

Reti di Calcolatori

UniVR - Dipartimento di Informatica

Fabio Irimie

1° Semestre 2024/2025

Indice

1 Introduzione	3
2 Architetture di rete	4
2.1 Reti locali	4
2.1.1 Organizzazione del Backbone	5
3 Modalità di comunicazione	6
3.1 Reti a commutazione di circuito	6
3.1.1 Vantaggi	7
3.1.2 Svantaggi	8
3.2 Reti a commutazione di pacchetto	8
3.2.1 Vantaggi	9
3.2.2 Svantaggi	9
4 Ritardi di trasmissione	10
4.1 Ordine di grandezza	11
4.2 Strumenti per calcolare il ritardo end-to-end	11
4.2.1 Ping	11
4.2.2 Traceroute	12
4.3 Quantità di dati trasferiti	13
5 Modello a strati	13
5.1 Stack ISO/OSI	14
5.2 Stack TCP/IP	15
5.3 Entità coinvolte nella comunicazione	16
5.3.1 Identificazione dei processi	16
5.3.2 Ottenimento dell'IP e della porta	17
6 Indirizzi IP	18
6.1 Suddivisione dei bit	18
6.1.1 Identificazione del prefisso e del suffisso	18
6.2 Indirizzi IP riservati	19
6.3 Conoscere il proprio indirizzo IP	20
6.4 Esercizi	21
6.5 Subnetting	21
6.5.1 Creazione delle sottoreti partendo da un blocco di indirizzi assegnato	22
7 Livello applicativo	23
7.1 Esempi di applicazioni e relativi protocolli di trasporto	24
7.2 Socket	24
7.3 Protocolli web	25
7.3.1 Protocollo HTTP	25
7.3.2 DNS (Domain name system)	29
7.4 Protocolli di posta elettronica	32

8 Livello di trasporto	33
8.1 Protocollo TCP (Transmission Control Protocol)	33
8.1.1 Formato del pacchetto	33
8.1.2 Gestione delle connessioni	36
8.1.3 Affidabilità	39
8.1.4 Velocità di trasmissione	41
8.1.5 Controllo di flusso	42
8.1.6 Controllo di congestione	43
8.1.7 Finestra di trasmissione	45
8.2 Protocollo UDP (User Datagram Protocol)	51
8.2.1 Formato del pacchetto	51
9 Livello di rete	51
9.1 Protocollo IP (Internet Protocol)	51
9.1.1 Formato del pacchetto	51
9.1.2 Frammentazione	53
9.2 Routing/instradamento	56
9.2.1 Algoritmi per il calcolo dei cammini minimi	59

1 Introduzione

Il problema principale che bisogna affrontare è la comunicazione tra 2 calcolatori, cioè lo scambio di informazioni. Per far comunicare 2 calcolatori c'è bisogno di alcuni requisiti:

1. **Protocollo:** È un insieme di regole che sovraintende alla comunicazione, in cui si definiscono:

- Il formato dei messaggi
- Le azioni da intraprendere nel gestire i messaggi stessi

Questo perchè per comunicare tutti devono "parlare la stessa lingua".

2. **Architettura di rete:** Come, fisicamente, trasportare i messaggi

Esempio 1.1. Prendiamo ad esempio la scrittura e la spedizione delle lettere.

Ci sono 2 utenti che vogliono scambiare delle lettere.

Per gestire il trasporto della lettera essa viene messa all'interno di una **busta**, che contiene informazioni su dove deve essere spedita. Una volta in busta, va imbucata in una cassetta delle lettere da cui poi verrà prelevata e mandata alla cassetta delle lettere del secondo utente dalla **rete** di distribuzione degli uffici postali.

L'utente poi preleverà la lettera dalla cassetta delle lettere e dopo aver controllato le informazioni sulla busta, la aprirà e leggerà il contenuto.

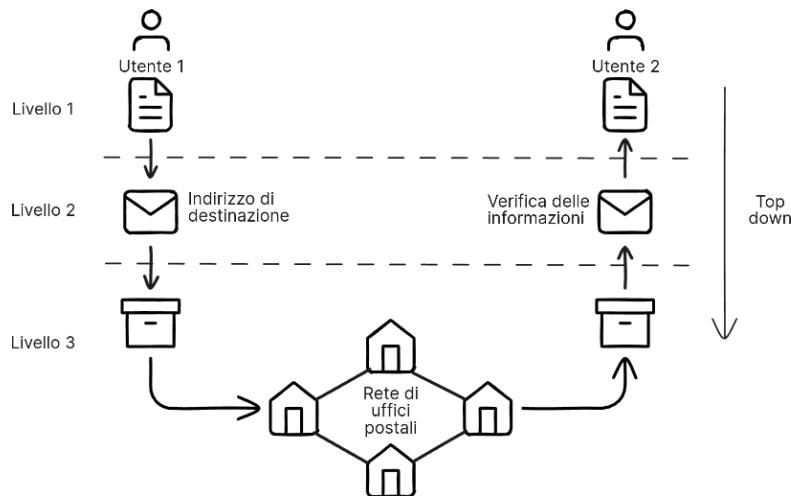


Figura 1: Esempio di comunicazione tra 2 utenti

Il **Protocollo** è il linguaggio utilizzato per comunicare tra i 2 utenti, mentre l'**Architettura di rete** è tutta quella infrastruttura che trasporta il messaggio tra i 2 utenti.

La rappresentazione dei sistemi di comunicazione di solito viene fatta nella modalità **top-down**, cioè si parte dal livello applicativo, quello più alto, fino a scendere nei livelli più bassi in cui si trova la vera e propria architettura della rete.

2 Architetture di rete

Di solito si fa riferimento all'architettura più utilizzata, cioè la rete **Internet**. Si possono distinguere i seguenti elementi base:

- **Calcolatori (End host)**
- **Router (Intermediate host)**
- **Collegamenti**

2.1 Reti locali

Le reti locali, o **LAN** (Local Area Network), sono caratterizzate da un **router di bordo** a cui sono collegati gli end host tramite **cavi fisici** o collegamenti wireless.

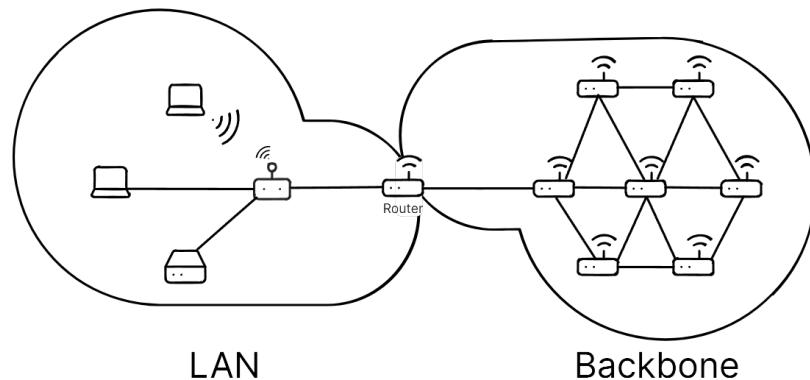


Figura 2: Rete locale

Per collegare diverse LAN tra loro esiste la **backbone**, cioè è una rete di router collegati tra di loro con una topologia gestita dal gestore della rete. Questi router sono geograficamente distribuiti su tutto il territorio.

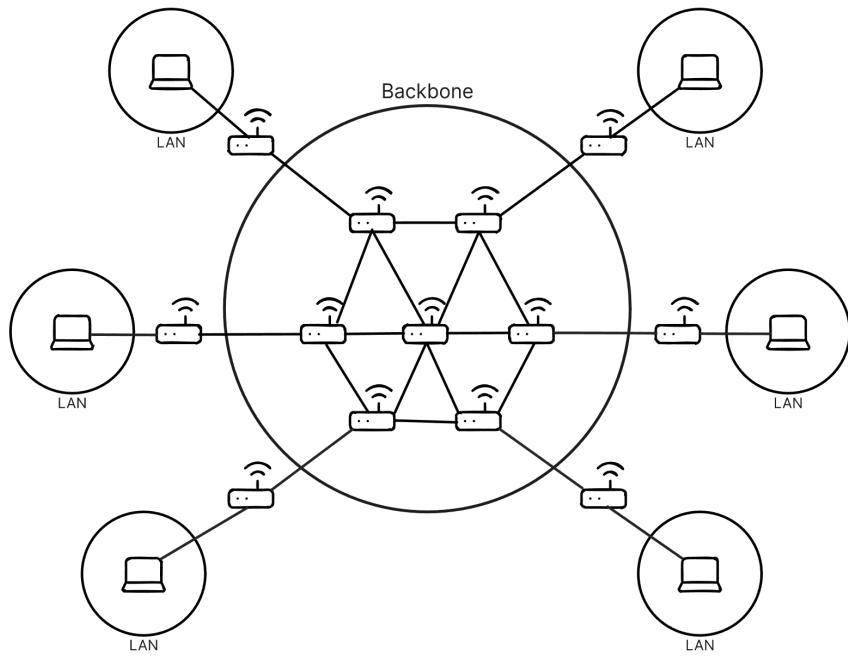


Figura 3: Intercollegamento di LAN

Nella maggior parte delle volte le connessioni tra i router all'interno della backbone sono cablate, di solito tramite fibre ottiche, soltanto in rari casi si usano connessioni wireless.

2.1.1 Organizzazione del Backbone

Il backbone è composto da diverse reti che appartengono a diverse organizzazioni permettendo di creare diverse interconnessioni tra le reti. Queste organizzazioni si chiamano **Internet Service Provider** (ISP). Gli ISP hanno diversi livelli:

1. **Livello 1**: Hanno una connessione internazionale e quindi sono in grado di comunicare con tutti gli altri ISP.
2. **Livello 2**: Lavorano a livello nazionale.
3. **Livello 3**: Lavorano a livello locale.

Gli ISP di livello 1 sono collegati tra di loro per permettere la comunicazione tra ISP di livello 1 diversi. Anche gli ISP di livello più basso permettono la comunicazione tra di loro o tra gli ISP di livello superiore, tutto questo grazie ad accordi commerciali tra le varie organizzazioni.

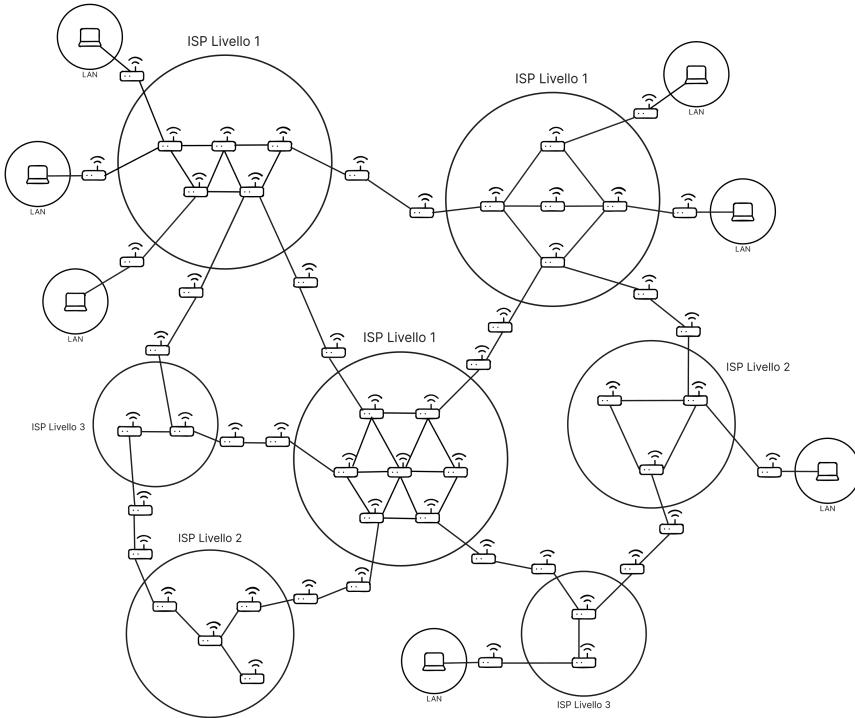


Figura 4: Livelli di ISP

Internet è la **Rete delle reti**, cioè è la rete che collega tutti gli ISP tra di loro ed è un organizzazione gerarchica, cioè è sufficiente creare collegamenti con un sottoinsieme di ISP operanti sul territorio per permettere il collegamento a tutta la rete.

Di conseguenza, per raggiungere un utente, in genere, si segue un percorso gerarchico. Un esempio è la rete stradale, dove per raggiungere una città si seguono le strade principali e poi si scende in quelle secondarie.

La scelta del percorso segue criteri basati su distanza e tempo.

3 Modalità di comunicazione

La gestione del trasporto dei messaggi è gestita dalla rete, però con che modalità trasferisco l'informazione tra 2 utenti?

3.1 Reti a commutazione di circuito

È la modalità di commutazione che è stata utilizzata per la prima volta.

In questa modalità le risorse (capacità del canale di trasmissione) vengono riservate **end-to-end** per la comunicazione, cioè viene letteralmente riservato un circuito che viene utilizzato dai 2 utenti.



Figura 5: Comutazione di circuito

Ogni canale è completamente dedicato alla comunicazione tra i 2 utenti, quindi se più utenti vogliono comunicare tra di loro, bisogna riservare altre risorse.

3.1.1 Vantaggi

- Risorse dedicate
- Ritardo deterministico

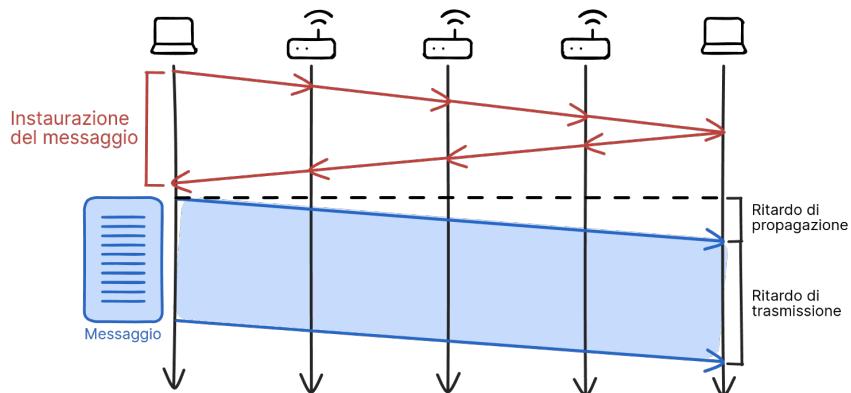


Figura 6: Ritardo

- **Ritardo di trasmissione:** Tempo necessario per trasmettere il messaggio
- **Ritardo di propagazione:** Tempo necessario per trasmettere il messaggio da un nodo all'altro

Se il messaggio è grande L bit e il canale riservato è di B bit/s, allora il tempo di trasmissione sarà:

$$T = \frac{L}{B}$$

Il ritardo di trasmissione e di propagazione è deterministico, perché dato il circuito di trasmissione, il tempo di trasmissione è noto.

3.1.2 Svantaggi

Nel corso di utilizzo sporadico si ha uno spreco di risorse, perché il circuito viene riservato per tutta la durata della comunicazione, anche se i 2 utenti non stanno comunicando.

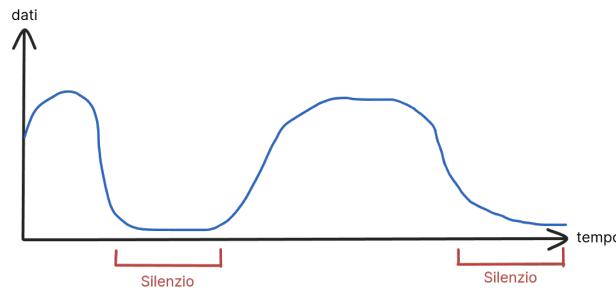


Figura 7: Spreco di risorse

3.2 Reti a commutazione di pacchetto

È la modalità di commutazione più utilizzata al giorno d'oggi.

L'informazione (messaggio) viene suddivisa in **pacchetti** e ad ogni pacchetto viene aggiunto un **header** per permettere:

- La consegna del pacchetto stesso
- La ricostruzione del messaggio

Il messaggio è l'informazione da trasferire, mentre il pacchetto è una porzione del messaggio stesso.

Il messaggio prima della trasmissione viene separato in unità più piccole e a queste unità viene aggiunta un'intestazione che serve a rendere le unità indipendenti per poterle trasmettere in modo indipendente. L'intestazione permette la consegna del pacchetto perché contiene la destinazione del pacchetto e la ricostruzione del messaggio perché contiene il numero di sequenza del pacchetto.

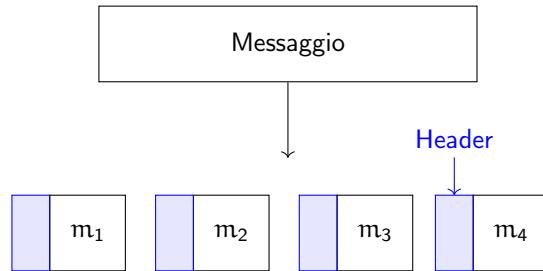


Figura 8: Messaggio e pacchetti

I pacchetti vengono salvati all'interno di un **buffer**, cioè una memoria temporanea, in attesa di essere trasmessi. Man mano che i pacchetti vengono inviati vengono accumulati nel buffer dei router e questo avviene per qualsiasi collegamento.

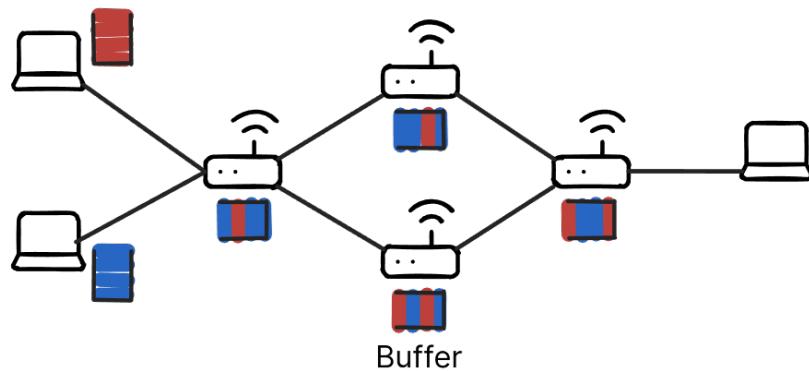


Figura 9: Rappresentazione del buffer

I pacchetti possono arrivare in ordine diverso rispetto a quello di trasmissione, però grazie all'header è possibile ricostruire il messaggio originale.

3.2.1 Vantaggi

- Utilizza le risorse solo quando ci sono pacchetti da trasmettere e questo viene chiamato **commutazione statistica**.
- **Multiplazione statistica**, cioè utilizzo lo stesso canale per trasmettere più pacchetti di utenti diversi.

3.2.2 Svantaggi

- Potenziale perdita dei pacchetti: La memoria dei buffer ha una capacità finita, quindi se il tasso di ricezione dei pacchetti è superiore al tasso di smaltimento del buffer, esso inizia a riempirsi. Le perdite aumentano la complessità di gestione della rete.
- Ritardi aumentati

I router prima di trasmettere i pacchetti, deve aspettare di ricevere tutti i pacchetti. Questo si chiama **store & forward**. Di conseguenza più aumentano i router, più aumenta il ritardo di trasmissione.

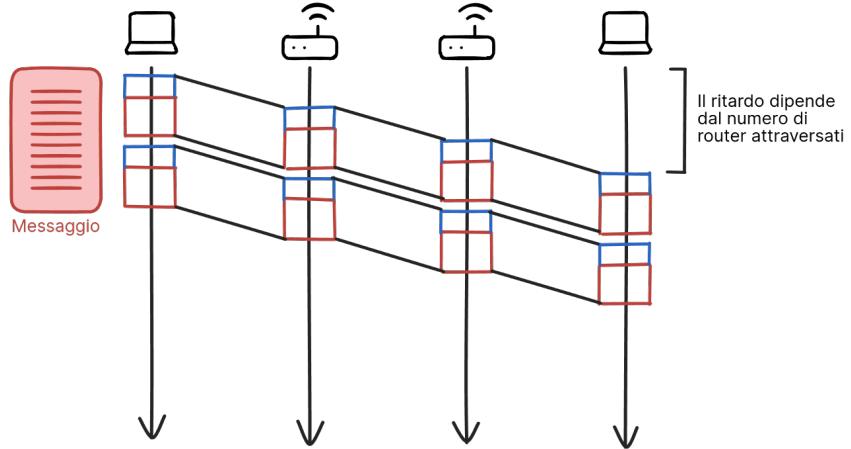


Figura 10: Ritardo di trasmissione dei pacchetti

4 Ritardi di trasmissione

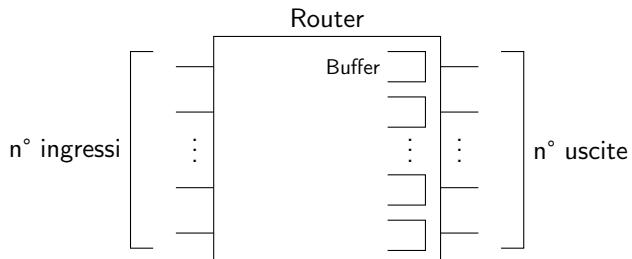


Figura 11: Struttura di un router

Su un singolo router le componenti principali del ritardo sono:

- **Ritardo di elaborazione al nodo:** Tempo necessario per elaborare il pacchetto.
- **Ritardo di accodamento:** È il tempo speso nel buffer prima che il pacchetto venga trasmesso ed è la componente principale tra tutti i ritardi. ($\frac{L}{B}$)
- **Ritardo di trasmissione:** Tempo necessario per trasmettere il pacchetto.
- **Ritardo di propagazione:** Tempo necessario per trasmettere il pacchetto da un nodo all'altro.

4.1 Ordine di grandezza

Dipende dalla distanza e dalla velocità di trasmissione del collegamento. La distanza si distingue in:

- **Locale**: < 10ms
- **Internazionale**: 20 – 40ms
- **Intercontinentale**: > 100ms

4.2 Strumenti per calcolare il ritardo end-to-end

4.2.1 Ping

Dati 2 utenti in 2 LAN diverse, il ping manda un pacchetto all'utente di destinazione e l'utente che lo ha mandato prima o poi riceverà un messaggio di risposta (**echo reply**). Il tempo che passa tra l'invio del pacchetto e la ricezione del messaggio di risposta è il ritardo end-to-end.

Non si può sapere se il ritardo è asimmetrico o no, cioè se il ritardo di andata è uguale al ritardo di ritorno, ma si può soltanto calcolare il ritardo totale.

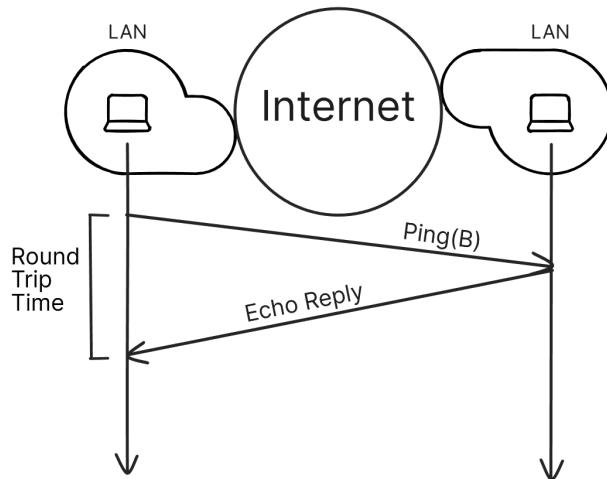


Figura 12: Ping

Se si esegue il comando `ping` da un terminale si riceve il seguente output:

```

ping www.google.com
PING www.google.com[142.251.209.4] 56(84) bytes of data.
64 bytes from mil04s50-in-f4.1e100.net (142.251.209.4): icmp_seq=1 ttl=115 time=15.2 ms
64 bytes from mil04s50-in-f4.1e100.net (142.251.209.4): icmp_seq=2 ttl=115 time=15.4 ms
64 bytes from mil04s50-in-f4.1e100.net (142.251.209.4): icmp_seq=3 ttl=115 time=15.4 ms
64 bytes from mil04s50-in-f4.1e100.net (142.251.209.4): icmp_seq=4 ttl=115 time=15.0 ms
^C
--- www.google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 14.964/15.221/15.393/0.171 ms

```

Figura 13: Output di ping

1. Viene rieseguito il ping facendo riferimento al **server fisico**
2. Tra parentesi viene rappresentato l'indirizzo che identifica il server, chiamato **indirizzo IP**
3. Alla fine c'è il tempo di risposta del server

4.2.2 Traceroute

Vengono mandati 3 messaggi al primo router e si calcolano i tempi di risposta tra il primo utente e il primo router, poi viene fatta la stessa cosa con i seguenti router fino ad arrivare all'utente di destinazione.



Figura 14: Traceroute

Se si esegue il comando traceroute da un terminale si riceve il seguente output:

```

traceroute -4 www.uv.es
traceroute to www.uv.es (147.156.200.249), 30 hops max, 60 byte packets
1 modemtim.homenet.telecomitalia.it (192.168.1.1) 7.589 ms 7.476 ms 9.427 ms
2 * * *
3 172.17.121.114 (172.17.121.114) 59.708 ms 172.17.121.116 (172.17.121.116) 59.672 ms 172.17.121.114 (172.17.121.114) 59.637 ms
4 172.17.128.136 (172.17.128.136) 59.600 ms 59.564 ms 172.17.120.134 (172.17.128.134) 59.526 ms
5 172.19.184.52 (172.19.184.52) 61.473 ms 172.19.184.50 (172.19.184.50) 62.829 ms 172.19.184.54 (172.19.184.54) 61.404 ms
6 172.19.177.62 (172.19.177.62) 61.346 ms 50.631 ms 50.538 ms
7 be200.milan26.mil.seabone.net (195.22.192.144) 56.724 ms 51.451 ms 195.22.205.116 (195.22.205.116) 51.375 ms
8 185.100.113.147 (185.100.113.147) 79.754 ms 185.100.113.145 (185.100.113.145) 81.267 ms 81.217 ms
9 nl-ams.nordu.net (88.249.209.203) 79.607 ms 67.327 ms 73.148 ms
10 213.46.175.38 (213.46.175.38) 79.312 ms 78.287 ms 79.858 ms
11 ndn-gw.mx1.lon.uk.geant.net (109.105.102.98) 78.147 ms 71.796 ms 77.514 ms
12 lag-1-0.rte.lon.uk.geant.net (62.40.98.60) 75.712 ms 78.651 ms 72.774 ms
13 lag-2-0.rte.lon2.uk.geant.net (62.40.98.65) 68.119 ms 68.000 ms 66.487 ms
14 lag-8-0.rte.par.fr.geant.net (62.40.98.167) 66.376 ms 66.318 ms 66.267 ms
15 ae4-0.rt1.bil.es.geant.net (62.40.98.222) 69.157 ms 71.086 ms 64.995 ms
16 rediris-las-geant.bil.es.geant.net (83.97.98.19) 46.398 ms 51.586 ms 53.018 ms
17 ehu-rt2-ethtrunk5.ciemat.rt2.madrid.rediris.es (130.206.245.5) 52.932 ms 52.877 ms 54.535 ms
18 uv-pnat-router.red.rediris.es (130.206.211.262) 62.778 ms 58.059 ms 62.832 ms
19 quarburquartest.red.uv.es (147.156.200.154) 56.624 ms 55.719 ms 59.404 ms
20 www.uv.es (147.156.200.249) 52.868 ms 59.342 ms 52.681 ms

```

Figura 15: Output di traceroute

1. Indica il numero di router attraversati
2. Stampa i valori di ritardo dei 3 pacchetti trasmessi
3. È il nome logico del router che si sta attraversando, da questo nome si può dedurre la posizione geografica del router e l'ISP a cui appartiene
4. Gli asterischi indicano che il router è stato configurato in modo da ignorare i pacchetti e non mandare alcuna risposta, questo perchè serve soltanto a inoltrare i pacchetti ad un altro router.

4.3 Quantità di dati trasferiti

La quantità di informazioni che si riesce a trasmettere si misura in $\frac{\text{bit}}{\text{s}}$ e questa informazione dipende dalla capacità di tutti i canali di trasmissione attraversati. Ogni canale avrà una dimensione diversa e si può determinare la banda totale a disposizione (**Throughput**) dal **collo di bottiglia**, cioè dalla minore capacità di trasmissione tra tutti i canali attraversati.

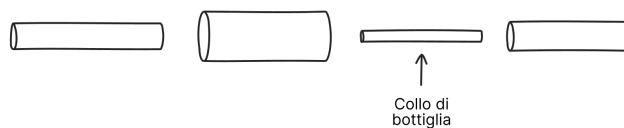


Figura 16: Throughput

5 Modello a strati

Il modello a strati affronta la **comunicazione tra due entità** secondo la modalità **divide et impera**.



Per scambiare informazioni le due entità devono comunicare. La comunicazione si divide in:

- **Comunicazione logica:** Gestisce le problematiche relative all'informazione
Ad esempio:
 - Linguaggio utilizzato
 - Come comportarsi nello scambio
- **Comunicazione Fisica:** Come trasferire i diversi bit. Il contenuto dell'informazione non è importante.

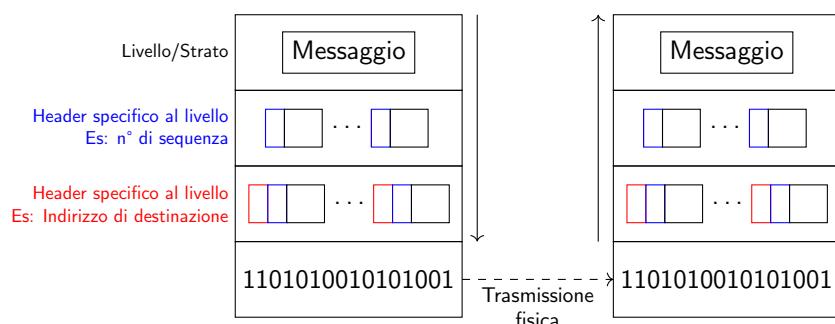


Figura 17: Comunicazione tra due entità

Ad ogni **livello** (o strato) viene elaborata parzialmente l'informazione e trasformata. Ogni livello aggiunge un **header** che contiene un'informazione specifica a quel livello, seguendo un **protocollo** (specifico per quel livello).

Ogni livello ha uno o più protocolli associati, e l'insieme dei protocolli di tutti i livelli è chiamato **stack protocollare**.

5.1 Stack ISO/OSI

Il modello **ISO/OSI** (International Standard Organization / Open System Interconnection) definisce dei livelli a seconda del sistema:

- **End system:**



Figura 18: Stack ISO/OSI per l'end system

- **Intermediate system:**

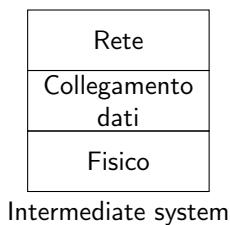


Figura 19: Stack ISO/OSI per l'intermediate system

5.2 Stack TCP/IP

Nella pratica (rete Internet) si utilizza lo stack protocollare **TCP/IP**:

- **End system:**



Figura 20: Stack TCP/IP per l'end system

- **Intermediate system:**

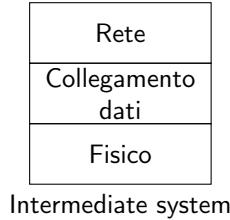


Figura 21: Stack TCP/IP per l'intermediate system

Il nome deriva dai due principali protocolli utilizzati:

- Protocollo di trasporto: **TCP** (Transport Control Protocol)
- Protocollo di rete: **IP** (Internet Protocol)

5.3 Entità coinvolte nella comunicazione

Su un calcolatore possono girare più applicazioni. Ogni applicazione può avere una connessione attiva, quindi ci possono essere più connessioni attive contemporaneamente.

Un applicazione può avere più istanze di comunicazione, cioè più connessioni attive contemporaneamente.

Di conseguenza **l'istanza di un'applicazione** è la vera e propria entità all'interno di una comunicazione. Quando si parla di entità si fa riferimento a uno specifico processo che gira su un calcolatore

5.3.1 Identificazione dei processi

Per identificare un processo servono 2 informazioni:

1. **Indirizzo IP:** Identifica il calcolatore
2. **Porta:** Codice numerico che identifica il processo all'interno del calcolatore

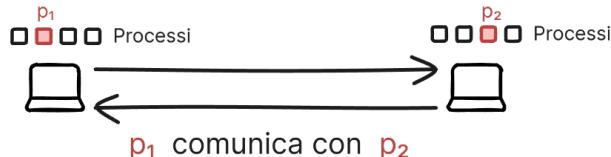


Figura 22: Comunicazione tra processi

Un flusso di comunicazione è identificato univocamente dalla tupla:

$$(IP_A, IP_B, Porta_A, Porta_B)$$

La porta viene assegnata ad un processo soltanto quando esso inizia a comunicare.

5.3.2 Ottenimento dell'IP e della porta

Queste informazioni sono contenute negli header aggiunti ad ogni livello.



Figura 23: Porta e IP

Un pacchetto si può rappresentare nel seguente modo:

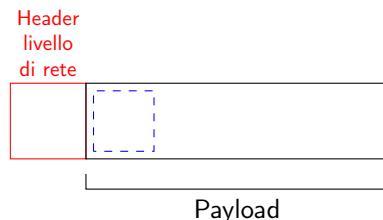
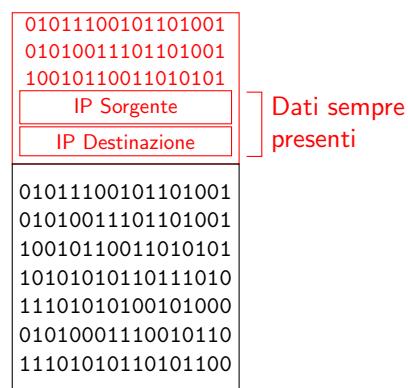


Figura 24: Rappresentazione di un pacchetto

oppure si può rappresentare anche come:



6 Indirizzi IP

Sono identificativi **univoci** di un'interfaccia di un host della rete Internet. Un end system può avere soltanto un'interfaccia, ma un router deve avere minimo 2 interfacce per poter permettere la comunicazione tra più entità.

Esempio 6.1. Un esempio di indirizzo IP è:

1001 1101 0001 1011 0001 0011 0111 1011

Per facilitare la lettura e la gestione degli indirizzi IP si usa la notazione **decimale puntata**. In questa notazione si considerano blocchi di 8 bit, di conseguenza si avranno 4 blocchi. Ogni blocco viene tradotto in un numero intero decimale compreso tra 0 e 255 e separato da un punto:

157.27.19.123

6.1 Suddivisione dei bit

I bit dell'indirizzo IP hanno tutti la stessa importanza?

Prendiamo ad esempio i numeri telefonici della rete fissa:

0039 045 802 7059
Prefisso Suffisso

- Il prefisso è l'identificativo di una specifica organizzazione
 - Il suffisso identifica un utente specifico all'interno dell'organizzazione

Allo stesso modo in un indirizzo IP si ha **prefisso e suffisso**

- **Prefisso:** Identifica una rete specifica
 - **Suffisso:** Identifica un'interfaccia di un host di una specifica rete

6.1.1 Identificazione del prefisso e del suffisso

Nel caso dei numeri telefonici si inserisce una barra per separare il prefisso dal suffisso:

045/7021412

Negli indirizzi IP il numero di bit dedicati al prefisso dipende dalla dimensione della rete ed è indicato da una barra seguita da un numero.

157.27.19.123/16

Significa che i primi 16 bit dell'indirizzo IP sono dedicati al prefisso.

$$\underbrace{10011101}_{n} \quad \underbrace{00011011}_{32-n} \quad \underbrace{00010011}_{\text{ }} \quad \underbrace{01111011}_{\text{ }}$$

Quindi il numero di indirizzi si calcola come:

indirizzi = 2^{32-n}

Esempio 6.2.

$$n = 20 \rightarrow 2^{12} = 4096$$

$$n = 24 \rightarrow 2^8 = 256$$

Per identificare il numero di bit del prefisso i calcolatori utilizzano una sequenza di 32 bit in cui i bit associati al prefisso sono posti a 1 e gli altri a 0, ad esempio:

$$/16 \rightarrow 11111111\ 11111111\ 00000000\ 00000000$$

Questa sequenza è chiamata **maschera** perché è abbinata all'indirizzo IP.

Esempio 6.3.

IP:	10011101	00011011	00010011	01111011
Mask:	<u>11111111</u>	<u>11111111</u>	<u>00000000</u>	<u>00000000</u>
	Prefisso		Suffisso	

Facendo l'AND bit a bit tra l'IP e la maschera si ottiene l'indirizzo di rete a cui quell'indirizzo IP appartiene.

La maschera può essere rappresentata anche in:

- **Notazione decimale puntata:**

$$/16 \rightarrow 255.255.0.0$$

- **Notazione esadecimale:** si divide l'IP in gruppi da 4 bit e si convertono in esadecimale:

$$/16 \rightarrow 0xffff0000$$

Per capire che un numero è esadecimale, esso è prefissato da 0x.

6.2 Indirizzi IP riservati

Non tutti gli indirizzi IP possono essere assegnati a degli host, infatti ce ne sono alcuni riservati per delle funzioni specifiche:

1. **This host:** È l'indirizzo IP del calcolatore stesso

0.0.0.0

2. **Local broadcast** (o limited broadcast): Indirizzo IP che permette di inviare un messaggio a tutti i calcolatori della rete locale

255.255.255.255

3. **Indirizzo di rete:** I primi n bit identificano una rete, gli altri $32 - n$ sono tutti posti a 0. Questo IP identifica la rete e nessun host può avere questo indirizzo

101010001100000000000000000000000000000000
 n $32-n$

4. **Directed broadcast**: È un IP che permette di mandare un messaggio a tutti gli utenti di una rete specifica, anche esterna alla rete locale

6.3 Conoscere il proprio indirizzo IP

L'indirizzo ip e la dimensione del prefisso dipendono dalla rete a cui si è collegati. Per conoscere il proprio indirizzo ip si usa il comando: ifconfig

```
ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
            loop txqueuelen 1000 (Local Loopback)
            RX packets 39 bytes 5012 (4.8 KiB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 39 bytes 5012 (4.8 KiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    1 inet 157.27.146.164 2 netmask 255.255.224.0 3 broadcast 157.27.159.255 4
        inet6 2001:760:2204:222:d2:0:41:3408 prefixlen 128 scopeid 0x0<global>
        inet6 fe80::c1a6:d355:4044:f133 prefixlen 64 scopeid 0x20<link>
            ether b2:76:5a:44:c9:9e txqueuelen 1000 (Ethernet)
            RX packets 17886 bytes 2998716 (2.8 MiB)
            RX errors 0 dropped 18 overruns 0 frame 0
            TX packets 20083 bytes 35308084 (33.6 MiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

1. **Interfaccia di rete**: Rappresenta il collegamento tra il calcolatore e la rete.
2. **Indirizzo ip**
3. **Indirizzo broadcast**
4. **Maschera**

Altre informazioni sulla rete si possono ottenere con il comando: whois. Questo comando si connette ad un server che contiene informazioni sugli indirizzi IP. L'output di questo comando da varie informazioni sull'IP messo come argomento, ma si può notare che la maschera è diversa da quella che viene data in output dal comando ifconfig.

- Per ifconfig il prefisso è da 20 bit, quindi $2^{12} = 4096$ indirizzi
- per whois il prefisso è da 16 bit, quindi $2^{16} = 65536$ indirizzi

Questo è dovuto al **subnetting**, cioè la creazione di sottoreti per partizionare l'organizzazione distribuita sul territorio, quindi dei 65546 indirizzi, alcuni verranno assegnati ad una sottorete e altri ad un'altra.

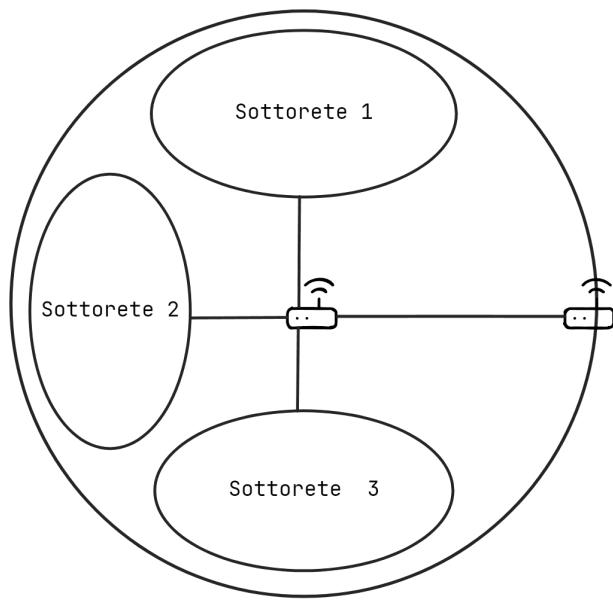


Figura 25: Subnetting

Questa suddivisione dall'esterno non viene vista e questo è il motivo per cui ipconfig mostra una maschera diversa da quella che si vede con il comando whois.

6.4 Esercizi

Conversione di IP decimale/binario

Esercizio 6.1. Dato il seguente IP:

1110 0111 1101 1011 1000 1011 0110 1111

la traduzione in decimale è la seguente:

231.219.139.111

Esercizio 6.2. Dato il seguente IP:

221.34.255.82

la traduzione in binario è la seguente:

1101 1101 0010 0010 1111 1111 0101 0010

6.5 Subnetting

È il processo che permette di suddividere una rete in sottoreti.

6.5.1 Creazione delle sottoreti partendo da un blocco di indirizzi assegnato

Un'organizzazione richiede un blocco di indirizzi in base alle proprie necessità. Prendendo come esempio l'Università di Verona, essa ha come indirizzo di rete il seguente:

157.27.0.0/16

si nota che i primi 16 bit sono dedicati al prefisso:

<u>10011101</u>	<u>00011011</u>	<u>00000000</u>	<u>00000000</u>
Prefisso		Suffisso	

Partendo da tale blocco si vogliono creare 2 sottoreti di pari dimensione. Per fare ciò si può prendere il primo bit del suffisso, separarlo dal suffisso e associarlo al prefisso.

<u>10011101</u>	<u>00011011</u>	<u>0</u>	<u>0000000</u>	<u>00000000</u>
Prefisso		Sottorete	Suffisso	

Il bit assegnato al prefisso può avere 2 valori, quindi si possono creare 2 sottoreti:

Sottorete 1:	<u>10011101</u>	<u>00011011</u>	<u>0</u>	<u>0000000</u>	<u>00000000</u>
Prefisso			Suffisso		
Sottorete 2:	<u>10011101</u>	<u>00011011</u>	<u>1</u>	<u>0000000</u>	<u>00000000</u>
Prefisso			Suffisso		

In notazione decimale puntata diventa:

Sottorete 1: 157.27.0.0/17

Sottorete 2: 157.27.128.0/17

Quello che cambia è che il prefisso è diventato di 17 bit.

Da un blocco /16 $\rightarrow 2^{32-16} = 65536$ indirizzi si ottengono 2 blocchi /17 $\rightarrow 2^{32-17} = 2^{15} = 32768$ indirizzi ciascuno.

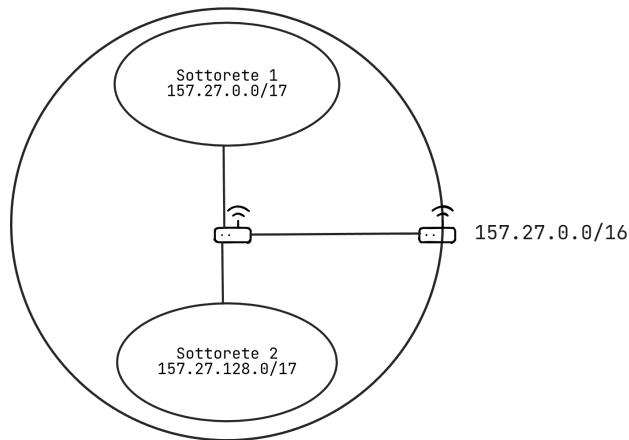


Figura 26: Risultato del subnetting

I blocchi di indirizzi si chiamano **CIDR** (Classless Inter-Domain Routing).

- **Nota storica:** In passato si usava la notazione **classful** in cui il numero di bit dedicati al prefisso era predeterminato e venivano usati i bit iniziali per distinguere i diversi casi. Ad esempio:

- **Classe A:** 8 bit di prefisso se l'IP iniziava con 0:

$\underbrace{0}_{\text{Prefisso}} \quad \underbrace{1010010}_{\text{Suffisso}} \quad \underbrace{00000000}_{\text{Prefisso}} \quad \underbrace{00000000}_{\text{Suffisso}}$

- **Classe B:** 16 bit di prefisso se l'IP iniziava con 10:

$\underbrace{10}_{\text{Prefisso}} \quad \underbrace{010010 \quad 10110100}_{\text{Suffisso}} \quad \underbrace{00000000}_{\text{Prefisso}} \quad \underbrace{00000000}_{\text{Suffisso}}$

- **Classe C:** 24 bit di prefisso se l'IP iniziava con 110:

$\underbrace{110}_{\text{Prefisso}} \quad \underbrace{010010 \quad 10110100}_{\text{Suffisso}} \quad \underbrace{00000000}_{\text{Prefisso}} \quad \underbrace{00000000}_{\text{Suffisso}}$

L'informazione del prefisso era codificata nell'indirizzo stesso e quindi non c'era bisogno di aggiungere un'informazione all'IP come ad esempio la maschera. Per l'alta richiesta di indirizzi, l'indirizzamento classful non era sufficiente:

Classe	N° reti	N° host
A	$2^7 \approx 16K$	$2^{24} \approx 16M$
B	$2^{14} \approx 16K$	$2^{16} \approx 65K$
C	$2^{21} \approx 2M$	$2^8 \approx 256$

Si esaurirono presto le classi A e B, quindi si passò al CIDR.

A volte assegnare delle sottoreti di uguale dimensione è uno spreco perché certe esigenze richiedono meno indirizzi e altre di più, quindi si dovrebbe suddividere la rete in sottoreti di dimensione **diversa**.

7 Livello applicativo

Il livello applicativo è il livello più alto dello stack protocollare:



Figura 27: Stack TCP/IP

Questo livello genera un messaggio che può essere di tipo diverso, ad esempio:

- Richiesta HTTP
- Risposta HTTP
- Email
- Il frame di un video

Quando un messaggio viene generato da un'applicazione si deve decidere quale protocollo utilizzare a livello di trasporto:

- **TCP** (Transmission Control Protocol): È un protocollo orientato alla connessione (**Connection oriented**) e affidabile. Questo protocollo dà la garanzia che il messaggio arrivi a destinazione.
- **UDP** (User Datagram Protocol): È un protocollo non orientato alla connessione (**Connectionless**) e non affidabile. Questo protocollo non dà la garanzia che il messaggio arrivi a destinazione e quindi non è detto che il messaggio arrivi.

Un servizio di trasporto **affidabile** garantisce che la consegna arrivi a destinazione.

7.1 Esempi di applicazioni e relativi protocolli di trasporto

- **Web**: Un esempio di applicazione è il browser web che utilizza il protocollo HTTP per trasferire dati attraverso la rete. Quando si consegnano testo e immagini non si devono commettere errori, quindi in questi casi il protocollo HTTP si appoggia a TCP.
- **Posta elettronica**: Un'altro esempio è la posta elettronica che utilizza il protocollo SMTP per inviare la posta. Questo protocollo richiede un servizio affidabile, quindi se un pacchetto viene perso, il protocollo TCP si preoccupa di recuperarlo.
- **Audio/video**: Per la trasmissione di audio e video di solito i protocolli sono proprietari, ma in questo caso non è necessaria l'affidabilità, basta garantire la tolleranza alle perdite, quindi ci si appoggia al protocollo UDP.
- **DNS**: Per navigare su internet si usa il protocollo **DNS** (Domain Name System) che è il servizio che traduce i nomi logici nei corrispettivi indirizzi ip.

7.2 Socket

Quando un'applicazione deve mandare un messaggio ad un'altra applicazione essa apre un **socket** verso la destinazione. Un socket è un'interfaccia di comunicazione (astrazione software) che identifica un flusso informativo bidirezionale. Per aprire un socket bisogna specificare:

- Protocollo di trasporto
- Indirizzo IP di destinazione
- Indirizzo IP sorgente

- Porta di destinazione
- Porta di sorgente

L'apertura di un socket è effettuata soltanto da uno dei due processi.



Tra i due processi comunicanti si possono identificare due ruoli distinti:

1. **Processo Client:** È il processo che è responsabile dell'inizio della comunicazione. Il client ha un indirizzo IP dinamico, cioè cambia in base alla rete a cui è collegato.
2. **Processo Server:** È il processo che sta su un host sempre raggiungibile e sempre in ascolto. Il server ha un indirizzo IP statico.

7.3 Protocolli web

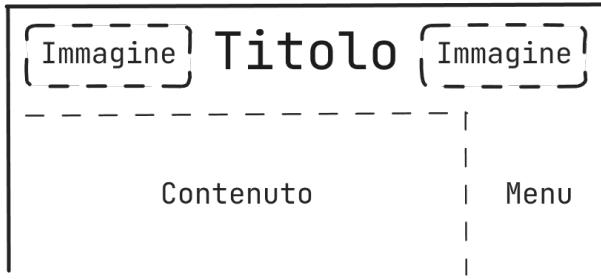
7.3.1 Protocollo HTTP

Il protocollo **HTTP** (HyperText Transfer Protocol) è un protocollo di livello applicativo usato per la trasmissione di documenti ipertestuali. Questo protocollo è di tipo **testuale** (i messaggi sono scritti in ASCII), determina il formato dei messaggi ed è basato sul modello **richiesta/risposta**:

- **Richiesta:** Il client apre un socket, invia un messaggio di richiesta al server e aspetta la risposta.
- **Risposta:** Il server deve inviare un messaggio di risposta alla richiesta del client

Se la richiesta non fosse ben formata il server risponde con un messaggio di errore. Per cui il protocollo prevede sempre, ad ogni richiesta, che ci sia una risposta, positiva o negativa.

Il server contiene delle pagine web, con eventualmente altri contenuti ad esempio immagini. Il file di testo principale (`index.html`).



Un file `html` definisce sia la struttura che il contenuto di una pagina web.



La struttura delle richieste HTTP è la seguente:

1. Riga di richiesta
2. Una o più righe di intestazione

Esempio 7.1. Un'esempio di richiesta HTTP è:

```

1  GET /index.html HTTP/1.1 // -- Riga di richiesta
2  Host: www.univr.it      // -- Riga di intestazione
3  User-Agent: Mozilla/5.0 // (intestazione facoltativa)
4  Accept-language: en     // (intestazione facoltativo)
5

```

che corrisponde a una richiesta di una pagina web del seguente tipo:

www.univr.it / index.html
Server File richiesto

Per fare una richiesta si utilizzano dei **metodi**:

- GET: Richiede una rappresentazione di una risorsa specifica
- POST: Invia informazioni al server
- HEAD: Richiede una risposta identica a quella di una richiesta GET ma senza il corpo del messaggio, quindi solo con le intestazioni
- PUT: Carica una rappresentazione di una risorsa specifica
- DELETE: Cancella la risorsa specificata

La struttura delle risposte HTTP è la seguente:

1. Riga di stato:
 - Versione del protocollo
 - Codice di stato
 - Messaggio di stato
2. Una o più righe di intestazione
3. Il corpo del messaggio

Esempio 7.2. Un'esempio di risposta HTTP è:

```
1  HTTP/1.1 200 OK          // -- Riga di stato
2  Date: Mon, 23 May 2005    // --
3  Content-Type: text/html   // | Righe di intestazione
4  Content-Length: 122      // --
5    
                                // Riga vuota obbligatoria~~~
6  <html>...</html>          // -- Corpo del messaggio
7
```

Per inviare e ricevere messaggi HTTP si può usare il comando `nc` o `netcat` e scrivere una richiesta HTTP come mostrato nell'esempio 7.1.

Altri elementi del protocollo HTTP che possono estendere il funzionamento sono i **cookies**. I cookies sono un meccanismo utilizzato dai server web per capire se è già avvenuta un'interazione con un determinato client in passato. L'IP non basta perché è dinamico.

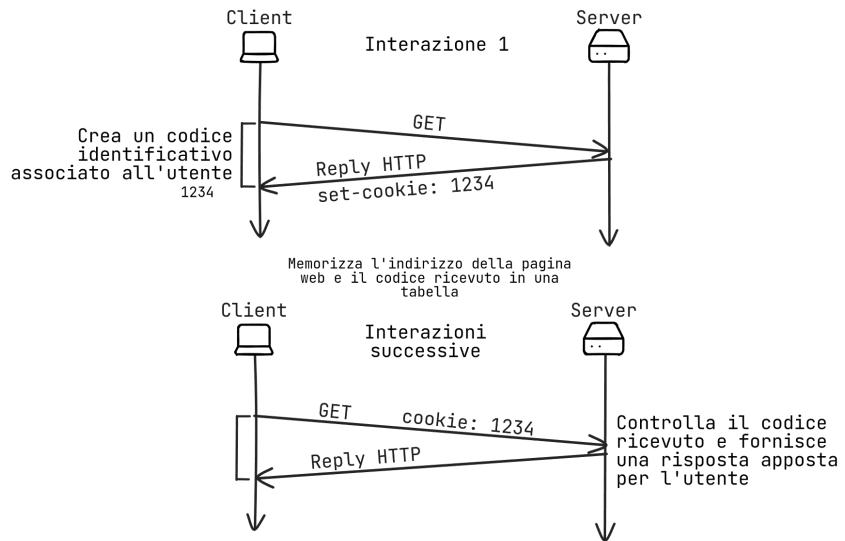


Figura 28: Esempio di cookies

Per diminuire il ritardo di accesso alle risorse si usa un meccanismo di **caching** che offre diversi vantaggi:

- Si riduce il ritardo di accesso
- Viene diminuito il carico sui server

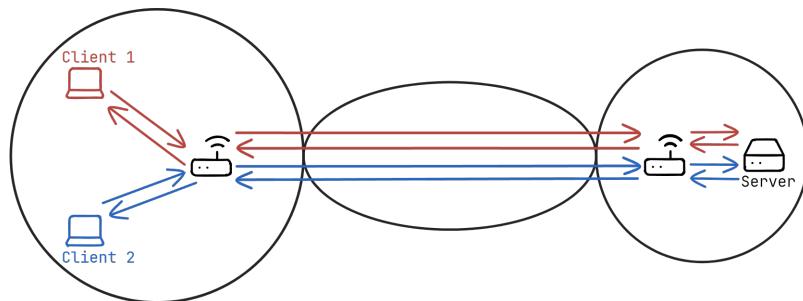


Figura 29: Traffico senza cache

Senza cache il ritardo che sente il secondo client è uguale al ritardo che sente il primo client e il carico è doppio.



Figura 30: Traffico con la cache

Con la cache il ritardo che sente il secondo client è minore e il carico è dimezzato perché la cache contiene già i dati che il secondo client vuole siccome sono stati caricati dal primo client.

Una buona cache lavora con un 70% – 80% di **hit rate**.

Se il contenuto sul server di origine cambia, la cache potrebbe contenere dati obsoleti. Per risolvere questo problema, il protocollo HTTP prevede il metodo **GET condizionale** in cui una riga di intestazione della richiesta è:

```

1 ...
2 If-Modified-Since: <data>

```

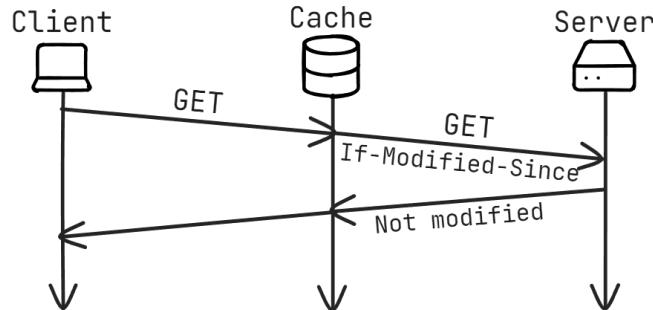


Figura 31: Comunicazione con la cache

Nella risposta del server (la prima con il contenuto, o le successive con "not modified") viene indicato anche il periodo di validità del contenuto:

7.3.2 DNS (Domain name system)

Lo scopo principale del DNS è la traduzione dei nomi logici (ad esempio `www.univr.it`) nei corrispondenti indirizzi IP (ad esempio `157.27.0.1`). Questo viene fatto tramite una tabella di traduzione che si trova in un server DNS.

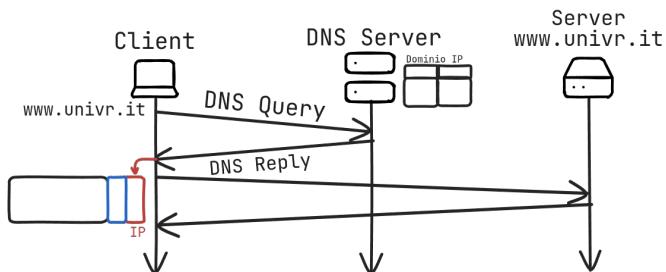


Figura 32: Comunicazione con il server DNS

Il server DNS ha un indirizzo IP statico:

- Può essere inserito esplicitamente dall'utente dalle impostazioni di rete
 - Viene fornito dalla rete stessa quando viene assegnato l'indirizzo IP (DHCP)

Per conoscere l'associazione tra i nomi logici e gli indirizzi IP, il server DNS utilizza un database **distribuito** e **gerarchico**:

- **Server DNS radice** (sono circa 7-10): Contengono i puntatori a una serie di server chiamati **server top level domain** (TLD)
 - **Server Top Level Domain** (TLD): Gestiscono le informazioni relative ai domini di primo livello (ad esempio .com, .it, .org). Ciascun TLD punta a una serie di server chiamati **Server DNS Locali**.
 - **Server DNS Locali**: Gestiscono il dominio di una organizzazione specifica

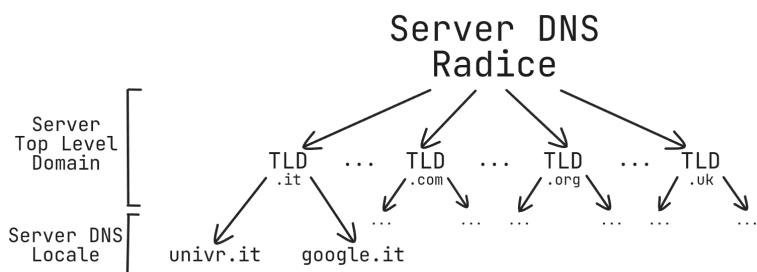


Figura 33: Albero dei server DNS

Un indirizzo (URL) è composto da più parti separate da un punto:

www . univr . it
Server specifico DNS Locali TLD

Il client interagisce solamente con il server DNS locale:



Figura 34: Comunicazione tra client e server tramite DNS

1. Il client invia una richiesta al server DNS locale chiedendo l'indirizzo IP di un determinato nome logico
2. Il server DNS locale manda una richiesta ad uno dei server radice salvati nella sua configurazione per ottenere l'indirizzo IP del server TLD
3. Il server radice manda una risposta con l'indirizzo IP del server TLD
4. Il server DNS locale manda una richiesta al server TLD per ottenere l'indirizzo IP del server DNS locale del sito richiesto
5. Il server TLD manda una risposta con l'indirizzo IP del server DNS locale
6. Il server DNS locale è in grado di contattare il server DNS locale del sito richiesto e chiede l'indirizzo IP del sito www.site.org
7. Il server DNS locale manda una risposta con l'indirizzo IP del sito al DNS locale
8. Una volta ricevuto l'IP, può essere inviato al client e venire utilizzato per contattare il server stesso

Il server DNS locale memorizza le risposte, e quindi gli indirizzi IP dei server DNS TLD e server DNS locali, con cui ha interagito recentemente.

Il rapporto tra il server DNS e la network cache è il seguente:



Figura 35: Comunicazione tra client e server tramite DNS

7.4 Protocolli di posta elettronica

Ci sono diversi protocolli di posta elettronica che variano in base all'invio e alla ricezione:

- **Invio:** SMTP (Simple Mail Transfer Protocol)
- **Ricezione:**
 - POP (Post Office Protocol), non viene più utilizzato, perché non sicuro
 - IMAP (Internet Message Access Protocol)
 - HTTP (Webmail)

Tutti i protocolli si appoggiano su TCP. Dal punto di vista dell'architettura, ogni dominio di posta elettronica (la porzione che segue la @) ha un server SMTP che gestisce caselle degli utenti appartenenti a quel dominio.



Figura 36: Invio e ricezione di posta elettronica

Quando il server `univr.it` riceve un messaggio, controlla il dominio. Se il dominio è `univr.it`, allora mette il messaggio nella casella di posta. Se il dominio è diverso, il server SMTP contatta il server SMTP di destinazione per la consegna del messaggio attraverso il protocollo SMTP.

8 Livello di trasporto

A livello di trasporto esistono 2 protocolli principali:

- TCP
- UDP

8.1 Protocollo TCP (Transmission Control Protocol)

8.1.1 Formato del pacchetto

La grandezza dell'header TCP è di 20byte tutti scritti in 5 righe da 4 byte. I campi dell'header sono i seguenti:



Figura 37: Formato del pacchetto TCP

- **Porta sorgente/destinazione** (bit 0-16): Identificatori dei processi sorgente e destinazione coinvolti nella comunicazione. Ci sono 2 tipi di porte:

- **Porte statiche**: Vanno dalla 0 alla 1023 e sono identificativi associati a protocolli definiti dallo standard (HTTP, SMTP,...) utilizzati **lato server**, quindi un client non può utilizzarle.
- **Porte dinamiche**: Vanno dalla 1024 alla 65535 e sono identificativi assegnati dal sistema operativo lato client quando viene aperto un socket.

- **Sequence number**:

- A livello applicativo vengono generati messaggi di dimensione arbitraria.
- A livello "collegamento dati" alla scheda di rete è associato un parametro chiamato **MTU** (Maximum Transmission Unit) che indica la dimensione massima di un pacchetto che può essere inviato. Il **MSS** (Maximum Segment Size) indica la dimensione massima di un segmento che può essere inviato.

Il sequence number identifica l'ordine dei segmenti rispetto al messaggio originale. Viene anche chiamato **offset rispetto al byte iniziale del segmento** e ad ogni sequence number viene sommata una costante uguale per ogni segmento chiamata **Initial Sequence Number (ISN)**.



Figura 38: Segmentazione di un messaggio

- **Acknowledgment number:** Il TCP è un protocollo affidabile, cioè garantisce la consegna di tutti i segmenti, quindi c'è bisogno di un meccanismo di conferma della ricezione. Questo meccanismo è chiamato **Positive acknowledgment with retransmission** e quindi per ogni segmento ricevuto il destinatario manda un acknowledgment con il sequence number del prossimo segmento che si aspetta, ad esempio:

Esempio 8.1. Prendendo come esempio la segmentazione del messaggio precedente si ha:

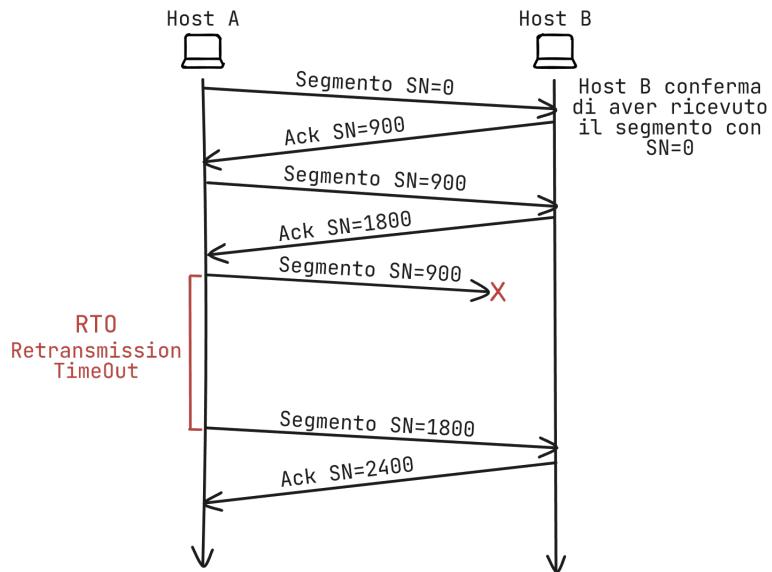


Figura 39: Positive acknowledgment with retransmission

Un acknowledgment è un segmento TCP senza dati, quindi ha solamente

l'header.

- **Offset:** Questo campo serve per indicare la grandezza dell'header e serve a sapere se ci sono delle opzioni aggiuntive.
- **Reserved:** Questo campo è riservato per usi futuri o per le opzioni.
- **Checksum:** Serve per controllare la presenza di errori nel segmento. La checksum è il risultato di una funzione che prende in input il segmento e restituisce un valore a dimensione fissa univoco per quel segmento. Se il segmento viene modificato durante la trasmissione, la checksum cambia e il destinatario fa un controllo mettendo in input ciò che ha ricevuto e se il confronto:
 - Va a buon fine: con **alta probabilità** non ci sono stati errori
 - Non va a buon fine: ci sono **sicuramente** errori e il segmento viene scartato

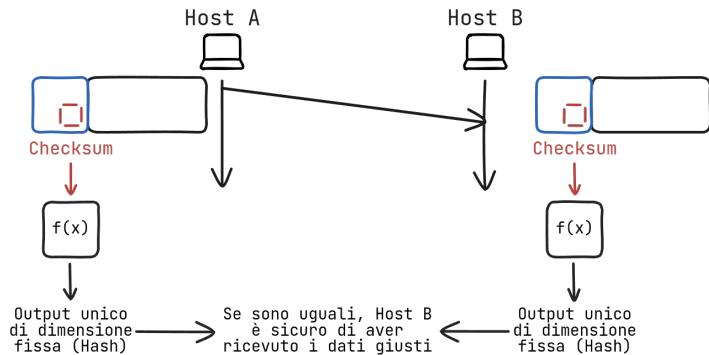


Figura 40: Checksum

8.1.2 Gestione delle connessioni

TCP è un protocollo **connection oriented**, cioè prima di scambiare dati, client e server devono stabilire che vogliono comunicare e questo avviene tramite l'invio di messaggi di servizio (senza dati), questa fase è chiamata **instaurazione della connessione**.

N.B.: Instaurare una connessione non ha niente a che fare con la creazione di un circuito o riservare delle risorse di rete

Quindi avviene uno scambio di segmenti TCM (header senza payload) chiamata **Three-way handshake**:



Figura 41: Three-way handshake

- **Syn:** Il client invia un segmento con il flag SYN a 1. Questo segmento contiene l'initial sequence number del client (ISN), cioè un numero scelto casualmente da cui il client inizia a numerare i byte inviati. Questo numero serve anche come ulteriore sicurezza per la cifratura.
- **Syn-Ack:** Il server risponde con un segmento con i flag SYN e ACK a 1. Questo segmento contiene l'initial sequence number del server e l'acknowledgment number uguale all'ISN del client + 1.
- **Ack:** Il client risponde con un segmento con il flag ACK a 1 e l'acknowledgment number uguale all'ISN del server + 1.
- **Altri parametri:**
 - **MSS (Maximum Segment Size):** Indica la dimensione massima di un segmento che il client e il server possono ricevere. Client e server utilizzeranno il valore minimo tra le 2 MSS comunicate. La MSS è contenuta nelle opzioni dell'header TCP, che sono utilizzate solo se necessario.

Dopo aver instaurato la connessione gli host si possono scambiare i messaggi:

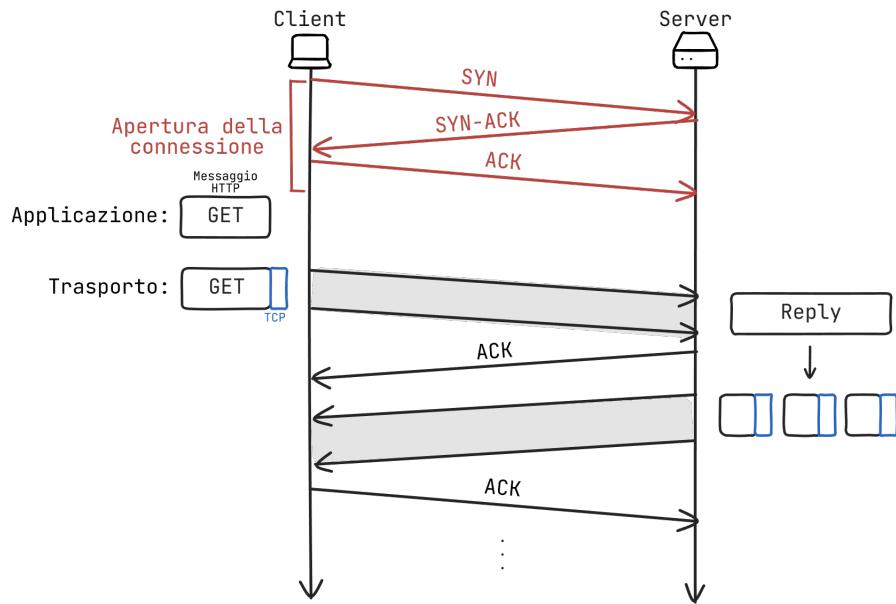


Figura 42: Connessione TCP

Quando lo scambio dei messaggi è terminato, la connessione viene chiusa attraverso uno scambio di header. Siccome il canale è biirezionale, la chiusura deve avvenire indipendentemente nelle due direzioni. I 2 messaggi sono:

- **FIN:** Sarà un messaggio con il flag FIN a 1
- **ACK:** Sarà un messaggio con il flag ACK a 1

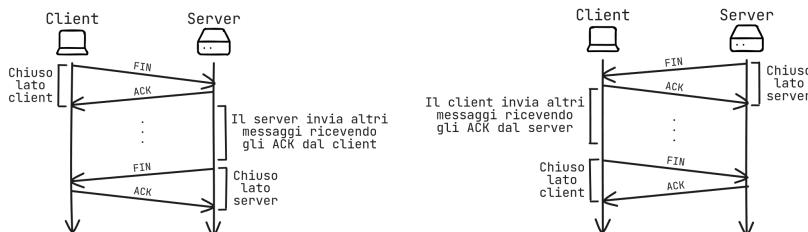


Figura 43: Chiusura non simultanea



Figura 44: Chiusura simultanea

8.1.3 Affidabilità

TCP è un protocollo **affidabile**, cioè quando un host invia un segmento si aspetta di ricevere il riscontro entro un tempo massimo (RTO, Retransmission TimeOut).

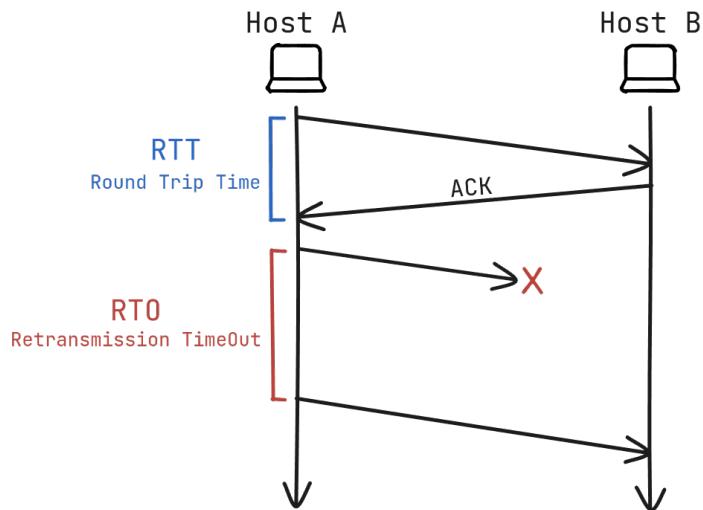


Figura 45: Affidabilità

L'RTO dovrà tenere in conto l'RTT (Round Trip Time), cioè il tempo che intercorre tra l'invio di un segmento e la ricezione del corrispondente ACK. All'apertura della connessione si misura il RTT durante il Three-way handshake.

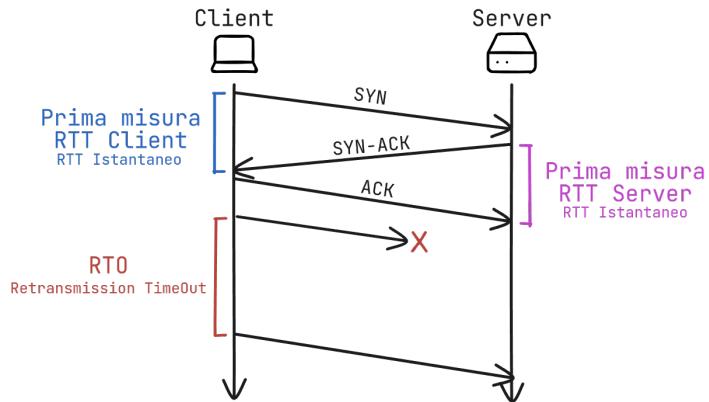


Figura 46: RTT istantaneo

Per l'invio del primo messaggio SYN il RTO è impostato a 500ms (caso peggiore). Per aggiornare i valori di RTT e RTO ogni host, per ogni segmento inviato, misura l'RTT istantaneo e usa tale valore per aggiornare una variabile chiamata SRTT (Smoothed Round Trip Time).

$$SRTT_{attuale} = \alpha \cdot SRTT_{precedente} + (1 - \alpha) \cdot RTT_{istantaneo}$$

dove $0 < \alpha < 1$ che di solito vale $\frac{7}{8}$.

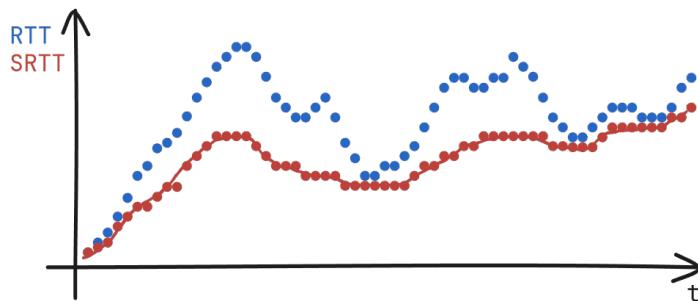


Figura 47: SRTT

Quindi l'SRTT è una **stima del valore medio** del RTT calcolata tramite l'Exponential Weighted Moving Average (EWMA). Questo metodo di calcolo è efficiente perché non deve tenere in memoria tutta la sequenza, ma solo l'ultimo valore.

Dato l'SRTT, l'RTO è pari a:

$$RTO = \beta \cdot SRTT_{attuale}$$

dove solitamente $\beta = 2$. L'RTO si **adatta dinamicamente** alla variazione del RTT. In caso di perdite, il RTO per il segmento ritrasmesso viene temporaneamente raddoppiato.

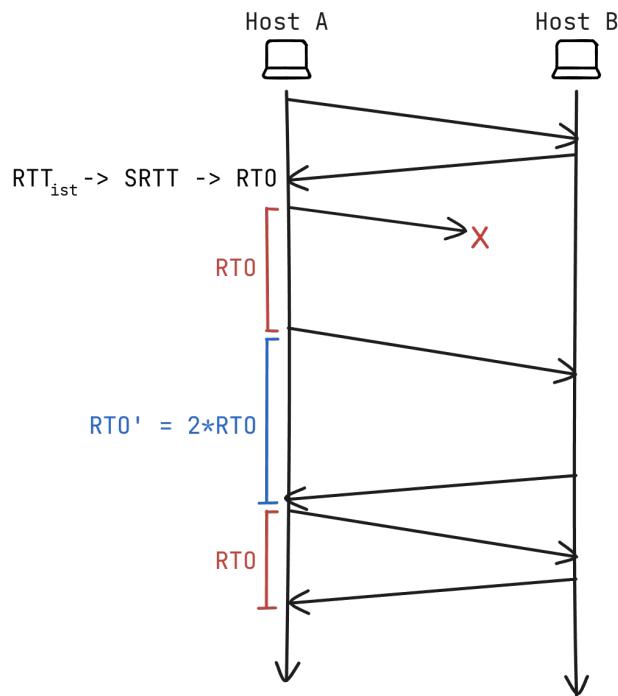


Figura 48: Esempio del raddoppio dell'RTO

Se si inviasse un segmento ad ogni RTT, la velocità di trasmissione sarebbe:

$$\frac{1\text{segm}}{\text{RTT}}$$

Quindi se un segmento è di 1500byte e un RTT è di 10ms, allora la velocità di trasmissione sarebbe:

$$\frac{1500\text{byte}}{10\text{ms}} = 120\text{Kbit/s}$$

8.1.4 Velocità di trasmissione

Invece di trasmettere un singolo segmento ad ogni RTT, se ne trasmettono di più contemporaneamente.



Figura 49: Invio di più segmenti contemporaneamente

Consideriamo la congestione come la perdita di un segmento, cioè un riscontro non ricevuto.

- **Controllo di flusso:** È un'azione preventiva per limitare la congestione
 - Quanti segmenti contemporaneamente?
 - Come gestisco i riscontri?
- **Controllo di congestione:** È una reazione in caso di congestione
 - Cosa succede se uno dei segmenti viene perso (e gli altri no)?

8.1.5 Controllo di flusso

Il controllo di flusso è un meccanismo che invia più segmenti contemporaneamente con un meccanismo a **finestra scorrevole** (sliding window):

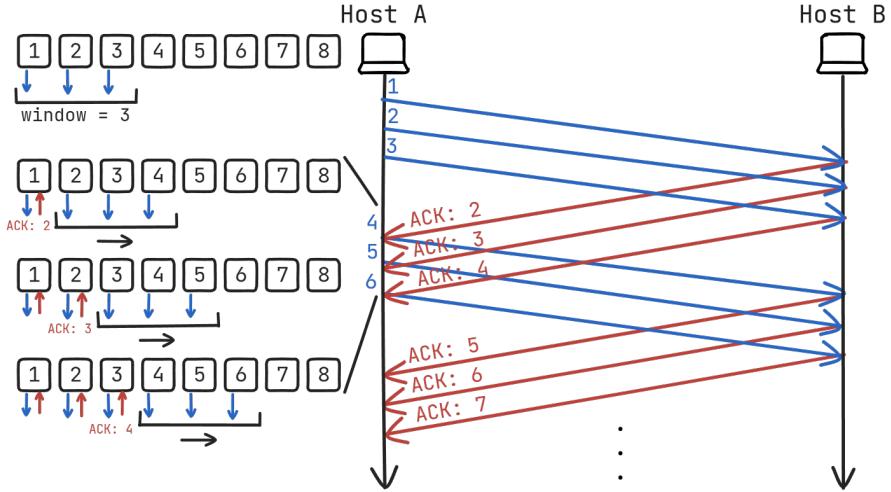


Figura 50: Esempio di finestra scorrevole

Questo meccanismo prevede che si salvino i segmenti inviati e si aspettino i riscontri per poter inviare i segmenti successivi. Se w è il numero di segmenti da inviare, allora ad ogni RTT si invieranno:

$$\frac{w \text{ segmenti}}{\text{RTT}}$$

8.1.6 Controllo di congestione

In caso di perdita di un segmento, la finestra non scorre più perché il riscontro del segmento perso non arriva, quindi bisogna aspettare il RTO e quando scade si ritrasmette il segmento perso. Nel frattempo l'host che non ha ricevuto il segmento manda più ACK in cui chiede di ritrasmettere il segmento perso (**ACK duplicati**). L'invio del singolo segmento chè è stato perso è chiamato **Selective Repeat**. Le informazioni dei pacchetti non ricevuti sono contenute nell'header TCP nel campo delle opzioni.



Figura 51: Selective Repeat

Lo scorrimento della finestra viene impedito anche se la ricezione dei segmenti avviene in ordine diverso da quello di invio.



Figura 52: Selective Repeat con segmenti disordinati

Se un router riceve tanti pacchetti di fila che non entrano nel buffer, adotta una politica secondo cui scarta dei pacchetti a caso dal buffer per fare spazio ai nuovi pacchetti.

8.1.7 Finestra di trasmissione

L'unico segnale per determinare la dimensione della finestra è il riscontro, quindi si utilizzano i riscontri (o la loro assenza) per variare in maniera dinamica la grandezza della finestra.

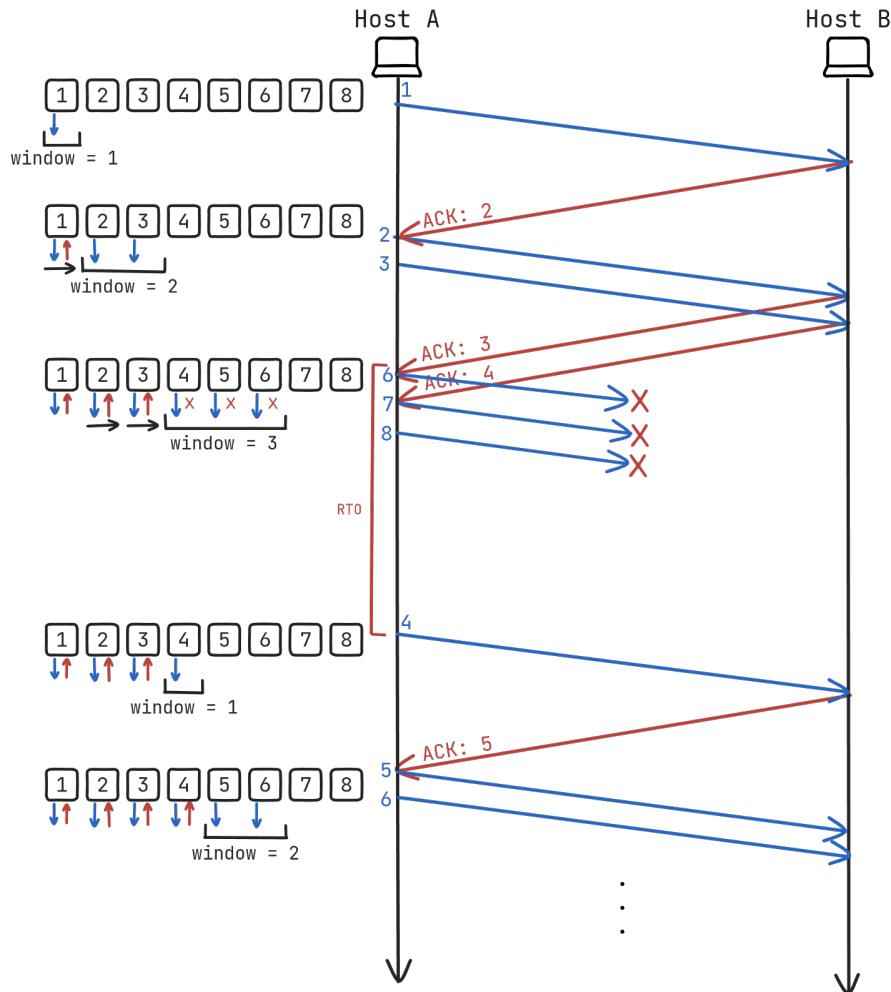


Figura 53: Dimensione dinamica della finestra

- **Riscontri positivi regolari:** Si può aumentare la dimensione della finestra. Ci sono 2 possibilità di aumento della finestre:

1. **Slow start:** Per ogni riscontro ricevuto, si fa scorrere la finestra verso destra di un segmento e si aumenta la sua dimensione di un segmento

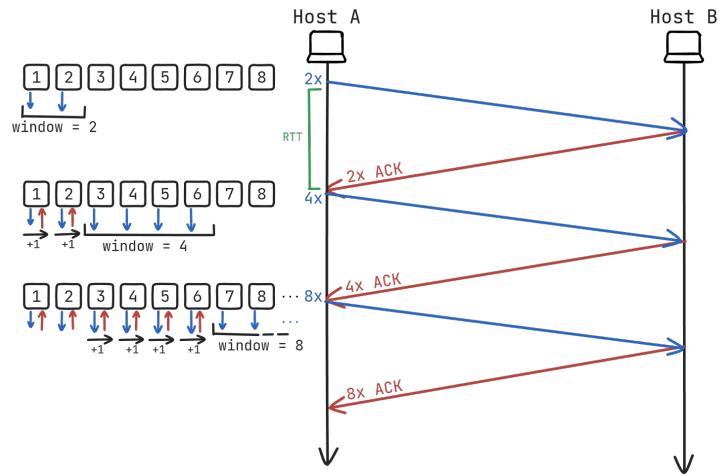


Figura 54: Esempio Slow start

Il grafico della crescita della finestra è esponenziale:

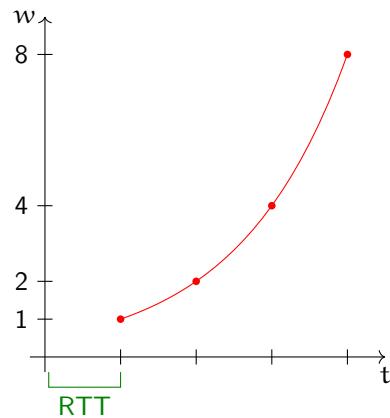


Figura 55: Grafico Slow start

2. **Congestion avoidance:** Per ogni riscontro ricevuto, si aumenta la finestra di $\frac{1}{w}$ dove w è la dimensione della finestra precedente.



Figura 56: Esempio Congestion avoidance

Il grafico della crescita della finestra è lineare:



- **In caso di perdite:** Il RTO scade, si ritrasmette il segmento perso e si riduce la dimensione della finestra. Ci sono 2 possibilità di riduzione della finestra:
 - Vanilla:** Una volta scaduto il RTO si riduce la finestra a 1 e si ritrasmettono i segmenti persi.
 - Fast retrasmitt/Fast recovery:** In caso di ricezione di 3 ACK duplicati di seguito, si pone la finestra a $\frac{w}{2}$ e si ritrasmette il segmento indicato negli ACK duplicati (se ci sono delle opzioni nell'header che indicano altri segmenti mancanti si ritrasmettono anche quelli).

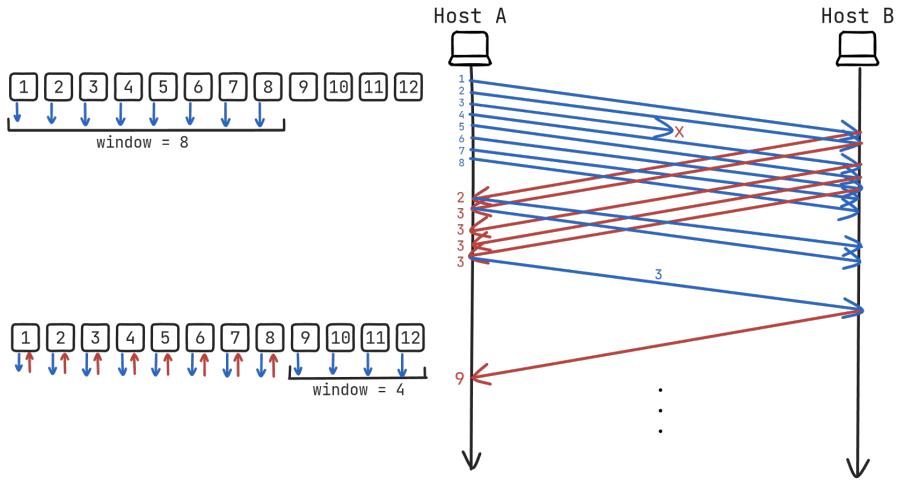


Figura 57: Esempio Fast retrasmit/Fast recovery

L'algoritmo di controllo della congestione è il seguente:

- **Variabili:**

1. cwnd: Congestion Window, è la dimensione della finestra di trasmissione attuale
2. rto: Calcolato dinamicamente in base all'RTT istantaneo e SRTT
3. rcvwnd: Recieve Window, è la finestra massima di ricezione
4. ssthresh: Slow Start Threshold, è una soglia usata per distinguere l'uso dell'algoritmo slow start o congestion avoidance

- **Inizializzazione:** All'inizio della connessione non si sanno tante informazioni, quindi si pongono le variabili nel modo più "conservativo" possibile:

1. cwnd = 1
2. rto = in base al valore dell'RTT misurato nel 3-way handshake
3. rcvwnd = viene comunicato dalla destinazione in ogni riscontro nel campo **window**
4. ssthresh = rcvwnd (in alcune implementazioni è $\frac{rcvwnd}{2}$)

- **Algoritmo:**

1. Invia un numero di segmenti pari alla cwnd
2. Quando arrivano i riscontri:

```

1 if cwnd < ssthresh // Uso lo slow start
2   cwnd = min(cwnd + numAck, rcvwnd, ssthresh)
3 else // Uso il congestion avoidance
4   cwnd = min(cwnd + numAck / cwnd, rcvwnd)
5
6 // Torno al punto 1
7

```

3. Se non arrivano i riscontri (quando scade il RTO):
 - (a) Pongo la $ssthresh = \frac{cwnd}{2}$ (cwnd al momento della perdita)
 - (b) Pongo la $cwnd = 1$ (Vanilla tcp)
 - (c) Per i segmenti ritrasmessi pongo la $rto = 2 rto$
 - (d) Torno al punto 1.

Esempio 8.2. Un esempio (semplificato in cui RTT è stabile) di evoluzione della trasmissione con TCP è il seguente:

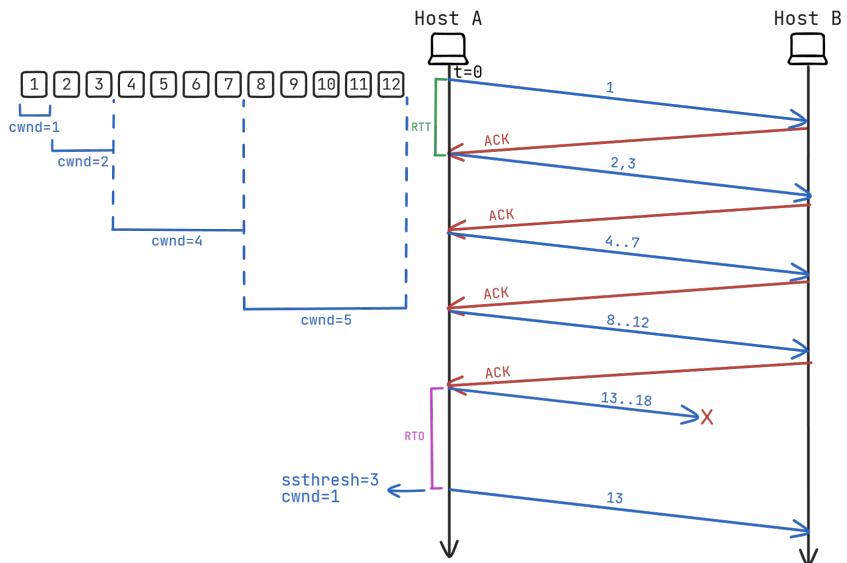


Figura 58: Evoluzione della finestra di trasmissione

Il grafico del cambio della dimensione della finestra è il seguente:

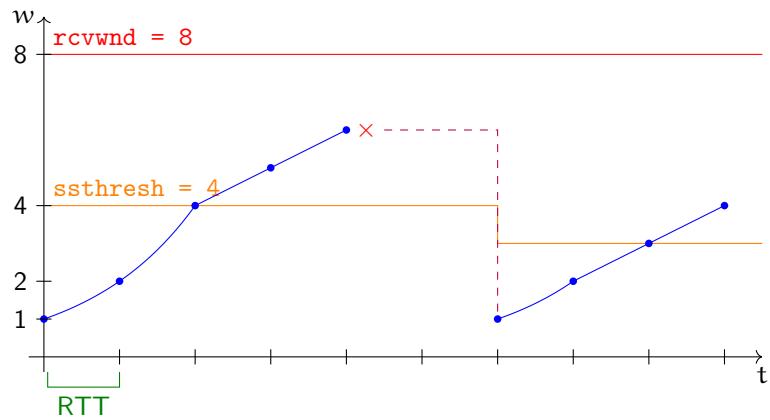


Figura 59: Grafico Slow start

Nel lungo termine, con numeri più grandi, il comportamento della finestra sarà:

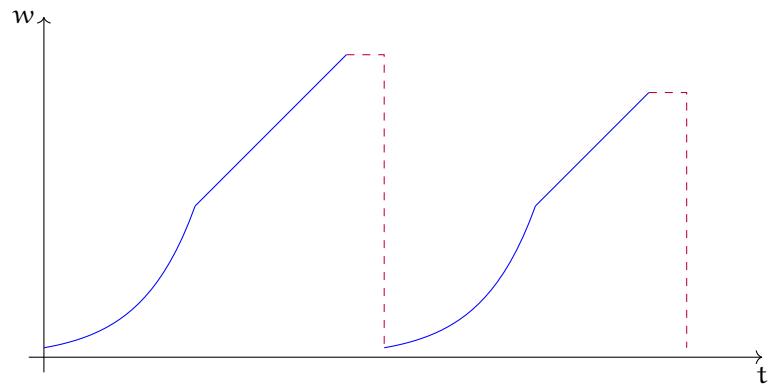


Figura 60: Andamento Vanilla

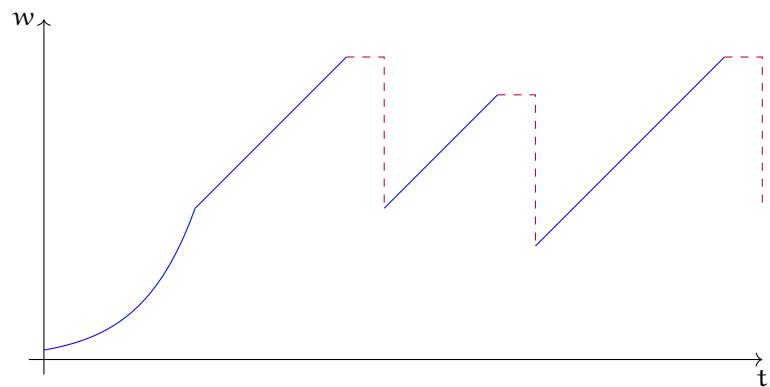


Figura 61: Andamento Fast retrasmitt/Fast recovery

8.2 Protocollo UDP (User Datagram Protocol)

Il protocollo UDP è un protocollo **connectionless**, cioè non c'è uno scambio preliminare prima di inviare i dati, ed è anche **non affidabile**, cioè se i segmenti vengono persi non vengono ritrasmessi.

8.2.1 Formato del pacchetto

La grandezza dell'header UDP è di 8byte tutti scritti in 2 righe da 4 byte. I campi dell'header sono i seguenti:

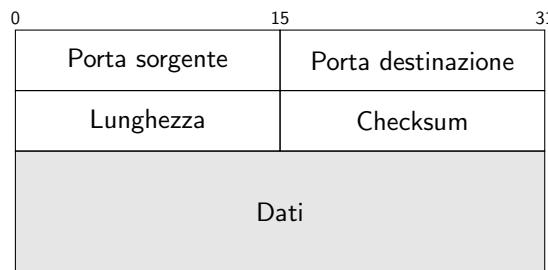


Figura 62: Formato del pacchetto UDP

- **Porta sorgente/destinazione:** Sono gli stessi campi presenti anche nel pacchetto TCP.
- **Lunghezza:** Indica la lunghezza del segmento UDP, cioè la somma della lunghezza dell'header e dei dati.
- **Checksum:** Serve per controllare la presenza di errori nel segmento. Bisogna avere un modo per capire se ci sono stati degli errori nella trasmissione dei dati per poter scartare il pacchetto.

Siccome UDP non ha il sequence number, sta all'applicazione che utilizza UDP gestire come ordinare i pacchetti.

9 Livello di rete

9.1 Protocollo IP (Internet Protocol)

9.1.1 Formato del pacchetto

Il formato del pacchetto IP è il seguente:

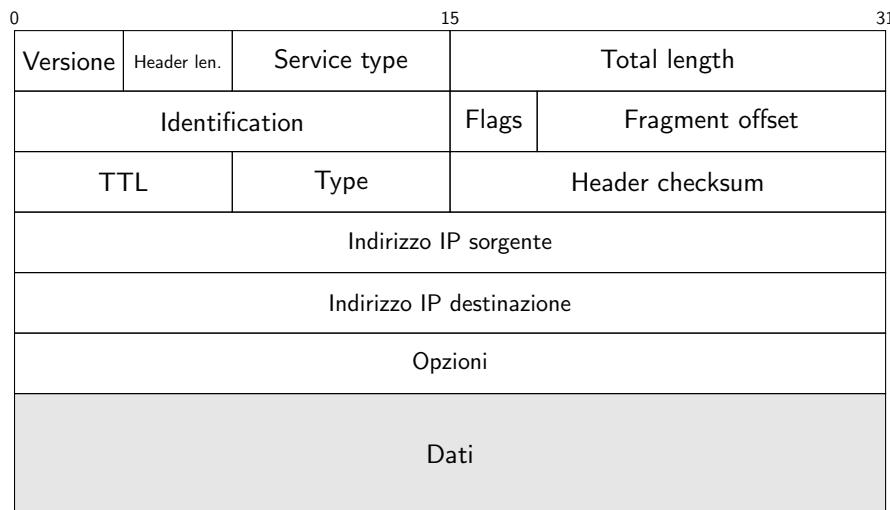


Figura 63: Formato del pacchetto IP

- **Versione** (4 bit): Indica la versione del protocollo IP utilizzato. Attualmente si utilizza la versione 4 (IPv4), la versione 6 (IPv6) è già implementata ma non ancora diffusa. Questo campo è stato pensato per poter supportare più versioni, siccome non si sapeva se il protocollo sarebbe stato utilizzato per molto tempo.
- **Lunghezza dell'header** (4 bit): Indica la lunghezza dell'header, che di solito è di 20 bit, ma può essere più lungo se ci sono delle opzioni.
- **Service type** (8 bit): È un codice che identifica una classe di servizio, cioè, come vengono gestiti i pacchetti nei buffer, ad esempio possono esserci delle priorità diverse per ogni buffer.
- **Lunghezza totale** (16 bit): Indica la lunghezza totale del pacchetto, cioè la somma della lunghezza dell'header e dei dati.
- **Identification** (16 bit): È un numero progressivo dato dalla sorgente (livello di rete) ad ogni pacchetto (non ha nessuna relazione con il numero di sequenza del TCP). Questo campo è utilizzato per il riassemblaggio dei pacchetti.
- **Flags** (3 bit): Sono dei bit che indicano se il pacchetto è frammentato o meno.
- **Fragment Offset** (13 bit): Indica l'offset del frammento rispetto al pacchetto originale (diviso per 8).
- **TTL (Time To Live)** (8 bit): È un contatore inizializzato dalla sorgente, che di solito vale 64 o 128, che indica il numero massimo di router che il pacchetto può attraversare, quindi ogni router che elabora tale pacchetto decrementa tale valore di 1, e se il valore diventa 0, il router scarta il pacchetto e manda un messaggio di errore alla sorgente.

In caso di problemi, la rete deve ricalcolare i percorsi e nel frattempo si possono creare dei **routing-loop** (momentanei). Il TTL permette di non caricare la rete in caso di routing-loop.

- **Type** (8 bit): È un codice che identifica il protocollo usato nei dati trasportati nel payload.

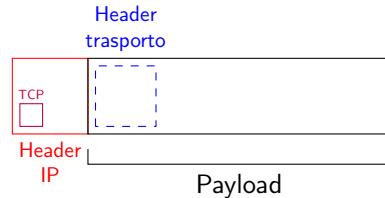


Figura 64: Type contiene il codice del protocollo di trasporto

- **Header Checksum** (16 bit): Serve per controllare la presenza di errori nell'header del pacchetto.

9.1.2 Frammentazione

(Da non confondersi con la segmentazione del livello trasporto). Data una tecnologia di trasmissione (scheda di rete, a livello data link) c'è una dimensione massima di trasmissione chiamata **MTU** (Maximum Transmission Unit).

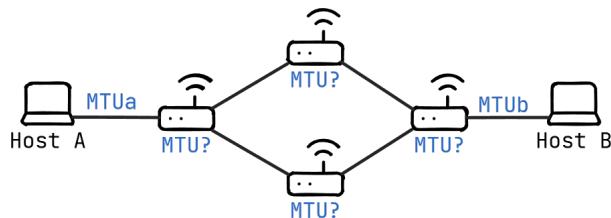


Figura 65: Maximum Transmission Unit

Se un pacchetto, durante il percorso, incontra un router che ha una MTU minore di quella del pacchetto, il router **frammenta** il pacchetto in pacchetti più piccoli con MTU compatibile con la linea di uscita. Questo processo è chiamato **frammentazione**.



Figura 66: Esempio di frammentazione

Solo l'host di destinazione (non il router della lan a cui appartiene l'host, nè i router intermedi) è responsabile del riassemblaggio dei pacchetti.

Quando la destinazione riceve un frammento fa partire un timer (250ms – 500ms), se non riceve tutti i frammenti entro la scadenza del timer, scarta tutti i frammenti, altrimenti ricompone il pacchetto e lo consegna a livello di trasporto.

I campi dell'header IP coinvolti nella frammentazione sono:

- **Identification**

- **Flags**

- Flag M:

- * 0: Se il pacchetto non è stato frammentato, oppure se è l'ultimo frammento
- * 1: Se il pacchetto è un frammento, tranne l'ultimo

- **Fragment Offset**

Esempio 9.1. Prendiamo ad esempio un pacchetto che ha come MTU = 5000byte e genera un pacchetto di 4000byte + 20byte di header IP.



Figura 67: Pacchetto non frammentato

Durante il percorso il pacchetto passa attraverso un router con link di uscita con MTU = 1420byte (1400 per il payload e 20 per l'header)



Figura 68: Pacchetto frammentato

Gli offset sono:

- Primo frammento: $\frac{0}{8} = 0$
- Secondo frammento: $\frac{1400}{8} = 175$
- Terzo frammento: $\frac{2800}{8} = 350$

9.2 Routing/instradamento

Un pacchetto potrebbe essere ovunque nella rete, quindi data la destinazione, il livello di rete deve **fare il possibile** (servizio **best effort**) per consegnare il pacchetto a tale destinazione.

Il **routing** è diviso in:

- **Consegna diretta:** La consegna di un pacchetto dall'host A all'host B è diretta e non passa per nessun router:

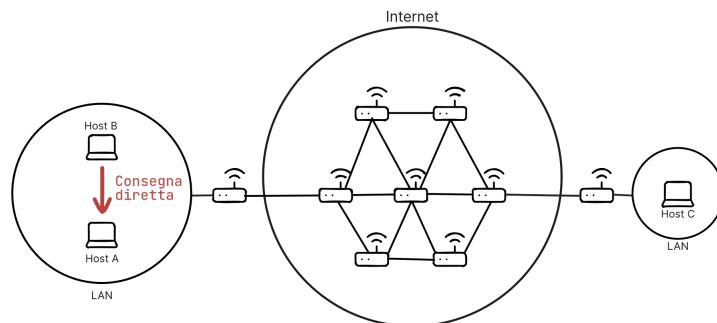


Figura 69: Consegnna diretta

- **Consegna indiretta:** La consegna di un pacchetto dall'host A all'host C è indiretta e passa per uno o più router:

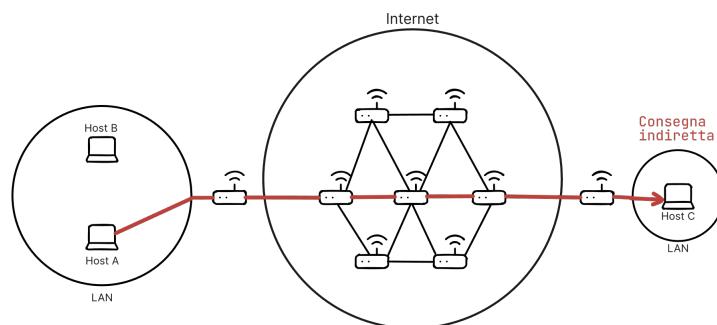


Figura 70: Consegnna indiretta

Un host conosce, oltre al proprio indirizzo IP, la maschera della rete a cui appartiene e conosce l'indirizzo IP del router di bordo della rete a cui appartiene:



Figura 71: Indirizzo delle interfacce di un router

Data una destinazione (IP_D) l'host confronta:

$$IP \& \text{Maschera} = IP_{\text{Rete}_1}$$

con

$$IP_D \& \text{Maschera} = IP_{\text{Rete}_2}$$

- Se i due valori sono uguali, allora la destinazione appartiene alla stessa rete dell'host, di conseguenza si può fare la consegna diretta.
- Se non lo sono, la destinazione appartiene ad un'altra rete, quindi si deve fare la consegna indiretta delegando la consegna al router.

Definizione 9.1. Il routing è il processo di scoperta del cammino "migliore" da una sorgente a tutte le possibili destinazioni.

Un router tiene memorizzata una **tabella do routing** che contiene

- **Indirizzo di destinazione**
- **Interfaccia di uscita**
- **Costo**



Figura 72: Tabella di routing

Il cammino migliore dipende dai criteri adottati da chi gestisce la rete (ISP). Alcuni esempi di criteri sono:

- **Distanza** (cammino più corto) in termini di numero di **hop** (router attraversati), (tutti gli archi hanno peso = 1)
- **Numero di chilometri**
- **Velocità di trasmissione** (velocità nominale del link)

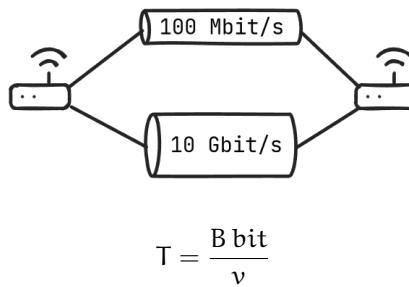


Figura 73: Velocità nominale del link

I pesi sono proporzionali all'inverso della banda.

- **Livello di congestione**

Questo processo viene astratto con l'utilizzo dei **grafi**:



Figura 74: Diagramma di rete

I router sono i **nodi** e i collegamenti sono gli **archi** del grafo:

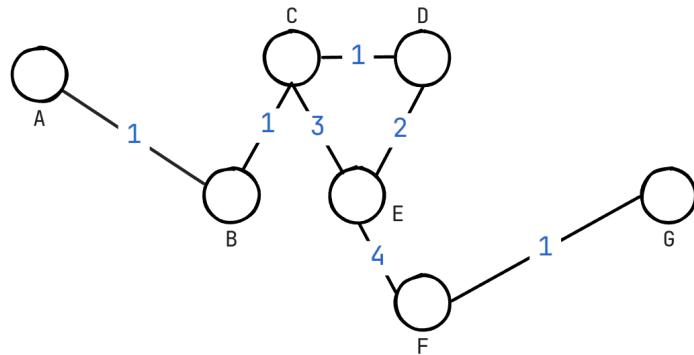


Figura 75: Grafo di rete

Agli archi viene associato un numero detto **costo** (o **peso**) che caratterizza l'arco stesso in base al criterio utilizzato.

Definizione 9.2. Il costo di un cammino tra un nodo i e un nodo j è la somma dei pesi degli archi che appartengono al cammino. Di conseguenza il routing è il calcolo del cammino a **costo minimo**.

9.2.1 Algoritmi per il calcolo dei cammini minimi

Ad ogni algoritmo corrisponde un protocollo (diverso dal protocollo IP) che viene trasportato dal protocollo IP. Periodicamente, ogni router, invia ai propri vicini diretti un messaggio di aggiornamento (che cambia in base al tipo di algoritmo) che si distingue dagli altri messaggi, perché nell'header IP c'è il campo **type**. Una volta che un router riceve un messaggio di aggiornamento, aggiorna la propria tabella di routing e inoltra il messaggio ai propri vicini, aggiornando così tutti i componenti della topologia della rete in caso di cambiamento.

Gli algoritmi per il calcolo del cammino minimo in un grafo sono di due tipi:

- **Stato dei collegamenti** (Link state): È un algoritmo centralizzato.

Ogni nodo della rete ha a disposizione l'intera topologia della rete (il grafo con nodi e archi e i pesi degli archi). Ogni nodo calcola autonomamente i cammini minimi (centralizzato). Una volta che tutti i router hanno calcolato il cammino minimo, essi sanno quale sarà il prossimo hop per ogni destinazione.

Uno degli algoritmi più utilizzati è l'algoritmo di **Dijkstra**.

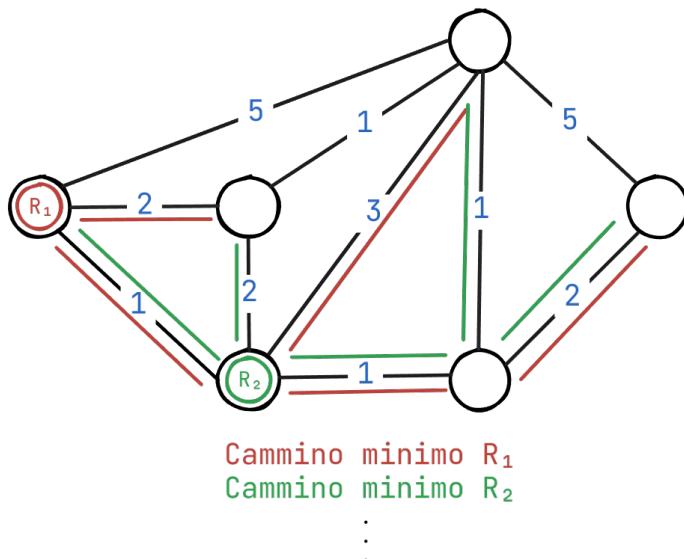


Figura 76: Calcolo dei cammini minimi

I nodi si scambiano messaggi periodici per controllare se sono ancora raggiungibili (**keep alive**) o messaggi in caso di guasti o cambio di topologia.

Il principale protocollo basato su link state è **OSPF** (Open Shortest Path First).

- **Vettori distanza** (Distance vector): È un algoritmo distribuito

Considerando una topologia in cui è presente un nodo i con un nodo vicino diretto (adiacenti con collegamento diretto) j e un nodo k non adiacente ad i , definiamo il **costo dell'arco** da i a j come $c(i,j)$.

Il costo di un cammino dal nodo i al nodo k viene rappresentato come:

$$D(i,k) = \sum_{l,n \in \text{cammino}} c(l,n)$$

Osservazione:

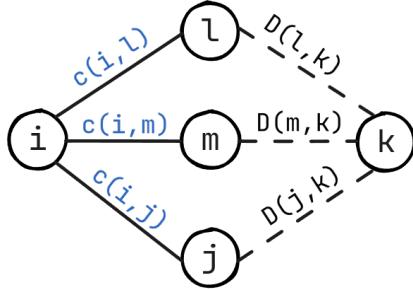


Figura 77: Distanza tra due nodi

Consideriamo un nodo i e una destinazione qualsiasi k . Assumiamo che il nodo i abbia un insieme finito di vicini diretti per cui i conosce il costo del collegamento diretto. Supponendo che ogni vicino di i conosca il cammino minimo alla destinazione D e questa informazione sia esposta a i , allora il costo del cammino minimo da i a k è:

$$D(i, k) = \min \left(\begin{cases} c(i, l) + D(l, k) \\ c(i, m) + D(m, k) \\ c(i, j) + D(j, k) \end{cases} \right)$$

In generale:

$$D(i, k) = \min_{v \in \text{vicini diretti di } i} (c(i, v) + D(v, k))$$

Il next hop sarà il nodo v che permette di minimizzare la distanza tra il nodo i e la destinazione k .

Questo funziona per ogni nodo i e per ogni destinazione k e permette di non dover conoscere l'intera topologia della rete, ma soltanto i vicini diretti.

L'algoritmo distance vector è l'algoritmo di **Bellman-Ford**:

```

1 i <- nodo corrente
2
3 // Fase di inizializzazione
4 for d in destinazioni
5   if d = vicino_diretto
6     D(i,d) <- c(i,d)
7   else
8     D(i,d) <- infinito
9
10 // Fase di scambio di informazioni periodiche
11 // Ogni 3 min (dipende dall'implementazione)
12 // Il nodo invia ai propri vicini il distance vector.
13 // Il distance vector e' l'unione delle colonne
14 // "destinazione" e "costo" della tabella di routing del nodo.
15 for v in vicini_diretti
16   for d in destinazioni
17     send(D(i,d), j)
18

```

```

19 // Con i distance vector ricevuti dai vicini, il nodo aggiorna
20 // la tabella di routing per ogni destinazione con la formula:
21 // D(i,k) = min(c(i,v) + D(v,d))
22 for d in destinazioni
23   for v in vicini_diretti
24     D(i,d) = min(D(i,d), c(i,v) + D(v,d))

```

Si può dimostrare che:

- Se il grafo è completamente connesso (ciascun nodo può raggiungere un qualsiasi altro nodo con un cammino che sta all'interno del grafo) e
- Se i pesi degli archi sono tutti positivi

allora l'algoritmo di Bellman-Ford **converge alla soluzione ottima** (quella in cui i nodi hanno nelle tabelle di routing i cammini minimi effettivi per ogni destinazione).

Converge perché più il grafo è grande più ha bisogno di tempo per propagare i messaggi di aggiornamento, di conseguenza all'inizio i valori nelle tabelle di routing sono diversi da quelli ottimi, ma con il tempo convergono alla soluzione ottima. Il **numero di iterazioni** è proporzionale al **diametro** del grafo (il numero di hop massimo tra due nodi all'interno del grafo).

Esempio 9.2. Consideriamo il seguente grafo focalizzandoci sul nodo A e la sua tabella di routing verso qualsiasi destinazione:

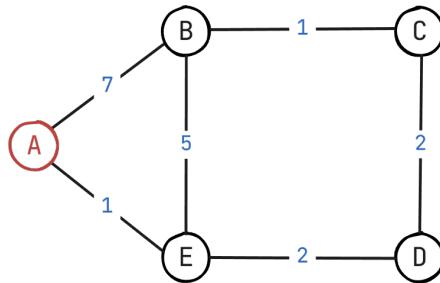


Figura 78: Grafo di rete

Al momento dell'inizializzazione il nodo A vede soltanto i vicini diretti.

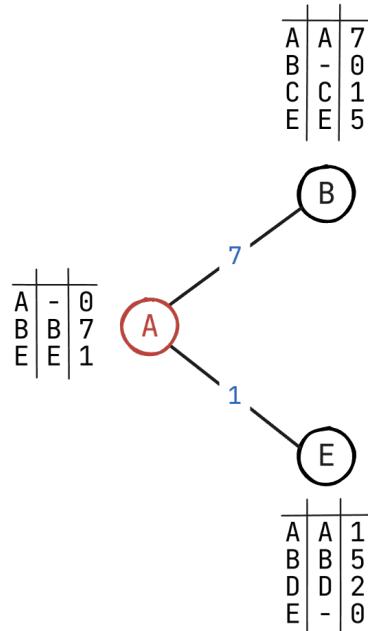


Figura 79: Tabelle di routing all'inizio

Dal punto di vista del nodo A, alla prima iterazione, vengono ricevuti i distance vector di B e di E che sono:

	Dst	Cst		Dst	Cst
B =	A	7	E =	A	1
	B	0		B	5
	C	1		D	2
	D	5		E	0

Per la destinazione B il nodo A calcola:

$$D(A, B) = \min \left(\begin{cases} c(A, B) + D(B, B) = 7 + 0 = 7 \\ c(A, E) + D(E, B) = 1 + 5 = 6 \end{cases} \right)$$

Siccome nella tabella di A si aveva un costo da A → B di 7, avendo trovato un costo minimo attraverso E, il nodo A aggiorna la tabella per la destinazione B con un costo di 6 e next hop E.

La nuova tabella di routing del nodo A diventa:

Dst	Next hop	Cst	Dst	Next hop	Cst
A	-	0	A	-	0
B	B	7	→ B	E	6
E	E	1	⋮	⋮	⋮

L'aggiornamento avviene per ogni destinazione contenuta nei distance vector ricevuti. Ad esempio, per la destinazione D ha un'unica riga che dice:

$$D(A, D) = c(A, E) + D(E, D) = 1 + 2 = 3$$

Il nodo A aggiorna la tabella di routing per la destinazione D con un costo di 3 e next hop E.

Dst	Next hop	Cst	Dst	Next hop	Cst
A	-	0	A	-	0
B	E	6	B	E	6
:	:	:	D	E	3
			:	:	:

Il processo continua per ogni destinazione e per ogni nodo.

Finché le tabelle vengono sincronizzate, si avranno tabelle sbagliate che possono portare a **loop** nella rete (per evitarli c'è il TTL).

- **Software Defined Network (SDA)**: C'è un'entità centrale che sovraintende l'intera rete e decide i cammini minimi.