Reti di calcolatori

Università di Verona
Imbriani Paolo -VR500437
Professor Damiano Carra

November 21, 2024

Contents

1 Introduzione **4**

2 Architetture di Rete **4**
2.1 Com'è organizzato il backbone? 5
2.2 Modalità di trasferimento dell'informazione tra due utenti 5
2.2.1 Commutazione di circuito 5
2.2.2 Commutazione di pacchetto 6
2.3 Ritardi di trasmissione 6

3 Modello a strati **7**
3.1 Stack ISO/OSI 8
3.2 Stack TCP/IP 9
3.3 Entità coinvolte nella comunicazione 10
3.3.1 Identificazione dei processi 10
3.3.2 Ottenimento dell'IP e della porta 10

4 Indirizzi IP **12**
4.1 Suddivisione dei bit 12
4.1.1 Identificazione del prefisso e del suffisso 12
4.2 Come suddividere gli indirizzi 13
4.2.1 Indrizzi IP riservati 13
4.3 Subnet Mask 13
4.3.1 Notazione Esadecimale 14
4.4 Subnetting 16
4.4.1 Nota storica 16

5 Livello Applicativo **18**
5.1 Esempi di applicazioni e relativi protocolli di trasporto 18
5.2 Socket 19
5.3 Il protocollo HTTP 19
5.4 Metodi HTTP 20
5.5 Messaggio di risposta HTTP 20
5.6 Altri elementi del protocollo HTTP $\rightarrow$ COOKIE 20
5.7 Caching 21
5.8 DNS - Domain Name System 21
5.9 Protocolli di posta elettronica 22

6 Livello di Trasporto **23**
6.1 Header del protocollo TCP 23
6.1.1 Gestione delle connessioni 24
6.1.2 RTO e RTT 24
6.2 Controllo di flusso e il controllo di congestione 25
6.2.1 Slow Start 26
6.2.2 Congestion Avoidance 26
6.2.3 Vanilla 26
6.2.4 Fast Retransmit/Fast Recovery 27
6.2.5 Algoritmo di controllo della congestione del TCP 27
6.3 Protocollo UDP (User Datagram Protocol) 27

6.3.1 Header del protocollo UDP 27

7 Livello di Rete **29**
7.1 Il protocollo IP (Internet Protocol) 29
7.1.1 Header dell'IP 29
7.1.2 Frammentazione IP 29
7.2 Routing / Instradamento 30
7.2.1 Consegna diretta/indiretta 30
7.3 Algoritmi di link-state 31
7.3.1 Algoritmo di Dijkstra 31

1 Introduzione

Il problema principale che andremo ad affrontare nel corso è quello di permettere la comunicazione tra 2 calcolatori. Questo è possibile solo se le due macchine parlano la stessa "lingua". Gli strumenti e requisiti necessari sono i seguenti:

1. Un **protocollo** è un insieme di regole che sovraintende alla comunicazione, in cui si definiscono il formato dei messaggi e le azioni da intraprendere nel gestire i messaggi stessi
2. **L'architettura di rete**, ovvero, come fisicamente questi messaggi verranno trasportati.

Esempio

Immaginiamo che ci siano due utenti, A e B, che si vogliono mandare una lettera. Come si farebbe normalmente, una volta aver scritto la lettera, la si inserisce allì'interno di una busta che contiene informazioni importanti per la giusta riuscita della spedizione, ovverro l'indirizzo della consegna del messaggiio. Una volta fatto ciò, la si inserisce all'interno di una cassetta della posta affidandoci ai servizi di posta che porterà la lettera al destinatario e l'utente B finalmente recupera il messaggio.

La comunicazione tra due utenti è una catena di montaggio che costruisce e decostruisce l'informazione per permettere ai dati di viaggiare da un utente all'altro. Quali sono le tecniche di instradamento necessario per far viaggare il messaggio in rete? Affronteremo questo tipo di comunicazione attraverso l'approccio topdown ovvero partendo dal livello più alto (quello riferito al layer dell'applicazione) fino ad arrivare a quello fisico e più basso.

2 Architetture di Rete

Distinguiamo diversi elementi di base che fanno parte di un architettura di rete:

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

Il router è quell'apparrato che decide quale strada il pacchetto deve fare per raggiungere il destinatario. La LAN (Local Area Network, Rete locale) è caratterizzata dal fatto che al suo interno contiene gli end-host. La backbone tipicamente contiene specificatemente gli apparati in una topologia che decide il gestore della rete stessa. La differenza tra le due è che nella LAN ci sono gli end-host mentre nel backbone si trovano solo collegamenti tra gli apparati che permettono il viaggio dei messaggi sul territorio.

Quindi, una prima definizione di Internet è possibile farla grazie a questi elementi: infatti se dovessimo astrarla ad alto livello, ci sono varie LAN collegate tramite il backbone (dove tipicamente la tecnologia utilizzata è quella cablata, al contrario delle LAN dove è un mix tra cablata e wireless).

2.1 Com'è organizzato il backbone?

Tramite l'ISP (Internet Service Provider) che appunto posseggono una parte di rete e che permettono di farla usufruire agli utenti.

- Livello 1: Estensione Internazionale (Copre diverse nazioni)
- Livello 2: Lavorano a livello nazionale
- Livello 3: Locale

Tipicamente proprio come dei router di bordo, gli ISP di livello 1 sono collegati a loro volta altri ISP di livello 1. Questo permette agli utenti di collegarsi a diversi ISP di livello 1 anche se molto lontani tra loro. Globalmente quindi:

Definizione

Internet è un insieme di reti organizzato gerarchicamente.

Per raggiungere un utete, in genere si segue un percorso gerarchico. Ma ovviamente la scelta del percorso segue criteri basati su:

1. distanza
2. tempo

Non necessariamente il percorso più breve è quello più veloce o viceversa. Il discorso delle migliore tecniche di instradamento verranno spiegeate più avanti nel corso.

2.2 Modalità di trasferimento dell'informazione tra due utenti

Ci sono due opzioni:

- Reti a commutazione di circuito
- Reti a commutazione di pacchetto

2.2.1 Commutazione di circuito

Le risorse (capacità del canale di trasmissione) vengono riservate end-to-end per la communicazione. **Viene riservato un circuito** che viene utilizzato dai due utenti. Quindi per esempio, nel momento che viene effettuata una chiamata, viene riservato un canale dove fino alla fine della chiamata rimarrà occupato. Quindi ci sarà una porzione di tempo dove il circuito verrà instarauto (ritardo) e poi il messaggio è pronto per essere trasmesso.

Pro:

- Risorse dedicate
- Ritardo deterministico

Contro:

- nel caso di utilizzo sporadico, ho uno spreco di risorse

2.2.2 Commutazione di pacchetto

L'informazione (o messaggio) viene suddiviso in **pacchetti**. Ad ogni pacchetto viene aggiunta un'intestazione per permettere la consegna del pacchetto stesso e la ricostruzione del messaggio. Il messaggio viene spezzato in unità più piccole e a ciascuna di queste unità viene aggiunta un intestazione che ha almeno due scopi:

1. Rendere indipendenti questi singoli pacchetti in maniera che possano essere spediti separatamente.
2. Specificare l'ordine delle unità per andare a ricostruire il messaggio.

Il router in base all'intestazione che possiede il pacchetto sa in che direzione spedirli ed ecco perché sono liberi di essere trasmessi in maniera indipendente. Il messaggio può anche arrivare fuori sequenza che tanto verrebbe ricostruito nel giusto ordine.

Pro:

- Uitlizza le risorse solo quando ha pacchetti da trasmettere.
- Multiplazione statistica, ovvero vado a mettere insieme pacchetti che vengono da fonti diverse

Contro:

- Potenziale perdita dei pacchetti
- Ritardi aumentati che dipende dal numero di Router attraversati lungo il viaggio del pacchetto

Il router adotta la tecnica del *Store* & *Forward* ovvero prima memorizza l'intero pacchetto, leggere l'intestazione e in base alla destinazione decidi quale è il successivo router a cui mandare il pacchetto.

2.3 Ritardi di trasmissione

Come abbiamo già visto, tramite lo Store & Forward la commutazione a pacchetto introduce la possibilità che i messaggi arrivino in ritardo. Si può dividere in diverse componenti:

1. Ritardo di elaborazione al nodo. Il router deve semplicemente decidere quale uscita instradare per il messaggio.
2. Ritardo di accodamento (tempo speso nel buffer prima che il pacchetto venga trasmesso) ed è la componente principale (ordine di grandezza decisamente più grande rispetto a quello dell'elaborazione del nodo)
3. Ritardo di trasmissione $\rightarrow \frac{\text{Grandezza messaggio}}{\text{Grandezza del canale}}$
4. Ritardo di propagazione

Tuttavia vogliamo chiederci quale sia effettivamente l'ordine di grandezza del ritardo? Possiamo distinguere tra i tipi di destinazione:

- locale (fondamentalmente nella stessa nazione, dove è $< 10ms$)
- internazionale ($20 - 40ms$)
- intercontinentale ($> 100ms$)

Quali sono gli strumenti che misurando il ritardo end-to-end tra una sorgente e una destinazione?

- Il primo strumento è il ping. Dati 2 utenti e 2 diverse, viene inviato un messaggiod di ping e capire il tempo che intercorre tra l'invio del messaggio di ping e la ricezione della risposta (*echo reply*). Questo tempo viene chiamato *Round-trip-time*. Questo tempo stima il ritardo complessivo per raggiungere l'utente e ricevere la risposta. Non è detto che il tempo sia simmetrico o asimmetrico.
- Traceroute, dati 2 utenti manda dei messaggi anche ai router intermedi. È un ping potenziato dove controlla il ritardo anche tra i router intermedi.

Quante informazioni riesco a trasmettere? $\frac{bit}{s}$

Questa informazione dipende dalle capacità di tutti i canali di trasmissione attraversati La banda end-to-end a disposizione viene definito "*troughput*" ed è determinato dal collo di bottiglia.

3 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.

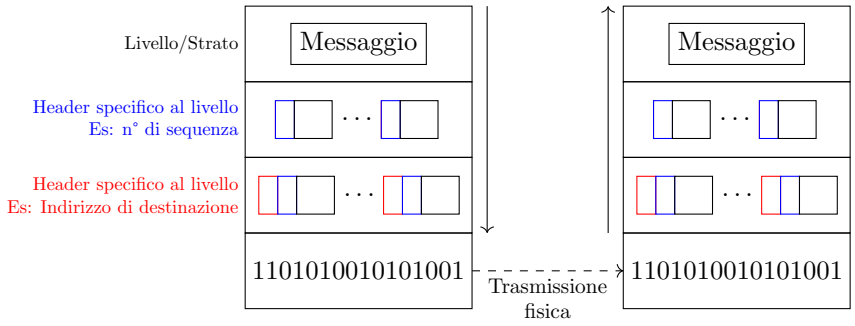


Figure 1: Comunicazione tra due entità

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**.

3.1 Stack ISO/OSI

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

- **End system**:

Figure 2: Stack ISO/OSI per l'end system

- **Intermediate system**:

Figure 3: Stack ISO/OSI per l'intermediate system

3.2 Stack TCP/IP

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

- **End system**:

Figure 4: Stack TCP/IP per l'end system

- **Intermediate system**:
  1. Rete
  2. Collegamento dati
  3. Fisico

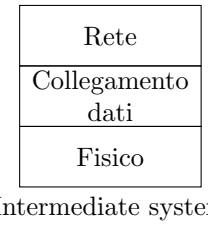


Figure 5: Stack TCP/IP per l'intermediate system

Il nome deriva dai due principali protocolli utilizzati:

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

3.3 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 istanzee 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

3.3.1 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

Figure 6: 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.

3.3.2 Ottenimento dell'IP e della porta

Queste informazioni sono contenute negli header aggiunti ad ogni livello.

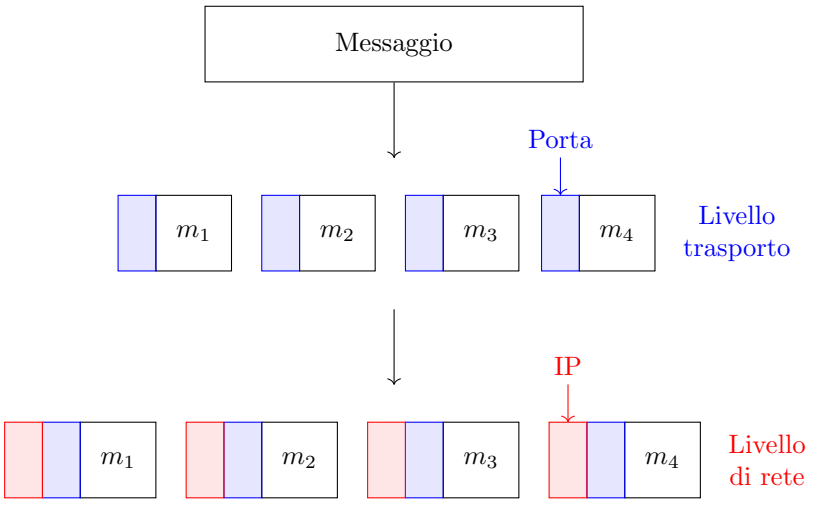


Figure 7: Porta e IP

Un pacchetto si può rappresentare nel seguente modo:

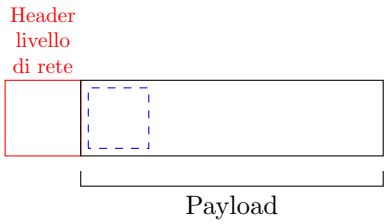


Figure 8: Rappresentazione di un pacchetto

4 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

Un esempio di indirizzo IP è:

10011101 00011011 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

4.1 Suddivisione dei bit

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

Prendiamo ad esempio i numeri telefonici della rete fissa:

$$\underbrace{0039\ 045\ 802}_{\text{Prefisso}}\ \underbrace{7059}_{\text{Suffisso}}$$

- 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

4.1.1 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.

4.2 Come suddividere gli indirizzi

Quanti indirizzi per gli host contiene una rete e ha un prefisso/n?

$$\overbrace{10011101\ \ 00011011\ \ 0001\,0011\ \ 0111\,1011}^{32bit}$$

$$\#\text{ indirizzi} \rightarrow 2^{32-n}$$

Esempio

$$n = 20 \rightarrow 2^{12} = 4096$$
$$n = 24 \rightarrow 2^{8} = 256$$

In base al numero di indirizzi che devo riservare devo capire quale tipo di suffisso mi riesca a sostenere il numero di host.

4.2.1 Indrizzi IP riservati

1. **This host**
$$0000\ \ \ldots\ \ 0000 \rightarrow 0.0.0.0$$
2. **Local/Limited Broadcast**
$$\underbrace{1111\ \ \ldots\ \ 1111}_{\text{32 bit a 1}} \rightarrow 255.255.255.255$$
3. **Indirizzo di rete**
$$\underbrace{1010100010}_{\text{n prefisso}}\underbrace{00000000}_{\text{32-n bit a 0}}$$
indirizzo con il quale viene identificata la rete
4. **Directed Broadcast**
$$\underbrace{1010100010}_{\text{n prefisso}}\underbrace{11111111}_{\text{32-n bit a 1}}$$
indirizzo con il quale viene identificato il broadcast alla rete

4.3 Subnet Mask

Come faccio a conoscere il mio indirizzo IP e la dimensione del prefisso della rete a cui sono collegato?

L'indirizzo IP e la dimensione del prefisso dipendono dalla rete a cui sono collegato. Per conoscere il proprio indirizzo IP si utilizza il comando "ifconfig".

Sul mio Mac in questo momento ho l'indirizzo:
Indirizzo: 157.27.201.95/?
Netmassk: 0xFFFFF000
Broadcast: 157.27.207.255

Per identificare il # host del prefisso i calcolatori:

Esempio

$$/16 \Longrightarrow 11111111111111110000000000000 \ \ldots \ 0000$$

Questa sequenza è chiamata **"maschera"** ed è una sequenza che viene abbinata all'indirizzo IP

$$\begin{gathered}\text{IP: } 101010010101001 \ \ldots \ 10100100 \\ \text{Mask: } \underbrace{111111....11111}_{\text{prefisso}} \ \underbrace{0000...0000}_{\text{suffisso}}\end{gathered}$$

IP & Mask si ottiene all'indirizzo di rete a cui quell'indirizzo IP appartiene

La maschera può essere rappresentata:

- Notazione decimale
- Notazione esadecimale

4.3.1 Notazione Esadecimale

Gruppi di 4 bit.

$$\begin{gathered}0000 \to 0 \\ 0001 \to 1 \\ 0010 \to 2 \\ \vdots \\ 1111 \to F\end{gathered}$$

2^4 configurazioni $\to$ 16 simboli

Il simbolo "$0x$" vuol dire che sta rappresentando con la notazione esadecimale:

Esempio

$$0xFFFFF000 \to \underbrace{1111 \ \ 1111 \ \ 1111 \ \ 1111 \ \ 1111}_{20} \ \ \underbrace{0000 \ \ 0000 \ \ 0000}_{12}$$

Quindi la rete a cui sono connesso è /20.

Altre inforamzioni sulla rete (dato un indirizzo IP si possono ottenere con il calcolo "*whois*"

Perché ifconfig dice che il mio indirizzo è associato ad una rete /20, mentre whois dice che è associata ad una rete /16?

Per "ifconfig"

$$\underbrace{1010010101010100}_{20 \text{ prefisso}} \ \ldots \ 0100 \to 2^{32-20} \to 2^{12} = 4096 \text{ indirizzi}$$

Per "whois"

$$\underbrace{1000100001000100}_{16 \text{ prefisso}} \ \ldots \ 0100 \to 2^{32-16} \to 2^{16} = 65536 \text{ indirizzi}$$

Questo è dovuto al **SUBNETTING**. Dove la rete più grande (UNIVR) ha diviso in sottoreti più piccole suddivise in base all'area geografica: Borgo Roma, Borgo Venezia, Centro... Di questi 65536 indirizzi, 4096 vengono dati a Borgo Roma per esempio. In questo modo tengo separate le reti quanod sono all'interno. Tuttavia da fuori, questa separazione non è visibile.

Esercizio di traduzione da binario a decimale

$$\underbrace{\underbrace{1110}_{128+64+32} \ \underbrace{10111}_{4+2+1}}_{231} \quad 1101 \ \ 1011 \ \ 1000 \ \ 1011 \ \ 0110 \ \ 1111$$

$$\rightsquigarrow 231.$$

Esercizio di traduzione da decimale a binario

221.34.255.82

1. Sottrazioni successive

$$221 \overset{?}{>} 128 \to 1$$

$$221 - 128 = 93 \overset{?}{>} 64 \to 1$$

$$93 - 64 = 29 \overset{?}{>} 32 \to 0$$

$$29 - 16 = 13 \overset{?}{>} 8 \ \to 1$$

$$13 - 8 = 5 \overset{?}{>} 4 \to 1$$

$$5 - 4 = 1 \overset{?}{>} 2 \to 0$$

2. Divisioni successive

$$\begin{array}{ll} 221/2 & 1 \\ 110/2 & 0 \\ 55/2 & 1 \\ 27/2 & 1 \\ 13/2 & 1 \\ 6/2 & 0 \\ 3/2 & 1 \\ 1/2 & 1 \\ 0 & \end{array}$$

E poi ricordare di leggere al contrario $\to$ 11011101.

4.4 Subnetting

Il **subnetting** è una tecnica che ci permette di suddividere una rete in sottoreti. La domanda sorge spontanea: Com'è possibile creare delle sottoreti partendo da un blocco di indirizzi assegnato?

$$\underbrace{157.27.0.0/16}_{\text{Indirizzo di rete}}$$

Che in binario diventa:

$$\overbrace{\underbrace{1001\ \ 1101}_{157}\ \ \underbrace{0001\ \ 1011}_{27}}^{\text{prefisso}}\ \underbrace{0000\ \ 0000\ \ 0000\ \ 0000}_{\text{suffisso}}$$

Da questo blocco, creare 2 sottoreti di pari dimensione.

Esempio

Possiamo prendere il primo bit del suffisso, scorporarlo dal suffisso e associarlo al prefisso.

$$\overbrace{\underbrace{1001\ \ 1101}_{157}\ \ \underbrace{0001\ \ 1011}_{27}\ 0}^{\text{prefisso}}\ \underbrace{000\ \ 0000\ \ 0000\ \ 0000}_{\text{suffisso}}$$

$$\overbrace{\underbrace{1001\ \ 1101}_{157}\ \ \underbrace{0001\ \ 1011}_{27}\ 1}^{\text{prefisso}}\ \underbrace{000\ \ 0000\ \ 0000\ \ 0000}_{\text{suffisso}}$$

In notazione decimale:

$$157.27.0.0/17$$

$$157.27.128.0/17$$

Da un blocco /16 $\rightarrow 2^{32-16} \rightarrow 65536$ indirizzi, ottengo 2 blocchi /17 $\rightarrow 2^{32-17} = 2^{15} \rightarrow 32768$ indirizzi. Grazie a questo bit, il router potrà capire a quale sottorete inviare i pacchetti.

Definizione

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

4.4.1 Nota storica

In passsato gli indirzzamenti erano basati su classi e denominati *classful*. Il numero di bit dedicati al prefisso era predeterminato e venivano usati i bit iniziali per distinguere i diversi casi.

Esempio

- Se l'indirizzo IP iniziava con lo 0 allora il prefisso era di 8 bit.

$$\underbrace{0\,001\;\;1101}_{8\text{ bit}}\;\;0001\;\;10110000\;\;0000\;\;0000\;\;0000 \in \text{Classe A}$$

- altrimenti si guardava il secondo bit. Se era uguale a 0 allora il prefisso era di 16 bit.

$$\underbrace{1\,0\,01\;\;1101\;\;0001\;\;1011}_{16\text{ bit}}0000\;\;0000\;\;0000\;\;0000 \in \text{Classe B}$$

- Altrimenti, si guardava il terzo bit, se era uguale a 0 allora il prefisso era di 24 bit.

$$\underbrace{11\,0\,1\;\;1101\;\;0001\;\;10110000\;\;0000}_{24\text{ bit}}\;\;0000\;\;0000 \in \text{Classe C}$$

Tuttavia per l'alta richiesta di indirizzi, l'indirizzamento *classful* non era sufficiente.

	# Reti	Dimensione blocco
A	$2^7 = 128$	$2^{24} \to 16M$ indirizzi
B	$2^{14} = 16k$	$2^{16} \to 16k$ Indirizzi
C	$2^{21} = 2M$	$2^8 \to 256$ indirizzi

5 Livello Applicativo

Ritornando allo stack ISO/OSI concentriamoci sulla parte applicativa:

End system

Quando un messaggio (di qualsiasi tipo, che sia un email, frame di un video, richiesta/risposta HTTP) viene generato da un applicazione, si deve decidere quale protocollo utilizzare a livello di trasporto.

- **TCP** che sta per *Transmission Control Protocol* (Connection Oriented e affidabile) oppure
- **UDP** che sta per *User Datagram Protocol* (Connectionless e Non affidabile)

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

5.1 Esempi di applicazioni e relativi protocolli di trasporto

Un esempio di protocollo di trasporto dati possono essere:

- Web $\Longrightarrow$ **HTTP** $\Longrightarrow$ quando io invio un messaggio devo assicurarmi che non contenga errori, quindi il protocollo HTTP si appoggia al **TCP**.
- Posta elettronica $\Longrightarrow$ **SMTP** $\Longrightarrow$ anche'esso deve essere affidabile poiché non posso permettermi che l'email inviate possiedano errori.
- Streaming Audio/Video $\Longrightarrow$ Proprietari $\Longrightarrow$ Tollerante alle perdite $\Longrightarrow$ **UDP**
- **DNS** (Domain Name System) è il sistema che trasforma i nomi del dominio in indirizzi IP e l'esecuzione di questo sistema avviene attraverso il protocollo **UDP**.

Quando un applicazione deve mandare un messaggio ad un altra applicazione, *"apre"* un **SOCKET** verso la destinazione.

5.2 Socket

Definizione

Un socket è un'astrazione software che identifica un flusso informativo.
Per *aprire* un socket serve specificare le seguenti cose:

1. Il *protocollo di trasporto* utilizzato (TCP o UDP)
2. Indirizzo IP di destinazione
3. Indirizzo IP sorgente
4. Porta di destinazione (processo)
5. Porta sorgente

Tra i due processi comunicanti possiamo identificare due ruoli distinti:

1. Il processo **CLIENT**
   - È responsabile dell'apertura dell'inizio della comunicazione
   - Ha un indirizzo *dinamico*
2. Il processo **SERVER**
   - Sta su un host sempre raggiungibile e sempre in ascolto
   - Tipicamente ha un indirizzo IP *fisso*

5.3 Il protocollo HTTP

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

1. Il client apre un canale di comunicazione (Socket [5.2]), avvia un messaggio di richiesta e il server invia un messaggio di risposta.
2. Il server mantiene nella pagina web → file di testo + altri contenuti. Il file di testo principale (`index.html`) contiene la struttura della pagina. Un file `html` definisce sia la struttura sia il contenuto di una pagina web.

Il messaggio di richiesta è formato da 2 parti.

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

Esempio

```
GET /index.html/HTTP/1.1
Host:  www.univr.it
User_Agent:  Mozilla/4.0
Accept_Language:  en
```

Corrisponde nel browser ad aver inserito:

`wwww.univr.it/index.html`

5.4 Metodi HTTP

GET è una parola chiave che è definito come "metodo" dove andiamo a richiedere una pagina ad un server.
Tra gli altri metodi che esistono, vi è anche il ***POST***, che invia le informazioni al server. Un ulteriore metodo è ***DELETE*** che serve per cancellare informazione/risorse sul server.

5.5 Messaggio di risposta HTTP

Come è strutturato un messaggio di risposta HTTP?

1. Riga di stato
2. Una o più righe di intestazione
3. Dati

Esempio

- `HTTP/1.1 codice_di_risposta descrizione_della_risposta`
- `Etichetta: valore`
- Esempi di risposta e descrizione:

  200 OK
  404 NOT FOUND
  400 BAD REQUEST

Per aprire una connessione su macchine *Linux-based* si usa il comando `nc`.

```
1 nc -vc indirizzo_url port
```

5.6 Altri elementi del protocollo HTTP → COOKIE

Meccanismo utilizzato dai server web per saper se ha interagito in passato con un determinato client. Non basta solo l'indirizzo IP per capire se un client ha interagito con un server.

I cookie sono delle coppie chiave-valore che vengono inviati dal server web al client (browser) attraverso l'intestazione HTTP della risposta. Quando il client riceve i cookie, li salva localmente e li invia automaticamente al server nelle successive richieste HTTP, a meno che non siano scaduti o non rientrino nei parametri di sicurezza stabiliti.

I cookie servono principalmente a gestire sessioni e a memorizzare informazioni sullo stato dell'utente in un'applicazione web. Alcuni usi tipici includono:

- **Autenticazione**: quando un utente effettua l'accesso a un sito, il server invia un cookie che identifica l'utente in modo da non dover eseguire il login a ogni nuova richiesta.
- **Personalizzazione**: salvare preferenze dell'utente, come la lingua preferita o il tema del sito.
- **Tracciamento**: possono essere usati per tenere traccia delle attività di navigazione dell'utente per fini analitici o pubblicitari.

5.7 Caching

Il *caching* è un meccanismo per diminuire il ritardo per accedere ad una risorsa. Cache di rete è il ritardo nella consegna dei messaggi. Come sotto prodootto abbiamo che diminuisce il carico della rete.

Una buona cache lavora con $70 - 80\%$ di cache hit.

Cne cosa succede se il contenuto nel server di origine cambia?
Il protocollo HTTP prevede il meotdo GET-condizionale in cui una riga di intestazione della richiesta è `if_modified_since:<data>`.

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

5.8 DNS - Domain Name System

Lo scopo principlae del DNS è la traduzione dei nomi logici come ad esempio `www.univr.it` nel corrispettivo indirizzo IP (ad esempio 157.25...).

Come conosco l'indirizzo IP del server DNS?

→ Può essere inserito manualmente dall'utente nelle impostazioni

→ Viene fornito dalla rete stessa quando viene assegnato l'indirizzo IP (DHCP)

Come fa il DNS a conoscere per ciascun dominio, quindi per ciascuno nome logico presente su internet, conoscere l'associazione tra il dominio e l'indirizzo IP (potenzialmente miliardi)?

Non esiste un singolo server DNS ma c'è un sistema distribuito e gerarchico di server che si parlano tra di loro.

- **Server DNS radice** (7-10): gestiscono le informazioni relativi ai domini di primo livello (.it, .fr, .com, ...)
- **Server DNS locali** gestiscono le informazioni relative ai domini di organizzazione specifica (univr.it, repubblica.it, ...)

In un indirizzo **URL:**

`www` . `univr` . `it`
www = Server
univr = Locali
it = TLD

Il client interagisce solamente con il server DNS locale e con una serie di passaggi gli viene recuperato l'indirizzo IP da lui richiesto tramite l'indirizzo logico.
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.

Qual è il rapporto tra DNS e Network Cache?

5.9 Protocolli di posta elettronica

- Invio
  - → SMTP (Simple Mail Transfer Protocol)
- Ricezione
  - → POP (Non più utilizzato) (Post Office Protocol)
  - → IMAP (Internet Mesage 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 dell'indirizzo di posta elettronica) deve avere associato uno specifico server SMTP che gestisce le caselle degli utenti appartenenti a quel dominio.
Quando il server univr.it riceve un messaggio, controlla il dominio, se il dominio è `univr.it`, allora mette il messaggio sulla corrispondente casella di posta. Se il dominio è diverso, il server SMTP controlla il server SMTP di destinazione per la consegna del messaggio attraverso il protocollo SMTP.

6 Livello di Trasporto

6.1 Header del protocollo TCP

L'header del protocollo TCP (Transmission Control Protocol) è una parte fondamentale del protocollo, che viene utilizzata per gestire la trasmissione affidabile dei dati tra due dispositivi su una rete. Ogni segmento TCP, ovvero ogni unità di dati inviata, contiene un header che include informazioni necessarie per la corretta gestione e interpretazione della connessione e dei dati.
L'header TCP standard è composto da 20 byte (minimo) e può essere esteso con campi opzionali. Vediamo i campi principali:

→ **Porta Sorgente/Destinazione**: Indentificatori dei processi sorgente e destinazione coinvolti nella comunicazione. Esistono due tipi di porte:

- Statiche, ovvero sono i numeri di porte che vanno da 0 - 1023, identificativi associati a protocolli definiti dallo standard (HTTP, SMTP, ...) utilizzati **lato server**.
- Dinamiche, ≥ 1024 e sono identificativi assegnati dal sistema operativo **lato client** quando viene aperto un socket.

→ **Numero di sequenza**, numero sequenziale che identifica univocamente ogni byte di dati nel flusso TCP. È cruciale per il riordinamento dei pacchetti, specialmente se alcuni di essi vengono ricevuti fuori ordine.

- A livello applicativo vengono generati messaggi di dimensione arbitraria
- A livelllo "Collegamenti Dati" (L2) alla scheda di rete è associato un parametro chiamato:
  - MTU Maximum Transmission Unit
  - MSS Maximum Sequent Size

→ **Numero di riscontro o di acknowledgment**[1]: Questo campo è utilizzato per confermare la ricezione dei pacchetti. Contiene il numero di sequenza del byte successivo che il ricevente si aspetta di ricevere. L'uso del numero di riscontro rende possibile il controllo di flusso e la conferma di ricezione dei dati.

→ **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 a buon fine: ci sono **sicuramente** degli errori

[1] ACK: È un segmento TCP senza dati (Solo header)

6.1.1 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*:

- **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: 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.

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). L'RTO dovrà tenere in conto l'RTT (Round Trip Time), cioè il tempo che impiega un segmento a viaggiare da un host all'altro e tornare indietro.

6.1.2 RTO e RTT

L'RTO dipende dal valore del RTT (Round Trip Time → tempo che impiega un segmento a viaggiare da un host all'altro e tornare indietro.). All'apertura della connessione, misura il RTT durante il Three-way handshake. Per l'invio del primo messaggio (Syn) l'RTO è impostato a 500 ms.

Come vengono aggiornati i valori di RTT e RTO? Ogni Host, per ongi segmento inviato, misuro l'RTT instantaneo e usa tale valore per aggiornare una variabile chiamata: *SRTT* $\rightarrow$ *Smoothed Round Trip Time.*

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

$$0 < \alpha < 1$$

L'SRTT è uno **stima** del valore medio del RTT. Questa tecnica si chiama *Exponential Weighted Moving Average* (EWMA).
Dato l'SRTT:

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

dove tipicamente $\beta = 2$. Quindi in poche parole, il segmento viene ritrasmesso dopo un tempo pari al doppio del tempo medio di ritorno. In caso di perdite, l'RTO per il segmento ritrasmesso viene temporaneamente raddoppiato. Se invio un segmento ad ogni RTT, la velocità di trasmissione è $\frac{1seg}{k\pi} \rightarrow \frac{1500byte}{100ms} =$ 1200 byte/sec.

6.2 Controllo di flusso e il controllo di congestione

Per aumentare la velocità di trasmissione invece di trasmettere un singolo segmento ne trasmetto di più contemporaneamente. Per esempio, se ne inviassi due alla volta, raddoppierai la velocità di trasmissione. Tuttavia questo genera diversi problemi. Come gestisco i riscontri e cosa succede se uno dei segmenti viene perso? Ci sono due tecniche che ci aiutano in questo caso:

1. Il controllo di flusso $\rightarrow$ Azione preventiva per limitare la congestione. Inviare più segmenti contemporaneamente con un meccanismo o finestra scorrevole

2. Il controllo di congestione $\rightarrow$ Reazione in caso di congestione

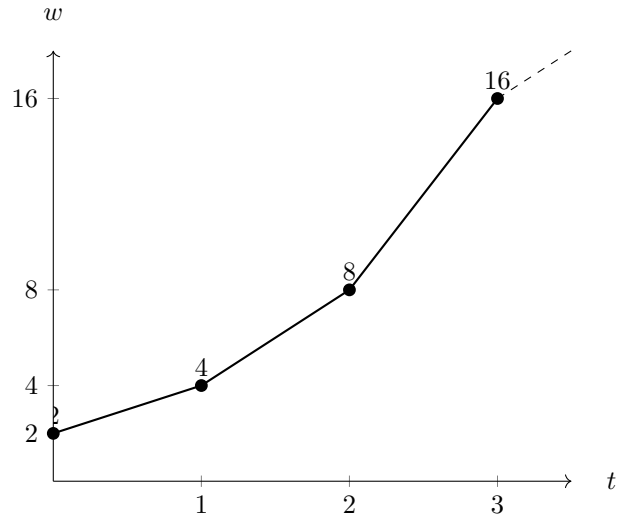
$$\text{Congestione} \iff \text{Perdita di un segmento (Riscontro non ricevuto)}$$

Grazie all'ACK che mi dice quale è il prossimo segmento che si aspetta, posso essere sicuro di non perdere i pacchetti ed essere ancora più affidabile. Quant'è la dimensione della finestra di trasmissione? Utilizzzare i riscontri (o la loro assenza) per variare in modo dinamico la window size.

- Se ricevo regolarmente i riscontri, aumento la finestra di trasmissione
  - Slow Start
  - Congestion Avoidance
- In caso di perdite (RTO scende), diminuisco la finestra di trasmissione
  - Vanilla
  - Fast Retransmit/Fast Recovery

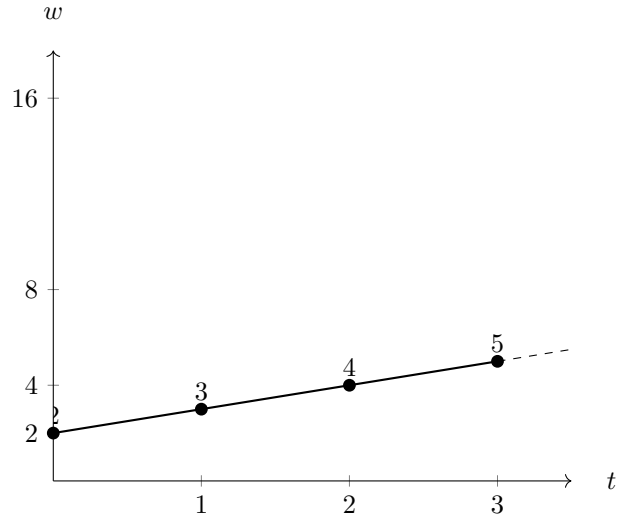
6.2.1 Slow Start

Per ogni riscontro ricevuto si fa aumenta la finestra di un segmento. (Oltre a farla scorrere verso destra di un segmento).



6.2.2 Congestion Avoidance

Per ogni riscontro ricevuto, aumento la finestra di $1/w$ dove w è la dimensione precedente della finestra.



6.2.3 Vanilla

Se non arriva il riscontro, si attende lo scadere del RTO, si pone la finestra = 1 e si ritrasmettono i segmenti persi.

6.2.4 Fast Retransmit/Fast Recovery

In caso di ricezione di ACK duplicati, dopo averne ricevuti 3, si pone la finestra a $w' = \frac{w}{2} \leftarrow$ (finestra al momento della perdita) e si ritrasmette il segmento indicato negli ACK duplicati. (E se ci sono le "opzioni" presenti che indicano gli altri segmenti segmenti mancanti, si ritrasmettono anche quelli).

6.2.5 Algoritmo di controllo della congestione del TCP

Variabili:

- **CWND** → Congestion Window → dimensione di trasmissione attuale
- **RTO** → Calcolo dinamicamente in base all'RTT instantaneo e SRTT.
- **RCVWND** → Receive Window → Finestra massima di ricezione
- **SSTHRESH** → Slow Start Threshold → Soglia usata per distinguere Slow Start e Congestion Avoidance

```
// Inizializzazione
CWND = 1
RCVWND = Viene caricato dalla destinazione in ogni riscontro
STTHRESH = RCVWND (in alcune implementazioni (RCVWND/2))
RTO = in base a valore del RTT misurato nel 3-way-handshake
// Algoritmo
1 . Invia un numero di seguenti pari alla CWND
2. Quando arrivano i riscontri
  if CWND < SSTHRESH // uso slow_start
    CWND = min(CWND + #ack, RCVWND, STTHRESH)
  else               // uso Congesture Avoidance
    CWND = CWND + (#ack/CWND)
Torno al punto 1.
3. Se non arrivano i riscontri -> Scade il RTO
pongo la STTHRESH = CWND al momento della perdita/2
pongo la CWND = 1 // Variante Vanilla TCP
per i segmenti ritrasmessi pongo RTO = 2RTO
Torno al punto 1.
```

Esempio dell'evoluzione della trasmissione TCP (semplificato poiché siamo nel caso RTT sia mediamente stabile)

6.3 Protocollo UDP (User Datagram Protocol)

Il protocollo UDP:

- Connectionless: 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.
- Non affidabile: in caso di perdita, i segmenti *non* vengono ritrasmessi

6.3.1 Header del protocollo UDP

Le particolarità del suo header è che contiene soltanto:

- Porta Sorgente

- Porta destinazione
- Lunghezza (del messaggio)
- Checksum, serve per controllare gli errori nel segmento

Possiamo notare come non è presente il sequence number, infatti non consegna i segmenti in ordine che vengono inviati dal livello applicativo. Proprio per questo, il protocollo è molto leggero durante la trasmissione di dati in tempo reale, infatti non carica molto il processore essendo che non deve gestire efficaciamente i dati.

7 Livello di Rete

7.1 Il protocollo IP (Internet Protocol)

7.1.1 Header dell'IP

L'header del protocollo IP contiene le seguenti specifiche:

- Versione del protocollo, indica la versione del protocollo IP utilizzato. Attualmente la versione che usiamo è IPv4 ma nel futuro si utilizzerà IPv6
- Lunghezza dell'header → 20 byte se non ci sono opzioni può essere più lungo
- Service Type → Codice che identifica una classe del servizio
- Lunghezza totale del pacchetto → Header + Payload
- Identification, flag e Fragment Offset → Usati per la frammentazione dei pacchetti
- Time to Live → Numero di hop massimi che il pacchetto può fare (Max 64 Router) ogni volta che attraversa un Router il valore viene decrementato di uno se il valore diventa 0, manda un messaggio di errore alla sorgente e scarta il pacchetto. In caso di problemi di routing, la rete deve ricalcolare il percorso e nel frattempo si possono creare dei routing-loop. (momentanei)
- Type → Codice che identifica il protocollo di trasporto (TCP, UDP, ICMP, ...)
- Header Checksum → Controllo degli errori dell'header

7.1.2 Frammentazione IP

Data una tecnologia di trasmissione (Scheda di rete, a livello datalink) c'è una dimensione massima di trasmissione chiamata MTU.
Che cosa succede se un pacchetto durante il percorso incontra una MTU minore di quella del pacchetto? *Il pacchetto viene frammentato in pacchetti più piccoli.*

→ Il *router* con link di uscita con MTU < dimensione del pacchetto, frammenta il pacchetto in più pacchetti con MTU compatibile con il link di uscita.

I campi dell'header coninvolti nella frammentazione sono:

- **Identification** → Numero progressivo dato dalla sorgente (livello di rete) ad ogni pacchetto (nessuna relazione con il # sequenza del TCP)
- **Flag** → 3 bit:
  - → Bit "M", se il pacchetto non è stato frammentato 0 oppure se è l'ultimo frammento. altrimenti 1 se il pacchetto è un frammento tranne l'ultimo
- **Fragment Offset** → Indica la posizione del frammento all'interno del pacchetto originale (diviso per 8.)

Esempio

MTU = 1500 byte, pacchetto = 4000 byte +20 byte header IP.
Nel percorso, il pacchetto passa attraverso un router con link di uscita:

- 1400 byte per il payload
- 20 byte per l'header IP

Quindi posso dividere il pacchetto fino ad un max di 3 frammenti, 2 da 1400 byte e 1 da 1200 byte.

Identification: Avranno lo stesso ID del pacchetto originario.
Flag: Per quanto riguarda la flag del primo frammento, il bit M sarà 1, per gli altri 0.
Fragment Offset: Frammento 1 $= 0$, Frammento 2: $\frac{1400}{8} = 175$, Frammento 3: $\frac{2800}{8} = 350$

Chi fa il riassemblaggio?

- Il riassemblaggio viene fatto dal destinatario (host finale) e non dai router.
- Quando la destinazione riceve un frammento fa partire un timer (250-500 ms) per attendere gli altri frammenti. Se li riceve entro la scadenza del timer procede con il riassemblaggio, altrimenti scarta i frammenti ricevuti.

7.2 Routing / Instradamento

Scopo del livello di rete: Data una destinazione $\overbrace{\text{fare il possibile}}^{\text{best effort}}$ per consegnare il pacchetto al destinatario. Quindi per fare in modo che il pacchetto arrivi a destinazione bisogna effettuare il routing (instradamento) del pacchetto.

7.2.1 Consegna diretta/indiretta

Un host conosce, oltre al proprio indirizzo IP, la maschera della rete a cui appartiene e l'indirizzo IP (dell'interfaccia) del router di bordo.

Data una destinazione (IP_d) l'host confronta $\overbrace{\text{IP e Maschera}}^{\text{prefisso immutato e suffisso a 0}}$ e IP_d e maschera. Se i due valori **sono uguali** la destinazione appartiene alla stessa rete e quindi avviene la **consegna diretta**. Altrimenti vuol dire che la destinazione appartiene ad un'altra rete e quindi faccio la **consegna indiretta** ovvero *demando la consegna al router*.

Definizione

Il *routing* è un processo di scoperta del cammino "migliore" da una sorgente a tutte le possibili destinazioni.

"Migliore" perché tutto ciò dipende dai criteri adottati da chi gestisce la rete (ISP).

Esempio

Tipi di criteri:

- Distanza (cammino più corto), in termini di numero di hop (# router attraversati) o in termini di chilometri.
- Velocità di trasmissione
- Livello di congestione (ma che non viene utilizzato)

Data la rappresentazione dei grafi della rete (dove gli archi hanno associato un costo), *il routing* non è altro che il **calcolo del cammino con il costo minimo**. Chi ha già conoscenza nel campod degli algoritmi questo tipo di approccio è conososciuto come *Shortest Path Problem* e vengono applicati algoritmi di Pathfinding.

Dato una topologia (router e collegamenti) derivo il grafo e assegno i pesi secondo un criterio:

- # di hop $\rightarrow$ Tutti gli archi hanno peso 1
- velocità di trasmissione (capacità nominale del link) i pesi sono proporzionale all'inverso della banda

Algoritmi per il calcolo del cammino minimo:

- Stato dei collegamenti: Link-State
- Vettori di distanza: Distance-Vector

7.3 Algoritmi di link-state

- Ogni nodo ha una tabella di routing
- Ogni nodo conosce la topologia della rete
- Ogni nodo calcola il cammino più breve verso tutti gli altri nodi

Ad ogni algoritmo corrisponde un protocollo di routing. Protocllo basato su link-state: OSPF (Open Shortest Path First).

7.3.1 Algoritmo di Dijkstra

Definizione

L'algoritmo di Dijkstra è un algoritmo che permette di trovare il cammino più breve tra due nodi di un grafo pesato, con pesi non negativi.

I nodi si riscambiano i messaggi (Questi messaggi trasportano informazioni sulla topologia della rete.)

1. periodici, per controllare se sono ancora raggiungibili
2. in caso di guasti/cambio di topologia