

# Reti di Calcolatori

Damiano Trovato

Secondo semestre, anno 2024/25

# Indice

<b>1 Introduzione alle reti di calcolatori</b>	<b>5</b>
1.1 L'importanza delle interconnessioni . . . . .	5
1.1.1 Cos'è una rete informatica . . . . .	5
1.2 Cos'è un protocollo . . . . .	6
1.2.1 Stack di protocolli . . . . .	6
1.2.2 Header . . . . .	6
1.2.3 Il mezzo fisico . . . . .	6
1.2.4 Obiettivo . . . . .	6
1.3 Tipi di comunicazione . . . . .	7
1.3.1 Connection oriented e le sue criticità . . . . .	7
1.3.2 Temporizzazione come soluzione . . . . .	7
1.3.3 Topologie di rete . . . . .	7
1.3.4 PAN, LAN, MAN, WAN . . . . .	7
<b>2 Application Layer</b>	<b>8</b>
2.1 Introduzione al livello di applicazione . . . . .	8
2.1.1 Paradigma Client-Server . . . . .	8
2.1.2 Paradigma peer-to-peer . . . . .	8
2.1.3 Socket . . . . .	8
2.1.4 Indirizzi IP e porte. . . . .	9
2.1.5 Comunicazione inter-process e non . . . . .	9
2.2 Definizione di un protocollo applicativo . . . . .	10
2.3 Trasferimento dati - TCP e UDP . . . . .	11
2.3.1 TCP - Transmission Control Protocol . . . . .	11
2.3.2 UDP - User Datagram Protocol . . . . .	12
2.4 Telnet . . . . .	13
2.4.1 Funzionamento di Telnet . . . . .	13
2.4.2 Remote Desktop Protocol . . . . .	13
2.5 FTP - File Transfer Protocol . . . . .	14
2.5.1 Client-Server . . . . .	14
2.5.2 Vulnerabilità del protocollo FTP . . . . .	14
2.6 HTTP - HyperText Transfer Protocol . . . . .	15
2.6.1 In cosa consiste la connessione HTTP . . . . .	15
2.6.2 Versioni di HTTP . . . . .	15
2.6.3 Struttura di una richiesta HTTP . . . . .	16
2.6.4 Struttura di una risposta HTTP . . . . .	17
2.6.5 Parentesi Stateless, Stateful e Cookies . . . . .	18
2.6.6 I cookies . . . . .	18
2.6.7 DHTML . . . . .	19
2.7 Server proxy . . . . .	19
2.8 SMTP . . . . .	20
2.8.1 Le tre fasi di SMTP . . . . .	20
2.8.2 Protocolli del mail server - POP3 e IMAP . . . . .	21
2.9 DNS - Domain Name System . . . . .	22
2.9.1 DNS - Transport Layer . . . . .	22
2.9.2 Struttura del DNS . . . . .	22
2.9.3 Primary e secondary DNS . . . . .	23
2.9.4 Name resolution - Modo iterativo . . . . .	23
2.9.5 Name resolution - Modo ricorsivo . . . . .	23
2.9.6 Caching e Updating DNS . . . . .	23

2.9.7	Forma di un record di un DNS . . . . .	24
2.9.8	Forma di un messaggio DNS . . . . .	24
2.9.9	nslookup . . . . .	25
2.9.10	Problemi di sicurezza e criticità dei DNS . . . . .	25
2.10	SNMP . . . . .	26
2.10.1	Attori del SNMP . . . . .	26
2.10.2	Messaggi SNMP . . . . .	26
2.10.3	Versioni di SNMP . . . . .	26
<b>3</b>	<b>Transport Layer</b>	<b>27</b>
3.1	Introduzione al Transport Layer . . . . .	27
3.1.1	Connessione logico, connessione reale . . . . .	27
3.2	Multiplexing e Demultiplexing . . . . .	28
3.2.1	Multiplexing . . . . .	28
3.2.2	Demultiplexing . . . . .	28
3.3	UDP - User Datagram Protocol . . . . .	29
3.3.1	Caratteristiche di UDP . . . . .	29
3.3.2	Checksum (per UDP, ma anche TCP) . . . . .	29
3.4	Reliable Data Transfer - Trasferimento affidabile . . . . .	30
3.4.1	RDT - Introduzione . . . . .	30
3.4.2	RDT 1.0 - Canale perfetto, zero errori, zero perdite . . . . .	30
3.4.3	RDT 2.0 - Canale imperfetto, errori sul forward . . . . .	31
3.4.4	RDT 2.1 - Errori sul forward e sul backward . . . . .	32
3.4.5	RDT 2.2 - Eliminiamo il NAK! . . . . .	33
3.4.6	RDT 3.0 - Errori e perdite . . . . .	34
3.5	Calcolo del Throughput . . . . .	35
3.5.1	Throughput teorico . . . . .	35
3.5.2	Throughput effettivo . . . . .	36
3.5.3	Considerazioni . . . . .	36
3.5.4	Ping . . . . .	36
3.6	Modalità Pipeline . . . . .	37
3.6.1	Contro della modalità pipeline . . . . .	37
3.6.2	Algoritmi per la pipeline - Go-Back-n . . . . .	38
3.6.3	Algoritmi per la pipeline - Selective Repeat . . . . .	40
3.7	TCP . . . . .	42
3.7.1	TCP-State Model . . . . .	43
3.7.2	Fasi di apertura della connessione (3-way-handshake) . . . . .	43
3.7.3	Fasi della chiusura della connessione (4-way-handshake) . . . . .	44
3.7.4	Struttura dei segmenti del TCP . . . . .	45
3.8	Migliorare il Througput - MMS e MTU . . . . .	47
3.8.1	MMS, MTU . . . . .	47
3.9	Migliorare il Througput - Retransmission Time-Out . . . . .	47
3.9.1	Stimare il Round Trip Time . . . . .	47
3.9.2	Exponential Weighted Moving Average - EWMA . . . . .	48
3.9.3	Deviazione del RTT . . . . .	48
3.9.4	Risultato finale - Retransmission Time-out . . . . .	48
3.10	Timer . . . . .	49
3.10.1	Fast-Retransmit . . . . .	49
3.11	TCP - Controllo di flusso . . . . .	50
3.11.1	Finestra di ricezione . . . . .	50
3.11.2	Algoritmo di Nagle . . . . .	50
3.12	Connessione TCP . . . . .	51
3.12.1	2-way handshake (un'intuizione inconcludente) . . . . .	51
3.12.2	3-way handshake . . . . .	51
3.12.3	4-way handshake (2*2-way handshake) per chiusura connessioni . . . . .	52
3.13	Controllo della congestione . . . . .	53
3.13.1	Scenario 1 - due mittenti e receiver, router con buffer illimitato . . . . .	53
3.13.2	Scenario 2 - due mittenti e receiver, un router con buffer limitato . . . . .	54
3.13.3	Terzo scenario - quattro mittenti, più router limitati e percorsi . . . . .	55
3.14	Meccanismi per gestire la congestione . . . . .	56
3.14.1	Due approcci . . . . .	56

3.14.2	AIMD - Additive Increase Multiplicative Decrease . . . . .	56
3.14.3	Fast Recovery - TCP Tahoe e Reno . . . . .	56
3.14.4	ECN - Explicit Congestion Notification . . . . .	57
3.15	TCP Fairness . . . . .	57
<b>4</b>	<b>Panoramica del Network layer</b>	<b>58</b>
4.1	Introduzione al Network Layer . . . . .	58
4.1.1	Instrandare vs Trasferire . . . . .	58
4.1.2	Data plane vs Control plane . . . . .	58
4.2	Routing . . . . .	59
4.2.1	Self-Organizing-Routing . . . . .	59
4.2.2	SDN - Software Defined Networking . . . . .	59
4.2.3	Virtual Circuit Service, circuito virtuale . . . . .	59
4.2.4	Packet Switching, commutazione di pacchetto . . . . .	59
4.2.5	Router . . . . .	59
4.2.6	Indirizzamento logico . . . . .	59
4.3	Frammentazione e riassemblaggio . . . . .	60
4.4	Modelli di servizio di rete . . . . .	60
4.4.1	Internet Service Model - Best Effort . . . . .	60
<b>5</b>	<b>Network Layer - Piano dei dati</b>	<b>61</b>
5.1	I router . . . . .	61
5.1.1	Elementi principali di un router . . . . .	61
5.1.2	Porte d'ingresso del router . . . . .	62
5.1.3	Tabelle d'inoltro e destination-based forwarding . . . . .	62
5.1.4	Struttura di commutazione . . . . .	63
5.1.5	Accodamento . . . . .	64
5.1.6	Packet discarding policy . . . . .	65
5.1.7	Scheduling dei pacchetti . . . . .	65
5.2	I protocolli del livello di rete . . . . .	65
5.3	IPv4 - Internet Protocol v4 . . . . .	66
5.3.1	Indirizzo IP . . . . .	66
5.3.2	Assegnamento degli indirizzi IP . . . . .	66
5.3.3	Formattazione dei Datagram IPv4 . . . . .	67
5.3.4	Un problema: la frammentazione . . . . .	68
5.3.5	Indirizzamento IPv4 . . . . .	69
5.3.6	CIDR - Classless InterDomain Routing . . . . .	70
5.3.7	Come vengono usate le maschere di rete . . . . .	71
5.4	Elementi del livello di collegamento e ARP . . . . .	72
5.4.1	Cos'è un indirizzo MAC? . . . . .	72
5.4.2	Il protocollo ARP . . . . .	72
5.5	Ottenere gli indirizzi IP . . . . .	75
5.6	DHCP - Dynamic Host Configuration Protocol . . . . .	76
5.6.1	DHCP - Un protocollo plug-and-play . . . . .	76
5.6.2	Come funziona DHCP . . . . .	76
5.7	NAT - Network Address Translation . . . . .	77
5.7.1	Problema da risolvere . . . . .	77
5.7.2	Soluzione, NAT! . . . . .	77
5.8	IPv6 . . . . .	78
5.8.1	Parentesi storica . . . . .	78
5.8.2	Funzionalità rimosse . . . . .	78
5.8.3	Struttura dei datagrammi IPv6 . . . . .	79
5.8.4	Funzionalità di IPv6 . . . . .	80
5.8.5	Sugli indirizzi IPv6 . . . . .	80
5.8.6	EUI-64 - Extended Unique Identifier . . . . .	81
5.8.7	Neighbour Discovery Protocol (il nuovo arp!) . . . . .	81
5.8.8	IPv6 over Ethernet . . . . .	81
5.8.9	Da IPv4 a IPv6 . . . . .	81
5.9	ICMP - Internet Control Message Protocol . . . . .	82
5.10	Firewall . . . . .	82
5.10.1	DMZ - Demilitarized Zone . . . . .	82

<b>6 Network Layer - Piano di controllo</b>	<b>83</b>
6.1 Algoritmi di Instradamento (routing) . . . . .	83
6.1.1 Centralizzati, non centralizzati . . . . .	83
6.1.2 Algoritmi statici e dinamici . . . . .	83
6.1.3 Algoritmi sensibili o insensibili al carico . . . . .	84
6.2 Instradamento link-state . . . . .	85
6.2.1 Flooding e Link state broadcast . . . . .	85
6.2.2 Alcune convenzioni . . . . .	86
6.2.3 L'algoritmo di Dijkstra . . . . .	86
6.2.4 Complessità dell'algoritmo . . . . .	86
6.3 Instradamento Distance Vector . . . . .	87
6.3.1 Algoritmo . . . . .	87
6.3.2 Poisoned reverse . . . . .	87
6.4 Routing in Internet . . . . .	88
6.4.1 RIP - Routing Information Protocol (intra-AS) . . . . .	88
6.4.2 OSPF - Open Shortest Path First (intra-AS) . . . . .	89
6.4.3 BGP - Border Gateway Protocol . . . . .	90
<b>7 Data Link Layer</b>	<b>91</b>
7.1 Introduzione al livello Data Link . . . . .	91
7.1.1 Servizi del Data Link Layer . . . . .	91
7.1.2 Implementazione del DLL - NIC . . . . .	92
7.1.3 Comunicazione tra due Network Interface Controller . . . . .	92
7.2 Data Framing . . . . .	93
7.2.1 Three-Level Encoding . . . . .	93
7.2.2 Codifica a blocchi . . . . .	94
7.2.3 Quantità di informazioni . . . . .	95
7.3 Gestione degli errori . . . . .	96
7.3.1 Ridondanza: elemento di base per l'error detection . . . . .	96
7.3.2 Error Detection . . . . .	96
7.3.3 Parity check . . . . .	97
7.3.4 CRC - Cyclic Redundancy Check . . . . .	98
7.3.5 Come effettua il CRC . . . . .	98
7.3.6 Esiti del CRC . . . . .	98
7.4 Error correction - Perché è difficile? . . . . .	99
7.4.1 Distanza di Hamming . . . . .	99
7.4.2 Correzione degli errori - distanza di Hamming . . . . .	99
7.4.3 Dimostrazione distanza minima . . . . .	100
7.4.4 Come si effettua la verifica della distanza di Hamming? . . . . .	101
7.5 Gestire gli accessi multipli . . . . .	102
7.5.1 Protocolli a suddivisione del canale . . . . .	102
7.5.2 Protocolli ad accesso casuale . . . . .	103
7.5.3 Protocolli senza collisioni . . . . .	104
7.6 Ethernet . . . . .	106
7.6.1 Ethernet CSMA/CD . . . . .	106
7.6.2 Tecnologie standardizzate Ethernet . . . . .	106
7.6.3 Cavi . . . . .	107
7.6.4 Connettore RJ-45 . . . . .	108
7.7 Frame Ethernet . . . . .	109
7.7.1 Codifica di Manchester . . . . .	110
7.8 Gigabit Ethernet . . . . .	111
7.8.1 Filtro DuoBinary . . . . .	111
7.8.2 Decodifica 4D . . . . .	112
7.9 Strutture Ethernet . . . . .	113
7.9.1 Differenze tra hub e bridge . . . . .	113
7.9.2 Switch . . . . .	113
7.10 VLAN . . . . .	114
7.10.1 Scopo delle VLAN . . . . .	114
7.10.2 VLAN basata su porte . . . . .	114
7.10.3 LAN vs Dispositivi Legacy . . . . .	115

# Capitolo 1

## Introduzione alle reti di calcolatori

### 1.1 L'importanza delle interconnessioni

Tutto il mondo è connesso tramite le reti: Internet è la più grande rete di interconnessione esistente, e il suo utilizzo ha stravolto il mondo per com'era conosciuto. Le reti velocizzano il trasferimento di dati: ciò avviene tramite un canale fisico, che permette il trasferimento di informazioni secondo dei protocolli stabiliti. Partecipano a queste interconnessioni non solo calcolatori, ma dispositivi (*devices*) di qualsiasi tipo: server, data-center, dispositivi IoT e altro ancora. Data l'importanza delle reti al giorno d'oggi, diventa altrettanto importante **creare un'infrastruttura solida**, e un **insieme di protocolli** che assicurino la stabilità del sistema.

#### 1.1.1 Cos'è una rete informatica

È una rete che permette di collegare più dispositivi (*devices*), e metterli in comunicazione. Individuiamo due tipi di interconnessioni:

- **Connessione fisica.**  
Si riferisce al mezzo fisico (cavo, sistemi wireless).
- **Struttura logica.**  
Si riferisce ai protocolli che garantiscono il funzionamento della rete.

## 1.2 Cos'è un protocollo

I protocolli di rete sono delle **regole** che definiscono il **formato**, e l'**ordine**, la **semantica** e la **sintassi** dei "messaggi" mandati e ricevuti tra entità interconnesse tramite la rete, nonché le azioni da effettuare per **garantire un determinato servizio**. Garantire che i **protocolli** vengano usati dai **devices** appartenenti a una **rete**, è condizione necessaria per permetterne la **mutua comunicazione**.

### 1.2.1 Stack di protocolli

I protocolli di rete differiscono tra loro non solo per le loro caratteristiche, ma anche per il **livello** di appartenenza negli stack dei protocolli (ISO/OSI, TCP/IP).

Ogni protocollo è infatti associato (più o meno rigidamente) ad un livello di uno stack, un'astrazione che ordina insiemi di protocolli dall'alto verso il basso, dipendentemente dal loro ruolo. In **alto** abbiamo i protocolli più astratti e vicini all'**utente**, in **basso** quelli più vicini al **sistema fisico**. Un messaggio che sfrutta un protocollo del livello più alto, "passa"<sup>1</sup> attraverso protocolli dei layer inferiori. Ogni layer aggiunge un **header** al *payload*, fornendo informazioni specifiche, per permettere la consegna del messaggio e la sua corretta interpretazione. Il ricevente, infatti, sfrutterà gli header relativi ai vari protocolli (dal basso verso l'alto) per **interpretare** l'informazione.

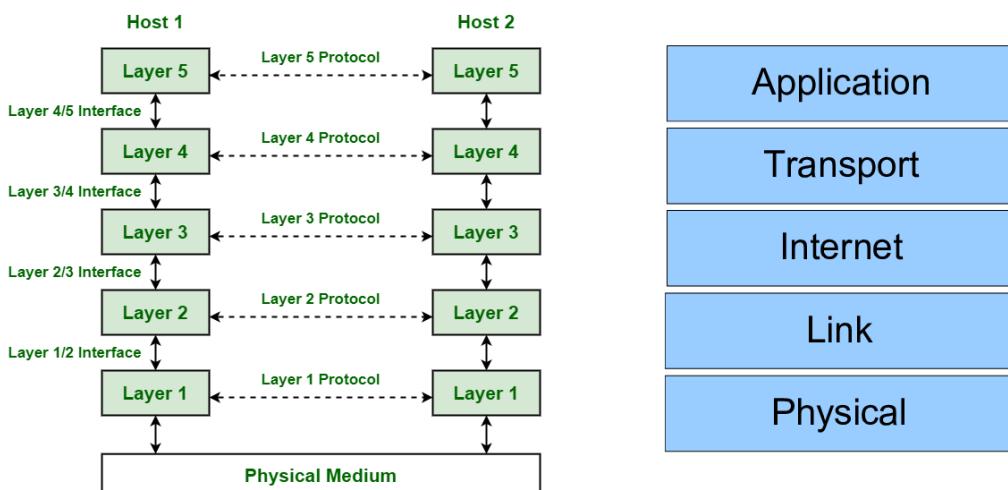


Immagine 1.1: Lo stack di nostro riferimento.

### 1.2.2 Header

Un **header** è un insieme di dati, sottoforma di bitstream, che offrono dati relativi al protocollo di appartenenza. Ogni protocollo, di un dato livello, inserirà un header con dei dati fondamentali per la trasmissione e l'interpretazione del messaggio.

### 1.2.3 Il mezzo fisico

È fondamentale che il canale fisico sia gestito in maniera opportuna: il segnale che viene mandato è soggetto alla **resistenza** del mezzo fisico e alle imperfezioni della rete con conseguenti **distorsioni**.

All'interno del mezzo fisico, ciò che viene trasferito, sono informazioni **logiche** codificate in livelli di tensione **fisici**: il sistema (solitamente) interpreta un segnale che supera i 2 volts come 1 (True), sotto i 0.8 volts 0 (False). Incluso tra 0.8 e 2 è indeterminato.

### 1.2.4 Obiettivo

L'obiettivo, nel costruire l'architettura fisica e la struttura logica di una rete, è quello di ottenere un sistema affidabile (*reliable*) e libero da errori. Nel corso dello studio delle reti, andremo a conoscere molteplici protocolli e meccanismi per aumentare la *reliability* della rete.

<sup>1</sup>Un esempio? Una richiesta HTTP (protocollo del layer applicativo), sfrutta il protocollo TCP (del layer di trasporto), che sfrutta IP (del layer di rete). Un protocollo del livello *n*-esimo sfrutterà protocolli di tutti i livelli inferiori

## 1.3 Tipi di comunicazione

Un servizio di comunicazione tra host può essere di due tipologie.

- **Connectionless system.**

Non ha un meccanismo che permette di verificare se l'informazione è arrivata al ricevente (parzialmente o totalmente), e se la connessione è stata stabilita con successo. Il sistema non prevede segnali di ACK (*acknowledgement signal*).

- **Connection oriented.**

Un po' come la doppia spunta blu di WhatsApp, **un segnale di ACK** ci permette di dare informazioni relative alla ricezione del messaggio da parte del ricevente.

### 1.3.1 Connection oriented e le sue criticità

I sistemi connection oriented offrono informazioni relative alla ricezione del messaggio, ma non sempre è affidabile. Il segnale di ACK potrebbe essere bloccato (a causa di anomalie e guasti nelle infrastrutture), potrebbe arrivare in ritardo, perdersi o essere mal-interpretato.

Non è sempre affidabile. La *reliability* è una criticità, cui gravità aumenta se ricordiamo la seguente cosa: una rete è usata per collegare dispositivi, e quindi utenti, lontani. Le informazioni sono inviate e ricevute "alla cieca".

Analogia simpatica: è l'equivalente del "ricevuto" nelle comunicazioni tramite walkie-talkie.

### 1.3.2 Temporizzazione come soluzione

Se entro un determinato lasso di tempo, non si è ricevuto il segnale di ACK relativo alla ricezione di un'informazione, l'informazione viene rimandata.

Questo meccanismo permette di **risolvere le criticità** relative alla comunicazione connection oriented.

### 1.3.3 Topologie di rete

Esistono due tipi principali di connessione.

- **Point to Point.**

Connessione tramite un canale fisico condiviso esclusivamente tra due dispositivi.

- **Broadcast.**

Connessione tramite un canale fisico condiviso tra più dispositivi. Potrebbe causare collisioni dei pacchetti con conseguenti problemi di distorsione dei dati e impossibile interpretazione dei dati ricevuti.

Le reti possono assumere varie topologie:

- Reti ad anello.
- Reti a stella.
- Reti a bus.

### 1.3.4 PAN, LAN, MAN, WAN

Le reti distribuite si differenziano tra loro per l'ampiezza della distribuzione:

- **PAN.**

Personal Area Network, pochi metri di range.

- **LAN.**

Local Area Network, 10 metri, 100 metri o 1 km.

- **MAN.**

Reti che coprono aree metropolitane di raggio dai 10 ai 100km.

- **WAN.**

Ricoprono un'intera nazione, si parla di un raggio che va dai 1000 ai 5000km.

Internet è considerabile una WAN (di WAN).

# Capitolo 2

# Application Layer

## 2.1 Introduzione al livello di applicazione

È il livello più alto nello stack dei protocolli, ed è quello più vicino all'utente. È infatti il livello delle **applicazioni di rete**: la vastità delle funzionalità offerte dalle reti, sono alla base del successo di Internet. Trasferire e-mail o file? VoIP? Comandare via remoto un altro computer? Questa è solo la punta dell'iceberg di ciò che le applicazioni di rete del livello applicativo ci permettono di fare. Come abbiamo già detto, il layer application sfrutta i layer inferiori: primo tra tutti, il livello di trasporto, di cui parleremo dopo. Ogni applicazione di rete si può basare su uno tra due paradigmi (o architetture). Il paradigma **Client-Server**, e il paradigma **Peer-to-Peer**.

### 2.1.1 Paradigma Client-Server

In questa architettura evidenziamo due attori principali: il **server**, un host sempre attivo che risponde alle richieste, e i **client**, host che mandano richieste al server. HTTP, IMAP, FTP sono alcuni tra i protocolli che sfruttano questo paradigma.

- **Server.**

(Se tutto va bene, solitamente) sempre attivo, ha un indirizzo IP<sup>1</sup> statico che facilita l'accesso al server. Il server può essere un dispositivo o una struttura di più dispositivi (come in un data center). **Riceve richieste** e fornisce risposte.

- **Client.**

**Manda richieste** al server. Può disconnettersi e (solitamente) ha un IP dinamico. In questa architettura, i client non comunicano direttamente tra loro. Ad esempio, i browser di due utenti (e quindi due client differenti) non comunicano mai direttamente tra loro.

### 2.1.2 Paradigma peer-to-peer

In un'infrastruttura di rete potrebbe esserci la necessità di far comunicare più client tra loro senza passare attraverso un server: la comunicazione **peer-to-peer** permette ciò. I client sono idem-potenti, sullo stesso livello. Ogni host (detto anche peer) può essere sia server che client. È un'architettura scalabile, aggiungere un altro peer (nodo, partecipante alla comunicazione) è un processo immediato.

I peers non devono essere sempre accesi, e i loro indirizzi IP possono cambiare.

La comunicazione peer-to-peer ha come svantaggio la **complessità** in termini **di gestione** (lo scambio di pacchetti deve essere gestito in maniera consona per evitare errori) e il **forte impatto sulla banda** dei peers coinvolti. BitTorrent segue questo paradigma.

### 2.1.3 Socket

Un socket è un'interfaccia software che permette ad un processo di mandare dati verso la rete. Non sarebbe sbagliato immaginarli come delle porte, che possono essere aperte proprio tramite software: nello specifico, sono aperte tramite delle **primitive del sistema operativo**, il quale si occuperà della loro gestione indipendentemente dall'applicazione. I socket sono quindi a cavallo tra il livello applicativo e quello di trasporto, il quale che non sarà oggetto di interesse (perlomeno, approfondito) da parte di un ipotetico sviluppatore di applicazioni di rete. Ogni connessione, coinvolge sempre due socket, uno per lato.

---

<sup>1</sup>Un indirizzo IP permette di identificare univocamente un dispositivo su una rete.

## 2.1.4 Indirizzi IP e porte.

Un host runna  $n$  processi. Un indirizzo IP (IPv4 o IPv6) identifica un host, ma non il processo coinvolto nella connessione: ne consegue che l'IP è identificativo (logico) per gli host, ma non per i processi degli host. L'identificazione del processo di una macchina con IP noto, avviene tramite il numero di porta.

IP address: 128.119.245.12  
Port number: 80

In questo esempio, la porta 80 è quella dedicata al protocollo HTTP, e sarà la porta di destinazione per le richieste ad un web server.

### In merito alle porte

Un numero di porta è un intero a 16 bit, e si hanno quindi  $2^{16} - 1 = 65536$  porte possibili. I numeri di porta sono divisi in tre gruppi:

- **Well Known Ports.**  
(0 - 1023) per servizi di sistema.
- **Registered Ports.**  
(1024 - 49151) sono assegnati dall'ICANN (Internet Corporation for Assigned Names and Numbers) per usi e servizi specifici.
- **Dynamic and/or Private Ports.**  
(49152-65535)

### Le porte dei protocolli studiati nel corso

20 FTP – Data Port  
21 FTP – Command Port  
22 SSH  
23 Telnet  
25 SMTP  
53 DNS  
80 HTTP  
110 POP3  
161 SNMP (Agent)  
162 SNMP (Manager)  
443 HTTPS  
465 SMTP  
994 POP3

## 2.1.5 Comunicazione inter-process e non

Un processo è definito come un programma eseguito da un host.

- I processi di uno stesso host possono comunicare tra loro tramite regole di **IPC**, definite dal sistema operativo.
- Due processi di due host differenti comunicano scambiando pacchetti, sfruttando i protocolli del transport layer e quelli inferiori.

## 2.2 Definizione di un protocollo applicativo

Un protocollo del livello applicativo è caratterizzato dalle seguenti caratteristiche

- **Tipo di messaggi.**

Come messaggi di richiesta e di risposta.

- **Sintassi dei messaggi.**

I campi dei messaggi. Nei protocollo HHTP e SMTP, ad esempio, sono definiti dei campi ben precisi (alcuni di loro obbligatori).

- **Semantica dei messaggi.**

Il significato delle informazioni all'interno dei campi e come interpretarli.

- **Regole** sul come e quando mandare e rispondere ai messaggi.

Sono spiegati in **documenti tecnici RFC** (Request for Comments) pubblicati dall'IETF (Internet Engineering Task Force) e dall'Internet Society (ISOC). **Definiscono standard**, protocolli e best-practices per il funzionamento di Internet e delle reti di comunicazione.

I protocolli possono essere **aperti** (come HTTP o SMTP) o **proprietari** (come quello di Skype).

## 2.3 Trasferimento dati - TCP e UDP

Per offrire una spiegazione completa dell'application layer, diamo una breve introduzione a quelli che sono i protocolli del livello di trasporto maggiormente usati (quali TCP e UDP) per garantire il trasferimento delle informazioni. Ogni protocollo di trasporto ha le proprie specifiche, ed è scelto in maniera **opportuna** per l'applicazione che lo sfrutta:

- **Integrità dei dati.**

Posso permettermi di perdere dei dati? O necessito di un trasferimento totalmente lossless?

- **Throughput.**

Quantità di dati trasferiti in un'unità di tempo. Larghezza di banda e distanza tra gli host influenza questo valore.

- **Timing.**

Posso avere della latenza o devo offrire un servizio quanto più responsivo e istantaneo possibile?

- **Sicurezza.**

I dati possono essere trasmessi in chiaro? o richiedono una forma di crittografia?

Ad esempio, audio o video in **streaming** perdonano un **throughput variabile**, più basso e anche la **perdita di pacchetti**, a discapito della qualità. Una perdita di pacchetti è inaccettabile per il trasferimento di file e l'alta latenza non è ottimale per video conferenze o gaming.

TCP e UDP sono ad oggi protocolli del layer di trasporto più importanti. Introduciamoli in breve

### 2.3.1 TCP - Transmission Control Protocol

Offre le seguenti specifiche:

- **Trasporto affidabile.**

(*Reliable transport*) tra mittente e destinatario, tramite meccanismi di *ACKnowledgment* dei pacchetti.

- **Controllo del flusso (flow control).**

Il mittente, cercherà di **non sovraccaricare** il **ricevente**, per minimizzare la perdita di pacchetti.

- **Controllo della congestione (congestion control).**

La banda viene limitata quando la **rete è sovraccaricata**.

- **Connection oriented.**

È richiesto un setup tra client e server per effettuare la connessione. Usa un three-way handshake per stabilire una connessione.

- **Non fornisce garanzie** sulla latenza, tantomeno sul minimo throughput e sulla sicurezza

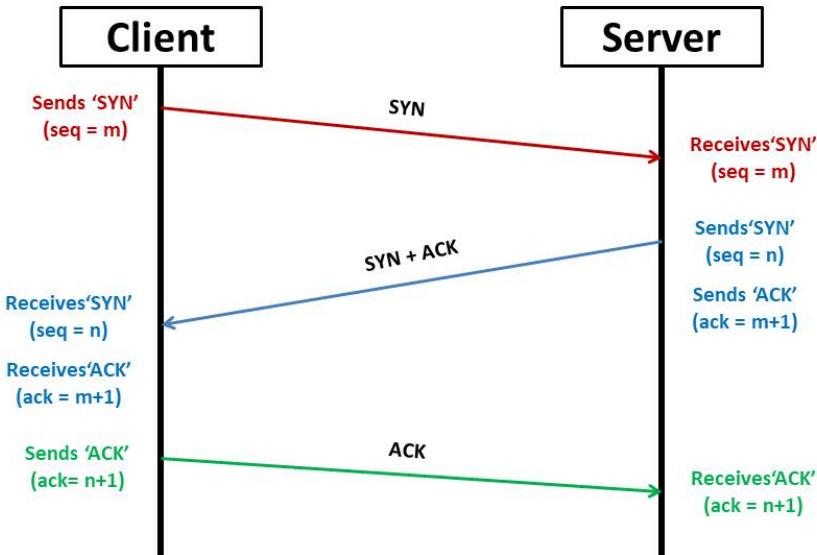


Immagine 2.1: Three-way handshake

### 2.3.2 UDP - User Datagram Protocol

È un protocollo minimal, scarno e *unreliable*.

- **Connectionless.**

Non ha alcun tipo di handshake che stabilisca la connessione tra i due host.

- **Unreliable.**

Nessuno scambio di ACK.

- **Alcun controllo del flusso.**

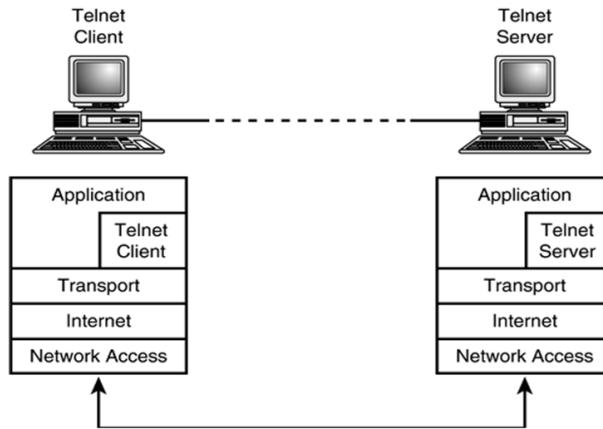
UDP non gestisce il proprio throughput in funzione della congestione della rete, o del sovraccarico del receiver.

- **Alcuna garanzia sul throughput minimo.**

UDP non è un protocollo sicuro: scopriremo però come il suo essere "ridotto all'osso", *barebones*, sia il suo più grande punto di forza. È usato nei protocolli applicativi VoIP, streaming audio-video e gaming.

## 2.4 Telnet

Telnet (da TErminAL NETwork) è un protocollo applicativo che permette di controllare il terminale di un'altro host **da remoto**.



È basato sul paradigma client-server (dove il pc remoto è il server), usa **TCP** la **porta 23** come porta destinataria (verso il server). Non è un protocollo sicuro: Telnet trasmette i dati (anche di autenticazione) in chiaro, l'autenticazione non è sempre obbligatoria, ed è vulnerabile a sniffing e man-in-the-middle. **SSH** (*Secure Shell Protocol*) è simile al protocollo Telnet, ma sicuro, non trasmettendo i dati in chiaro. SSH usa la porta 22.

### 2.4.1 Funzionamento di Telnet

Basato sul protocollo di trasferimento TCP, Telnet sfrutta il three-way handshake per stabilire la connessione.

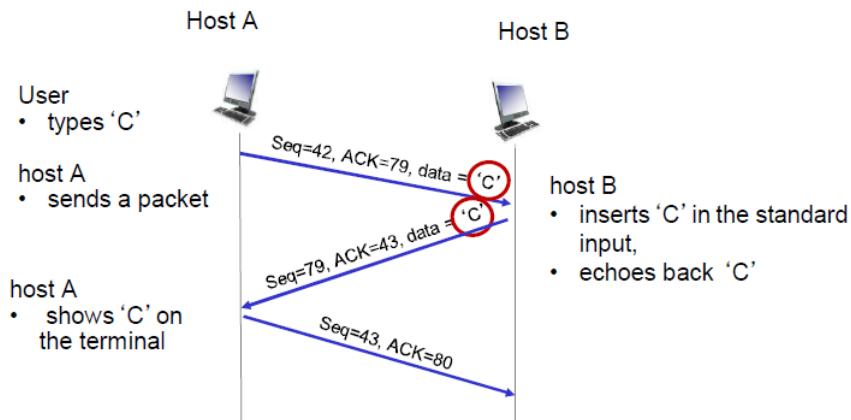


Immagine 2.2: Esempio di Telnet

Un fatto interessante, è che Telnet può essere usato come strumento di debug per Server HTTP o Mail Server, inserendo come porta destinazione, una porta diversa dalla 23.

### 2.4.2 Remote Desktop Protocol

È un protocollo con presupposti simili a quelli di Telnet, ma che permette la trasmissione dell'intera interfaccia del dispositivo remoto.

#### Telnet in breve

Terminale remoto, TCP porta 23. Architettura client-server. Autenticazione trasmessa in chiaro, quindi non sicuro, sconsigliato e deprecato grazie ad SSH. SSH sfrutta la porta 22.

## 2.5 FTP - File Transfer Protocol

È un protocollo utilizzato per lo **scambio di file tra due host**. Basato sul paradigma **client-server**, sfrutta **TCP**. Offre inoltre un **meccanismo di autenticazione** (che avviene dopo lo stabilimento della connessione, ma precedentemente a qualsiasi scambio dati).

### 2.5.1 Client-Server

Col protocollo FTP, il client carica o scarica file verso o dal server. Il server ospita i file e gestisce le richieste. La porta di controllo è la porta 21, su cui il server riceve le richieste. La porta del client usata per lo scambio dei dati è casuale, e trasmessa tramite il comando **PORT**. La porta del server per lo scambio dati è la porta 20.

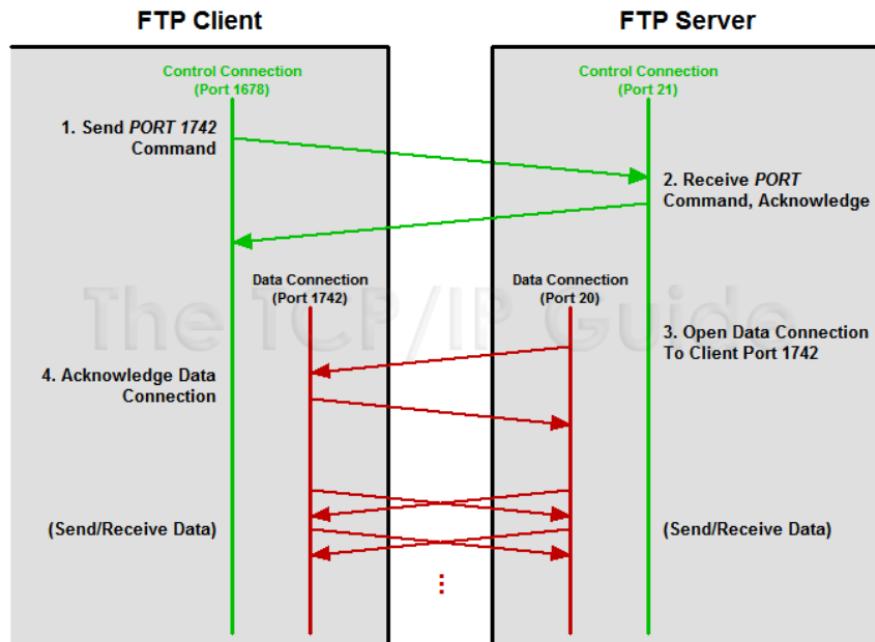


Immagine 2.3: FTP

### 2.5.2 Vulnerabilità del protocollo FTP

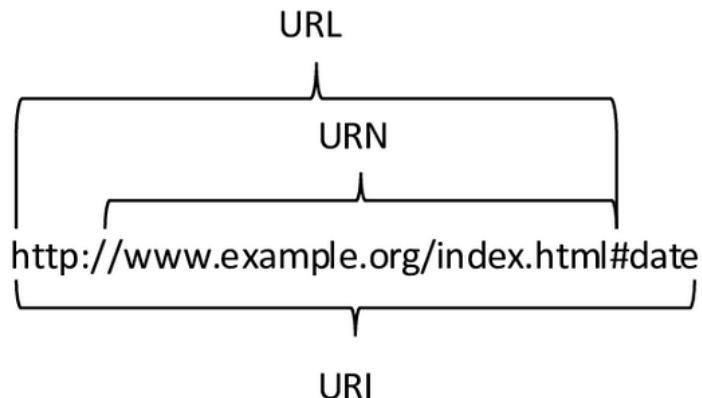
- Abilitare l'accesso all'intero file sistem tramite FTP è una mossa rischiosa: bisogna regolare quali porzioni rendere accessibili.
- Le credenziali di accesso e i dati (di default) sono trasmessi in chiaro, rendendo FTP vulnerabile ad attacchi man-in-the-middle e allo sniffing. Il protocollo **SFTP** risolve il problema implementando l'encryption di SSH. Usa la stessa porta di SSH, la porta 22.

#### FTP in breve

Upload e download di file su host remoto, TCP porta 21 (porta di controllo) e TCP porta 20 (porta dati) del server. Architettura client-server. Autenticazione e dati trasmessi in chiaro, quindi non sicuro, sconsigliato e deprecato grazie ad SFTP. SFTP sfrutta la porta di SSH, la porta 22.

## 2.6 HTTP - HyperText Transfer Protocol

**HTTP** permette ad un host di richiedere delle risorse, chiamate oggetti, ad un Web Server. Questi oggetti possono essere pagine HTML, file multimediali di vario tipo (e, col modello DHTML, affiancate alle pagine HTML, abbiamo anche fogli di stile CSS e file Javascript). Per accedere a queste risorse usiamo un URL (*Uniform Resource Locator*). È quindi il protocollo alla base delle pagine Web, e segue (banalmente) l'architettura Client-Server. La porta dedicata al protocollo, lato server, è la porta 80, ed è la porta da cui il server riceverà le richieste. Ricordiamo che la porta sorgente del client, è stabilita (casualmente) dall'OS.



### 2.6.1 In cosa consiste la connessione HTTP

HTTP utilizza TCP e il suo Three-Way Handshake.

1. Il client **richiede la connessione TCP sulla porta 80** del server.
2. Il **server accetta** la connessione TCP da parte del client.
3. **Messaggi HTTP vengono scambiati** tra il browser (del client) e il web server (del server).
4. **Si chiude la connessione TCP.**

Questi passi avvengono in un tempo chiamato *Round Trip Time (RTT)*.

### 2.6.2 Versioni di HTTP

Il vasto utilizzo di HTTP, ha portato alla creazione di molteplici versioni nel corso degli anni.

- **HTTP/1.0**

Ogni trasferimento di oggetto implica un'apertura e chiusura di una connessione TCP. **Per ogni oggetto, una connessione TCP dedicata.** Overhead per ogni oggetto.

- **HTTP/1.1**

Usa una **connessione TCP persistente**. La connessione viene **aperta all'inizio del trasferimento** di tutti gli oggetti, e **chiusa alla fine**. Meno overhead, ma ogni download avviene in maniera sequenziale.

- **HTTP/2**

Comprime gli header e effettua i download in parallelo. Riduce ulteriormente le tempistiche.

- **HTTP/3**

Il più recente, introduce numerose innovazioni, tra cui l'uso del protocollo di trasferimento QUIC (creato su UDP!).

#### HOL Blocking

L'**Head-of-Line blocking** è un fenomeno che avviene quando più richieste di risorse HTTP sono in coda. In HTTP/1.1, se  $O_1$ , una richiesta di un oggetto molto grande, viene inserita prima della richiesta di 3 oggetti molto piccoli  $O_2, O_3, O_4$ , la priorità verrà data all'oggetto  $O_1$ , in quanto quest'ultimo è situato in testa. In HTTP/2, un insieme di algoritmi di **valutazione della priorità** e **slicing** degli oggetti in porzioni più piccole, risolve l'HOL blocking minimizzando i tempi di attesa per ogni oggetto, migliorando la *User Experience*.

### 2.6.3 Struttura di una richiesta HTTP

- **Request Line.**
  - **Metodo.**  
Indica il tipo di richiesta effettuata dal client.
  - **URL.**  
Esplicita la risorsa desiderata.
  - **Versione.** Esplicita la versione del protocollo HTTP.
- **Header lines.**  
Più righe contenenti coppie Nome campo: Valore, una per riga.
- **Body.**  
Che permette di specificare ulteriori parametri.

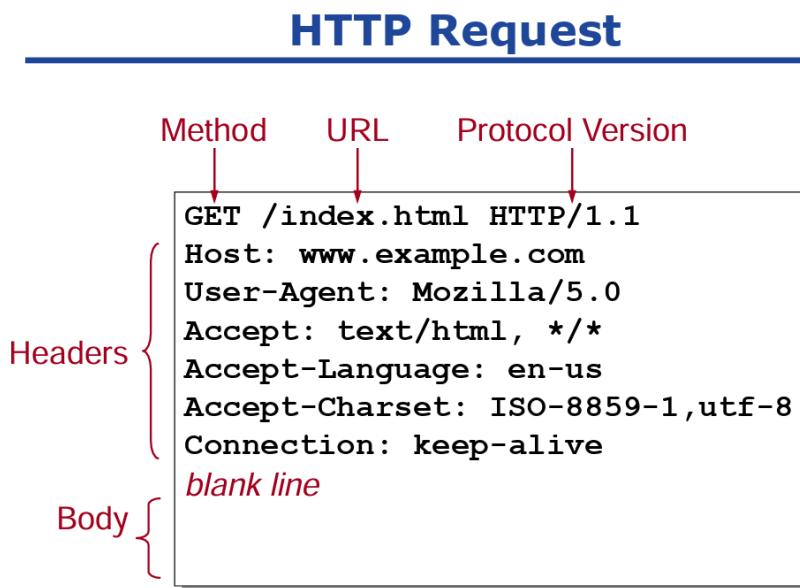


Immagine 2.4: Esempio di richiesta HTTP

#### Alcuni metodi richieste HTTP

- **GET.**  
Richiede dei dati dal server.
- **POST.**  
Usato per mandare dei dati d'input da un client.
- **HEAD.**  
Richiede esclusivamente l'header che sarebbe mandato da una richiesta HTTP GET ad un determinato URL.
- **PUT.**  
Usato per effettuare l'upload dei dati. Effettua un rimpiazzamento totale di un file che esiste a un determinato URL.

## 2.6.4 Struttura di una risposta HTTP

- **Status Line.**
  - **Version.**  
Ribadisce la versione del protocollo HTTP in uso.
  - **Codice di stato.**  
Fornisce dettagli sulla corretta (o meno) elaborazione di una richiesta HTTP.
  - **Messaggio di stato.** Esprime in linguaggio naturale il significato dello Status Code.
- **Header lines.**  
Più righe contenenti coppie Nome campo: Valore, una per riga.
- **Body.**  
Contiene i dati richiesti dal client nella rispettiva HTTP Request.

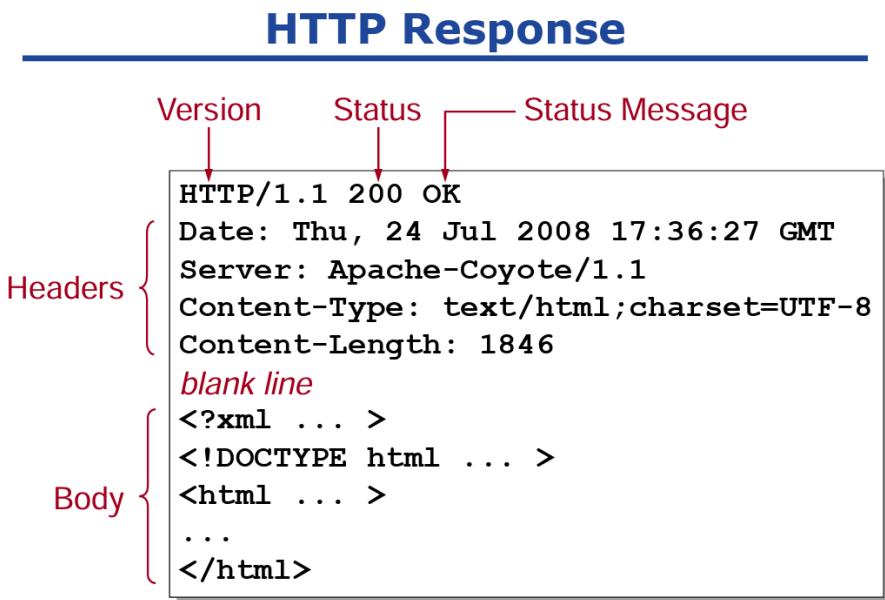


Immagine 2.5: Esempio di risposta HTTP

### Alcuni codici di risposta HTTP

Information codes.

100 Continue  
Success codes.  
200 OK  
203 Non-Authoritative Information  
204 No Content

Redirection Codes.

305 Use Proxy

Client Error Codes.

400 Bad Request  
403 Forbidden  
404 Not Found  
405 Method Not Allowed

Server Error Codes.

500 Internal Server Error  
505 HTTP Version Not Supported

## 2.6.5 Parentesi Stateless, Stateful e Cookies

Il protocollo HTTP è di default, un protocollo di tipo **stateless**. Tuttavia, con l'introduzione dei **cookies**, HTTP può memorizzare degli stati, diventando **stateful**<sup>2</sup>.

- **Stateless.**

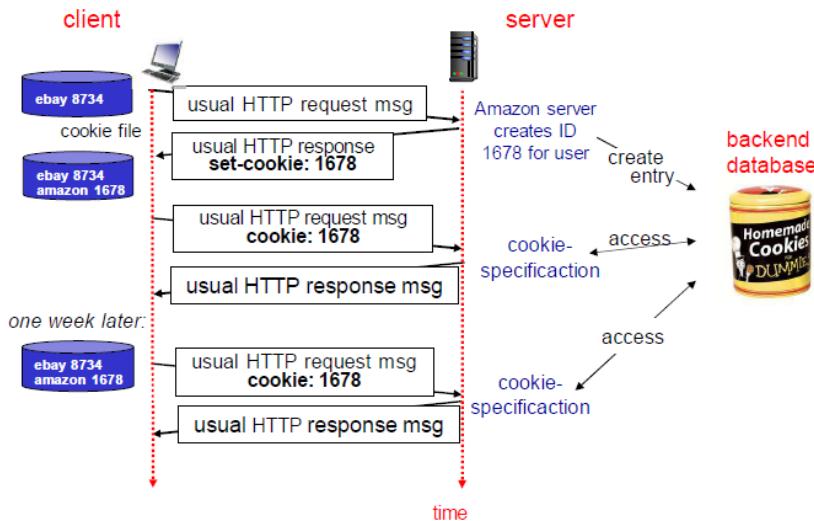
Il comportamento di un protocollo stateless **non cambia in funzione di stati intermedi** della attuale sessione e delle sessioni precedenti.

- **Stateful.**

Un protocollo stateful **salva informazioni** relative a stati intermedi e richieste precedenti, **costruendo uno storico degli stati**, e il suo comportamento varia. Un crash del server potrebbe portare a storici di attività e stati inconsistenti. In tal caso, bisognerà ricostruire lo storico. Tra i due tipi di protocolli, questo è il più complesso da gestire.

## 2.6.6 I cookies

Permettono ad un server associato al Web Server di memorizzare alcuni dei dati relativi alle sessioni dell'utente che li accetta. Una volta che un utente accetta i cookies su una piattaforma, viene creato un record (con chiave primaria uguale al cookie id) in un database back-end relativo al sito, che può essere quello del sito stesso o meno. I cookies permettono di implementare **carrelli online** e **accesso semplificato** all'interno di pagine web. Nel pratico, i cookies sono dei veri e propri file gestiti dal proprio web browser.



### I cookies - rappresentano un rischio per la privacy degli utenti?

Seppur apparentemente convenienti, i cookies sono in tutto e per tutto **un rischio per la privacy** degli utenti che li accettano. In particolare, se i **cookie di prima parte** tracciano azioni dell'utente solo sul sito da esso visitato, i tanto sentiti **cookie di terze parti** permettono di tracciare le azioni degli utenti anche a siti **non visitati in maniera diretta**, ma semplicemente in quanto embedded in altri siti.

1. GET da parte del client ad un sito *A*: cookie di terze parti vengono accettati.
2. Le informazioni prese dal cookie del sito *A* vengono prelevate anche da siti di terze parti (integriti nella pagina, embedded), e quindi finiscono anche nel database back-end di un sito *B*.
3. Supponiamo il sito esterno sia embedded in un altro sito *C*. Se lo stesso client è che ha visitato il sito *A*, visiterà anche il sito *C*, il sito *B* otterrà entrambe le informazioni, potendo effettivamente costruire uno storico dell'attività dell'utente.

Gli utenti e le loro attività online, possono essere tracciate, soprattutto se ai cookies associamo meccanismi di identificazione univoci! Il **Regolamento Generale sulla Protezione dei Dati** stabilisce e regolamenta l'integrazione dei cookies nei siti web, garantendo che i cookies *analytics* e non essenziali (non-tecnici) possano essere bloccati.

<sup>2</sup>Un protocollo stateless può comunque avere un log delle azioni relative alle sessioni precedenti, ma ciò non cambierà il suo comportamento. Un web-server PHP hostato con Apache non diventa magicamente stateful se aggiungiamo un log. Rettifica importante, soprattutto in sede d'esame.

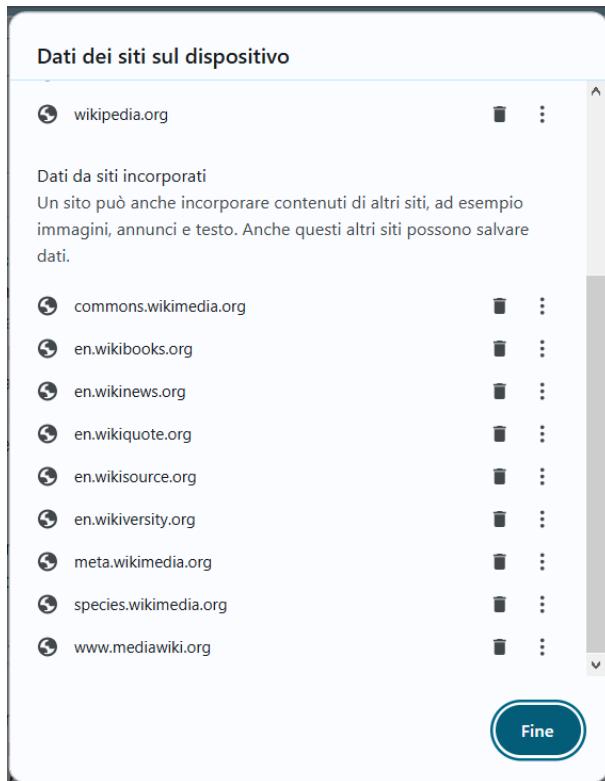


Immagine 2.6: Cookies di pagine embedded a wikipedia.org

### 2.6.7 DHTML

È possibile anche costruire una pagina web dinamica col Dynamic HTML: è un insieme di tecnologie che permettono di costruire pagine web con rendering dinamico in funzione dell'utente. Queste tecnologie includono JavaScript e fogli di stile CSS.

### HTTP in breve

HyperText Transfer Protocol permette agli host di richiedere oggetti (pagine HTML, media e altro) ad un Web Server, e al Web server di rispondere, tramite opportune richieste e risposte HTTP. Usa TCP (sfruttato in maniera differente tra le varie versioni di HTTP), sulla porta 80 del server. Architettura client-server. È stateless, ma possono essere annesse funzionalità stateful tramite l'ausilio di cookies, con i pro e contro del caso.

## 2.7 Server proxy

È un server intermedio che permette di **controllare, filtrare e mascherare il traffico** da parte di uno o più client. Nello specifico:

1. I client mandano le richieste al proxy.
2. Il proxy le invia al server destinazione.
3. Il server riceve la richiesta del proxy.
4. Il server invia la risposta al proxy.
5. Il proxy invia la risposta del server al client originale.

I proxy server si prestano alla perfezione per mascherare IP e monitorare le richieste da parte degli host che li usano.

## 2.8 SMTP

Il **Simple Mail Transfer Protocol** è un protocollo di invio di e-mail. Si occupa del trasferimento dei messaggi dal mail server del mittente a quello del destinatario. Usa il protocollo **TCP** e sulla **porta 25**.

- **User Agent.**

Software che permette agli utenti di leggere, rispondere, inoltrare, salvare e comporre messaggi. Sono i comuni Microsoft Outlook, Apple Mail, Google Gmail. Permettono di inserire o leggere messaggi sul mail server.

- **Mail Server.**

È il server di posta da cui vengono recuperati, prelevati e inviati i messaggi. Se i due client non sono connessi allo stesso mail server, allora avviene un trasferimento tra i due mail server.

### 2.8.1 Le tre fasi di SMTP

1. **Handshaking.**

Connessione alla **porta 25** del mail server, il client si identifica tramite il comando **HELO**, il server invia un messaggio di benvenuto.

2. **Transfer of messages.**

> **MAIL FROM:** <[indirizzo@email.com](mailto:indirizzo@email.com)> per specificare email del mittente.  
> **RCPT TO:** <[destinatario@email.it](mailto:destinatario@email.it)> per specificare i destinatari.  
> **DATA** per iniziare a scrivere la propria mail. Si può andare a capo, una mail si chiude con una riga con esclusivamente il carattere ". ". Si può ripetere questo passaggio per scrivere più mail: queste verranno mandate tutte assieme con la **stessa sessione TCP** persistente.

3. **Closure.**

> **QUIT.** Viene chiusa la connessione.

Header	Significato
To:	Indirizzi e-mail del destinatari primari
Cc:	Indirizzi e-mail dei destinatari secondari
Bcc:	Come Cc: ma gli indirizzi e-mail non sono visibili
From:	Indirizzo e-mail di chi ha scritto il messaggio:
Sender:	Indirizzo e-mail dell'effettivo mittente
Received:	Ogni server attraverso cui passa la mail lascia una traccia in questo campo
Return Path:	Può identificare un cammino a ritroso verso il mittente
Date:	Data e ora di invio
Reply-to:	Indirizzo e-mail a cui le risposte dovrebbero essere inviate
Message-id:	Id univoco del messaggio
In-Reply-To:	Id del messaggio a cui si ha risposto
References:	Altri id di messaggi
Keywords:	Parole chiave scelte dall'utente
Subject:	Breve riepilogo del contenuto del messaggio

Tabella 2.1: Tutti gli header dell'SMTP. In blu, i campi obbligatori

### Sugli allegati

Gli **allegati** delle e-mail sono trasferiti tramite il protocollo **MIME** (Multipurpose Internet Mail Extension).

## 2.8.2 Protocolli del mail server - POP3 e IMAP

Erroneamente, si potrebbe pensare che il Mail Server giri localmente all'interno della macchina dell'utente. Tuttavia, questo obbligherebbe ogni utente a tenere il proprio dispositivo sempre acceso e in ascolto sulla rete, per assicurare che non venga persa alcuna mail: il server non sarebbe sempre attivo, caratteristica che nel paradigma client-server è fondamentale. Deleghiamo quindi la gestione del *pull* delle e-mail dal Mail Server a dei protocolli, riservando il *push* al protocollo SMTP.

- **IMAP.**

Internet Message Access Protocol. Memorizza le e-mail sul mail server. Questo permette l'accesso alle stesse e-mail da più dispositivi, e grava meno sulla memoria. Usa TCP sulla porta 143 (non crittografata) e 993 (crittografata).

- **POP3.**

Post Office Protocol. Memorizza le e-mail in locale sulla macchina dell'utente, rimuovendole dal Mail Server, non supportando la sincronizzazione delle e-mail su più dispositivi. Usa TCP sulla porta 110 (non crittografata) e 995 (crittografata).

Feature	POP3	IMAP
Where is protocol defined?	RFC 1939	RFC 2060
Which TCP port is used?	110	143
Where is e-mail stored?	User's PC	Server
Where is e-mail read?	Off-line	On-line
Connect time required?	Little	Much
Use of server resources?	Minimal	Extensive
Multiple mailboxes?	No	Yes
Who backs up mailboxes?	User	ISP
Good for mobile users?	No	Yes
User control over downloading?	Little	Great
Partial message downloads?	No	Yes
Are disk quotas a problem?	No	Could be in time
Simple to implement?	Yes	No
Widespread support?	Yes	Growing

Immagine 2.7: Protocolli POP3 e IMAP a confronto

### SMTP, POP3 e IMAP in breve

- **Simple Mail Transfer Protocol** offre modo al client di mandare, *pushare*, email ad un Mail Server. Usa TCP sulla porta 25 del mail server.
- **Postal Office Protocol (POP3)** permette al client di scaricare, *pullare*, in locale le proprie e-mail, ma solo su un dispositivo. Nel mail server non rimarrà traccia di email che sono state prelevate con questo protocollo. Usa TCP sulla porta 110 del mail server.
- **Internet Message Access Protocol (IMAP)** permette al client di accedere alle e-mail presenti nella propria casella postale del mail server. Non grava sulla memoria del dispositivo, e permette l'accesso da più dispositivi. Usa TCP sulla porta 143 del mail server.

## 2.9 DNS - Domain Name System

Gli indirizzi IP identificano in maniera univoca gli host in rete e la loro locazione: la struttura di un indirizzo IP, fornisce infatti informazioni sulla struttura della (sotto)rete e la **posizione dell'host** all'interno di essa. Gli IP **non sono facili da ricordare**, per un umano, in quanto costituiti esclusivamente da **caratteri numerici**. Gli *hostname* nascono con lo scopo di associare **un nome a un IP**: diventa però necessario introdurre un meccanismo che permetta la traduzione degli hostname in indirizzi IP. Nasce così il DNS.

```
151.97.240.12  <-  www.dmi.unict.it
151.97.240.4   <-  www.unict.it
151.97.6.236   <-  www.ing.unict.it
151.97.252.132 <-  galileo.dmi.unict.it
```

Il protocollo DNS supporta inoltre:

- **Alias names** sugli hostname canonici.
- Gestione dei reindirizzamenti nel caso di cambio di indirizzo IP del sito.
- Distribuzione del carico su più server (tra server DNS primari e secondari).

### 2.9.1 DNS - Transport Layer

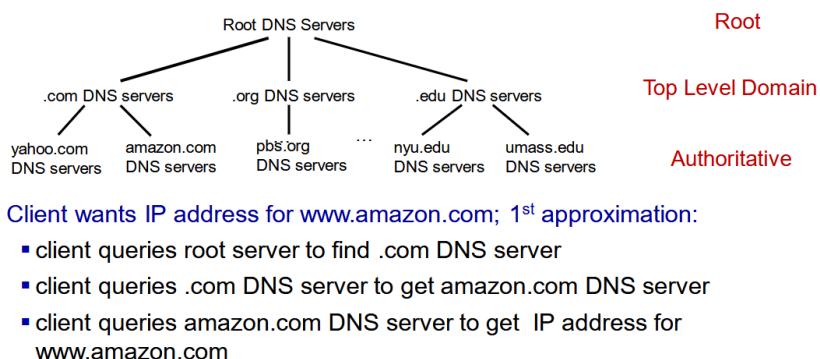
Le richieste **sono mandate alla porta 53** dei server DNS. Il protocollo usa sia **UDP**, che **TCP**. Per le banali query del servizio, si predilige UDP, e TCP è riservato per risolvere circostanze particolari, come informazioni troncate. TCP è sempre usato durante i **zone transfer**, ossia quando un DNS primario vuole effettuare una copia dei suoi record ad uno o più server DNS secondari.

### 2.9.2 Struttura del DNS

È un **database distribuito**, con una **struttura gerarchica** ad albero che **velocizza la ricerca dell'IP**. Individuiamo le classi di DNS della gerarchia:

- **Root server.**  
Contengono gli indirizzi IP del TLD server. I root server sono 13, gestiti da 12 aziende, e nel complesso si hanno più di 1000 copie di questi record.
- **Top-Level Domain Server.**  
Si occupano degli indirizzi IP dei server autoritativi, e sono organizzati per domini di primo (.com .gov .org .net .edu) e di secondo livello (.it .uk .us).
- **Authoritative Domain Name Server.**  
Effettuano l'ultima associazione indirizzo IP - hostname.

Su questa struttura gerarchica, aggiungiamo anche una **distribuzione del carico di lavoro** tra più server (**primari e secondari**), **meccanismi di caching** vari e **distribuzione di questi server a scala mondiale**: questi fattori rendono il DNS uno dei servizi più **robusti** (gestione non centralizzata), **affidabili e veloci** della rete.



### 2.9.3 Primary e secondary DNS

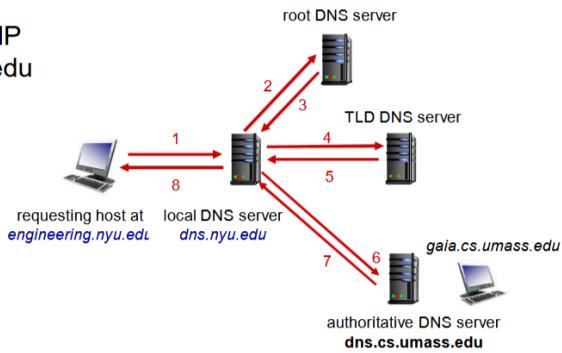
A causa dell'alto numero di richieste ai server DNS, a ogni **DNS primario**, contenente le copie originali dei record in modalità lettura e scrittura, **coincidono molteplici DNS secondari**, contenenti copie di sola lettura. Lo scopo è quello di **bilanciare il carico** di lavoro tra i DNS.

### 2.9.4 Name resolution - Modo iterativo

**Example:** host at  
engineering.nyu.edu wants IP  
address for gaia.cs.umass.edu

**Iterated query:**

- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



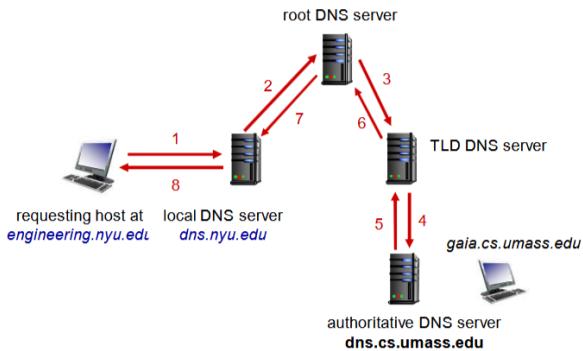
Le query iterative offrono una strategia di controllo che **distribuisce bene il carico** sui vari server. Tuttavia, il client dovrà gestire molte più richieste, una per DNS coinvolto. (N.B. la prima query dell'immagine è ricorsiva!)

### 2.9.5 Name resolution - Modo ricorsivo

**Example:** host at engineering.nyu.edu wants IP address for  
gaia.cs.umass.edu

**Recursive query:**

- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?



Le query ricorsive offrono una **risposta diretta** al client **senza ulteriori richieste** da mandare. Ogni server comunicherà con quello successivo e quello precedente (**favorendo il caching**, i server DNS si scambieranno le loro risposte), con una logica molto più semplice di quella iterativa. Tuttavia, a causa della ricorsione, si avrà **maggior latenza** (soprattutto quando non avvengono cache hit), e **maggior carico di lavoro** sulla parte alta della gerarchia dei server DNS.

### 2.9.6 Caching e Updating DNS

Ogni volta che un qualunque DNS Server (name server) riceve una risposta e "impara" come mappare una determinata informazione, la **memorizza nella propria cache** per evitare future richieste nella gerarchia di risoluzione. Ciò permette di ridurre il carico sui server e migliorare le prestazioni. Tipicamente, i TLD server vengono memorizzati nella cache dei DNS locali per ridurre le query ai root. Le entry delle cache (anche quelle locali) devono essere **aggiornate periodicamente** dopo un tempo identificato dal **TTL** (Time To Live): esso è un valore associato ai record DNS che indica per quanto tempo un resolver DNS **può mantenere nella cache una risposta prima di doverla aggiornare**. Il caching velocizza significativamente il funzionamento dei DNS. Un cache hit eviterà la richiesta a tutti DNS server dei livelli inferiori della gerarchia: per questo la cache di ciascun DNS è il **primo elemento che viene consultato per qualsiasi richiesta**. Una cache hit è sempre un grande vantaggio in termini di velocità.

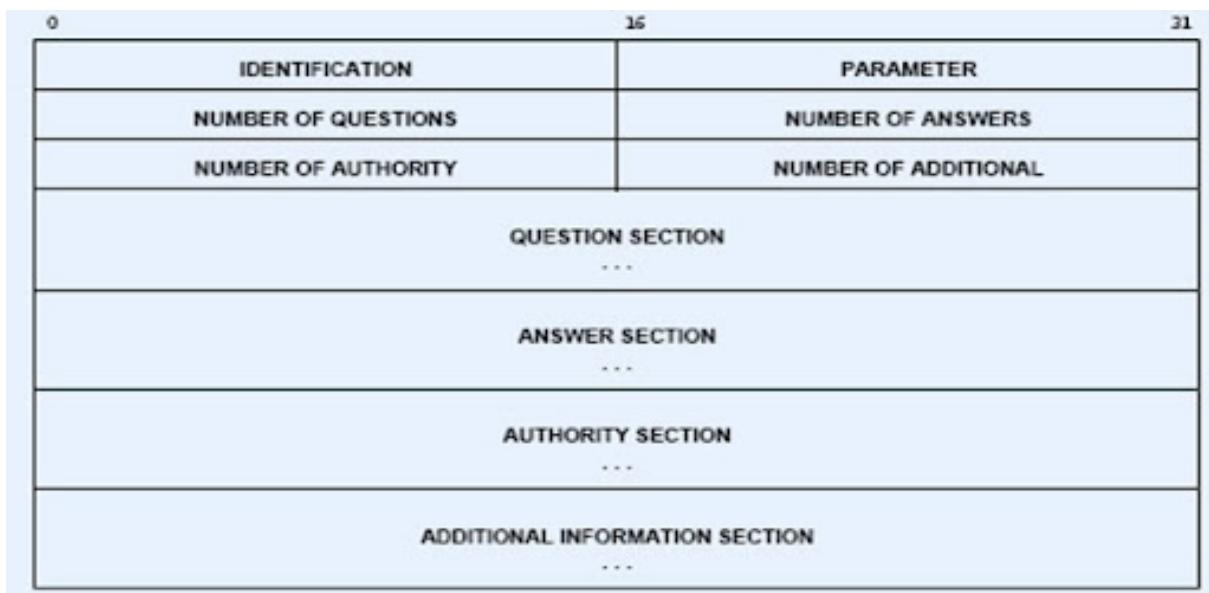
## 2.9.7 Forma di un record di un DNS

RR format: (name, value, type, ttl)

- type=A:  
name è l'hostname, value è l'IP address
- type=NS:  
name è il dominio, value è l'hostname del server autoritativo su questo dominio.
- type=CNAME  
Canonical Name. name è un alias per il nome canonico value.
- type=MX  
value è il nome del mail server associato all'hostname name.

## 2.9.8 Forma di un messaggio DNS

Formato uguale tra query e risposte.



- **Sezione di intestazione.**

Sono i primi 12 byte, e contiene vari campi.

- Identificazione, 16 bit. Permette di associare risposte a richieste.
- Flag, 16 bit. Tra questi, un bit che specifica query/reply.
- Numero di domande, numero record di risposta, numero record autoritativi, numero record addizionali. Specificano il numero di record e domande delle prossime sezioni. Ognuno è un numero a 16 bit.

- **Sezione delle domande.**

Includere informazioni relative alle richieste che stanno per essere effettuate, un campo con il nome richiesto e uno con il tipo di domanda sul nome richiesto.

- **Sezione delle risposte.**

Presente nei messaggi di risposta. I record contengono al loro interno le informazioni specificate nel paragrafo precedente. Una risposta può tornare più indirizzi.

- **Sezione autoritativa.**

Contiene record di altri server autoritativi.

- **Sezione aggiuntiva.**

Contiene altri record utili correlati alla richiesta.

### 2.9.9 nslookup

Questo programma permette di effettuare richieste DNS a un determinato dominio.

```
> nslookup
Server predefinito: vodafone.station
Address: **indirizzo mac**

> wikipedia.com
Server: vodafone.station
Address: **indirizzo mac**

Risposta da un server non autorevole:
Nome: wikipedia.com
Addresses: 2a02:ec80:600:ed1a::3          //IPv6
           185.15.58.226                  //IPv4

> overleaf.com
Server: vodafone.station
Address: **indirizzo mac**

Risposta da un server non autorevole:
Nome: overleaf.com
Address: 34.120.52.64

> overleaf.it
Server: vodafone.station
Address: **indirizzo mac**

*** vodafone.station non è in grado di trovare overleaf.it: Non-existent domain
```

### 2.9.10 Problemi di sicurezza e criticità dei DNS

- **DDoS attacks su Root Server.**

Consiste nel sovraccaricamento di un DNS root tramite richieste spazzatura. Il traffic filtering può essere una soluzione per filtrare il traffico osservando la semantica degli URL. Questo approccio potrebbe rifiutare richieste da utenti reali. Al giorno d'oggi, a questo scopo, vengono sfruttate alcune tecniche d'intelligenza artificiale. Un'altra soluzione può essere mantenere una lista di IP di TLD server notoriamente non malevoli.

- **Man-in-the-middle.**

Un attacco che consiste nell'intercettare le richieste tra i DNS.

- **DNS poisoning/DNS spoofing.**

Consiste nel sostituire risposte DNS effettive con risposte fittizie, ottenendo un **reindirizzamento errato**. I danni da DNS poisoning si propagano a lungo termine, a causa del **caching**.

### DNS in breve

Risoluzione degli hostname in indirizzi IP. UDP (ma anche TCP), porta 53 del server ricevente. Architettura client-server. Sfrutta un sistema di server distribuiti, e organizzati in maniera gerarchica, per facilitare la ricerca. Meccanismi di caching, distribuzione del carico su server primari e secondari, e in generale l'alto numero di server nel mondo, offrendo un sistema solido e veloce.

## 2.10 SNMP

Il **Simple Network Management Protocol** è un protocollo di rete **connectionless** che permette la **gestione di una rete** e il **monitoraggio dei dispositivi** ad essa collegati. Gli **amministratori** possono raccogliere informazioni sulle prestazioni e lo stato dei dispositivi.

### 2.10.1 Attori del SNMP

- **Manager SNMP.**

Applicazione che comunica con i dispositivi della rete, per raccogliere informazioni. Può inviare richieste agli agenti SNMP.

- **Agente SNMP.**

È un software presente in ogni dispositivo della rete da monitorare. Raccoglie informazioni relativi al dispositivo (quali stato, configurazione, prestazioni, etc.) e risponde alle richieste del manager con queste informazioni.

- **MIB (Management Information Base).**

È un database, presente in ogni agente SNMP, che definisce le informazioni che possono essere raccolte o modificare tramite SNMP.

- **Protocollo SNMP.**

È il protocollo vero e proprio utilizzato nella comunicazione tra agenti e manager. È basato sul paradigma client-server. Il manager è il client, l'agente fa da server.

Usa il protocollo **UDP** e la **porta 161** degli agenti per mandarne messaggi, e la **porta 162** del manager per ricevere le **notifiche (traps)**.

Le **traps** sono generate successivamente a **malfunzionamenti, eventi e anomalie**.

### 2.10.2 Messaggi SNMP

#### Da manager a agente

- **GetRequest.**

Richiesta di recupero del valore di una variabile.

- **SetRequest.**

Richiesta di cambiamento del valore di una variabile.

- **GetNextRequest.**

Richiesta di recupero di possibili variabili e dei loro valori.

#### Da agente a manager

- **GetResponse.**

Restituisce il valore di una o più variabili.

- **Trap.**

Notifica in maniera asincrona e senza richiesta del manager situazioni anomale o eventi significativi della rete.

### 2.10.3 Versioni di SNMP

La prima versione di SNMP presentava anumerose criticità in termini di sicurezza e performance migliorabili. La seconda e la terza versione migliorano la sicurezza del protocollo, le performance. In particolare con la terza versione, viene introdotta la crittografia per i dati e un sistema di autenticazione.

#### SNMP in breve

**Simple Network Management Protocol** offre un servizio per la gestione. UDP, **porta 161 degli agenti, porta 162 del manager**. Architettura client-server: i server sono gli agenti. Ogni dispositivo agente ha una base di dati contenente oggetti per gestire i dispositivi di rete e i loro attributi. I dispositivi possono segnalare anomalie al manager tramite delle traps. È un protocollo **connectionless** con **comunicazioni asincrone**.

# Capitolo 3

## Transport Layer

### 3.1 Introduzione al Transport Layer

Il transport layer è situato tra l'application e il network layer. I protocolli di questo livello, permettono all'application layer di usufruire di una **connessione logica tra processi**, e quindi di una comunicazione **end-to-end**.

#### 3.1.1 Connessione logico, connessione reale

Introducendo il concetto di **protocollo**, abbiamo capito che questi forniscono delle specifiche di "incapsulamento" e delle specifiche di "interpretazione" dei dati. Inoltre, abbiamo capito che in termini esclusivamente **logici**, l' $i$ -esimo layer di un host  $A$ , comunica direttamente con l' $i$ -esimo layer dell'host  $B$ .  $A$  incapsula i dati con header opportuni,  $B$  interpreta in maniera opportuna seguendo le indicazioni del protocollo e degli header.

Ogni layer ha un ruolo specifico, in particolare:

- Il **livello di trasporto** crea una **connessione logica tra processi**.
- Il **livello di rete**, crea una **connessione logica tra host**.
- Il **livello di collegamento** (Data Link) si occupa della **comunicazione fisica** tra due host (fisicamente) collegati.

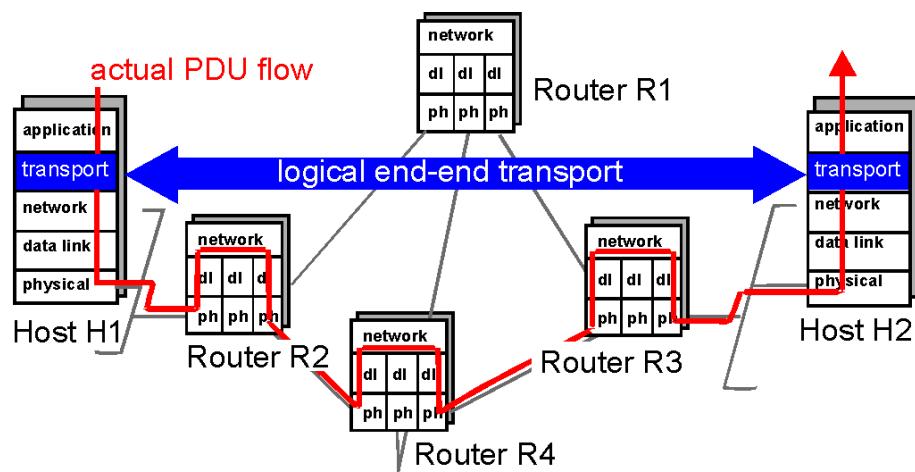


Immagine 3.1: Trasferimento logico vs effettivo

Nonostante la comunicazione **end-to-end logica** tra il transport layer degli host coinvolti esista, nella pratica la questione è ben più complessa: ogni protocollo di un dato livello, sfrutta infatti i protocolli dei livelli inferiori (come da disegno) per trasmettere con successo. Segue un esempio:

1. Un host desidera mandare una richiesta HTTP. Crea il messaggio HTTP secondo il protocollo dell'application layer.
2. Passa il messaggio HTTP al livello di trasporto, che lo incapsula in un segmento TCP.
3. Il livello di trasporto, passa il segmento al livello di rete, che lo incapsula in un pacchetto (o datagramma) IP.
4. Infine, il livello Data Link, si occupa di incapsulare in un frame il pacchetto fornito dal livello di rete.
5. Avviene la trasmissione.

Al sender, corrisponderà una sequenza di passaggi simili, ma dal basso verso l'alto, con un decapsulamento guidato dalle informazioni degli header, che permetteranno ad ogni layer di interpretare correttamente i dati ricevuti. Per ora, non approfondiamo nemmeno quanto l'inoltro da un router all'altro complichì la situazione!

## 3.2 Multiplexing e Demultiplexing

Sono le tecniche utilizzate dal livello di trasporto, per garantire che **il trasporto da host a host**, fornito dal livello di rete, diventi una comunicazione **da processo a processo**.

### 3.2.1 Multiplexing

Il **multiplexing avviene al mittente**, è il processo con cui un host incapsula i frammenti di dati provenienti da uno o più sockets in **segmenti del transport layer** con opportuni header, che verranno poi usati in fase di demultiplexing dal receiver.

### 3.2.2 Demultiplexing

Il **demultiplexing vviene al destinatario**. Consiste nella **consegnata dei segmenti al socket corretto**, e quindi al processo corretto, sfruttando le specifiche degli header, tra cui, chiaramente, la porta.

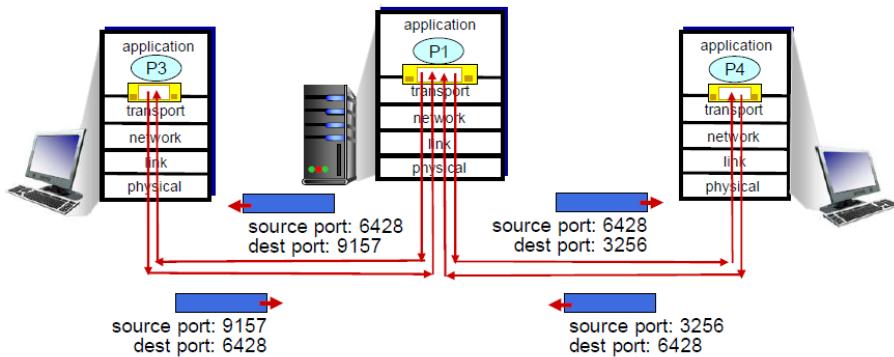


Immagine 3.2: Esempio di demultiplexing connectionless

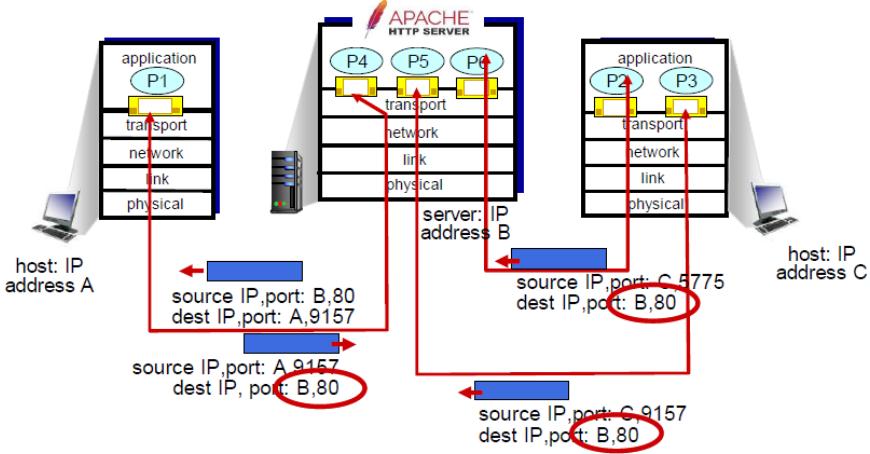


Immagine 3.3: Esempio di demultiplexing connection-oriented

### 3.3 UDP - User Datagram Protocol

TCP e UDP sono indubbiamente i **protocolli di trasporto dati** più popolari. Se il TCP è noto per la sua affidabilità (tramite i meccanismi di ACK da esso implementati), lo stesso non si può dire di UDP. Nonostante ciò, le funzionalità ridotte di UDP garantiscono un protocollo *bare bones*, ridotto all'osso, **molti più veloce** e con meno *overhead*.

#### 3.3.1 Caratteristiche di UDP

- È un protocollo di trasporto **bare-bones**, leggero, ma unreliable. Nessun handshake avviene tra il mittente e il destinatario. Ogni segmento UDP è gestito in maniera indipendente dagli altri, e non esistono vere e proprie "sessioni UDP". Per questo, è definito come un protocollo **connectionless**.
- L'header del protocollo UDP occupa 8 byte. 2 byte per la porta del mittente, 2 per la porta del destinatario, 2 per la lunghezza del messaggio (header UDP + dati) e 2 per il checksum. L'header del protocollo TCP occupa dai 20 ai 60 byte: ben di più rispetto a UDP!
- È usato da applicazioni di streaming multimediale, comunicazione VoIP e DNS. Si presta ad applicazioni che riescono a lavorare anche con **perdite di pacchetti**.

#### 3.3.2 Checksum (per UDP, ma anche TCP)

È un meccanismo che consente di **rilevare errori di trasmissione**. Questi errori spesso consistono in **bit-flip**, che possono stravolgere del tutto il significato dei dati trasmessi. Detto ciò, la verifica consiste nel calcolo di un **checksum**, in italiano **somma di controllo**, sia al mittente (che la inserirà nel frame) che al destinatario. La somma è calcolata in virtù dei dati contenuti nel frame. Se le due somme non dovessero coincidere, il pacchetto sarà da considerarsi fallito, e verrà scartato.

## 3.4 Reliable Data Transfer - Trasferimento affidabile

Data una rete di dispositivi comunicanti, quello che spesso si potrebbe pensare, è che la comunicazione sia sempre, a priori, affidabile. - *Ogni pacchetto inviato arriva al mittente!* - E invece no. **Il canale di comunicazione è da considerarsi inaffidabile**, per una serie di motivi legati alla natura fisica del canale, ma anche a causa della "consapevolezza" limitata degli host: due host non conoscono reciprocamente i loro stati, se non comunicando. E se dei pacchetti si perdono? Nonostante ciò, tramite software, è possibile garantire un **trasporto affidabile** che rende più **robusta** la comunicazione. *Creare un canale di comunicazione affidabile partendo da un canale fisico inaffidabile* è l'obiettivo posto dal topic **Reliable Data Transfer**.

### 3.4.1 RDT - Introduzione

Reliable Data Transfer è un meccanismo, descritto tramite **macchine a stati finiti**, che offre, nelle sue versioni, delle soluzioni software (per sender e receiver) per creare da canali fisici inaffidabili, connessioni affidabili. I meccanismi descritti da RDT, sono gli stessi seguiti da alcuni dei protocolli del livello di trasporto (banalmente, TCP).

	Errors on forward channel	Errors on backward channel	ACK / NAK	Packet loss
RDT 1.0	no	no	-	no
RDT 2.0	yes	no	ACK / NAK	no
RDT 2.1	yes	yes	ACK / NAK	no
RDT 2.2	yes	yes	ACK	no
RDT 3.0	yes	yes	ACK	yes

Immagine 3.4: Versioni di RDT

Osserviamo in breve le caratteristiche dei vari RDT prima di studiarne le FSM al dettaglio:

- 1.0. Non include nessun meccanismo effettivo, ma fa da base per le versioni successive.
- 2.0. Gestisce i pacchetti corrotti tramite segnali di ACK/NAK, ma non gestisce ACK e NAK corrotti.
- 2.1. Gestisce gli errori sul backward channel<sup>1</sup>.
- 2.2. Si elimina il simbolo dedicato NAK, utilizzando in maniera differente gli ACK.
- 3.0. Tutte le precedenti versioni gestiscono solo pacchetti corrotti. Con questa versione, si verifica anche la perdita dei pacchetti, introducendo il meccanismo di time-out.

### 3.4.2 RDT 1.0 - Canale perfetto, zero errori, zero perdite

Presuppone un **canale perfetto** senza errori di bit e perdita di pacchetti. Sarà la base per le successive versioni di RDT. Gli automi rimangono in attesa di un certo evento. La verifica dell'evento avviene tramite il cosiddetto "polling" o interrogazione.

Il **polling rate** stabilisce ogni quanto interrogare la condizione. Il polling rate DEVE essere stabilito, in quanto, altrimenti la verifica avverrebbe un numero di volte pari alla frequenza della CPU, e si avrebbero comportamenti anomali. La FSM del sender si attiva quando riceve una richiesta per mandare dei dati.



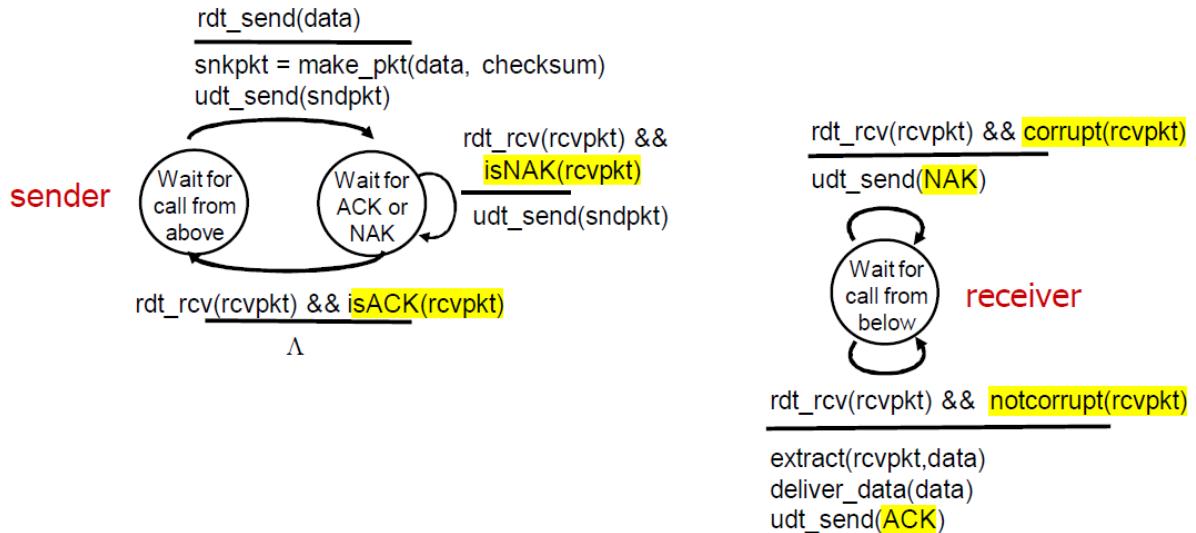
Quella del receiver, quando riceve dei pacchetti. `deliver_data(data)` si riferisce a socket.

<sup>1</sup>Il backward channel è canale per l'invio dei segnali di ACK/NAK durante una connessione.

### 3.4.3 RDT 2.0 - Canale imperfetto, errori sul forward

Introduce un **checksum** e un **meccanismo di ACK** (senza checksum). Il sender manda un pacchetto e aspetta un feedback.

Si ipotizza che non si perdano pacchetti e che il backward sia perfetto.



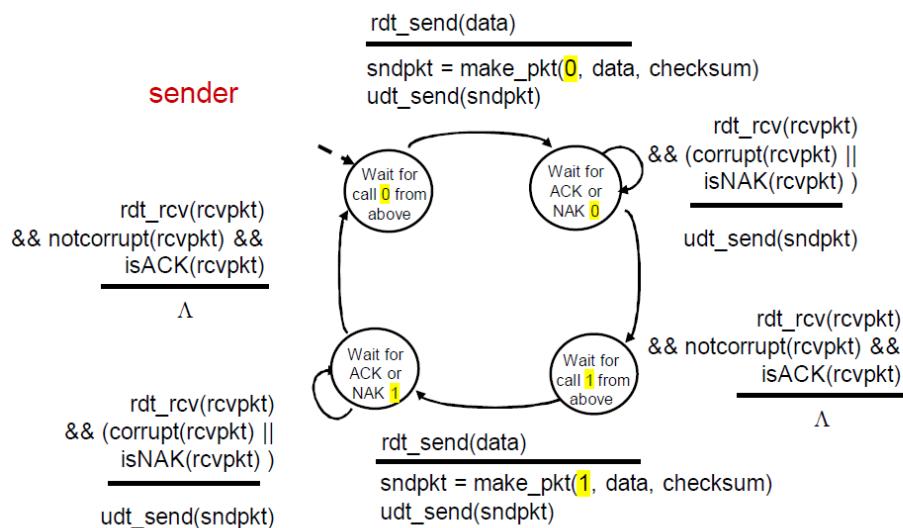
- Il sender che ha appena mandato il pacchetto attende un ACK o un NAK.
- Il receiver verifica se il pacchetto è corrotto o meno, e in caso, manda un NAK o un ACK.
- Se il sender riceve ACK, ritorna allo stato iniziale ( $\Lambda$ ), altrimenti rimanda il pacchetto arrivato corrotto.

Ipotizzare che il backward sia perfetto, è che ACK e NAK arrivino sempre integri, è azzardato: un ACK potrebbe corrompersi, diventare un NAK o non arrivare mai. Un hacker potrebbe sfruttare questa debolezza sostituendo tutti i pacchetti ACK con un NAK o un pacchetto differente, secondo l'approccio **Man-In-The-Middle**.

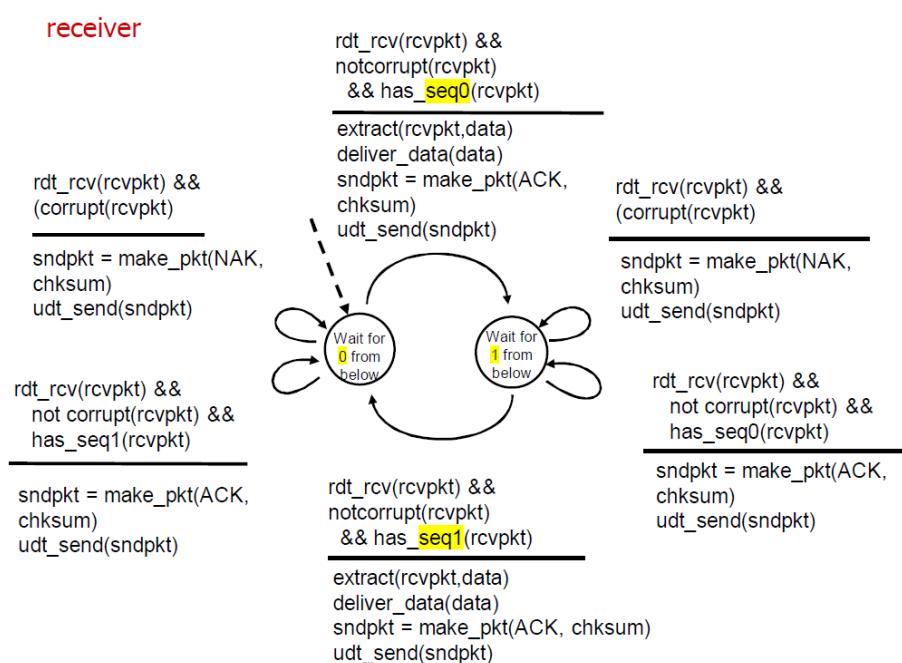
#### 3.4.4 RDT 2.1 - Errori sul forward e sul backward

L'RDT 2.1 nasce dall'esigenza di gestire un backward channel imperfetto, integrando un **checksum sugli ACK/NAK**. A ogni pacchetto viene annesso un **bit sequenza**, per disambiguare i pacchetti e gli ACK. A ogni pacchetto mandato, il sender integra un bit che si alterna tra 0 e 1. Quando il sender riceve un ACK/NAK corrotto, rimanda il pacchetto. Se il receiver riceve due volte lo stesso pacchetto con lo stesso bit sequenza, sa di dover **rimandare l'ACK**.

## Sender



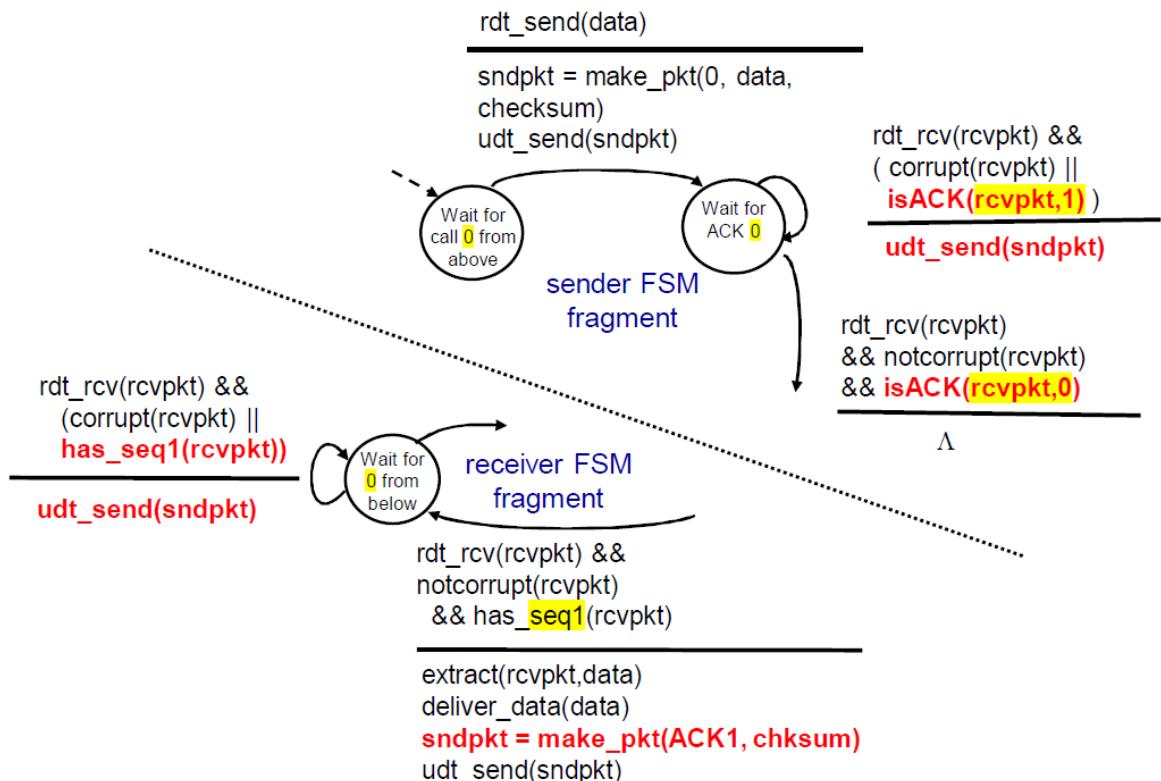
## Receiver



### 3.4.5 RDT 2.2 - Eliminiamo il NAK!

Rimuove il segnale NAK, cambiando leggermente il funzionamento dell'ACK:

- Se il pacchetto è stato consegnato correttamente, viene mandato il segnale di ACK relativo al pacchetto.
- Se il pacchetto non è stato consegnato correttamente, viene mandato il segnale di ACK dell'ultimo pacchetto ricevuto correttamente.

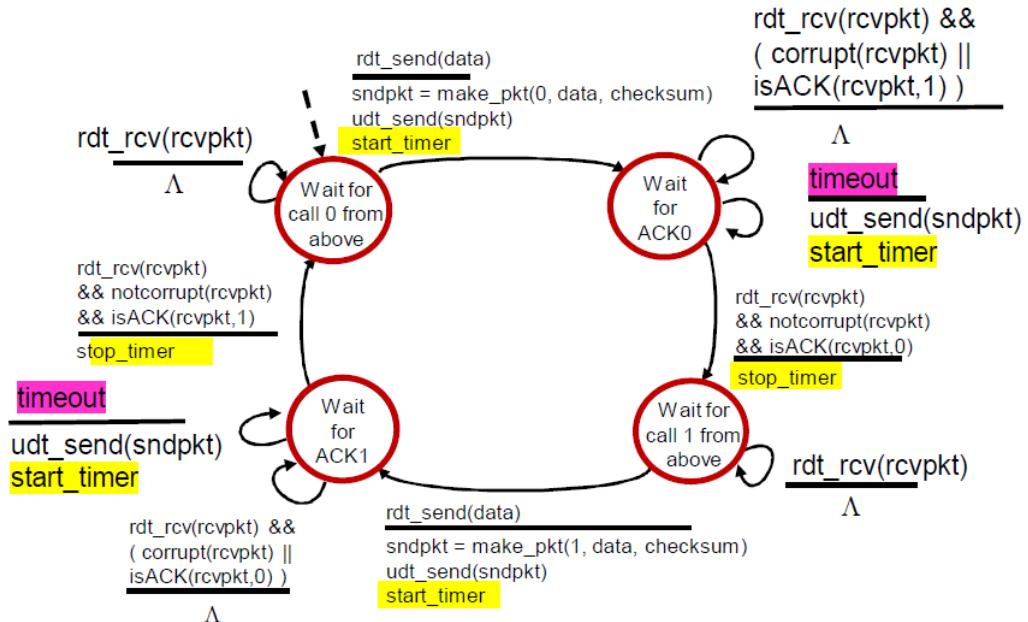


Nelle versioni precedenti, il NAK e l'ACK sono due segnali differenti. Un segnale che non è né ACK né NAK, a causa di una distorsione del segnale o un bit-flip, porta a dei segnali che il sistema non sa come gestire. **Rimuovere la classe dei segnali NAK ci permette di non cadere in queste ambiguità.** Tuttavia, non contempliamo ancora la perdita di pacchetti.

### 3.4.6 RDT 3.0 - Errori e perdite

Assumiamo (finalmente) che il canale possa perdere pacchetti (sia dati, che ACKs). Gestiamo il tutto con la **temporizzazione**.

#### Sender



#### Receiver

Macchina a stati finiti analoga a quella di RDT 2.2. La temporizzazione è gestita esclusivamente al sender.

## 3.5 Calcolo del Throughput

Il **throughput** è un'indice della capacità di trasmissione di un canale di comunicazione.

$$\text{Throughput} = \frac{\text{Numero di bit (o byte) trasmessi con successo}}{\text{Tempo impiegato}}$$

### 3.5.1 Throughput teorico

Consideriamo un canale con **Bandwidth**  $BW = 10 \text{ Mbps}$  (Megabit per secondo). Questo vuol dire in un secondo possono essere trasmessi  $10^7$  bit. Un bit viene trasmesso in  $\frac{1}{10^7} = 0.1\mu\text{s}$  (microsecondo).

#### Tempo di trasmissione di un frame

Consideriamo ora di voler trasmettere un frame di 1500 byte, ovvero  $1500 \cdot 8 = 12000$  bit. Per trasmettere un frame, impieghiamo il seguente tempo:

$$T_{\text{frame}} = \frac{\text{Dimensione del frame (in bit)}}{\text{Bandwidth (in bit/s)}} = \frac{12000 \text{ bit}}{10^7 \text{ bit/s}} = 0.0012s = 1200\mu\text{s}$$

#### Tempo di propagazione

Nella trasmissione dobbiamo anche considerare il tempo di propagazione, che è il tempo richiesto al segnale per arrivare fisicamente da host *A* a *B*: supponiamo che lo spazio di trasmissione sia pari ad  $1\text{km}$ . Supponiamo che la velocità di trasmissione, che dipende dal mezzo che usiamo e dalle sue condizioni, sia  $v \approx 200.000\text{km/s}$ , e che sia un valore costante. Il tempo di propagazione è dato dal rapporto spazio/velocità.

$$T_{\text{propagazione}} = \frac{\text{Spazio di trasmissione}}{\text{Velocità di trasmissione}} = \frac{1\text{km}}{200000\text{km/s}} = \frac{1}{200000}s = 5 \cdot 10^{-6}s = 5\mu\text{s}$$

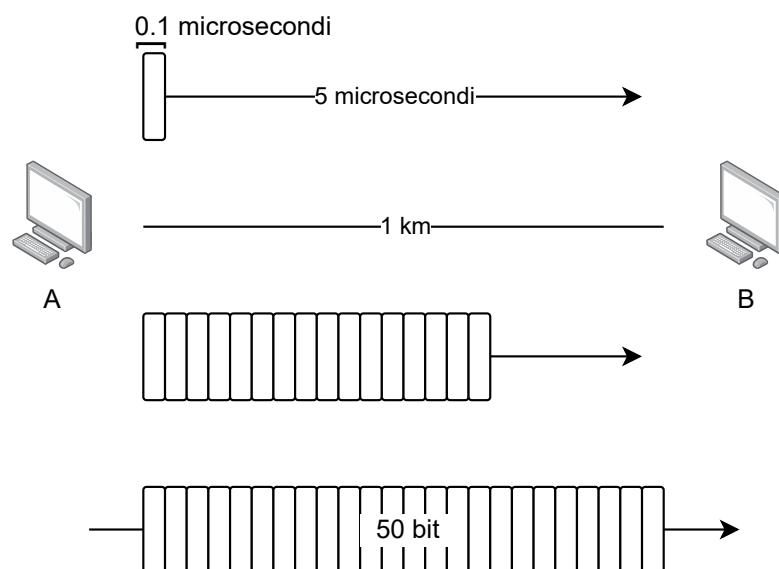
Con questa informazione possiamo anche andare a calcolare il numero di bit che possono occupare il canale contemporaneamente, ovvero

$$\text{Bandwidth (in bit/s)} \cdot T_{\text{propagazione}} (\text{in s}) = 10^7 \text{ bit/s} \cdot 5 \cdot 10^{-6} \text{ s} = 50 \text{ bit}$$

#### Tempo di trasferimento

Ora abbiamo tutte le informazioni necessarie per calcolare il tempo di trasferimento complessivo. Esso indica il tempo che il frame necessita per essere trasferito (ipotizzando che non ci siano da considerare variabili relative ai tempi di connessione, ACK, etc.).

$$T_{\text{trasferimento}} = T_{\text{frame}} + T_{\text{propagazione}} = 1200\mu\text{s} + 5\mu\text{s} = 1205\mu\text{s}$$



### 3.5.2 Throughput effettivo

Quello appena calcolato, è da definirsi un **throughput teorico**. Il **throughput effettivo** è influenzato anche da altri elementi di attesa, quali *Round Trip Time*, tempi di elaborazione e di trasferimento relativi agli **ACK**.

#### RTT - Round Trip Time

Il Round Trip Time può essere stimato, e dedicheremo una sezione proprio alle metodologie di stima. È definito come il tempo che intercorre tra l'inizio dell'invio di un messaggio, e la ricezione della sua conferma.

#### Tempo di trasmissione ACK

Il segnale di ACK ha solitamente dimensione pari a  $64 \text{ byte} = 64 \cdot 8 \text{ bit} = 512 \text{ bit}$ . Ne consegue che:

$$T_{ACK} = \frac{\text{Dimensione dell'ACK (in bit)}}{\text{Bandwidth (in bit/s)}} = \frac{512}{10^7} = 0.0000512s = 51.2\mu s$$

#### Tempo di trasferimento effettivo

Stimiamo un tempo di elaborazione dell'ACK di  $3\mu s$  e un RTT di  $10\mu s$  ( $5\mu s \cdot 2$ , ovvero il  $T_{propagazione}$ , ma in andata e ritorno). Otteniamo quindi un tempo complessivo pari a:

$$T_{\text{effettivo}} = T_{frame} + T_{ACK} + RTT + \text{Elaborazione ACK} = 1200 + 51.2 + 10 + 3 = 1264.2\mu s$$

#### Calcolo del Throughput effettivo

Calcoliamo ora il throughput effettivo, mettendo a rapporto il numero di bit per frame da trasmettere con il tempo di trasferimento totale

$$\frac{\text{Dimensione del frame (in bit)}}{\text{Tempo totale di trasmissione (in s)}} = \frac{12000}{1264.2 \cdot 10^{-6}} = 9.492 \text{ Mbps}$$

Osserviamo con piacere che otteniamo un valore molto simile alla larghezza di banda. I fattori esterni non hanno creato particolari **colli di bottiglia** alla nostra capacità di trasmissione! Il mezzo è ben sfruttato.

#### Conclusioni sul throughput reale

Il throughput reale è fortemente influenzato dal tempo di elaborazione dei segnali di ACK e dall'RTT. Nel caso appena studiato, la **bandwidth è ben sfruttata**, ma quando l'RTT sarà dell'ordine dei millisecondi<sup>2</sup>, e non dei microsecondi, otterremo dei cali di performance non da poco.

### 3.5.3 Considerazioni

Il **tempo di propagazione** è il tempo richiesto ad un segnale per trasmettersi da un punto all'altro. Precedentemente, abbiamo stimato l'*RTT* come due volte il  $T_{propagazione}$ . Nel protocollo TCP è presente il meccanismo di Three Way Handshake: questo implica ulteriore ritardo per l'RTT, che può di gran lunga superare la stima calcolata. Se i valori sono diversi, nel calcolo, andremo a considerare il maggiore tra i due.

Inoltre, se la bandwidth effettiva calcolata è superiore a quella teorica, allora la bandwidth teorica funge da collo di bottiglia: la bandwidth effettiva sarà uguale a quella teorica, in quanto limite massimo.

### 3.5.4 Ping

Il comando **ping** è un utility che permette di calcolare l'RTT necessario per la connessione ad un server, e fornisce anche la percentuale di pacchetti persi. Usa il protocollo ICMP.

<sup>2</sup>Recap sui prefissi: milli- =  $10^{-3}$ , micro- =  $10^{-6}$ , nano- =  $10^{-9}$ , kilo- =  $10^3$ , mega- =  $10^6$ , giga- =  $10^9$ ;

## 3.6 Modalità Pipeline

È una modalità che consente di aumentare il throughput di una rete. Invece di inviare un singolo frame e aspettare il rispettivo ACK (modalità esposta in precedenza, nota come modalità **stop-and-wait**), si stabilisce una window di  $n$  frame di cui si **potranno aspettare gli ACK contemporaneamente**. Questo **riduce di molto l'influenza del RTT sul throughput**, riducendo di molto l'*overhead* delle singole trasmissioni. L'utilizzo della modalità pipeline è chiaramente più **vantaggiosa della stop-and-wait**, soprattutto nel contesto di **reti ad alta latenza**. La finestra d'invio verrà successivamente indicata con il nome **cwnd**, nel contesto del controllo della congestione.

### Esempio

Vogliamo mandare pacchetti di 1000 byte.

$$BW = 1 Gbps = 1000 Mbps = 1 \cdot 10^9 bps.$$

$$RTT = 30 ms.$$

Il tempo teoricamente richiesto sarebbe

$$T_{frame} = \frac{\text{Dimensione dati (in bit)}}{\text{Bandwidth (in bit/s)}} = \frac{8 \cdot 10^3 \text{ bit}}{1 \cdot 10^9 \text{ bit/s}} = 8 \cdot 10^{-6} \text{ s} = 8 \mu\text{s}$$

a cui aggiungere  $30000 \mu\text{s}$ . Otteniamo un throughput effettivo di

$$\frac{\text{Dimensione del frame (in bit)}}{\text{Tempo totale di trasmissione (in s)}} = \frac{8000 \text{ bit}}{30000 \mu\text{s}} = \frac{8000 \text{ bit}}{30,008 ms} = \frac{8000 \text{ bit}}{0,030008 s} = 266595 bps \approx 267 Kbps$$

Creando un grande collo di bottiglia alla nostra bandwidth di  $1 Gbps$ . Tuttavia, il canale potrebbe ospitare contemporaneamente un numero di bit pari a

$$\text{Bandwidth} \cdot RTT = 1 \cdot 10^9 \cdot 3 \cdot 10^{-2} = 3 \cdot 10^7 \text{ bit}$$

Tuttavia, se decidiamo di sfruttare la modalità pipeline, potremmo pensare di mandare più frame contemporaneamente. Il canale, infatti, supporterebbe al più:

$$\frac{30\,000\,000 \text{ bit}}{8 \cdot 10^3 \text{ bit/frame}} = 3750 \text{ frame}$$

Stabilendo una finestra di  $n$  frame, con un valore di  $n$  opportuno, sarà possibile sfruttare al meglio la bandwidth del canale! Idealmente, in questo caso,  $n = 3750$ .

### 3.6.1 Contro della modalità pipeline

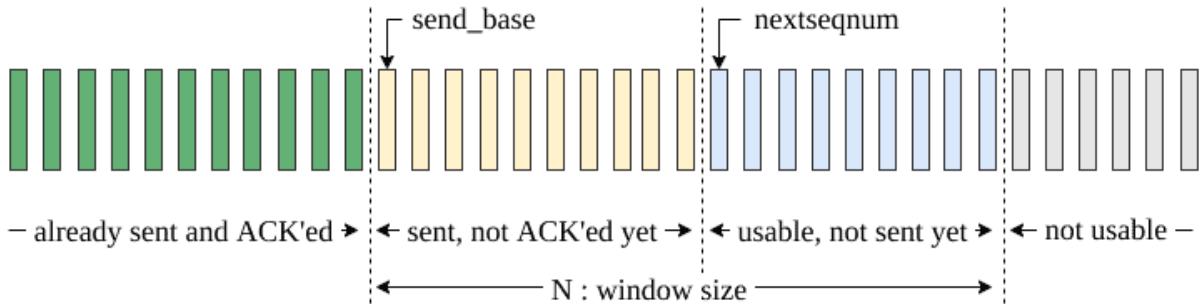
Questa modalità richiede una gestione più complessa della ricezione degli ACK:

- Bisogna gestire in maniera opportuna invio e ricezione e grandezza della finestra di frame.
- Diventa obbligatorio usare più di un bit per il numero di sequenza, in modo proporzionale alla dimensione della finestra.

### 3.6.2 Algoritmi per la pipeline - Go-Back-n

Usa un approccio **sliding windows** per gestire la pipeline. Il server stabilisce una window di  $n$  frame.

#### Sender



Ogni volta che il primo frame della finestra viene mandato, questa "scorre" di un frame verso quelli ancora da mandare. Individuiamo quattro insiemi disgiunti di frame:

- I frame mandati non confermati (ACKed, prima parte della finestra).
- I frame non mandati ma utilizzabili (seconda parte della finestra).
- Prima della finestra, abbiamo i frame mandati e confermati.
- Dopo la finestra, i frame non utilizzabili.

#### Receiver

Riceve i pacchetti e risponde con gli ACK rispettivi. Quando non riceve il pacchetto giusto<sup>3</sup>, scarta il pacchetto, manda un duplicato dell'ultimo ACK ricevuto correttamente.

#### Receiver view of sequence number space:



#### Criticità del Go-Back-N

Osserviamo che se l' $n$ -esimo pacchetto non è stato ricevuto con successo, tutti i successivi pacchetti mandati verranno scartati. Nell'immagine della prossima pagina, si evidenzia come il primo inoltro dei pacchetti 3, 4 e 5 sia praticamente cestinato. È un algoritmo solido e reliable, ma rischia grosse perdite di tempo a causa dell'ACK cumulativo.

---

<sup>3</sup>Un esempio di pacchetto non corretto? Il pacchetto con numero di sequenza 3 dopo aver ricevuto il pacchetto 1 è sbagliato: manca il pacchetto 2.

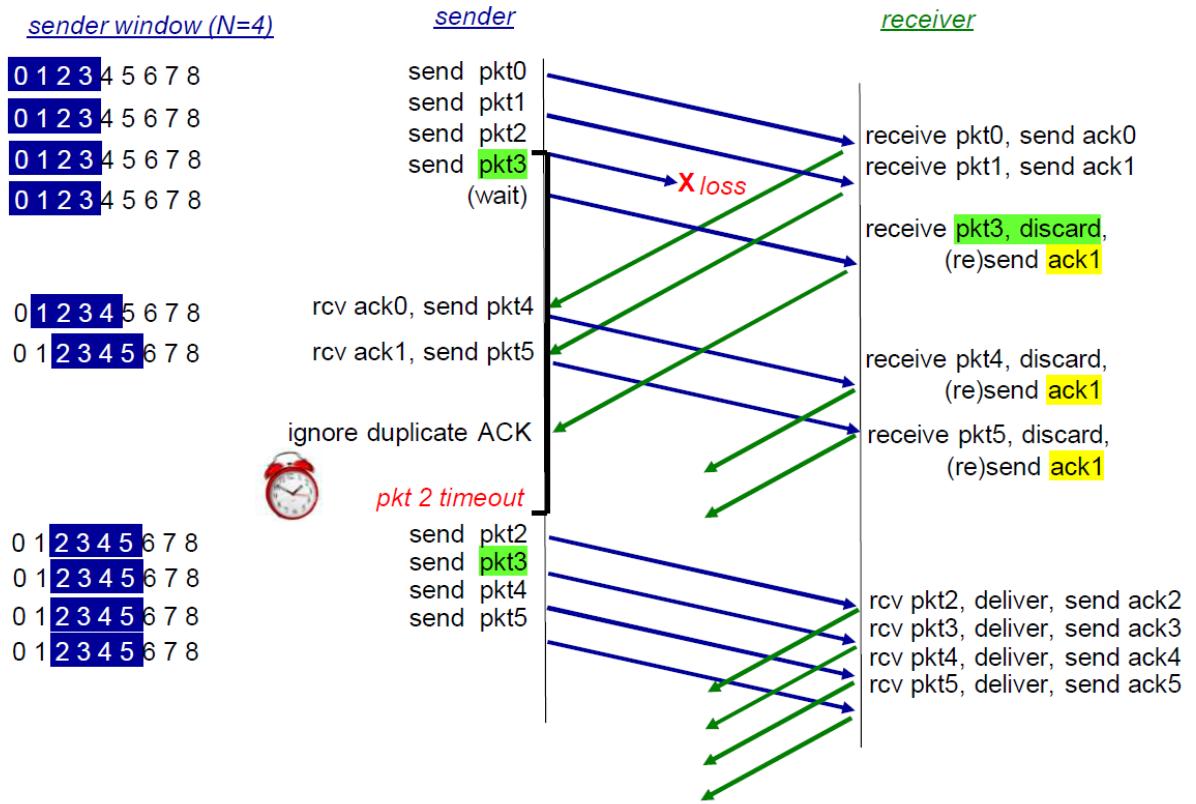
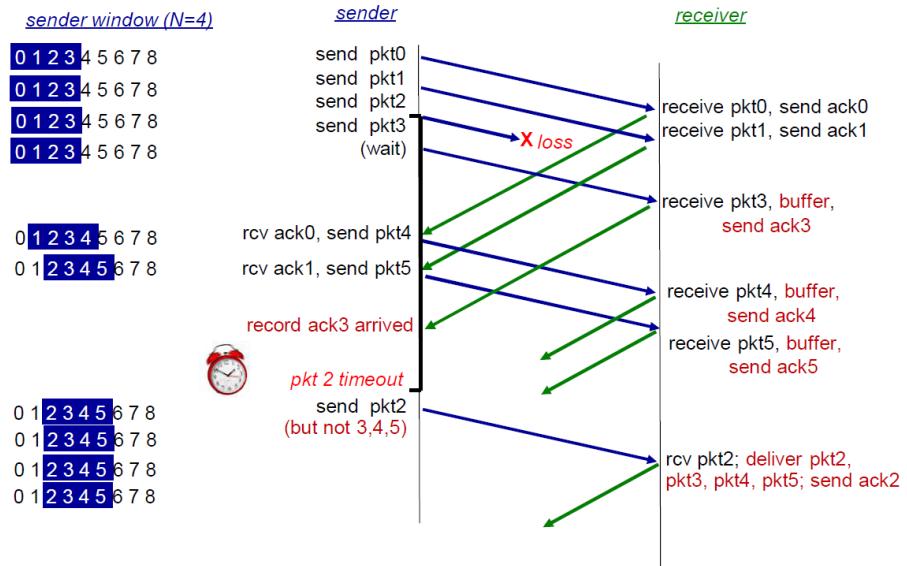


Immagine 3.5: Go-back-n.

### 3.6.3 Algoritmi per la pipeline - Selective Repeat

Col selective repeat, i frame che si perderebbero durante il time-out, sono bufferizzati in una memoria all'interno del dispositivo (in una look-up-table), in modo da velocizzarne il caricamento e non dover richiedere il reinvio dal sender. Gli ACK dei pacchetti ricevuti correttamente vengono rimandati regolarmente.



#### Sender

- Trasmette fino a  $n$  pacchetti **senza attendere ACK**.
- Ogni pacchetto ha un **timer personale** per la ritrasmissione.
- Ricevuto l'ACK dell' $n$ -esimo pacchetto, viene **marcato come ricevuto**.
- Il primo frame della windows è sempre il primo frame della sequenza non ancora confermato.

#### Receiver

- I pacchetti non ricevuti in ordine vengono inseriti in un **buffer**.
- A ogni pacchetto ricevuto, consegue l'invio di un **ACK**.
- Se l' $n$ -esimo pacchetto viene ricevuto, e ci sono **pacchetti subito successivi bufferizzati**, questi vengono **passati al layer superiore**.
- Il receiver potrebbe ricevere un pacchetto con **numero di sequenza precedente alla finestra**: in tal caso, **reinvierà l'ACK**.

#### Pro del Selective Repeat

**Meno ritrasmissioni inutili**, maggiore efficienza in canali con alta perdita di pacchetti.

#### Contro del Selective Repeat

- Il buffer non ha **memoria infinita**, potrebbe saturarsi.
- Possono esserci **ambiguità** causati dai **codici sequenza ripetuti**. Per non incorrere in questi problemi, deve essere rispettata la seguente disequazione:

$$K\text{-seq} (\text{Numero max sequenza}) \geq 2 \cdot \text{Window Size}$$

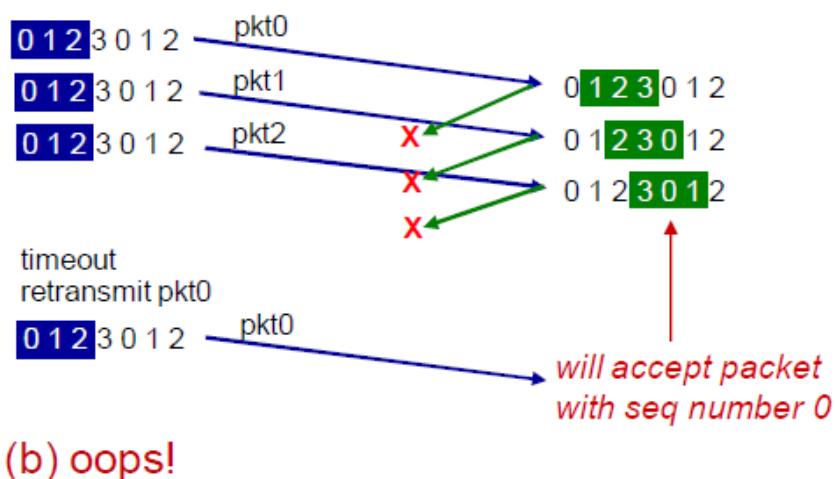
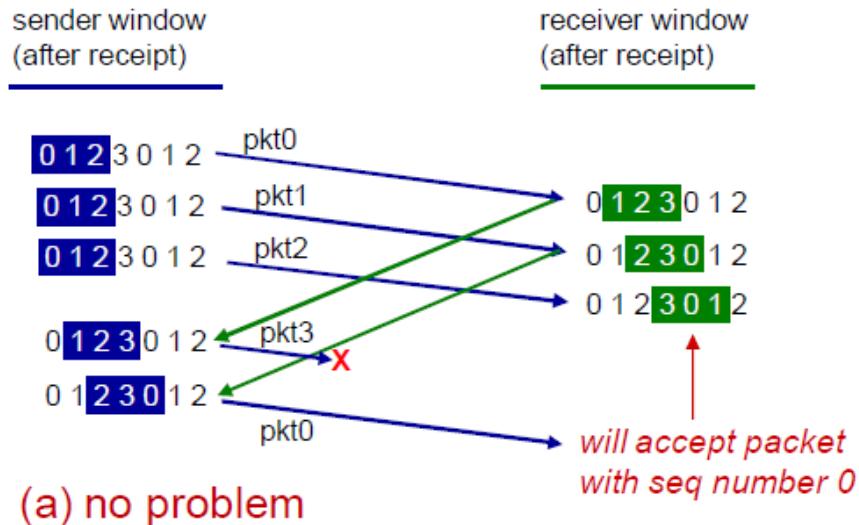


Immagine 3.6: Ambiguità dei codici sequenza nel Selective Repeat

### 3.7 TCP

Sta per **Transmission Control Protocol**. Assieme a UDP, è il **protocollo del layer di trasporto più importante**. Sfruttato da tantissime applicazioni del layer superiore, offre un trasporto affidabile, basato sui meccanismi della modello RDT 3.0. Tra le sue funzionalità, offre:

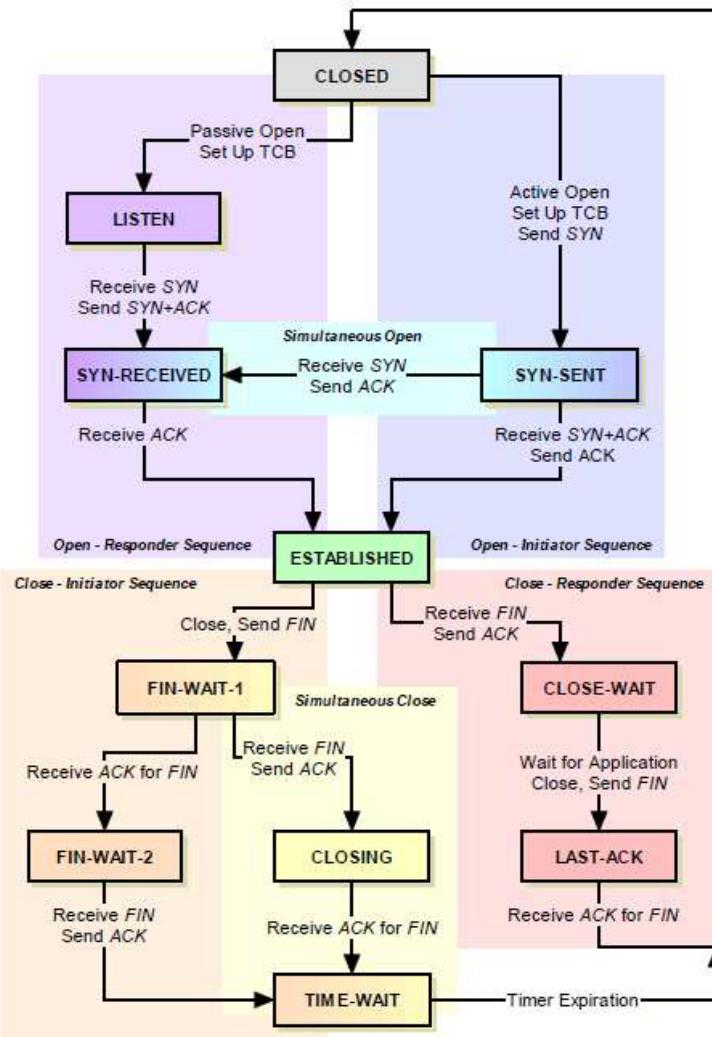
- **Addressing / multiplexing.**
- Stabilimento, gestione e terminazione della **connessione**.
- **Data Handling** e impacchettamento.
- **Trasferimento dei dati** (banalmente).
- Fornisce **specifiche** relative alla reliability della rete.
- **Controllo del flusso** e gestione della **congestione**.

**Cosa non fornisce il TCP?**

- Non garantisce **sicurezza**.  
È **vulnerabile** ad hacking, attacchi man-in-the-middle, sniffing e simili.
- Non da **specifiche sull'application layer**.  
Non fa ipotesi sui layer superiori (ed è giusto che sia così! Altrimenti le applicazioni si dovrebbero adattare alle ipotesi fatte da TCP).
- Non mantiene i **message boundaries**.  
Non stabilisce i confini dei messaggi. Sono i protocolli del layer applicativo che stabiliscono come interpretare i messaggi e ricostruire i dati, tramite caratteri di terminazione, lunghezze prestabilite o header.
- Non da **garanzie sul throughput**.
- **Non supporta trasferimento multicast**, ossia trasferimento da un mittente a molti destinatari in una sola operazione.

### 3.7.1 TCP-State Model

È un diagramma degli stati che descrive le fasi di una connessione TCP.



### 3.7.2 Fasi di apertura della connessione (3-way-handshake)

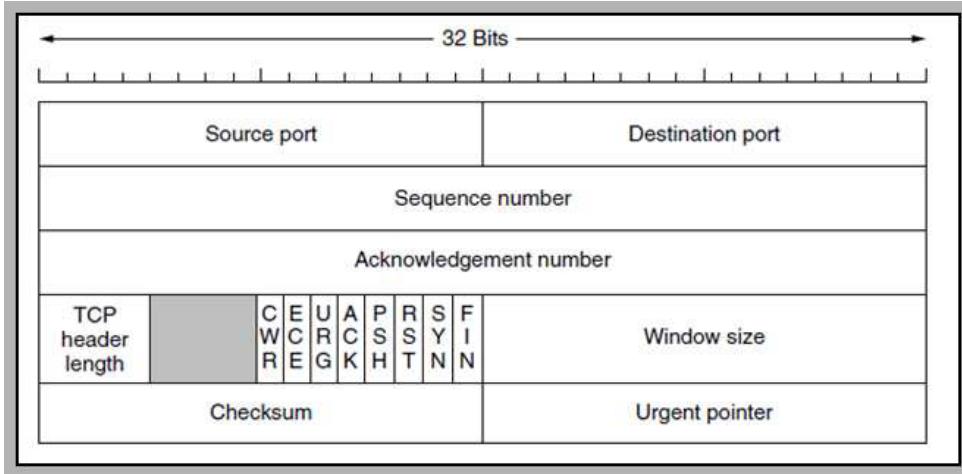
1. **CLOSED**.
  - (a) Stato iniziale, nessuna connessione attiva.
  - (b) Un'applicazione può avviare una connessione o attenderne una in ingresso.
2. **LISTEN** (solo lato server).
  - (a) Il server è in ascolto, pronto a ricevere connessioni (passive open).
  - (b) Se riceve un segmento SYN, risponde con SYN-ACK e passa allo stato SYN
3. **SYN-SENT** (solo lato client).
  - (a) Il client avvia una connessione inviando un segmento SYN
  - (b) Se riceve SYN+ACK risponde con ACK e la connessione è stabilita (passando a ESTABLISHED)
4. **SYN-RECEIVED**
  - (a) Il server ha ricevuto SYN e inviato il SYN-ACK
  - (b) Quando riceve l'ACK finale dal client, la connessione è stabilita (ESTABLISHED).
5. **ESTABLISHED**
  - (a) Stato operativo della connessione in cui i dati vengono trasmessi tra client e server.
  - (b) La connessione si trova in questo stato finché non viene chiusa.

### 3.7.3 Fasi della chiusura della connessione (4-way-handshake)

6. FIN-WAIT-1.
  - (a) Il lato che inizia la chiusura invia un segmento FIN.
  - (b) Attende un ACK per il FIN o un FIN dall'altro lato.
7. CLOSE-WAIT.
  - (a) Il lato che riceve il FIN invia un ACK e passa in CLOSE-WAIT.
  - (b) Aspetta che l'applicazione chiuda il socket, poi invia un FIN.
8. FIN-WAIT-2
9. LAST-ACK.
  - (a) Il lato che ha ricevuto il FIN deve terminare, invia un FIN e attende un ACK.
10. TIME-WAIT
  - (a) Dopo aver ricevuto l'ultimo ACK, il lato iniziatore della chiusura entra in TIME-WAIT per un certo tempo (solitamente il doppio del Maximum Segment Timeline).  
Questo assicura che l'altro lato abbia ricevuto l'ACK finale prima di tornare a CLOSED.
11. CLOSING
  - (a) Caso raro in cui entrambi i lati inviano un FIN contemporaneamente.
  - (b) Entrambi inviano e ricevono un ACK, poi passano a TIME-WAIT.

### 3.7.4 Struttura dei segmenti del TCP

Gli header dei segmenti TCP occupano dai **20 ai 60 byte**, (ricordiamo che gli header UDP occupano solo 8 byte).



- **Source port** e **Destination port** indicano le porte dei processi del sender e receiver associati alla connessione TCP. 16 bit ciascuno.
- **Sequence number.**  
Viene incrementato di  $n$  ogni volta che  $n$  byte sono trasmessi. Mandato un pacchetto TCP di 1400 byte, il prossimo valore sarà il precedente numero di sequenza + 1400. 32 bit.
- **Acknowledgment number.**  
Se ACK è abilitato, ritorna il sequence number del prossimo pacchetto che si aspetta di ricevere. 32 bit.

CLIENT		SERVER
#seq = 1		
Len = 669	-->	<-- ACK (669 + 1 =) 670
#seq = 670	-->	<-- ACK (670 + 1460 =) 2130
Len = 1460	-->	<-- ACK 3590
#seq = 2130	-->	[...]
Len = 1460	-->	
		END OF TCP CONNECTION

Notiamo che **il protocollo TCP è full-duplex**: gli host ricevono e mandano dati nella stessa connessione.

- **TCP header length o Data Offset.**  
Specifica il numero di double word (4 byte) dell'header del segmento TCP. 4 bit.
- Dello spazio libero per sviluppi futuri del protocollo. 4 bits.
- **8 flags**, un bit per flag.
  - **CWR** Congestion Window Reduced.  
A 1 indica che l'host sorgente ha ricevuto un segmento TCP con il flag ECE impostato a 1.
  - **ECE** Explicit Congestion Notification-Echo.  
A 1 indica che l'host supporta ECN durante il 3-way handshake
  - **URG.**  
A 1 indica che nel flusso sono presenti dati urgenti specificati nel campo urgent pointer.
  - **ACK.**  
A 1 indica che il campo Acknowledgment number è valido.

- **PSH.**  
A 1 indica che i dati in arrivo non devono essere bufferizzati ma passati subito ai livelli superiori dell'applicazione.
- **RST.**  
A 1 indica che la connessione non è valida; viene utilizzato in caso di grave errore; a volte utilizzato insieme al flag ACK per la chiusura di una connessione.
- **SYN.**  
A 1 indica che il mittente del segmento vuole aprire una connessione TCP con l'host destinatario e specifica nel campo Sequence number il valore dell'Initial Sequence Number (ISN); questo sincronizza i numeri di sequenza dei due host. L'host che ha inviato il SYN deve attendere dall'host remoto un pacchetto SYN/ACK.
- **FIN.**  
A 1 indica che l'host mittente del segmento vuole chiudere la connessione TCP. Il mittente attende la conferma dal ricevente (con un FIN-ACK). La connessione sarà ancora chiusa per metà: l'host che ha inviato FIN non potrà più inviare dati, mentre l'altro host ha il canale di comunicazione ancora disponibile. L'altro host dovrà inviare il pacchetto con FIN impostato. Dopo il relativo FIN-ACK, la connessione sarà considerata completamente chiusa.

- **Window size.**

Indica la dimensione della finestra di ricezione del mittente. 16 bit.

- **Checksum.**

Serve a verificare la validità del segmento. 16 bit.

- **Urgent pointer.**

Punta a un dato urgente, ma solo se **URG =1**. Indica l'offset in byte dal sequence number ai dati urgenti all'interno del flusso. 16 bit.

- Sotto troveremo anche dei byte dedicati ad ulteriore possibili **impostazioni**, e, chiaramente, lo spazio dedicato ai **dati da inviare**.

## 3.8 Migliorare il Throughput - MSS e MTU

I meccanismi appena esposti rendono TCP un protocollo affidabile. Tuttavia, anche il **throughput** è un fattore da tenere in considerazione. Per massimizzare il throughput, dobbiamo analizzare una serie di parametri.

### 3.8.1 MSS, MTU

- **MSS - Maximum Segment Size.**

Indica la dimensione massima dei dati che possono essere inviati in un singolo segmento TCP (esclusi gli header TCP/IP). Questo valore viene solitamente scelto in virtù dal frame più grande che il sender può trasmettere.

- **MTU - Maximum Transmission Unit.**

$$\text{MSS} + \text{dimensione dell'header TCP} + \text{dimensione dell'header IP}$$

Il protocollo TCP richiede un MTU minimo di 576 bytes, e quindi un MSS minimo di 536 bytes + 40 bytes di header IP (20 byte) e TCP (20 bytes).

## 3.9 Migliorare il Throughput - Retransmission Time-Out

Affrontando l'argomento del **Reliable Data Transfer**, abbiamo spiegato che il meccanismo di **temporizzazione è fondamentale** per ottenere un trasferimento dati **reliable**. Nonostante ciò, bisogna trovare un buon **compromesso** per non impattare negativamente sul throughput. **Il time-out va stabilito in funzione del Round Trip Time.**

- **Time-out troppo lungo?**

Reazione troppo lenta alla perdita dei pacchetti. Conseguenza di un time-out di molto maggiore dell'RTT.

- **Time-out troppo corto?**

Gli ACK potrebbero non arrivare in tempo, richiedendo ritrasmissioni non necessarie. Time-out quanto o poco più lungo dell'RTT.

Per decidere il tempo opportuno per il time-out, bisogna avere una stima <sup>4</sup> valida dell'RTT.

### 3.9.1 Stimare il Round Trip Time

Il Sample\_RTT è la misura del tempo di trasmissione di un segmento. La misura avviene dall'invio del segmento fino alla ricezione dell'ACK.

Un algoritmo che potrebbe essere utilizzato per stimare il time-out è quello di **Karn**: questo tuttavia, non tiene conto di possibili ritrasmissioni dei pacchetti non-acknowledged, che nella realtà avvengono molto spesso.

Utilizzeremo quindi una **media pesata** chiamata **Exponential Weighted Moving Average**.

---

<sup>4</sup>Il round trip time è un valore misurabile, ma variabile. L'obiettivo che ci poniamo è stimare sempre il prossimo valore dell'RTT in funzione dei precedenti. Ricordiamo inoltre che ping permette di ottenere il Round Trip Time con un server.

### 3.9.2 Exponential Weighted Moving Average - EWMA

Il peso, ovvero l'influenza dei campioni passati sulla media calcolata decresce in maniera esponenziale.

$$\text{Estimated\_RTT}_n = (1 - \alpha) \cdot \text{Estimated\_RTT}_{n-1} + \alpha \cdot \text{Sample\_RTT}_n$$

Dove il  $\text{Sample\_RTT}_n$  è l' $n$ -esima misurazione. Il valore  $\alpha$  è detto **weighted value**, e stabilisce il peso dell'ultima misura rispetto a quello dell'ultima stima sulla media. Un valore tipico è  $\alpha = \frac{1}{8} = 0,125$ . Il valore di  $\alpha$  varia, non è raro l'utilizzo di tecnica try and error per trovare il valore che permette di ottenere stime ottime. Euristiche varie e tecnologie AI sono utilizzate per ottenere stime via via più accurate.

$$\text{E\_RTT}_{n-2} = (1-\alpha) \text{E\_RTT}_{n-3} + \alpha \text{S\_RTT}_{n-2}$$

$$\text{E\_RTT}_{n-1} = (1-\alpha) \text{E\_RTT}_{n-2} + \alpha \text{S\_RTT}_{n-1}$$

$$\text{E\_RTT}_n = (1-\alpha) \text{E\_RTT}_{n-1} + \alpha \text{S\_RTT}_n \quad \alpha = 0.125 \quad 1-\alpha = 0.875$$

$$\text{E\_RTT}_n = (1-\alpha)((1-\alpha) \text{E\_RTT}_{n-2} + \alpha \text{S\_RTT}_{n-1}) + \alpha \text{S\_RTT}_n$$

$$\text{E\_RTT}_n = (1-\alpha)^2 \text{E\_RTT}_{n-2} + \alpha(1-\alpha) \text{S\_RTT}_{n-1} + \alpha \text{S\_RTT}_n$$

$$\text{E\_RTT}_n = (1-\alpha)^2 ((1-\alpha) \text{E\_RTT}_{n-3} + \alpha \text{S\_RTT}_{n-2}) + \alpha(1-\alpha) \cdot \text{S\_RTT}_{n-1} + \alpha \cdot \text{S\_RTT}_n$$

$$\text{E\_RTT}_n = (1-\alpha)^3 \text{E\_RTT}_{n-3} + \alpha(1-\alpha)^2 \cdot \text{S\_RTT}_{n-2} + \alpha(1-\alpha) \cdot \text{S\_RTT}_{n-1} + \alpha \cdot \text{S\_RTT}_n$$

$$\text{E\_RTT}_n = 0,6699 \text{E\_RTT}_{n-3} + 0,0957 \text{S\_RTT}_{n-2} + 0,1094 \text{S\_RTT}_{n-1} + 0,125 \text{S\_RTT}_n$$

(terzultima misurazione)      (penultima misurazione)      (ultima misurazione)

### 3.9.3 Deviazione del RTT

La natura variabile dell'RTT ci porta a voler tener conto anche del valore della deviazione standard.

$$\text{Dev\_RTT} = (1 - \beta) \cdot \text{Dev\_RTT} + \beta \cdot |\text{Sample\_RTT} - \text{Estimated\_RTT}|$$

Un valore tipico è  $\beta = \frac{1}{4} = 0.25$ .

### 3.9.4 Risultato finale - Retransmission Time-out

Dati alla mano, possiamo finalmente stabilire l'RTO, ovvero il valore da dare al timer.

$$\text{Retransmission Time-Out} = \text{Estimated\_RTT} + 4 \cdot \text{Dev\_RTT}$$

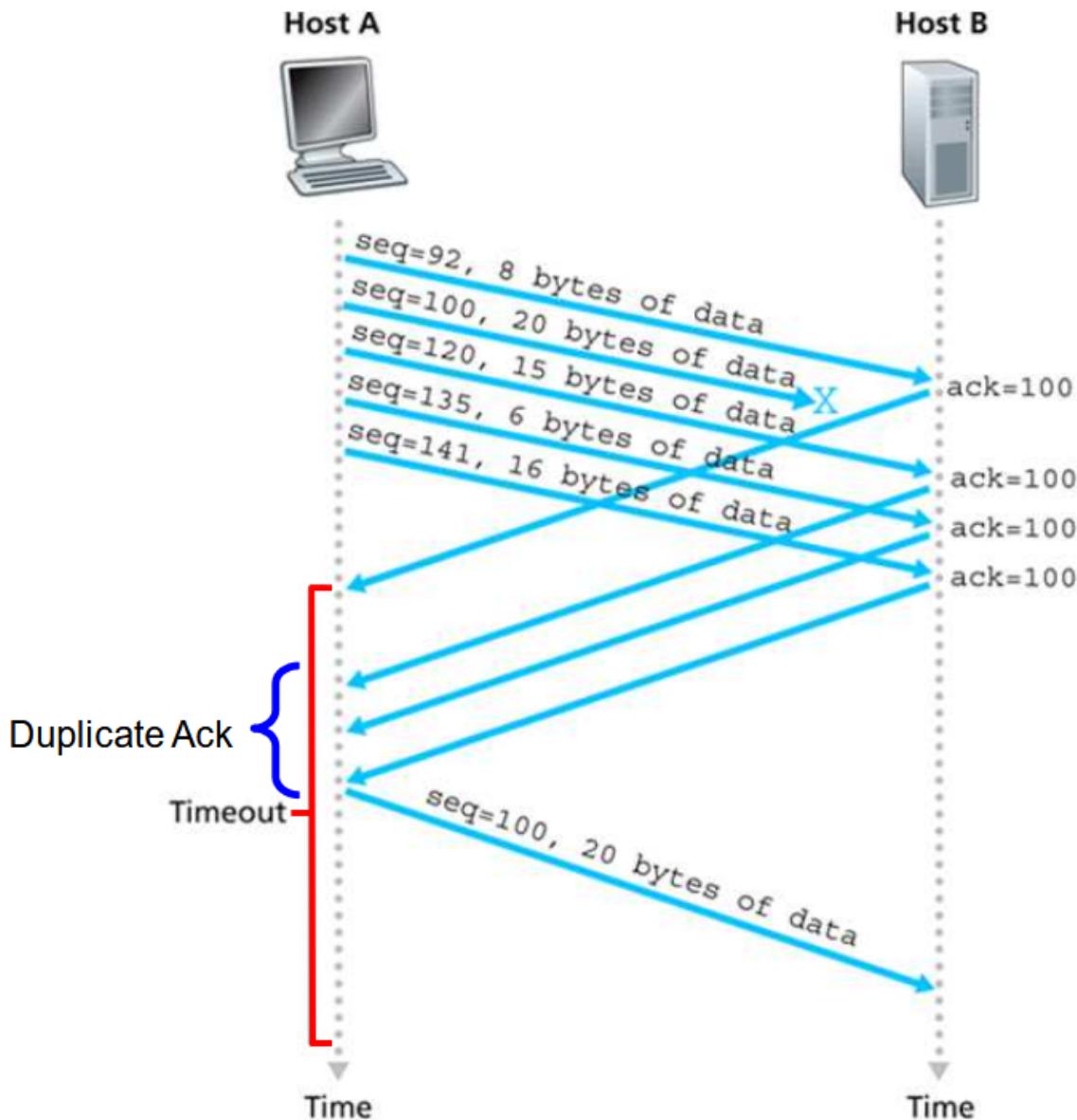
Con un valore di partenza pari a un secondo (RFC 6298).

## 3.10 Timer

Nonostante RDT stabilisca che ad un pacchetto venga associato un timer, TCP riduce la complessità di gestione gestendo esclusivamente un timer alla volta. Dopo la ricezione dell'ACK dell' $n$ -esimo pacchetto, il timer per il segmento  $n + 1$  non ancora acknowledged viene avviato.

### 3.10.1 Fast-Retransmit

Per ridurre il tempo di attesa dato dal time-out, si include un'altra condizione per le ritrasmissioni. Questo meccanismo è detto fast retransmit: se il sender manda tre ACK consecutivi per lo stesso pacchetto mancante (ovvero l'ACK dell'ultimo pacchetto consegnato con successo), viene attivata istantaneamente la ritrasmissione del pacchetto.



Questo meccanismo si basa sulla considerazione che, dopo il terzo ACK relativo allo stesso pacchetto non pervenuto, è probabile che questo si sia perso, e vada ritrasmesso.

## 3.11 TCP - Controllo di flusso

Il problema del produttore-consumatore si ripresenta anche qui: se il Network Layer fornisce dati al buffer del layer di trasporto ad una velocità superiore rispetto a quella con cui l'applicazione può toglierli dal buffer, si corre il rischio di saturarlo. **Gestire in maniera consona il flusso è fondamentale** per evitare di dover scartare pacchetti, ed il protocollo TCP implementa dei meccanismi con questo scopo. **UDP non ha meccanismi equivalenti.**

### 3.11.1 Finestra di ricezione

Il controllo del flusso viene gestito tramite una variabile chiamata **Receive Window**, **rwnd**, che specifica al mittente lo spazio libero nel buffer di ricezione del destinatario. Il valore **rwnd** viene aggiornato e trasmesso al mittente tramite un campo dell'header TCP degli ACK.

Quando viene instaurata una connessione TCP, il destinatario riserva un buffer di ricezione denominato **RcvBuffer**, di cui la dimensione può essere configurata tramite le opzioni del socket.

$$\text{rwnd} = \text{ReceiverBufferSize} - (\text{LastByteReceived} - \text{LastByteRead})$$

Dove

- **LastByteReceived** è il numero dell'ultimo byte ricevuto dal livello di rete e inserito nel buffer.
- **LastByteRead** è il numero dell'ultimo byte letto dal buffer dall'application layer.

#### Circostanza problematica

Quando il buffer del destinatario si riempie completamente, il receiver manda  $\text{rwnd} = 0$ , il sender si blocca. Da qui in poi, sia sender che receiver si bloccheranno: come fa il receiver ad aggiornare il sender senza ACK da mandare? Per risolvere questo problema, il protocollo TCP impone un meccanismo in cui il sender invia dei **probe packets** contenenti un singolo byte di dati forzando una risposta dal ricevente.

### 3.11.2 Algoritmo di Nagle

L'algoritmo di Nagle è una tecnica utilizzata nei protocolli di trasporto, in particolare nel protocollo TCP, per ridurre l'overhead generato dall'invio di piccoli pacchetti (ricordiamo che l'RTT è un tempo che viene atteso ad ogni trasmissione di dati). Questo è particolarmente utile nelle applicazioni che generano molte trasmissioni di dati di piccole dimensioni, come Telnet, che invia pacchetti di dimensioni minime (tipicamente di 1 byte). L'obiettivo è **aggregare dati di piccole dimensioni** nel buffer di trasmissione, **per inserirli in pacchetti più grandi quando l'RTT della rete è alto**.

#### Algoritmo

1. Verificare se ci sono dati tra trasmettere. Se sì, step 2, altrimenti step 7.
2. Verificare se  $\text{rwnd} \geq \text{MSS}$  e che i dati siano  $\geq \text{MSS}$ . Se sì, step 5, altrimenti step 4.
3. Verifica se stai aspettando un ACK. Se sì, step 5, altrimenti 6.
4. Accoda i dati da mandare fino alla ricezione dell'ACK, vai a step 7.
5. Manda i dati, end.

#### Pseudocodice

```
if available_data > 0:  
    if window_size >= MSS AND available_data >= MSS:  
        send_a_MSS_segment  
    else:  
        if waiting_for_an_ack == true:  
            enqueue_data_in_buffer /* until an acknowledge is received */  
        else:  
            send_data
```

In programmi che richiedono bassa latenza e alta reattività, alcuni sistemi operativi possono disabilitare questo algoritmo tramite l'opzione **TCP\_NODELAY** nel socket API.

## 3.12 Connessione TCP

Il three-way handshake è tra gli aspetti caratterizzanti del protocollo TCP. Andiamo adesso ad analizzarlo nel dettaglio, e ad analizzare perché un two-way handshake non sarebbe funzionale.

### 3.12.1 2-way handshake (un'intuizione inconcludente)

Prima dell'introduzione del three-way handshake, si è considerato un approccio più semplice (e vicino a quello umano) con un **two-way handshake**. Si è presto realizzato che un two-way handshake apre le porte a situazioni di **connessioni half open**.

1.  $A$  richiede la connessione a  $B$ .
2.  $B$  manda un'ACK, apre la connessione
3.  $A$  riceve l'ACK, partecipa alla connessione.

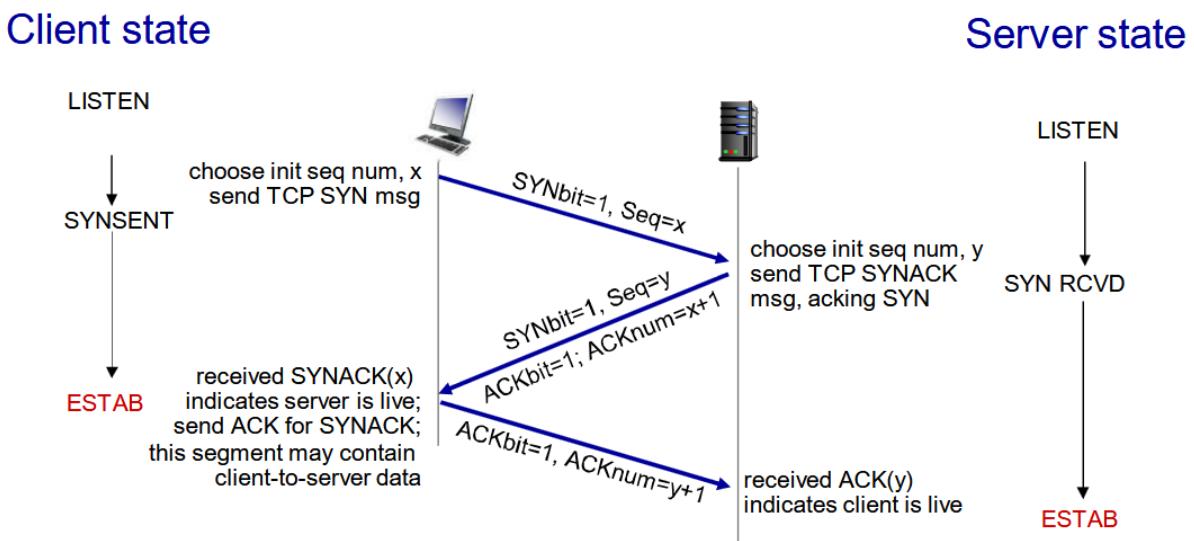
**E se l'ACK si perde?**

1.  $A$  chiede la connessione a  $B$ .
2.  $B$  manda un'ACK e apre la connessione.
3. L'ACK si perde o il client  $A$  crasha. La connessione del server  $B$  rimane aperta e in attesa.

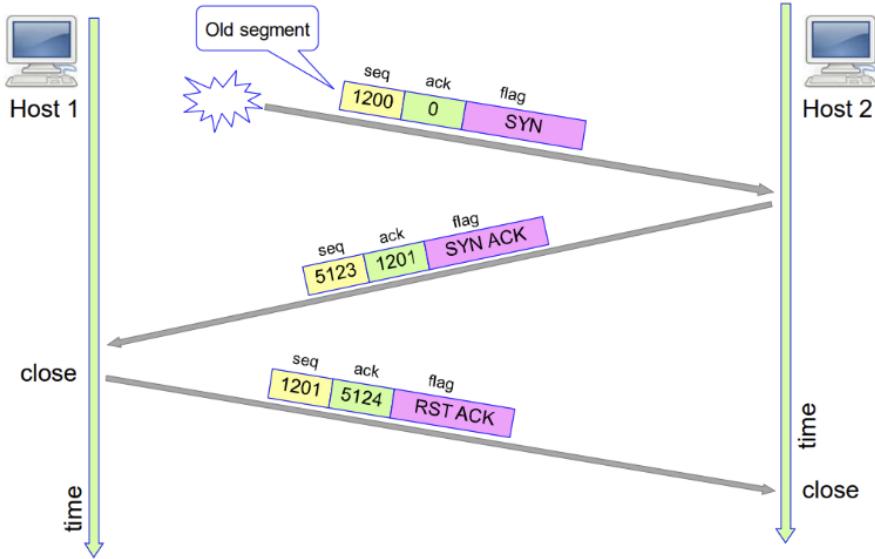
Il server  $B$  entrerà nello stato **ESTABLISHED**, il client  $A$  invece no. Si ottiene quindi una connessione half-open anomala e inutilizzabile.

### 3.12.2 3-way handshake

Il three-way handshake gestisce queste circostanze tramite un flag **RST** che può essere messo a 1 in un segmento TCP.



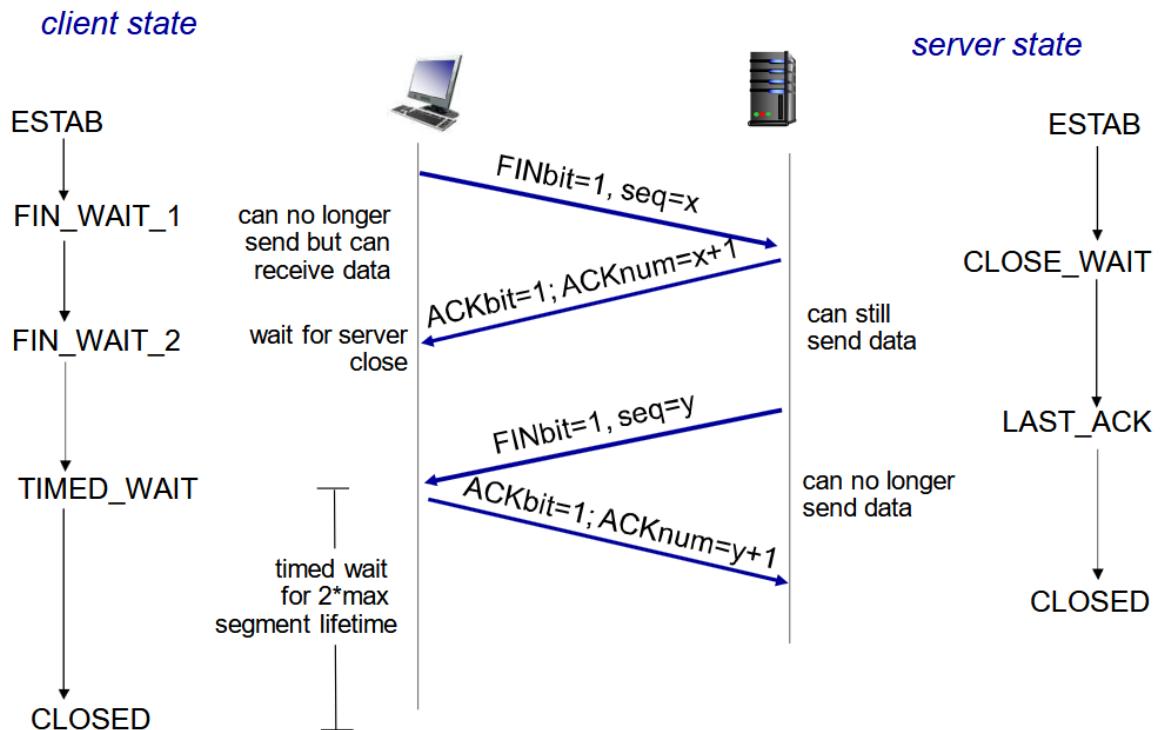
Se l'host  $A$  riceve un segmento TCP che **non riconosce** (e quindi da una connessione che non è stata stabilita con successo, presumibilmente half-open), questo manderà un segmento TCP di **RST ACK** con **RST=1** per chiedere l'interruzione immediata di questa presunta mezza connessione.



### 3.12.3 4-way handshake ( $2 \times 2$ -way handshake) per chiusura connessioni

La chiusura delle connessioni TCP avviene tramite due 2-way handshake.<sup>5</sup>

1. Il client termina l'invio dei dati: invia segmento con flag FIN.
2. Il server risponde con ACK.
3. Il server è pronto a chiudere la connessione: invia segmento con flag FIN.
4. Il client risponde con ACK, chiudendo definitivamente la connessione.



Osserviamo che dopo l'invio dell'ultimo FIN-ACK, si avvierà un timer **TIME-WAIT** di tempo pari a 2 volte tempo di vita massimo di un segmento: ciò darà il tempo ai pacchetti residui nella rete di arrivare (o di far scadere il loro timer) per evitare che vengano ricevuti a connessione ormai chiusa.

<sup>5</sup>Si è deciso di sottolineare questa parte in quanto è una puntalizzazione molto importante, soprattutto in sede d'esame.

### 3.13 Controllo della congestione

La **gestione delle trasmissioni** è fondamentale per evitare non solo di sovraccaricare i receiver (eventualità gestita tramite il **controllo del flusso**), ma anche per evitare il sovraccarico della rete, con conseguente congestione. Il protocollo TCP gode di meccanismi per gestire la congestione, per ridurre la velocità di trasmissione e minimizzare i danni della congestione.

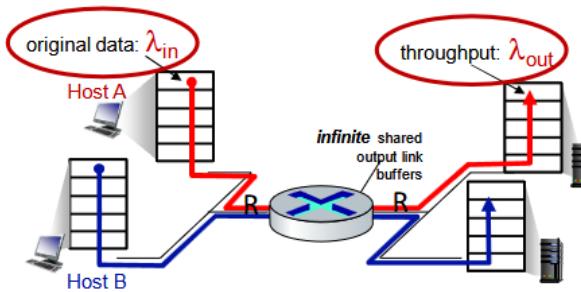
#### 3.13.1 Scenario 1 - due mittenti e receiver, router con buffer illimitato

Nell'analisi della congestione di rete, è importante conoscere dei valori, quali

- $\lambda_{in}$ , frequenza media di invio (dai mittenti).
- $\lambda_{out}$ , la frequenza di arrivo di byte al ricevente.
- $R$ , la capacità del collegamento condiviso.

Quando  $\lambda_{in} \geq \lambda_{out}$ , inizia a crearsi congestione.

Consideriamo due host mittenti  $A$  e  $B$  che inviano a due host destinatari attraverso un router intermedio, e supponiamo che il suo buffer sia illimitato.



- Se  $\lambda_{in} \leq \frac{R}{2}$ , il router riesce ad inoltrare i pacchetti senza ritardi significativi, e  $\lambda_{out} = \lambda_{in}$ . Dividiamo  $R$  in due perché abbiamo due host che mandano.
- Se  $\lambda_{in}$  aumenta fino a  $\frac{R}{2}$ , il  $\lambda_{out}$  massimo si stabilizza su  $\frac{R}{2}$ .
- Se  $\lambda_{in} > \frac{R}{2}$ , il throughput non aumenta, ma il numero di pacchetti accodati nel buffer del router cresce senza limite, aumentando il ritardo medio di trasmissione.

### 3.13.2 Scenario 2 - due mittenti e receiver, un router con buffer limitato

Quando il buffer di un router è pieno, pacchetti vengono scartati. Di fronte a circostanze simili, TCP prevede ritrasmissione, con conseguente aumento della latenza complessiva aumenta e riduzione frequenza media di invio  $\lambda_{in}$  diminuisce, in quanto vengono rimandati gli stessi pacchetti.  $\lambda'_{in}$  tiene conto sia dei dati originali, che delle ritrasmissioni. Distinguiamo ora dei casi di quest scenario.

#### 1. Nessuna perdita di pacchetti - Perfect Knowledge.

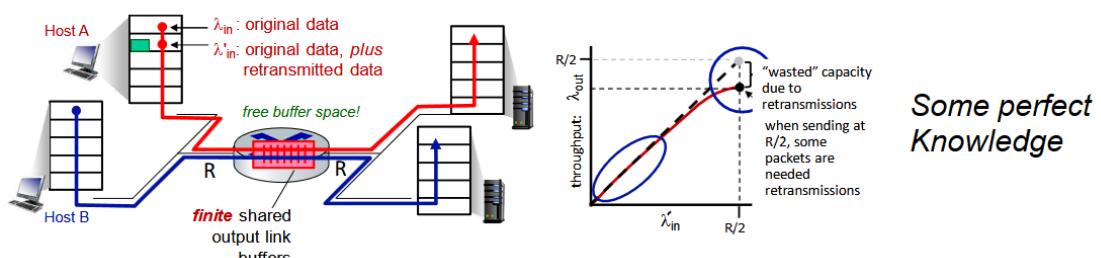
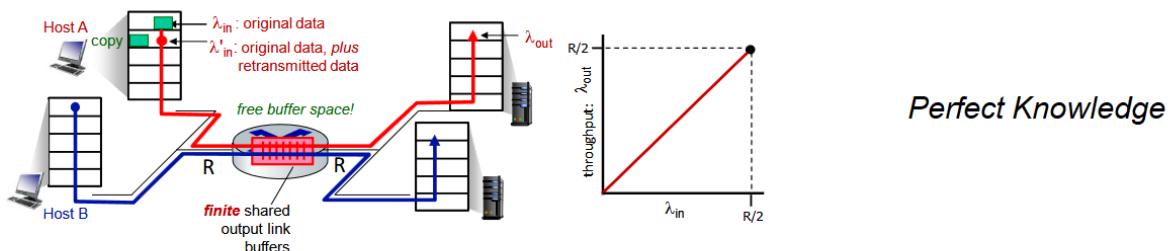
Se i sender fossero in grado di determinare in anticipo se il router ha spazio disponibile nel buffer, invierebbe un pacchetto solo quando può essere immediatamente elaborato, ottenendo  $\lambda_{in} = \lambda'_{in}$ ,  $\lambda_{out} = \lambda_{in} \leq \frac{R}{2}$  e zero latenza (perlomeno, causata dal router).

#### 2. Ritrasmissione solo per pacchetti persi - Some perfect Knowledge.

I sender non conoscono lo stato del buffer, e ritrasmettono un pacchetto solo quando hanno la certezza che sia stato perso. Supponendo che  $\lambda'_{in} = \frac{R}{2}$ ,  $\lambda_{out} < \frac{R}{2}$  a causa delle ritrasmissioni.

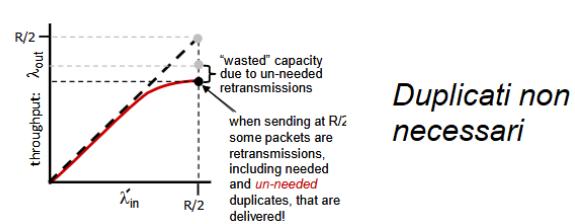
#### 3. Caso reale - duplicati non necessari.

I sender non conoscono lo stato del buffer, e potrebbero avere un **timer di ritrasmissione troppo breve**. Questo implica un utilizzo innecessario del buffer del router, della banda e delle risorse di elaborazione. Questo potrebbe addirittura portare ad un throughput di circa  $\lambda_{out} = \frac{R}{4}$ .



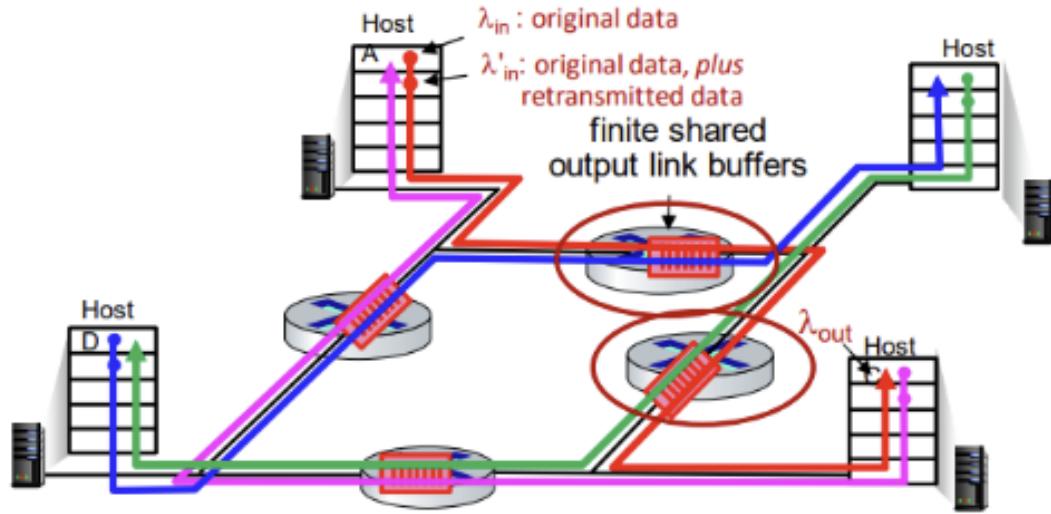
#### Realistic scenario: un-needed duplicates

- packets can be lost, dropped at router due to full buffers – requiring retrasmissons
- but sender times can time out prematurely, sending **two** copies, **both** of which are delivered



### 3.13.3 Terzo scenario - quattro mittenti, più router limitati e percorsi

In questo scenario, quattro mittenti inviano pacchetti su percorsi composti da più collegamenti, condividendo router intermedi. Ogni host utilizza un meccanismo di ritrasmissione basato su timeout, e la rete ha buffer di dimensione finita.



Abbiamo quattro sender che inviano pacchetti in quattro percorsi con più router, e quindi più possibili percorsi. Ogni router trasmette a capacità  $R$ . A valori di  $\lambda_{\text{in}}$  piccoli,  $\lambda_{\text{in}} = \lambda'_{\text{in}}$ , non si hanno overflow dei buffer e nemmeno ritrasmissioni. Al crescere del valore di  $\lambda_{\text{in}}$ , inizia il rischio di ritrasmissioni, per cui  $\lambda'_{\text{in}} > \lambda_{\text{in}}$ .

## 3.14 Meccanismi per gestire la congestione

Abbiamo quindi capito che la congestione è un fenomeno che si riscontra quando **il traffico si avvicina o supera la capacità della rete  $R$** . Per limitarlo, si opera sulla finestra di trasmissione degli host **cwnd**, aumentando e riducendo in maniera opportuna il numero di pacchetti consentiti contemporaneamente sulla rete (senza aver ricevuto un ACK).

### 3.14.1 Due approcci

- **Approccio Network-assisted.**

I router forniscono un **feedback diretto** agli host sender e receiver (di un flusso passante lo stesso router) in merito alla congestione della rete. I router possono fornire il livello di congestione, o esplicitare una **frequenza di invio**.

- **Approccio end-to-end.**

La rete non da alcun tipo di feedback: la congestione è **dedotta** da ritardi dei pacchetti e ritardi.

### 3.14.2 AIMD - Additive Increase Multiplicative Decrease

Aumento lineare e decremento moltiplicativo della **cwnd**.

- Il sending rate è incrementato di 1 MSS a ogni RTT, fino a quando non è individuata una perdita di segmenti.
- Dimezza il rate d'invio per ogni evento di perdita segmenti.

Ogni incremento viene fatto operando sulla variabile **cwnd**, ovvero la finestra di congestione. Il grafico che si ottiene osservando la crescita (e decrescita) del sending rate nel tempo, è detto **saw-tooth**, dente di sega. Questa tecnica ha tre fasi:

1. **Slow Start**, inizio lento.

Il mittente inizia con una congestion window di 1, ma che cresce, moltiplicata per 2 ad ogni RTT, fino a una soglia stabilita dalla variabile **ssthresh**. Oltre quella soglia, si entra in congestion avoidance.

2. **Congestion Avoidance**.

Superata la **ssthresh**, e quindi circa la soglia di  $\frac{cwnd}{2}$ , inizia la congestion avoidance: **la crescita diventa lineare** e  $ssthresh = cwnd/2$ .

3. **Fast Recovery**.

Congestion avoidance procede normalmente fino a un **time-out** o tre **ACK**.

(a) Se si verifica un time-out, **cwnd** è settato a 1 MSS.

(b) Se si verifica un triplo ACK, **cwnd** è dimezzato.

Il valore di **cwnd** incrementa di 3 MSS, e quindi  $ssthresh = (cwnd + 3)/2$ .

### 3.14.3 Fast Recovery - TCP Tahoe e Reno

- **TCP-Tahoe.**

Se si perde un pacchetto (individuata da timeout o 3 ACK duplicati), la **ssthresh** viene dimezzata, e il **cwnd** viene azzerato, ripartendo da 1 MSS a priori. Si rientra sempre in slow start dopo qualsiasi evento di perdita.

- **TCP-Reno.**

Se si perde un pacchetto (solo se individuata 3 ACK duplicati), fast recovery! **ssthresh** viene dimezzata, e il **cwnd** riparte da un valore  $\approx ssthresh$ .

Se si rileva la perdita di pacchetti tramite un **time-out**, si rientra in **Slow Start**.

TCP Reno è preferibile quando la rete è abbastanza stabile e le perdite di pacchetti sono occasionali: in questo caso, il suo meccanismo di Fast Recovery permette di mantenere buone prestazioni evitando di ritornare sempre allo slow start. TCP Tahoe è invece più adatto in canali altamente congestionati, perché reagisce in modo più drastico alle perdite (tornando sempre a slow start), offrendo un comportamento più prudente e sicuro a scapito dell'efficienza. Insomma, preferire Tahoe a Reno dipende dal canale di trasmissione.

### 3.14.4 ECN - Explicit Congestion Notification

È un approccio alternativo alla gestione della congestione della rete. Sfrutta due bit nel campo Type Of Service dell'header IP per comunicare la congestione agli host sender.

- Se un router rileva congestione prima di perdere pacchetti, marca il pacchetto con i bit ECN-Capable ECN = 10 (o ECN = 01, sono equivalenti) nell'header IP. Il campo rimane ECN = 00 se ciò non è supportato.
- Se la congestione è alta, il router imposta il bit ECN = 11
- Arrivato il pacchetto al destinatario, questo manda un ACK con il bit ECN = 11, che significa "Congestion Experienced". In questo modo, il lavoro di notifica è delegato al receiver, e non al router, che manipolerà esclusivamente i bit di quel campo IP.
- Il mittente che riceve l'ACK col flag ECN = 11 abbassa la velocità di invio.

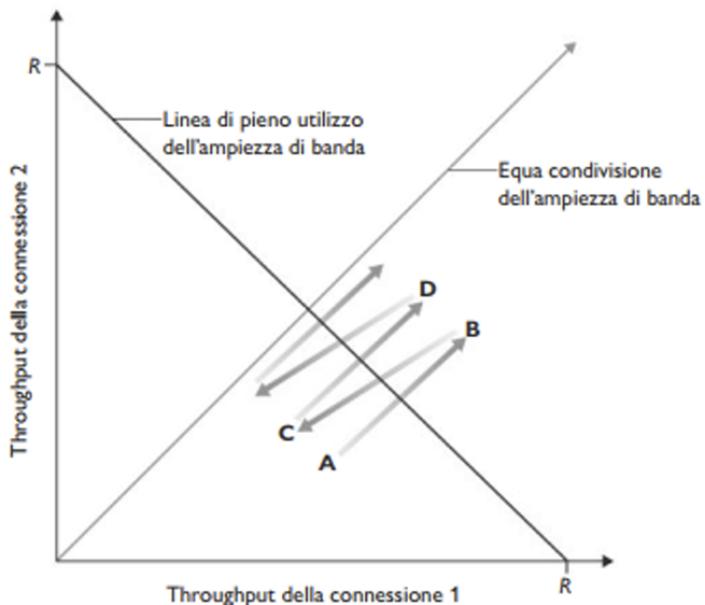
## 3.15 TCP Fairness

TCP è un protocollo fair? Cosa intendiamo per fairness?

Si desidera che il controllo della congestione in TCP sia tale che, date  $K$  connessioni attraverso la stessa rete con capacità trasmissiva  $R$ , la cwnd di ogni connessione sia  $\frac{R}{K}$ .

### Dimostrazione della fairness

Date due connessioni con pari  $MSS$  e  $RTT$  (e nessun'altra connessione attraverso il collegamento) che operano in **congestion avoidance**. Con un throughput massimo pari a  $R$ , e tenendo a mente che al crescere di uno, diminuisce l'altro, il throughput in una situazione di equilibrio è pari a  $\frac{R}{2}$ .



A parità di  $RTT$  e  $MSS$ , gli host in **congestion avoidance crescono in maniera lineare**: 1  $MSS$  ad ogni  $RTT$ . Detto ciò, quando la somma del throughput di entrambi gli host supera  $R$ , questi si **dimezzano**. Con  $x$  = throughput del primo host e  $y$  = throughput del secondo host, e il comportamento di incremento lineare e divisione della finestra di congestione, **il throughput delle connessioni convergerà sempre alla bisettrice  $x = y$** . In circostanze più vicine a quelle reali, le connessioni con  $RTT$  più basso ottengono un throughput maggiore a causa della velocità con cui possono ottenere la connessione. Ricordiamo che l'incremento, nella fase di congestion avoidance, avviene una volta per  $RTT$ .

## Capitolo 4

# Panoramica del Network layer

### 4.1 Introduzione al Network Layer

Il compito principale del Network Layer è instradare i pacchetti del livello transport, gestendo il trasferimento e la comunicazione host-to-host.

#### 4.1.1 Instrandare vs Trasferire

Sono tra le responsabilità principali del livello di rete, e non sono la stessa cosa. Non sono sinonimi, ma solo **concetti collegati**.

- **Routing, instradamento.**

Definiamo come **routing** il processo di **selezione e definizione di cammini** (path, route) per i pacchetti all'interno o tra reti. Indica anche i processi relativi alla **gestione del traffico** generale della rete. *Routing is the process of selecting and defining paths for IP-packet traffic within or between networks as well as the process of managing network traffic overall.*

- **Forwarding, trasferimento.**

Avviene dopo il routing, **consiste nell'inoltro effettivo dei pacchetti**, secondo i termini stabiliti dal routing.

#### 4.1.2 Data plane vs Control plane

La vastità degli argomenti che riguardano il Network Layer, ci spingono a suddividere la discussione tra **data plane** e **control plane**.

- **Data plane.**

Il dataplane è la parte del router o dello switch che si occupa del trasferimento effettivo dei pacchetti, inoltrandoli da un'interfaccia all'altra secondo i termini stabiliti del control plane. Opera in modo veloce e automatico, spesso in hardware. È quindi il responsabile del forwarding!

- **Control plane.**

Il control plane è responsabile di prendere decisioni sul percorso dei pacchetti, costruendo e aggiornando le tabelle di routing seguendo dei protocolli dedicati. Gestisce la logica e la configurazione della rete. È chiaramente il responsabile del routing.

## 4.2 Routing

Esistono due approcci differenti:

- **Traditional Routing Algorithms.**

Effettuato dai singoli router, senza una visibilità globale, ma dei propri neighbours. È una struttura distribuita, più resistente della prossima soluzione, ma più lenta. Ogni router ha al proprio interno una tabella d'instradamento, cui valori vengono aggiornati di volta in volta.

- **Software Defined Network.**

È un'architettura centralizzata esterna, che sa tutto della rete. Ottiene path migliori e più velocemente, ma se l'SDN va giù, il sistema crolla. Inoltre, riceverà un altissimo numero di richieste non-distribuite, in quanto, in questo tipo di architettura, è proprio il controller remoto a calcolare e distribuire le tabelle di inoltro.

### 4.2.1 Self-Organizing-Routing

Scambiandosi informazioni, ottengono una visione generale della rete. Determinano un routing migliore tramite algoritmi quali Bellman-Ford e Dijkstra. Questi algoritmi sono rinforzati tramite euristiche, tecniche AI e di reinforcement learning. Andremo ad approfondire questa parte affrontando il control-plane.

### 4.2.2 SDN - Software Defined Networking

Il control plane è centralizzato in un remote controller.

Conosce tutta la topologia della rete, e prende decisioni sui percorsi che devono seguire i pacchetti, sia come aggiornare le tabelle di inoltro nei router. Manda istruzioni ai router sull'instradamento e riceve aggiornamenti dai router sul traffico. Il control agent (CA) è un modulo che gira sui router che permette di comunicare col Remote Controller. Il remote agent non agisce mai in autonomia dal remote controller.

### 4.2.3 Virtual Circuit Service, circuito virtuale

È una modalità di commutazione in cui, tra due peer, viene costruito un **circuito virtuale** in fase di connessione (secondo i soliti metodi di routing). L'inoltro dei pacchetti **di quella connessione** procederà esclusivamente sul percorso stabilito. In queste circostanze è chiaro che i router **non siano state-less**: manterranno infatti delle informazioni di routing relative a quella connessione. Un qualsiasi **malfunzionamento** dei router nel percorso, implica una **riconnessione da zero**. In compenso, il **controllo della congestione** è molto più **semplice**.

### 4.2.4 Packet Switching, commutazione di pacchetto

La modalità a circuito virtuale è molto diversa alla più usata modalità di **packet switching** (ovvero della **commutazione a pacchetto**) in cui tutti i **pacchetti**, anche della stessa connessione, sono inoltrati separatamente sulla base delle informazioni contenute negli **header**. Instradano ogni pacchetto secondo le proprie tabelle di routing, e non pensando ad un circuito fisso tra due host.

### 4.2.5 Router

Al giorno d'oggi sono dei "mini-stack ISO/OSI", cui livello più alto è il transport. Attualmente, i router contengono un *firmware*<sup>1</sup> nella propria memoria flash.

- Esamina gli **header** dei datagrammi IP che gli passano attraverso.
- Sposta i datagrammi **dalla porta di input** a quella di **output** per trasferire i datagramma nel path stabilito. Effettuano un **forward** dei pacchetti.

### 4.2.6 Indirizzamento logico

Consiste nell'assegnamento di **indirizzi logici** ai dispositivi (tramite indirizzi IP) per **identificare** in modo univoco **ogni nodo nella rete**.

<sup>1</sup>Sono programmi scritti nella memoria flash, solitamente in linguaggio di basso livello (Assembly, C di basso livello). Sono software che la CPU può eseguire direttamente grazie al program counter. All'accensione del dispositivo, il codice viene copiato nella SRAM, e la CPU lo esegue. Questi firmware sono velocissimi, ed è il motivo per cui rispondono alle esigenze dei router.

## 4.3 Frammentazione e riassemblaggio

I datagrammi IP troppo grandi per il router devono essere **frammentati e riassemblati al mittente** per renderne possibile il trasferimento. Questo è uno dei compiti del data plane del Network Layer (la MTU dell'Ethernet è di 1500 byte). In dei casi, si vorrebbe poter avere la possibilità di controllare se permettere o meno la frammentazione. Questo è ottenuto tramite dei flag opportuni dell'header IPv4 (Don't Fragment e More Fragment). Scopriremo anche che IPv6 non permette la frammentazione da parte dei router intermedi.

## 4.4 Modelli di servizio di rete

In inglese, **Network Service Model**. Stabiliscono le caratteristiche del trasporto dei datagrammi da host a host.

I servizi possono fare riferimento ai singoli pacchetti, quali

- **Consegna garantita.**
- Consegnata **entro  $n$  millisecondi**.

O a interi flussi di datagrams, come

- **Consegna dei datagrammi in ordine.**
- **Bandwidth minima garantita.**
- **Maximum jitter**, garantisce che la variazione massima tra i ritardi dei pacchetti.

Su Internet, garantire questi servizi è semplicemente utopia. In campo militare o finanziario, alcuni di questi servizi devono essere garantiti. Nelle reti degli ATM, viene usato un servizio di **Available Bit Rate**. Garantisce una bandwidth minima utilizzando la banda di rete disponibile.

### 4.4.1 Internet Service Model - Best Effort

In Internet, le **garanzie in termini di reliability** sono affidate al **livello di trasporto**, tramite **RDT** e **meccanismi vari del TCP**. Il network, nel caso di Internet, non si occupa di assicurare i servizi appena esposti, ma di offrire dei meccanismi di instradamento e inoltro dei pacchetti per i livelli superiori.

#### Best Effort - Complice della popolarità di Internet?

La semplicità dei meccanismi dietro ai servizi **best-effort di Internet**, ha indubbiamente permesso a quest'ultimo di **espandersi a macchia d'olio**.

Con una larghezza di banda sufficientemente ampia, le applicazioni funzionano piuttosto bene la maggior parte delle volte. **Good enough!**

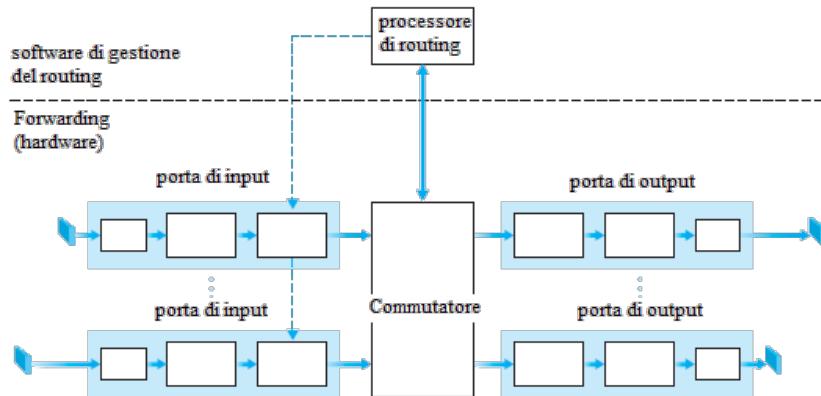
# Capitolo 5

## Network Layer - Piano dei dati

### 5.1 I router

I router sono sicuramente i dispositivi più responsabili dell'instradamento e dell'inoltro all'interno delle reti. Sono quindi un elemento fondamentale per il Network Layer, e quindi da studiare approfonditamente.

#### 5.1.1 Elementi principali di un router



- **Porte di ingresso.**

Sono le **terminazioni a livello fisico** per i collegamenti in ingresso al router. Inoltre, è sempre nella porta d'ingresso che si effettua la **decisione di forwarding**. Ogni porta contiene infatti una copia della tabella di routing. In questo caso, il termine "porta" fa riferimento a interfacce fisiche d'input (e output), ben diversi dalle porte software dei processi e dei socket.

- **Struttura di commutazione.**

Connette fisicamente le porte d'ingresso a quelle di uscita. Una sorta di rete interna al router. È una struttura che deve permettere una **commutazione ad alta velocità**, sfruttando la tabella di forwarding fornita dal piano di controllo.

- **Porte di uscita.**

Memorizzano i pacchetti che provengono dalla struttura di commutazione e li trasmettono sul collegamento d'uscita. Nei collegamenti bidirezionali, la porta d'uscita è accoppiata alla porta d'ingresso sulla stessa scheda di collegamento (detta line card).

- **Processore d'instradamento.**

Esegue le funzioni del piano di controllo. Il funzionamento varia tra i router tradizionali e quelli SDN<sup>1</sup>. Nei primi, il processore d'instradamento gestisce le tabelle di inoltro. Nel secondo caso, si occupano di gestire la comunicazione con il controller remoto.

In un router, mentre il processore d'instradamento è principalmente software, le porte di ingresso, di uscita e le strutture di commutazione, sono gestite da hardware dedicato. In questo modo, è possibile ottenere le performance richieste dalla rete, operando in lassi temporali dell'ordine dei millisecondi.

<sup>1</sup>Software Defined Networking

### 5.1.2 Porte d'ingresso del router

Le porte d'input svolgono sia una funzione di terminazione (elettrica) per il livello fisico, sia l'elaborazione per il livello di collegamento.

Usando le informazioni della tabella d'inoltro, viene determinata la porta d'uscita dei pacchetti (attraverso la struttura di commutazione).

La tabella di inoltro viene elaborata e aggiornata dal processore di instradamento (control plane), e viene copiata e mandata a line card e porte d'ingresso, in modo da permettere alle porte d'ingresso di prendere decisioni senza invocare il processore d'instradamento centralizzato.

### 5.1.3 Tabelle d'inoltro e destination-based forwarding

forwarding table	
Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

La decisione della porta d'inoltro può avvenire tramite un processo di longest-prefix-matching (in italiano, corrispondenza a prefisso più lungo). L'indirizzo di destinazione viene confrontato con una lista di prefissi associati a dei collegamenti: quello dal match migliore, detterà la porta d'uscita dei pacchetti.

Destination Address Range	Link interface
11001000 00010111 00010*** *****	0
11001000 00010111 00011000 *****	1
11001000 00010111 00011*** *****	2
otherwise match!	3

examples:

11001000 00010111 00010110 10100001 which interface?
11001000 00010111 00011000 10101010 which interface?

### Ternary Content Addressing Memory

Lo string matching non è un problema semplice, nemmeno nel caso dei prefissi: soprattutto in un contesto come quello dei router, che devono operare ad altissime velocità, risulta fondamentale implementare questi meccanismi in maniera hardware.

Entrà quindi in gioco la TCAM. La Ternary Content-Addressable Memory (TCAM) è un tipo specializzato di memoria ad alta velocità che può cercare all'interno del suo intero contenuto in un solo ciclo di clock. Il termine ternario si riferisce alla capacità della memoria di memorizzare e interrogare i dati utilizzando tre diversi valori: 0, 1 e X (perfetta per i prefissi!). La TCAM è utilizzata nel contesto dei router per memorizzare le tabelle di ricerca degli indirizzi, ed effettuare in tempo costante il controllo di tutti i prefissi, per trovare il prefisso dal matching migliore con l'indirizzo di destinazione fornito.

### 5.1.4 Struttura di commutazione

È il cuore del router, mezzo tramite cui i pacchetti vengono commutati (inoltrati) dalla porta d'ingresso alla porta d'uscita. La commutazione può essere ottenuta attraverso varie strutture di commutazione:

- **Commutazione in memoria.**

Avviene sotto il controllo diretto della CPU (ovvero il processore di instradamento), con le porte di ingresso e uscita trattate come dispositivi di I/O. I primi router usavano questa struttura, che con una determinata bandwidth della memoria tale da permettere di leggere e scrivere  $B$  pacchetti per unità di tempo, il throughput complessivo d'inoltro deve necessariamente essere  $< \frac{B}{2}$ . Le operazioni di inoltro su due porte differenti non potevano nemmeno avvenire in contemporanea, a causa dell'impossibilità di effettuare due operazioni di lettura/scrittura contemporaneamente.

Alcuni router attuali effettuano commutazione in memoria, ma la ricerca dell'indice e la memorizzazione del pacchetto nella locazione di memoria opportuna avviene sulle line card di ingresso. Alcuni di questi router, somigliano a sistemi multiprocessore a memoria condivisa.

- **Commutazione tramite bus.**

In questo approccio le porte di ingresso trasferiscono un pacchetto direttamente alle porte di uscita tramite un bus condiviso e senza intervento dal processo di instradamento.

Si ottiene aggiungendo al pacchetto un'etichetta di commutazione: questa indicherà la porta locale di output da cui dovrà uscire. Il pacchetto verrà ricevuto da tutte le porte d'output, ma solo la porta corrispondente lo raccoglierà. Un difetto di questo sistema? Solo un pacchetto alla volta può occupare il bus: la larghezza di banda della commutazione sarà limitata da quella del bus. Più pacchetti da commutare saranno messi in coda. È comunque un valido sistema per router in reti di accesso e aziendali. Gli switch Cisco della serie 5600 hanno una bandwidth di 32 Gbps.

- **Commutazione attraverso rete di interconnessione.**

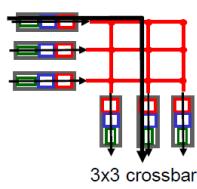
Usa una rete di interconnessione più sofisticata, con una matrice di commutazione (*crossbar switch*): consiste in  $2n$  bus che collegano  $n$  porte di ingresso a  $n$  porte d'uscita. Ogni bus verticale intersecca tutti i bus orizzontali a un punto d'incrocio che può essere aperto o chiuso dal controller della struttura di commutazione in qualsiasi momento. Quando un pacchetto che entra dalla porta  $A$  deve essere inoltrato alla porta d'uscita  $Y$ , viene chiuso l'incrocio tra i bus  $A - Y$ .

Analogamente, questo avviene per l'entrata  $B$  e l'uscita  $X$ . Quando sono coinvolte porte d'uscita differenti, i pacchetti possono essere trasferiti in parallelo, altrimenti si accorderanno alla porta d'output.

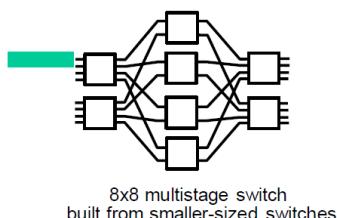
Architetture più sofisticate utilizzano strutture per la commutazione a più stadi, che permette di spostare in contemporanea pacchetti da più porte verso una sola porta.

Tra questi, i router Cisco CRS usano un'ulteriore strategia a  $n$  strutture di commutazione: il pacchetto da trasferire viene suddiviso in  $k$  frammenti (chunk). Ogni frammento verrà trasferito attraverso una struttura, mentre la porta di uscita si occuperà del riassemblaggio. Questo permette di avere ancora  $n - k$  strutture di commutazione libere per eventuali altri pacchetti.

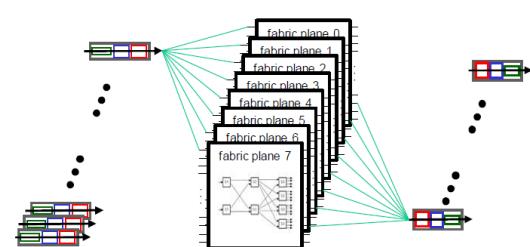
Matrice di commutazione  
(*crossbar switch*)



Commutazione a più stadi  
(*multistage switch*)



Commutazione a più livelli, scalabile  
(*multi-switching planes*)



### 5.1.5 Accodamento

L'accodamento è un fenomeno che, se non viene gestito correttamente, può portare alla saturazione totale della memoria e alla perdita di pacchetti.

#### Accodamento alle porte d'ingresso

Quando la struttura di commutazione non è sufficientemente rapida rispetto alle linee di ingresso, è proprio nelle porte d'input che si ha accodamento. I pacchetti sono serviti con un meccanismo FCFS (*First-Come-First-Served*).

Abbiamo già detto che pacchetti che devono uscire da porte differenti possono essere trasferiti in parallelo. Quando due pacchetti devono uscire dalla stessa porta, uno dei due dovrà attendere. Supponiamo di avere due porte  $A, B$  e due uscite  $X, Y$ . Supponiamo arrivino due pacchetti  $x_1, x_2$  per  $X$ , uno dalla porta  $A$ , uno dalla porta  $B$ . Se  $x_1$  va verso la porta  $X$ ,  $x_2$  andrà in coda. Un eventuale pacchetto  $y_1$  che dalla porta  $B$  vuole andare alla porta  $Y$  soffrirà di un fenomeno noto come **HOL blocking**, proprio a causa dell'accodamento di  $x_2$ .

#### Accodamento alle porte d'uscita

Supponiamo che  $R_{switch} = N \cdot R_{line}$ , ovvero che il tasso di trasferimento della struttura sia  $N$  volte più veloce della velocità di trasferimento nelle linee d'input e output.

Nel tempo richiesto per ricevere o trasmettere un pacchetto, una porta potrebbe ricevere fino a  $N$  pacchetti, causando la formazione di una coda nella porta d'uscita. Il numero di pacchetti nella coda d'uscita è direttamente proporzionale alla dimensione del buffer di quella porta. Quando il buffer è saturo e arriva un ulteriore pacchetto, si deve stabilire se effettuare un **drop-tail**, scartando il nuovo pacchetto, o rimuoverne uno o più di uno tra quelli già in coda, a favore del nuovo arrivato.

#### Quanta memoria assegnare al buffer

RFC 3439 stabiliva che la grandezza del buffer  $B$  dovesse essere uguale al round trip time medio  $RTT$  moltiplicato per la calacità del link  $C$ .

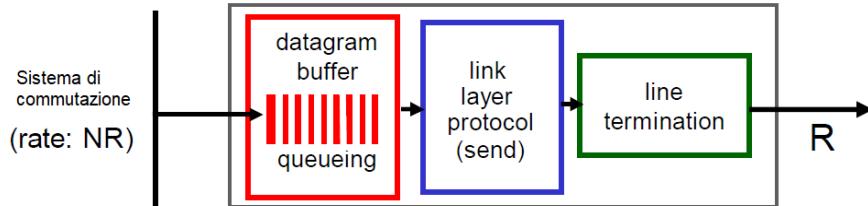
Un esempio? Un collegamento con bandwidth 10 Gbps con un RTT di 250 ms necessiterebbe di un buffer  $B$  pari a

$$B = RTT \cdot C = 10 \text{ Gbps} \cdot 250 \text{ ms} = 10 \cdot 0,25 = 2,5 \text{ Gbit}$$

Anche se recenti studi teorici, suggeriscono che, con un numero  $N$  di flussi TCP indipendenti, la quantità di buffer necessaria sia

$$B \approx RTT \cdot \frac{C}{\sqrt{N}}$$

Buffer troppo grandi aumentano i delay (soprattutto nei router di casa, che non gestiscono in maniera ottimale la coda, causando **bufferbloat**). RTT troppo lunghi rendono i mittenti TCP meno reattivi. Alcuni controlli della congestione basati sul ritardo permettono di mantenere la linea occupata, senza riempire il buffer.



### 5.1.6 Packet discarding policy

#### Tail drop

Quando la coda è piena, i pacchetti in arrivo vengono scartati a priori, senza considerare priorità o tipologia del traffico. Dropped pacchetti in questo modo potrebbe causare **burst di drop**.

#### Priority queueing

Il buffer è diviso in più code, ognuna con priorità diversa. I pacchetti vengono etichettati con un livello di priorità (o può essere stabilita dal router). In questo modo, i pacchetti ad alta priorità non saranno scartati, quando sono presenti in coda altri pacchetti a priorità più bassa.

### 5.1.7 Scheduling dei pacchetti

Come determiniamo l'ordine con cui trasmettere i pacchetti su una determinata porta d'uscita?

#### FIFO (o FCFS)

First-In-First-Out o First-Come-First-Served. Il paradigma più semplice: chi prima arriva, prima viene mandato.

#### Priority scheduling

Estende il modello FIFO-FCFS con un meccanismo di priorità: vengono definite più code per varie classi di priorità. Ad esempio, con due code (alta e bassa priorità), è possibile distribuire i pacchetti dei servizi in tempo reale (o informazioni di gestione di rete) e i pacchetti di servizi non in tempo reale. Le singole code seguono solitamente il principio FIFO, ma, date due code non vuote, verrà sempre preso il pacchetto nella coda ad alta priorità.

#### Round-Robin

Vengono di nuovo suddivisi i pacchetti in classi, non più (mandatoriamente) suddivise per priorità di servizio. A turno, ogni coda viene servita: viene trasmesso un pacchetto di classe 1, poi 2, fino alla  $n$ -esima classe. Poi di nuovo 1, 2 ...  $n$ . Col *work-conserving round robin*, quando una coda è vuota, viene del tutto saltata dalla selezione.

#### WFQ - Weighted Fair Queuing

Accodamento equo ponderato. È un Round Robin generalizzato, in cui ogni classe può ricevere un servizio differenziato. Ogni classe  $i$  avrà un peso  $w_i$ . Il WFQ assicura che l' $i$ -esima classe di pacchetto riceva sempre  $\frac{w_i}{\sum w_i}$ , dove  $\sum w_i$  indica la somma tra i pesi di tutti i servizi.

Non dimentichiamo, tuttavia, che i pacchetti sono unità discrete e che la loro trasmissione non può essere interrotta. Questo rapporto è quindi indicativo.

## 5.2 I protocolli del livello di rete

- **IP protocol.**

E la formattazione dei pacchetti, l'addressing e le convenzioni relative alla gestione dei pacchetti.

- **Path-selection algorithm.**

Che gestiscono l'instradamento dei pacchetti.

- **Protocollo ICMP.**

Dedicato agli errori e ai segnali dei router.

## 5.3 IPv4 - Internet Protocol v4

### 5.3.1 Indirizzo IP

Un indirizzo IP è un codice logico (non fisico, non 1-a-1) associato ad un host connesso a una rete che usa il protocollo Internet.

#### Classi di indirizzi IP

Un sistema che permetteva di distinguere vari tipi di indirizzi IP è quello per classi.

- Indirizzi di classe A.  
From 1.0.0.0 to 127.255.255.255. Il primo byte identifica la rete, gli altri gli host.
- Indirizzi di classe B.  
From 128.0.0.0 to 191.255.255.255. I primi due byte identificano la rete, gli ultimi due gli host.
- Indirizzi di classe C.  
From 192.0.0.0 to 223.255.255.255 I primi tre byte identificano la rete, l'ultimo gli host.
- Indirizzi di classe D (multicast).  
From 224.0.0.0 to 239.255.255.255. Sono riservati agli indirizzi multicast. Permettono quindi di trasmettere la stessa informazione a più dispositivi contemporaneamente senza dover duplicare l'informazione.
- Indirizzi di classe E (riservati).  
Riservati per usi futuri. From 240.0.0.0

#### Alcuni indirizzi specifici

Questi indirizzi sono riservati a delle funzioni specifiche.

- 0.0.0.0 - This Host, usato su un host cui indirizzo non è ancora stato specificato (ma solo in IPv4, non in IPv6, che usa un indirizzo IP basato sul MAC Address. Guarda EUI-64, pagina 81).
- 255.255.255.255 - Broadcast relativo alla propria subnet.
- 127.0.0.0 - 127.255.255.255 - Localhost, riferimento a se stesso.
- 10.0.0.0 - 10.255.255.255 - Private IP.
- 169.254.0.0 - 169.254.255.255 - Automatic Private IP Addressing (APIPA), usate per risolvere problemi relativi al DHCP.

### 5.3.2 Assegnamento degli indirizzi IP

1. Inizialmente, questo compito era affidato alla **IANA** (Internet Assigned Numbers Authority), ai tempi di Arpanet.
2. Nel 1992, sono stati creati i **RIR** (Regional Internet Registers), creati per gestire gli indirizzi e domini per ogni area assegnata
3. Nel 1998 un'associazione non-profit chiamata **ICANN** (Internet Corporation for Assigned Network and Numbers) è stata fondata per coordinare tutti i RIR del mondo.

#### I RIR attuali

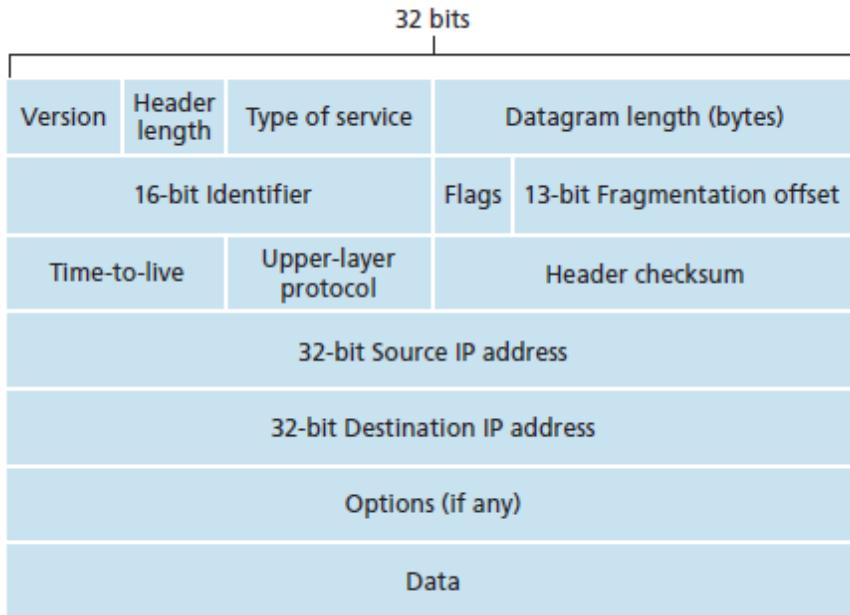
- APNIC - Asia Pacific Network Information Center
- ARIN - American Registry for Internet Numbers
- LACNIC - Latin American and Caribbean Internet Addresses Registry
- RIPE NCC - Réseaux IP Européens Network Coordination Centre
- AFRINIC - African Regional Internet Registry

I RIR sono responsabili per i Local Internet Registries, che gestiscono gli indirizzi a livello locale. Un esempio è il GARR-LIR<sup>2</sup>. I RIR forniscono vari set di indirizzi (statici e dinamici) agli ISP.

<sup>2</sup>GARR è il nome della rete italiana a banda ultralarga dedicata alla comunità dell'istruzione, della ricerca e della cultura (NREN), oltre che del consorzio che la gestisce.

### 5.3.3 Formattazione dei Datagram IP4

Questa è la struttura di un datagramma IP4. Osserviamo i singoli campi



- **Version.**

Indica la versione di IP usata dal datagramma. È fondamentale per capire come interpretare il pacchetto IP. 4 bit.

- **Internet Header Length.**

Indica la lunghezza dell'header del pacchetto IP, ovvero l'offset del campo dati, che varia da 5 word a 15 word (ovvero da 20 a 60 byte). Ciò dipende dalla presenza o meno del campo *Options*.

- **Type of Service.**

Da informazioni relative al servizio fornito, stabilendone la priorità.

- **Datagram length.**

Indica la dimensione in byte dell'intero pacchetto. Include quindi header e dati. 16 bit.

- **ID, Flags e Fragmentation Offset.**

Ne parleremo meglio dopo. Rispettivamente 16, 3 e 13 bit.

- **TTL - Time To Live.**

Specifica il tempo di vita massimo di un pacchetto. Per qualsiasi problema nella rete, un pacchetto potrebbe persistere nella vita per un quantitativo spropositato di tempo. Potrebbe essere addirittura una cosa voluta per un possibile attacco hacker. Il time to live ovvia a questo problema definendo un numero di salti, nel momento in cui il pacchetto viene mandato in rete: ogni salto attraverso un router implica un decremento. Usa 8 bit.

- **Protocol.**

Specifica il protocollo del transport layer. 4 bit.

- **Header checksum.**

8 bit dedicati alla somma di controllo. Un pacchetto cui header risulta corrotto, viene scartato. Le operazioni sono di natura logica o matematica, ed il controllo è fatto a ogni router.

- **Options.**

Quasi inutilizzate.

- **Source IP e Destination IP.**

32 bit ciascuno, 64 in totale. Sono due campi separati.

### 5.3.4 Un problema: la frammentazione

Alcuni campi che non abbiamo ancora affrontato, sono quelli relativi alla frammentazione. Quando un datagramma IP viene frammentato in più datagrammi IP più piccoli, queste informazioni permettono la ricostruzione del frammento originale a destinazione. La frammentazione avviene quando un router ha MTU (Max Transfer Size) tale da non consentire il trasferimento dell'intero datagramma IP.

- **ID.**

16 bit, identificano un insieme di pacchetti IP che (presumibilmente) ricostruiranno un pacchetto originale.

- **Flags.**

3 bit, 2 usati. I flags sono DF – **Don't Fragment**, = 1 "Non frammentare mai. Al massimo, scarta." MF – **More Fragments**, = 1 "Questo non è l'ultimo frammento".

- **Offset.**

13 bit, indica dove inizia il frammento rispetto all'inizio del pacchetto originale, espresso in unità da 8 bit. Permette di ricostruire in maniera opportuna i pacchetti frammentati.

Questo compito non spetta solitamente ai router, i quali ricordiamo occuparsi principalmente di forwarding. Una corruzione dei dati di frammentazione può avere gravi conseguenze in fase di ricostruzione.

- **Offset alterato** - il ricongiungimento dei pezzi non può essere eseguito correttamente.

- **ID alterato** - un frammento potrebbe cambiare pacchetto di appartenenza.

- **MF=1 → MF=0** - un frammento che non è l'ultimo potrebbe essere considerato l'ultimo.

#### Esempio di frammentazione

Dato un pacchetto con 4000 byte e un router con MTU di 1500 byte (come Ethernet).

- Se il frammento arriva con il flag DF = 1, viene scartato
- Con DF = 0 si procede alla frammentazione.

Vediamo come procederebbe la frammentazione.

1. Il primo frammento viene mandato con MF = 1 con offset a 0. Il segmento mandato è di 1480 byte, in quanto 20 byte saranno occupati dall'header TCP.  $4000 - 1480 = 2520$  byte da mandare rimanenti.
2. Il secondo frammento viene mandato con MF = 1 con offset a  $\frac{1480}{4} = 370$ .  $2520 - 1480 = 1040$  byte da mandare rimanenti.
3. Il terzo frammento viene mandato con MF = 0 ( $1040 - 1480 < 0$ ) con offset a  $\frac{1480}{2} = 740$ .

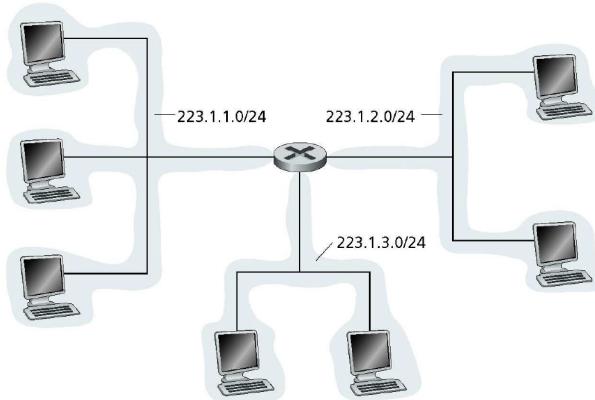
### 5.3.5 Indirizzamento IPv4

Abbiamo detto che un indirizzo IP identifica univocamente un dispositivo. Ciò è improprio: gli indirizzi IP identificano un'interfaccia di rete. Se i computer hanno una sola interfaccia di rete, e quindi un solo indirizzo IP, i router presentano  $n$  interfacce, e quindi  $n$  indirizzi IP. Ogni indirizzo IP identifica una subnet. Gli indirizzi IPv4 hanno lunghezza di 32 bit. Avremo quindi  $2^{32}$  indirizzi.

Gli indirizzi IP usano la seguente notazione : 193.32.216.9, detta notazione decimale puntata. Ogni interfaccia di host e router di Internet ha un indirizzo IP globalmente univoco (fatta a eccezione di quelle gestite da NAT, di cui parleremo più avanti).

#### Come vengono determinati gli indirizzi IP

Gli indirizzi IP non sono determinati in maniera arbitraria. Parte dell'indirizzo IP di una determinata interfaccia dipende dalla sottorete a cui è collegata. Questa struttura non arbitraria è la chiave dietro al funzionamento degli indirizzi IP.



L'interfaccia di rete di un router e i dispositivi a essa collegata (magari tramite uno switch o da un punto di accesso Wireless) costituiscono una sottorete [RFC 950]. Ad esempio, IP assegna ad una sottorete l'indirizzo 223.1.1.0/24. La notazione /24 è chiamata maschera di rete, ed indica che i 24 bit più a sinistra dell'indirizzo definiscono l'indirizzo della sottorete. Questo vuol dire che tutti i dispositivi partecipanti a questa rete avranno indirizzo IP nella forma 223.1.1.XXX<sup>3</sup>.

#### Regola generale per individuare sottoreti

*Per determinare le sottoreti, si sgancino le interfacce da host a router in maniera tale da creare isole di reti isolate delimitate dalle interfacce. Ognuna di queste reti isolate viene detta sottorete.*

<sup>3</sup>Con XXX nel range [2 – 254], escludendo 0 in quanto *forbidden*, 1 riservato ai router e 255 per il broadcast

### 5.3.6 CIDR - Classless InterDomain Routing

CIDR [RFC 4623] (pronunciato *Cider*) generalizza l'indirizzamento di sottorete nel seguente modo:

- L'indirizzo IP è nella forma  $a.b.c.d/x$
- $x$  indica il numero di bit della prima parte dell'indirizzo.
- La prima porzione è detta porzione di rete, o prefisso, dell'indirizzo IP.
- La seconda porzione (di  $32-x$  bit) distingue i dispositivi all'interno della stessa rete, o delle ulteriori sottoreti.

*Un esempio? La rete denotata da a.b.c.d/21 contiene anche la sottorete denotata da a.b.c.d/24, all'interno della stessa organizzazione.*

Questo metodo è stato introdotto dopo l'utilizzo del classful addressing, che segue la suddivisione in classi che abbiamo introdotto precedentemente.

Detto ciò, la suddivisione meno rigida del CIDR ha permesso alle aziende di scegliere un valore di  $x$  che minimizzi lo spreco di indirizzi IP, ottenendone invece un quantitativo che rispecchi quanto possibile le esigenze del sistema.

#### Da una domanda su stack overflow

Class based addressing basically divides the total IPv4 range into five classes:  
Class A through E; Whereas CIDR is based on concept of subnetting.

Your company wants 32 public ip addresses.  
If we assign them a class C address, for example 192.168.2.0, their company reserves all IP addresses in range 192.168.2.1 - 192.168.2.254.

But they only want 32 ip addresses which means 223 addresses will be wasted.  
This is the constraint of classful addressing. Now if we look at just subnetting, class c ip address has default subnet of 255.255.255.0 so if we divide the range of 192.168.2.0 in 6 subnets each containing 32 available ip addresses your problem is solved.

But, if we take this example to higher level we will require CIDR.  
According to traditional subnetting, we can not combine the addresses from the networks 192.168.2.0 and 192.168.3.0 because the netmask for class C addresses is 255.255.255.0. However, using CIDR notation, we can combine these blocks by referencing this chunk as 192.168.2.0/23.

This specifies that there are 23 bits used for the network portion that we are referring to.  
With this, the 24th bit can be either 0 or 1 and it will still match, because the network block only cares about the first 23 digits.

CIDR allows more control over addressing continuous blocks of IP addresses.  
This is much more useful than the subnetting we talked about originally.

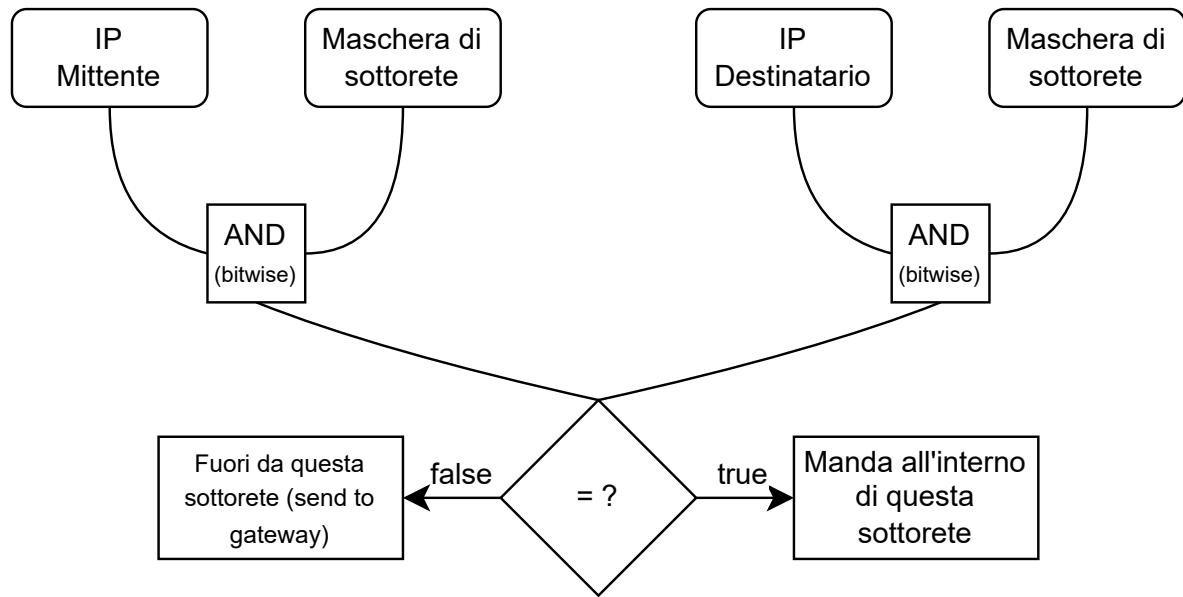
#### IP broadcast

255.255.255.255. Quando un host emette un datagramma con tale IP destinazione, il messaggio viene consegnato a tutti gli host sulla stessa sottorete.

### 5.3.7 Come vengono usate le maschere di rete

La maschera di rete, indicata in notazione CIDR con  $/x$  indica che i primi  $x$  bit a sinistra indicano la (sotto)rete. Nella maschera di rete, i primi  $x$  bit a sinistra saranno 1, gli altri  $32 - x$  a 0. Alcuni esempi di maschere di rete:

- Con  $/19$  la maschera sarà:  $255.255.224.0 - 11111111111111111000000000000000$
- Con  $/24$  la maschera sarà:  $255.255.255.0 - 11111111111111111111111110000000$
- Con  $/16$  la maschera sarà:  $255.255.0.0 - 11111111111111111000000000000000$



La maschera di rete può essere usata per estrapolare e confrontare l'indirizzo della sottorete di appartenenza di due host.

## 5.4 Elementi del livello di collegamento e ARP

Studiando il livello di collegamento, scopriremo che gli host e i router hanno un indirizzo sia a livello di rete (IP) che a livello di collegamento (indirizzo MAC). Il protocollo ARP (*address resolution protocol*) permette ai nodi l'associazione indirizzo IP → indirizzo MAC.

### 5.4.1 Cos'è un indirizzo MAC?

Un indirizzo *Media Access Control* è un codice a 6 byte che identifica univocamente una scheda di rete. Gli indirizzi MAC sono univoci: la IEEE si occupa di distribuire proprio questi indirizzi MAC alle varie aziende. I primi 3 byte sono stabiliti dall'IEEE, (viene scelta quindi una combinazione di bit su  $2^{24}$ ). Gli ultimi tre vengono dati univocamente ad ogni dispositivo prodotto dall'azienda stessa (sempre  $2^{24} = 16.777.216$ ).

### 5.4.2 Il protocollo ARP

Sta per *Address Resolution Protocol* [RFC 826], e traduce gli indirizzi IP in MAC address. Traduce gli indirizzi IP di dispositivi situati nella stessa sottorete. Convenzionalmente, gli indirizzi IP sono scritti in decimale, gli indirizzi MAC in esadecimale.

#### Come funziona?

Ogni nodo contiene una tabella ARP cui record contengono tre valori:

Indirizzo IP - Indirizzo MAC - Time To Live.

Il time-to-live specifica tra quanto tempo quel determinato record dovrà essere eliminato dalla tabella. Un tempo tipico sono 20 minuti.

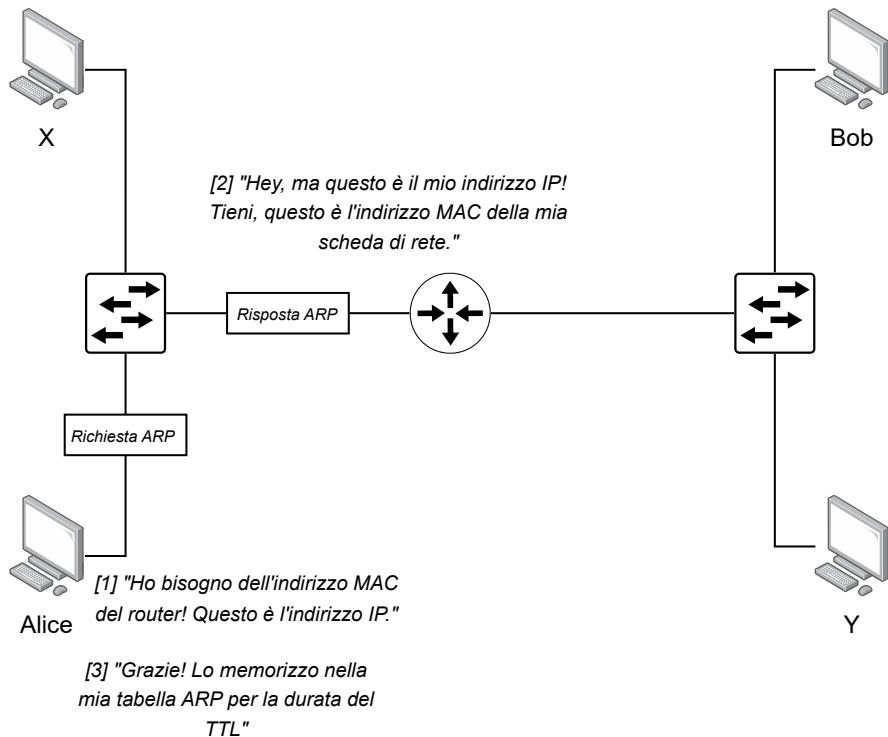
#### Caso 1 - Invio sulla stessa (sotto)rete

1. Se un datagramma deve essere mandato da un host *A* a un host *B* nella stessa sottorete, *A* dovrà conoscere l'indirizzo MAC di quel dispositivo.
2. Se l'indirizzo MAC è già presente nella ARP del nodo, questo manderà il datagramma senza ulteriori passaggi.
3. Altrimenti, manderà una cosiddetta "richiesta ARP" contenuta in un pacchetto ARP mandato all'indirizzo broadcast della rete FF-FF-FF-FF-FF-FF.
4. Il pacchetto conterrà sia l'indirizzo IP e MAC del mittente, che l'indirizzo IP del destinatario.
5. La richiesta broadcast verrà mandata a tutte le schede di rete della sottorete. La risposta ARP verrà mandata in modalità non-broadcast dal dispositivo con indirizzo IP uguale a quello della richiesta, verso il richiedente. La risposta ARP conterrà al suo interno l'indirizzo MAC del ricevente della richiesta, e la tabella ARP verrà aggiornata.

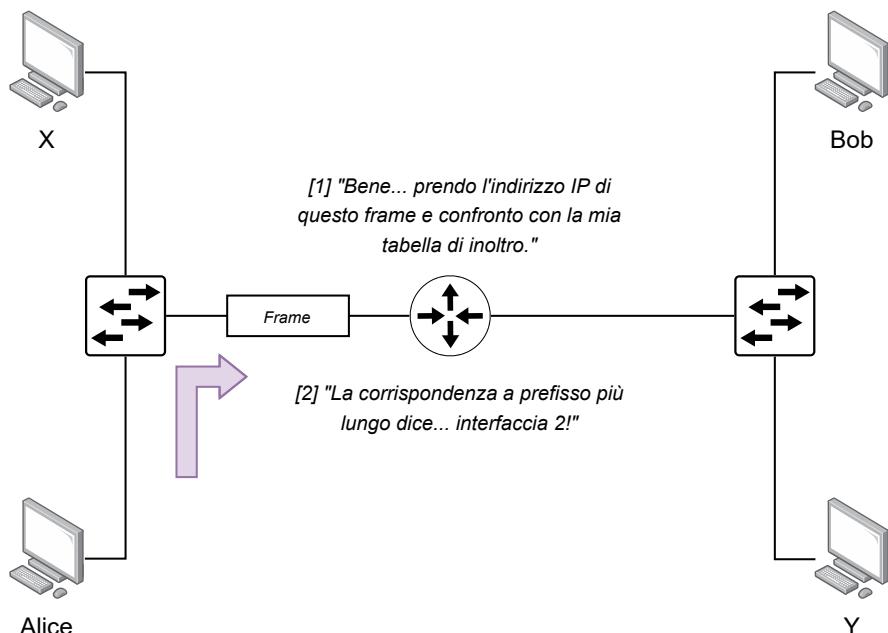
## Caso 2 - Invio su (sotto)rete diversa

Alice desidera mandare un messaggio a Bob, collegato ad una sottorete diversa.

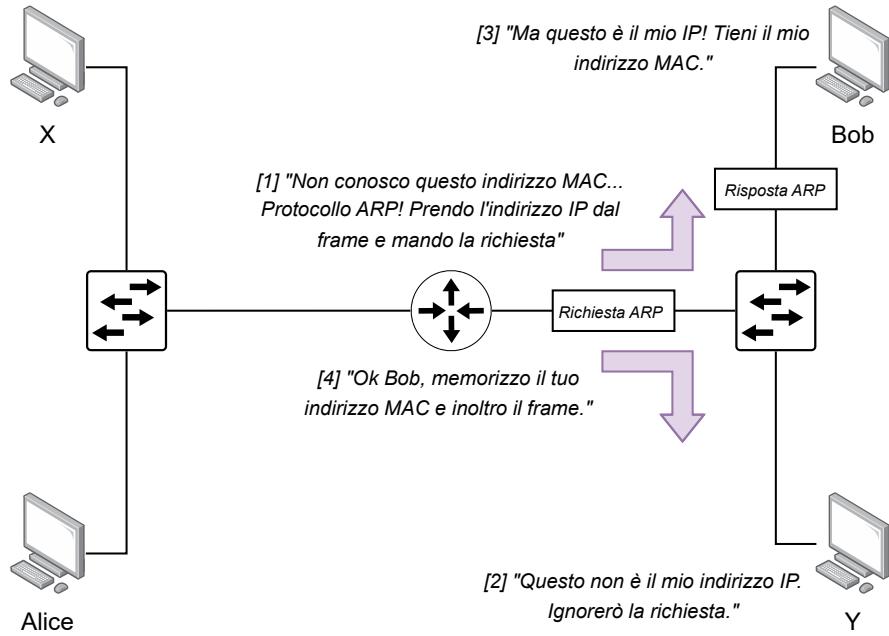
1. Alice manda il datagramma alla sua scheda di rete. Dovrà indicare l'indirizzo MAC dell'interfaccia del router a cui è collegato. Se l'indirizzo MAC è già presente nella tabella ARP di Alice, questo manderà il datagramma al router.
2. Altrimenti, userà il protocollo ARP per ottenere l'indirizzo MAC. Manderà una richiesta ARP all'indirizzo broadcast della rete FF-FF-FF-FF-FF-FF, contenente l'indirizzo IP del router. Il router, riconoscendo il proprio indirizzo, risponderà con una ARP reply condividendo ad Alice il proprio indirizzo MAC. Alice memorizzerà il nuovo indirizzo MAC appena scoperto nella propria tabella ARP, e manda il proprio frame.



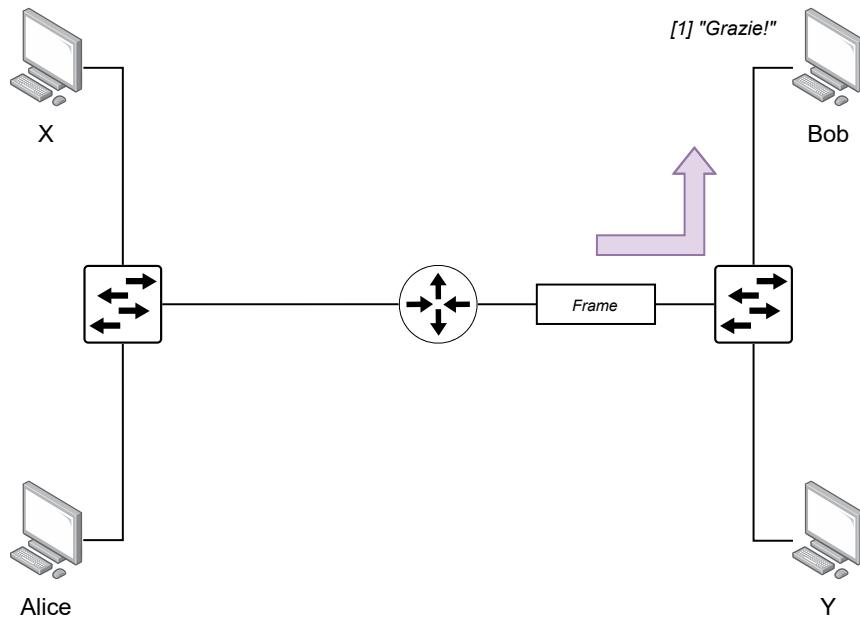
3. Il router confronta l'IP del pacchetto con la propria tabella di inoltro. Sa che il pacchetto deve essere inoltrato all'interfaccia 2, ma non conosce l'indirizzo MAC del destinatario Bob.



4. Il router manda la richiesta ARP, lo switch la manda in broadcast a tutti i dispositivi connessi. Bob riconosce l'indirizzo IP della richiesta ARP, e manda una risposta ARP. Il router riceverà l'indirizzo MAC di Bob, lo conserverà nella propria ARP table assieme al suo IP.



5. Il router avrà tutto il necessario per inoltrare il pacchetto a Bob, ed è ciò che farà.



Notiamo che, dopo aver ottenuto tutti gli indirizzi necessari, tramite protocollo ARP, per tutta la durata delle *entries* nella tabella ARP, l'inoltro sarà presumibilmente molto più veloce.

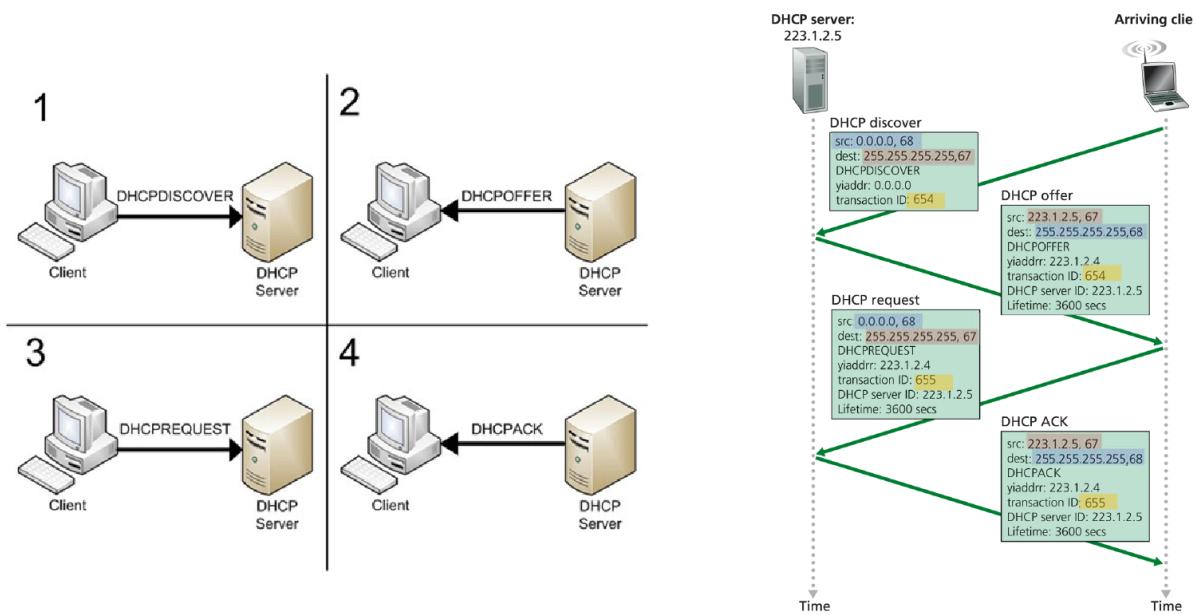
## 5.5 Ottenere gli indirizzi IP

Per ottenere un blocco di indirizzi IP da usare in una sottorete, bisogna rivolgersi a Internet Service Provider (ISP). Quest'ultimo attinge da un suo blocco di indirizzi IP già allocati, e ne riserva un numero inferiore per l'ente richiedente. La dimensione della maschera di rete stabilisce la dimensione del blocco allocato.

L'ente che ha acquisito gli indirizzi potrà gestirseli autonomamente, costruendo le proprie sottoreti. Ogni ISP richiede uno o più blocchi di indirizzi IP all'ICANN (Internet Corporation for Assigned Names and Numbers). L'Address Support Organization si occupa dell'allocazione e della gestione degli indirizzi all'interno delle rispettive regioni, e quindi dei RIR.

## 5.6 DHCP - Dynamic Host Configuration Protocol

Ottenuto un blocco di indirizzi, li si può assegnare alle interfacce dei router e degli host della propria struttura. La configurazione può avvenire manualmente, o sfruttando il **Dynamic Host Configuration Protocol** (DHCP) [RFC 2131]. Dipendentemente dalla configurazione del DHCP, gli host possono ricevere indirizzi IP persistenti e uguali a ogni accesso, o uno temporaneo.



### 5.6.1 DHCP - Un protocollo plug-and-play

Sarebbe impensabile gestire una rete (ad esempio) di un'università o di una scuola: impostare manualmente ogni dispositivo in una struttura con un alto via-vai rappresenterebbe sicuramente una criticità. DHCP imposta automaticamente un indirizzo IP per dispositivi della stessa sotto-rete.

### 5.6.2 Come funziona DHCP

È un protocollo client-server, in cui i client sono degli host appena connessi alla rete, richiedenti un nuovo indirizzo IP. Il server in ascolto è proprio un server DHCP, che può essere presente nella sottorete, o un relay DHCP all'interno del router che delega il lavoro ad un DHCP esterno.

La configurazione di un nuovo host avviene in 4 passaggi:

#### 1. Individuazione del server DHCP.

SOURCE: IP 0.0.0.0, PORT 68 - DESTINATION: IP 255.255.255.255 PORT: 67.

Il client manda un messaggio di tipo **DHCP discover**. Il messaggio è encapsulato in un pacchetto UDP mandato alla porta 67, e in un datagramma IP mandato in broadcasting, e quindi all'IP 255.255.255.255. Questo messaggio proviene dalla porta 68, e ha come IP del mittente l'ip 0.0.0.0.

#### 2. Offerta del server DHCP.

SOURCE: IP dhcp\_server\_ip, PORT 67 - DESTINATION: IP 255.255.255.255 PORT: 68.

Ricevuto il discover, uno o più server DHCP manderanno un broadcast **DHCP offer**.

Se il client riceve più offerte, potrà scegliere quello che più preferisce. L'offerta DHCP contiene l'indirizzo IP offerto, l'id transazione, l'IP del server THCP e il **lifetime** dell'IP offerto.

#### 3. Richiesta DHCP.

SOURCE: IP 0.0.0.0, PORT 68 - DESTINATION: IP 255.255.255.255 PORT: 67.

Il client risponde con un messaggio **DHCP request**, che riporta i parametri di configurazione.

#### 4. Conferma DHCP.

SOURCE: IP dhcp\_server\_ip, PORT 67 - DESTINATION: IP 255.255.255.255 PORT: 68.

Conferma da parte del server dei parametri richiesti con un **DHCP ACK**.

## 5.7 NAT - Network Address Translation

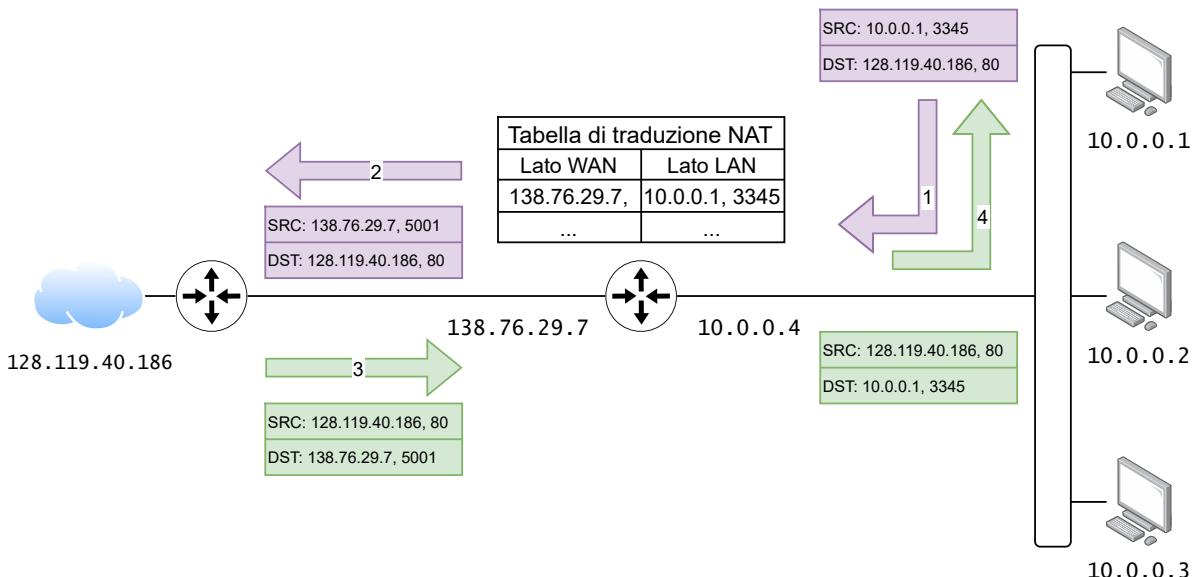
### 5.7.1 Problema da risolvere

Ogni dispositivo ha bisogno di un indirizzo IP. Ne consegue che, al crescere del numero dei dispositivi da connettere ad una rete LAN, (apparentemente) sarà necessario avere un blocco di indirizzi di dimensione crescente. Questo approccio porta problemi: cosa si fa se necessito dall'ISP degli IP contigui che sono stati già allocati?

### 5.7.2 Soluzione, NAT!

Nella rete si usa un router abilitato al NAT. A tutti i dispositivi della rete, verranno assegnati indirizzi di spazi di indirizzamento dedicati alle reti private, come lo spazio 10.0.0.0/8. Nell'esempio in questione, le nostre interfacce di rete avranno indirizzi dello spazio 10.0.0.0/24. Questi indirizzi non saranno visibili all'esterno, e quindi potranno essere riutilizzati in tutte le possibili reti private separate da router abilitati al NAT.

Il NAT ha il compito di tradurre gli indirizzi IP della propria sotto-rete sfruttando una NAT table, in cui i record sono del tipo



1. Il router riceve i pacchetti da mandare all'esterno. Il NAT controlla se la coppia IP e porta è già presente nella tabella. Se non è presente, il NAT inserisce nella sua tabella un nuovo record, che servirà per le traduzioni. Nel lato LAN l'ip e la porta dell'interfaccia effettiva, nel lato WAN l'IP dell'interfaccia usata dal router lato WAN, e una porta generata casualmente tra quelle non usate dal quell'IP. Il NAT può gestire ben oltre 60.000 connessioni simultane con una sola interfaccia (e indirizzo IP) lato WAN!
2. Il NAT controlla la propria tabella, e sostituisce IP e la porta **SOURCE** del pacchetto, con quelli associati lato WAN. Inoltre regolarmente il pacchetto.
3. Quando riceverà una risposta, i pacchetti arriveranno all'IP e alla porta che il NAT ha precedentemente assegnato.
4. Il NAT stesso si occuperà di tradurre, dal verso opposto, la coppia IP-porta lato WAN, a quella lato LAN, inoltrando regolarmente il pacchetto.

Il NAT non è assolutamente un protocollo perfetto. Implica una violazione del principio end-to-end, a causa della modifica dell'indirizzo IP e del numero di porta. Inoltre, causa problemi con le connessioni P2P e gli applicativi non NAT-friendly.

## 5.8 IPv6

Il protocollo IPv6 nasce dall'esigenza di avere uno spazio di indirizzamento più ampio (tamponare il problema col protocollo NAT non sarebbe stato ottimale e sufficiente a lungo termine), migliorare e semplificare il protocollo IPv4. Imparando dal suo predecessore, è stato possibile creare un protocollo:

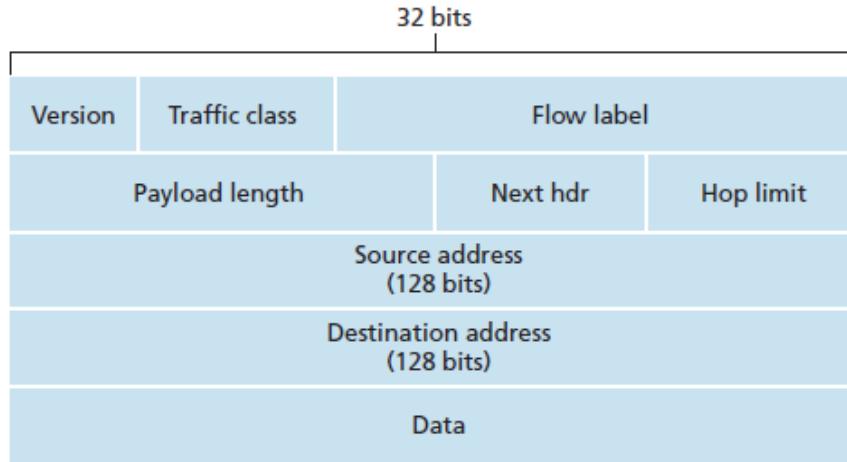
- Con header più semplici.
- Sicuro, tramite un supporto nativo per la sicurezza (IPsec<sup>4</sup>).
- Con Quality of Service migliorata.
- Supporto a comunicazione multicast e anycast.

### 5.8.1 Parentesi storica

Nei primi anni '90, si iniziò a progettare il successore di IPv4. Le motivazioni erano chiare: avere 32 bit per ogni indirizzo IP stava diventando limitante, la richiesta di indirizzi IP stava diventando sempre maggiore e meno soddisfacibile. I tecnici della **Internet Engineering Task Force** (IETF) hanno quindi colto l'occasione per creare un nuovo protocollo, migliore e con un margine di indirizzi IP molto più vasto.

IPv4 addresses Available in October 17, 2010 according to the RIPE															Mohammad Mahloujian October 18, 2010			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31			
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47			
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63			
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79			
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95			
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111			
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127			
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143			
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159			
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175			
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191			
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207			
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223			
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239			
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255			
<b>xx</b> Used		<b>xx</b> Allocated in Jan. 2010		<b>xx</b> Allocated in May 2010		<b>xx</b> Allocated in Oct. 2010										Total	256	
<b>xx</b> Available		<b>xx</b> Allocated in Feb. 2010		<b>xx</b> Allocated in June 2010		<b>xx</b> Allocated in Apr. 2010		<b>xx</b> Allocated in Aug. 2010								Total	256	
<b>xx</b> Not useable																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	
																Total	256	

### 5.8.3 Struttura dei datagrammi IPv6



- **Version - Versione.**

Indica la versione di IP usata dal datagramma. 4 bit.

- **Traffic Class - Classe di traffico.**

Somiglia al Type of Service di IPv4. Specifica la priorità di un determinato datagramma all'interno di un flusso. VoIP manderà datagrammi di priorità più alta rispetto a SMTP. 8 bit.

- **Flow label - Etichetta di flusso.**

Identifica un flusso di datagrammi. 20 bit.

- **Payload length - Lunghezza del Payload.**

Interpretato come un unsigned int, indica il numero di byte del datagramma IPv6 a seguire dell'header di dimensione fissa. Il massimo di dati che è possibile trasportare sul payload, è di  $2^{16} - 1 = 65,535$ . 16 bit.

- **Next header - Prossimo header.**

Specifica il tipo del prossimo header. Questo, può essere l'header del protocollo del livello di trasporto (TCP: 6, hex 0x06, UDP: 17, hex 0x11), o un header aggiuntivo (*extension header*). Ognuno di questi header aggiuntivi, contiene al suo interno anche il campo *next header*, in modo da poter costruire, su un datagramma, una "catena di header". In questa catena, l'ultimo header aggiuntivo contiene al suo interno il tipo dell'header del livello di trasporto. 8 bit. Tra questi header aggiuntivi, abbiamo, ad esempio **Authentication Header (AH)**: 51 e **Encapsulating Security Payload (ESP)**: 60, entrambi meccanismi di **IPsec**.

- **Hop limit - Limite di Hop.**

Un numero che viene decrementato ad ogni inoltro, da parte di router, del datagramma che lo contiene. Quando questo numero diventa 0, viene cestinato. 8 bit.

- **Indirizzi.**

Sorgente e destinazione, entrambi a 128 bit.

- **Dati.**

## 5.8.4 Funzionalità di IPv6

Sintetizziamo in breve gli aspetti *game changer* del protocollo IPv6.

- **Indirizzamento esteso.**

Aumenta la dimensione degli indirizzi.  $2^{128}$  indirizzi possibili, praticamente inesauribili.

- **Unicast, Multicast e Anycast.**

Supporta indirizzi unicast, multicast ed anycast. Gli indirizzi multicast e anycast identificano insiemi di interfacce. Mandando un pacchetto a un'indirizzo multicast, questo verrà mandato a tutte le interfacce. Mandando un pacchetto a un'indirizzo anycast, questo verrà all'interfaccia più vicina (o una qualunque).

- **Etichettatura dei flussi.**

Etichettando in un determinato modo i pacchetti di un determinato flusso, diventa possibile dare priorità e offrire delle garanzie di servizio migliori rispetto a determinati protocolli dell'application layer.

- **Rimossa frammentazione ai router intermedi.**

- **Sfoltiti gli header.**

## 5.8.5 Sugli indirizzi IPv6

Per convenzione, i molto più lunghi indirizzi IPv6, rispetto agli IPv4, sono scritti in esadecimale. Otto gruppi da quattro cifre, separati da due punti.

esempio di indirizzo IPv6:

8000:0000:0000:000:0123:4567:89AB:CDEF

o, equivalente:

8000::123:4567:89AB:CDEF

un indirizzo IPv4 mappato in IPv6:

::151.97.1.1

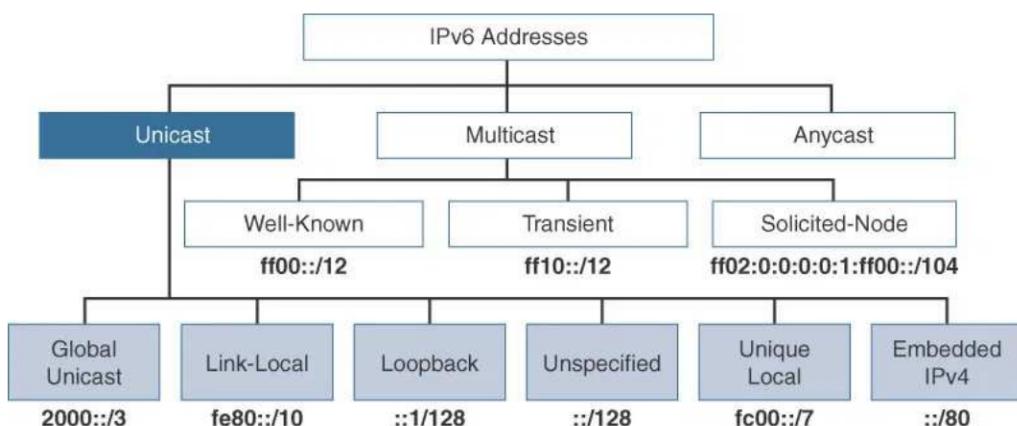
indirizzo di loopback:

::1

non specificato:

::

### Assegnazione indirizzi IPv6



### 5.8.6 EUI-64 - Extended Unique Identifier

L'assegnamento di un indirizzo IPv6 ad un'interfaccia di rete avviene tramite DHPC. Tuttavia, prima dell'assegnamento<sup>5</sup>, questo può essere automaticamente derivato dal suo MAC address a 48 bit, seguendo il metodo **EUI-64**. Utile in configurazioni senza DHCPv6.

1. Si prende l'indirizzo MAC.
2. Si divide in due metà (24 bit ciascuna).
3. Si inserisce in mezzo FF:FE. (In Windows, vengono inserite delle cifre casuali).
4. Si cambia il settimo bit (universal/local) del primo byte in 1 se è un indirizzo localmente amministrato, a 0 se è globalmente unico.

Questi indirizzi funzionano solo all'interno di una LAN.

### 5.8.7 Neighbour Discovery Protocol (il nuovo arp!)

È il protocollo simil-ARP di IPv6. Migliore di ARP, usa ICMPv6, e sfrutta due tipi di messaggi:

- **Neighbour Solicitation (NS).**  
Equivalenti di un ARP request. Richiede l'indirizzo MAC di un dispositivo con un determinato indirizzo IPv6.
- **Neighbour Advertisement (NA).**  
È l'equivalente dell'ARP reply. Può essere anche spontanea senza previa NS, per avvisare i dispositivi di cambiamenti del MAC address.

Questo protocollo permette anche di **autoconfigurare gli host**, e supporta **Router Advertisement**.

### 5.8.8 IPv6 over Ethernet

Come IPv4, anche i datagrammi IPv6 viaggiano dentro frame Ethernet. Il pacchetto Ethernet, in questo caso, conterrà nel campo Ethertype il codice 0x86DD, e non 0x0800 di IPv4.

### 5.8.9 Da IPv4 a IPv6

L'argomento transizione da IPv4 a IPv6 è un argomento **delicato**: l'infrastruttura di Internet è stata costruita su IPv4, ma in pochi avrebbero potuto prevedere un tale successo. IPv6 è il sostituto definitivo di IPv4, ma il processo di transizione sarà lento e durerà ancora anni. Non potrà mai essere rapido com'è stato, ad esempio, quello dal protocollo NCP (protocollo di trasporto risalente alla rete ARPANET) a quello TCP.

#### Dual Stack

I dispositivi (host o router) supportano entrambi i protocolli, IPv4 e IPv6. Quello scelto dalle applicazioni dipenderà dal tipo di connessione (con una preferenza verso IPv6). Questo aumenta la complessità di gestione del nodo, a causa del doppio IP, ma è la soluzione più semplice e stabile.

#### Tunneling

Se i due end-point di una comunicazione supportano IPv6, ma parte della rete che devono attraversare i messaggi no, ciò che si farà, sarà incapsulare l'intero datagramma IPv6 in datagrammi IPv4 prima di essere inoltrato all'interno della rete che supporta IPv4.

---

<sup>5</sup>Invece di usare l'indirizzo 0.0.0.0

## 5.9 ICMP - Internet Control Message Protocol

È un protocollo usato per la trasmissione di informazioni di controllo nella rete. Viene in IP ed è utilizzato per strumenti vari, tra cui ping e traceroute. ICMP esiste sia per IPv4 che per IPv6, e in IPv6 trova importante applicazione per i messaggi del **Neighbor Discovery Protocol**. ICMP include i campi:

- **Tipo.**

Specifica il formato dei messaggi. 8 bit.

- **Codice.**

Identifica il messaggio. 8 bit.

- **Checksum.**

16 bit.

- **Dati.**

La lunghezza di questo campo cambia in base al campo "tipo" e "codice".

## 5.10 Firewall

Sono delle **strutture software, hardware o ibride** cui compito è **bloccare e/o filtrare il traffico malevolo**, in base ai valori relativi ai campi degli header, o effettuano un **reindirizzamento** per un'ulteriore elaborazione. **Gestire tutto ciò che costituisce una minaccia per la sicurezza**, e che passa attraverso la rete, è il **compito principale dei firewall**, e per adempiere a questo scopo, sono definite delle regole di filtraggio intelligenti, basate su:

- Indirizzi IP e porte di origine e destinazione.
- Protocolli (TCP, UDP, ICMP...).
- Stato della connessione.
- **Contenuto del pacchetto**, in firewall con ispezione profonda.

I firewall posso operare a livello di rete, di trasporto e/o al livello applicativo, a seconda delle esigenze di controllo.

### 5.10.1 DMZ - Demilitarized Zone

Tra una rete locale e una WAN, solitamente, si crea una terza sottorete, chiamata **Demilitarized Zone**, o **DMZ**. Questa zona contiene sistemi isolati dalla rete interna, ma raggiungibili dall'esterno, una sorta di *façade* per l'esterno. È qui che i firewall possono:

- **Allow**, far passare un pacchetto.
- **Deny**, bloccare e rispedire un pacchetto al mittente.
- **Drop**, bloccare un pacchetto senza notificarne il mittente. Permette di risparmiare banda.

Un firewall può implementare il NAT, per mascherare gli IP interni alla rete esterna.

# Capitolo 6

# Network Layer - Piano di controllo

## 6.1 Algoritmi di Instradamento (routing)

Lo scopo di questi algoritmi è **determinare i migliori cammini** da usare dalle sorgenti ai destinatari. Il miglior cammino è valutato rispetto a vari fattori, che riprendono ciò che abbiamo affrontato nello studio dei problemi di ottimizzazione sui grafi, nella materia Algoritmi e Laboratorio. Vedi il mio GitHub per approfondimenti.

- *Shortest path*, o *shortest unweighted path*.

Il cammino, da nodo  $A$  a nodo  $B$  che passa attraverso il **minor numero di nodi**.

- *Shortest weighted path*, o *least-cost path*.

Il cammino  $p = \langle (v_1, v_2), (v_2, v_3) \dots, (v_{n-1}, v_n) \rangle$  tale che  $A = v_1$ ,  $B = v_n$  e

$$\min \sum_{i=1}^n c(v_{i-1}, v_i)$$

ovvero, tale che la **somma dei pesi** (o costi) del cammino da  $A$  a  $B$  sia **minima**.

Nonostante ciò, esistono molti altri fattori da **tenere in considerazione**, come **policy** di inoltro.

### 6.1.1 Centralizzati, non centralizzati

- Un **algoritmo di instradamento centralizzato**, calcola il percorso migliore avendo una conoscenza globale della rete. Ciò implica l'esistenza di un modo per conoscere i costi e la topologia relativi ad una rete. Il calcolo dei percorsi potrà poi essere effettuato da una struttura centralizzata e condivisa ai router, o replicato in ogni router.

Un esempio di algoritmo a informazioni globali (centralizzato), è **l'algoritmo link-state**.

- Un **algoritmo di instradamento decentralizzato**, effettua lo stesso calcolo in maniera distribuita e iterativa. Nessun nodo possiede informazioni di tipo globale relative alla rete, ma solo quelle relative ai collegamenti adiacenti. Lo scambio di informazioni permette gradualmente di valutare il percorso a distanza minima. L'algoritmo decentralizzato che tratteremo sfrutta un vettore, chiamato **vettore delle distanze**, presente in ogni router, e che contiene al suo interno le stime relative alla lunghezza dei cammini minimi. È detto algoritmo **distance-vector**.

### 6.1.2 Algoritmi statici e dinamici

- Negli **algoritmi di instradamento statici**, raramente i percorsi stabiliti cambiano. Solitamente ciò avviene sotto intervento umano.
- Negli **algoritmi di instradamento dinamici**, gli instradamenti variano in funzione di fattori quali volume del traffico, topologia della rete. L'algoritmo viene eseguito periodicamente o in funzione di variazioni nella topologia o di costi relativi ai collegamenti.

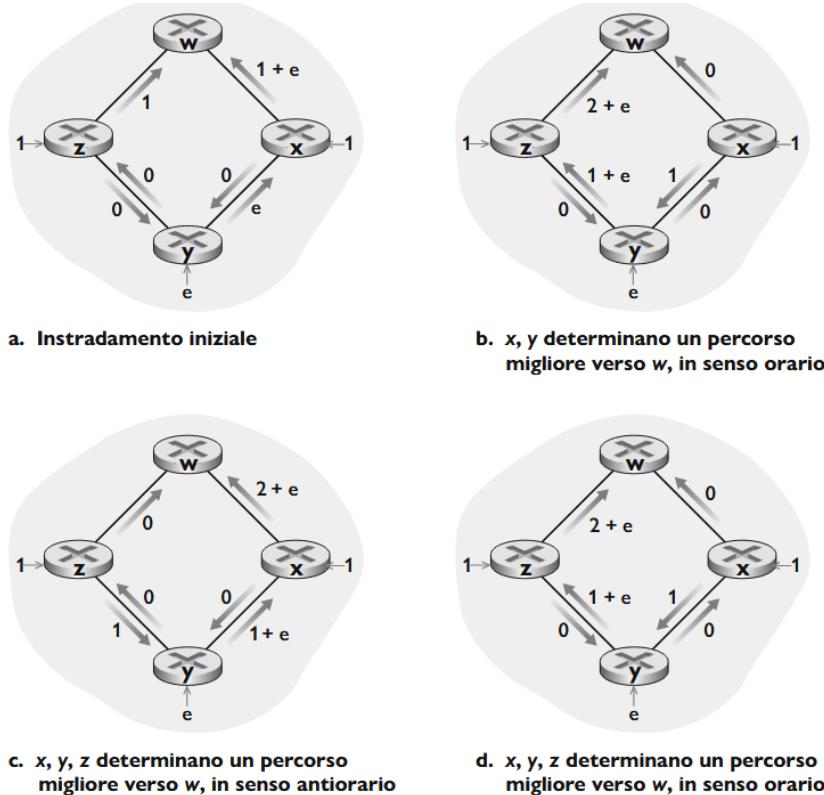
### 6.1.3 Algoritmi sensibili o insensibili al carico

- Gli **algoritmi sensibili al carico** usano il tasso di congestione dei collegamenti come criterio per valutarne il costo. Rispondono meglio ai cambiamenti della rete, ma sono molto sensibili a instradamento in loop e oscillazione dei percorsi.
- Gli **algoritmi insensibili al carico** non valutano gli stessi fattori.

Un'intuizione iniziale potrebbe farci preferire i primi ai secondi. Tuttavia, il costo di un collegamento calcolato in un momento di congestione, non esprime quello che dovrebbe essere il suo costo in condizioni normali. La congestione di rete potrebbe non durare abbastanza per rendere valida la scelta di un cammino migliore in quell'istante di tempo. Per Internet, si è preferito sfruttare algoritmi di instradamento insensibili al carico.

#### Problema delle oscillazioni

Negli algoritmi sensibili al carico appena accennati, si rischia lo scenario delle **oscillazioni dei percorsi**. Avviene quando il costo del collegamento è **pari al traffico trasportato**.



Questo scenario, causa cicli di ricalcolo continui che compromettono la stabilità dell'instradamento. La soluzione generale, consiste nell'**evitare che tutti i router eseguano l'algoritmo di routing allo stesso istante**.

## 6.2 Instradamento link-state

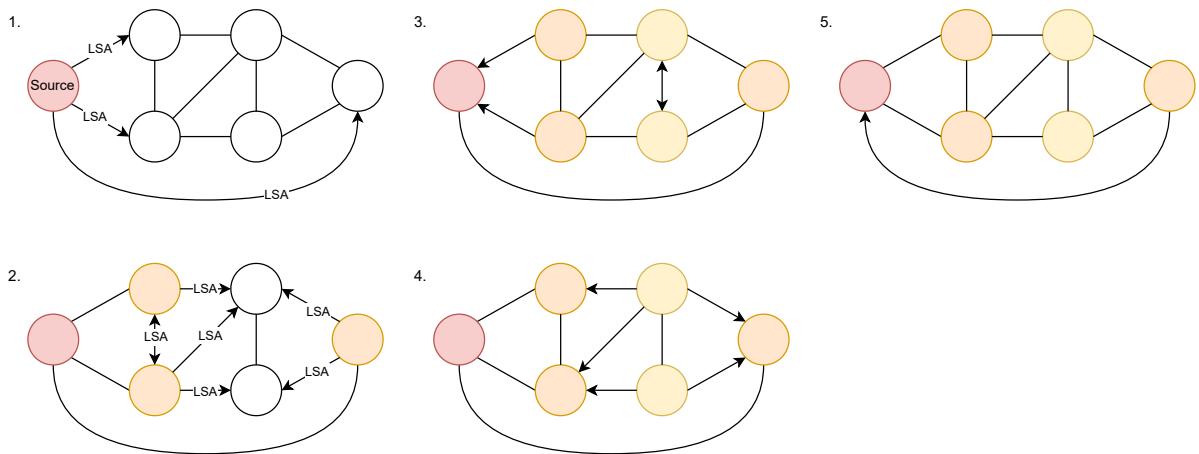
Un **algoritmo di instradamento centralizzato**, calcola il percorso migliore avendo una conoscenza globale della rete.

### 6.2.1 Flooding e Link state broadcast

Conoscere la topologia e i costi della rete è quindi fondamentale per effettuare tutti i calcoli necessari. Il protocollo **Open Shortest Path First** (OSPF), sfrutta il **link-state broadcast**, ovvero l'invio di un particolare tipo di messaggio, chiamato **link-state advertisement** che viene distribuito nella rete tramite una tecnica chiamata **flooding**.

#### Flooding

È un protocollo di instradamento usato dai router: questi **inoltrano un pacchetto** in ingresso su tutte le **linee ad eccezione di quella da cui proviene**. Viene usato per **studiare la topologia della rete** e condividere tra i nodi informazioni relative ai **costi**.



In reti più complesse e con più cicli, il flooding può creare un numero enorme di pacchetti duplicati, che potrebbero saturare la rete. È quindi consigliato introdurre dei meccanismi che controllino il flooding, come i seguenti:

- ID per pacchetto, per evitare i duplicati.
- Impostare un limite di hop.
- Rimuovere i cicli dal grafo della rete.
- Inoltrare i messaggi solo lungo uno spanning tree.

### 6.2.2 Alcune convenzioni

- $G$  è il grafo.
- $G.V$  è l'insieme dei nodi del grafo,  $G.E$  l'insieme degli archi.  $w$  è una funzione (o una struttura dati contenente) il peso, che stabilisce il costo di un arco che unisce due nodi adiacenti.
- $d.v$  indica il costo minimo stimato dal nodo sorgente nell'iterazione corrente. Parte sempre da  $+\infty$ , tranne per il nodo sorgente, per cui  $d.s = 0$ .
- $\delta(s, v)$  indica l'effettivo costo del cammino minimo dal nodo sorgente a  $v$ . Presumibilmente, il nostro obiettivo sarà ottenere  $d.v = \delta(s, v) \forall v \in G.V$ .
- $p.v$  è il predecessore di  $v$  all'interno del cammino di costo minimo dal nodo sorgente  $s$ .

### 6.2.3 L'algoritmo di Dijkstra

Dato in input un grafo  $G$ , un nodo sorgente  $s$  e una struttura dati contenenti gli archi e il loro peso  $w$ , Dijkstra calcola tutti i cammini minimi che partono da  $s$ , verso ogni nodo  $v \in G.V$ . Si suddivide in due fasi.

#### Inizializzazione

1. Si pone  $d.v = +\infty$  su tutti i nodi, tranne per il nodo sorgente  $s$ , per cui  $d.s = 0$ .
2. Si pone come  $p.v = null$  per ogni nodo.
3. Si crea un insieme  $S = \emptyset$ , che conterrà tutti i nodi di cui si conosce il cammino di peso minimo da  $s$ , inizialmente vuoto. Lo utilizzeremo nel corso dell'algoritmo.
4. Si crea una struttura dati  $Q = \emptyset$ , di cui parleremo meglio dopo.
5. Si inseriscono tutti i nodi all'interno di  $Q$ .

#### Iterazione

1. A ogni passaggio, estrarre il nodo  $u$  con  $d.u$  minimo da  $Q$ .  
Alla prima iterazione, il nodo estratto sarà  $s$ .
2. Inserisci  $u$  nell'insieme  $S$ .
3. Verifica se è possibile ridurre  $d.v$  dei nodi  $v$  adiacenti ad  $u$ , passando proprio dal nodo  $u$ . La verifica si ripete su ogni nodo adiacente ad  $u$ .
4. Se l'insieme  $S = G.V$ , smetti di iterare, altrimenti torna al passo 1 dell'iterazione.

Nota bene: ogni volta che è possibile ridurre  $d.v$  passando da  $u$ , bisogna aggiornare il contenuto di  $Q$ .

### 6.2.4 Complessità dell'algoritmo

$Q$  sarà una struttura dati su cui effettuare operazioni di inserimento (all'inizio dell'algoritmo), estrazione del minimo e aggiornamento delle chiavi. Definiamo  $Q$  come una **coda con priorità**. L'implementazione di  $Q$  influenzerà la complessità dell'algoritmo.

- Con  $Q$  implementato con un array semplice, avremo una complessità  $O(|V|^2)$ .
- Con  $Q$  implementato con un min-heap, si otterrà una complessità  $O(|V| + |E| \log |V|)$ .

## 6.3 Instradamento Distance Vector

L'algoritmo distance-vector sfrutta l'algoritmo **Bellman-Ford**, creando un nuovo algoritmo asincrono, che girerà sui singoli nodi della rete. Proprio per questo motivo, introduciamo una notazione  $D_x[v]$ .  $D_x$  è il vettore delle distanze, e  $D_x[v]$  è la distanza stimata da  $x$  a  $v$ .

### 6.3.1 Algoritmo

Su ciascun nodo  $x \in G.V$ , girerà il seguente algoritmo:

#### Inizializzazione

1.  $D_x[x] = 0$ . Il nodo  $x$  dista 0 da se stesso.
2.  $\forall v \in G$ ,  $D_x[y] = w(x, y)$ . Chiaramente, se  $u$  non è adiacente ad  $x$ ,  $D_x[u] = +\infty$
3. Inoltra  $D_x$  a tutti i nodi adiacenti.

#### Iterazione

1. Aspetta fino a quando non ricevi un vettore delle distanze, o finché non cambia il costo verso un nodo adiacente.
2. Ricevuto il vettore delle distanze da un nodo  $D_y$ , si andrà a calcolare, per ogni  $v \in G.V$ ,

$$D_x[v] = \min\{D_x[v], D_y[v] + w(x, y)\}$$

ovvero, se esiste, calcolo il cammino di peso inferiore per andare da  $x$  a  $v$ , passando per  $y$ . Se questa condizione si verifica, e la  $D_x[v]$  di qualsiasi nodo  $v$  cambia, inoltra al vettore delle distanze a tutti i nodi adiacenti.

3. Ritorna allo step 1 dell'iterazione.

Questo algoritmo è un algoritmo **asincrono**, che **itera un numero indefinito di volte su ogni nodo**, in cui i valori minimi calcolati non cambiano, se non successivamente a cambiamenti della rete. In tal caso, ogni nodo cui vettore delle distanze cambia, inoltrerà quest'ultimo agli adiacenti.

### 6.3.2 Poisoned reverse

Esistono però delle **circostanze problematiche**, causate dalla **visibilità limitata** di ogni nodo rispetto ai costi complessivi della rete, che potrebbero causare **loop di inoltro di pacchetto**. In tal caso, si può introdurre un meccanismo di **poisoned reverse**, che permette ad un nodo  $x$  di mentire a  $y$  in una circostanza del genere:

1.  $y$ : voglio inoltrare un pacchetto a  $z$  per arrivare a  $x$ , perché costa meno.
2.  $z$ : voglio inoltrare un pacchetto a  $y$  per arrivare a  $x$ , perché costa meno.

Si può arrivare a questa circostanza quando il peso di un arco non adiacente a  $z$ , ma adiacente a  $y$ , varia. In tal caso,  $z$  vedrà che  $y$  vuole arrivare ad  $x$  passando da se stesso, osserva che lui farebbe l'opposto, e allora "mente" a  $y$  dandogli un costo fittizio  $D_z[x] = +\infty$ , interrompendo il loop.

## 6.4 Routing in Internet

Per facilitare il routing in Internet, quest'ultimo è organizzato in **sistemi autonomi** (AS). Ogni sistema autonomo è trattato come una singola entità amministrativa.

- All'interno di sistema autonomo, tutti i router eseguono lo stesso algoritmo di routing, chiamato **protocollo di instradamento intra-AS**.
- Per l'instradamento tra AS distinti si usano invece **protocolli di instradamento inter-AS**.

Questa organizzazione in sistemi autonomi, garantisce una maggiore **autonomia amministrativa** e migliore **scalabilità**. Ogni sistema autonomo è identificato da un **numero di sistema autonomo (ASN)**.

### 6.4.1 RIP - Routing Information Protocol (intra-AS)

È un protocollo di routing basato sull'algoritmo **Distance Vector** (Bellman-Ford) per il routing interno ai sistemi autonomi. La sua metrica è il **numero di hop**, ovvero il numero di router da attraversare per raggiungere una determinata destinazione.

- Supporta al più 15 hop. Una destinazione  $x$  con  $d.x = 16$  hop è irraggiungibile.
- I router scambiano informazioni relative alle tabelle di routing ogni 30 secondi.
- Un percorso non aggiornato per 180 secondi è marcato come irraggiungibile.
- Dopo altri 120 secondi viene rimosso dalla tabella (garbage collection).

Sfrutta inoltre due tipi di messaggi:

- REQUEST per richiedere informazioni di routing ai router vicini.
- RESPONSE per fornire aggiornamenti sulla propria tabella di routing.

#### Tabelle di routing RIP

Contengono:

- Indirizzo  $x$  di destinazione.
- Distanza minima (in hop) verso  $x$ .
- Prossimo hop per arrivare a  $x$ .
- Un timeout.
- Un timer per la garbage collection.

#### Versioni di RIP

- **RIP v1** supporta solo indirizzamento classful senza subnet. Il campo next hop non è esplicito, e non supporta autenticazione.
- **RIP v2** supporta CIDR, Poison Reverse, autenticazione e next hop esplicito nel messaggio.
- **RIPng** (next generation) per IPv6, basata su RIP v2. Non supporta l'autenticazione.

#### 6.4.2 OSPF - Open Shortest Path First (intra-AS)

È un protocollo di routing intra-AS di tipo **Link State** usato soprattutto nei contesti di rete medio-grandi e nei **backbone degli ISP**. È molto scalabile e robusto. Ogni router OSPF:

- Costruisce una **mappa topologica completa** del sistema autonomo usando il flooding. Costruisce un grafo della rete.
- Esegue localmente l'algoritmo di Dijkstra e determina l'albero dei cammini minimi da se stesso.
- Mantiene un Link-State Database con informazioni sui collegamenti noti.
- Manda aggiornamenti periodici tramite dei Link-State Advertisements, rendendo il protocollo molto robusto.

I messaggi sono del tipo:

- **HELLO**, trasmissione periodica per Neighbor Discovery.
- **Database Description**, per condividere le informazioni link-state tra i vari router
- **LS Request**, per richiedere parti specifiche del link-state database di un vicino.
- **LS Update**, per effettuare il flooding.
- **LS ACK**, dei segnali di ACK relativi al Link-State update.

E ogni nodo effettua questi passaggi:

1. **Scoperta dei vicini**, in cui i router scoprono i nodi adiacenti inviando messaggi **HELLO**.
2. **Sincronizzazione dei database**, **EXCHANGE**, con scambio e confronto delle informazioni Link-State.
3. **Aggiornamento dei database**, **FLOODING**, con messaggi tramite cui i cambiamenti dei cammini minimi sono propagati.

### 6.4.3 BGP - Border Gateway Protocol

È un protocollo di routing **inter-AS**, e quindi tra sistemi autonomi. È fondamentale per il funzionamento di Internet, tanto quanto IP, in quanto è il **collante tra tutti i vari AS**.

- **Comunicare la raggiungibilità delle sotto reti.**

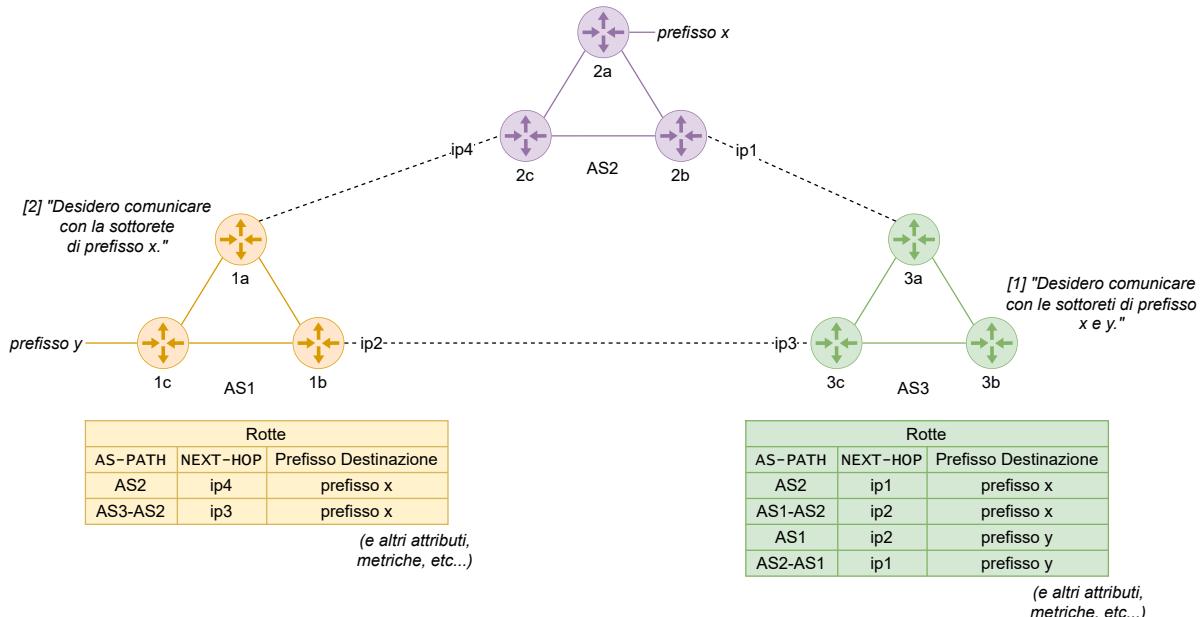
Permettendo a ciascun sistema di rendersi visibile rispetto i **prefissi di rete** che gestisce. È tramite BGP che tutti i router conoscono le sottoreti vicine.

- **Determinare i percorsi ottimali verso le sottoreti.**

Un router può apprendere più percorsi verso un **determinato prefisso** (ovvero una sottorete). La selezione del percorso migliore avviene in base alle politiche del sistema autonomo e alle informazioni di raggiungibilità fornite da BGP.

#### Funzionamento di BGP

Lo scambio di informazioni avviene tramite dei messaggi scambiati, per ogni coppia di router comunicante, in una sessione TCP. Tra router dello stesso sistema autonomo, sono dette **sessioni iBGP**. Altrimenti, si dice **sessione eBGP**.



Supponiamo si voglia **dichiarare la raggiungibilità del prefisso x**. L'annuncio di raggiungibilità verrà passato ai nodi adiacenti, che costruiranno una tabella BGP contenente alcuni attributi, tra cui:

- **AS-PATH:** indica i sistemi autonomi tramite cui è passato questo annuncio.
- **NEXT-HOP:** indica l'IP dell'interfaccia di rete tramite cui "entrare" nel primo sistema autonomo dell'**AS-PATH**.
- Il prefisso x da raggiungere.

L'insieme prefisso x + attributi è detto **route, rotta**. Valutando ogni **NEXT-HOP** secondo una metrica stabilita, i router possono stabilire il percorso migliore.

Code Point	Metric Type
0	IGP Metric
1	Min Unidirectional Link Delay [RFC7471]
2	TE Metric [RFC3630]
3	Hop Count (refer [RFC5440])
4	SID List Length
5	Loss Rate
6-250	Unassigned
251-255	Private Use (not to be assigned by IANA)

Figure 3: Metric Type Code Point

# Capitolo 7

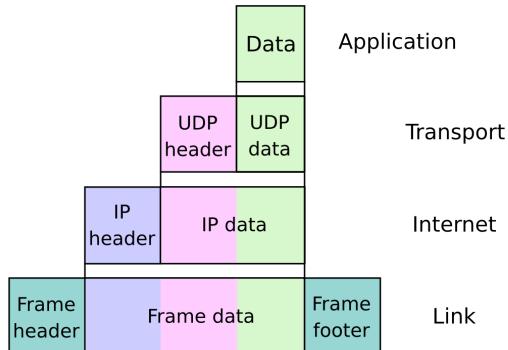
## Data Link Layer

### 7.1 Introduzione al livello Data Link

È il layer che si trova a cavallo tra il mondo logico, e quello fisico. Fornisce al livello di rete un **servizio di trasmissione** per flussi di bit (**bitstream**).

#### 7.1.1 Servizi del Data Link Layer

- **Data framing.**
  - Dal basso verso l'alto, dal fisico al logico raggruppando simboli/bit del layer fisico in frame e **trasformando il flusso grezzo di bit** in informazioni strutturate.
  - Dall'alto verso il basso, dal logico al fisico. incapsulando i datagrammi del livello di rete in frame contenenti due campi, un campo dati *D* e i campi d'intestazione.



- **Gestire l'accesso al mezzo fisico**, nel caso di **canali condivisi**.  
I dispositivi, nel DLL, sono identificati tramite indirizzi MAC, dei veri e propri indirizzi fisici. L'accesso al canale condiviso, nel caso dei **collegamenti broadcast**, dev'essere gestito tramite dei **protocolli d'accesso multiplo**, per evitare **collisioni**.
- **Fornire una consegna dei dati reliable, affidabile**.  
Se il mezzo fisico è abbastanza affidabile (fibra, doppino di rame...), non è necessario implementare meccanismi del genere, in quanto ridondanti rispetto a quelli presenti nel livello di trasporto e application. Nel caso di mezzi di trasmissione meno affidabili, come nel caso delle reti wireless, il trasporto reliable è ottenuto tramite meccanismi di ACK/NACK e ritrasmissioni.
- **Gestire gli errori del canale di trasmissione** (rilevandoli e, se possibile, correggendoli).  
Gli errori possono essere causati da attenuazione del segnale, rumore, interferenze tra campi elettromagnetici o dal mezzo fisico usurato, e sono gestiti tramite checksum, controllo di ridondanza ciclico (CRC), controllo di parità, codici di Hamming o meccanismi simili che conseguono lo stesso scopo.

- **Regola il flusso dati** tra sorgente e destinazione.

I collegamenti possono essere full-duplex o half-duplex, permettendo o non permettendo la trasmissione contemporanea di dati da entrambi i nodi del collegamento. Il pacing di trasmissione deve essere proporzionato a quello di ricezione. I due nodi devono comunicare in maniera opportuna per gestire questa situazione.

### Nozione utile - Rumore

Definiamo come rumore, una forma di distorsione (voluta o non voluta) che si presenta all'interno di un segnale elettrico effettivo. Qualsiasi circuito genera rumore, ed è dovuto alla presenza di resistenze (resistenze vere e proprie e resistenza intrinseca dei conduttori elettrici e i materiali usati). Nelle simulazioni dei circuiti elettrici, è possibile inserire un rumore elettrico chiamato **rumore gaussiano**.

#### 7.1.2 Implementazione del DLL - NIC

In ogni host singolo host (device e router) di una rete, è implementato il Data Link Layer. Ciò avviene tramite una combinazione di **hardware, software e firmware** presente all'interno dei NIC (Network Interface Controller) dei dispositivi di rete. Buona parte del DLL è implementato in hardware, come il protocollo Ethernet nell'adattatore Intel 700, o il protocollo Wi-Fi nel chipset Atheros AR5006. Tuttavia, **è il software a farsi ponte tra il livello di rete, e il livello mondo fisico**.

#### 7.1.3 Comunicazione tra due Network Interfacec Controller

Lato mittente:

1. Il datagramma da trasmettere viene incapsulato in un frame del DLL.
2. I campi d'intestazione del DLL vengono riempiti.
3. (Se presenti) vengono stabiliti i valori dei bit nei campi per la rilevazione e correzione degli errori.
4. Il pacchetto viene trasferito.

Dal lato del ricevente:

1. Il Network Interface Controller riceve l'intero frame.
2. (Se presenti) effettua le verifiche relative ai campi di rilevazione e correzione errori.
3. Estraie il datagramma.
4. Manda il datagramma al livello di rete+.

## 7.2 Data Framing

Nella trasmissione dei dati, è fondamentale trovare una **codifica efficace** che permetta di **interpretare in maniera univoca il segnale elettrico** in stringhe di 0 e 1. A differenza di ciò che si potrebbe pensare, questo problema non è per niente banale. **Distinguere una sequenza continua di zeri o di 1 da un'interruzione** della trasmissione o da un malfunzionamento **non è possibile con una codifica a due livelli**.

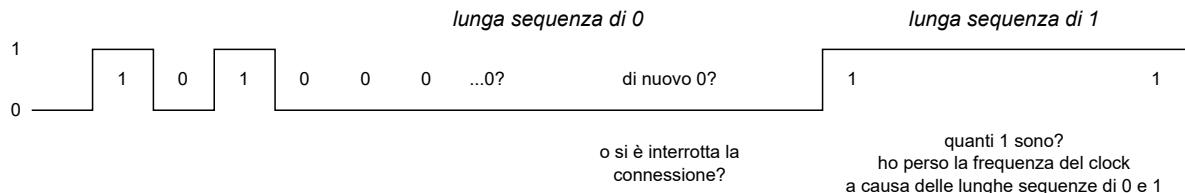


Immagine 7.1: Esempio ipotetico di two-level encoding

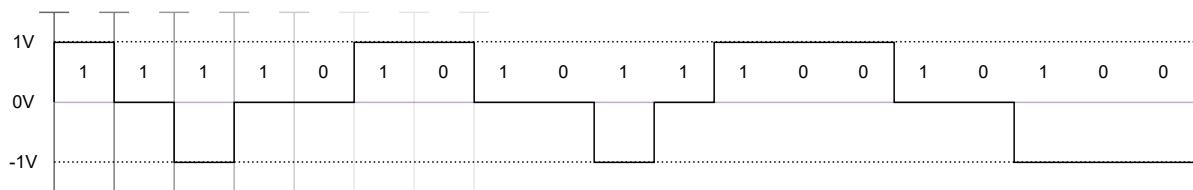
### 7.2.1 Three-Level Encoding

Prevede tre livelli di segnale: alto - zero - basso.

Segue inoltre delle regole, quali:

- Prossimo bit 0.  
Nessuna transizione del segnale.
- Prossimo bit 1 e livello di segnale  $\neq 0$ .  
Il segnale passa a 0.
- Prossimo bit 1 e livello di segnale = 0.  
Il segnale passa ad un livello opposto a quello del più recente segnale diverso da 0.

Osserviamo che le transizioni avvengono solo per trasmettere bit = 1. Inoltre, essendo questa una **trasmissione asincrona**, una trasmissione di 4 bit = 1, ossia un ciclo completo, fornirà al receiver un vero e proprio **clock recovery**, che stabilisce una frequenza di lettura<sup>1</sup> consona, rispetto ai dati in arrivo.



Questo meccanismo di clock permette al receiver di intuire il numero di bit = 0 da leggere quando, in un quanto di tempo, il segnale rimane ad un certo livello. Chiaramente, lunghe stringhe di zeri possono ancora desincronizzare il receiver, ed è per questo che andremo a parlare (più avanti) di meccanismi che risolvono il problema della **perdita della sincronizzazione**.

#### Esempio di Three-level encoding e Five-level encoding

- La Fast Ethernet (100 BASE TX) utilizza una codifica a tre livelli di segnale: +1V, 0V, -1V. In realtà questi valori non sono statici: sono supportati dei **range di tollerabilità**.
- Specificando più di tre range di voltaggi, e quindi più di tre livelli, diventa possibile trasferire 2 bit alla volta. La Gigabit Ethernet (1000 BASE T) utilizza una codifica a cinque livelli: +1V, +0.5V, 0V, -0.5V, -1V. Questa codifica permette di trasmettere in una sola volta i bit 00, 01, 10, 11, dipendentemente dal livello di segnale.

<sup>1</sup>Per questo è comune che, in determinati protocolli, la trasmissione sia preceduta da una sequenza di bit chiamata **preamble**. Ne parleremo meglio più avanti.

## 7.2.2 Codifica a blocchi

Il three-level encoding è la base per la trasmissione dei dati. Tuttavia, questo meccanismo, presenta due criticità:

- **Non è DC-Balanced.** Il livello di tensione medio non è necessariamente mantenuto tendente a 0. Può portare complicazioni di vario tipo.
- Se ci sono inoltre lunghe sequenze di zero, si ha un problema di **perdita di sincronizzazione**.

La **codifica a blocchi** sostituisce blocchi di bit di una determinata dimensione, con blocchi leggermente più grandi ( $4 \rightarrow 5$ ,  $8 \rightarrow 10$ ), traducendo le sequenze in maniera tale da garantire un numero minimo di transizioni.

### Codifica 4B5B

Associa a blocchi di 4 bit, blocchi di 5 bit. Richiede una bandwitdh del 25% più capiente. È utilizzata per fast ethernet.

Nome	4B	5B	Descrizione
0	0000	11110	hex data 0
1	0001	01001	hex data 1
2	0010	10100	hex data 2
3	0011	10101	hex data 3
4	0100	01010	hex data 4
5	0101	01011	hex data 5
6	0110	01110	hex data 6
7	0111	01111	hex data 7
8	1000	10010	hex data 8
9	1001	10011	hex data 9
A	1010	10110	hex data A
B	1011	10111	hex data B
C	1100	11010	hex data C
D	1101	11011	hex data D
E	1110	11100	hex data E
F	1111	11101	hex data F
I	-NONE-	11111	Idle
J	-NONE-	11000	SSD #1
K	-NONE-	10001	SSD #2
T	-NONE-	01101	ESD #1
R	-NONE-	00111	ESD #2
H	-NONE-	00100	Halt

Questa codifica a blocchi non garantisce DC-balance.

### Codifica 8B10B

Nella mappatura la sequenza a 8 bit viene divisa in due sottosequenze: una a 5 bit e una a 3 bit. Queste sezioni vengono mappate in modo tale da essere codificate in 6 e 4 bit rispettivamente: concatenate, si ottiene la sequenza di 10 bit. Tra queste sequenze, 12 sono dette **sequenze di controllo**, come i **comma**, che permettono al ricevente di allineare in maniera opportuna il flusso di bit. È una codifica **DC-Balanced**.

### 7.2.3 Quantità di informazioni

Il seguente concetto appartiene alla branca della **teoria dell'informazione tradizionale**. Dati  $n$  di simboli distinti (in questo caso, livelli di tensione), come H-M-L o HH-HM-HL-MH-MM-ML-LH-LM-LL, è possibile trasportare un numero di informazioni binarie, quantificate in bit, pari a

$$x = \log_2 n$$

Nel pratico, non potendo esprimere bit parziali, il numero effettivo è dato da

$$x = \lfloor \log_2 n \rfloor$$

Con  $n = 3$  livelli, abbiamo  $\lfloor \log_2 3 \rfloor = 1$ . Con  $n = 5$  livelli, abbiamo  $\lfloor \log_2 5 \rfloor = 2$ . Moltiplicando il numero ottenuto, per il numero di step (o simboli) al secondo, otteniamo il **limite teorico di bitrate**.

$$\text{Limite teorico bit-rate} = \text{Simboli al secondo} \cdot \log_2 n$$

#### Piccolo approfondimento sulla quanittà di informazioni - non è nel programma

Questa formula è anche detta *Misura di Hartley*, e si tratta di una versione semplificata del concetto di **entropia informativa di Shannon**, valida solo nel caso di simboli (o eventi) **equiprobabili**.

## 7.3 Gestione degli errori

- L'**error detection** consiste nell'individuare errori di trasmissione in un frame. Tuttavia non effettua correzione.
- La **correzione** permette di individuare errori di trasmissione, ed effettuare correzioni (di dimensioni limitate).

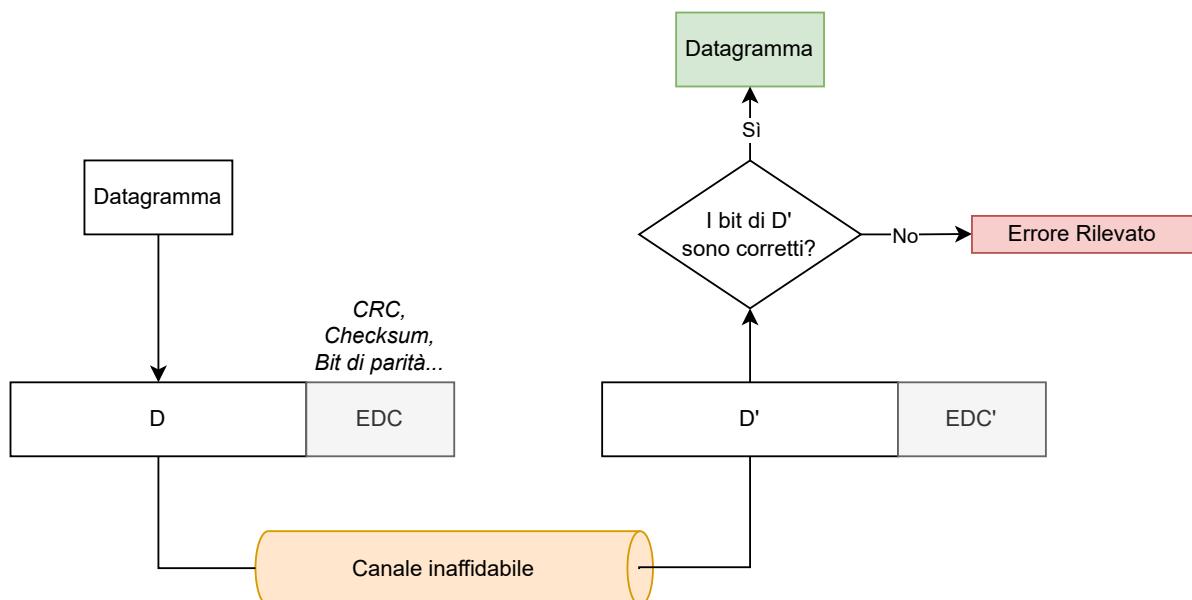
Entrambi i metodi sono basati sull'analisi di informazioni ridondanti, per individuare anomalie nell'informazione.

### 7.3.1 Ridondanza: elemento di base per l'error detection

Immaginiamo di voler trasmettere dei dati anagrafici all'interno di un generico pacchetto. Aggiungere allo stesso pacchetto un codice fiscale, mi fornisce delle informazioni sì ridondanti, ma che mi permettono di individuare possibili errori di trasmissione successivamente alla comunicazione.

### 7.3.2 Error Detection

Un datagramma inoltrato attraverso un mezzo-fisico che ammette errori, contiene al suo interno una sequenza di bit  $D$  e un *EDC* (Error Detection and Correction bits). L'*EDC* può essere un qualsiasi tipo di controllo, CRC, checksum, controllo di parità.



### 7.3.3 Parity check

Consiste in un bit che specifica se il numero di bit = 1 è pari o dispari. Può essere un controllo parità a uno o due dimensioni.

#### Monodimensionale

Consente esclusivamente di individuare un errore di trasmissione

```
D:          Parity bit:  
0010011010010101      0      nessun errore rilevato!  
D:          Parity bit:  
0001001010010101      1      nessun errore rilevato!  
D:          Parity bit:  
0010010010010101      0      errore rilevato!
```

#### Bidimensionale

Consente di individuare e di correggere un errore.

Original Data				
10011001	11100010			
Row parities				
10011001	0			
11100010	0			
00100100	0			
10000100	0			
11011011	0			
Column parities				
100110010	111000100	001001000	100001000	110110110
Data to be sent				

### 7.3.4 CRC - Cyclic Redundancy Check

È un metodo di error detection, non correzione. È detto anche controllo dei **codici polinomiali**. Sfrutta l'aritmetica a mod 2, in cui addizione e sottrazione diventano operazioni di *XOR*.

La sequenza di bit  $D$  è rappresentata sottoforma di polinomio di grado  $d-1 = |D|-1$ , con coefficienti pari al valore del bit nella sequenza. Il bit più a sinistra è il coefficiente di grado  $d-1$  ( $x^{d-1}$ ), quello più a destra è il termine noto ( $x^0$ ). Alla sequenza  $D$  10010111 coinciderà il polinomio:

$$x^7 + x^4 + x^2 + x^1 + x^0 = x^7 + x^4 + x^2 + x^1 + 1$$

- Sia  $D(x)$  il polinomio associato al nostro messaggio  $D$ .
- Sia  $G(x)$  il polinomio generatore dello standard di rete. La cifra più significativa è sempre 1.
- Sia  $r$  il grado di  $G(x)$ , ed è quindi stabilito dallo standard.
- Sia  $R(x) = x^r \cdot D(x) \bmod G(x)$

### 7.3.5 Come effettua il CRC

Al sender.

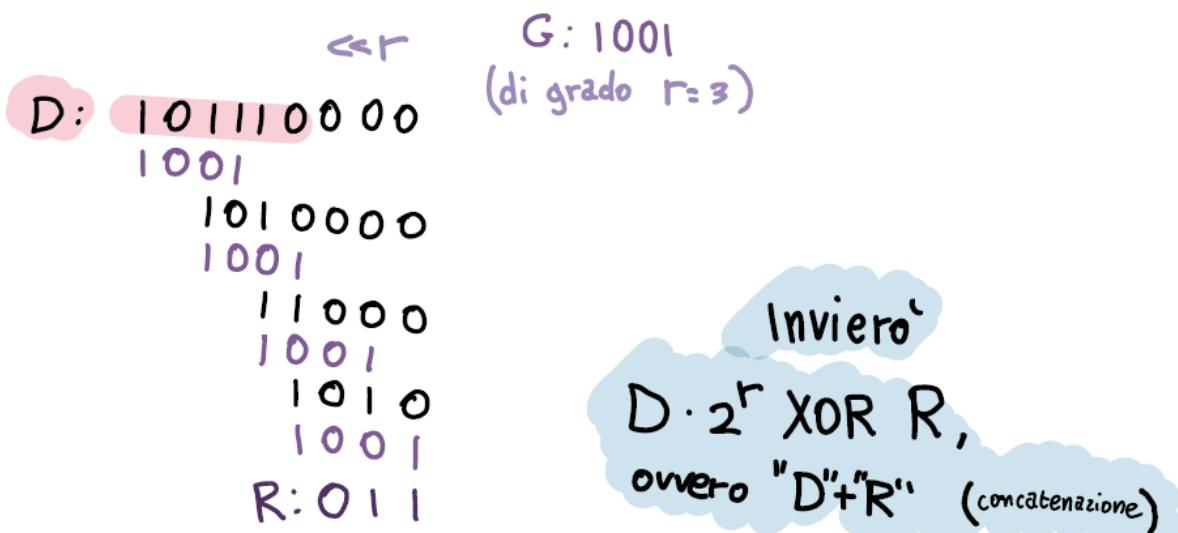
1.  $D$  sono i dati che vogliamo trasmettere.
2. Calcoliamo  $D \cdot 2^r$ , nel pratico  $D << r$ .
3. Effettuamo la divisione mod 2 tra  $\frac{D \cdot 2^r}{G}$ .
4. Il resto della nostra divisione sarà il nostro valore di  $R$ .
5. Il messaggio che manderemo sarà il risultato della concatenazione di  $D + R$ .  
In termini di bit:  $D \cdot 2^r \text{ XOR } R$ .
6. Questo valore sarà sempre divisibile per  $G$ , con resto nullo.

Al receiver.

1. Calcola  $\frac{\text{Stringa ricevuta}}{G}$ .
2. Se questa divisione ha resto, ci sono stati problemi di trasmissione.

### 7.3.6 Esiti del CRC

Questo sistema permette di individuare errori a burst al più di  $r$  bit.



## 7.4 Error correction - Perché è difficile?

Correggere un errore di trasmissione è molto difficile. Il linguaggio naturale gode di contesti (e semantica), tramite cui è possibile estrapolare il significato di una frase, anche quando essa è incompleta! "*Tetto casa rosso*" → "*Il tetto della casa è rosso!*". All'interno di sequenze di bit, non esiste un modo del genere per correggere gli errori. Quello della correzione è un problema molto complesso, verso cui Internet non verte. Quelle di error correction, tuttavia, sono metodologie importanti da conoscere, in quanto utilizzate in molteplici contesti relativi alla trasmissione dati, come all'interno delle RAM.

### 7.4.1 Distanza di Hamming

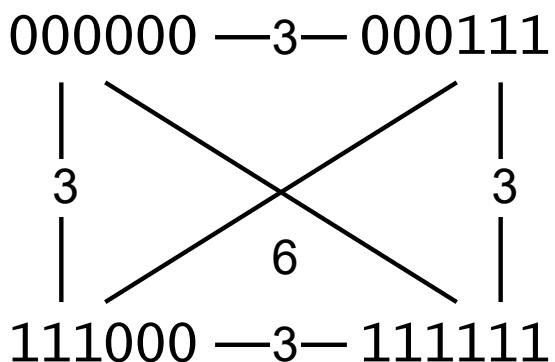
Date due codeword, è possibile stabilire una distanza, chiamata **distanza di Hamming**, che indica il numero di bit che differiscono tra le codeword.

```
10001100 XOR
11000100 =
01001000 -> d=2
```

Definiamo **vocabolario** un insieme di codeword. Un vocabolario si dirà **completo** se, con  $n$  bit, avrà  $2^n$  codeword.

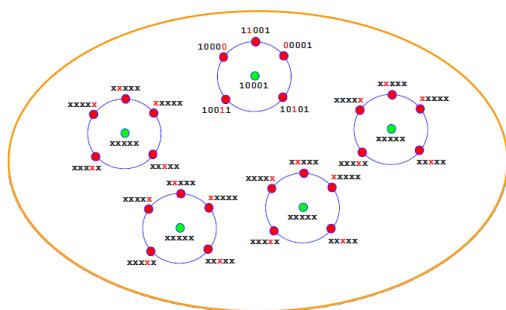
### 7.4.2 Correzione degli errori - distanza di Hamming

Limitando un vocabolario a delle stringhe con distanza di Hamming minima fissata, si può ottenere un dizionario del genere: con  $\min d = 3$ .

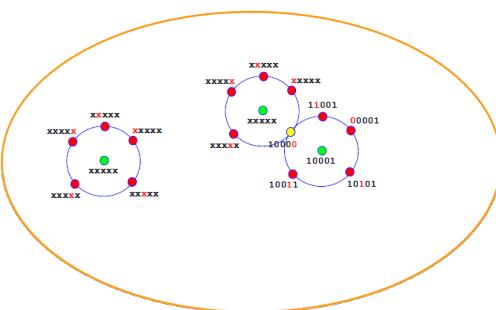


Questo limita il numero di stringhe valide, permettendo di introdurre meccanismi di riconoscimento e correzione di errori!

- Per **correggere**  $e$  bit di errori, abbiamo bisogno di un vocabolario con distanza minima  $d = 2e + 1$ .
- Per **individuare**  $e$  bit di errori, abbiamo bisogno di un vocabolario con distanza minima  $d = e + 1$ .



*Insiemi totalmente disgiunti:  
è possibile individuare e correggere gli errori*



*Insiemi parzialmente disgiunti:  
è possibile individuare, ma non correggere, gli errori*

### 7.4.3 Dimostrazione distanza minima

Dato un vocabolario con  $m$  bit e  $r$  control bit.

$$m + r = n$$

Avremo quindi  $2^n$  combinazioni, con  $2^m$  valide.

Data una codeword valida, si ottengono  $n$  codeword non valide modificando un singolo bit.

#### Formula generale

$$\sum_{i=0}^e \binom{m+i}{r} \leq 2^r$$

Per correggere un errore, avremo:

m	r	n
1	2	3
2	3	5
3	3	6
4	3	7
5	4	9
6	4	10
7	4	11
8	4	12
9	4	13
10	4	14
11	4	15
12	5	17

[...]

Per esempio, volessimo trasferire un messaggio di 5 bit, e volessimo verificare un solo errore  $e$ , possiamo calcolare:

$$\begin{aligned} \sum_{i=0}^1 \binom{m+i}{r} &\leq 2^r \\ \binom{5}{r} + \binom{6}{r} &\leq 2^r \end{aligned}$$

- Con  $r = 0$ ,  $\binom{5}{0} + \binom{6}{0} \leq 2^0 \rightarrow 1 + 1 \leq 1 \rightarrow 2 \leq 1$  - falso!
- Con  $r = 1$ ,  $\binom{5}{1} + \binom{6}{1} = \frac{5!}{4!} + \frac{6!}{5!} = \frac{120}{24} + \frac{720}{120} = 5 + 6 = 11 \leq 2^1$  - falso!
- Con  $r = 2, \dots$  - falso
- Con  $r = 3, \dots$  - falso
- Con  $r = 4, \dots$  - falso
- Con  $r = 5$ ,  $\binom{5}{5} + \binom{6}{5} = 1 + \frac{6!}{5!} = 1 + \frac{720}{120} = 1 + 6 = 7 \leq 2^5$  - vero!

#### Reminder utile - formula del coefficiente binomiale

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

#### 7.4.4 Come si effettua la verifica della distanza di Hamming?

I bit di controllo sono sempre posizionati in bit di posizioni potenze di due ( $1, 2, 4, 8, 16, \dots$ ).

Il valore del bit di controllo  $b_r$  è dato dallo  $XOR$  di tutti i valori ottenibili con quel  $b_r = 1$ . Ad esempio, con  $\text{xx1x001x0}$ , abbiamo:

- $b_1 = b_3 \otimes b_5 \otimes b_7 \otimes b_9 = 1 \otimes 0 \otimes 1 \otimes 0 = 0$
- $b_2 = b_3 \otimes b_6 \otimes b_7 = 1 \otimes 0 \otimes 1 = 0$
- $b_4 = b_5 \otimes b_6 \otimes b_7 = 0 \otimes 0 \otimes 1 = 1$
- $b_8 = b_9 = 0$

E quindi, la stringa che attendiamo, sarà  $001100100$ .

#### Controllo al mittente

001101100  
^ ^ ^ ! ^

$\hat{\phantom{x}}$  = bit da controllare  
 $|$  = errore di comunicazione

- $b_1 = b_3 \otimes b_5 \otimes b_7 \otimes b_9 = 1 \otimes 0 \otimes 1 \otimes 0 = 0$
- $b_2 = b_3 \otimes b_6 \otimes b_7 = 1 \otimes 1 \otimes 1 = 1$  Errore! Nella stringa ricevuta,  $b_2 = 0$ .
- $b_4 = b_5 \otimes b_6 \otimes b_7 = 0 \otimes 1 \otimes 1 = 0$  Errore! Nella stringa ricevuta,  $b_3 = 0$ .
- $b_8 = b_9 = 0$

L'errore è situato nello slot  $b_{2+4} = b_6$ . Per correggerlo, effettuiamo un bitflip.

## 7.5 Gestire gli accessi multipli

In una rete, possiamo individuare due tipi di collegamento:

- **Collegamenti punto a punto.**

Sono collegamenti diretti tra trasmittente e ricevente.

- **Collegamenti broadcast.**

Sono collegamenti in cui più nodi trasmittenti e riceventi sfruttano lo stesso canale broadcast.

La gestione di un canale broadcast non è un problema per niente intuitivo da risolvere. I protocolli di accesso multiplo gestiscono questo problema, fissando le modalità con cui i nodi regolano le trasmissioni sul canale condiviso.

Quando due nodi cercano di trasmettere sullo stesso canale nello stesso istante, avviene una **collisione**. Le collisioni causano **perdite di frame**, e, in eccessiva frequenza, un generale spreco di banda. I **protocolli di accesso multiplo** gestiscono i canali in maniera consona per non rientrare in situazioni del genere. Esistono tre tipi principali di protocolli per l'accesso multiplo:

- **Protocolli a suddivisione del canale.**

Basati sulla suddivisione del canale in porzioni accessibili esclusivamente ad un nodo.

- **Protocolli ad accesso casuale.**

In cui l'accesso al canale avviene in maniera casuale. Il metodo in questione ammette collisioni, che verranno gestite in maniera opportuna.

- **Protocolli a rotazione - senza collisioni.**

Coordinando opportunamente l'accesso al canale, è possibile evitare le collisioni.

### 7.5.1 Protocolli a suddivisione del canale

#### TDMA - Time Division Multiple Access

Supponiamo una canale broadcast supporti  $N$  nodi, e che abbia velocità  $R$  bps.

TDMA, suddividerà il tempo in **intervalli di tempo**, e ogni intervallo di tempo in  $N$  slot temporali. Ogni slot sarà dedicato ad un nodo: come conseguenza, ogni nodo avrà un tasso di trasmissione di  $R/N$  bps. Risolve effettivamente il problema delle collisioni, ma quando si è in uno slot temporale dedicato a un nodo che **non deve trasmettere nel canale**, avverrà un'attesa non necessaria. È un **protocollo equo**, che **previene le collisioni**, ma **limita la banda** di ogni nodo a  $R/N$ , e porta i nodi in attesa anche quando gli altri non devono trasmettere nulla.

#### FDMA - Frequency Division Multiple Access

Consiste nella suddivisione dello spettro del canale in varie **bande di frequenza**. A ogni nodo è associata una banda fissa. Questa suddivisione limita la banda, sempre ad  $R/N$ , ma non blocca la trasmissione dei nodi.

#### CDMA - Code Division Multiple Access

Consiste nell'associazione di un codice univoco ad ogni nodo. I dati inviati sono codificati secondo quel codice. Con codici opportuni, diventa possibile trasmettere simultaneamente dati a più nodi, e per i nodi, di interpretare in maniera corretta i bit dei dati codificati. Ogni nodo deve conoscere i codici altrui per interpretare correttamente i dati.

### 7.5.2 Protocolli ad accesso casuale

Ogni nodo trasmette alla massima velocità consentita ( $R$  bps), e ogni volta che avviene una collisione, i nodi ritrasmetteranno dopo un periodo di tempo casuale (*random delay*). Questo ritardo casuale è indipendente tra i vari nodi, e consente, con buona probabilità, di non subire ulteriori collisioni.

*Attenzione, nei seguenti appunti si è deciso di invertire l'ordine con cui, a lezione, sono stati esposti i protocolli ALOHA e Slotted ALOHA. Si è preferito di seguire l'ordine del libro, e di trattare ALOHA come una versione totalmente asincrona di slotted ALOHA.*

#### Slotted ALOHA

- Tutti i frame contengono  $L$  bit.
- Il tempo è suddiviso in slot  $L/R$  secondi.
- La trasmissione del frame da parte del nodo, inizia all'inizio degli slot.
- I nodi sono sincronizzati in modo che tutti sappiano quando iniziano gli slot.
- Se in uno slot due o più frame collidono, tutti i nodi della rete rilevano l'evento prima della fine dello slot.

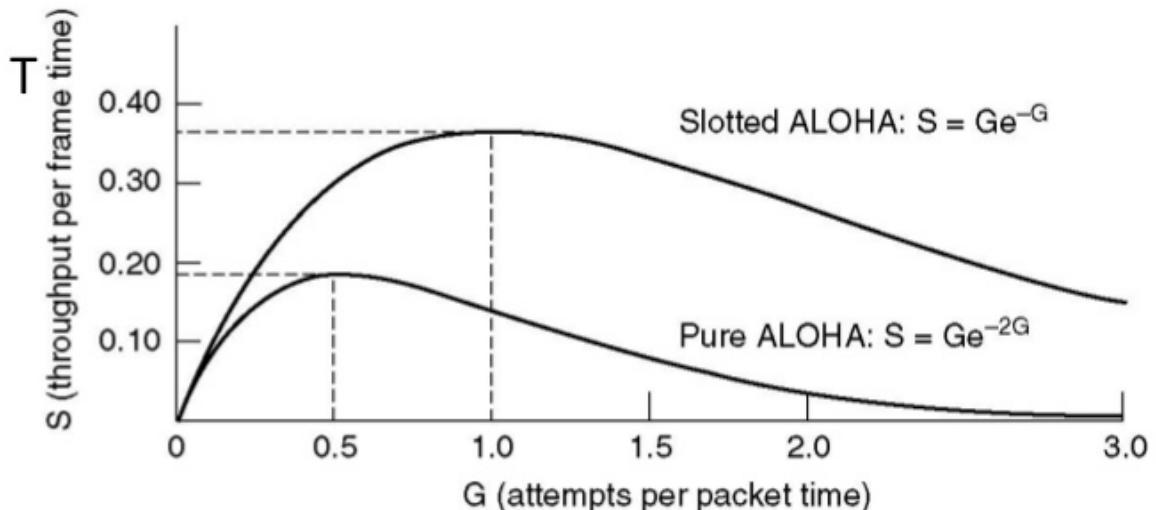
Indichiamo  $k \in [0, 1 - K]$ ,  $k, K \in \mathbb{N}$ . I nodi slotted ALOHA seguono il seguente comportamento:

1. Quando un nodo ha un nuovo frame da spedire, attende l'inizio dello slot successivo, per poi trasmettere il frame.
2. Se non si verifica una collisione, la trasmissione è conclusa.
3. Se si verifica una collisione, il nodo la rileva prima del termine dello slot e ritrasmette il frame dopo  $k$  frame, fino a quando l'operazione non ha successo.  $k$  è un valore casuale minore di  $K$ .  $K$  cresce ad ogni collisione.

È un protocollo semplice, che consente ad ogni nodo di lavorare al massimo delle performance possibili, senza limitare i bps a priori. Inoltre, è un protocollo altamente scalabile e abbastanza decentralizzato: l'unica condizione, è che gli slot siano sincronizzati per tutti i nodi. Tuttavia, ammette collisioni e idle slots, uno spreco generale di slot e un aumento della latenza.

#### Unslotted ALOHA

È una versione totalmente asincrona dello slotted ALOHA. Il nodo mittente aspetterà un tempo pari al Round Trip Time, in attesa di un ACK. Se l'ACK non arriva, il nodo intuirà che è avvenuta una collisione. In tal caso, aspetterà un quanto di tempo di durata casuale (ma inclusa in un range determinato). Le performance migliori, sono ottenute dalla versione con gli slot temporali.



È un grande prezzo da pagare quello per la completa asincronia: l'algoritmo slotted ha un throughput complessivo due volte più grande di quello slotted. L'intuizione è banale: nella versione slotted di ALOHA, due frame possono essere mandati solo all'inizio dello slot temporale. Nella versione unslotted, i frame possono collidere anche a metà trasmissione.

## CSMA - Carrier Sense Multiple Access

I protocolli ALOHA, portano i nodi a decidere di trasmettere indipendentemente dall'attività degli altri nodi. I protocolli CSMA introducono un meccanismo di **rilevamento della portante** del canale, per "ascoltare" se **altri nodi stanno già trasmettendo**. In questo modo, i nodi trasmetteranno esclusivamente quando non rilevano alcuna trasmissione già presente nel canale.

Al contrario di ciò che si potrebbe pensare, questo protocollo non sconfigge definitivamente la problematica delle collisioni. La velocità di trasmissione (nonostante spesso sia altissima) è un fattore non trascurabile.

1. Il nodo *A* trasmette all'istante  $t_0$ .
2. *B* misura la portata del canale all'istante  $t_1 > t_0$ .
3. All'istante  $t_1$  i bit trasmessi da *A* potrebbero non essersi propagati fino a *B*.
4. *B* trasmette. Collisione!

## CSMA-CD - Carrier Sense Multiple Acces con Collision Detection

Un miglioramento è ottenuto se si introduce un meccanismo di collision detection sopra quello di rilevamento del carico: la scheda che avrà trasmesso un determinato frame continuerà a misurare il carico del canale. Se rileva rumore o energia di altre trasmissioni nel canale, il nodo interromperà istantaneamente la trasmissione del frame, manderà un **segnale di jamming con backoff** a tutti gli host, informando tutti i nodi della collisione, e attenderà un quantitativo di tempo casuale per poi ritrasmettere. Questo tempo è chiamato **tempo di backoff**, non è fissato (intuitivamente renderebbe inutilizzabile il protocollo), e dipende dall'algoritmo scelto dal protocollo.

Un esempio di algoritmo, è quello di **attesa binaria esponenziale**, o *binary exponential backoff*: all' $n$ -esima collisione, si estrae un valore dall'insieme  $K = \{0, 1, 2, \dots, 2^n - 1\}$ . Si aspetterà quindi un tempo  $T = K \cdot T_{\text{slot}}$ . Il tempo  $T_{\text{slot}}$  dipende dal protocollo: in Ethernet, coincide col tempo richiesto per trasmettere 512 bit,  $n$  non supera mai 10, e quindi  $0 \leq K \leq 1023$ .

È un ottimo protocollo, ma la collision detection deve avvenire rapidamente: risulta difficile da implementare in reti wireless.

### 7.5.3 Protocolli senza collisioni

Questi protocolli **prevengono le collisioni** introducendo dei **meccanismi preliminari alle trasmissioni**, ottenendo una sincronizzazione consona tra i vari nodi.

#### Protocollo bit map - una sorta di prenotazione!

Con  $N$  nodi, il tempo verrà in primis suddiviso in  $N$  mini-slot che consentono la trasmissione di un bit. Prima delle trasmissioni effettive, ogni nodo inserirà 1 o 0 nel proprio mini-slot. In questo modo, ogni nodo potrà annunciare se ha intenzione di trasmettere un frame (1) o di non trasmettere (0). Alla fine di questa trasmissione iniziale, si avrà una **mappa di bit**, che permetterà ai nodi di seguire un ordine di trasmissione tale da non causare collisioni.

#### Binary Backward Counting - una sorta di... torneo?

In questo protocollo, ogni nodo presenta un ID binario. Questo ID permetterà di risolvere il problema delle collisioni col seguente approccio. Supponiamo *A, B, C, D* siano dei nodi intenzionati a trasmettere:

1. Supponiamo che i nodi *A, B, C, D* abbiano rispettivamente id 0010, 1010, 0100, 1100.
2. Tutti i nodi condividono nel proprio mini-slot, la cifra più significativa del proprio id.  
 $A : 0, B : 1, C : 0, D : 1$ .
3. Escono dalla competizione i nodi con prima cifra 0. Si passa alla seconda cifra significativa.
4.  $B : 0$   $D : 1$ . *D* ha vinto la contesa, trasmetterà.

Potrebbe essere unfair sotto determinate circostanze: gli ID binari sono calcolati anche in virtù dei MAC Address, rendendo alcuni dispositivi implicitamente avvantaggiati.

### Taking turns protocol

Ogni nodo manda 1 nel canale per segnalare di voler trasmettere dei frame. I nodi che hanno inviato 1, si inseriscono in un anello logico. Il primo dei frame di questo nodo sarà il primo a trasmettere. Dopo un tempo  $T$ , questo nodo passerà un **token** al prossimo nodo dell'anello, in modo da permettere al prossimo nodo di trasmettere. Tuttavia, **un guasto di un nodo blocca il sistema** di token-passing. Inoltre, l'anello è un circuito chiuso: occorrerà quindi includere l'intera subnet all'interno dell'anello di trasmissione, per non avere nodi bloccati.

## 7.6 Ethernet

Ethernet è di gran lunga lo standard di reti cablate più diffuso al mondo, per le reti locali. Rappresenta la **prima LAN ad alta velocità con vasta diffusione**. È riuscito a rimanere al passo tramite versioni via via più veloci, e mantenendo i costi delle componenti hardware non particolarmente alti. È parte dello standard IEEE 802.3.

### 7.6.1 Ethernet CSMA/CD

È il protocollo d'accesso multiplo al canale condiviso usato da Ethernet. Lo abbiamo trattato nel paragrafo 7.5.2.

### 7.6.2 Tecnologie standardizzate Ethernet

Le tecnologie standardizzate Ethernet presentano una denominazione fissa:

- Un **numero** che indica la **velocità dello standard**.
- BASE, in quanto trasferisce **solo traffico Ethernet**.
- Un'**etichetta** che ne specifica il **mezzo fisico**.

Nome	Cavo	Lunghezza massima	Nodi	Vantaggi e caratteristiche note
10base5	Coassiale (spesso)	500m	100	Primo cavo: ormai deprecato
10base2	Coassiale (snello)	185m	30	Non richiede un hub
10base-T	UTP	100m	1024	Sistema più economico
10base-F	Fibra Ottica	2000m	1024	Copre grandi distanze

Tabella 7.1: Tecnologie Ethernet a 10Mbps

Il coassiale spesso soffre per la sua estrema rigidità, il coassiale snello si rompe troppo facilmente.

Nome	Cavo	Lunghezza massima	Nodi	Vantaggi e caratteristiche note
100base-T4	UTP	100m		Economico, encoding 8B6T
100base-TX	UTP	100m		Full Duplex, encoding 4B5B
100base-FX	Fibra Ottica	2000m		Copre grandi distanze

Tabella 7.2: Tecnologie Ethernet a 100Mbps

Un'estensione degli standard Ethernet è Gigabit Ethernet, detto anche IEE 802.3z.

Nome	Cavo	Lunghezza massima	Nodi	Vantaggi e caratteristiche note
1000Base-SX	Fibra multimodale	550m		Più segnali simultanei
1000Base-LX	Fibra multimodale	5000m		Copre grandissime distanze
1000Base-CX	2 coppie di STP	25m		
1000Base-T	4 coppie di UTP	100m		Full Duplex, encoding a 5 livelli

Tabella 7.3: Tecnologie Ethernet a 1Gbps

Inoltre, 1000Base-T non usa il 4B5B e introduce un **forward error correction**.

### 7.6.3 Cavi

La trasmissione di un cavo è influenzata dal come è stato realizzato. Alcuni fattori da tenere in considerazione, sono i seguenti:

- Il passaggio di corrente, che induce un **campo magnetico**, crea **corrente indotta**. Questa scorre dal verso opposto al flusso di corrente di partenza, il quale subirà una vera e propria **opposizione**. Anche i cavi adiacenti possono creare campi magnetici e quindi flussi opposti: questo fenomeno si chiama **diafonia**.
- La **lunghezza** del cavo è **direttamente proporzionale** alla sua **resistenza**: per coprire grandi distanze, è necessario usare dei **ripetitori**.

#### Parti dei cavi di rame

- **Conduttore.**
- **Jacket.**  
Materiale isolante esterno.
- **Shield.**  
Schermatura complessiva su tutti i cavi.
- **Foil.**  
Schermatura su una coppia di cavi.

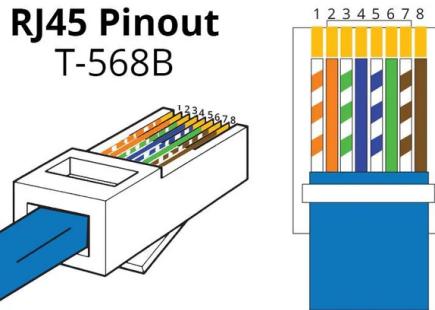
Detto ciò, **UTP** sta per **Unshielded Twisted Pair**, **STP** sta per **Shielded Twisted Pair**, **FTP** sta per **Foiled Twisted Pair**, **SFTP** sta per **Shielded Foiled Twisted Pair**.

#### Categorie cavi di rame

- **Cat5**, che supporta fino a 100Mbps e opera sui 100Mhz di frequenza;
- **Cat5e** (Cat5 enhanced), che segue maggiormente gli standard IEE e supporta fino a 1Gbps, operando sempre su 100Mhz di frequenza;
- **Cat6**, che supporta fino a 10Gbps e opera sui 250Mhz di frequenza;
- **Cat6a**, che supporta fino a 10Gbps, ma con maggiori distanze e sulla frequenza di 500Mhz.

#### 7.6.4 Connettore RJ-45

È la terminazione dei cavi usati da Ethernet. Sono del tipo 8P8C (otto posizioni, otto contatti). Esistono inoltre due schemi di cablaggio, T568A e T568B, che differiscono per un inversione delle coppie 2 e 3. T568B è il prediletto nei sistemi recenti, secondo lo standard ANSI/TIA ((American National Standards Institute e Telecommunications Industry Association).



Pin	Colore	Polarità	Utilizzo
1	bianco/arancione	+	Trasmissione (Tx)
2	arancione	-	Trasmissione (Tx)
3	bianco/verde	+	Ricezione dati (Rx)
4	blu	-	Bidirezionale dati (Bi3)
5	bianco/blu	+	Bidirezionale dati (Bi3)
6	verde	-	Ricezione dati (Rx)
7	bianco/marrone	+	Bidirezionale dati (Bi4)
8	marrone	-	Bidirezionale dati (Bi4)

La polarità opposta nei segnali trasmessi nei cavi twisted pair permette di ridurre il rumore con estrema facilità. Supponiamo che un bit venga trasmesso a 1V in due coppie di cavi con polarità opposta ( $A : +1V$ ,  $B : -1V$ ). Nei sistemi **differenziali**, il segnale sta nella differenza tra le coppie di segnali, e quindi:

$$1V - (-1V) = 1V + 1V = 2V$$

Supponiamo adesso di indurre un rumore di  $+0.5V$  al nostro cavo (e quindi su entrambi i cavi della coppia).

$$(1V + 0.5V) - (0.5V - 1V) = 1.5V - (-0.5V) = 1.5V + 0.5V = 2V$$

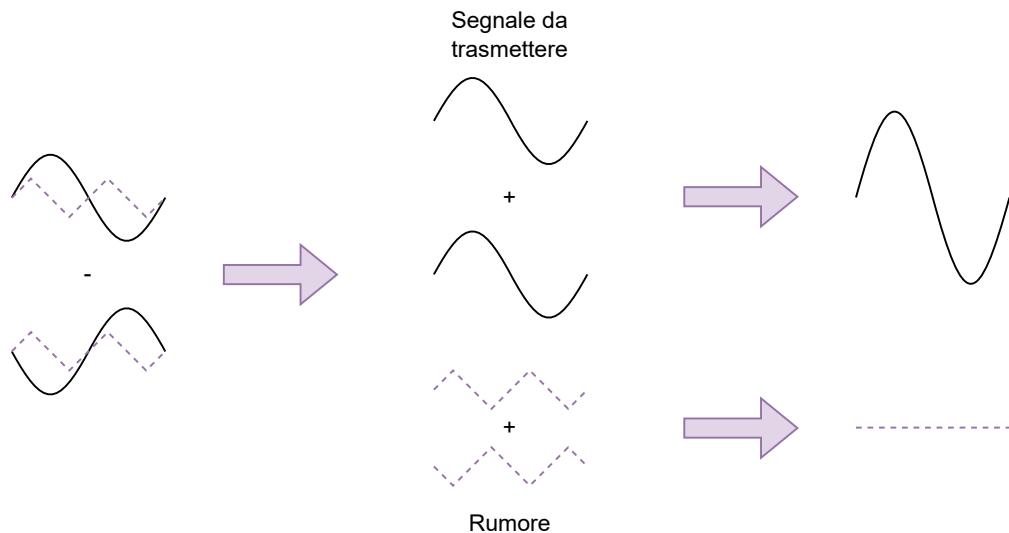


Immagine 7.2: Esempio di cancellazione del rumore

## Distinzione: cavi diretti e cavi incrociati

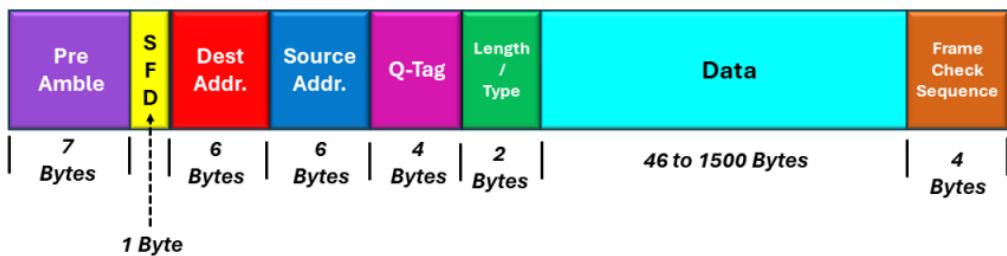
- **Cavo incrociato.**

La disposizione dei doppini è diversa tra le due estremità. È usato per collegare dispositivi simili (router a router).

- **Cavo diretto.**

La disposizione dei doppini è uguale tra le due estremità. È usato per collegare dispositivi diversi (pc a router).

## 7.7 Frame Ethernet



- **Preambolo.**

7 byte a 10101010 per sincronizzare i clock delle schede di rete.

- **Start Frame Delimiter.**

Un byte a 10101011 per interrompere il pattern del preambolo e indicare l'inizio del frame.

- **MAC address destinatario.**

6 byte.

- **MAC address mittente.**

6 byte.

- **Q-Tag/VLAN Tag.**

È il tag presente **esclusivamente nelle frame Ethernet che supportano le VLAN**. Non è sempre presente, in quanto non tutti i frame Ethernet sono *tagged*. 4 byte.

- **Type/length.**

2 byte che identificano il tipo di connessione (EtherType, IPv4, IPv6, ARP,...), se ha valore superiore a 1500, altrimenti, la lunghezza del data field.

- **Data field.**

Minimo 46 byte, massimo 1500 byte. Se il frame da inviare non supera la dimensione minima, si usa un **padding**. Se il frame supera la dimensione massima, si frammenta.

- **Frame check sequence.**

4 byte dedicati al controllo CRC per rilevare (e/o correggere) errori.

### 7.7.1 Codifica di Manchester

È una codifica che nasce per ovviare ad alcune delle criticità relative alla trasmissione che già conosciamo:

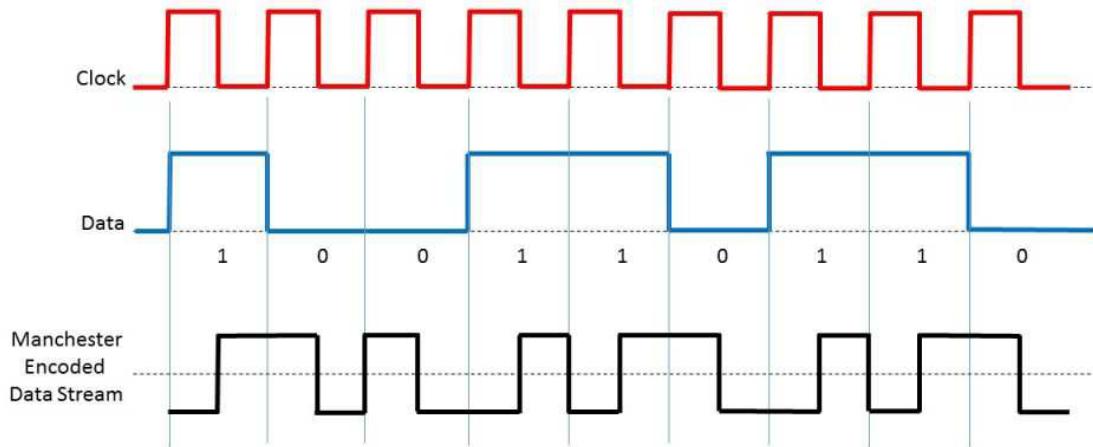
- **Ambiguità delle lunghe sequenze.**

La codifica di Manchester forza delle transizioni obbligatorie quando trasmette dei bit. Diventa più facile riconoscere malfunzionamenti e interruzioni della trasmissione.

- **Sincronizzazione precaria.**

Ogni bit contiene al suo interno una transizione. Non può perdere la sincronizzazione.

Viene realizzata tramite un'operazione di *XOR* tra il segnale di clock e il segnale dati. Ogni bit è codificato come un fronte di salita (1) o un fronte di discesa (0). In questo modo, il *bitstream* e il segnale di clock sono, in un certo senso, trasmessi contemporaneamente.



Tuttavia, questa codifica è influenzata dalla polarità del segnale trasmesso. Se il segnale codificato viene invertito, bisogna invertire anche la sua codifica. Possiamo risolvere il problema usando la codifica di Manchester differenziale.

#### Codifica di Manchester differenziale

Rende la codifica di Manchester insensibile alla polarità. Ogni bit presenta almeno una transizione a metà periodo di clock. Il valore del bit dipenderà dalla presenza o meno di una transizione (in particolare, fronte di salita) all'inizio del periodo di clock.

- Bit 0: transizione.
- Bit 1: assenza di transizione.

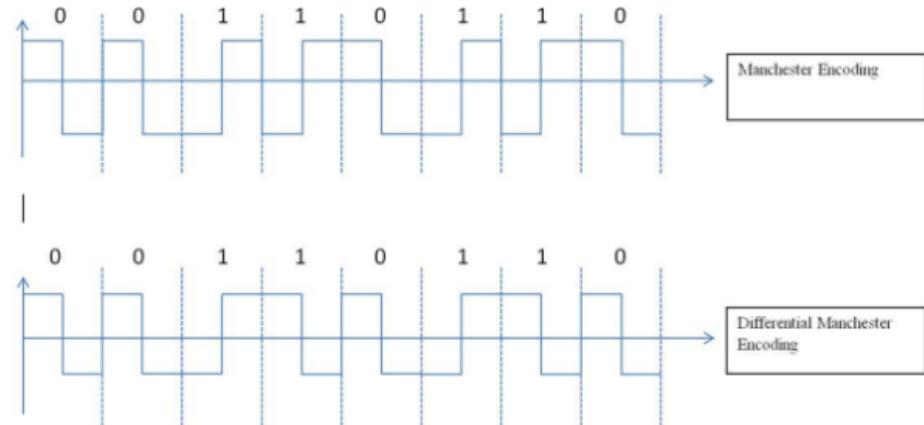


Immagine 7.3: Codifiche di Manchester a confronto

## 7.8 Gigabit Ethernet

È lo standard Ethernet che raggiunge i 1000Mbps, ovvero 1Gbps. Andiamo a scomporre come le scelte prese permettano di raggiungere queste velocità.

**1. Rimossa la codifica 4B5B.**

$100 \rightarrow 125$  Mbps. Rimuoviamo l'overhead del 25% della codifica 4B5B. La sincronizzazione è garantita dalla codifica di Manchester.

**2. 4 doppini di rame usati allo stesso tempo.**

$125 \rightarrow 4 \cdot 125 = 500$  Mbps.

**3. Trasmissione full-duplex.**

500 Mbps full-duplex.

**4. 5 livelli per baud invece di 3.**

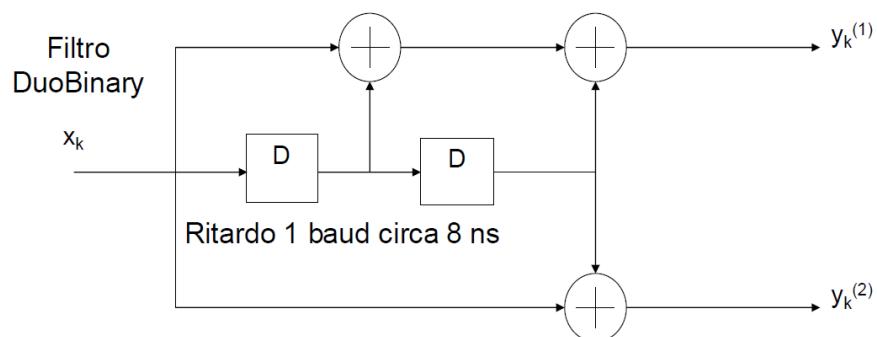
5 livelli su ogni canale di trasmissione. Con 4 canali, otteniamo  $5^4 = 625$  combinazioni, e quindi  $\log_2(625) \approx 9$  bit

**5. Trasmissione finale:** Con  $10^6$  frequenza di simbolo  $125 \cdot 10^6 \cdot 9\text{bit} = 1$  Gbit lordo.

**6. Forward Error Correction.**

### 7.8.1 Filtro DuoBinary

Il seguente filtro, detto DuoBinary, si occupa della codifica.



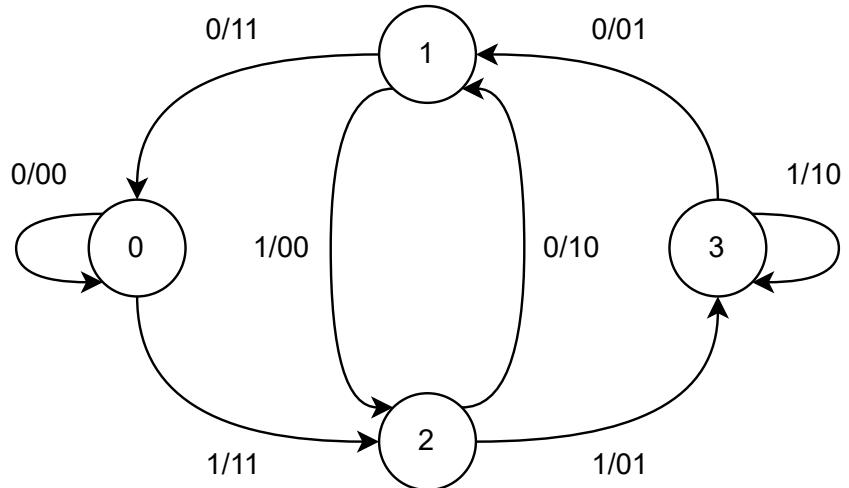
1. Il bit in entrata viene sdoppiato in due segnali.

2. Introduce un ritardo ad uno dei due segnali.

3. Si stringe lo spettro del segnale: **un simbolo viene codificato in due simboli.**

### 7.8.2 Decodifica 4D

La decodifica 4D, detta di Viterbi, viene eseguita tramite un automa a stati finiti.



Ciò che viene fatta, è una **Trellis code modulation**. Raddoppiando il numero di bit per rappresentare un bit d'informazione, gli errori peseranno meno sull'informazione complessiva. Tuttavia è grazie alle transizioni dell'automa limitate che è possibile capire quando è avvenuto un errore: la distanza di Hamming delle stringhe possibili valutate rispetto a quella consegnata, ci permette di stabilire come effettuare la correzione.

## 7.9 Strutture Ethernet

Hub e bridge sono strutture fisiche che permettono di collegare le varie sottoreti. Sono strutture molto diverse, e al giorno d'oggi l'hub è obsoleto. Il bridge è alla base di tutti gli switch moderni.

### 7.9.1 Differenze tra hub e bridge

L'ormai obsoleto hub:

- Lavora principalmente al livello fisico.
- Ripete e rigenera il segnale su tutte le porte.
- Il dominio di collisione tra tutti i dispositivi collegati è unico.
- Broadcast domain unico.

Il bridge, invece:

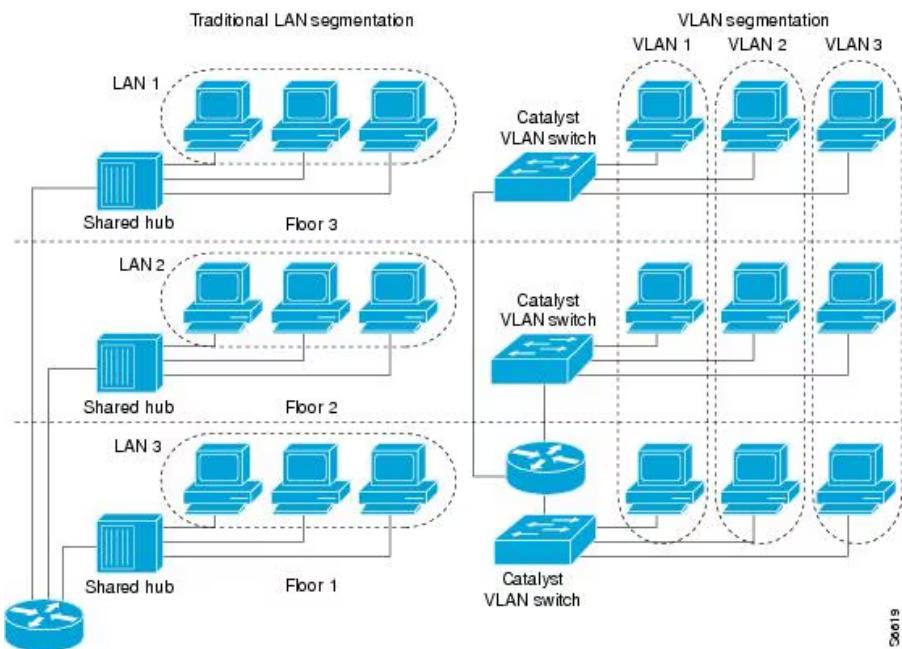
- Lavora a livello data link.
- Filtra e inoltra in base all'indirizzo MAC
- Nel bridge ogni porta crea (idealmente) un dominio di collisione separato.
- Broadcast domain unico.

### 7.9.2 Switch

È un dispositivo che trasmette pacchetti ricevuti a uno o più dispositivi destinatari. È più espandibile in termini di numero di porte, rispetto al bridge, e effettua operazioni di **inoltro** (individuare a quale interfaccia mandare il frame) e **filtraggio** (capire quando scartare un frame). Uno switch contiene al suo interno una **switch table** con record del tipo Indirizzo MAC - Interfaccia - Tempo. Il tempo viene utilizzato per stabilire quando un dispositivo non è più presente nella rete: verrà dedotto in quanto non verranno mandati più pacchetti al suo indirizzo MAC per molto tempo. Gli switch sono full duplex, senza collisioni.

## 7.10 VLAN

Le LAN virtuali permettono di creare delle LAN logiche indipendenti sulla stessa struttura fisica.



Permette di risparmiare su costi, sulla manodopera e sulla complessità dietro il ricablaggio di una rete, per modificarne la struttura logica. Sono quindi delle vere e proprie LAN logiche separate che risiedono sopra la struttura fisica. Le LAN virtuali rimappano le tabelle di forward del bridge senza modificare la cablatura. Una porta può essere associata a una e una sola VLAN, altrimenti si riscontrerebbero collisioni.

### 7.10.1 Scopo delle VLAN

- **Riutilizzo dell'architettura di partenza** con conseguente risparmio.
- **Flessibilità.**  
Molto più facile spostare gli utenti tra le VLAN.
- **Prestazioni migliori.**  
Isolando il broadcasting nelle VLAN,
- **Sicurezza.**  
È semplice isolare dei sistemi tramite le VLAN.

### 7.10.2 VLAN basata su porte

Lo switch può essere **partizionato logicamente** in più insiemi di porte. Le porte possono essere associate a delle VLAN. Nell'header dei frame DLL è presente un identificatore di VLAN.

- **Tag Protocol Identifier.**  
16 bit.
- **Priority.**  
Identifica il livello di priorità. 3 bit.
- **Canonical Format Identifier.**  
1 bit che specifica se il frame è untagged. Il flag a 0 indica che il meccanismo di VLAN non è supportato.
- **Virtual Lan Identifier.**  
12 bit.

### 7.10.3 LAN vs Dispositivi Legacy

I dispositivi legacy che non supportano le VLAN, ignorano tutti i frame tagged VLAN a loro in arrivo.

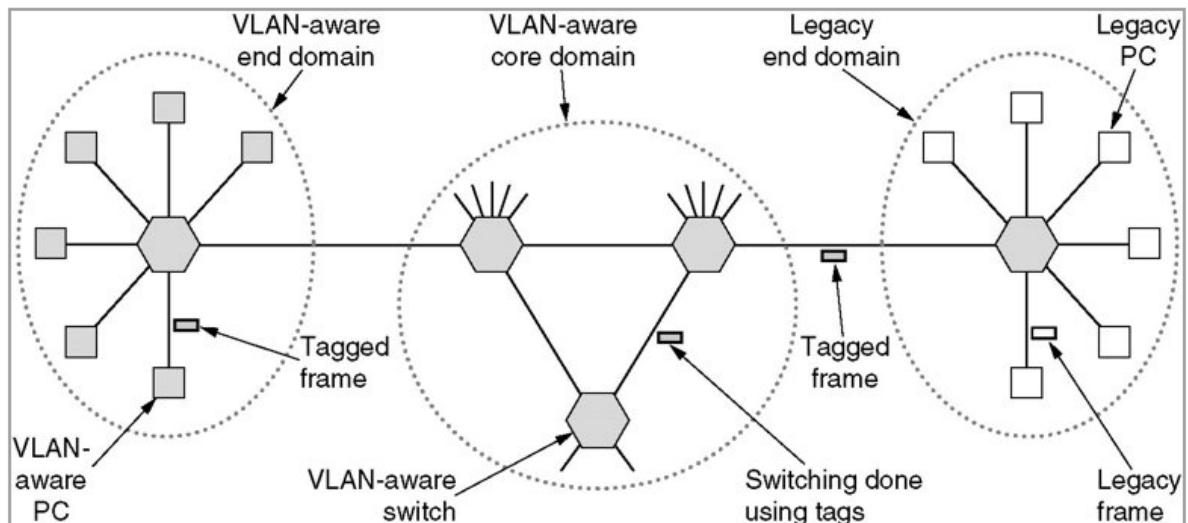
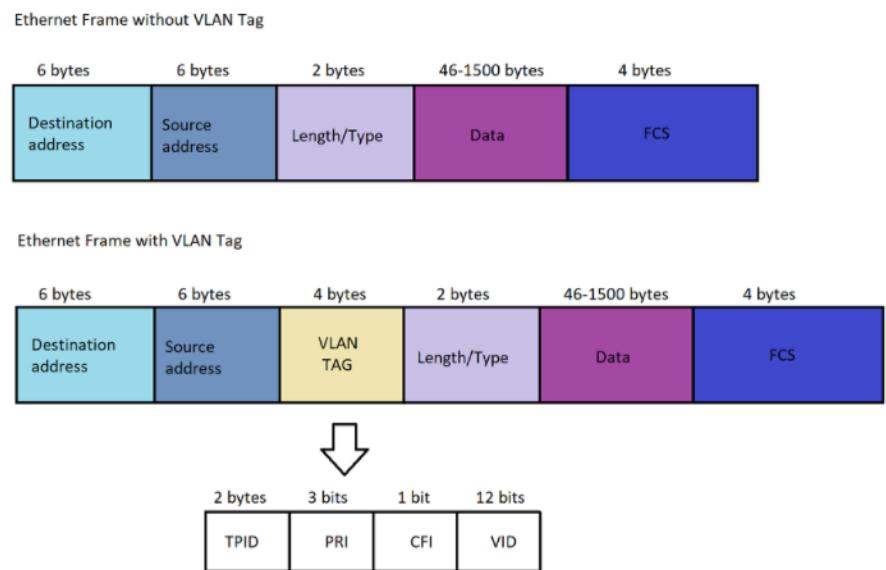


Immagine 7.4: Una rete con dispositivi VLAN e legacy