificano per diversi fattori:

- **CENTRALIZZATO:** calcola il percorso avendo una conoscenza globale e completa della rete, l'algoritmo ha una visione di tutti i collegamenti e dei relativi costi. Queste informazioni deve poterle avere prima di iniziare il calcolo e quindi può essere implementato su un controller remoto oppure copiato su ogni router. Tali algoritmi vengono chiamati: algoritmi *LINK STATE*.
- **DECENTRALIZZATO:** calcola il percorso in maniera distribuita e iterativa. Praticamente ogni nodo conosce solo i costi degli archi vicini, attraverso lo scambio di informazioni con i nodi vicini che faranno il medesimo lavoro con i propri vicini si creerà una stima del costo del (o dei) percorso verso una (o più) destinazione. Tali algoritmi vengono chiamati: algoritmi *DISTANCE VECTOR*.
- **STATICI:** Algoritmi applicabili su percorsi che cambiano raramente e sono di fatto statici
- **DINAMICI:** Algoritmi che determinano gli instradamenti al variare del volume di traffico o della topologia della rete. Sono più affidabili per i cambiamenti della rete ma allo stesso tempo possono creare problemi con instradamenti *loop* e oscillazione dei percorsi.
- **SENSIBILE AL CARICO:** Algoritmi che sono sensibili al variare del costo degli archi, costo basato sulla congestione.
- **INSENSIBILE AL CARICO:** L'opposto del sensibile al carico. **DUH!**

6.2 Link State Routing - LS

Ricordiamo che un algoritmo di questo tipo deve conoscere la topologia della rete e i costi di tutti i collegamenti, questa conoscenza è l'input dell'algoritmo. Questo input si ottiene tramite un invio iterativo, distribuito e in broadcast verso i nodi della rete, questo viene ottenuto tramite link-state broadcast. Tutti i nodi quindi dispongono di una vista identica e completa della rete, da questo punto di può lanciare l'algoritmo LS su ogni nodo per avere gli stessi risultati. L'algoritmo che andiamo ad utilizzare è uno degli algoritmi più importanti nella teoria dei grafi: **algoritmo di Dijkstra**. Quindi cerchiamo di capire come funziona l'algoritmo, specifichiamo che questo corso non è focalizzato sugli algoritmi o sulle strutture dati, quindi vedremo l'algoritmo in una specifica situazione e non approfondiremo oltre, sproniamo il lettore ad andare a informarsi meglio su questo algoritmo partendo anche da un algoritmo alla base di Dijkstra: algoritmo di Prim.
Per comprendere al meglio l'algoritmo facciamo un esempio della sua esecuzione, premettendo alcune notazioni:

- $n \rightarrow$ nodo generico appartenente al grafo
- $C(n) \rightarrow$ costo complessivo da origine al nodo n preso in considerazione in un dato istante dell'iterazione (somma dei costi degli archi da origine a n)
- $p(n) \rightarrow$ nodo predecessore di n scelto in base al cammino minimo che porta ad n
- $v \rightarrow$ insieme di triplette dal formato $(n, c(n), p(n))$ rappresentanti i nodi visitati che non hanno più alcun arco da esplorare
- $s \rightarrow$ priority queue dove risiedono triplette formate da $(n, c(n), p(n))$ rappresentanti i nodi scoperti ma da esplorare. La priorità dell'elemento è tanto più alta quanto è minore $C(n)$

Si noti che al principio, a parte il nostro nodo di partenza che ha $C(nododipartenza) = 0$, si pone $C(n)$ per ogni nodo del grafo pari a $+\infty$.

Ora vediamo un esempio sul funzionamento dell'algoritmo Dijkstra.

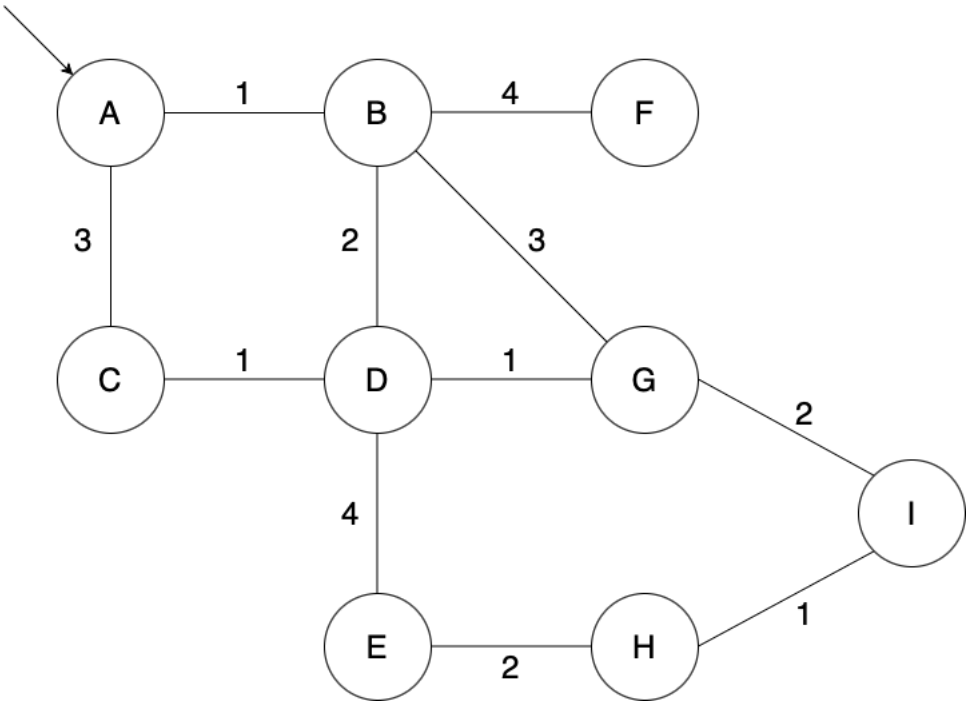


Figura 6.1: Grafo di esempio per esempio sull'algoritmo Dijkstra

Prendiamo in considerazione il grafo in figura ed eseguiamo l'algoritmo Dijkstra che ha come nodo di partenza **A** e come nodo di destinazione (o goal) il nodo **I**.
Inizio dell'esecuzione suddivisa in passi:

1. `V={(A, ∅ , None)}`
   `S={∅}`
   Esploriamo gli archi di `A` e scopriamo il nodo `B` e il nodo `C`, aggiorniamo di conseguenza `S` →
   `S={(B, 1, A),`
   `(C,3,A)}`

2. Esploriamo il nodo **B** e trviamo i nodi **F, G, D, B** di conseguenza non ha alrti archi, **A** non verrà preso in considerazione dato che è presente in **V**.

A questo punto possiamo aggiornare sia **V** e **S** →

```
V={(A, ∅ , None),
(B, 1, A)}

S={(C, 3, A),
(D, 3, B),
(G, 4, B),
(F, 5, B)}
```

3. Esploriamo quindi il nuovo primo elemento della priority queue che è il nodo `C`, il nodo `A` è presente in `V` quindi l'unico nodo che ci rimane è `D` che non è presente `V` ma solo in **S**, esplorando notiamo che **C(D)** passando per **C** è pari a `4`. Dati che questo costo è maggiore del costo passando per **B** non viene eseguito nessun aggiornamento dal nodo **D** in **S** ma **V** e **S** sono comunque aggiornati come segue →

```
V={(A, ∅ , None),
(B, 1, A)
(C, 3, A)}

S={(D, 3, B),
(G, 4, B),
(F, 5, B)}
```

4. Esploriamo nuovamente il primo elemento di **S**.
   Il nodo **D** ha solo **G** e **E** nodi da esplorare e aggiorniamo nuovamente **V** e **S** →

```
V={(A, ∅ , None),
(B, 1, A)
(C, 3, A),
(D, 3, B)}

S={(G, 4, B),
(G, 4, D),
(F, 5, B),
(E, 7, D)}
```

5. Esploriamo ancora il primo elemento di **S**.
   Il nodo **G** (passando da **B**) ha solo il nostro nodo goal **I** come vicino e quindi aggiorniamo di conseguenza **V** e **S** →

   V={(A, ∅ , None),
   (B, 1, A)
   (C, 3, A),
   (D, 3, B),
   (G, 4, B)}

   S={(F, 5, B),
   (I, 6, G),
   (E, 7, D)}

6. Troviamo in questo caso il nodo **F** da esplorare che notiamo non avere altri archi a parte quello con **B** che è già esplorato, quindi aggiorniamo semplicemente **V** e **S** →

   V={(A, ∅ , None),
   (B, 1, A)
   (C, 3, A),
   (D, 3, B),
   (G, 4, B),
   (G, 4, D),
   (F, 5, B)}

   S={(I, 6, G),
   (E, 7, D)}

7. Arriviamo a notare che in cima alla coda di priorità abbiamo il nostro nodo goal e di conseguenza possiamo capire che passando per i nodi con priorità inferiore a **I** (e quindi con costo maggiore di **I** stesso) non è possibile avere un cammino minimo migliorare e ci si può fermare.

Aggiorniamo **V** e **S** un'ultima volta →

```
V={(A, ∅ , None),
(B, 1, A)
(C, 3, A),
(D, 3, B),
(G, 4, B),
(G, 4, D),
(F, 5, B),
(I, 6, G)}

S={(E, 7, D)}
```

Facciamo notare che l'algoritmo comunque finirebbe il proprio lavoro esplorando tutti i nodi ma implementando una coda di priorità si può decidere di avere questa piccola ottimizzazione.

8. Ora si prende in considerazione **V** e si esegue **BACKTRACKING** partendo da **I**, notiamo come **G** abbia però due precedenti che sono **B** oppure **D**, questo perchè il percorso minimo può essere più di uno solo, infatti avremo alla fine del backtracking →

   `A → B → G → I`

   `A → B → D → G → I`

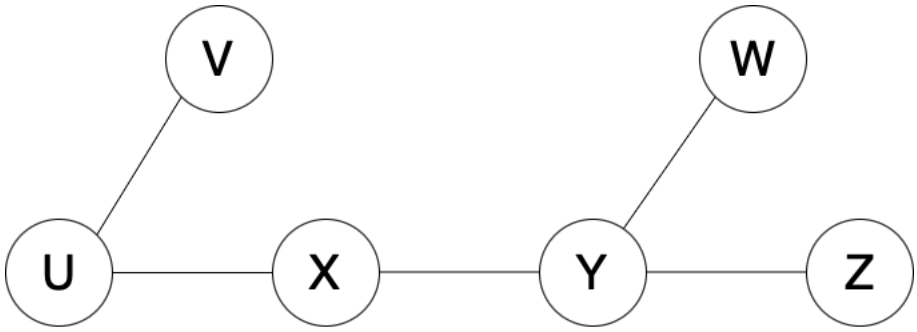
   Questi due percorsi sono allora equivalenti a livello di costo, entrambi i percorsi hanno costo `6`, tutti gli altri percorsi saranno sicuramente con un costo maggiore.

Ora che abbiamo capito come funziona l'algoritmo enunciamo la complessità temporale dell'algoritmo con alcune note sull'efficienza che si hanno dipendentemente dalle strutture dati che si implementano, denotiamo con $|N|$ la cardinalità dei nodi e con $|E|$ la cardinalità degli archi (in inglese Edge).

Se abbiamo come struttura degli array allora avremo
$O(|V|^2 + |E|) \simeq O(|V|^2)$
Dato che $|V|^2$ è il temine trainante.

Se implementiamo la struttura **HEAP** avremo
$O((|V| + |E|) \log_2 |V|) \simeq O(|V| \log_2 |V|)$

Tramite questo algoritmo avremo le nostre tabelle di inoltro, facciamo un esempio più piccolo per comprendere meglio



DESTINAZIONE	LINK
V	(U, V)
W	(U, X)
X	(U, X)
Y	(U, X)
Z	(U, X)

Figura 6.2: Esempio di tabelle di inoltro tramite algoritmo di Dijkstra

Capito come funziona l'algoritmo Dijkstra e capito come si possano ricavare le tabelle di inoltro ci soffermiamo sugli algoritmi LS che hanno:

- **CONDIZIONE INIZIALE:** Ogni router monitora e mantiene aggiornato lo stato dei suoi link incidenti. Implementata tramite un semplice `HELLO PROTOCOL`.

- **PASSO 1:** Ogni router invia in broadcast lo stato dei propri collegamenti, necessita ovviamente di un meccanismo di flooding affidabile.
- **PASSO 2:** Ogni router calcola i cammini minimi tra esso e tutti gli altri nodi, utilizzando il medesimo algoritmo.

Dato che i router eseguono l'algoritmo in maniera sincrona crea un problema ovvero le **OSCILLAZIONI**.
Se poniamo il caso, che è quello che vogliamo per avere meno congestione possibile, che i costi dei collegamenti riflettano il traffico sui collegamenti stessi, allora accade che si tenderà periodicamente a saturare uno o più collegamenti lasciando completamente liberi gli altri. Questo problema lo si risolve desincronizzando l'esecuzione dell'algoritmo, un aspetto buffo è dato dal fatto che i router tenderanno ad **AUTOSINCRONIZZARSI** e di conseguenza si necessita di una randomicità periodica dell'esecuzione dell'algoritmo per risolvere definitivamente il problema.

6.3 Distance-Vector Routing

Un algoritmo di routing distance-vector è iterativo, asincrono e distribuito, vediamo cosa significa:

- **ITERATIVO:** L'algoritmo non termina ma si blocca, si ripete fin tanto che non avviene un ulteriore scambio di informazione tra vicini
- **ASINCRONO:** L'algoritmo non richiede che i nodi operino contemporaneamente
- **DISTRIBUITO:** Ogni nodo non riceve informazioni globali ma solo relativi ai propri vicini a cui riferisce i risultati del calcolo

L'algoritmo DV viene denominato anche Bellman-Ford Routing poichè è basato sulla formula e l'algoritmo di Bellman-Ford, la formula è formalmente espressa come segue

$d_x(y) = min_v\{c(x,v) + d_v(y)\}$
$min_v \rightarrow$ riguarda il minito tra tutti i vicini `v` di `x`
$d_v(y) \rightarrow$ il costo del percorso a costo minimo dal nodo pedice
al nodo in parentesi

Questa formula è molto semplice, il significato è che il costo da un nodo `x` a un nodo `y` totale può essere suddiviso come il costo per andare da `x` a un vicino `v` e il costo totale da `v` al nodo destinazione `y`.
Questa formula è alla base dell'algoritmo Bellman-Ford per i cammini minimi che funge da supporto per l'algoritmo di routing DV che consente di creare le tabelle di inoltro.
Ora accenniamo il funzionamento dell'algoritmo Bellman-Ford nel suo utilizzo generale per determinare il cammino minimo nel grafo per poi capire meglio i vettori dell'algoritmo DV e come da questi vengono create le tabelle di inoltro. Si inizia dal dare al nodo sorgente il valore 0 e valore infinito a tutti gli altri. L'idea di base è quella di partire dal nodo sorgente ed esplorare i nodi adiacenti e assegnare a loro il costo per raggiungerli, il costo è determinato dal costo iniziale dal nodo sommato al costo dell'arco per raggiungere l'adiacente. Ovvero se siamo nel nodo sorgente che vale 0 il costo di un arco per raggiungere l'adiacente, ipotizziamo 3, allora un adiacente ha costo totale $0+3=3$.
A questo punto si procede nello stesso modo per ogni nodo: si esplorano i nodi adiacenti e si assegna loro il valore dell'arco percorso per raggiungerli più quello già assegnato al nodo da cui si è partiti, si aggiunta il valore solo se il nuovo valore è minore rispetto al preesistente, ovviamente alla prima iterazione tutti i valori saranno aggiornati poichè abbiamo impostato il valore dei nodi come $+\infty$. Oltre al valore per raggiungere il nodo si salva anche il predecessore, ovvero il nodo correlato al valore immediatamente precedente, questo permette backtracking.
Siamo certi allora che con N nodi dopo $N-1$ iterazioni l'algoritmo ha implementato il valore minimo per ogni nodo.
La complessità di questo algorimto è maggiore rispetto a quella di Dijkstra, dato $|V|$ numero di vertice e $|E|$ numero di archi la complessità sarà $O(|V| \cdot |E|)$. Diversamente da Dijkstra, Bellman-Ford, ci consente la dinamicità, ovvero non si necessitò di reiterare l'intero algoritmo su tutto il grafo se si modificano i costi di un sottoinsieme degli archi ma basterà iterarlo sui nodi interessati, lo vedremo meglio.
Quindi iniziando ad analizzare l'algoritmo DV diciamo che utilizza una variante su ogni nodo dell'algoritmo di Bellman-Ford, infatti avremo per ogni nodo il calcolo dei costi degli adiacenti che vengono posti in un vettore e inviati appunto ai vicini.
Facciamo un esempio $\rightarrow$

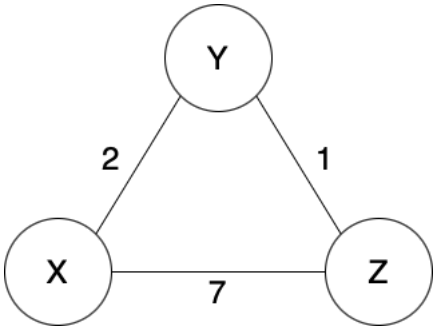


Figura 6.3: Figura per esempio di Distance Vector

Il vettore di `X` sarà allora:

$D_x = [0, 2, 7]$

Di conseguenza anche per `Y` e `Z` calcoleremo i loro vettori che saranno:

$D_y = [2, 0, 1]$
$D_z = [7, 1, 0]$

A questo punto ogni nodo invia il proprio vettore ai vicini che confrontano, calcolando nuovamente i costi, il proprio vettore con quello dei vicini andando a formare le tabelle:

Tabella nodo `X`

	X	Y	Z
X	0	2	7
Y	/	/	/
Z	/	/	/

Figura 6.4: Prima Iterazione

Arrivano i vattori di `Y` e `Z` e si aggiorna:

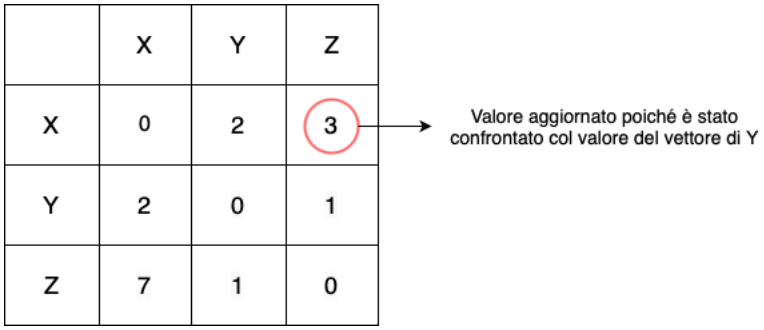


Figura 6.5: Aggiornamento dei valori nella tabella, singolo esempio

Allora si invia nuovamente il vettore $D_x = [0, 2, 3]$ poiché è stato possibile aggiornarlo.
In maniera asincrona anche `Z` ha calcolato la propria tabella dati che ha ricevuto un aggiornamento scoprendo di poter spendere `3` e non più `7` passando da `Y` esattamente come `X` e quindi aggiorna la propria tabella e invia nuovamente ai vicini il proprio vettore.
Alla fine di tutti gli aggiornamenti avremo un'unica tabella calcolata indipendentemente da ognuno dei 3 nodi che sarà:

	X	Y	Z
X	0	2	3
Y	2	0	1
Z	3	1	0

Figura 6.6: Tabella calcolata da ogni singolo nodo dopo tutti gli aggiornamenti

Va da sé che il lavoro svolto in questo esempio è il più complesso dato che bisognava esplorare tutto dal principio, generalmente però sono pochi i costi da aggiornare per ogni tabella.

6.3.1 Modifica dei costi e problemi nei collegamenti

Quando i costi dei collegamenti variano ci possiamo trovare in due situazioni: i collegamenti diminuiscono i propri costi oppure li aumentano. Premettiamo che queste due situazioni hanno due sviluppi molto differenti, in maniera concisa questi sviluppi si delineano con la frase: "Le buone notizie si propagano velocemente, le cattive notizie si propagano lentamente".
Vediamo nel dettaglio cosa accade nel nostro caso con i nodi `X`, `Y` e `Z`:

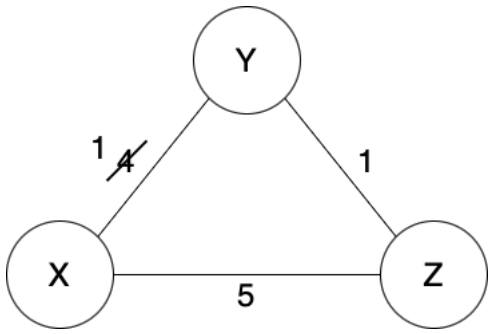


Figura 6.7: Esempio di diminuzione dei costi

Denotiamo che all'istante t_0 il nodo `Y` rileva un cambiamento del costo del proprio collegamento che passa da `4` a `1`.
Allora `Y` aggiorna la propria tabella e invia il proprio vettore ai propri vicini all'istante t_1.
All'istante t_2, `Y` riceve da `Z` il vettore aggiornato poichè il nodo `X` partendo da `Z` non ha più costo `5` ma `2`.
Arrivato a questo punto l'algoritmo arriva nel suo stato di quiete temporale in pochi passaggi. Quindi questo ci concretizza la prima parte della frase "Le buone notizie si propagano velocemente". Ora analizziamo "Le cattive notizie si propagano lentamente" ponendoci nel caso seguente:

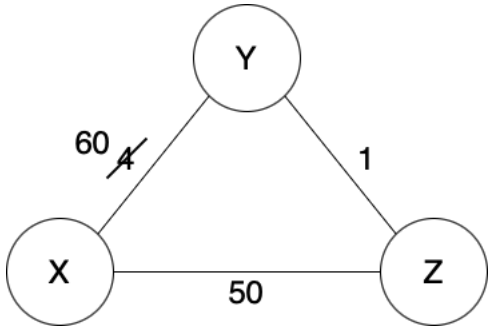


Figura 6.8: Le cattive notizie si propagano lentamente

Vediamo come $c(Y, X)$ passa da `4` a `60`, a questo punto abbiamo che `Z` ha come costo di arrivo a `X`, quindi $c(Z, X)$, pari a `5` poiché passando per `Y` si avrà $c(Z, Y) + c(Y, X) = 1 + 4 = 5$. Quando viene modificato $c(Y, X)$ da `4` a `60`, `Y` ricalcola il proprio vettore con la formula Bellman-Ford, quindi:

$$C_y(X) = min\{c(Y, X) + C_x(X), c(Y, Z) + C_z(X)\} =$$
$$min\{60 + 0, 1 + 5\} = 6$$

Questo costo calcolato è però errato poiché `Z` ha comunque il costo dell'instradamento verso `X` calcolato passando da `Y`.A questo punto avremo `Y` che invia a `Z` il proprio vettore aggiornato il quale rielabora il proprio:

$$C_z(X) = min\{c(Z, X) + C_x(X), c(Z, Y) + C_y(X)\} =$$
$$min\{50 + 0, 1 + 6\} = 7$$

Nuovamente `Z` invia il proprio vettore a `Y` che nuovamente farà il proprio aggiornamento, questo loop di aggiornamenti si denomina **PROBLEMA DEL CONTEGGIO ALL'INFINITO**. Infatti prima di arrivare ad uno stato di quiete si dovrà calcolare il nuovo valore un determinato numero di volte che potrebbe tendere all'infinito. Nel nostro caso l'algoritmo ha bisogno di 44 iterazioni, dopo queste iterazioni `Z` si renderà conto che il percorso con il costo migliore è quello del collegamento diretto. Se invece di un cambiamento del costo così basso si avesse avuto un cambiamento da `4` a `10 000` si avrebbero molte più iterazioni. Questo problema ci spinge ad avere un **INSTRADAMENTO CICLICO**, infatti i pacchetti verranno instradati in maniera compulsiva tra `Y` e `Z` senza mai raggiungere `X` fintanto che il valore non viene calcolato correttamente.

6.3.2 Split Horizon Semplice e con Inversione Avvelenata

L'idea alla base dello **split horizon semplice** è l'omissione dei percorsi che ha appreso proprio da quel vicino. Facciamo un esempio pratico: Il router **A** invia pacchetti a **C** attraverso **B**, di conseguenza la topologia sarà $A \rightarrow B \rightarrow C$, A apprenderò la presenza di C da B ma quando invierà a B la propria tabella ometterà il percorso verso C. Nel caso di interruzione del collegamento tra B e C grazie all'omissione di A verso B di poter raggiungere C proprio tramite B, non si crea il ciclo, poiché altrimenti B non sarebbe a conoscenza che il percorso di A per raggiungere C include lui stesso e quindi ci sarebbero dei rimbalzi dei pacchetti tra A e B. Questa versione di risoluzione del problema può essere migliorata grazie all'**INVERSIONE AVVELENATA**, in questo caso lo spit horizon permette ad A di inviare a B l'informazione che ha un collegamento verso C ma lo pone a $+\infty$ anche se così non è. Questo metodo però non risolve in generale la problematica, si limita a risolverla nel caso in cui si crei il problema tra nodi adiacenti ma se il ciclo si creasse tra nodi non adiacenti non sarebbe di nessun aiuto.

6.3.3 Confronto LS vs DV

Prima di addentrarci nell'instradamento all'interno di un ISP e tra ISP confrontiamo le due classi di algoritmi visti finora. Possiamo immediatamente dire che nel DV ciascun nodo dialoga SOLO con i vicini informandoli delle stime di costi con tutti gli altri nodi a lui vicini. LS invece dialoga via broadcast con tutti i nodi comunicando solo i costi dei collegamenti verso i suoi vicini. Possiamo inoltre confrontare tali algoritmi in 3 macro-aree:

- **COMPLESSITA' DEI MESSAGGI:** LS impone che ogni nodo conosca i costi di tutti i collegamenti avendo quindi una complessità di invio pari a $O(|V| \cdot |E|)$, ogni volta che un costo varia deve essere rispedito in broadcast. DV invece si occupa solo di messaggi tra nodi adiacenti, la complessità varia in base a molti fattori. Inoltre se una variazione di costo si presenta c'è bisogno di una propagazione nella rete.
- **VELOCITA' DI CONVERGENZA:** LS necessita di una complessità $O(|V|^2)$ per eseguire Dijkstra e di una complessità $O(|V| \cdot$

$|E|$) per eseguire l'invio di messaggi. DV invece converge lentamente (ricordiamo le cattive notizie), ma in generale più lentamente di LS e può causare cicli di instradamento dovuti al conteggio all'infinito.

- **ROBUSTEZZA:** In caso di guasti LS può comunicare dei costi errati ma dato che ogni nodo calcola solo le proprie tabelle di inoltro, si crea un certo grado di isolamento che si traduce in robustezza. Per DV invece un calcolo errato può significare una propagazione dell'errore nell'intera rete.

Di fatto questo ci porta a dire che non vi è un algoritmo migliore rispetto all'altro e infatti in internet vengono utilizzati entrambi.

6.4 Sistemi Autonomi e Instradamento Interno

Finora abbiamo trattato la rete come un sistema che ha al suo interno una collezione di router. Questa visione per quanto corretta si differenzia dalla realtà poichè è molto semplicistica e perchè abbiamo visto che internet è una rete di reti, quindi abbiamo più ISP interconnessi tra loro. La nostra visione incorre allora in due problemi:

- **SCALABILITA':** Il numero sempre crescente di router determina tempi inconcepibili per calcolare, memorizzare e comunicare informazioni di instradamento.
- **AUTONOMIA AMMINISTRATIVA:** Dato che parliamo di interconnessione di ISP è desiderabile che ognuno di loro voglia amministrare i propri router come meglio crede.

Queste complessità si risolvono organizzando i router in **SISTEMI AUTONOMI**, ogni ISP sceglie se raggruppare i propri router in un singolo AS oppure crearne diversi, nel secondo caso ogni AS viene identificato tramite un numero univoco da ICANN. In questo modo ogni ISP gestisce il proprio **INSTRADAMENTO INTERNO AL SISTEMA AUTONOMO** tramite un protocollo.

6.5 Open Shortest Path First - OSPF

OSPF è un protocollo link-state che utilizza il flooding di informazioni per lo stato dei collegamenti e Dijkstra per il percorso a costo mini-

mo. Ogni router costruisce un grafo dell'intero AS, a questo punto esegue localmente Dijkstra per comprendere tutti i cammini minimi verso tutte le sottoreti. I costi dei collegamenti vengono impostati dall'amministratore.
Se lo stato di un collegamento cambia, allora OSPF invia informazioni di instradamento via broadcast ai router nel AS, questo inoltre viene fatto periodicamente anche se lo stato non è mutato.
OSPF ha i seguenti vantaggi:

- **SICUREZZA:** Gli scambi tra router possono essere autenticati, quindi solo router appartenenti al AS possono partecipare al protocollo. Si hanno due modalità di autenticazione: semplice o tramite MD5.
- **PERCORSI CON LO STESSO COSTO:** Quando risultano più percorsi con il medesimo costo, OSPF permette di usarli tutti senza appesantirne uno solo.
- **SUPPORTO PER INSTRADAMENTO UNICAST E MULTICAST:** Per il multicast OSPF diventa MOSPF e sfrutta il database dei collegamenti OSPF aggiungendo una tipologia di annuncio nello stato dei collegamenti al meccanismo broadcast.
- **GERARCHIE:** OSPF configura i router in aree, una delle più importanti è l'**Area di Dorsale** tramite i propri **Router di Confine**, che instradano i pacchetti verso l'esterno. Un AS determina che i pacchetti siano inviati prima ai router di confine e attraverso la dorsale raggiungano l'area dell'AS di destinazione.

6.6 Instradamento tra ISP e BGP

Abbiamo visto come vengono instradati i pacchetti all'interno degli AS ma come abbiamo visto gli AS sono collegati tra loro.
Il protocollo di instradamento **INTER-AS** è il **BORDER GATEWAY PROTOCOL (BGP)**, la sua importanza è estrema poiché è il collante che tiene insieme gli ISP. Il protocollo è della tipologia distance-vector, decentralizzato e asincrono. BGP è molto complesso ma bisogna comprenderlo almeno superficialmente.
BGP ha come ruolo quello di stabilire le destinazioni esterne all'AS

come occorrenze delle tabelle di inoltro. Premettiamo che i pacchetti non vengono instradati verso un preciso indirizzo IP ma verso una sottorete specificata tramite **PREFISSI CIDR**. Le occorrenze BGp nella tabella di inoltro sono del tipo `(x, I)` dove `x` è un prefisso e `I` è un'interfaccia del router. BGP per fare questo svolge due operazioni fondamentali:

- **OTTENERE INFO SULLA RAGGIUNGIBILITA' DEI PREFISSI DI SOTTORETE DA PARTE DEI SISTEMI CONFINANTI:** BGP permette alla sottorete di essere conosciuta a tutti i router di Internet
- **DETERMINARE PERCORSI OTTIMI VERSO SOTTORETI:** Date le informazioni della precedente operazione BGP viene eseguito localmente (nel router) per determinare i percorsi migliori.

6.6.1 Distribuzione

Definiamo tre AS: `AS1, AS2, AS3`, una sottorete `x` che appartiene a `AS3`, ogni AS ha router interni e un router gatewat che connette l'AS di appartenenza agli altri AS. Gli AS definiti sono così definiti:

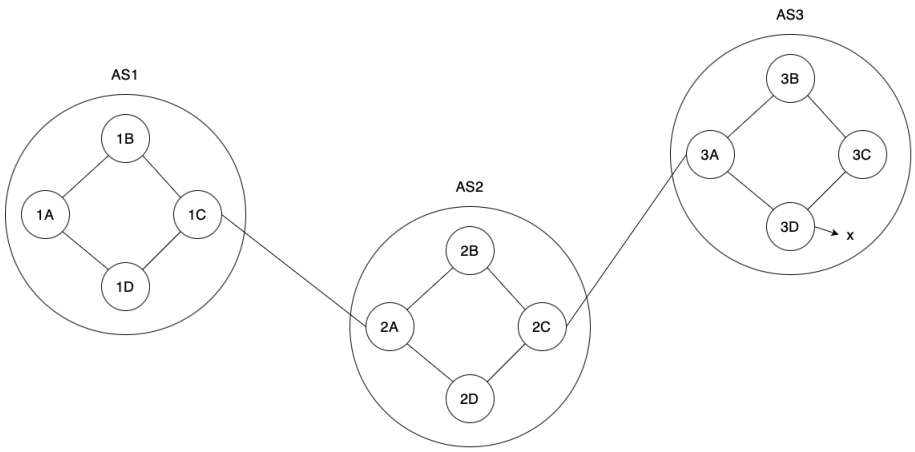


Figura 6.9: Topologia di tre Anonimous Systems

`AS3`, che detiene `x`, invia a `AS2` il messaggio `"AS3 x"` per far sapere ad `AS2` che tramite lui si raggiunge `x`, allo stesso modo `AS2` invia ad

`AS1: "AS2; AS3 x"`. `AS1` tramite questi messaggi conosce il percorso per arrivare a `x`, per propagare queste informazioni BGP utilizza la sessione BGP tramite TCP sulla porta 179, le connessioni TCP sono semi-permanenti. Se la sessione è tra router interni all'AS allora è una sessione BGP interna, o iBGP, se connette AS allora si chiama sessione BGP esterna, o eBGP.

6.6.2 Selezione delle Rotte Migliori

Quando BGP annuncia le rotte si avvale degli **Attributi BGP** oltre al messaggio che abbiamo descritto, una **ROTTA** è infatti l'unione del prefisso a dei suoi attributi. I due più importanti sono:

- **AS-PATH:** elenca gli AS attraverso i quali è passato l'annuncio del prefisso. In questo modo si evitano reiterazioni sugli annunci.
- **NEXT-HOP:** indica l'indirizzo IP del router che deve essere usato come next-hop verso la destinazione specificata (nella figura sarebbe `2A`)

6.7 Algoritmi di Instradamento

Vediamo un algoritmo semplicistico denominato **Hot Potato** e un algoritmo che invece **BGP** adatta veramente:

- **HOT POTATO:**
  1. Dal protocollo inter-AS (OSPF) si apprende che la sottorete `x` è raggiungibile attraverso più gateway
  2. Si usa l'informazione di instradamento per determinare i costi dei percorsi a costo minimo verso gateway
  3. A questo punto hot potato sceglie il gateway a costo minimo più conveniente, senza comprendere i costi successivi
  4. Dalla tabella di inoltro si determina l'interfaccia `I` che conduce al gateway a costo minimo, si scrive `(x, I)` sulla tabella di inoltro
- **Algoritmo usato da BGP:**
  L'algoritmo esegue dei passaggi di eliminazione per determinare la rotta migliore:

1. Alle rotte si assegna un valore di **Preferenza Locale**, questo valore è assegnato dal router stesso oppure importato da un altro
2. Tra le rotte che hanno medesima preferenza locale si scelgono quelle con `AS-PATH` più breve
3. Tra le rotte con `AS-PATH` più breve si scelgono quelle che hanno il `NEXT-HOP` più vicino
4. Se vi è più di una rotta allora si sceglie in base agli identificatori BGP

6.7.1 Anycast IP

BGP implementa inoltre il servizio **anycast IP** ovvero la possibilità di inviare a più indirizzi IP, da non confondere col multicast, poichè l'anycast consegna pacchetti ad uno qualsiasi della rete o sottorete che deve raggiungere, uno per tutta la rete, solitamente il più vicino. Questo servizio è utile soprattutto per l'utilizzo di CDN che devono replicare i propri contenuti e distribuirli, utilizzando il servizio con BGP, infatti il CDN può assegnare a ogni macchina un unico indirizzo e facilitare dunque le rotte verso le proprie macchine, così facendo si assicura di avere il CDN più vicino a distribuire i contenuti.

6.8 Il Piano di Controllo SDN

Identifichiamo quattro caratteristiche fondamentali in una architettura SDN:

- **Inoltro Basato sui Flussi:** I pacchetti possono essere inoltrati da uno switch SDN sul valore dei campi dell'intestazione del livello di trasporto, rete e collegamento. Questo approccio è opposto all'approccio tradizionale dei router dove l'inoltro del datagramma veniva effettuato basandosi esclusivamente sull'indirizzo IP di destinazione. Le regole di inoltro indipendentemente dall'approccio sono stabilite all'interno delle tabelle di flusso, il piano di controllo SDN in una architettura SDN ha il compito di calcolarle, gestirle e installare le loro occorrenze all'interno degli switch.
- **Separazione Piano dei Dati e Piano di Controllo:** La separazione in questa architettura è netta, gli switch che eseguono re-

gole match-action si occupano del piano dati, mentre i server SDN gestiscono il piano di controllo determinando e amministrando le tabelle di flussi.

- **Funzioni di Controllo Esterne al Piano Dati (Switch):** Il piano di controllo viene implementato su server distinti e remoti rispetto agli switch. Il piano di controllo viene implementato tramite due componenti: il controller SDN e un insieme di applicazioni di controllo di rete. Il controller SDN mantiene informazioni di stato della rete e le fornisce alle applicazioni di controllo, fornisce inoltre il mezzo attraverso il quale le applicazioni possono monitorare, programmare e controllare i dispositivi di rete sottostanti.
- **Una Rete Programmabile:** Le applicazioni sono i cervelli del piano di controllo SDN usando le API fornite dal controller SDN per determinare e controllare il piano dei dati negli switch. Attraverso queste applicazioni la rete diventa programmabile.

6.8.1 Controller SDN e Applicazioni di Controllo

Il piano di controllo abbiamo visto che è suddiviso in controller SDN e in applicazioni di controllo. Queste due componenti sono suddivise su tre livelli di funzionalità, vediamo questi tre livelli *bottom-up*:

1. **Livello di Comunicazione:** Questo livello si occupa della comunicazione tra il controller SDN e i dispositivi di rete. Dati che il controller deve gestire gli switch, gli host e altri dispositivi, necessita di un protocollo che trasferisca informazioni e che mantenga aggiornata la visione dei dispositivi al controller SDN. Tale comunicazione passa attraverso l'interfaccia denominata **SOUTH-BOUND** e il protocollo di maggiore utilizzo è **OPEN FLOW**.
2. **Livello di Gestione Globale della Rete:** Il controller deve mantenere informazioni aggiornate sullo stato dei link, degli host, degli switch, ecc., solo in questo modo si possono prendere decisioni finali coerenti, dato che il piano di controllo ha come scopo ultimo quello di generare tabelle di flusso che contengono contatori molto importanti per le applicazioni di controllo si mantengono

copie di queste tabelle. Tutte queste informazioni costituiscono lo stato globale della rete.

3. **Interfaccia con il Livello di Applicazione di Controllo della Rete:** Le API dell'interfaccia *north-bound* permettono al controller di comuncare con le applicazioni di controllo che possono allora leggere/scrivere lo stato della rete e le tabelle di flussi nel livello di gestione di stato globale della rete. Questo permette un pronto intervento delle applicazioni ai cambiamenti di stato della rete. Le API più comunemente utilizzate sono di tipo REST.

6.8.2 Open Flow

Il protocollo Open Flow opera tra un controller SDN e un dispositivo sottostante comunicando tramite API OpenFlow con connessione TCP su porta `6653`.
Vediamo ora i più famosi ed utilizzati messaggi inviati dal controller allo switch:

- **CONFIGURATION:** Permette al controller di interrogare e impostare parametri nella configurazione di uno switch
- **MODIFY-STATE:**Permette al controller di aggiungere, modificare o cancellare le occorrenze delle tabelle di flussi dello specifico switch e di impostare proprietà sulle sue porte
- **READ-STATE:** Permette al controller di acquisire statistiche e valori di contatori delle tabelle di flussi dello switch nonchè delle sue porte
- **SEND-PACKET:** Permette al controller di inviare un pacchetto specifico tramite una specifica porta dello switch, il messaggio contiene esso stesso il pacchetto

Vediamo ora i messaggi più utilizzati dallo switch al controller:

- **FLOW-REMOVED:** lo switch informa il controller che una occorrenza della propria tabella di flussi è stata cancellata, per timeout oppure tramite MODIFY-STATE
- **PORT-STATUS:** lo switch informa il controller che una sua porta ha cambiato stato

- **PACKED-IN:** questo messaggio permette allo switch di inviare un pacchetto al controller perchè necessita di un'ulteriore elaborazione.

6.9 ICMP - Internet Control Message Protocol

Ora che abbiamo visto il piano di controllo SDN, apriamo una parentesi sulla gestione e iniziamo parlando del protocollo **ICMP** e dei suoi messaggi. Il suo utilizzo solito è il notificare errori.
Viene spesso considerato parte di IP anche se effettivamente opera sopra IP dato che i messaggi ICMP sono incapsulati all'interno dei datagrammi IP, esattamente come avviene per i segmenti TCP e i datagrammi UDP, ed esattamente come per i messaggi di livello di trasporto, gli host che ricevono un datagramma IP con specifiche ICMP si effettua un demultiplexing.
I messaggi ICMP si descrivono di un campo **TIPO** e un campo **CODICE** e contengono l'intestazione e i primi `8 byte` del datagramma IP che ha provocato l'errore. Vediamo alcuni messaggi ICMP:

TIPO	CODICE	DESCRIZIONE
0	0	Risposta ECHO (a ping)
3	0	Rete non raggiungibile
3	1	Host non raggiungibile
3	3	Porta non raggiungibile
8	0	Richiesta ECHO

Figura 6.10: TIPO e CODICE dei messaggi ICMP più comuni con descrizione

Si può vedere come ICMP non ha un esclusivo uso di notificare er-

rori. Un programma che effettua `ping` invia messaggi ICMP con `tipo: 8` e `codice: 0`, la richiesta di `echo`, l'host che riceve la richiesta risponde a sua volta con un messaggio ICMP con `tipo: 0` e `codice: 0`. Bisogna anche sottolineare un utilizzo di ICMP che è stata sostituita da TCP ma mai effettivamente deprecato, ICMP infatti veniva utilizzato per un controllo di congestione, il router che rileva una congestione invia un messaggio ICMP con `tipo:4` e `codice:0` all'host che riduce il tasso trasmissivo.

6.10 Gestione della Rete

La rete come abbiamo visto è un complesso sistema di software e hardware e spesso gestire una rete per un amministratore è una sfida, infatti sia ora con SDN, sia prima di SDN la rete necessita di un sistema di monitoraggio e controllo della rete. Nello specifico però vediamo cosa significa gestire una rete, analizzando architettura, protocolli e informazioni di base utilizzati dagli amministratori.

6.10.1 Infrastruttura di Gestione

Vediamo i componenti chiave della gestione di rete:

- **Server di Gestione:** Applicazione controllata dal responsabile ed eseguita da un elaboratore del Network Operations Center (NOC). Il componente ha la funzionalità di verificare la situazione della rete e dove si intraprendono azioni verso i dispositivi di rete. Questo è permesso dalla raccolta, l'analisi e l'elaborazione, anche con visualizzazione, delle informazioni del comportamento della rete.
- **Dispositivo di Rete Gestito:** Un dispositivo di rete gestito o anche oggetto da gestire è un componente hardware o software della rete come un host, un router, un bridge, una stampante ecc, ma anche un protocollo di instradamento
- **Management Information Base (MIB):** È una base di dati gestionali dove si archiviano informazioni proprie dell'oggetto da gestire che il server di gestione può recuperare e anche modificare. Di pari passo al MIB abbiamo anche il SMI ovvero **Structure Of Management Information**, cioè un linguaggio di specifica dei

dati, assicura che la semantica e la sintassi dei dati della gestione della rete siano consistenti, ben definite e non ambigui.

- **Agente di Gestione:** È un processo in esecuzione sull'oggetto da gestire che si occupa di comunicazione con il server di gestione ed eseguire le azioni che vengono dettate dallo stesso server di gestione.
- **Protocollo di Gestione:** È uno strumento di comunicazione tra l'agente e il server, le due macchine parlano rispettando un determinato protocollo di gestione ma bisogna sottolineare che non gestisce esso stesso la rete ma uno strumento dell'amministratore per operare la gestione che può poi riguardare altri protocolli (con OSPF).

6.11 Simple Network Management Protocol - SNMP

SNMPv2 è un protocollo di livello applicazione che consente la comunicazione tra server di gestione e agente, infatti trasporta informazioni tra entità. Il suo utilizzo è la modalità *Richiesta-Risposta*, una entità di gestione invia una richiesta a un agente e come conseguenza l'agente esegue delle azioni e invia una risposta. L'eccezione di questa prassi è data dal messaggio **TRAP** dove un agente invia un messaggio pur non avendo ricevuto una risposta. Schematizziamo la struttura dei messaggi, chiamati **PDU - Protocol Data Unit** per SNMPv2 prima di analizzare la tipologia:

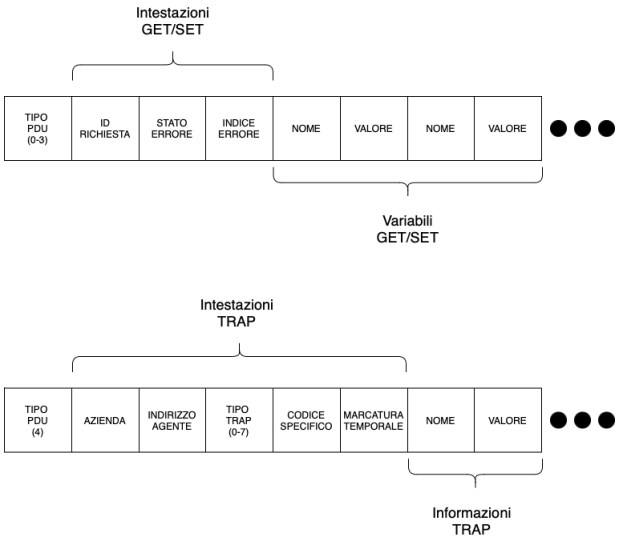


Figura 6.11: Struttura dei messaggi SNMPv2

Vediamo ora la tipologia di messaggi di SNMPv2:

- **GetRequest, GetNextRequest, GetBulkRequest**
  Questi messaggi sono inviati dal server di gestione all'agente per richiedere valori o oggetti MIB.
  GetRequest può richiedere un valore MIB qualsiasi, mentre per elenchi e tabelle si usa **GetNextRequest**.
  GetBulkRequest restituisce grandi porzioni di dati.
  L'agente invia un PDU Response contenente identificatori e valori che fanno riferimento agli oggetti MIB richiesti.

- **SetRequest**
  Permette al server di impostare il valore di uno o più oggetti MIB, l'agente risponde con Response contente `"NoError"`

- **TRAP**
  Messaggio **asincrono** che viene inviato dall'agente al server perchè si è verificato un evento di cui il server ha chiesto notifica. Tra i vari eventi ci sono `Reboot` a freddo, a caldo, disponibilità di un collegamento, perdita di un vicino o mancata autenticazione. **Il server non è obbligato a rispondere.**

`SNMPv2` utilizza **UDP** come protocollo di trasporto.

Capitolo 7

Link Layer

Ora iniziamo la nostra trattazione sull'ultimo livello della pila TCP/IP che vediamo in quanto informatici, poiché lasciamo ad ingegneri e fisici il compito di occuparsi del livello fisico, che noi assumiamo funzioni perfettamente,quindi che sia stato progettato in maniera consona. Nel corso della nostra analisi sul livello di collegamento vedremo come vi siano principalmente due tipologie di canali:

- La tipologia di canali broadcast dove un gruppo di host si connettono alle LAN wireless, reti satellitari, reti di accesso HFC. Reso possibile da un protocollo per coordinare la trasmissione di frame gestendo l'accesso al mezzo, **Medium Access Protocol)**.
- La tipologia di canale di comunicazione punto a punto, reso possibile dal protocollo **PPP - Point-to-Point Protocol**, che sicuramente ha una più semplice gestione ed è anche il più utilizzato

Premettiamo alcune definizioni che utilizzeremo: chiamaeremo **NODO** facendo riferimento a qualunque dispositivo che opera a livello di collegamento e chiameremo **COLLEGAMENTO** ciascun canale di comunicazione che collegano i nodi adiacenti lungo il percorso.
L'idea di base è sempre la stessa: ogni nodo che deve spedire un datagramma lo incapsula in un **FRAME** del livello di collegamento e lo immette nel collegamento stesso. Quello che però bisogna comprendere è che non abbiamo una sola tipologia di collegamento, per avere un'idea generale possiamo fare una analogia col mondo dei trasporti. Immaginiziamo di avere una agenzia di viaggi che deve programmare un viaggio per una persona da una città A ad una città D.

Per fare questo l'agenzia programma un viaggio in macchina da A a B, un viaggio in treno da B a C e infine un viaggio in aereo da C a D.
Quindi avremo la nostra agenzia come protocollo di instradamento, sarà compito delle varie compagnie di auto, treni e aerei muovere il passeggero (che funge da datagramma), sebbene facciamo il medesimo lavoro lo fanno in maniera diversa quindi il tratto di viaggio è il collegamento mentre il modo in cui il passeggero viaggia è il protocollo utilizzato.

7.1 Servizi del Livello di Collegamento

Iniziamo una panoramica generale dei servizi offerti dai vari protocolli di collegamento, si sottolinea il fatto che i seguenti servizi non vengono implementati tutti da tutti i protocolli ma vedremo i protcolli e i servizi propri del singolo più avanti:

- **FRAMING:**
  Quasi tutti i protocolli incapsulano i datagrammi in un frame prima di trasmetterlo. Il frame si costituisce come al solito di campi di intestazione e del campo dati dove vengono inseriti i datagrammi. Ogni protocollo ha una specifica proprio il frame.

- **ACCESSO AL COLLEGAMENTO:**
  Le regole necessarie per immettere i frame nel mezzo trasmissivo vengono dettate dal protocollo che controlla l'accesso al mezzo trasmissivo, **MAC**. Il più semplice utilizzo del protocollo MAC (Medium Access Control) è la comunicazione punto a punto con un singolo host e un singolo collegamento in cui l'host immette. Un caso molto più interessante è un collegamento dove bisogna gestire un canale ad accessi multipli, in questo il protocollo MAC è un caso di studio più complesso.

- **CONSEGNA AFFIDABILE:**
  I protocolli di collegamento gestiscono la consegna affidabile del datagramma, diverso dal controllo del livello di trasporto poiché viene fatto a livello software, mentre a livello di collegamento si ha un controllo hardware. Sebbene la tipologia sia diversa abbiamo una metodologia simile, abbiamo ACK e ritrasmissioni sebbene vi siano anche metodi per correggere l'errore localmente. Questo ser-

vizio non viene implementato da tutti i protocolli di collegamento poiché è inutile sul collegamento con basso tasso di errore.

- **RILEVAZIONE E CORREZIONE DEGLI ERRORI:** Alcuni protocolli implementano un servizio di rilevazione e correzione degli errori inserendo, da parte del nodo che trasmette, dei bit di controllo. Infatti ci possono essere disturbi lungo il collegamento che alterano il frame e il datagramma all'interno ed è uno spreco di risorse inviare datagrammi errati. Il **CHECKSUM** effettuato dal livello trasporto poiché effettuato mediante hardware. Si riesce infatti ad individuare il punto preciso in cui il frame è errato.

7.2 Dov'è implementato il Livello di Collegamento

Abbiamo già visto le **LINE CARD** dei router e sappiamo che per quanto riguarda questi dispositivi il livello di collegamento è implementato proprio qui.
Quindi ora ci concentriamo sui sistemi periferici, per un dato collegamento abbiamo il protocollo di collegamento che è implementato da un **ADATTATORE DI RETE (NETWORK ADAPTER)**, viene anche definito **SCHEDA DI RETE (NIC, NETWORK INTERFACE CONTROLLER)**.
L'adattatore di rete ha come cuore il controller a livello di collegamento, solitamente questo controller implementa la maggioranza dei servizi a livello di collegamento. Questo ci fa notare come il livello di collegamento è molto dipendente dall'hardware, molto viene implementato a livello hardware.
Il comportamento è lo stesso visto finora, il lato mittente viene gestito dal proprio controller che prende un datagramma, passato dai livelli superiori e lo incapsula in un frame a livello di collegamento dove pone i doverosi campi di intestazione, seguendo poi il protocollo di accesso al collegamento e lo immette sul collegamento stesso. Il lato destinatario fa "l'opposto", il controller riceve il frame da cui estrapola il datagramma e lo consegna al livello superiore (quello di rete). Ci si rende conto di come il livello di collegamento viene implementato sulla scheda di rete ma tutta la parte di assemblaggio e anche di verifica viene eseguita dalla CPU, quindi a livello software. Possiamo quindi decretare come

il livello di collegamento sia effettivamente il punto cruciale della pila dove si incrociano hardware e software.

7.3 Tecniche di Rilevazione e Correzione Errori

Vediamo quali sono le tecniche più famose di rilevazione e correzione degli errori, vedremo sinteticamente questo aspetto solo per dare un'idea di quello che effettivamente è l'argomento.
In linea di massima abbiamo:

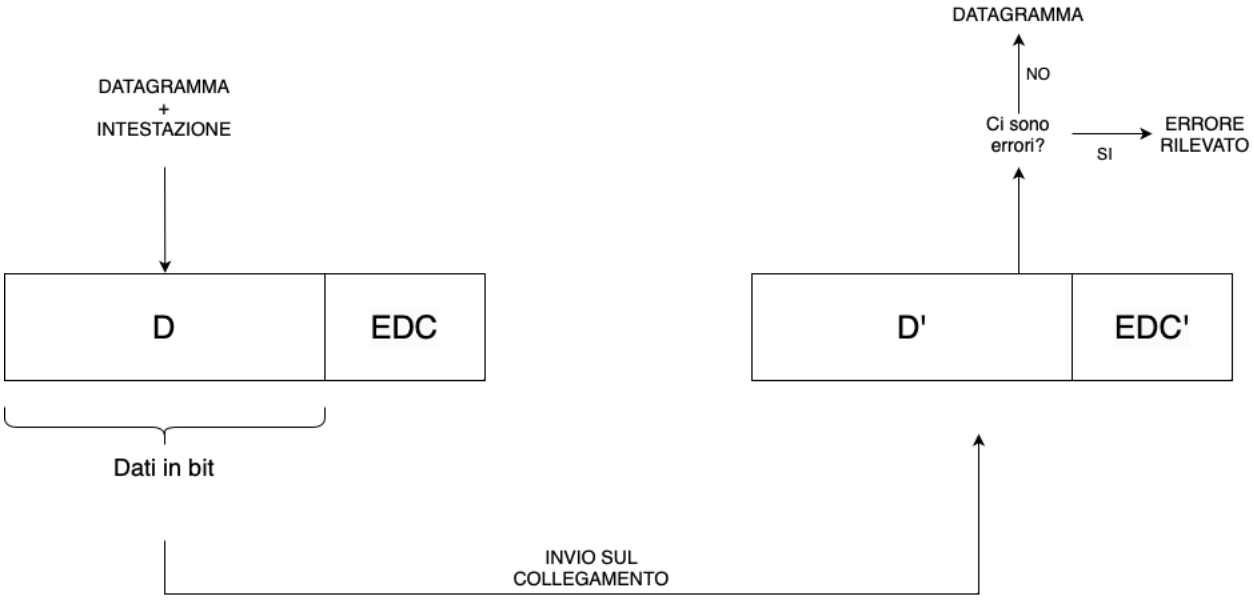


Figura 7.1: Esempio di Error Detection And Correction

EDC è l'acronimo di **Error Detection And Correction** che sono essenzialmente dei bit che ci consentono di rilevare un errore. Purtroppo a volte questo non basta è possibile che si verifichi la spiacevole situazione dove non vengono rilevati errori pur essendoci, se il nodo ricevente si trova in questa situazione allora potrebbe consegnare un datagramma alterato al livello di rete. *Ricordiamo che vi sono altri controlli sugli errori a livelli più alti della pila.*
Vediamo allora le tre tecniche più famose per controllo e correzione errori:

- **Controllo di Parità - Parity Check**
- **Checksum**
- **Controllo di Ridondanza Ciclica - Cyclic Redundancy Check**

7.3.1 Controllo di Parità

Questa è la tecnica più semplice di rilevazione di errore, non si occupa di correggerlo.
Utilizza un unico bit di parità (parity bit) che viene aggiunto alle informazioni da inviare, il bit viene scelto in base alle informazioni a cui di aggiunge, se vuole ottenere un numero pari di bit a 1 e quindi nel caso in cui le informazioni abbiano i bit a 1 in un numero pari allora il valore del bit di parità sarà 0, altrimenti avrà valore 1.
Se il numero di bit a 1 è dispari allora si rileva almeno un errore. Questa situazione allora ci possiamo aspettare si abbia nel momento in cui il numero di errori siano dispari, se gli errori sono in numero pari l'errore non viene rilevato. Statisticamente si è appurato che gli errori, se avvengono, avvengono a raffica, di conseguenza abbiamo che gli errori non vengono rilevati, in una situazione con un solo bit di parità, con una probabilità quasi del 50%.
Da questa semplice regola possiamo generalizzare un approccio che ci consente anche di applicare la correzione dell'errore.
Questa generalizzazione prende il nome di **Parità Bidimensionale**, ciò che viene fatto è essenzialmente che i bit di informazione vengono suddivisi in i righe e j colonne e si calcola per ognuna un valore di parità. I risultati di $i + j + 1$ *bit* di parità contengono bit per la rilevazione dell'errore nei frame a livello di collegamento. In questo modo il controller ricevente non solo può rilevare l'errore ma può identificare il posizionamento del bit errato e quindi correggerlo.
Questa proprietà che rileva e corregge l'errore è detta **Forward Error Correcrion**, questa tecnica è solitamente molto apprezzata perchè diminuisce drasticamente le ritrasmissioni.

7.3.2 Checksum di Internet

Come abbiamo già visto nel capitolo dedicato al livello di trasporto i bit dell'informazione da recapitare sono trattati come una sequenza di

numeri interi da K *bit*, quindi si sommano gli interi e il risultato lo si usa per la rilevazione degli errori. Il checksum in questa specifica situazione, del livello di collegamento, si basa sull'approccio appena descritto usando numeri interi da 16 *bit*. Il complemento a 1 di questa somma viene immesso nell'intestazione e spedito. Il ricevente allora controlla che non vi siano errori calcolando la somma dei numeri da 16 *bit* e il checksum, se il risultato è un numero dove tutti i bit sono a 1 allora non vi sono errori, in caso contrario è stato rilevato.
Bisogna fare una precisazione importante: IP esegue il checksum solo dell'intestazione poichè sappiamo che il contenuto è controllato dal checksum di TCP/UDP.

7.3.3 Controllo a Ridondanza Ciclica - CRC

La tecnica più utilizzata è il CRC che si basa sui **Codici di Controllo a Ridondanza Ciclica** anche detti **Codici Polinomiali**. L'idea è di interpretare la stringa di bit da inviare come un polinomio dove i bit della stringa sono i suoi coefficienti e intendere le operazioni sui bit come operazioni in aritmetica polinomiale. Consideriamo d *bit* che costituiscono i dati (D) da trasmettere e supponiamo che sorgente e destinazione si siano accordati su una stringa di $r+1$ *bit* che chiameremo **GENERATORE - G**, il bit più significativo del nostro generatore deve essere 1.
Allora il mittente concatenerà a D un numero r di bit, R, così da avere un totale di $d+r$ *bit*, divisibile per G in maniera esatta.
Da questo capiamo che se la divisione $(d+r)/G$ dà resto allora sicuramente si è verificato un errore.
I calcoli in CRC sono in aritmetica in modulo 2 e questo ci consente di stabilire che addizione e sottrazione sono sostanzialmente uno XOR.
Moltiplicazione e divisione invece sono eseguite in base 2 e quindi abbiamo un semplice *shift* (destro o sinistro).
Dato questo possiamo dire che $D \times 2^r XOR R$ ci restituirà esattamente $d+r$ *bit*. Tornando al discorso principale avremo che:

$$D \times 2^r \ XOR \ R \equiv nG$$

Di conseguenza avremo che:

$$R = resto \ di \ (D \times 2^r / G)$$

Questa tecnica consente di rilevare errori con un burst inferiore a $r + 1$ *bit*, ovvero tutti gli errori consecutivi non oltre gli r *bit* e quindi si può calcolare la probabilità di avere un burst più lungo di r che è esattamente $1 - 0,5^r$.
Sottolineiamo che CRC può rilevare qualsiasi numero dispari di errori nei bit.

7.4 Panoramica su collegamenti broadcast e Protocolli di Accesso Multiplo

Nell'introduzione al livello di collegamento abbiamo visto che esistono due tipi di collegamento di rete:

- Il collegamento punto a punto è costituito da un trasmittente e da un unico ricevente posti alle due estremità di un collegamento. I più famosi protocolli per questo tipo di collegamento sono **PPP (Point-to-Point Protocol** e **HDLC (High Level Data Link Control**.
- Il collegamento broadcast prevede più trasmittenti e più riceventi connessi allo stesso canale broadcast. Il termine broadcast indica il fatto che nel momento in cui un frame viene trasmesso da un mittente e ad ogni ricevente viene recapitata una copia del frame.

Prima di addentrasi nella trattazione dei protocolli a livello di collegamento bisogna soffermarsi sul problema di coordinazione dell'accesso di più nodi trasmittenti, denominato **Problema dell'Accesso Multiplo**.
Nelle reti per risolvere questo problema si utilizzano i Protocolli di Accesso Multiplo che decretano le modalità con cui i nodi regolano le trasmissioni sul canale condiviso. La situazione che cerchiamo di sbrogliare è la seguente: se ogni nodo ha la possibilità di trasmettere sul medesimo canale condiviso allora possono eseguire la trasmissione nello stesso momento e di conseguenza i nodi riceventi recepiranno i frame contemporaneamente. In questa casistica si generano **COLLISIONI** per cui nessun nodo ricevente riuscirà ad interpretare i frame poichè risulteranno ingarbugliati tra loro. Da ciò si hanno conseguenze come la perdita di frame mentre il canale rimane inutilizzato, inoltre se questa situazione si verifica con una frequenza elevata la maggior parte della

banda del canale verrebbe sprecata.
Descritto il problema enunciamo le soluzioni: i protocolli di accesso multiplo, che possono essere categorizzate in tre famiglie:

- PROTOCOLLI A SUDDIVISIONE DEL CANALE
  Channel Partitioning Protocol
- PROTOCOLLI AD ACCESSO CASUALE
  Random Access Protocol
- PROTOCOLLI A ROTAZIONE
  Taking-Turn Protocol

Date le famiglie idealizziamo gli specifici protocolli, con alcune proprietà. Definiamo un canale broadcast con velocità R *bit* al secondo:

- Il nodo che deve inviare dati ha un throughput pari a R *bps*
- Se abbiamo M nodi che devono trasmettere ognuno deve disporre pari a R/M *bps*. Non in maniera assoluta, ovvero R/M *bps* non è sempre la velocità del nodo istante per istante ma in un dato intervalli di tempo.
- Il protocollo è **DECENTRALIZZATO**, ovvero non abbiamo nodi che fungono da **Single Point Of Failure**
- Il protocollo è semplice ed economico da implementare

7.4.1 Protocolli a Suddivisione del Canale

All'inizio del nostro percorso nel mondo delle reti di calcolatori ci siamo imbattuti nel multiplexing a divisione di tempo, **TDM**, e a divisione di frequenza, **FDM**, sappiamo allora che queste due tecniche ci consentono di suddividere la larghezza di banda del canale di broadcast fra i nodi che lo condividono. Nello specifico sappiamo come, avendo un collegamento con velocità di trasmissione R *bps* e che supporta N nodi, TDM suddivide il tempo in **Intervalli Temporali** e ognuno viene nuovamente nuovamente suddiviso in N **Slot Temporali**. ogni slot viene assegnato a uno degli N nodi che può trasmettere i bit del pacchetto solo nel suo slot temporale. Gli slot generalmente vengono dimensionati per la trasmissione di un singolo pacchetto. TDM quindi è imparziale poichè ogni nodo ottiene, ad ogni suo slot, un tasso trasmissiovo pari a R/N *bps*. Il problema di TDM è duplice:

> Se uno solo dei nodi deve trasmettere, deve comunque attendere il proprio slot temporale ed inoltre abbiamo un tasso trasmissivo che si fissa sempre R/N *bps* anche se un solo nodo sta trasmettendo.

Notiamo allora che TDM non è la migliore scelta per quanto riguarda un protocollo di accesso multiplo.
FDM dal canto suo suddivide il canale in N canali da R/N *bps* e come TDM evita le collisione e divide la larghezza di banda in maniera sempre equa, proprio per questo abbiamo lo stesso problema: se un solo nodo deve trasmettere sarà limitato a R/N *bps*. Sapendo questo possiamo vedere un terzo protocollo di suddivisione del canale: **Accesso Multiplo a Divisione di Codice - CDMA**.
Praticamente il protocollo assegna un codice ad ogni nodo che è univoco per codificare i dati. Se i codici sono generati accuratamente, CDMA ha una proprietà non da poco: *i nodi potranno inviare simultaneamente i dati lungo il collegamento ai vari riceventi senza problemi di interpretazione derivanti dalle interferenza degli altri dati inviati dagli altri nodi.*

7.4.2 Protocolli ad Accesso Casuale

Questa famiglia di protocolli per l'accesso multiplo concede ai nodi la possibilità di trasmettere alla massima velocità del canale, R *bps*. Le collisioni semplicemente non vengono gestite, se si verificano il nodo ritrasmette ripetutamente i frame finché non vengono ricevuti correttamente dal destinatario. Le ritrasmissioni avvengono dopo un periodo di tempo casuale, **Random Delay**, ogni nodo che è coinvolto in una collisione seleziona in maniera casuale un ritardo indipendente da quello degli altri nodi. Questi ritardi consentono di evitare ulteriori collisioni proprio perché questi ritardi sono casuali. Vediamo ora quattro protocolli di questa famiglia:

- **SLOTTED ALOHA**
- **ALOHA**
- **CSMA**
- **CSMA/CD**

Slotted ALOHA

Assumiamo ora che:

- Tutti i frame hanno dimensione di esattamente L *bit*
- Il tempo è suddiviso in slot di L/R *secondi*
- La trasmissione dei frame inizia solo all'inizio di ogni slot
- I nodi sono sincronizzati in maniera che tutti sappiamo quando iniziano gli slot
- Se in uno slot due o più frame collidono tutti i nodi lo rilevano prima del termine dello slot
- Definiamo con p una probabilità che determiniamo con un numero compreso tra 0 e 1

Iniziamo allora ad analizzare **Slotted ALOHA** che è uno dei protocolli più semplici della famiglia ad accesso casuale.
Definiamo le operazioni di Slotted ALOHA:

- Un nodo che deve trasmettere un frame attende fino all'inizio dello slot successivo e poi effettuare l'invio
- Se non si verifica una collisione non occorre effettuare una ritrasmissione, l'operazione ha avuto successo e il nodo può predisporre il nuovo frame per l'invio successivo
- Se si verifica una collisione il nodo la rileva prima del termina dello slot e effettua una ritrasmissione con probabilità p durante i successivi slot finché l'operazione non ha successo

Questa ultima operazione significa che dopo una collisione rilevata nello slot corrente, si attenderà il prossimo dove vi sarà da fare una *scelta* tra ritrasmettere oppure no, l'evento di ritrasmissione verrà eseguito con probabilità $(1 - p)$. Come se il nodo lanciasse una moneta truccata in cui testa e croce non hanno probabilità entrambe del 50% ma una p e l'altra $(1 - p)$.
Ogni nodo lancia la propria moneta in maniera **INDIPENDENTE** da tutti gli altri.
Delineato come Slotted ALOHA effettivamente opera, vediamo i vantaggi che il protocollo offre:

- Consente a un singolo nodo di trasmettere continuamente frame alla massima velocità del canale quando un solo nodo è attivo
- È fortemente decentralizzato
- I nodi sono tra loro sincronizzati ma la rilevazione delle collisioni e la decisione di ritrasmettere sono operazioni eseguite indipendentemente

Ciò che bisogna comprendere ora è quanto il protocollo sia efficiente dipendentemente da dalla quantità di nodi sono attivi.
Possiamo evidenziare due problemi: alcuni slot saranno luogo di collisioni e di conseguenza sprecati, inoltre altrettanti slot rimarranno vuoti per quanto detto sulla probabilità di ritrasmissione.
Alla fine avremo gli slot dove un solo nodo è attivo, cioè quelli non sprecati o vuoti, che denominiamo **Slot RIUSCITO**.
Definiamo l'efficienza del protocollo come la frazione degli slot riusciti in presenza di un elevato numero di nodi che hanno una alta frequenza di frame da inviare. La base di partenza è lo zero perché non vi sarebbe efficienza nel caso in cui il protocollo non avesse alcuna forma di controllo. Per calcolare meglio l'efficienza supponiamo che il protocollo stabilisca che la probabilità p sia uguale per tutti i nodi. Assumiamo di avere N nodi, quindi al verificarsi di una collisione il successo è determinato dal fatto che allo slot successivo un solo nodo abbia la probabilità vincente mentre $N-1$ nodi siano nella probabilità $(1-p)^{N-1}$. Il nodo vincente ha probabilità $p(1-P)^{N-1}$ se i nodi attivi sono N abbiamo che la probabilità diventa $Np(1-P)^{N-1}$. Quindi dobbiamo trovare p in maniera tale che l'efficienza sia massima. Grazie a qualcuno questi calcoli sono già stati fatti e si è visto che l'efficienza è pari a $\frac{1}{e} = 0,37$ quindi solo il 37% degli slot sono riusciti e la velocità trasmissiva è di $0,37$ R bps anziché R. Con Slotted ALOHA un collegamento da 100 $Mbps$ trasmette in realtà a meno di 37 $Mbps$.

ALOHA

ALOHA è il predecessore di Slotted ALOHA, ancora in uso, che è completamente decentralizzato e privo di slot.
In ALOHA appena arriva un frame il nodo lo trasmette immediatamente e integralmente, se si verifica una collisione la ritrasmissione avviene istantaneamente con probabilità p. Se la probabilità decreta che non

debba essere ritrasmesso, il nodo attenderà un intervallo pari al tempo necessario ad inviare il frame, scaduto l'intervalli si ripeterà la scelta con probabilità.
Per capire l'efficienza di ALOHA sappiamo che in ogni istante la probabilità che stia trasmettendo è p, se allora la trasmissione inizia nell'istante t_0, nell'intervallo $(t_0 - 1, t_0]$ nessun nodo deve poter trasmettere affinché il frame sia inviato con successo, perché in caso contrario la nuova trasmissione si sovrappone al frame inviato a partire a t_0.
L'evento in cui nessun altro nodo trasmetta nell'intervallo $(t_0 - 1, t_0]$ è di $(1 - p)^{N-1}$, allora possiamo dire che la probabilità che un nodo trasmetta con successo è di $p(1 - P)^{2(N-1)}$. Eseguendo i calcoli che ha fatto qualcun altro (grazie chiunque tu sia) capiamo che l'efficienza di ALOHA è di $\frac{1}{2e}$ ovvero esattamente la metà di Slotted ALOHA che si traduce come il costo della sua completa decentralizzazione.

Accesso Multiplo con Rilevamento della Portante - CSMA

Alla base di **CSMA** vi sono due regole che rispettano i "protocolli" che usiamo noi esseri umani:

- **Ascoltare prima di parlare:** Nelle reti si chiama **Rilevamento della Portante**, ovvero un nodo ascolta il canale prima di trasmettere. Se il nodo si accorge che il canale è occupato aspetta finché non si libera prima di trasmettere.
- **Se qualcuno inizia a parlare insieme a voi, smettete di parlare:** Nelle reti si chiama **Rilevamento della Collisione**, durante la trasmissione il nodo rimane in ascolto del canale. Se si rende conto che un nodo inizia a trasmettere termina la propria trasmissione.

Da queste regole si sono derivate **CSMA** e **CSMA/CD**, e varie loro versioni. Poniamo l'esempio in cui un nodo A inizia la trasmissione avendo notato che il canale è libero, nel tempo che la trasmissione avendo notato che il canale è libero, nel tempo che la trasmissione di A si propaga un nodo B non riesce a rilevare che il canale non è libero e come da protocollo inizia a trasmettere, in un istante però si rileva la collisione. Questo ci fa capire come il ritardo di propagazione è di importanza capitale nei canali broadcast, se questo ritardo è grande la possibilità di avere collisioni aumenta.

CSMA/CD - Con Rilevamento delle Collisioni

Come dicevamo al rilevamento della collisione cessa immediatamente la trasmissione. Sappiamo che attenderà un intervalli di tempo per ritrasmettere, ma se la ritrasmissione avviene per entrambi i nodi dopo la stessa quantità di tempo la collisione avviene nuovamente. Il quantitativo di tempo si chiama **TEMPO DI BACKOFF**.
Se l'intervalli è grande e il numero di nodi per cui avviene una collisione è piccolo l'intervallo di attesa prima di ritrasmettere è ampio, se però il numero di nodi che collidono è grande e l'intervallo è piccolo è molto probabile che i valori di attesa casuale siano molto simili e quindi si ritornerà ad una situazione di collisione. Per risolvere questo problema si una l'algoritmo di **ATTESA BINARIA ESPONENZIALE - Binary Exponential Backoff**, quando un nodo riscontra l'$n-esima$ collisione sceglie un valore casuale in un insieme $K = \{0, 1, ..., 2^n - 1\}$, all'aumentare delle collisioni accresce l'insieme esponenzialmente. Quindi la cardinalità dell'insieme cresce esponenzialmente con il crescere delle collisioni, nello specifico raddoppia il suo valore massimo per ogni collisione. In tal modo è molto probabile che la trasmissione di un nuovo frame abbia immediatamente successo mentre gli altri nodi rimangono in attesa esponenziale.

7.4.3 Protocolli a Rotazione

Finora abbiamo visto ALOHA e CSMA che non riescono ad avere un throughput vicino a R/N *bps* con N numero di nodi attivi.I ricercatori hanno quindi sviluppato la famiglia di **PROTOCOLLI A ROTAZIONE**, in questa famiglia esistono molti protocolli, ognuno con decine di varianti, dato che questa trattazione è basilare nel mondo delle reti, vedremo solo due dei più importanti protocolli.

Protocollo Polling

Si determina un nodo principale che interpella tutti gli altri nodi.
Quindi il nodo principale invia un messaggio al nodo 1 che può trasmettere un dato numero massimo di frame. Alla fine della trasmissione il nodo principale invia un messaggio al nodo 2 che inizia a sua volta la propria trasmissione.
Il nodo principale riesce a capire se il nodo ha terminato la trasmissione

ascoltando il canale e osservando la mancanza di segnale. Questo protocollo elimina collisioni e slot vuoti e di conseguenza ha una efficienza elevata. Comunque abbiamo alcuni svantaggi, per esempio abbiamo il ritardo dovuto alle notifiche per avviare le trasmissioni (il **POLLING**), inoltre il problema peggiore è il guasto ad un nodo, se accade il collegamento è inattivo.

Protocollo Token-Passing

In questo caso il nodo principale non esiste ma esiste un frame detto **TOKEN** che circola tra i nodi seguendo un ordine prefissato. Questo consente al nodo che detiene il token di poter trasmettere e soprattutto se il nodo non ha frame da trasmettere può passarlo immediatamente al nodo successivo. Questo protocollo è altamente efficiente e completamente decentralizzato, il problema però è sempre che se un nodo si guasta cade l'intero sistema, inoltre se un nodo non riesce ad inviare il frame bisogna avviare un protocollo di recupero che lo rimette in circolo e tutto questo comporta tempo sprecato.

7.4.4 Reti HFC e il Protocollo DOCSIS

DOCSIS - Data Over Cable Service Interface Specification, è la specifica dell'architettura della rete cablata HFC, possiamo dire che è un insieme di protocolli delle famiglie che abbiamo appena visto, in poche parole è un esempio perfetto per riassumere tutto quello che abbiamo visto finora.
In una rete HFC sappiamo che il collegamento è suddiviso in canale di *downstream* e il canale di *upstream* utilizzando il protocollo *FDM*, il canale di downstream ha una frequenza di 6 MHz con throughput di circa 40 $Mbps$, upstream invece ha una frequenza di $6,4$ MHz con throughput di circa 30 $Mbps$. Entrambi i canali di downstream e upstream sono broadcast infatti nel canale downstream dal CMTS vengono ricevuti da tutti i modem, non ci sono problemi di accesso multipli poiché CMTS è un unico punto. Diversamente il canale di upstream è più interessante perchè i modem invece condividono il canale e vogliono tutti parlare con il CMTS, quindi avvengono collisioni.
Per questo il canale di upstream viene suddiviso con protocollo TDM in *mini-slot*, il CMTS invia un messaggio **MAP** tramite il canale di downstream, tramite questo messaggio indica al modem che può tra-

smettere nell'intervallo di tempo che specifica il CMTS nel messaggio. Quindi ogni mini-slot è proprio del modem quindi il CMTS può assicurare che non ci siano le collisioni durante il mini-slot. Dobbiamo fare attenzione però perché il canale di downstream è comunque broadcast quindi i messaggi possono collidere.
Un modem cablato non può sentire se un canale è occupato oppure se si sono verificate collisioni. Allora il modem assume che si è verificata una collisione se non riceve un messaggio di allocazione nel successivo frame, in questo caso entra in campo l'algoritmo di attesa binaria esponenziale. In caso di basso traffico gli slot assegnati per la richiesta di slot possono essere usati per la trasmissione. Di conseguenza una rete di accesso HFC è effettivamente un paniere di protocolli utilizzati.

7.5 Reti Locali Commutate

Bisogna ora prestare la nostra attenzione su come funzionano le reti locali commutate. In queste reti non vi sono dispositivi che operano a livello di rete, vi sono solo gli **SWITCH** che operano a livello di collegamento quindi non usano algoritmi di instradamento per determinare i percorsi attraverso la rete ma un altro metodo. Il nostro studio sulle reti commutate inizierà dalle LAN commutate, poi passeremo al protocollo Ethernet, vedremo come operano gli switch e come gli switch sono utilizzati su LAN a larga scala.

7.5.1 Indirizzi a Livello di Collegamento e Protocollo ARP

Sebbene host e router operino anche a livello di rete e quindi detengono indirizzi IP hanno anche indirizzi a livello di collegamento, in questa sezione cerchiamo di capire perché questo aspetto è importante e perché entrambi gli indirizzi sono fondamentali per il corretto funzionamento di Internet, inoltre ci approcceremo a **ARP - Address Resolution Protocol** che risolve indirizzi IP in indirizzi a livello di collegamento.

7.5.2 Indirizzi MAC

Gli indirizzi a livello di collegamento sono gli adattatori a detenerli, cioè le schede di rete, non gli host o i router, quindi più interfacce si hanno sullo stesso dispositivo più indirizzi a livello di collegamento detiene. É curioso evidenziare come gli switch non hanno indirizzi a livello di

collegamento poiché hanno il solo scopo di trasportare datagrammi tra host e router e lo esegue in maniera trasparente. Nella terminologia della rete gli indirizzi a livello di collegamento vengono chiamati: indirizzi fisici, indirizzi LAN o anche indirizzi MAC, sono generalmente espressi in notazione esadecimale e sono lunghi 6 *byte*, di conseguenza si hanno 2^48 indirizzi disponibili. L'indirizzo MAC solitamente è permanente, ultimamente però di è sviluppata la possibilità di modificarlo via software. Dato che sono limitati e perlopiù permanenti si ha che non esistono due schede con lo stesso indirizzo MAC, questo è possibile grazie all'IEEE a cui la società di costruzione delle schede si rivolge per acquistare blocchi di indirizzi MAC. I blocchi sono da 2^24 indirizzi con i primi 24 *bit* dell'indirizzo fissati. Quando una scheda di rete invia un frame all'interno pone l'indirizzo MAC di destinazione e poi lo inoltra nella LAN.
Uno switch può anche fare broadcasting inviando tramite tutte le interfacce il proprio frame. Ogni scheda controlla se l'indirizzo MAC è il proprio e in caso affermativo lo gestisce altrimenti lo scarta. Se invece una scheda di rete vuole inviare in broadcasting un frame allora l'indirizzo MAC sarà composto da 48 *bit* a 1, ovvero **FF-FF-FF-FF-FF-FF**.

7.5.3 Protocollo ARP per la risoluzione di indirizzi

Siamo arrivati al punto di avere due tipologie di indirizzi, quello a livello di rete (indirizzi IP) e quello a livello di collegamento (indirizzi MAC). Questo in una sottorete pone il problema su come un nodo che deve inviare un frame che incapsula l'indirizzo MAC del destinatario possa conoscere questo indirizzo MAC. Ci viene in aiuto **ARP - Address Resolution Protocol**. Nell'esempio che segue assumiamo che lo switch della nostra semplice sottorete invii i frame sempre in broadcast, vedremo più avanti come gli switch operano veramente.
Il nodo trasmittente deve inviare un datagramma a un nodo nella stessa sottorete, il datagramma viene incapsulato in un frame che deve avere il MAC di destinazione, quindi ciò che ARP ci consente è quello di risolvere l'indirizzo IP di destinazione nell'indirizzo MAC di destinazione. Simile a quello che fa DNS ma è importante sottolineare la differenza principale: *ARP opera solo nelle sottoreti quindi restituisce errore nel momento in cui si cerca di conoscere un indirizzo MAC di un indirizzo IP fuori dalla sottorete in cui si opera.*
Il funzionamento di ARP è presto detto: ogni nodo immagazzina in

RAM una **TABELLA ARP**, questa tabella possiede tre colonne: **INDIRIZZO IP, INDIRIZZO MAC, TTL**. Questa tabella è comoda se l'indirizzo IP è presente perché si recupera l'indirizzo MAC molto velocemente, una situazione più interessante è quella dove la tabella in RAM non ha l'entry dedicata all'indirizzo della destinazione che si vuole raggiungere.
Il nodo trasmittente crea un pacchetto ARP (un pacchetto speciale) dove pone il priprio indirizzo MAC e il proprio indirizzo IP, inoltre immette l'indirizzo IP del destinatario. Fatto questo chiede alla propria scheda di rete di inviare in broadcast il pacchetto, in questo modo tutti i nodi della sottorete riceveranno il pacchetto ma solo il nodo il cui indirizzo IP corrisponde a quello cercato, se esiste, risponderò con il proprio indirizzo MAC incapsulato in un **ARP Reply**. É curioso notare che ARP funziona con una richiesta broadcast ma con una risposta in un frame standard. La particolarità e la comodità di ARP è che non richiede all'amministratore di configurare nulla poiché si costruisce automaticamente, se un nodo riceve un ARP Reply il nuovo indirizzo MAC viene salvato nella tabella in RAM e posto un TTL di 20 minuti, in maniera tale che se un nodo si scollega dalla rete allora prima o poi verrà cancellato il proprio IP e MAC dalla tabella.

7.5.4 Inviare un Datagramma FUORI dalla Sottorete

Per comprendere al meglio questo passaggio premettiamo uno schema di una rete composta da un router che collega due sottoreti:

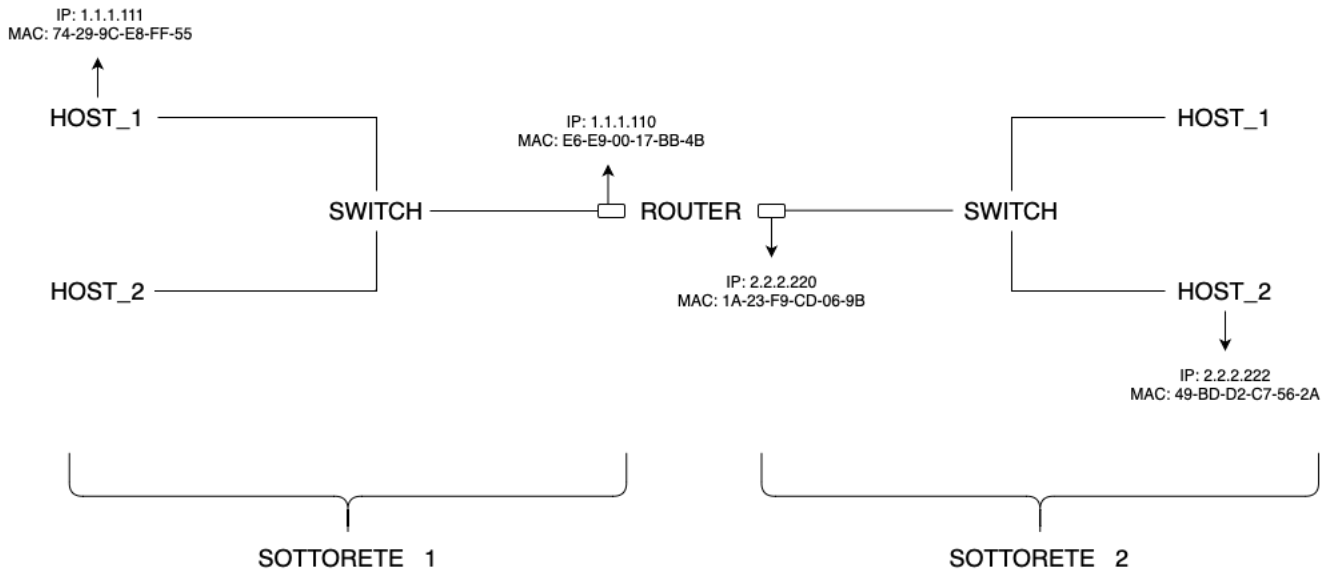


Figura 7.2: Schematizzazione di due sottoreti comunicanti tramite un router

Poniamo il caso che l'**Host 1** della *Sottorete 1* voglia inviare un datagramma all'**Host 2** della *Sottorete 2*.
Quindi quello che succede è che l'**Host 1** crea un frame dove incapsula il datagramma e imposta l'indirizzo MAC di destinazione con l'indirizzo MAC dell'interfaccia del router con indirizzo IP: `1.1.1.110`, e lo invia nella sottorete.
Il frame raggiunge il router, che come sappiamo lavora anche a livello di rete, quindi spacchetta il frame e nota che l'indirizzo IP di destinazione si trova nella *Sottorete 2*, quindi incapsula nuovamente il datagramma in un frame dove l'indirizzo MAC di destinazione è quello dell'**Host 2** nella *Sottorete 2* e lo passa all'interfaccia con indirizzo IP: `2.2.2.220` che lo inoltra nella corretta sottorete. Tutta la risoluzione degli indirizzi MAC in questo esempio viene sempre eseguita come spiegato precedentemente.

7.6 Ethernet

Si può definire **Ethernet** come una tecnologia, o una famiglia di tecnologie, che ha conquistato le LAN cablate. Amministratori di rete

sono riluttanti dal cambiare in favore di tecnologie diverse perché più complesse e costose. Inoltre sebbene alcune tecnologie avessero un tasso trasmissivo più ampio rispetto a ethernet, quest'ultimo si è sempre evoluto portando rivoluzioni sempre più importanti fino ad arrivare ad essere una *Ethernet Commutata* che alza i tassi trasmissivi e rende i componenti hardware molto economici. Questa evoluzione è iniziata negli anni '90 quando si utilizzava una topologia di rete locale a **stella** per mezzo di un **Hub**, dato che ethernet è sempre una tipologia di connessione broadcast. L'Hub è un dispositivo a livello fisico che agisce sui singoli bit "amplificandoli".
All'inizio degli anni 2000 si è passati dall'utilizzo di Hub all'utilizzo di switch che diversamente dall'Hub è priva di collisioni ed inoltre è un dispositivo che opera a livello 2, quello di collegamento.

7.6.1 Frame Ethernet

Prima di vedere nel dettaglio i campi che costituiscono il frame ethernet sottolineiamo che:

> Il payload di un frame ethernet è solitamente un datagramma IP ma può anche incapsulare altre tipologie di pacchetti a livello di rete. Inoltre ethernet è un servizio **senza** connessione, questo significa che una scheda di rete immette nella LAN i frame senza alcuna forma di handshake preventivo con il destinatario. Infatti quando il destinatario calcola il CRC e trova un errore non invia nessun ACK o altre forme di notifica al trasmittente, semplicemente scarta il frame. Infatti nel caso di utilizzo del protocollo UDP il ricevente può avere lacune nei dati che riceve, d'altra parte con TCP questo problema non sussiste come abbiamo visto ma di questo, ovvero che i dati possano essere nuovi o ritrasmessi, ethernet non è a conoscenza.

Ora vediamo la struttura di un frame:

Figura 7.3: Schematizzazione di un frame Ethernet

- **PREAMBOLO:** Il frame inizia con il *Preambolo* che sono `8 byte` di cui 7 sono impostati `10101010` mentre l'ottavo è `10101011`. I primi sette byte servono per risvegliare la scheda di rete del ricevente e per sincronizzare il proprio clock con quello del trasmittente. Per quanto gli standard di trasmissione sia deinito a `10 Mbps`, `100 Mbps` o `1 Gbps` per trasmittente, possiamo dire che ci sarà sempre una variazione che non si può sapere a priori. Infine l'ultimo byte serve a far capire dove finisce il preambolo e dove inizia il resto del frame.
- **INDIRIZZO DI DESTINAZIONE:** Campo di `6 byte` che contiene l'indirizzo MAC della destinazione, che può essere anche l'indirizzo MAC broadcast.
- **INDIRIZZO SORGENTE:** Campo di `6 byte` che contiene l'indirizzo MAC della sorgente.
- **TIPO:** Questo campo di `2 byte` consente ad Ethernet di supportare vari protocolli di rete oltre ad IP. Per esempio il campo prende il valore esadecimale `0806` per il protocollo ARP. Questo campo si ricollega al campo protocollo nel datagramma e il campo porta nei segmenti. Così si nota la comunicazione tra un livello e un altro.
- **CAMPO DATI:** Campo da `46` a `1500 byte`. MTU di ethernet è infatti `1500 byte`. Solitamente questo campo è riempito dal datagramma IP. Se il datagramma è compreso tra `46` e `1500 byte` bisogna riempire il campo fino al massimo.
- **CRC:** Campo da `4 byte` che consente alla scheda di rete ricevente di rilevare bit di errore nel frame.

7.7 Switch a Livello di Collegamento

Gli switch sono dispositivi che come ormai sappiamo operano al livello di collegamento e hanno lo scopo di filtrare e inoltrare pacchetti di livello 2, lavorano essendo trasparenti ai nodi che li usano, ciò significa che il nodo non è a conoscenza che lo switch riceverà, filtrerà e inoltrerà il frame, ma si occupa solo di impostare la destinazione finale. Il tasso di ricezione delle interfacce dello switch può accedere la capacità del collegamento perché le interfacce sono dotate di buffer.

7.7.1 Inoltro e Filtraggio

L'inoltro consiste nell'individuazione dell'interfaccia verso cui il frame deve essere diretto e nell'inviarlo a quell'interfaccia. Il filtraggio invece è la funzionalità dello switch che determina se un frame debba essere inoltrato a qualche interfaccia o scartato.
Queste due funzionalità vengono eseguite mediante una **TABELLA DI COMMUTAZIONE - SWITCH TABLE** composta da:

- INDIRIZZO MAC del nodo
- Interfaccia dello switch che conduce al nodo
- Il momento in cui la voce (entry) per quel nodo è stata inserita nella tabella

Vediamo un caso specifico per vedere le casistiche che genera lo switch. Ipotizziamo uno switch che riceve sull'interfaccia `x` un frame con un indirizzo MAC specifico.
Ora analizziamo tre scenari possibili:

1. Non ci sono voci all'interno della tabella che riportano l'indirizzo MAC di destinazione. In questo caso lo switch manda il frame tutte le interfacce in broadcast tranne che sull'interfaccia `x`.
2. Esiste una voce in tabella che associa l'indirizzo MAC all'interfaccia `x` da dove l'indirizzo è stato ricevuto e in questo caso lo switch scarta (filtra) il frame.
3. Esiste una voce in tabella che associa l'indirizzo MAC all'interfaccia diversa da cui è stato ricevuto e in questo caso lo switch pone il frame nel buffer dell'interfaccia dedicata all'inoltro su quel collegamento.

7.7.2 Autoapprendimento

Uno switch costruisce automaticamente, dinamicamente e in maniera autonoma le proprie tabelle. Questo autoapprendimento avviene come segue:

1. La tabella è inizialmente vuota
2. Di ogni frame che riceve lo switch archivia nella sua tabella:

- L'indirizzo MAC sorgente
- L'interfaccia da cui arriva il frame
- Il momento in cui arriva (timestamp)

3. Dopo un periodo di tempo, detto **aging time**, che lo switch non riceve frame da un determinato indirizzo sorgente, lo cancella dalla tabella. In questo modo se un calcolatore viene sostituito, l'indirizzo MAC del precedente calcolatore viene eliminato automaticamente.

Gli switch sono dispositivi **PLUG-AND-PLAY**, in quanto non richiedono interventi dell'amministratore di rete o dell'utente. Basta collegare i segmenti di LAN alle sue interfacce senza dover configurare nulla al momento dell'installazione oppure all'eliminazione di un host da un segmento LAN.

7.8 Proprietà della commutazione a Livello 2

Possiamo identificare alcuni vantaggi dall'utilizzo degli switch piuttosto che il semplice collegamento broadcast. Questi vantaggi si traducono nelle proprietà degli switch, le vediamo qui di seguito:

- **ELIMINAZIONE DELLE COLLISIONI:** Gli switch forniscono un aumento delle prestazioni su LAN poiché riescono ad eliminare le collisioni, non vi è quindi spreco di banda. Infatti gli switch inseriscono i frame all'interno dei buffer non trasmettendo più di un frame su ogni segmento di LAN, in un dato istante. Il massimo throughput di uno switch è la somma dei tassi trasmissivi di tutte le sue interfacce.
- **COLLEGAMENTI ETEROGENEI:** Una proprietà molto importante è che gli switch possono utilizzare diverse tipologie di collegamento, sia per quanto riguarda velocità di trasmissione, sia per quanto riguarda il mezzo trasmissivo in sé. Questo permette di poter cambiare nuovi dispositivi con quelli già installati.
- **GESTIONE:** Gli switch facilitano la gestione di rete, se c'è un malfunzionamento, che sia un collegamento interrotto o un dispositivo che inonda il collegamento di frame, gli switch rilevano il

malfunzionamento e interrompono la connessione. Queste operazioni vengono poi incapsulate in informazioni e messe a disposizione dell'amministratore di rete. L'amministratore in questa maniera può rilevare gli errori velocemente, riesce a correggerlo e a determinare come la LAN dovrà evolvere nel corso del tempo.

7.9 LAN Virtuali - VLAN

Ora sappiamo che ogni gruppo di macchine (dipendentemente dallo scopo, dal tema, dalla appartenenza) costituisce una LAN commutata. Per esempio le macchine dell'università costituiscono una LAN ma poi abbiamo i vari dipartimenti che hanno il proprio gruppo di macchine e quindi c'è la necessità di distinguere i gruppi appartenenti ai vari dipartimenti. Allora si frutta una struttura gerarchica di switch per separare i vari gruppi, gli switch di ogni gruppo si connette poi a uno (o più) switch di più "alto livello" che a sua volta si connette al router e ai vari server dell'università. Questo permette di localizzare le varie LAN commutate dei vari dipartimenti. Bene, questa configurazione funziona idealmente in maniera efficiente, il problema è che in maniera reale invece comporta problematiche grosse:

- **MANCANZA DI ISOLAMENTO DEL TRAFFICO:** Si può pensare che la localizzazione delle LAN possa anche isolare il traffico. In realtà comunque il traffico deve attraversare la rete istituzionale (lo switch di alto livello), questo però comprometto il traffico su sicurezza e riservatezza.
- **USO EFFICIENTE DELLO SWITCH:** Per ogni gruppo di host si necessita di uno switch, però se il gruppo è piccolo si spreca il singolo switch.
- **GESTIONE DEL TRAFFICO:** Se uno stesso dispositivo (meglio un utente) appartiene a più di un gruppo dovrebbe cambiare la posatura della rete ogni volta che si muove.

Per risolvere questi problemi e questo sconveniente modo di configurare una rete si può utilizzare uno switch che supporta l'utilizzo di una **Virtual Local Area Netwotk - VLAN**. Una VLAN permette di definire più reti locali virtuali mediante una singola infrastruttura

fisica di rete locale. Quello che fisicamente avviene in una configurazione VLAN è che le porte dello switch (interfacce) vengono raggruppate per formare una LAN, quindi per esempio se abbiamo uno switch da `96` porte e tre gruppi di host, avremo il gruppo 1 più piccolo a cui vengono dedicate dalla porta `2` fino alla `10`, il gruppo 2 dalla porta `11` alla porta `50` e dalla `51` alla `96` vengono riservate al gruppo 3, in questo modo ogni gruppo di porte riservare formerà un dominio broadcast, i frame della singola VLAN sono isolati effettivamente. La configurazione viene fatta mediante un software di gestione dello switch che si occupa di mantenere una tabella di associazione tra porte e l'hardware si limita a consegnare il frame tra le porte della stessa VLAN.
Questo però ancora non risolve il problema della gestione del traffico. Il problema però è risolvibile in maniera semplice: lo switch integra funzioni di router. Quindi ciò che avviene in pratica è che se le due (o più) VLAN devono comunicare possono farlo senza problemi poiché la VLAN passerà il frame al settore dello switch che si occupa del routing verso un'altra VLAN.
Sorge ora un problema: supponiamo di avere un primo edificio dove vi è uno switch che mette a disposizione due VLAN per due dipartimenti e un secondo edificio con uno switch che mette a disposizione due VLAN per gli stessi dipartimenti, come dovrebbero comunicare?
Un approccio scalabile è il **VLAN TRUNKING**, in questo caso si ha una porta speciale per ogni switch e per ogni switch la propria porta speciale appartiene ad entrambe le VLAN, ogni frame di ogni VLAN viene inoltrato in questa porta speciale. Bisogna comprendere ora come lo switch ricevente possa capire a quale VLAN appartiene il frame ricevuto. IEEE mette a disposizione una etichetta VLAN di `4 byte` che si suddividono in due campo:

- **TPID - TAG PROTOCOL IDENTIFIER** di `2 byte`
- **TAG CONTROL INFORMATION** di `2 byte`

Ci si è concentrati su VLAN basate su porte ma si possono configurare VLAN basate su indirizzi MAC o addirittura VLAN estese attraverso router IP e quindi basate sui protocolli a livello di rete.

7.10 MultiProtocol Label Switching - MPLS

Per iniziare diamo una generica definizione per capire di cosa stiamo parlando:

> **MPLS** è una tecnologia che permette di instradare flussi di traffico multiprotocollo da un nodo origine ad un nodo destinazione tramite l'utilizzo di identificativi (label) tramite coppie di router adiacenti oppure per mezzo di operazioni sulle label.

MPLS, o meglio una rete MPLS, assegna le proprie etichette ai pacchetti. Queste etichette consentono di prendere decisioni di inoltro sui pacchetti basate esclusivamente sulle etichette stesse, eliminando la necessità di esaminare il pacchetto. Di conseguenza elimina il problema di dipendenza da una particolare tecnologia del livello di collegamento. Infatti si vengono a creare circuiti end-to-end su un tipo qualsiasi di mezzo di trasporto, utilizzando qualsiasi protocollo.
MPLS opera a un livello che solitamente viene considerato nel mezzo tra il livello di rete e quello di collegamento, viene per questo definito livello `2.5`.
MPLS consente di instaurare una connessione tra nodi adiacenti interni alla retei di trasporto, di pari passo a questa opzione abbiamo però la possibilità di effettuare instradamento classico basato su IP. Questo ci palesa che con un apparato unico (Router IP/MPLS, anziché router e switch) si hanno due tipologie di commutazione (a circuito e a pacchetto).
Per creare una connessione tra nodi interni, MPLS aggiunge una etichetta ai pacchetti IP, l'etichetta viene aggiunta tramite router MPLS/IP ai bordi e poi viene instradato tramite commutazione di etichetta. Questa tipologia di commutazione è basata su una tabella di associazione pacchetto/etichetta è più efficiente e veloce rispetto alle tabelle di routing classiche, di conseguenza si ha un throughput più elevato.
Per eseguire questa commutazione MPLS necessita di avere un percorso stabilito detto **LABEL SWITCHED PATH**, che deve essere valido e instaurabile. La variante MPLS-TE (Traffic Engineering) controlla inoltre che siano congrue le varie capacità tra nodo origine, destinazione e richiesta. MPLS determina una dimensione fissa in cui l'unità di informazione debba essere frammentata, i `48 byte` diventano `53 byte`

con l'header. Su MPLS si sono sviluppati una miriade di protocolli per sviluppare nuove tecnologie:

- MPLS-TE
- VPLS
- HVPLS
- VPWS
- EoMPLS
- VPN

7.11 Ricapitolone di fine STACK

Visione integrale della richiesta di una pagina web

Per ricapitolare tutto quello studiato finora, analizziamo un semplice esempio dove si richiede una pagina web. Vedremo tutti i passaggi necessari con tutti i vari protocolli necessari a soddisfare la richiesta in merito.
L'esempio prende in causa un client che si connette ad uno switch Ethernet della propria scuola e scarica la homepage di `www.google.com`

7.11.1 INIZIAMO (DHCP, UDP, IP e Ethernet)

Supponiamo che un computer venga collegato a un cavo Ethernet connesso allo switch Ethernet della scuola che a sua volta è connesso al router della scuola. Il router a sua volta è connesso a un ISP, in questo esempio è *Comcast.net*, che fornisce il servizio DNS alla scuola. Assumiamo inoltre che il server DHCP sia eseguito dal router. Come abbiamo appreso appena la macchina è connessa alla rete non può fare nulla poiché non possiede un indirizzo IP, quindi su parte da qui: si esegue il protocollo DHCP per ottenere un indirizzo IP e altre informazioni.

1. Il sistema operativo crea un **messaggio DHCP REQUEST** che viene incapsulato in un **segmento UDP** con porta di destinazione `67` e porta sorgente `68`. Il segmento viene poi incapsulato in un **datagramma IP** dove l'indirizzo di destinazione è

`255.255.255.255 - BROADCAST` e l'indirizzo sorgente è `0.0.0.0` perché la macchina ancora non ha un indirizzo IP.

2. Il datagramma IP viene incapsulato in un **frame Ethernet** a cui viene posto indirizzo MAC di destinazione `FF:FF:FF:FF:FF:FF (BROADCAST)`, in questa maniera lo switch inoltra il frame a tutti i dispositivi connessi allo switch. Il MAC sorgente invece è quello della macchina `00:16:D3:23:68:8A`.

3. Questo frame con la richiesta DHCP è il primo frame inviato dalla macchina allo switch ethernet. Lo switch lo inoltra in broadcast in maniera che venga inoltrato anche sulla porta che lo connette al router.

4. Il router riceve il frame Ethernet che contiene la richiesta DHCP sulla sua interfaccia con MAC `00:22:6B:45:1F:1B`, dal frame viene quindi estratto il datagramma IP. Il **payload**, del datagramma, che è il **segmento UDP**, viene preso in causa per effettuare **demultiplexing** verso UDP, di conseguenza viene estratto il messaggio **DHCP Request** dal segmento e passato al **server DHCP**.

5. Il server DHCP in esecuzione sul router, supponiamo, possa allocare indirizzo IP nel **blocco CIDR** `68.85.2.0/24`. Quindi supponiamo che il server DHCP assegni l'indirizzo `68.85.2.101`. Quindi il server DHCP crea un **messaggio DHCP ACK** contenente l'indirizzo IP scelto, l'indirizzo IP del **server DNS** (`68.87.71.226`), l'indirizzo IP del gateway di default, `68.85.2.1`, e la maschera di rete. Questo messaggio viene incapsulato in un segmento UDP che a sua volta viene incapsulato in un datagramma IP e a sua volta incapsulato in un frame Ethernet, con MAC sorgente, quello del router, `00:22:6B:45:1F:1B`, e con MAC destinazione quello della macchina `00:16:D3:23:68:8A`.

6. A questo punto il frame è pronto e viene inviato dal router allo switch, che è in grado di **autoapprendere**, quindi avendo ricevuto precedentemente il frame della macchina, conosce l'interfaccia giusta in cui inoltrare il frame del router per raggiungere il MAC di destinazione (`00:16:D3:23:68:8A`).

7. La macchina allora riceve il frame Ethernet contenente il DHCP ACK, si estrae il datagramma IP, si estrae il segmento UDP e infine estrae DHCP ACK. In questo momento il client DHCP memorizza l'indirizzo IP assegnatogli, quello del DNS e le altre informazioni. Nella **tabella di inoltro locale** la macchina salva l'indirizzo IP del gateway, così la macchina invierà tutti i datagrammi al di fuori della propria sottorete. Quindi ora siamo nella condizione di poter richiedere la nostra pagina web, o quasi!

7.11.2 CONTINUIAMO (DNS e ARP)

Come sappiamo per poter inviare la richiesta della pagina *www.google.com* deve conoscere l'indirizzo IP della macchina, quindi entra in gioco il **protocollo DNS**.

8. Il sistema operativo del computer allora crea un messaggio **DNS Query** inserendo la stringa *"www.google.com"* nella sezione riguardante la richiesta del messaggio DNS. Viene incapsulato quindi in un segmento UDP con porta di destinazione `53`, e a sua volta incapsulato in un datagramma IP con indirizzi IP di destinazione `68.87.71.226` (l'indirizzo del server DNS) e indirizzo IP sorgente `68.85.2.101`.

9. Ora il datagramma deve essere incapsulato in un frame, qui la nostra macchina si rende conto che pur conoscendo l'indirizzo IP del gateway non ne conosce l'indirizzo MAC necessario per il frame. Quindi utilizza il protocollo ARP.

10. La macchina crea allora una **ARP Query** con l'indirizzo IP di destinazione `68.85.2.1` (il gateway), il messaggio viene incapsulato in un frame con MAC di destinazione `FF:FF:FF:FF:FF:FF` e invia il frame Ethernet allo switch che consegna in broadcast, compreso il gateway.

11. Il router gateway riceve il frame con la **ARP Query** e scopre che l'indirizzo IP all'interno dell'interrogazione è proprio il suo, quindi crea una **ARP Reply** con all'interno il suo MAC `00:22:6B:45:1F:1B`. Incapsula l'ARP Reply in un frame Ethernet con MAC di destinazione quello della nostra macchina (`00:16:D3:23:68:8A`), il frame viene allora inviato allo switch che lo consegnerà alla nostra macchina.

12. La macchina riceve il frame e estrae l'indirizzo MAC del gateway

13. La macchina è finalmente in grado di eseguire una richiesta DNS. Questo viene fatto tramite un datagramma IP che un indirizzo IP di destinazione che è quello del server DNS mentre il frame ha come indirizzo MAC quello del gateway! Allora viene inviato il frame allo switch che lo consegna al gateway.

7.11.3 E ANCORA (Instradamento INTRA-DOMINIO al Server DNS)

14. Il gateway che riceve il frame estrae il datagramma IP e ricerca l'indirizzo IP di destinazione estrapolato, si basa poi sulla tabella di inoltro scoprendo (qui assumiamo sia così nel nostro esempio) che bisogna inviare il datagramma al router più a sinistra nella rete Comcast. Allora il datagramma viene incapsulato in un frame e viene trasmesso lungo l'interfaccia determinata.

15. Il router designato allora riceve il frame, estrae il datagramma IP, e determina l'indirizzo IP di destinazione quindi determina l'interfaccia sulla quale inoltrare il datagramma. Questo è determinato dalla tabella di inoltro costituita tramite **protocollo Intra-Dominio (RIP, OSPF o IS-IS)** e dal **protocollo Inter-Dominio: BGP**.

16. Alla fine di questo processo avremo il server DNS che riceve il datagramma IP con la richiesta DNS della nostra macchina. Il server allora estrapola la richiesta e ricerca il nome *www.google.com* all'interno del database DNS e trova il **Record di Risorsa DNS** che contiene l'indirizzo `64.233.168.105` che corrisponde al dominio cercato.
    Quindi stiamo assumendo che il valore cercato sia nella **Cache DNS** (sviluppata dal **server Autoritativo di Google**), se così non fosse il server DNS dovrebbe contattare il server autoritativo per conoscere il valore.
    Allora il server DNS scrive l'indirizzo IP cercato all'interno di un **DNS Reply**, lo pone in un segmento UDP che viene posto all'interno di un datagramma IP e spedito verso il router della scuola attraverso la rete Comcast e quindi allo switch per arrivare alla nostra macchina.

17. La nostra macchina allora estrae l'indirizzo IP del server corrispondente a *www.google.com* dal messaggio DNS.
    La nostra macchina è finalmente pronta, ha tutte le carte in regola per contattare *www.google.com* per la richiesta della pagina.

7.11.4 FINE FINALMENTE! (TCP e HTTP)

18. Il nostro calcolatore può creare la **Socket TCP**, prima di inviare la richiesta con un messaggio **HTTP GET**, effettua l'**Handshaking a tre vie**. Quindi la macchina crea un **segmento TCP SYN** con porta di destinazione `80` e lo incapsula in un datagramma IP con destinazione `64.233.169.105`, incapsulato a sua volta in un frame con MAC di destinazione posta a quello del gateway, infine lo invia allo switch.

19. Il router della scuola inoltra il datagramma nella rete Comcast e poi quella Google tramite i protocolli di intra e inter dominio detto precedentemente.

20. Il nostro datagramma alla fine giunge *www.google.com*, da qui si estrae il segmento TCP SYN eseguito il Demultiplexing verso la socket associata alla porta `80`.
    Si crea allora la socket per la connessione TCP tra il **server HTTP di Google** e la nostra macchina. Viene quindi generato un **segmento TCP SYN/ACK** incapsulato in un datagramma indirizzato al client sulla nostra macchina, incapsulato in un frame e spedito alla macchina.

21. Il frame attraversa la rete Google, Comcast e infine arriva alla scuola fino alla nostra macchina. Si effettua demultiplexing del datagramma verso la socket TCP che passa allo **stato "CONNESSIONE"**.

22. La nostra socket è pronta e quindi trasmette byte a *www.google.com*, il browser allora crea il **messaggio HTTP GET** contenente l'**URL** da leggere. Il messaggio si incapsula in un segmento TCP, poi in un datagramma e inviato a *www.google.com*

23. Il server HTTP di *www.google.com* legge il **messaggio HTTP** tramite la socket TCP e crea un **messaggio HTTP Risposta**

dove si ha come payload il contenuto della pagina web e lo invia alla socket TCP

24. Il datagramma contenente la risposta HTTP raggiunge tramite Google, Comcast, router della scuola, switch e infine interfaccia della nostra macchina. Il browser legge dalla socket la risposta HTTP (DE-capsulamento) estrae il codice HTML della pagina, lo interpreta e visualizza la pagine web.

FINALMENTE!

Capitolo 8

Reti Wireless e Mobili

Iniziamo ora l'ultima parte del nostro percorso introduttivo nel mondo delle reti ed internet. Ampliamo quindi la nostra trattazione alle reti locali wireless (IEEE 802.11) e alle reti cellulari (2G, 3G, 4G, ecc). Scendiamo nel dettaglio iniziando ad elencare alcuni elementi presenti in una rete wireless:

- **HOST WIRELESS:** Come per le reti cablate definiamo questi host come periferici nei quali vengono eseguite applicazioni. L'host wireless può essere quindi un portatile, un tablet, un telefono o un computer fisso. Gli host possono essere mobili o meno.
- **COLLEGAMENTI WIRELESS:** Gli host wireless si interconnettono tra loro o a una stazione base tramite un **Canale di Comunicazione Wireless**. Come siamo ormai in grado di capire è che differenti tecnologie comportano differenti tassi trasmissivi a distanza di trasmissione.
- **STAZIONE BASE:** Componente fondamentale delle reti wireless, nella controparte cablata non riusciamo a trovare qualcosa che somigli. Tale componente è responsabile della ricezione e dell'invio dei pacchetti degli host wireless a essa associati. Per associazione di un host a una stazione base si intende che:
  - L'host è nell'area di copertura della stazione base
  - La stazione base è utilizzata per lo scambio dati

  Un esempio pratico delle stazioni base sono i **ripetitori di cella** oppure gli **Access Point** in **LAN 802.11**

Si suole dire che gli host vengono detti operanti in **Modalità Infrastruttura**, infatti tutti i servizi sono forniti dalla stazione base. Diversamente da quelle che si chiamano **Reti AD HOC**, gli host non hanno infrastruttura dove connettersi e quindi devono per forza gestire indipendentemente i servizi tradizionali.
Dato che la stazione base ha una copertura ben precisa, quando un host si muove pian piano uscendo dalla copertura di una stazione base per entrare nella copertura di una nuova stazione base, questa operazione, dove per un lasso di tempo l'host è connesso contemporaneamente a due stazioni base, viene chiamata **HANDOFF (o anche HANDOVER)**.

- **INFRASTRUTTURA DI RETE:** La rete più ampia alla quale l'host può (o vuole) connettersi

Tutti questi elementi possono essere intrecciate tra loro e formano due principali famiglie di reti:

- Se un pacchetto nella rete attraversa uno o più collegamenti wireless (detti HOP)
- Se vie è una infrastruttura o meno

Queste due famiglie sviluppano quattro situazioni interessanti:

1. **HOP SINGOLO CON INFRASTRUTTURA:** Reti con stazione base che si collega ad una rete cablata più grande. Tutte le comunicazioni hanno luogo tra stazione base e host mediante singolo hop wireless. 802.11 e 4G LTE sono di questa categoria.
2. **HOP SINGOLO SENZA INFRASTRUTTURA:** Non si hanno stazione base collegata alla rete wireless. Uno dei nodi di questa rete può coordinare la trasmissione degli altri nodi. Bluetooth e 802.11 ad hoc sono in questa categoria.
3. **HOP MULTIPLI CON INFRASTRUTTURA:** Abbiamo una stazione base collegata tramite cavo a una rete più grande. Nonostante questo alcuni nodi potrebbero dover fare affidamento su altri nodi wireless per comunicare con la stazione base. Reti MESH Wireless e alcune Reti Sensori Wireless ricadono in questa categoria.

4. **HOP MULTIPLI SENZA INFRASTRUTTURA:** Non si hanno stazioni base e i nodi hanno una lata probabilità di dover trasmettere i pacchetti e parecchi nodi per raggiungere la destinazione. I nodi possono essere mobili e la connettività tra nodi può subire variazioni, fanno parte della categoria la reti **MOBILI AD HOC NETWORK**, se i nodi mobili sono veicoli la rete è **VEHICULAR AD HOC NETWORK**.
   Su questa categoria la ricerca è ancora nel piano del lavoro, infatti i protocolli di questa categoria sono molto complessi.

8.1 Collegamenti Wireless

Come si può ben immaginare se una scheda Ethernet di sostituisce con una **scheda wireless** si potrebbe pensare che i livelli superiori non abbiano bisogno di modifiche. Quindi ci concentriamo sulle differenze che esistono a livello di collegamento.

- **ATTENUAZIONE DEL SEGNALE:** Le radiazioni elettromagnetiche si attenuano se vi sono ostacoli che devono attraversare. Il segnale si attenua anche se la distanza cresce.
- **INTERFERENZE DA PARTE DI ALTRE SORGENTI:**Sorgenti rafio che trasmettono nella stessa banda di frequenza interferiscono tra loro, non solo anche il rumore elettrico ambientale interferisce (microonde, motori, ecc).
- **PROPAGAZIONE SU PIÚ CAMMINI:** É una condizione che si verifica quando una parte delle onde elettromagnetiche si riflette su oggetti e terreno provocando disturbi per chi riceve data la differente distanza percorsa. Gli oggetti in movimento tra sorgente e destinazione provocano propagazione multiple.

Queste sono le principali ragioni che hanno spinto i progettisti a dotare i protocolli di collegamento wireless di CRC molto potenti per la rilevazione degli errori, inoltre tali protocolli supportano il trasferimento dati affidabile. Una caratteristica importante del livello di collegamento wireless è il **Rapporto Segnale Rumore (SNR, signal-to-noise ratio)** che è una misura relativa dell'intensità del segnare ricevuto, misurato in dB. Ci basti sapere che a SNR grandi corrispondono facilità

per il ricevente nel distinguere segnale e rumore di fondo. Una caratteristica ulteriore è il **BER - Bit Error Rate** o **Tasso di Errore sul Bit**, altro non è che una probabilità che un bit trasmesso sia ricevuto sbagliato dal ricevente.
Con queste caratteristiche delineiamo alcune proprietà interessanti:

- Su un dato schema di modulazione, maggiore è SNR minore sarà BER
- Per un dato SNR, una tecnica di modulazione con più elevato tasso di trasmissione dei bit ha un BER più elevato
- La selezione dinamica delle tecniche di modulazione del livello fisico può essere usata per adattare la tecnica di modulazione alle condizioni del canale

Per quanto riguarda il broadcasting abbiamo problemi diversi rispetto al cavo:

- **PROBLEMA DEL TERMINALE NASCOSTO:** Supponiamo che le stazioni A e C trasmettano a B se vi sono ostacoli lungo il cammino tra A e C è possibile che non riescano a sentirsi sebbene stiano interferendo entrambe con B
- **FADING:** Questo problema sorge se A e C non hanno una potenza tale da poter essere rilevabili l'una dall'altra ma stanno comunque trasmettendo a B interferendo

8.2 CDMA - Code Division Multiple Access

Prendendo la palla al balzo per quanto riguarda i canali di broadcast ricordiano che se due o più host condividono un canale si deve avere un protocollo di accesso al canale.
CDMA è un protocollo di suddivisione del canale, si basa sul fatto che ogni host detiene un codice univoco, detto **Chipping Sequence**, tramite cui si possono effettuare operazioni di codifica e decodifica dei dati, infatti gli host utilizzano la medesima frequenza del canale senza preoccuparsi delle collisioni poiché si sfrutta la proprietà di **ORTOGONALITÁ** dei codici (che sono una sequenza di bit, tali bit d'ora in poi li chiameremo *chip* per non confonderli coi bit relativi ai dati).
L'ortogonalità dei codici ci permette di:

- $CODIFICARE = DATI \times CHIPPING\_SEQUENCE$
- $DECODIFICARE = DATI\_CODIFICATI \times CHIPPING\_SEQUENCE$

La tecnica, tramite cui la banda non viene più suddivisa ma le trasmissioni possono essere sulla stessa frequenza, è detta **SPREADING**.
Infatti i codici sono **OVSC - Orthogonal Variable Spreading Code** che potrebbero essere di lunghezza diversa anche se li vedremo per semplicità di lunghezza fissa. Vedremo più avanti come e perché questi codici possono essere di lunghezza variabile per ora ci basti sapere che codici di lunghezza minore permettono trasmissioni a bit rate più elevati rispetto a codici di lunghezza maggiore.
Assumiamo ora che sebbene il singolo bit possa avere valore 1 o 0 daremo valore 1 al chip quando vale 1 e -1 quando il chip vale 0, questo sarà chiaro tra poco nell'esempio che segue.
Supponiamo che il tasso trasmissivo dei bit di dati non codificati stabilisca l'unità di tempo, di conseguenza, il tempo per trasmettere un bit sia di uno slot.
Definiamo la nostra chipping sequence così:

C_m = -1 -1 -1 1 -1 1 1 1

di lunghezza M = 8

Ognuno dei bit dei dati originali sarà codificato tramite moltiplicazione col nostro codice, allora se nello slot 0 dovremmo inviare il bit 1 avremo:

$$Z = 1 \cdot C_m$$

Z viene dunque inviato al ricevente che per attuare la decodifica eseguirà:

$$d = \frac{\sum_{m=1}^{M} Z_m \cdot C_m}{M}$$

dove M è il numero totale di chip del nostro codice, nel nostro caso 8, quindi nel nostro esempio si avrebbe:

$$d = \frac{1+1+1+1+1+1+1+1}{8} = \frac{8}{8} = 1$$

Ovvero il bit che è stato effettivamente inviato!
Questo ci rassicura sul fatto che ogni bit inviato in maniera codificata viene ricevuto e decodificato in maniera corretta, ora però dobbiamo chiederci come sia possibile inviare il medesimo bit nel mentre che altri bit da diversa origine sono inviati nello stesso canale e nello stesso istante. Niente di più semplice, si continua a fare sempre la stessa procedura, vediamo un esempio.
Supponiamo di avere $Sender^1$ e $Sender^2$ che vogliono inviare ognuno i propri bit codificati sullo stesso canale. Dall'altra parte avremo $Reciver^1$ e $Receiver^2$ che deterranno le chipping sequence rispettivamente del $Sender^1$ e $Sender^2$. Nel nostro caso analizzeremo il $Receiver^1$ poiché $Receiver^2$ è speculare.
Definiamo allora:

$C^1_m =$ -1 -1 -1 1 -1 1 1 1 M = 8

$C^2_m =$ 1 1 -1 1 1 1 -1 1 M = 8

Supponiamo che $d^1 = 1 \; -1$ che sono i dati da inviare per $Sender^1$, e supponiamo che $d^2 = 1 \; 1$ che sono i dati da inviare per $Sender^2$.
Avremo allora:

$Z^1 = d^1 \, c^1_m =$ -1 -1 -1 1 -1 1 1 1 | 1 1 1 -1 1 -1 -1 -1 -1

slot 0 slot 1

$Z^2 = d^2 \, c^2_m =$ 1 1 -1 1 1 1 -1 1 | 1 1 -1 1 1 1 1 -1 1

slot 0 slot 1

Quindi i sender invieranno Z^1 e Z^2 sullo stesso canale contemporaneamente e quindi nel canale verranno (nei rispettivi slot) sommati.

Quindi nel canale avremo:

SLOT 0 = 0 0 -2 2 0 2 0 2

SLOT 1 = 2 2 0 0 2 0 -2 0

Questi dati codificati e sottoposti a interferenza arrivano al $Receiver^1$ che effettuerà l'operazione di decodifica:

$$d^1_{(SLOT0)} = \frac{\sum_{m=1}^{M} Z_m \cdot C^1_m}{M}$$
$$= \frac{0+0+2+2+0++2+0+2}{8} = \frac{8}{8} = 1$$
Esattamente il bit che il $Sender^1$ ha inviato nello $SLOT0$

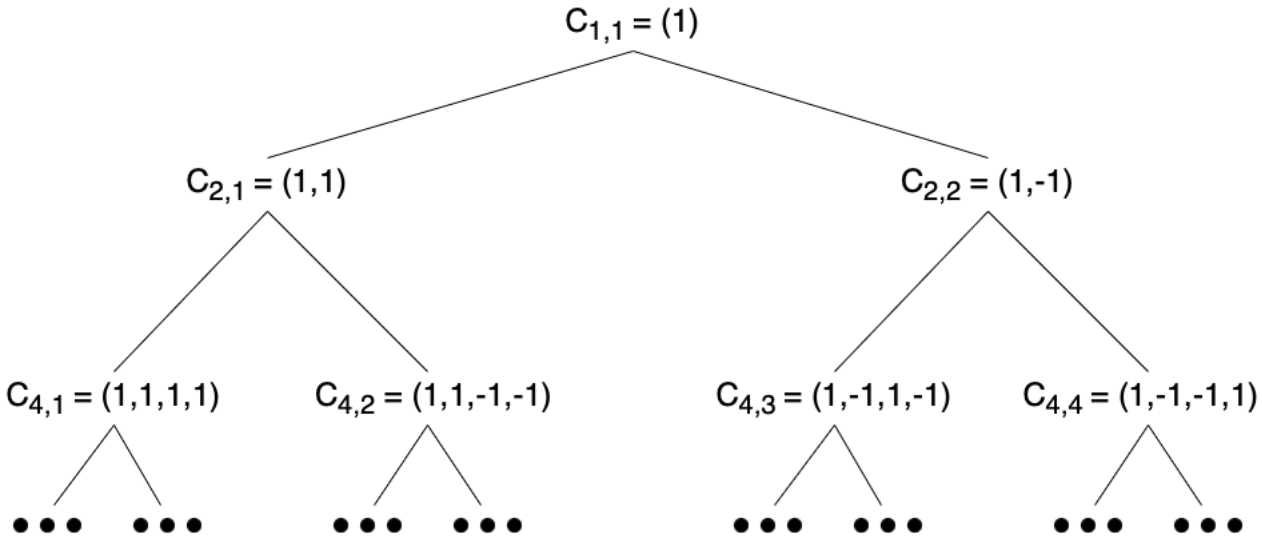
$$d^1_{(SLOT1)} = \frac{\sum_{m=1}^{M} Z_m \cdot C^1_m}{M}$$
$$= \frac{-2-2+0+0-2+0-2+0}{8} = -\frac{8}{8} = -1$$
Anche qui esattamente il bit che il $Sender^1$ ha inviato nello $SLOT1$

PRECISIAMO: Z_m non ha un apice di riferimento perché è il medesimo derivato dalla somma di Z^1_m e Z^2_m!

In maniera speculare il $Receiver^1$ eseguirà l'operazione di decodifica utilizzando la chipping sequence C^2_m (che ovviamente diversa da C^1_m).

Ora cerchiamo di comprendere come vengono generati i codici ortogonali che utilizziamo nella codifica e decodifica.
Iniziamo col capire che il codice ortogonale si può vedere come una matrice monodimensionale. Una matrice ortogonale è una matrice invertibile tale che la sua trasposta coincide con la sua inversa.
Parlando potabile e prendendo in esempio C^1_m e C^2_m utilizzati precedentemente e calcoliamo il loro prodotto scalare ci rendiamo conto che è zero.
La generazione di questi codici si basa su una teoria matematica denominata **TRASFORMATA DI WALSH-HADAMARD**, tramite

cui deriviamo l'**ALBERO DI CODICI DI SPREADING**.
Un esempio è il seguente:



É facile constatare che per ogni livello dell'albero si hanno codici ortogonali, prendendo due codici a caso si avrà come loro prodotto scalare 0.
Per costruzione dell'albero i codici possono essere assegnati anche con lunghezza diversa e come già detto avranno la conseguenza che codici di lunghezza minore permetteranno bit rate più elevato. Per chiarezza si sottolinea il fatto che sebbene i codici siano di lunghezza diversa rimangono ortogonali tra loro, se utilizzo $C_{2,2}$ è come se stessi utilizzando $C_{4,3}$. La spiegazione di questa proprietà esula gli obiettivi del corso e inoltre bisognerebbe addentrarsi in un'altra materia: la matematica, e me ne sto alla larga!

8.3 IEEE 802.11 Wireless LAN (o Wi-Fi)

Le LAN wireless sono oramai una tecnologia molto diffusa, emerge poi negli anni '90 lo standard che la gestisce: **IEEE 802.11**, conosciuto come **Wi-Fi**.
Vedremo quindi la struttura dei pacchetti 802.11,il protocollo di accesso al mezzo e come collaborano le reti 802.11 con le reti Ethernet cablate.
Vi sono numerose varianti dello standard 802.11, sebbene utilizzino lo

stesso protocollo di accesso al mezzo, CSMA/CD, e anche la stessa struttura del frame. Tutte le variazioni hanno però retrocompatibilità. Vediamo alcune varianti:

- *802.11b con 2,4 GHz fino a 11 Mbps*
- *802.11a con 5 GHz fino a 54 Mbps*
- *802.11g con 2,4 GHz fino a 54 Mbps*
- *802.11n con 2,5 GHz e 5 GHz fino a 450 Mbps*
- *802.11ac con 5 GHz fino a 1300 Mbps*

Tutte le versioni sono predisposte per essere utilizzate con stazioni base e **AD HOC**.
Gli standard più recenti usano antenne a più ingressi e uscite, quindi sfruttano una tecnologia **MIMO - Multiple Input Multiple Output**, questo è ottimo perché si riesce effettivamente ad utilizzare il massimo tasso trasmissivo dello standard e si riesce inoltre a capire eventuali guasti e interferenze.

8.3.1 Architettura di 802.11

Nell'architettura dello standard 802.11 abbiamo il blocco principale che è costituito dl **BSS - Base Service Set**, che contiene una o più stazione base centrale, detta **AP - Access Point**.
Il BSS, che banalmente si riassume come una sottorete, si connette ad internet tramite AP che è connesso a uno switch o router che si connette poi alla rete esterna. Le stazioni 802.11 hanno indirizzi MAC a 6 *byte* cablati nel firmware della scheda di rete. Se l'Access Point è collegato alla rete Ethernet che collega il BSS a Internet, sono chiamate **Wireless LAN con infrastruttura**, vi sono anche reti **AD-HOC** che costituiscono un BSS ma senza AP **(Non le tratteremo)**.

Canale e Associazioni

Nel momento in cui l'amministratore di rete deve installare un AP, gli assegna uno o due **SSID - Service Set Identifier**. Quando un dispositivo cerca di connettersi cercherà l'AP tramite SSID. Nel momento in cui si installa il nostro AP bisogna assegnare un canale, 802.11 ricordiamo che opera in un delta di 85 MHz, questi 85 MHz definiscono

11 canali sovrapposti. Praticamente ogni 4 canali si ha 1 canale non sovrapposto e quindi si avranno i canali 1, 6 e 11 che hanno capacità globale trasmissiva di 33 Mbps. Descriviamo ora una situazione diciamo interessante: **Giungle Wi-Fi**. Praticamente abbiamo un luogo dove abbiamo cinque AP, il nostro dispositivo necessita di **associarsi** a uno di loro per potersi unire alla sottorete e avere la possibilità di inviare pacchetti. L'AP, come da specifiche 802.11, invia periodicamente alcuni frame detti **Frame Beacon** contenenti il SSID e l'indirizzo MAC, questo per determinare la propria presenza. Sapendo questo la stazione wireless (banalmente il nostro dispositivo) analizza gli 11 canali in cerca proprio dei frame beacon. Una volta conosciuti gli SSID degli AP scegliamo a quale AP connetterci e quindi ci associamo a quell'AP. L'algoritmo di associazione non è contemplato dallo standard 802.11, tale algoritmo è lasciato ai progettisti dell'host wireless, generalmente si associa all'AP che ha potenza di segnale più alta. Questo comporta un problema: se il segnale è forte ma l'AP è sovraccaricato allora verrà scelto nonostante ci sia un AP con segnale debole ma molto scarico.
Il comportamento degli AP che inviano frame beacon si chiama **SCANSIONE PASSIVA**. Abbiamo invece una **SCANSIONE ATTIVA** dove la stazione invia un frame broadcast e lo ricevono tutti gli AP in copertura. Quindi risponderanno tutti gli AP, sceglieremo il nostro AP e invieremo il frame di associazione che avranno come conseguenza un frame di risposta di avvenuta associazione.

8.4 Protocollo MAC 802.11

Ora che siamo sicuri di essere associati ad un AP dobbiamo capire come accedere al canale. 802.11 utilizza CSMA/CA, ovvero il protocollo CSMA con **prevenzione delle collisioni**. Il protocollo MAC 802.11 prevede notevoli differenze rispetto al protocollo MAC Ethernet sebbene entrambi utilizzino CSMA.
Partiamo col dire che essendo i canali wireless soggetti a tassi di errore nei bit più elevati si utilizza uno schema di avvenuta ricezione o ritrasmissione a livello di collegamento, detto **ARQ**. Prima di capire come funziona CSMA/CA vediamo lo schema di avvenuta ricezione/ritrasmissione.
Un frame inviato da una stazione in una rete wireless potrebbe non raggiungere mai la destinazione. Quando una stazione destinazione riceve

un frame che passa il controllo CRC, attende un tempo detto **SIFS - short interframe space**, e poi invia il frame di avvenuta ricezione. Se la stazione trasmittente in un arco di tempo stabilito non riceve il frame allora suppone un errore e ritrasmetterà il frame secondo le regole di CSMA/CA. Se il frame viene trasmesso un numero finito di volte senza conferma allora viene scartato. Possiamo ora comprendere CSMA/CA e supponiamo che una stazione voglia trasmettere un frame:

1. Se inizialmente la stazione percepisce che il canale è **inattivo** allora trasmette il suo frame dopo un tempo denominato **DIFS - distributed interframe space**

2. Altrimenti la stazione sceglie un valore casuale di ritardo che sfrutta l'attesa esponenziale binaria e decrementa il valore dopo DIFS, se il canale è percepito inattivo. Se il canale è percepito come occupato il contatore è fermo.

3. Quando il contatore è a zero la stazione smette di ascoltare e invia interamente il frame attendendo la risposta

4. Se la conferma arriva allora si sceglie il prossimo frame e si ritorna al passo 2. Se non si riceve conferma si ritorna al passo 2 ma con un valore di ritardo incrementato

Da questo si comprende come 802.11 voglia evitare le collisioni poiché non viene rilevata e soprattutto perché comunque il frame verrebbe inviato per intero. Infatti se due stazioni rilevano che il canale è occupato entrambe entreranno nello stato di ritardo casuale, e si spera che i valori non siano simili.
Appena il canale sarà inattivo allora vi sarà una stazione con ritardo minore che lo occuperà e quindi l'altra bloccherà il contatore fino a nuovamente canale inattivo.
In questo scenario ovviamente le collisioni sono ancora possibili banalmente se consideriamo il problema del terminale nascosto e dell'attenuazione di segnale.

8.4.1 Terminali Nascosti - RTS e CTS

Il protocollo MAC 802.11 include anche un elegante soluzione per evitare collisioni derivati dal problema del terminale nascosto. Come abbiamo anticipato il problema del terminale nascosto può causare problemi

di collisioni, supponiamo di avere un AP che ha una copertura che raggiunge due host $H1$ e $H2$, la copertura di $H1$ e $H2$ però non riesce a coprire uno spazio talmente ampio da inglobarsi vicendevolmente. Quindi nel momento in cui $H1$ è a metà della sua trasmissione si suppone che $H2$ voglia inviare il proprio frame, trascorso il DIFS, non captando il segnale di $H1$, invia il frame e si verifica una collisione. MAC 802.11 introduce allora due frame: **RTD - Request To Send** e **CTS - Clear To Send**, questi due frame permettono di riservare l'accesso al canale.
Quindi ciò che avviene è che il trasmittente quando vuole inviare un frame, prima invia all'AP un frame RTD contenete il tempo necessario all'invio del frame contenente i dati e del frame ACK, quando l'AP riceve il frame RTS allora invia in broadcast il frame CTS.
CTS ha due scopi, ovvero comunica al trasmittente che è possibile inviare ma soprattutto comunica di non trasmettere a tutte le altre stazioni. Questo aumenta la performance per due motivi principali:

- Risolve il problema del terminale nascosto
- Se RTS e CTS fossero coinvolti in collisioni sono talmente piccoli da sviluppare una collisione di breve durata. Soprattutto però se vanno a buon fine i dati e l'ACK avranno sicuramente successo

8.4.2 Pacchetto IEEE 802.11

I frame dell'802.11 hanno una composizione molto simile a Ethernet ma ci sono delle differenze importanti.
Elenchiamo tutti i campi vedendo poi nel dettaglio i più importanti:

- Controllo del frame - (16 *bit*), si compone in:
  - Versione del protocollo - 2 *bit*
  - Tipo - 2 *bit*
  - Sottotipo - 4 *bit*
  - Verso AP - 1 *bit*
  - Da AP - 1 *bit*
  - Frammentazione - 1 *bit*
  - Copia - 1 *bit*
  - Alimentazione - 1 *bit*

  - Altri Dati - 1 *bit*
  - WEP - 1 *bit*
  - Riservato - 1 *bit*
- Durata - 16 *bit*
- Indirizzo 1 - 48 *bit* [6 *byte*]
- Indirizzo 2 - 48 *bit* [6 *byte*]
- Indirizzo 3 - 48 *bit* [6 *byte*]
- Numero di Sequenza - 16 *bit*
- Indirizzo 4 - 48 *bit* [6 *byte*]
- Payload da 0 *byte* fino a 2312 *byte*
- CRC - 32 *bit*

Payload e CRC

Il payload è solitamente un datagramma IP oppure un pacchetto ARP, sebbene la grandezza massima sia 2312 *byte* in realtà non si raggiungono mai i 1500 *byte*. Come già sappiamo CRC, che sono 32 *bit*, ci permettono di rilevare errori nei bit, bisogna considerare che nello standard 802.11 è un campo molto utile perchè abbiamo una probabilità di errore più alta.

Campi Indirizzo

La differenza più evidente tra pacchetto 802.11 e Ethernet, è sicuramente i 4 campi indirizzo da 6 *byte*, che ci consente di avere 4 indirizzi MAC. Spieghiamo subito che l'Indirizzo 4 è riservato per l'utilizzo di reti AD-HOC quindi lo metteremo da parte concentrandoci sugli altri 3 che sono sfruttati per reti con infrastruttura.
Lo standard 802.11 definisce i campi come segue:

- L'indirizzo 2 è l'indirizzo MAC della stazione che trasmette il frame. Se invece della stazione, il trasmittente, è l'AP allora il campo sarà riempito dall'indirizzo MAC dell'AP
- L'indirizzo 1 è l'indirizzo MAC del ricevente

- L'indirizzo 3 è l'indirizzo MAC dell'interfaccia del router che connette il BSS (o meglio la sottorete di cui fa parte) alle altre sottoreti.

Per capire meglio il ruolo cruciale che indirizzo 3 ha per poter interconnettere il BSS a una rete cablata, facciamo un esempio.
Supponiamo di avere due BSS, BSS1 e BSS2, ognuna con una propria AP e un gruppo di host wireless, ogni AP si connette (cablato) a un router R1. Immaginiamo di avere in BSS1 uno specifico host wireless H1 che analizzeremo. Ricordiamo che AP non arriva al livello di rete ma si ferma al livello di collegamento.
Come primo caso ipotizziamo di dover trasportare un datagramma dall'interfaccia del router R1 all'hosto wireless H1, ricordiamo che R1 non è a conoscenza della presenza dell'AP.
Allora R1 conosce l'indirizzo IP di H1, utilizzerà ARP per determinare il MAC di H1. Appena conosce tale indirizzo incapsula il datagramma in un frame Ethernet con campo sorgente contenente il proprio MAC e nel campo di destinazione il MAC di H1.
Il frame che giunge all'AP viene convertito in un frame 802.11 prima di trasmetterlo nel canale wireless. In questo frame l'AP inserisce nel campo Indirizzo 1, l'indirizzo MAC di H1 e nel campo Indirizzo 2 il proprio indirizzo MAC. Nel campo Indirizzo 3 inserisce l'indirizzo MAC dell'interfaccia di R1.
Come secondo caso vediamo la situazione in cui H1 risponde con l'invio di un datagramma a R1.
H1 crea un frame 802.11 e riempie:

- **Indirizzo 1 = indirizzo MAC AP**
- **Indirizzo 2 = indirizzo MAC H1**
- **Indirizzo 3 = indirizzo MAC R1**

Quando AP riceve il frame 802.11 lo converte in Ethernet inserendo come sorgente l'indirizzo MAC di H1 e come destinazione l'indirizzo MAC di R1.

Campi Numero di Sequenza, Durata e Controllo Pacchetto

Il campo **numero di sequenza** ha la stessa funzione che aveva quando abbiamo parlato di livello di trasporto, quindi permette al ricevente di

distinguere tra un frame appena trasmesso e una ritrasmissione. Il campo **durata** è riservato per la quantità di tempo che il canale è riservato per l'invio del frame dati e il frame di conferma (Dati, RTS e CTS).
Il campo **controllo del frame** è articolato in diversi sottocampi, poco interessanti, ci basti sapere quali sono i sottocampi, e li abbiamo già listati.
Diciamo solo che **Tipo** e **Sottotipo** servono per RTS, ACK e CTS.
I campi **Verso AP** e **Da AP** definiscono la funzione degli altri.

Sottorete IP: Mobilità

Immaginiamo ora, come spesso accade, di avere diversi BSS tutti su una stessa sottorete, quindi i nostri AP sono connessi direttamente ad uno switch e non più ad un router.
Quindi AP1 e AP2 si sovrappongono in copertura, e H1 si sta spostando da BSS1 a BSS2. H1 inizia a ricevere i frame beacon di AP2 che avrà comunque lo stesso SSID di AP1, allora si disconnette da AP1 e si connette a AP2 mantenendo IP e sessioni TCP. Questo è possibile per la caratteristica dello switch di autoapprendere. Sorge il problema però se la mobilità è frequente perché lo switch non è progettato per modificare frequentemente le tabelle di indirizzi MAC/porta.
Il trucco realizzato, che è temporaneo, è che AP2 invii un frame Ethernet per informare lo switch di H1 tramite se stesso. I progettisti di 802.11 stanno lavorando ad una soluzione migliore.

Funzionalità Avanzate

Vediamo ora un paio di funzionalità avanzate non propriamente descritte nello standard 802.11, infatti tali funzionalità sono rese disponibili per mezzo di meccanismi base dello standard. Si lascia quindi una maggiore libertà di implementazione, e di conseguenza proprietaria.

Adattamento del Tasso Come abbiamo accennato precedentemente esistono diverse tecniche per la modulazione associate ai diversi scenari di SNR. Supponiamo che vi sia un utente mobile (host wireless) lontano 20 *metri* dalla stazione base e che abbia un SNR elevato.
L'utente allora può comunicare con la stazione mantenendo un BER basso utilizzando una tecnica di modulazione a livello fisico che fornisce

un tasso trasmissivo alto.
Se l'utente inizia ad allontanarsi sempre di più, SNR tenderà a diminuire, d'altro canto però la tecnica di modulazione non cambia e il BER si innalza. Arriveremo allora al caso limite in cui la distanza è tale che nessun frame sarà recapitato correttamente. Per quanto si è deciso di sviluppare un meccanismo di selezione della tecnica di modulazione a livello fisico in maniera adattiva, basandosi su caratteristiche istantanee o recenti del canale comunicativo.
Il funzionamento si basa sulla filosofia probing vista nella sezione del controllo congestione TCP. Se un nodo non riceve due ACK per due frame consecutivi il tasso trasmissivo ricade nel valore subito più basso. Se per 10 frame tutti ricevono ACK allora avremo un incremento al valore subito superiore del tasso trasmissivo.

Gestione dell'Energia Se parliamo di dispositivi mobili parliamo sempre di batterie, infatti bisogna considerare il problema (o meglio l'ottimizzazione) della gestione dell'energia. Lo standard 802.11 consente la limitazione del tempo di trasmissione, ascolto e ricezione per impedire inutili sprechi.
Un nodo riesce a cambiare stato da attivo a dormiente in maniera autonomia, imposta un bit a 1 nel frame da inviare all'AP notificando che si sta per disattivare. Inoltre imposta un frame in modo da riattivarsi poco prima che l'AP invii i frame beacon. L'AP a sua volta essendo a conoscenza della disattivazione del nodo memorizza i frame destinati a quel nodo in modo da inviarli al suo risveglio.

8.5 Bluetooth e Zigbee

Nella famiglia di protocolli definita da IEEE 802 abbiamo anche due standard molto utilizzati: 802.15.1 e 802.15.4 rispettivamente comunemente denominati come **Bluetooth** e **Zigbee**.
Il bluetooth opera su un raggio limitato e a bassa potenza e a basso costo. Rientra nella categoria **WPAN - Wireless Personal Area Network**, opera nella banda a $2,4GHz$ in modalità TDM con slot da 625 *microsecondi*, con 79 *canali* cambiando canale in maniera pseudo casuale da uno slot all'altro. La strategia si chiama **FHSS - Frequency Hoping Spread Spectrum**, raggiungendo un tasso trasmissivo di 4 $Mbps$, sono ovviamente delle reti AD HOC quindi si necessita di una

auto-organizzazione.
L'organizzazione si chiama **PICONET** dove si sceglie un nodo principale, il **master**, e tutto gli altri sono **slave**.
Il master trasmette su slot dispari, mentre lo slave può trasmettere solo dopo che il master ha terminato e nello stesso slot, ed esclusivamente verso il master, anche questo deciso dal master che commuta lo stato da *In Sosta* a *Attivo*.
Zingbee invece si concentra su dispositivi con energia e cicli di vita inferiori al bluetooth, non tutti i dispositivi necessitano di grande banda, ad esempio: sensori, interruttori, dispositivi di sicurezza, eccetera.
In questo protocollo abbiamo due tipologie di nodi: **Reduced-Function**, tipo slave, e **Full-Function**, tipo master, diversamente dal bluetooth possiamo avere più dispositivi di tipo full-function che creano una **rete mesh** che consente a questi "master" di collaborare tra loro e scambiarsi frame.

8.6 Cellulare e Accesso a Internet

Al giorno d'oggi i cellulari sono diffusissimi, quindi meritano una loro trattazione sull'accesso a internet. Iniziamo facendo una panoramica sull'architettura di una rete cellulare, dividendola per generazione.

8.6.1 Panoramica Architettura di una Rete Cellulare

Partiamo col dire che i termini che andremo ad utilizzare sono conformi al **GSM - Global System for Mobile Communication**. La tecnologia cellulare è da sempre categorizzata per generazioni, quindi le più datate supportavano solo traffico telefonico. Per esempio il **1G** era progettato come FDMA per comunicazioni audio. Presto si passò al **2G** che sebbene fossero digitali si parla sempre di solo audio, arriverà in seguito il **2.5G** che implementerà anche il traffico dati. Avremo poi il **3G** che supporta dati e fonia implementando trasporto e collegamento molto più efficienti. Dobbiamo infine aspettare per il **4G** basato su tecnologia **LTE** dove il core della rete è interamente basato su IP, implementando, senza discriminazione, fonia e dati a velocità della dimensione di *Mb*.

Architettura di Rete 2G

In rete cellulare, quest'ultimo termine si riferisce alla suddivisione della rete in **celle**, ognuna con la propria copertura. Ogni cella contiene una **Base Transceiver Station - BTS** che scambia segnali con le stazioni mobili. Una rete 2G utilizza una combinazione di FDM e TDM, quindi se abbiamo F sottobande ognuna divisa in T slot avremo la possibilità di effettuare $F \times T$ chiamate contemporaneamente. Le BTS sono poi asservite a una **Base Station Controller** (ognuna serve solitamente fino a 10 BTS) che è la responsabile dell'allocamento dei canali radio BTS agli utenti mobili, tramite tecnica di **paging**, ovvero trovare la cella dell'utente ed eseguire l'**handoff**. L'insieme della BSC e delle sue BTS costituiscono uno dei numerosi **BSS - Base Station System**.
Le BSC (circa 5) fanno riferimento a un **MSC - Mobile Switching Center**, che è responsabile dell'autenticazione dell'utente, dello stabilire e terminare chiamate e dell'esecuzione dell'handoff.
Un MSC può sia essere connesso a sua volta a un **MSC-Gateway** che si occupa di connettere gli utenti alla rete pubblica, sia essere lui stesso un MSC-Gateway.

Architettura di Rete 3G

La terza generazione volle implementare in maniera migliore l'idea data da 2.5G, per questo mantenne la tecnologia dedicata alla comunicazione audio del 2G, che abbiamo appena visto, e implementò parallelamente un livello di collegamento dedicato ai dati. Il core della rete 3G implementò allora due nuove tipologie di nodi:

- **Serving GPRS Support Node - SGSN**
- **Gateway GPRS Support Node - GGSN**

Dove **GPRS** sta per **Generalized Packet Radio Service**, un servizio dati adottato nel 2G (o meglio nel 2.5G).
Il compito di un SGSN è quello di consegnare datagrammi a/da nodi mobili presenti nella rete di accesso radio. Interagisce con l'MSC della rete cellulare dedicato a quell'area fornendo servizi di autenticazione e handoff, e inoltra i datagrammi al GGSN che è l'ultimo componente del core che invia effettivamente il datagramma in rete, nascondendo a quest'ultima la mobilità dei nodi.
Spingendoci nel verso opposto e dando un'occhiata ai confini di questa

architettura troviamo **RNC - Radio Network Controller**, che è connessio sia a un MSC sia a un SGSN, si occupa di controllare i **Node B**, quelli che nel 2G abbiamo chiamato BTS.
Il lato interessante dell'architettura 3G consiste nell'uso della tecnica **Direct Sequence Wideband CDMA**, il servizio associato a questa tecnica è **HSPA - High Speed Packet Access**, che consente un downlink fino a 14 $Mbps$.

Architettura di Rete 4G

Lo standard **4G LTE - Long Term Evolution**, prende due migliorie importanti:

- **Core Network interamente basato su IP**
- **Rete di Accesso Migliorata**

Premettiamo alcune osservazioni ad alto livello:

- L'architettura 4G è "all-IP", sia voce che dati vengono trasportati in datagramma IP da/per il dispositivo wireless (UE - User Equipment) al packet gateway (P-GW) che collega la rete di confine 4G al resti della rete
- Il piano dati e il piano di controllo 4G hanno una netta separazione
- Netta separazione tra la rete di accesso radio e il nucleo della rete completamente basato su IP

Principali componenti dell'architettura 4G:

- **eNode B:** Discendente dal BTS e RNC, il suo ruolo è quello di trasmettere datagrammi tra UE e P-GW. I datagrammi dell'UE sono incapsulati nell'**eNode B** e inviati in tunneling al P-GW attraverso il nucleo di rete completamente basato su IP: **EPC - Evolved Packet Core**. Il tunnel è assoggettato al QoS (qualità del servizio).
- **P-GW Packet Data Network Gateway:** Assegna gli indirizzi Ip agli UE e si occupa del QoS, svolge anche operazioni di incapsulamento e decapsulamento

- **S-GW Serving Gateway:** Nodo di appoggio della mobilità nel piano dei dati, tutto il traffico passa da S-GW
- **MME Mobility Management Entity:** Esegue la connessione e la gestione della mobilità per conto dell'UE residente nella cella che controlla
- **HSS Home Subscriber Server:** Contiene informazioni dell'UE quali la capacità di roaming, il profilo della qualità di servizio e le informazioni di autenticazione

8.7 LTE Radio Access Network

LTE utilizza una combinazione di multiplexing a divisione di frequenza e di tempo sul canale di downstream, in gergo si dice a divisione di frequenza ortogonale. Ovvero i segnali inviati sui canali a frequenze diverse vengono creati in modo da generare bassa interferenza anche quando le frequenze sono molto vicine. Più time slot vengono allocati, sulla stessa frequenza o su più frequenze, più un nodo mobile è in grado di raggiungere tassi di trasmissione elevati.

8.8 Gestione della Mobilità

Iniziamo questa difficile parte con alcune nozioni basilari: il luogo permanente in cui risiede un nodo mobile è detto **Rete di Appartenenza (Home Network)**, le entità che gestiscono la mobilità per conto del nodo mobile all'interno della rete di appartenenza sono dette **Agenti Domestici (Home Agent)**. La rete in cui il nodo viene a trovarsi occasionalmente è detta **Rete Ospitante (Foreign Network)** e i relativi agenti: **Agenti Ospitanti (Foreign Agent)**. Il **Corrispondetnte** è l'entità che desidera comunicare con il nodo mobile.

8.9 Indirizzamento

Per poter ottenere una trasparenza dell'utente mobile alle applicazioni di rete è buona pratica che l'indirizzo rimanga invariato pur spostando il nodo da una rete ad un'altra. Quindi quando il nodo è in una rete ospitante bisogna reinstradare il traffico verso quella rete.

Allora si cerca di portare le funzionalità di mobilità dal nucleo della rete alla sua periferia, un nodo per ottenere ciò sfrutta la rete di appartenenza. Gli agenti nella rete di appartenenza monitorano la rete nella quale il nodo mobile si trova a risiedere, tramite un protocollo tra il nodo mobile, o l'agente ospitante che lo rappresenta, e l'agente domestico aggiornando la localizzazione del nodo stesso.
L'approccio più semplice risulta porre l'agente ospitante nei router agli estremi della rete visitata.
L'agente ospitante allora definirà un **indirizzo di mediazione** detto **COA - Care Of Address**, per il nodo mobile.
Avremo allora due indirizzi per il nodo mobile:

- Indirizzo Permanente
- COA

8.10 Instradamento

Per quanto riguarda l'instradamento verso il nodo mobile possiamo distinguere due approcci:

1. **Instradamento Indiretto:** Il corrispondente indirizza il datagramma all'indirizzo permanente, completamente inconsapevole della mobilità del nodo. L'agente domestico allora indirizzerà il datagramma al COA del nodo mobile che verrà inoltrato al nodo mobile stesso, questo però avviene incapsulando nuovamente il datagramma del corrispondente all'interno di uno nuovo più grande e indirizzato al COA, che arrivato all'agente ospitante verrà estratto e lo invierà al nodo mobile. L'inverso, ovvero l'invio del datagramma dal nodo mobile al corrispondente, è relativamente semplice perché viene eseguito direttamente.

2. **Instradamento Diretto:** L'instradamento indiretto sviluppa un'inefficienza che sfocia nel **problema dell'instradamento triangolare**, ovvero il corrispondente invia i datagrammi all'agente domestico che vengono poi reinderizzati al nodo mobile, sebbene sia possibile che ci siano percorsi, all'interno della rete, più efficienti. L'instradamento diretto risolve questo problema pagando come costo la complessità. Introduciamo allora l'**agente corrispondente**, nella rete corrispondente, che interroga l'agente domestico

rispetto al COA del nodo mobile e poi invia i datagrammi direttamente al COA.
Questo comporta la necessità di un protocollo di localizzazione dell'utente mobile tramite cui l'agente domestico è sempre a conoscenza del COA per poterlo inviare al corrispondente. Sorge però il problema relativo al fatto che il nodo mobile possa spostarsi nuovamente in una nuova rete ospitante. Il corrispondente interroga solo una volta l'agente domestico quindi non può sapere che il nodo mobile non è più nella vecchia rete ospitante. In questo caso ci viene in soccorso un **agente di appoggio**, il nodo mobile nella nuova rete ospitante registra un nuovo COA che verrà fornito all'agente di appoggio fornendo così la possibilità al corrispondente di continuare a utilizzare il vecchio COA, l'agente di appoggio incapsulerà i datagrammi e li invierà al nuovo COA.

8.11 IP Mobile

Lo standard di IP Mobile è una famiglia di protocolli complessa e mai statica, essendo intricato e con molteplici modalità ci limiteremo a studiare o suoi tre componenti principali:

1. **Ricerca dell'Agente:** Il nodo mobile che arriva in una rete ospitante (o anche quella di appartenenza) deve conoscere l'agente che offre i servizi che gli occorrono in quella rete. Questa ricerca può essere fatta tramite **avviso**, oppure **richiesta**. Con avviso l'agente manda periodicamente in broadcast dei messaggi ICMP che hanno 9 nel campo **Tipo**, in questo messaggio include l'IP del router dove l'agente risiede e una estensione con campi:

   - **H Bit Agente Domestico**
   - **F Bit Agente Ospitante**
   - **R Bit Richiesta di Registrazione**
   - **M e G Bit per Modalità di Incapsulamento**
   - **Campo COA**

   Con la richiesta invece il nodo invia in broadcast un messaggio ICMP con tipo 10 e l'agente invia direttamente al nodo un avviso.

2. **Registrazione presso l'Agente Domestico:** Una volta che il nodo ha ricevuto il COA bisogna informare l'agente domestico:
   (a) Il nodo invia la richiesta di registrazione (datagramma UDP) all'agente ospitante, contenente l'indirizzo dell'agente domestico (HA) l'indirizzo permanente proprio (MA), il tempo di scadenza della registrazione e l'identificativo a 64 *bit*
   (b) L'agente ospitante che riceve la richiesta esegue l'inoltro inviando il COA, HA, MA e il formato di incapsulamento in un datagramma UDP verso l'agente domestico
   (c) L'agente domestico che riceve il datagramma associa il COA all'indirizzo permanente del nodo, in maniera da inoltrare tutto quanto. Invia poi una risposta all'agente ospitante con lo stesso identificativo
   (d) L'agente ospitante riceve risposta e la inoltra al nodo
3. **Instradamento Indiretto dei Datagrammi:** L'abbiamo già visto non lo riporto

8.12 Gestione della Mobilità in Reti Cellulari

GSM adotta un instradamento indiretto, inviando la chiamata alla rete di appartenenza e poi da li a quella visitata. La rete di appartenenza viene chiamata **Home PLMN - Home Public Land Mobile Network**, mentre la PLMN è la rete visitata. La rete di appartenenza mantiene un DB detto **HLR - Home Location Register**, che contiene il numero telefonico permanente. Se al momento l'utente movile sta operando nella rete di un altro fornitore, HLR conterrà informazioni utili per ottenere l'indirizzo nella rete visitata dove instradare la chiamata. Il corrispondente viene detto **MSC di appartenenza**. La rete visitata invece mantiene un DB detto **VLR - Visitor Location Register**, che contiene una voce per ogni utente mobile che si trova nella parte di rete da lui servita.

NON TRATTEREMO L'INSTRADAMENTO DELLE CHIAMATE VERSO MOBILI

8.13 Handoff in GSM

Quando una stazione mobile cambia la sua associazione da una stazione base a un'altra avviene un **handoff**, o passaggio di mano.
Le fasi sono:

1. La vecchia stazione base comunica all'MSC visitato che sta per avvenire un handoff e la nuova BS a cui l'utente sarà associato
2. L'MSC visitato inizializza un percorso verso la nuova BS segnalandole che sta per essere eseguito l'handoff
3. La nuova BS alloca e attiva un canale radio
4. La nuova BS trasmette all'MSC e alla vecchia BS che è pronta e chiede di informare la stazione mobile dell'handoff
5. La stazione mobile viene informata
6. Stazione mobile e nuova BS si scambiano qualche messaggio per completare l'attivazione del nuovo canale
7. La stazione mobile informa la nuova BS che è stata completata l'handoff e la BS lo comunica all'MSC
8. La vecchia BS rilascia le risorse dedicate

Se l'handoff avviene su un MSC differente da quello della vecchia stazione allora GSM definisce un MSC di appoggio dove vengono inizializzate le chiamate come un instradamento indiretto.

Bibliografia

[1] James F. Kurose e Keith W. Ross, Reti di Calcolatori e Internet: Un approccio Top-Down, Settima Edizione a cura di Antonio Capone e Sabrina Gaito, Pearson, febbrario 2017.

[2] Prof. Matteo Sereno, Corso Teorico di Reti di Elaboratori dell'Università degli studi di Torino - Dipartimento di Informatica, a.a. 2020/2021.

[3] Prof. Michele Garetto, Corso Teorico di Reti di Elaboratori dell'Università degli studi di Torino - Dipartimento di Informatica, a.a. 2020/2021.

[4] "IPv4" Wikipedia, Wikimedia Foundation, 6 Dicembre 2022, https://it.wikipedia.org/wiki/IPv4

[5] "Il protocollo IP" Ing. Gennaro Boggia, Corso di Reti di Telecomunicazione, 2013/2014, https://telematics.poliba.it/images/file/boggia/retitlc/Protocollo_IP_2014.pdf

[6] "Il livello Network del TCP/IP - Il protocollo IP (versione 4)" Prof. Vittorio Ghini, Corso di Reti di Calcolatori, 9/10/2001, http://www.cs.unibo.it/~ghini/didattica/reti_lpr/TAC003.pdf

[7] "Border Gateway Protocol" Wikipedia, Wikimedia Foundation, 13 Gennaio 2023, https://it.wikipedia.org/wiki/Border_Gateway_Protocol#Selezione_delle_rotte_migliori

[8] "Multiprotocol Label Switching" Wikipedia, Wikimedia Foundation, 21 Marzo 2023, https://it.wikipedia.org/wiki/Multiprotocol_Label_Switching

[9] "Matrice Ortogonale" Wikipedia, Wikimedia Foundation, 22 Marzo 2023, https://it.wikipedia.org/wiki/Matrice_ortogonale

[10] "Chip (CDMA)" Wikipedia, Wikimedia Foundation, 22 Marzo 2023, https://en.wikipedia.org/wiki/Chip_(CDMA)

[11] "Code-division multiple access" Wikipedia, Wikimedia Foundation, 22 Marzo 2023, https://en.wikipedia.org/wiki/Code-division_multiple_access