Reti di calcolatori

VR443470

dicembre 2022

Indice

1 Introduzione

Internet è una rete di calcolatori che interconnette miliardi di dispositivi di calcolo in tutto il mondo. Gli strumenti in una rete, per esempio cellulari o computer, vengono chiamati **host** (*ospiti*) o **sistemi periferici** (*end system*). Essi sono connessi tra di loro tramite una **rete di collegamenti** (*communication link*) e **commutatori di pacchetti** (*packet switch*). I collegamenti possono essere di vario tipo: cavi coassiali, fili di rame, fibre ottiche e onde elettromagnetiche.
Ogni collegamento detiene una sua **velocità di trasmissione** (*transmission rate*), ovvero la velocità di trasmissione dei dati. L'**unità di misura** è il bit per secondo (bit/secondo, *bps*).

L'insieme delle informazioni, o dati, che vengono inviati o ricevuti prendono il nome di **pacchetto**. L'**obbiettivo** di un commutatore di pacchetti è quello di ricevere un pacchetto che arriva da un collegamento in ingresso e di ritrasmetterlo su un collegamento d'uscita. I due principali commutatori di internet sono: *router* e i commutatori a livello di collegamento (*link-layer switch*). La sequenza di collegamenti e di commutatori di pacchetto attraversata dal singolo pacchetto è nota come **percorso** o **cammino** (*route* o *path*).

Quindi, in sintesi, le definizioni più rilevanti sono:

- **Internet.** Rete di calcolatori che interconnette i dispositivi di calcolo di tutto il mondo.
- **Host (*o* sistemi periferici).** Strumenti in una rete, per esempio computer.
- **Rete di collegamenti (*communication link*) e commutatori di pacchetto (*packet switch*).** Collega vari *host*, per esempio cavi coassiali o fili di rame.
- **Velocità di trasmissione (*transmission rate*).** È la velocità di trasmissione dei dati e solitamente la sua **unità di misura** è il bit per secondo, cioè *bps*.
- **Pacchetto.** Insieme delle informazioni che vengono inviate e ricevute.
- **Obbiettivo** commutatore di pacchetti. Ricevere un pacchetto proveniente da un collegamento in ingresso e ritrasmetterlo su un collegamento d'uscita. Per esempio i *router*.
- **Percorso (*route*) o cammino (*path*).** Sequenza di collegamenti e di commutatori di pacchetto attraversata dal singolo pacchetto.

2 ISP, TCP/IP, commutazione dei pacchetti e ritardi

2.1 ISP

I sistemi periferici accedono ad Internet tramite un servizio chiamato **Internet Service Provider** (ISP). Con **provider** si intende un insieme di commutatori di pacchetto e di collegamenti. Gli **obbiettivi** degli ISP è fornire ai sistemi periferici svariati tipi di accesso alla rete, come quello residenziale a larga banda (e.g. DSL), quello in rete locale ad alta velocità, quello senza fili (*wireless*) e in mobilità.
Esistono 3 **tipi** di livelli di ISP:

Livello 1. *Internazionale* (Telecom, TIM, ...);

Livello 2. *Nazionale* (Fastweb);

Livello 3. *Locale* (solitamente per professionisti).

Più è basso il livello, più gli ISP sono costituiti da *router* ad alta velocità interconnessi tipicamente tramite fibra ottica.

2.2 TCP/IP

I sistemi periferici, i commutatori di pacchetto e altre parti di Internet fanno uso di **protocolli** che controllano l'invio e la ricezione di informazioni all'interno della rete. Esistono **due principali protocolli** Internet: ***Transmission Control Protocol*** (TCP) e ***Internet Protocol*** (IP). In particolare, l'IP specifica il formato dei pacchetti scambiati tra router e sistemi periferici. Generalmente ci si riferisce a questi due protocolli tramite il nome collettivo TCP/IP.

2.3 Commutazione dei pacchetti

Esistono due diversi approcci per spostare quantità di dati all'interno di una rete: la **commutazione di circuito** e la **commutazione di pacchetto**.

Commutazione di circuito

Nella **commutazione di circuito** le risorse richieste lungo un percorso (buffer e velocità di trasmissione sui collegamenti) sono **riservate** per l'intera durata della sessione di comunicazione.
Vantaggi:

- ✓ **Velocità costante** durante il collegamento poiché le risorse sono riservate e non condivise. Questo si traduce in un **ritardo contenuto**.

Svantaggi:

- ✗ **Spreco di risorse** poiché i circuiti sono inattivi durante i periodi di silenzio, ovvero nei periodi in cui non c'è comunicazione;
- ✗ **Complicazioni** nello stabilire circuiti e nel riservare larghezza di banda *end-to-end*.

In questo contesto, i ritardi possono essere causati solamente per tre motivi: (1) a causa dell'instaurazione del circuito, (2) a causa della distanza tra sorgente e destinazione, (3) a causa della trasmissione vera e propria.

Commutazione di pacchetto

Nella **commutazione di pacchetto** la sorgente divide i messaggi in parti più piccole, ovvero in **pacchetti** assegnando a ciascuno un'intestazione. I pacchetti viaggiano attraverso collegamenti e commutatori di pacchetto dalla sorgente alla destinazione.
Vantaggi:

- ✓ **Ottimizzazione** delle risorse poiché c'è una condivisione di esse nei momenti di inattività.

Svantaggi:

- ✗ **Possibile perdita** di pacchetti nel caso in cui un buffer di un nodo sia saturo. Questo comporta un buffer overflow e una conseguente perdita;
- ✗ **Ritardo dovuto a *store and forward* e numero di nodi intermedi.** A causa dello *store and forward*, ogni nodo deve attendere di ricevere l'intero pacchetto prima di ritrasmetterlo. Inoltre, con l'aumentare dei nodi intermedi, il ritardo aumenta.
  (approfondimento *store and forward*)

2.4 Tipologie di ritardi

Esistono diverse tipologie di ritardo perché quando un pacchetto parte da un *host* (sorgente), passa attraverso una serie di *router* e conclude il viaggio in un altro *host* (destinazione). Questo comporta un ritardo in ciascun nodo (*host* o *router*). I principali ritardi sono: **ritardo di elaborazione**, **ritardo di accodamento**, **ritardo di trasmissione** e **ritardo di propagazione**. L'insieme di questi ritardi è chiamato **ritardo totale di nodo** (*nodal delay*).

Ritardo di elaborazione

Il tempo richiesto per esaminare l'intestazione del pacchetto e per determinare dove dirigerlo fa parte del **ritardo di elaborazione** (*processing delay*). Per dirigere si intende il tempo che impiega il *router* a determinare la sua parte di uscita.

Ritardo di accodamento

Una volta in coda, il pacchetto subisce un **ritardo di accodamento** (*queuing delay*) mentre attende la trasmissione sul collegamento. La lunghezza di tale ritardo dipenderà dal numero di pacchetto precedentemente arrivati, accodati e in attesa di trasmissione sullo stesso collegamento. In altre parole, è il tempo speso nel *buffer* prima che il pacchetto venga ritrasmesso.

Ritardo di trasmissione

Data L la lunghezza del pacchetto, in bit, e R *bps* la velocità di trasmissione del collegamento dal *router* A al *router* B, il **ritardo di trasmissione** (*transmission delay*) sarà $L \div R$. Questo è il tempo richiesto per trasmettere tutti i bit del pacchetto sul collegamento.
Più semplicemente, dipende dalla velocità di trasmissione e dalla dimensione del pacchetto ed è possibile sintetizzarlo con la formula:

$$t_{\text{trasm}} = \frac{\text{dim\_pacchetto}}{\text{velocità\_trasmissione}}$$

Ritardo di propagazione

Una volta immesso sul collegamento, un bit deve propagarsi fino al *router* B. Il tempo impiegato è il **ritardo di propagazione** (*propagation delay*). In altre parole è il tempo impiegato per percorrere la distanza verso il *router* successivo.

Strumenti di misurazione

Esistono diversi **strumenti per misurare il ritardo**:

- **PING.** Dato un indirizzo di destinazione, il calcolatore manda una serie di messaggi e misura il tempo che intercorre tra l'invio e la ricezione della risposta, chiamato anche *Rount Trip Time* (RTT).
- **TRACEROUTE.** Misura il *Round Trip Time* tra la sorgente e **tutti** gli apparati di rete intermedi.

2.5 Sintesi

- ***Internet Service Provider* (ISP).** Strumento utilizzato dai sistemi periferici per accedere ad Internet.
- ***Provider*.** Insieme di commutatori di pacchetto e di collegamenti, solitamente è un'azienda che fornisce servizi.
- **Obbiettivi ISP.** Fornire vari tipi di accesso alla rete ai dispositivi che si collegano (e.g. DSL, *wireless*, ecc.).
- **Tipi di ISP:**
  - **Livello 1.** *Internazionale* (Telecom, TIM, ...);
  - **Livello 2.** *Nazionale* (Fastweb);
  - **Livello 3.** *Locale* (solitamente per professionisti).
- **Definizione TCP/IP.** Protocolli più famosi utilizzati dai sistemi periferici, i commutatori di pacchetto e altre parti di Internet. N.B. il protocollo IP specifica il formato dei pacchetti scambiati tra *router* e sistemi periferici.
- **Definizione commutazione di circuito.** Le risorse sono riservate per l'intera comunicazione.
  - **Vantaggio commutazione di circuito.** Velocità costante grazie ad un canale dedicato e quindi ritardo contenuto.
  - **Svantaggio commutazione di circuito.** Spreco di risorse in caso di silenzi durante la comunicazione.
  - **Causa dei ritardi nella commutazione di circuito.** I motivi possono essere tre:
    I Instaurazione del circuito;
    II Distanza tra sorgente e destinazione;
    III Trasmissione vera e propria della comunicazione.
- **Definizione commutazione di pacchetto.** La sorgente divide i messaggi in parti più piccole chiamate **pacchetti**.
  - **Vantaggio commutazione di pacchetto.** Ottimizzazione delle risorse poiché c'è una condivisione durante l'inattività.
  - **Svantaggi commutazione di circuito.** Eventuale perdita di pacchetti nel caso in cui un nodo intermedio abbia il *buffer* saturo (generazione di *buffer overflow*); ritardo causato da *store and forward* poiché ogni pacchetto per essere inoltrato deve essere completamente trasmesso; all'aumentare dei nodi intermedi, il ritardo aumenta.
- **Ritardo di elaborazione (*processing delay*).** Tempo impiegato dal *router* per esaminare l'intestazione del pacchetto e determinare l'uscita.
- **Ritardo di accodamento (*queuing delay*).** Tempo impiegato dal pacchetto all'interno della coda del buffer del *router*.

- **Ritardo di trasmissione (*transmission delay*).** Tempo che dipende dal rapporto tra la dimensione del pacchetto e la velocità di trasmissione.
- **Ritardo di propagazione (*propagation delay*).** Tempo impiegato per percorrere la distanza verso il *router* successivo.
- **Strumenti per la misurazione del ritardo.** I due strumenti sono "PING" e "TRACEROUTE". La differenza è che PING misura il RTT tra sorgente e destinazione, mentre il TRACEROUTE misura il RTT tra sorgente e ogni nodo intermedio.

3 Tecnica di load balancing, Throughput e collo di bottiglia

3.1 Tecnica di load balancing

Nel momento in cui il **mittente** (sorgente) calcola il **percorso migliore** per inviare i suoi dati al destinatario, può accadere che **trovi due o più strade identiche**. Con quest'ultimo termine si intende che i percorsi con il costo minimo, e quindi i più efficienti, siano due o più. In questo caso, viene applicata la tecnica di load balancing.

La tecnica di **load balancing** prevede di suddividere il carico dei pacchetti in tutti i percorsi migliori trovati. In questo modo, la comunicazione non avrà un unico percorso sovraccaricato, ma il carico sarà diviso tra più percorsi.

3.2 Throughput

Un'altra misura che influisce sulle prestazioni in una rete di calcolatori è il throughput *end-to-end*. Esistono **due tipi di throughput**:

- **Throughput istantaneo**, in ogni istante di tempo p, è la velocità (misurata in bit per secondo, *bps*) alla quale il destinatario B sta ricevendo il file.
- **Throughput medio** è dato da una formula specifica. Se l'oggetto da inviare è formato da F bit e il trasferimento richiede T secondi affinché il destinatario B riceva tutti gli F bit, allora il throughput medio del trasferimento dell'oggetto da inviare è di

$$\frac{F}{T} \textit{ bit per secondo}$$

3.3 Collo di bottiglia

Quindi, la connessione *end-to-end* presenta criticità nel momento in cui più dispositivi dividono la strada tra sorgente e destinazione. Si parla, infatti, di **collo di bottiglia** (*bottleneck link*), nel momento in cui la velocità di trasferimento viene diminuita a causa di un canale più piccolo o a causa di un dispositivo con banda minore.

4 Architettura a livelli e incapsulamento

4.1 Architettura a livelli

Un'**architettura a livelli** consente di manipolare una parte specifica e ben definita di un sistema articolato e complesso.

Questa struttura è data dal fatto che fin quando ciascun **livello** (*layer*, o strato) fornisce lo stesso servizio allo strato superiore e utilizza gli stessi servizi dello stato inferiore, la parte rimanente del sistema rimane invariata al variare dell'implementazione a quel livello.

I **servizi** vengono offerti da un determinato livello a quello superiore, ovvero si tratta del **modello di servizio** (*service model*) di un livello. Più in generale, **ogni livello fornisce il suo servizio** effettuando determinate azioni all'interno del livello stesso e utilizzando i servizi del livello immediatamente inferiore.

Nel caso di sistemi grandi e complessi, che vengono costantemente aggiornati, la capacità di cambiare l'implementazione di un servizio senza coinvolgere altre componenti del sistema costituisce un ulteriore importante vantaggio legato alla stratificazione. Quindi, i pro e i contro di questa architettura sono:

- **Vantaggio**
  - Il sistema è strutturato e dunque permette il trattamento dei componenti senza stravolgere l'intera architettura o struttura.
- **Svantaggi**
  - Possibilità di duplicazione delle funzionalità tra due o più livelli, ovvero che un livello cloni le caratteristiche del livello inferiore;
  - Possibilità che la funzionalità presente ad un livello possa richiedere informazioni presenti solo ad un altro livello.

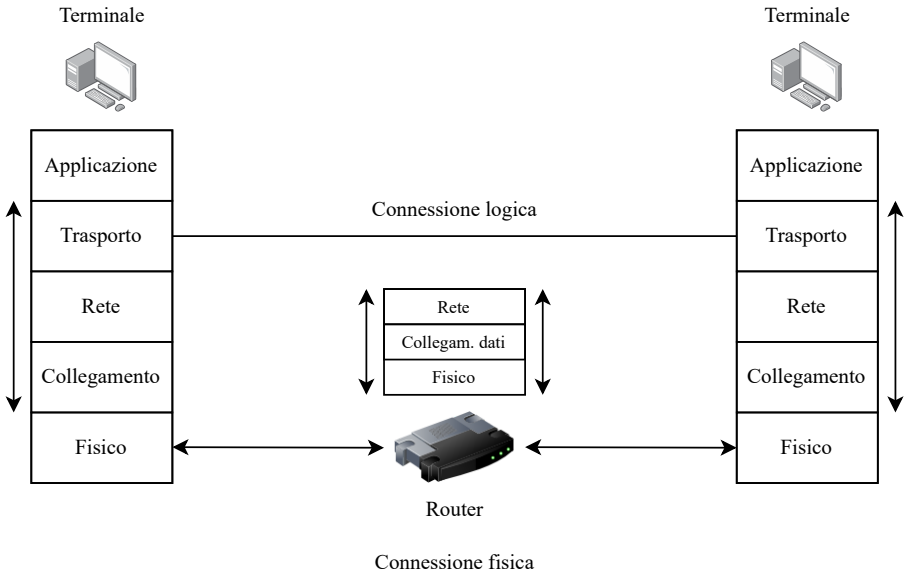
Ogni livello ha un **protocollo** e l'insieme dei protocolli vengono definiti **pila di protocolli** (*protocol stack*). La pila di protocolli di Internet consiste di cinque livelli:

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

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

4.2 Incapsulamento

L'**incapsulamento** (o imbustamento) è un modus operandi applicato nel momento in cui si deve inviare un messaggio.



La comunicazione avviene nel seguente modo:

1. Parte nel **livello di applicazione** del host mittente il quale crea un **messaggio a livello di applicazione** (*application-layer message*) concatenando informazioni aggiuntive, o meglio le informazioni di intestazione. Alla fine del processo di creazione, il messaggio viene passato al livello inferiore, quello di trasporto;

2. A **livello di trasporto** vengono aggiunte altre informazioni di intestazione. Le intestazioni di applicazione e trasporto formano il **segmento a livello di trasporto** (*transport-layer segment*) che incapsula il messaggio a livello di applicazione. Infine, il livello di trasporto passa il messaggio al livello di rete;

3. A **livello di rete** vengono aggiunte informazioni come gli indirizzi dei sistemi periferici di sorgente e di destinazione. Facendo così viene creato un **datagramma a livello di rete** (*network-layer datagram*). Infine, il messaggio viene passato al livello collegamento (*link*);

4. A **livello di collegamento** le informazioni aggiuntive creano un **frame a livello di collegamento** (*link-layer frame*);

5. A **livello fisico** vengono inviati i dati al router e qui termina l'incapsulamento.

Per cui ad ogni livello, il pacchetto ha due tipi di campi: l'intestazione e **payload** (il carico utile trasportato). Il payload è tipicamente un pacchetto proveniente dal livello superiore.

5 Indirizzi IP

5.1 Indirizzi IP

Un indirizzo IP consente di rendere **identificativo** e **univoco** un host all'interno della rete. Gli indirizzi IP vengono **rappresentati** con 32 bit e utilizzando una **notazione decimale puntata**. Prendendo in considerazione l'architettura di rete spiegata nel capitolo precedente, l'IP si posiziona al livello di rete, nel quale viene aggiunto al messaggio da inviare.

La **notazione decimale puntata** è una rappresentazione degli indirizzi IP che facilita la lettura. Il modus operandi per ottenere tale notazione è il seguente:

1. Dividere i bit in 4 gruppi, ovvero 8 bit per ciascun gruppo;
2. Traduzione di ogni gruppo da binario a decimale;
3. Divisione di ogni gruppo da un punto.

Negli indirizzi IP è importante dividere il prefisso dal suffisso poiché ogni parte ha un significato diverso:

- **Prefisso**, identifica una rete all'interno di Internet;
- **Suffisso**, identifica un host all'interno della rete.

Non esiste un numero specifico di indirizzi IP per il prefisso e per il suffisso. Questo perché dipendono entrambi dalla grandezza della rete; più è grande la rete e meno bit ha di prefisso.

Un ***esempio***: 157.27.12.63/16, dove 157.27 identifica il prefisso e 12.63 il suffisso.

5.2 Maschera e blocco CIDR

Per identificare il numero di bit presenti nel prefisso, il calcolatore utilizza una sequenza di 32 bit in cui i bit del prefisso sono posti tutti a uno e i restanti a zero. Questo metodo si chiama maschera e un esempio di **maschera** 16 (notazione: /16):

11111111111111110000000000000000

Che rappresenta l'indirizzo: 255.255.0.0

Per cui si può affermare che la maschera **identifica** la grandezza della rete. Pensandoci, più è grande la maschera e più piccola è la rete visto che c'è una stretta relazione con il prefisso di un indirizzo IP.

Un **blocco CIDR** (*Classless Inter-Domain Routing*) è un intervallo di indirizzi IP che sono disponibili nella propria rete.

5.3 Esercizio di traduzione e numero host

Dato il seguente indirizzo in notazione binaria:

$$11100111 \quad 11011011 \quad 10001011 \quad 01101111$$

Si rappresenta in notazione decimale puntata.

Per **prima cosa** si esegue la traduzione di ogni gruppo da binario a decimale:

- $11100111 \longrightarrow 2^7 \cdot 1 + 2^6 \cdot 1 + 2^5 \cdot 1 + 2^4 \cdot 0 + 2^3 \cdot 0 + 2^2 \cdot 1 + 2^1 \cdot 1 + 2^0 \cdot 1 = 231$
- $11011011 \longrightarrow 2^7 \cdot 1 + 2^6 \cdot 1 + 2^5 \cdot 0 + 2^4 \cdot 1 + 2^3 \cdot 1 + 2^2 \cdot 0 + 2^1 \cdot 1 + 2^0 \cdot 1 = 219$
- $10001011 \longrightarrow 2^7 \cdot 1 + 2^6 \cdot 0 + 2^5 \cdot 0 + 2^4 \cdot 0 + 2^3 \cdot 1 + 2^2 \cdot 0 + 2^1 \cdot 1 + 2^0 \cdot 1 = 139$
- $01101111 \longrightarrow 2^7 \cdot 0 + 2^6 \cdot 1 + 2^5 \cdot 1 + 2^4 \cdot 0 + 2^3 \cdot 1 + 2^2 \cdot 1 + 2^1 \cdot 1 + 2^0 \cdot 1 = 111$

E infine si riscrive la notazione in notazione decimale puntata:

$$231.219.139.111$$

Adesso viene eseguita la conversione binaria della notazione decimale puntata del seguente indirizzo:

$$221.34.255.82$$

Per **prima cosa** si esegue la traduzione di ogni gruppo. La traduzione non è banale, difatti si prenderà ciascun gruppo, si dividerà per due e nella colonna di destra verranno scritti i riporti. Infine, il numero binario sarà scritto dal numero di riporto del numero più basso, fino al numero più alto:

$221 \div 2$	1
$110 \div 2$	0
$55 \div 2$	1
$27 \div 2$	1
$13 \div 2$	1
$6 \div 2$	0
$3 \div 2$	1
$1 \div 2$	1

E così via. Unica accortezza da **ricordare** che se il numero di bit fosse meno di 8, si aggiungono zeri nella parte più significativa.

Dopo alcuni calcoli l'indirizzo IP in binario è:

$$11011101 \quad 00100010 \quad 11111111 \quad 01010010$$

Una rete con un suffisso di /20, quanti host contiene? Dato un indirizzo IP da 32 bit, se il suffisso ha 20 bit, allora: $2^{32} \div 2^{20} = 2^{12}$. Una rete con un suffisso di 20 bit, avrà un prefisso di 12 bit, ovvero $2^{12} \longrightarrow 4096$ indirizzi.

6 Indirizzi IP privati

6.1 Indirizzi IP privati

Gli indirizzi IP riservati sono indirizzi IP che non possono essere assegnati a nessun host. In particolare, sono:

- **Indirizzo di rete**: tutti i bit del suffisso sono posti a zero;
- **Indirizzo di Direct Broadcast**: tutti i bit del suffisso sono posti a uno;
- **Indirizzo con tutti i bit a zero**;
- **Indirizzo con tutti i bit a uno**;

6.2 Esercizio subnetting, creazione sottoreti partendo da un blocco di indirizzi

La tecnica di ***subnetting*** consiste nel suddividere una rete in più rete, chiamate sottoreti. L'esercizio fornisce un indirizzo IP:

180.190.0.0/16

E l'obbiettivo è quello di creare due sottoreti di grandezza identica.

Per **prima cosa** si deve effettuare la conversione da notazione decimale puntata a notazione binaria:

10110100 10111110 00000000 00000000

A questo punto è possibile creare due sottoreti come richiesto dall'esercizio.

Per farlo si identifica il primo bit utile nell'indirizzo binario. Esso è possibile trovarlo escludendo il prefisso (/16). Quindi, si conta dal bit più significativo 16 bit, escluso sé stesso, arrivando al bit numero 17, ovvero il più significativo del terzo gruppo:

10110100 10111110 ***x***0000000 00000000

Al posto della ***x*** verrà sostituito lo zero per creare la prima rete e l'uno per creare la seconda rete. Gli indirizzi saranno:

Ind. bin. della **prima rete**:	10110100	10111110	**0**0000000	00000000
Ind. bin. della **seconda rete**:	10110100	10111110	**1**0000000	00000000

Il **prefisso** delle sottoreti sarà aumentato di un solo bit, per cui saranno /17.

La **traduzione in decimale** sarà:

- **Prima rete:** 180.190.0.0/17
- **Seconda rete:** 180.190.128.0/17

La **maschera** delle sottoreti sarà formata da 17 bit posti a 1 partendo dal più significativo:

$$11111111 \quad 11111111 \quad 10000000 \quad 00000000$$

E la **traduzione della maschera** in notazione decimale puntata sarà:

$$255.255.128.0$$

Infine, la **dimensione dei blocchi** di rete:

$$\textbf{Pre-subnetting}: \quad 2^{32} \div 2^{16} = 2^{16} \quad \longrightarrow \quad 65'536 \text{ indirizzi}$$

$$\textbf{Post-subnetting}: \quad 2^{32} \div 2^{17} = 2^{15} \quad \longrightarrow \quad 32'768 \text{ indirizzi}$$

7 Socket e protocolli a livello di trasporto: TCP e UDP

7.1 Socket

I sistemi periferici collegati a Internet forniscono un'**interfaccia socket** (*socket interface*), che specifica come un programma eseguito su un sistema periferico possa chiedere a Internet di recapitare dati a un programma eseguito su un altro sistema periferico.

In altre parole, l'interfaccia socket è un insieme di regole che il programma mittente deve seguire in modo che i dati siano recapitati al programma di destinazione.

È importante anche sapere l'esistenza delle **API** (*Application Programmin Interface*) tra l'applicazione e la rete, poiché l'interfaccia socket non è altro che un'interfaccia di programmazione con cui le applicazioni di rete vengono costruite.

Infine, esistono due tipi di processi:

- Il **client**, ovvero colui che avvia la comunicazione e ha un IP dinamico;
- Il **server**, ovvero colui che attende di essere contattato per iniziare la sessione e ha un IP statico.

7.2 Protocolli nel livello di trasporto

Dopo il livello applicativo, si trova il livello di trasporto. Esso può utilizzare due possibili **protocolli** nelle reti TCP/IP: **TCP** e **UDP**.

7.2.1 Protocollo TCP

Il protocollo TCP prevede la fornitura di un servizio **orientato alla connessione** (*Connection oriented communication*) e il **trasporto affidabile dei dati**.

Le **caratteristiche** di questo protocollo sono:

- **Servizio orientato alla connessione**. Questa caratteristica deriva dal fatto che vengono scambiate delle informazioni di controllo prima che i messaggi veri e propri vengano processati dal livello applicativo. Questo scambio di messaggi prende il nome di **handshaking**.
  Successivamente, *se tutto è andato a buon fine*, si instaura una **connessione TCP** tra le socket dei due processi. La connessione viene chiamata ***full-duplex*** ovvero i due processi possono scambiarsi i messaggi contemporaneamente sulla connessione. Infine, per terminare la connessione, il mittente e il destinatario si **scambiano** alcuni **messaggi**.
- **Servizio di trasferimento affidabile**. Il protocollo è affidabile poiché i vari controlli effettuati permettono di:
  - Non perdere dati;
  - Inviarli nel giusto ordine;
  - Evitare duplicazioni.

Inoltre, questo protocollo implementa un meccanismo di **controllo della congestione**. Nel caso in cui si manifestasse, eseguirebbe una "strozzatura" del processo d'invio.

Infine, il protocollo negli ultimi anni implementa anche il ***secure sockets layer*** (**SSL**). Ovvero un servizio aggiuntivo che consente la cifratura dei messaggi. Attenzione che questo servizio deve essere attivo sia lato client che lato server, altrimenti si rischia che una delle due parti non possa decifrare il messaggio ricevuto.

7.2.2 Protocollo UDP

Al contrario del protocollo TCP, UDP è un protocollo di trasporto **leggero**, **rapido** e **minimalista**. Esso ***non*** necessita di connessione, per cui non esegue alcun handshaking e di conseguenza **non** fornisce neanche un **trasferimento dati affidabile**.

Per cui, quando processo invia un messaggio tramite un socket, UDP **non garantisce la consegna del messaggio**. Inoltre, i messaggi potrebbero giungere a destinazione non in ordine.

La **rapidità del protocollo** è dovuta anche al fatto che i pacchetti vengono inviati direttamente **senza** utilizzare un sistema di **controllo della congestione**. Tuttavia, il reale throughput end-to-end potrà essere inferiore a causa della banda limitata dei collegamenti coinvolti o a causa della congestione. Si ricorda che con il termine throughput ci si riferisce al tasso con quale il processo mittente può inviare i bit al processo ricevente.

- **Vantaggi**
  - **Leggero** poiché non ha bisogno di controllare che la connessione sia instaurata, ovvero il fenomeno di *handshaking* viene eliminato;
  - **Rapido** poiché non esiste nessun sistema di controllo della congestione, per cui i pacchetti vengono mandati uno dopo l'altro. Talvolta il throughput potrebbe essere minore a causa di eventuali colli di bottiglia;
  - **Minimalista** poiché non implementa tecniche particolari come detto in precedenza.
- **Svantaggi**
  - **Nessuna affidabilità** a causa della mancanza del fenomeno di *handshaking*. Quindi, **nessuna garanzia di consegna** del messaggio;
  - **Alta probabilità di congestione** dovuta alla mancanza di controllo di essa. Quindi, il buffer potrebbe riempirsi rapidamente;
  - **Messaggi non in ordine**.

7.3 Esercizio sull'indirizzamento

Dato il seguente IP:

$$140.120.84.20/20$$

Determinare l'indirizzo di rete.

Il **primo passo** è la traduzione dell'IP da notazione decimale puntata a notazione binaria:

$$140_{10} \longrightarrow 10001100$$

$$120_{10} \longrightarrow 01111000$$

$$84_{10} \longrightarrow 01010100$$

$$20_{10} \longrightarrow 00010100$$

Scrivendo l'indirizzo esteso:

$$10001100 \quad 01111000 \quad 01010100 \quad 00010100$$

Il **secondo passo** è l'azzeramento del suffisso (bits in rosso) dato che l'indirizzo di rete ha il prefisso *non* nullo e il suffisso con tutti i bit a zero. Il prefisso viene dato dall' esercizio, ovvero /20:

$$10001100 \quad 01111000 \quad 0101\mathbf{0000} \quad \mathbf{00000000}$$

Il **terzo** e ultimo **passo** è la conversione in decimale e scrivere l'indirizzo di rete in notazione decimale puntata:

$$10001100_2 \longrightarrow 140_{10}$$

$$01111000_2 \longrightarrow 120_{10}$$

$$01010000_2 \longrightarrow 80_{10}$$

$$00000000_2 \longrightarrow 0_{10}$$

E l'indirizzo di rete sarà:

$$140.120.80.0/20$$

7.4 Esercizio subnetting - Avanzato

Date 3 reti LAN, viene assegnato il blocco di rete 165.5.1.0/24. Creare 3 sottoreti per ogni rete LAN in modo da avere lo stesso numero di host.

Il **primo passo** è la classica traduzione in binaria. Si deve tradurre l'indirizzo da notazione decimale puntata a binario:

$$\begin{aligned} 165_{10} &\longrightarrow 10100101_2 \\ 5_{10} &\longrightarrow 00000101_2 \\ 1_{10} &\longrightarrow 00000001_2 \\ 0_{10} &\longrightarrow 00000000_2 \end{aligned}$$

Il **secondo passo** è quello di creare delle sottoreti. Per farlo si deve prendere in considerazione il suffisso. Prima di tutto, si scrive l'indirizzo in notazione binaria:

10100101 00000101 00000001 00000000

La creazione di 3 sottoreti prevede almeno due bit. Difatti, se venisse scelto un bit, si potrebbe creare al massimo 2 sottoreti. Per cui, dall'indirizzo in notazione binaria, si riservano (x in rosso) due bit nel suffisso per creare le sottoreti:

10100101 00000101 00000001 ***xx***000000

Riservando questi due bit, adesso si sono create quattro sottoreti:

LAN1.	10100101	00000101	00000001	**00**000000
LAN2.	10100101	00000101	00000001	**01**000000
LAN3.	10100101	00000101	00000001	**10**000000
LAN4.	10100101	00000101	00000001	**11**000000

La prima sottorete è assegnata alla LAN1, la seconda alla LAN2, la terza alla LAN3 e la quarta è considerata libera. Essa potrà essere utilizzata in futuro.

Il **terzo passo** è calcolare il nuovo prefisso delle sottoreti. Ogni sottorete è formata dal prefisso della rete originaria (/24), più due bit che sono serviti per creare le 3 reti. Quindi, il nuovo prefisso delle 3 reti è diventato /26.

Infine, il **quarto passo** è la scrittura in notazione decimale puntata delle tre reti e il calcolo degli indirizzi disponibili.

La conversione è semplice e si omettono i passaggi e le spiegazioni:

LAN1.	165.5.1.0/26
LAN2.	165.5.1.64/26
LAN3.	165.5.1.128/26
LAN4.	165.5.1.192/26

E la rete è passata da avere ($2^{32} \div 2^{24} = 2^8$) 256 indirizzi disponibili a ($2^{32} \div 2^{26} = 2^6$) 64 indirizzi.

7.4.1 Domanda bonus

Se la LAN1 avesse dovuto avere il doppio degli indirizzi rispetto alla LAN2 e alla LAN3, l'esercizio come si sarebbe svolto?

In questo caso specifico, partendo dal **passo numero due**, ovvero alla creazione delle sottoreti, si sarebbe proceduti in maniera diversa.

Invece di prendere due bit, si prende un solo bit che ci permette di creare due sottoreti: nella prima sottorete si assegna la LAN1 e nella seconda sottorete si assegnano LAN2 e LAN3.

10100101 00000101 00000001 x0000000

Creando le due sottoreti:

10100101 00000101 00000001 **0**0000000
10100101 00000101 00000001 **1**0000000

Il **terzo passo** è quello di creare le due sottoreti all'interno del secondo indirizzo. Due sottoreti necessitano di due bit, per cui la divisione sarà:

LAN2. 10100101 00000101 00000001 1**0**000000
LAN3. 10100101 00000101 00000001 1**1**000000

Infine, il **quarto passo** è quello di calcolare la maschera delle reti, riscrivere gli indirizzi in notazione decimale puntata e calcolare il numero di indirizzi possibili.

La maschera per la LAN1 è aumentata di 1 dalla rete di partenza, per cui /25. Mentre per la LAN2 e LAN3 è aumentata di 2 bit, ovvero /26.

La scrittura in notazione decimale puntata sarà:

LAN1. 165.5.1.0/25
LAN2. 165.1.128/26
LAN3. 165.1.192/26

E il numero di indirizzi della LAN1 saranno ($2^{32} \div 2^{25} = 2^7$) 128, mentre quelli della LAN2 e della LAN3 rimarranno a 64 indirizzi.

8 Protocollo HTTP

8.1 Protocollo HTTP

HTTP (*Hypertext Transfer Protocol*), protocollo a livello di applicazione del Web, costituisce il cuore del Web. Questo **protocollo** è implementato nei programmi in esecuzione su sistemi periferici diversi che comunicano tra loro scambiandosi messaggi HTTP.

Una **pagina web** (*web page*), detta anche documento, è costituita da oggetti. Un **oggetto** è semplicemente un file indirizzabile tramite un URL. La maggioranza delle pagine web consiste di un **file HTML principale** e diversi oggetti referenziati da esso.

Un **browser web** implementa il lato *client* di HTTP, ovvero l'utente stesso. Mentre il **web server** implementa il lato server di HTTP, ospita oggetti web, indirizzabili tramite URL.

HTTP utilizza il protocollo **TCP come protocollo di trasporto**. Il client HTTP per prima cosa inizia una connessione TCP con il server. Una volta stabilita, i processi client e server accedono a TCP attraverso le proprie socket.

Dato che i server HTTP non mantengono informazioni sui client, HTTP è classificato come **protocollo senza memoria di stato** (*stateless protocol*). Un **web server** è **sempre attivo**, ha un indirizzo IP fisso e risponde potenzialmente alle richieste provenienti da milioni di diversi browser.

8.2 HTTP con connessioni (non) persistenti

Sia i client che i server possono avere due diverse configurazioni: le connessioni persistenti e le connessioni non persistenti. Prima di iniziare la spiegazione, si introduce il concetto di **round-trip time** (**RTT**), che rappresenta il **tempo impiegato da un pacchetto per viaggiare dal client al server e poi tornare al client**. Esso include i **ritardi di propagazione**, di **accodamento** nei **router** e nei **commutatori intermedi** nonché di elaborazione del pacchetto.

8.2.1 Connessioni non persistenti

Per spiegare le connessioni non persistenti, si faccia riferimento ad un esempio in cui c'è il trasferimento di una pagina web dal server al client. Si supponga che la pagina consista di un file HTML principale e di 10 immagini, e tutti gli oggetti risiedano sullo stesso server.

1. Il processo client HTTP inizializza una connessione di tipo TCP con il server sulla porta 80, ovvero la porta di default per HTTP.
2. Il client HTTP, tramite la socket, invia al server un messaggio di richiesta del file principale.
3. Il processo server HTTP riceve il messaggio di richiesta attraverso la socket, recupera l'oggetto richiesto dalla propria memoria, lo incapsula in un messaggio di risposta che viene inviato al client attraverso la socket.
4. Il processo server HTTP comunica a TCP di concludere la connessione. Tuttavia, il protocollo TCP garantisce la consegna del messaggio, per cui non termina la connessione finché il client non ha ricevuto l'intero messaggio.
5. Il client HTTP riceve il messaggio di risposta. La connessione TCP termina a questo punto e il messaggio ricevuto indica che l'oggetto incapsulato è un file HTML. Il client estrae il file del messaggio di risposta, esamina il file HTML e trova i riferimenti ai 10 oggetti.
6. Infine, vengono ripetuti tutti i primi quattro passi per ciascuno degli oggetti referenziati.

Si conclude che l'utilizzo di connessione non persistenti, consente di chiudere ogni connessione aperta dopo l'invio dell'oggetto da parte del server e dopo aver ricevuto una conferma dal client.

Nell'esempio vengono create 11 connessioni TCP, una per ogni oggetto richiesto.

Come si può notare, queste connessioni presentano alcuni **limiti**:

1. Per ogni oggetto richiesto occorre stabilire e mantenere una nuova connessione.
2. Per ogni connessione si deve allocare buffer e mantenere variabili TCP sia nel client che nel server.
3. Ciascun oggetto subisce un ritardo di consegna di due RTT: uno per stabilire la connessione TCP e uno per richiedere e ricevere un oggetto.

Questi limiti richiedono un grande onere sul web server che deve aprire nuove connessioni ogni qualvolta ci deve essere uno scambio di dati. Il grande onere riguarda soprattutto il momento in cui il server riceve centinaia di richieste da parte dei client.

8.2.2 Connessioni persistenti

La particolarità delle connessioni persistenti è la possibilità da parte del server di lasciare aperta la connessione TCP dopo l'invio di una risposta. Così facendo, le richieste/risposte future da parte degli stessi client e server potranno essere trasmesse sulla stessa connessione.

Difatti, il server può inviare un'intera pagina web **sfruttando** solamente **una connessione** TCP permanente, oppure inviare più pagine web allo stesso client.

Le **richieste** possono essere effettuate **una dopo l'altra senza attendere** le risposte delle eventuali richieste pendenti (*pipelining*).

In generale, il server HTTP **chiude la connessione** quando essa rimane inattiva per un lasso di tempo arbitrario.

8.3 Formato dei messaggi HTTP

Esistono due tipi di formati di questo protocollo: il messaggio di richiesta HTTP e messaggio di risposta HTTP.

8.3.1 Messaggio di richiesta HTTP

I messaggi di richiesta possono avere un numero indefinito di righe, anche una sola.

Ogni riga alla fine ha un **carattere di ritorno a capo** (*carriage return*) e un **carattere di nuova linea** (*line feed*).

L'ultima riga è seguita da una coppia di caratteri di ritorno a capo e nuova linea aggiuntivi.

In generale, la **prima riga** è la **riga di richiesta** (*request line*) e quelle successive si chiamano **righe di intestazione** (*header lines*).

La **riga di richiesta** è formata da tre campi:

- Campo **metodo** (GET, POST, HEAD, PUT, DELETE)
- Campo **URL**
- Campo **versione HTTP**

Le **righe di intestazione** possono essere di molti tipi. La riga **Host** specifica l'host su cui risiede l'oggetto, la riga **Connection** indica se il server deve effettuare una connessione di tipo persistente o non persistente, la riga **User-agent** specifica il tipo di browser che sta effettuando la richiesta al server e, infine, la riga **Accept-language** indica se l'utente preferisce ricevere una versione dell'oggetto nella lingua specificata; in caso di mancanza, il server provvederà a fornire la versione di default.

8.3.2 Messaggio di risposta HTTP

Nei messaggi di risposta ci sono tre sezioni importanti: una **riga di stato iniziale**, le **righe di intestazione** e il **corpo**.

La **riga di stato iniziale** presenta tre campi:

- **Versione del protocollo**
- **Codice di stato**
- **Messaggio di stato** (approfondimento alla fine di questo paragrafo)

Le **righe di intestazione** possono essere di tipo **Connection** per comunicare al client la gestione della connessione, per esempio "Connection: close" per indicare l'intenzione di chiudere la connessione TCP dopo l'invio del messaggio; di tipo **Date** per indicare l'ora e la data di creazione e invio, da parte del server, della risposta HTTP; di tipo **Server** per indicare che il messaggio è stato generato da un determinato tipo di web server, simile alla riga "User-agent" nel messaggio di richiesta; di tipo **Last-Modified** per indicare l'istante e la data in cui l'oggetto è stato creato o modificato per l'ultima volta; di tipo **Content-Length** per indicare il numero di byte dell'oggetto inviato; di tipo **Content-Type** per indicare il tipo di oggetto specificato nel corpo.

Le **righe di corpo** sono il fulcro del messaggio. Esse contengono l'oggetto richiesto.

Nella **riga di stato iniziale**, i due campi **codice** e **messaggio di stato** indicano operazioni importanti. Qui di seguito si elencano i codici con i relativi messaggi, sono riportati solo i più importanti:

- 200 OK: la richiesta ha avuto successo e in risposta si invia l'informazione.
- 301 Moved Permanently: l'oggetto richiesto è stato trasferito in modo permanente; il nuovo URL viene specificato nell'intestazione, campo Location, del messaggio di risposta.
- 400 Bad Request: codice di errore generico che indica che la richiesta non è stata compresa dal server.
- 404 Not Found: il documento richiesto non esiste sul server.
- 505 HTTP Version Not Supported: il server non dispone della versione di protocollo HTTP richiesta.

8.4 Cookie

I ***cookie*** è un meccanismo utilizzato dal server per sapere se ha interagito precedentemente con un determinato client.

La **prima volta** che il *client* interagisce con un *server*, inviando una richiesta GET, il server crea un identificativo (codice) associato all'utente (e.g. 1234).

Il server **risponde** con una richiesta REPLY HTTP, impostando nell'intestazione il valore ***set-cookie*** al valore impostato al passaggio prima. Alla ricezione della risposta, il *client* memorizza l'indirizzo della pagina usato per fare la prima richiesta GET e il codice fornito dal server (*set-cookie*).

Dopo una serie di interazioni, quando il *client* effettuerà di nuovo una richiesta GET al server specifico, esso dovrà indicare come codice *cookie*, il codice salvato la prima volta. Invece, il server controllerà la storia associata all'utente con il codice fornito e restituisce una risposta specifica all'utente.