

Appunti di Reti Informatiche

Anno Accademico 2021-2022

D.B.

I seguenti appunti non sono appunti ufficiali del corso di Reti Informatiche e non sono stati revisionati da alcun docente.

Nelle pagine seguenti troverete degli appunti riepilogativi degli argomenti trattati a lezione dal docente, sul libro di testo consigliato e sulle slide fornite

Tutte le immagini presenti sono soggette a Copyright e sono prese dalle slide messe a disposizione da Prof. Anastasi e Prof. Pistolesi oltre che dal libro di testo: "Computer Networking: A Top-Down Approach" di J. Kurose e K. Ross.

Buona lettura!

Sommario

Introduzione	9
Cos'è Internet?	9
Protocolli	9
Reti periferiche	10
Hosts	10
Tipi di reti di accesso	10
Tipi di reti di accesso residenziali	10
Reti di accesso istituzionali.....	11
<i>Reti wired: Ethernet</i>	11
<i>Reti wireless: WiFi standard IEEE 802.11</i>	11
Reti di accesso pubbliche	11
Link di collegamento: wired	12
Link di collegamento: wireless.....	12
Network Core	13
Packet Switching.....	13
Internet: una rete di reti.....	14
Ritardi e perdita di pacchetti nelle reti packet switching.....	14
Ritardo totale nella trasmissione end-to-end e throughput	15
Layered structure	17
Incapsulamento	18
Layer 5: Application Layer	19
Client-Server	19
Peer-to-Peer	19
Hybrid Architetture	20
Processi comunicanti.....	20
Servizi disponibili per una applicazione.....	21
Servizi di Trasporto forniti da Internet	22
Protocolli livello applicazione	23
HTTP.....	23
http con connessioni NON persistenti.....	23
http con connessioni persistenti	23
Messaggio di richiesta HTTP.....	24
Messaggio di risposta HTTP	24
I cookie.....	25
Web Caching.....	26

FTP	27
Web mail.....	28
SMTP.....	28
MIME (Multipurpose Internet Mail Extensions).....	29
POP3 & IMAP (protocolli di accesso).....	29
DNS	30
Funzionamento: approccio iterativo	32
Funzionamento: approccio ricorsivo	32
Formato dei record nei DNS Server.....	33
Cosa significa registrare un nuovo dominio?	33
DNS e sicurezza.....	33
Applicazioni P2P	34
Ricerca dei contenuti.....	35
Distribuzione dei file.....	37
Layer 1: Physical	39
Layer 2: Data-Link.....	39
Servizi messi a disposizione.....	39
Error Detection.....	40
Parity check	40
Checksum.....	41
Cyclic Redundancy Check (CRC).....	41
Error correction	41
Bit di parità bidimensionale.....	41
Reliable Data Transfer (Reti a Connessione diretta)	41
RdT 1.0.....	41
RdT 2.0.....	41
RdT 2.1.....	42
RdT 3.0.....	43
Pipeline	44
PPP: Point to Point Protocol	46
Multiple Access.....	47
Mac Protocols a Partizionamento di risorse: TDMA	48
MAC Protocols a <i>Partizionamento</i> di risorse: FDMA.....	48
MAC Protocols a <i>Partizionamento</i> di risorse: CDMA.....	48
MAC Protocols Random Access.....	48
MAC Protocols Protocolli a turno	50

Indirizzamento fisico.....	50
Local Area Network (LAN)	51
Ethernet.....	51
Frame Ethernet.....	52
Algoritmo CSMA/CD per Ethernet.....	52
Reti basate su packet switching	53
Switch utilizzati in Ethernet.....	53
Proprietà dello Switched Ethernet	53
VLANs.....	54
Wide Area Network Switched Ethernet	55
ATM	55
Layer 3: Network (o IP level)	57
Internetworking: introduzione	57
Routing vs Forwarding.....	57
Internet (<i>datagram</i>) vs ATM (<i>virtual circuit</i>)	58
Il router.....	58
Funzionalità delle porte di ingresso	58
Logica di commutazione	59
Funzionalità delle porte di uscite	59
Protocollo IPv4.....	59
Formato dei datagram.....	60
Incapsulamento	61
Indirizzamento IPv4	62
Ottenere un indirizzo IP: organizzazione.....	63
Ottenere un indirizzo IP: host.....	63
Dynamic Host Configuration Protocol	63
Indirizzi riservati	64
Network Address Translation	65
Forwarding.....	66
Address Resolution Protocol: ARP	67
ICMP	67
Routing	68
Link State Algorithm	68
Distance Vector Algorithms.....	69
Routing ereditario.....	71
IGP (Interior Gateway Protocols).....	72

BGP (Border Gateway Protocol), algortimo inter-AS	73
IPv6	76
Formato del datagram IPv6.....	76
Transizione dall'IPv4 all'IPv6	76
Layer 4: Transport.....	77
Multiplexing e Demultiplexing	77
UDP	78
TCP.....	78
Struttura del segmento TCP	79
Stabilire una connessione.....	79
Chiusura di una connessione.....	79
Reliable Data Transfer	80
TCP flow control	83
Congestion Control.....	84
Network Security	87
I rischi della rete	87
Sicurezza	87
Confidenzialità e crittografia	88
Crittografia a chiave simmetrica.....	88
Crittografia chiave pubblica.....	89
Integrità del messaggio	90
Message Authentication Code	90
La firma digitale per assicurare l'autenticazione.....	91
Proteggersi da attacchi record and playback	92
Dove implementare i protocolli di sicurezza	92
Secure Email e PGP	93
SSL (Secure Sockets Layer).....	94
Sicurezza a livello rete: IPsec	95
Firewall	97
Wireless e Mobile Network	98
Tipi di reti Wireless	98
Classificazione in base all'area di copertura.....	98
Clasasificación in base all'infrastruttura e al numero di hop	98
Problema del nodo nascosto	99
Problema del nodo esposto.....	99
WiFi: standard IEEE 802.11.....	99

CSMA/CA (Collision Avoidance).....	100
Virtual Carrier Sensing	101
Frame WiFi.....	102
Mobilità	102
Power Management.....	102
Rete cellulare.....	103
Addressing and routing per utenti mobili	103
Mobile IP.....	105
Agent discovery	105
Registrazione dell'host mobile ad una foreign network	106
Impatto della mobilità	106
Reti senza infrastrutture: ad hoc network.	107
Bluetooth.....	107
Laboratorio	108
Indirizzamento riepilogo	108
Mostrare e configurare interfacce: comando ip	108
Gateway.....	108
DNS	109
DNS statico.....	109
DNS Server	109
DHCP	109
DHCP Server.....	109
Client DHCP.....	109
Test di connettività.....	110
Ping	110
Traceroute	110
Firewall	111
Firewall a filtraggio di pacchetto stateless	111
Netfilter e iptables.....	112
Ripasso sul C	114
Allocazione e rilascio della memoria.....	114
String.....	114
File	114
Comunicazione tra processi: socket.....	115
Creazione del socket	115
Struct per gli indirizzi.....	116

Conversione dell'endiannes	116
Conversione del formato dell'indirizzo	116
Creazione ed inizializzazione di un socket.....	117
Assegnare indirizzo a un socket (lato server).....	117
Mettere in ascolto un socket	117
Programmazione distribuita su Server.....	118
Programmazione distribuita su Client.....	119
Scambio di dati.....	119
Send().....	120
Recv()	120
Close()	120
Server Concorrenti.....	121
fork()	121
Socket non bloccanti.....	123
IO multiplexing.....	124
Select()	125
Socket UDP.....	127
Sendto()	127
Recvfrom()	127
Socket UDP "connesso"	127
Protocolli Text e Binary	128
Protocolli text.....	128
Protocolli binary.....	128
Server Apache.....	129
Configurazione.....	129
Direttive globali	130
Direttivi per i siti web e virtual host	130
Multi-Processing Modules.....	131

Introduzione

Cos'è Internet?

La definizione di Internet dipende dal punto di vista dell'osservatore.

Definizione Internet (1):

Un sistema che collega milioni di hosts o end systems sui quali girano processi applicativi che operano in rete; questi computer si collegano ad internet attraverso reti di accesso e attraverso link di comunicazioni con tecnologie diverse (fibra, cavo, radio, satellite).

Poiché sono presenti reti diverse sono necessari dei router che servono a collegare queste reti (un po' come degli interpreti).

Definizione Internet (2):

Internet è una rete di reti

Definizione Internet (3):

Se questa definizione fosse chiesta ad un programmatore sarebbe tipo: “Internet offre servizi di alle applicazioni (ad esempio posta elettronica, file sharing, social network, ecc)”.

Queste applicazioni sono applicazioni distribuite in quanto riguardano scambio di dati tra più hosts i quali si attaccano a Internet fornendo Application Programming Interface (**API**) (un set di regole che internet deve seguire per poter consegnare dati).

Notiamo come al giorno d'oggi gli hosts non siano solo computer ma siano “cose”: basti pensare agli orologi, ai frigoriferi, alle lavatrici ecc connesse ad internet (IoT).

Protocolli

Definizione Protocollo

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

Come si può intuire i protocolli sono necessari affinché le comunicazioni possano avvenire in quanto stabiliscono delle regole di comunicazione.

Esempi di protocolli sono: TCP, IP, HTTP, ecc.

Chi definisce questi protocolli?

IETF (Internet Engineering Task Force, o la IEEE, o altri enti) attraverso RFC (Request for comments): documento che viene scritto da chi sviluppa un'applicazione, e sul quale la community inizia a suggerire modifiche o migliorie, alla fine delle varie modifiche viene definito come standard.

Un Internet Protocol è molto simile ad un protocollo umano, facciamo un parallelismo:

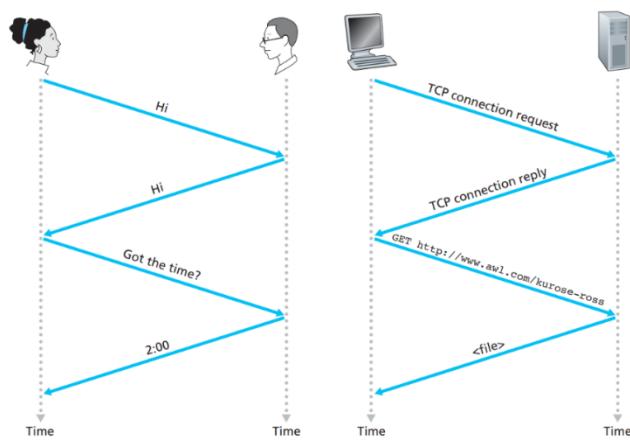


Figure 1.2 • A human protocol and a computer network protocol

Reti periferiche

In queste reti possiamo trovare host molto eterogenei tra loro (l'IoT come sappiamo si sta espandendo con una velocità impressionante) i quali sono collegati ad Internet attraverso delle reti di accesso e che, proprio per la loro diversità, possono essere collegati a tali reti in modi differenti (**wired o wireless**).

Le reti di accesso poi sono collegati alla **core network** formata da router interconnessi tra loro.

Hosts

Possono essere, ad esempio, computer (mainframe, pc, smartphone) sui quali girano applicazioni.

Gli hosts si trovano solo alla periferia della rete. E per quanto diverse possono essere queste applicazioni sono sempre strutturate secondo il modello Client-Server o secondo il modello P2P.

Client-Server: sul server gira un servizio (un processo), il quale viene messo a disposizione dei client (i processi client) i quali lo richiedono. Il client e server sono processi, per estensione vengono chiamati client e server anche le macchine dove girano questi processi.

Se i client sono solitamente gli host delle reti periferiche, i server al contrario non solitamente non è una macchina qualsiasi ma deve essere una macchina collegata ad un link di collegamento con banda elevata, deve avere molte risorse (molta potenza di calcolo, spazio di archiviazione, ecc).

Peer-to-Peer (P2P): non c'è più la distinzione netta tra client e server (non abbiamo un processo sempre client o sempre server) un peer può essere una volta client e una volta server, talvolta come nel caso di BitTorrent può essere tutti e due contemporaneamente.

Tipi di reti di accesso

Esistono diversi tipi di reti di accesso:

- rete di accesso residenziale (la rete domestica)
- rete di accesso istituzionale (ad esempio la rete universitaria)
- reti di accesso mobile

A noi di queste reti interessano per lo più la larghezza di banda, il bitrate (bits per second) e se la banda è dedicata o condivisa.

Tipi di reti di accesso residenziali

Dial-Up Modem

Non esistono più nei paesi sviluppati.

Utilizza la rete telefonica.

In questa tipologia di rete non si poteva navigare su internet e usare il telefono insieme (se ti fossi collegato a Internet non avresti potuto ricevere chiamate) e accesso a internet fino a 56Kbps.

DSL (Digital Subscriber Line)

Banda più larga della Dial-Up anche se usava sempre

l'infrastruttura di rete.

Upload: 1.8-2.5 Mbps / Downloads: 12-24Mbps. (Circa)

Linea dedicata al telefono, quindi si può navigare e telefonare in contemporanea.

Solitamente i contratti stipulati con gli ISP (Internet Service Provider) sono asimmetrici: ADSL, solitamente il client sta all'interno di reti di accesso per cui necessita di una larga più larga in download piuttosto che in upload.

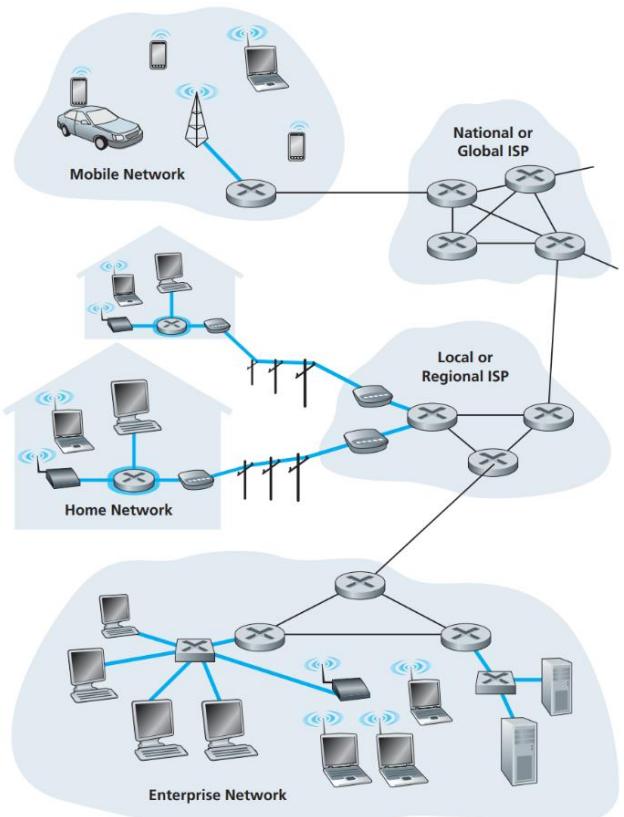


Figure 1.4 • Access networks

Cable Modem

Non usa l'infrastruttura telefonica ma la rete via cavo. (Molto usato negli USA).

Risulta essere un collegamento di tipo condiviso (non come l'adsl), uno stesso cavo coassiale riesce a coprire centinaia di abitazioni.

Anch'esso asimmetrico, ma non tanto quanto l'ADSL

Fiber to the Home (FTTH)

Linea dedicata che non usa l'infrastruttura telefonica, si hanno i cavi in fibra ottica che arriva direttamente a casa: teoricamente banda massima in download fino a Gbps.

Reti di accesso istituzionali

Le stesse tecnologie stanno venendo usate anche nelle reti residenziali per due motivi: costi ridotti e incremento dei dispositivi connessi ad internet all'interno di un'abitazione in modo eterogeno (Pc desktop possono necessitare del cavo Ethernet, smartphone necessitano della connessione WiFi).

Reti wired: Ethernet

In queste reti gli hosts sono collegati ad uno switch attraverso un doppino in ethernet, avendo così un canale dedicato nel quale dunque, il numero di dispositivi connessi alla rete non influisce sulle prestazioni percepite dall'host.

Lo switch viene poi collegato al router che permette l'accesso alla core Network e quindi a Internet.

Reti wireless: WiFi standard IEEE 802.11

Non è un canale dedicato come l'ethernet (il numero di utenti influisce sulla qualità della rete, sulla rete ethernet non influisce).

È necessario un access point al quale si collegano i dispositivi, il quale è collegato direttamente al modem, solitamente attraverso un cavo ethernet.

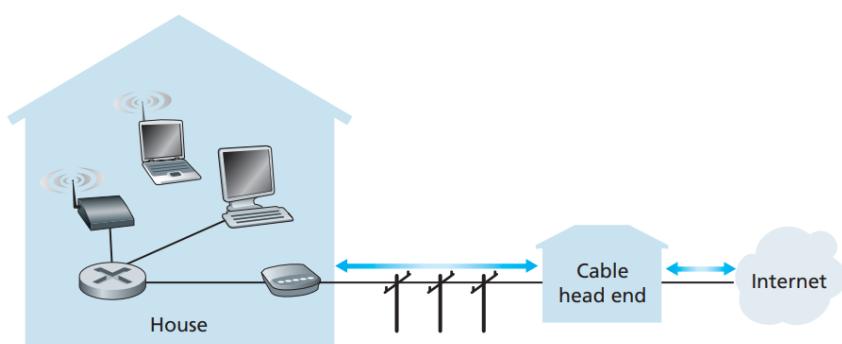


Figure 1.9 ♦ A typical home network

Come si diceva, oggi in molte abitazioni la rete di accesso è simile ad una rete istituzionale ma in scala ridotta, come si può notare dalla figura accanto.

Reti di accesso pubbliche

Utilizzano canali wireless come i dati mobili (4G, LTE) dei telefoni o WiFi pubblici.

Link di collegamento: wired

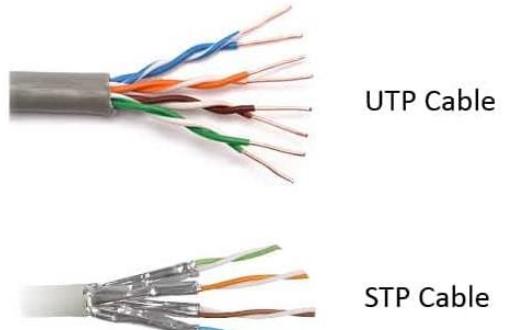
Twisted Pair

In italiano è detto doppino in rame, è cavo meno costoso e più diffuso.

Un twisted pair consiste in **due cavi in rame isolati intrecciati a spirale** tra loro così da ridurre l'interferenza elettrica causata dal passaggio di corrente nei cavi.

Esistono diversi tipi di twisted pair:

- **UTP**, sono i cavi non schermati
- **STP**, sono i cavi schermati, ovviamente essendo schermati sono meno soggetti alle interferenze dovute alle interferenze elettriche.



Questi possono essere di diversa categoria: in media un UTP è in grado di trasmettere **da 10Mbps a 10Gbps** (i più moderni).

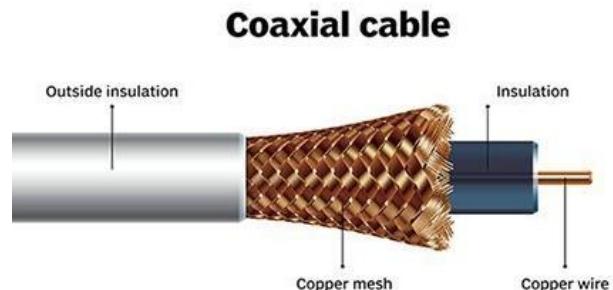
I problemi di questi cavi sono che:

- Sono soggetti ad interferenze elettriche;
- Non possono coprire grandi distanze, ma circa un centinaio di metri poiché il segnale si degrada e perde potenza, il che gli rende molto utilizzati nelle reti LAN (Local Area Network).

Cavo coassiale

Come il precedente trasmette impulsi elettrici ed utilizza quindi un conduttore, ma in questo caso sono due conduttori concentrici e paralleli.

Come notiamo il secondo conduttore è una maglia di rame, che funge da isolante e da conduttore "di ritorno" in quanto il cavo è bidirezionale.



Fibra Ottica

In questo caso il segnale propagato non è più un segnale elettromagnetico, ma un segnale luminoso. Ogni impulso è un bit. Questo mezzo fisico permette di avere valori di **bitrate molto molto più elevati (10/100Gbps)** ed un tasso di errore più basso (di parecchio) rispetto agli altri mezzi wired.

Anche qui sono presenti impurità (e non più disturbi elettromagnetici).

Le fibre ottiche riescono a coprire distanze maggiori e spesso risultano essere la spina dorsale delle infrastrutture MAN e WAN.

Link di collegamento: wireless

Con questi mezzi fisici il **segnale viene propagato nello spettro elettromagnetico**, il che comporta dei problemi legati all'ambiente: la riflessione del segnale, l'ostruzione dovuta agli oggetti (una porta, un muro, ecc), interferenze dovute all'ambiente esterno.

Una domanda sorge spontanea: "Cos'è meglio: wired o wireless?" Dipende, dalle necessità e dai costi.

Network Core

Il nucleo della rete presenta una **topologia a maglia**, costituita da **router interconnessi tra di loro**.

La copertura geografica può essere molto ampia (es: regionale, nazionale).

Esistono due tecniche di trasmissione delle informazioni:

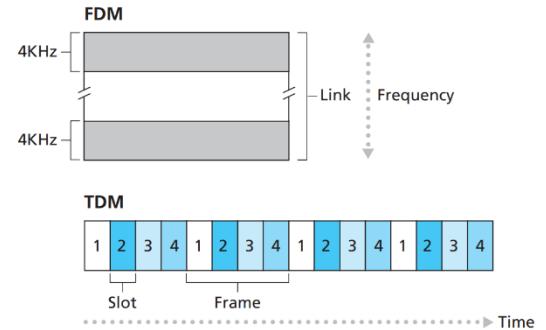
- **circuit-switching** (usata rete telefonica tradizionale):

Si stabilisce un circuito tra i due dispositivi comunicanti (tra il chiamante e il chiamato). Questo circuito dura per tutta la durata della comunicazione. Inizialmente questo circuito era fisico (centraliniste → centraline elettroniche).

Le prestazioni sono garantite (il circuito è dedicato solo ai due utenti).

Come si ottiene un circuito? Si utilizza la *suddivisione in frequenza* (ad esempio è la tecnica utilizzata dalla radio): basta che il ricevitore si sintonizzi sulla frequenza del trasmettitore. Oppure si *suddivide in base al tempo*.

Questo approccio va bene quando è presente un flusso continuo di informazioni, ad esempio nelle telefonate.



- **packet-switching** (quella usata attualmente e che si è sempre usata su Internet):

In questo caso mette un **link a disposizione** non più a una sola coppia di utenti ma a **più coppie di utenti** per convogliare su quel link il traffico generato da più sorgenti.

In questa maniera si risparmia, ma non è molto adeguato per trasmettere voce e video, ma comunque si può usare.

Questo approccio va bene quando il flusso dei dati è intermittente (basti pensare al web dove sono presenti i thinking-time, quando l'utente naviga ma non genera traffico, ad esempio sta leggendo una pagina web).

Packet Switching

I messaggi in una rete possono contenere qualsiasi tipo di informazione siano essi dati o parti del protocollo come messaggi di handshake.

Nella rete i messaggi (lunghi) si suddividono in pacchetti i quali vengono "smistati" dai router e dagli switch.

Store And Forward Trasmission: *in questo tipo di trasmissione il pacchetto, prima di essere rispedito dal router, deve essere ricevuto completamente.*

Questo porta inesorabilmente a dei rallentamenti; infatti, basti pensare a un collegamento a con un solo router tra due dispositivi dove se ci sono da mandare L bit su una linea che trasmette R bit al secondo, il tempo di trasmissione trascurando i ritardi, è di $2L/R$ (L/R da mittente a router + L/R da router a destinatario).

Ad ogni router sono attaccati più collegamenti e per ognuno di essi è presente una **output queue**, la quale gioca un ruolo fondamentale nella trasmissione dei pacchetti.

Internet: una rete di reti

Internet è una struttura organizzata: sappiamo che Internet è una rete di reti, ma queste reti non sono tutte uguali; infatti, ne esistono di più grandi e di più piccole, di più o meno importanti. **Questo porta inesorabilmente ad una struttura gerarchica.**

Definizione ISP

Internet Service Provider, ovvero il fornitore di accesso internet e che si occupa di gestire la rete.

Esistono diversi tipi di ISP, di diverso livello e diverse dimensioni.

I primi. I più grandi sono gli ISP di livello 1, i quali sono interconnessi tra loro e offrono i loro servizi agli ISP di livello 2, i quali a loro volta lo offrono agli ISP di livello inferiore e così via, fino ad arrivare alle reti di accesso.

È da notare come gli ISP non siano necessariamente compagnie di telecomunicazioni, ma potrebbero essere università o altri istituti.

Ritardi e perdita di pacchetti nelle reti packet switching

Idealmente ci piacerebbe che Internet fosse in grado di trasferire grandi quantità di dati tra due o più hosts instantaneamente e senza perdita di dati. Ma questo per ragioni spazio-temporali non è possibile.

Come sappiamo i dati (o pacchetti) viaggiano da un nodo ad un altro, e ad ogni nodo possono essere presenti dei ritardi come possiamo vedere dalla figura e che andremo ad analizzare nel seguito.

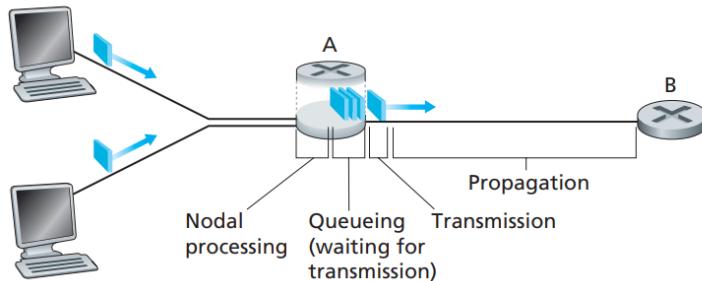


Figure 1.16 ◆ The nodal delay at router A

Presupposto: sul collegamento può essere trasmesso un pacchetto alla volta.

Nodal processing delay (o ritardo di elaborazione)

Quando un pacchetto arriva al router per prima cosa viene memorizzato, poi viene guardato l'ip a cui è destinato il pacchetto, viene consultata la **forwarding table** e si decide in quale link di uscita instradare il pacchetto e lo si accoda nella coda di uscita.

Tutte queste operazioni generano ritardo: **ritardo di elaborazione**, il quale è nell'ordine dei micro-secondi.

Definizione Forwarding table:

tabella nella quale vengono salvate quali sono le reti raggiungibili e attraverso quali delle reti direttamente collegate. Questa tabella può essere popolata staticamente dall'amministratore (priva di aggiornamenti e scarsa tolleranza ai guasti) oppure dai protocolli di routing dinamico.

Ritardo di accodamento

Una volta che il pacchetto viene accodato non è detto che questo sia il primo della coda; perciò, questo dovrà attendere prima di essere trasmesso. Si genera così il ritardo di accodamento, ma **quanto ritardo?**

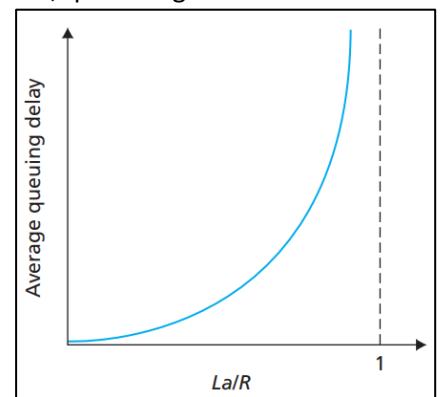
Dipende dal numero di pacchetti in coda, perciò questa è una componente variabile.

Supponiamo che a si il numero medio di pacchetti che arrivano ogni secondo, che L sia la lunghezza in bit di ogni pacchetto e che R sia la velocità di trasmissione.

Chiamiamo La/R intensità di traffico, il quale è un importante parametro ma che non basta a caratterizzare il ritardo di accodamento.

Se $La/R > 1$ significa che arrivano più bit di quanti se ne riescano a trasmettere, questo significa che la coda tende ad aumentare verso l'infinito (il che ovviamente è un male, ma è anche impossibile). Se $La/R \leq 1$, significa che il ritardo di accodamento è considerabile, ovviamente più ci si avvicina a 1 più il ritardo aumenta. L'ideale è che l'intensità di traffico non sia mai maggiore di 1.

Facciamo notare anche una cosa molto importante: abbiamo detto che **se l'intensità di traffico cresce, la coda (o meglio il buffer) cresce verso l'infinito**, ma questo è impossibile in quanto il buffer, essendo una memoria, **hai dei limiti fisici. Quindi cosa succede? Si perdono dei pacchetti**. Quali? O l'ultimo arrivato, o uno che era in coda secondo qualche criterio.



Possiamo fare un parallelismo tra questo ritardo e il casello autostradale, dove ogni volta passa un solo veicolo per volta e gli altri devono aspettare in coda, la quale può crescere fino all'infinito (nessuna macchina verrà persa o buttata fuori dalla strada però).

Ritardo di trasmissione

Una volta arrivato nella testa della coda: il pacchetto viene trasmesso (ovvero i bit vengono codificati in impulsi legati al mezzo trasmissivo, quindi siano essi impulsi luminosi o elettrici ad esempio).

Il ritardo di trasmissione dipende dalla grandezza del pacchetto e dalla velocità di trasmissione del mezzo. Supposta L la lunghezza del pacchetto in bit ed R la velocità di trasmissione, L/R è il ritardo di trasmissione.

Ritardo di propagazione

È il tempo necessario al segnale per propagarsi ed arrivare da un'estremità all'altra del collegamento.

Non è da confondere con il ritardo di trasmissione.

Il ritardo di propagazione dipende da due fattori: dalla velocità di propagazione del segnale sul mezzo trasmissivo e dalla lunghezza del mezzo trasmissivo (distanza tra due nodi).

Il ritardo di propagazione può essere calcolato a priori.

Ritardo totale nella trasmissione end-to-end e throughput

Per calcolare il ritardo totale sperimentato da un pacchetto durante il suo tragitto, supposto vi siano $N-1$ router:

$$\sum_{i=1}^N d_i$$

Dove d_i è il ritardo totale sperimentato su ogni router, ed è dato dalla somma di tutti i ritardi (elaborazione, accodamento, trasmissione, propagazione).

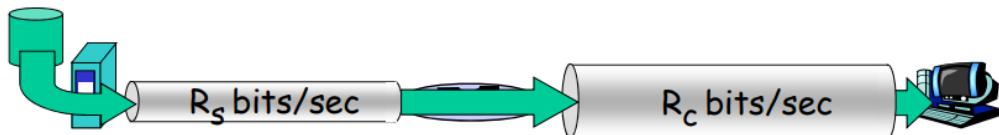
Definizione Throughput: è il numero di bit che si riesce a far passare attraverso una rete e far giungere a destinazione. Si misura in bps.

Attenzione: nonostante l'unità di misura sia la stessa, il throughput è diverso dal bitrate, il primo ci dice quanti bit passano effettivamente attraverso quella rete (o attraverso quel link), il secondo ci dice quanti ne possono passare al massimo.

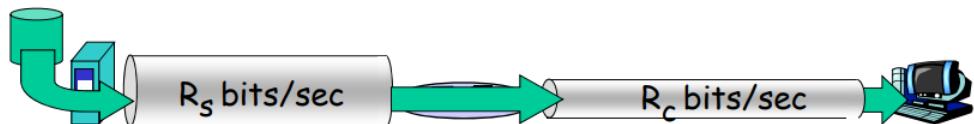
Perché c'è questa differenza? Perché i collegamenti alle reti (siano esse di accesso, istituzionali o pubbliche) non sono tutti uguali, hanno bitrate diversi e quindi si vengono a creare dei **colli di bottiglia**.

Supponiamo che due host collegati abbiano rispettivamente (per il proprio link di accesso) R_s e R_c bps di bitrate. Quale sarà il throughput?

Se $R_s < R_c$



Se $R_s > R_c$



È evidente che il throughput sia il minimo bitrate tra i due collegamenti. Questo perché dal “tubo” più piccolo non può “uscire più acqua” di quanta riesca a contenerla; in termini di rete: il link con il minore rate non può inoltrare pacchetti più velocemente di così.

E se ci fosse internet di mezzo? Potrebbe essere quello il collo di bottiglia?

Teoricamente sì, nella pratica non lo è quasi mai, ma lo sono invece i link delle reti di accesso.

Layered structure

Quando i messaggi vengono trasmessi da un host all'altro si ha come l'apparenza che essi navighino in maniera orizzontale, da applicazione ad applicazione. Facciamo un esempio: è come quando viene mandata una lettera da una persona A ad una persona B, quando la persona B lo riceve, vede solo il contenuto e il mittente. Un altro esempio può essere ad esempio il messaggio arrivato su Whatsapp: sappiamo solo il cosa e il chi, come se il viaggio del messaggio stesso fosse stato da smartphone a smartphone senza passare da altre parti.

In realtà non è così: i messaggi percorrono un percorso anche verticale.

Facciamo un parallelismo con un viaggio in aereo, quando si viaggia non si parte da una destinazione d istantaneamente e magicamente si arriva nell'altra, ma si passa attraverso il deposito bagagli, il gate, l'aereo stesso, il volo, per poi all'arrivo fare il percorso inverso.

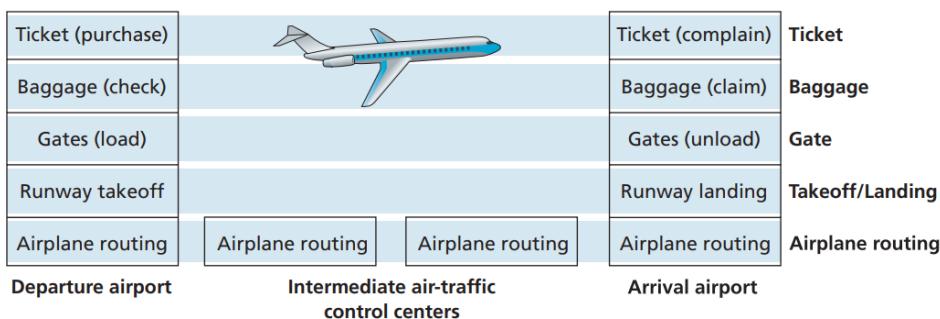
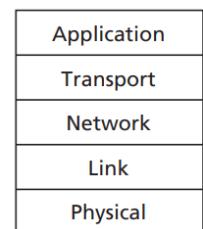


Figure 1.22 ♦ Horizontal layering of airline functionality

Anche i messaggi che viaggiano su internet attraversano un percorso analogo in quanto Internet possiede una struttura a 5 livelli.

Piccola nota storica: questo è uno dei tanti modelli di internet, si potrebbe strutturare su meno o su più livelli. Questo è il modello che si è affermato. Un altro modello, sviluppato dalla OSI, è il modello ISO-OSI, che prevede tra il livello trasporto e il livello applicativo la presenza di altri due livelli: session e presentation.



a. Five-layer Internet protocol stack

- **Livello applicazione:** è la sede delle applicazioni di rete e dei relativi protocolli. Per quanto riguarda Internet, tale livello include molti protocolli, quali HTTP e FTP.
- **Livello Trasporto:** in Internet troviamo due protocolli di trasporto: TCP e UDP.
- **Livello di Rete:** comprende il famoso protocollo IP, che definisce i campi dei datagrammi e come i sistemi periferici e i router agiscono su tali campi. **Esiste un solo protocollo IP**; tutti gli apparati di Internet che presentano un livello di rete lo devono supportare.
- **Livello di collegamento:** si occupa del trasferimento dei pacchetti tra elementi della rete vicini. Esempio di protocollo è il protocollo Ethernet.
- **Livello fisico:** si occupa del trasferimento dei bit “sul cavo”, o per meglio dire: sul mezzo di comunicazione.

Incapsulamento

Quando il messaggio passa da ognuno di questi livelli, ad esso vengono aggiunte nuove informazioni riguardanti i protocolli di quel determinato livello, e che riguardano solo i dispositivi che operano a quel livello.

In sostanza il pacchetto viene incapsulato e ad ogni livello vengono aggiunte delle intestazioni (headers).

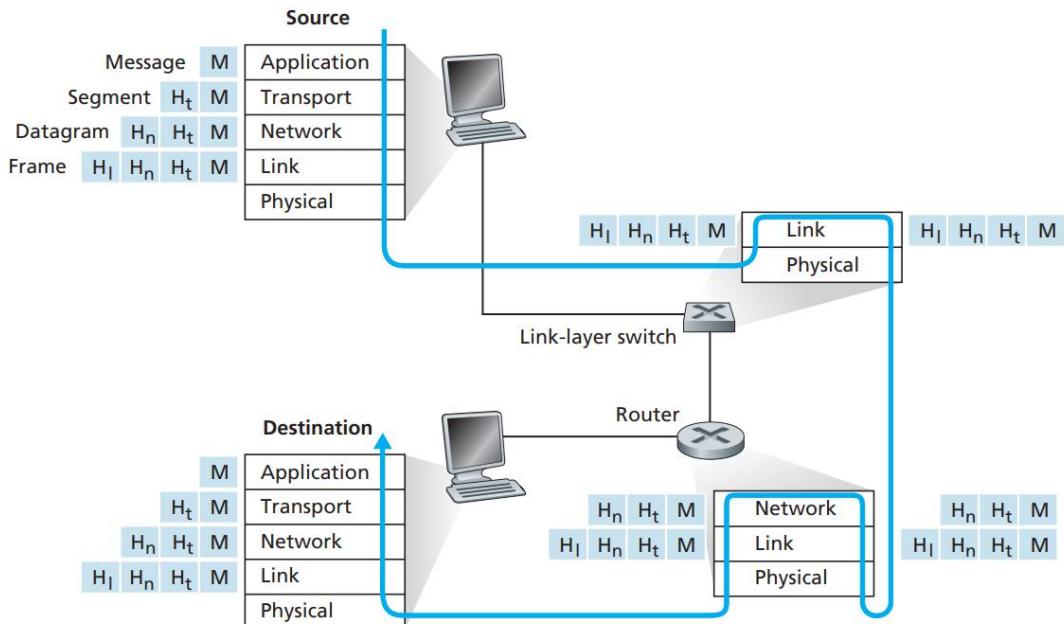


Figure 1.24 Hosts, routers, and link-layer switches; each contains a different set of layers, reflecting their differences in functionality

Come si può notare dalla figura il messaggio parte dal livello applicazione per poi scendere, aggiungendovi nuove informazioni, le quali vengono trasmesse per passare (solitamente) attraverso switch (che lavora al livello 2) e router (che lavora al livello 3), i quali “spacchettano” il pacchetto, andando a leggere (e a togliere temporaneamente) l’intestazione dei vari livelli di interesse, per poi riaggiungere gli headers tolti ma con nuove informazioni. Una volta che il pacchetto arriva al destinatario questo viene spacchettato fino ad arrivare al livello applicativo dove finalmente troverà il massaggio.

Ad ogni livello, il nome dato al pacchetto è diverso:

- Segment → transport layer
- Datagram → network layer
- Frame → Link and physical layer

Layer 5: Application Layer

Sappiamo che su Internet sono presenti molti tipi di applicazioni: email, file transfer, streaming video, ecc. Ma come vengono create queste applicazioni? Innanzitutto, bisogna ricordare che le **applicazioni vengono eseguite solo sugli host** (quindi **solo nella “periferia della rete”**), inoltre c’è da dire che esistono due tipi di architetture per le applicazioni:

- Architettura client-server
- Architettura Peer To Peer (P2P)

Client-Server

È un sistema centralizzato nel quale il processo che richiede il servizio e che **inizia la comunicazione** è il **client**, mentre quello che offre il servizio e aspetta è il **server**.

Caratteristiche del **Server**:

- È un host sempre attivo che offre continuamente un servizio poiché il client potrebbe richiedere il servizio (il processo server che sta girando sulla macchina server) in qualsiasi momento;
- Deve essere una macchina ricca di risorse: cioè deve avere un elevata capacità computazionale, deve essere in grado di gestire tante richieste anche contemporaneamente nel minor tempo possibile. Deve avere inoltre anche grandi capacità di memorizzazione;
- Ha un indirizzo IP permanente (non può cambiare, altrimenti il client non saprebbe dove mandare la richiesta, è come se un negozio cambiasse tutti i giorni indirizzo);
- Ha bisogno di *farms of server* per scalare e per la tolleranza ai guasti: se si guastasse un server poco male, ce ne sarebbero altri, anche se le prestazioni saranno leggermente degradate, il servizio saranno comunque disponibile.
Inoltre, per una questione di “tolleranza agli agenti esterni” (catastrofi naturali o antropologiche), i server che formano la farm non dovrebbero nemmeno essere nella stessa stanza/città);
- I servizi offerti sono diversi a seconda dell’applicazione;
- Deve avere un link con un alto bitrate (altrimenti si creerebbe un collo di bottiglia).

Caratteristiche del **client**:

- È un host che chiede un servizio comunicando sempre e solo con i server.
- Potrebbe non essere sempre connesso e può avere un IP dinamico.
- Il client non deve essere ricco di risorse, basta uno smartphone, un pc, ecc.

Peer-to-Peer

In questa architettura non c’è più distinzione netta dei ruoli.

Ogni peer si comporta da server quando offre un servizio e da client quando lo chiede (può esserlo in contemporanea).

I peer non devono essere always-on, cioè sempre disponibili; possono essere connessi a intermittenza e non è obbligato ad avere un IP permanente.

Un peer non è costretto ad essere ricco di risorse.

Questa architettura è fortemente scalabile ma difficile da gestire: le risorse sono messe a comune nella comunità dei peer e non è un modello sbilanciato come il client-server, nel quale le risorse sono tutte concentrare nel server.

Non c’è un limite imposto dalla capacità computazionale del server o dal link al quale è collegato il server (per questo è altamente scalabile).

Un esempio di applicazione che utilizzava architettura P2P era eMule: possiamo ben capire che uno dei problemi legati a queste architetture è la sicurezza, poiché le risorse sono sparse per il mondo è difficile gestirle e dunque rintracciarle ed è piuttosto “semplice” che qualche applicazione nasconde Trojan Horse o contenuti illegali legati alla violazione del Copyright.

Hybrid Architetture

Esistono applicazioni che sfruttano entrambe le architetture, ad esempio sfruttandone una per una funzionalità e una per un’altra. (Le applicazioni che solitamente pensiamo essere P2P in realtà molto probabilmente sono hybrid).

Ad esempio: VoIP (Voice over Internet Protocol, come skype) possiede server centralizzato, usato per trovare l’indirizzo remoto, ma c’è una comunicazione diretta tra i client (p2p); nella messaggistica istantanea invece la chat tra due utenti è in p2p, mentre la registrazione dell’utente è un servizio centralizzato.

Processi comunicanti

Una domanda fondamentale che ci poniamo, a prescindere dall’architettura, è: come comunicano i processi client e server? Se fossero all’interno di uno stesso host potrebbero utilizzare la memoria condivisa o scambiarsi dei messaggi, ma se, come avviene su Internet, l’host non è lo stesso?

Si utilizza sempre lo scambio di messaggi, ma il S.O. mette a disposizione delle **socket-based API**, ovvero delle interfacce basate sui socket.

I **socket** sono un’astrazione (così da fare credere al programmatore che qualcosa esista, per semplificarli la vita), come sono delle “prese di rete”, un po’ come la presa telefonica: **collegandosi al socket si ha**

l’accesso a Internet. Ma non basta collegarsi al socket, gli host necessitano di un **indirizzo IP** e di una **porta** sulla quale “ascoltare”.

Servizi disponibili per una applicazione

Quando si sviluppa un'applicazione bisogna sapere di quali servizi necessità in termini di:

- **Affidabilità:** ovvero l'arrivo a destinazione di tutti i messaggi, integri e nello stesso ordine di invio. In alcune applicazioni la comunicazione deve essere perfetta, quindi con tutte quelle caratteristiche che devono essere esigenti. Immaginiamo di trasferire un file eseguibile: se il file non arriva tutto o se arrivano distorti (magari un 0 al posto di un 1 a causa di errori) o se arrivano invertiti, l'eseguibile non funziona. Ma se uno sta svolgendo una chiamata o una videoconferenza è necessario che l'informazione arrivi tutta ed integra? No, l'importante è che arrivi (ovviamente se quei "vincoli" di affidabilità non sono rispettati al 100%, la qualità del servizio dell'applicazione cala drasticamente, ma funziona comunque, se si perdonano ad esempio poche informazioni).
- **Timing:** una applicazione può essere più o meno tollerante ai ritardi, alcune applicazioni richiedono un basso ritardo (per esempio giochi interattivi), per cui richiedono tempestività (si pensi ad un telefono attraverso Internet, se la voce arriva troppo in ritardo la comunicazione vocale diventa difficile)
- **Throughput:** alcune applicazioni, come quelle multimediali, **richiedono un minimo di throughput** (pensiamo allo streaming). Altrimenti ci potrebbero essere troppi momenti di interruzione. Queste applicazioni sono **bandwidth sensitive (sensibili alla banda)**. Dall'altra parte, come la posta elettronica, che non necessitano di un throughput minimo sono dette elastiche.
- **Sicurezza:** alcune applicazioni più di altre necessitano di sicurezza e riservatezza della comunicazione (quindi necessitano di cifrature, cosicché anche se un intruso sniffasse il messaggio, non disponendo della chiave di cifratura, non potrebbe decriptare il massaggio e non potrebbe leggere le informazioni). Può essere necessaria anche l'integrità dei dati, cioè che un intruso non modifichi il contenuto dei messaggi. Si pensi alle applicazioni che manipolano dati sensibili, come internet banking ecc.

Application	Data Loss	Throughput	Time-Sensitive
File transfer/download	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	No loss	Elastic (few kbps)	No
Internet telephony/ Video conferencing	Loss-tolerant	Audio: few kbps–1 Mbps Video: 10 kbps–5 Mbps	Yes: 100s of msec
Streaming stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps–10 kbps	Yes: 100s of msec
Instant messaging	No loss	Elastic	Yes and no

Figure 2.4 ♦ Requirements of selected network applications

Servizi di Trasporto forniti da Internet

Internet fornisce due servizi di trasporto attraverso due protocolli: TCP e UDP.

Stream service TCP, affidabile

È un servizio **orientato alla connessione**: richiede setup tra client e server (cioè le due parti devono stabilire la connessione per poter comunicare, il percorso della connessione sarà più o meno sempre lo stesso).

Il TCP garantisce:

- un trasporto affidabile;
- controllo del flusso, serve ad evitare che una delle due parti in gioco invii più dati di quanto l'altra sia in grado di gestire, così *si evita di perdere pacchetti a causa dell'overflow del buffer del client* (solitamente problema del client);
- controllo del congestionamento: questo perché un router riceve traffico da sorgenti diverse, e si rischia di perdere pacchetti nel buffer del router congestionato.

Cosa si fa per diminuire la congestione nel router? Si abbassa il rate di trasmissione, se lo fanno tutti, la congestione prima o poi finisce.

Perché si chiama servizio di tipo streaming: perché avendo stabilito la connessione tra due end-point è come se ci fosse il flusso (stream) di un liquido attraverso un tubo tra i due end-device.

Il servizio non offre: non c'è alcuna garanzia sul ritardo sperimentato dai pacchetti (o sulla perdita di messaggi). Nessun throughput minimo. Nessun tipo di sicurezza offerto dal protocollo tcp.

Datagram service UDP, non affidabile.

L'UDP:

- è non affidabile, la sorgente fornisce dei *pacchetti*, i quali solitamente arrivano, possono arrivare in ritardo o non arrivare, possono arrivare non integri e non è garantita la sequenzialità.
- Il servizio è di tipo datagram perché ogni pacchetto viene gestito individualmente (mentre prima erano gestiti in un gruppo/flusso). Approccio best-effort.
- Non c'è connessione;
- Non ci sono controlli nella comunicazione;
- Nessuna garanzia sul ritardo massimo o sul throughput minimo.

Offre qualche altra garanzia? Perché è utile? Non stabilendo la connessione, non introduce nuovi ritardi, quindi l'UDP risulta più veloce del TCP.

E per lo streaming?

UDP è più veloce, ma nessuna garanzia (solitamente usato questo), ma il TCP, introduce sì dei ritardi iniziali, ma garantisce un certo flusso e una certa congestione anche se non garantisce throughput minimo (come le applicazioni streaming vorrebbero).

Protocolli livello applicazione

Come tutti i protocolli definiscono i tipi di messaggi (di richiesta o di risposta) che devono essere scambiati tra le parti, definiscono la sintassi dei messaggi, definisce la semantica del messaggio, e definisce le regole per client e server.

Alcuni di questi protocolli sono di pubblico dominio come HTTP, FTP, SMTP (i documenti dove sono definiti i protocolli di pubblico dominio è RFC), ma esistono anche protocolli, come BitTorrent nel quale il documento descrittivo esiste ed è pubblico, ma non è standard (significa che non è stato mandato all'organizzazione per standardizzarlo, quindi non è un RFC); altri invece, sono protocolli proprietari, (come skype, teams, ecc), anche qui non standard, ma non è nemmeno di dominio pubblico.

Non bisogna confondere il protocollo con l'applicazione: il protocollo è una parte (importante) della applicazione, nella quale vengono definite le regole di come questa funziona l'applicazione.

HTTP

Il protocollo HTTP è un protocollo di tipo **request – response** nel quale il client fa la richiesta e il server manda un oggetto per rispondere a questa richiesta. Ricordiamo due cose:

- Una pagina web, nel quale si utilizza fortemente questo protocollo, “è formata” da diversi oggetti, solitamente da una pagina HTML principale e poi da tanti oggetti referenziati.
- Il server deve comunicare con macchine di diverso tipo, con SO diversi: non è un problema fintanto che ciascuna di esse segue le regole del protocollo.

Che tipo di servizio dal livello sottostante (quello di rete) utilizza il protocollo http? **Utilizza il TCP**, in quanto necessità di affidabilità nello scambio di messaggi.

Così facendo, il client e il server necessitano di **stabilire una connessione TCP** prima di poter iniziare a comunicare, considerando anche il fatto che il protocollo è **stateless** (ovvero senza stati, non ricorda le richieste precedenti: il vantaggio è che risulta molto semplice la sua implementazione), ci viene spontaneo chiederci: quanto tempo ci mette ogni singolo pacchetto?

Nota: è importante chiudere la connessione TCP alla fine della comunicazione per permettere al server di deallocare risorse.

http con connessioni NON persistenti

In questo caso per ogni connessione stabilita può essere trasmesso un solo oggetto. Considerando che le normali pagine web sono formate da ben più di un oggetto, possiamo facilmente capire che ormai questo tipo di protocollo non risulta essere più così vantaggioso.

Ma perché?

Questo perché ogni pacchetto sperimenta un ritardo di 2 **RTT (Round Trip Time, tempo che il pacchetto impiega a fare andata e ritorno tra client e server)** + il ritardo totale legato alla trasmissione dell'oggetto (in questo caso non intendiamo il ritardo: “tempo di trasmissione”, ma il ritardo totale).

Il primo RTT è legato al fatto che va stabilita la connessione TCP, nella quale è presente un three-way-handshake (tratteremo più avanti questo aspetto), il secondo RTT è legato al protocollo http stesso, con la richiesta e la risposta.

Al ricevimento della risposta, viene chiusa la connessione TCP.

http con connessioni persistenti

Come abbiamo visto, ogni singolo pacchetto sperimenta un ritardo di 2RTT + il ritardo totale di trasmissione: in questo tipo di HTTP, per ogni connessione possono essere spediti più oggetti. E questo è un grande vantaggio in termini di risorse allocate sul server e di ritardo, in quanto il primo RTT è sperimentato solo una volta.

In questo caso, la connessione TCP è chiusa dal server dopo un tot di tempo che non riceve richieste dal client.

Messaggio di richiesta HTTP

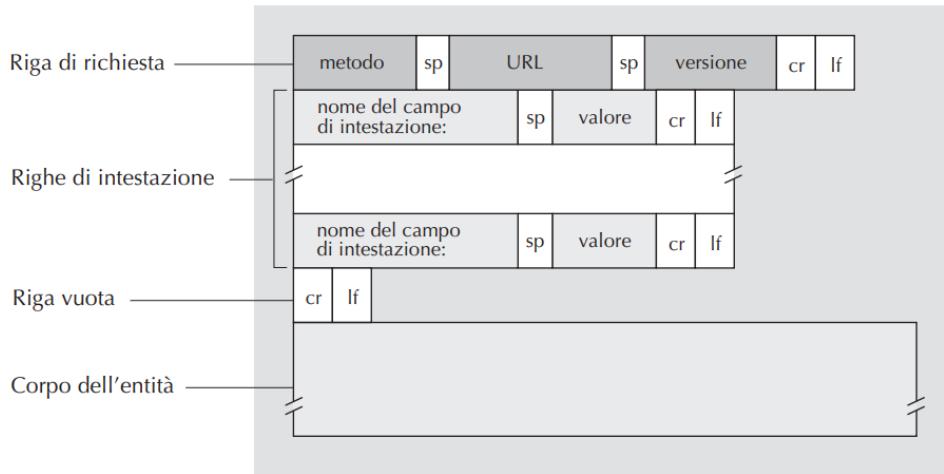
Come possiamo notare dalla figura accanto, il messaggio di richiesta HTTP è codificato in ASCII.

La prima riga è la riga di richiesta, seguita dalle righe di intestazione, per concludere poi con i caratteri carriage return and line feed.

Più in generale un messaggio di richiesta HTPP è fatto come nella figura sottostante.

Esempio di request HTTP

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```



Messaggio di risposta HTTP

Esempio di risposta HTTP

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (Linux)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

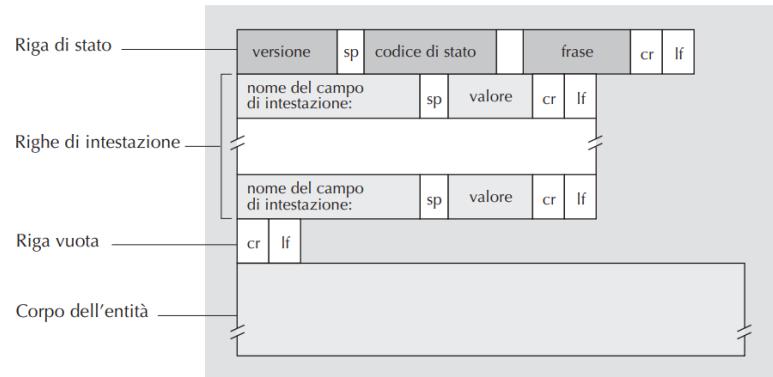
(data data data data data ...)
```

Come possiamo notare, anche la risposta è codificata in ASCII ed è molto simile al messaggio di richiesta. Ci sono sicuramente righe di intestazione diverse, come la data, la data dell'ultima modifica, il tipo di server ecc. Sono presenti anche i dati (se presenti) dopo le righe di intestazione.

Ma notiamo che nella **riga di stato (la prima)** è presente, oltre al tipo di protocollo, un codice che rappresenta appunto, lo stato.

Esempi di codice sono:

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



I cookie

I cookie sono un modo per rendere statefull l'HTTP, il quale è stateless.

Perché vogliamo farlo?

Perché tutte le volte che si deve comunicare col server, ad esempio compilando una form o inviando altre informazioni, con l'approccio stateless è necessario ogni volta inserire i dati e questo approccio risulta essere "noioso/fastidioso" per l'utente.

Possiamo **migliorare questo discorso attraverso i cookie, i quali necessitano di 4 componenti:**

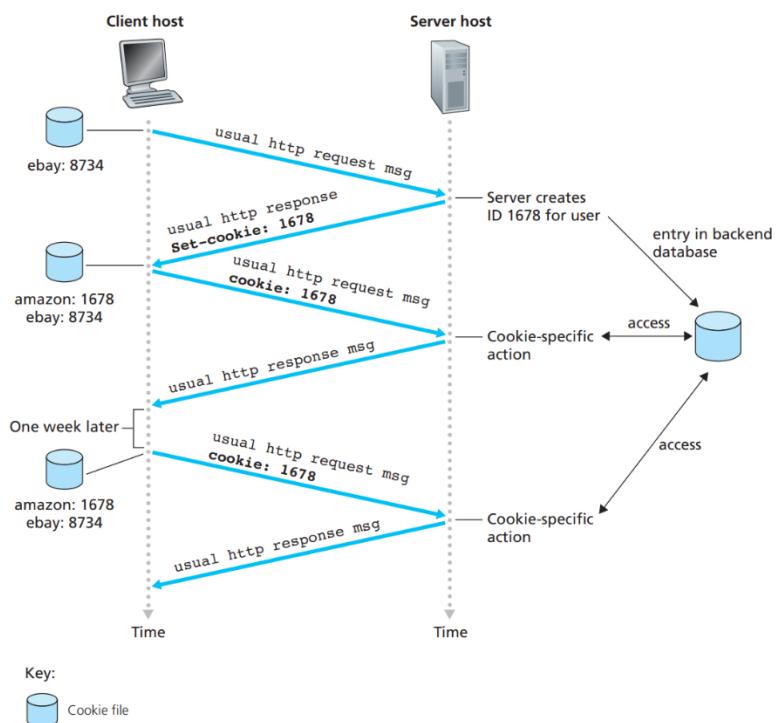
- una linea di intestazione dei cookie nella richiesta
- Una linea di intestazione dei cookie nella risposta
- Un file dei cookie lato cliente
- Un database per la gestione dei cookie lato server

Funzionamento:

- Alla prima richiesta da parte di un determinato client verso il server web, esso crea un id unico per il client che ha effettuato la richiesta. Questo id viene passato attraverso la nuova linea di intestazione nella risposta:

Set-cookie: xxxx

- Il browser del client va a scrivere nel file dei cookie salvato in locale:
SitoWeb, id xxxx
- Alla nuova connessione con il server web, il client si identifica tramite l'id che gli è stato assegnato, scrivendo nella linea di intestazione:
Cookie: xxxx
- D'ora in poi il server risponde alla solita maniera, mandando oggetti (uno per ogni pacchetto), ma può conservarsi nel suo database le informazioni che raccoglie dal client associando queste informazioni all'id del cookie assegnato all'utente. (Ad esempio: nome, cognome, carta di credito, preferenze, ecc.)



Tutte le tracce lasciate dall'utente in Internet, vengono memorizzate nel server, il cui gestore può farne ciò che vuole (ai fini della legge): avere vantaggi come l'auto completamento per le autorizzazioni, le raccomandazioni, i dati della carta di credito salvati portano, come altro lato della medaglia, problemi legati alla privacy e alla sicurezza delle informazioni (e anche sociologici-legali).

Web Caching

Ricevere una risposta ad una richiesta HTTP può richiedere del tempo, che vogliamo diminuire. Si può?

Sì, inserendo una cache tra client e server: il **proxy server** il quale è sia client che server nella realtà.

Dov'è posizionato? **Soltamente all'interno della rete di accesso ad Internet (specialmente se istituzionale)**, talvolta è fornito dall'ISP.

Perché è utile avere una cache all'interno della stessa rete istituzionale? Perché è probabile che gli utenti accedano, in buona parte, a pagine web comuni.

Il proxy-server è trasparente all'utente, il quale non ne conosce l'esistenza.

Ma come funziona?

1. Il browser del client stabilisce una connessione TCP con il proxy server, al quale fa una richiesta HTTP;
2. Se il proxy-server possiede l'oggetto richiesto salvato in memoria, risponde al client, il quale può usufruire dell'oggetto richiesto.
Altrimenti, inoltra la richiesta al web-server al quale è diretta la richiesta http.
3. In questo caso è il proxy server a fare da client; quindi, a stabilire una connessione TCP e fare una richiesta HTTP al web server, il quale risponde al proxy server.
4. Il proxy server memorizza la risposta ed inoltra la risposta al browser, funzionando effettivamente da server.

Il punto 2 in realtà è leggermente più complicato, ma vediamo il perché: se da un lato abbiamo dei vantaggi, dall'altroabbiamo uno svantaggio importante

Vantaggi	Svantaggi
Riduce (statisticamente) il tempo di risposta, specialmente se la probabilità di successo di avere già una pagina in memoria è abbastanza elevata	Aumenta il tempo di risposta solo in caso di mancanza dell'oggetto richiesto in memoria, ma se ben posizionato, capita di rado.
Riduce il traffico che il server web deve gestire. Si riduce il traffico su Internet.	Obsolescenza dei documenti: come in tutte le cache, i dati presenti in memoria potrebbero non essere in linea con quelli presenti nel server web.

Come si risolve il problema dell'obsolescenza?

Il proxy fa delle richieste al web server del tipo:

If-modified-since: date

Il server può rispondere in due modi:

- l'oggetto, perché è stato modificato dopo la data specificata nella richiesta
- **HTTP/1.1 304 Not Modified**

Se questa richiesta fosse fatta sempre, perderemmo una buona parte dei vantaggi introdotti, in pratica la richiesta condizionale non viene fatta sempre.

FTP

Il File Transfer Protocol (FTP) era il protocollo principale, prima del HTTP, ma al giorno d'oggi, esistono ancora applicazioni che usano questo protocollo.

È un'applicazione **client-server**: serve a trasferire file, principalmente da server verso cliente, ma succede anche che i file vengano trasferiti da client a server.

La porta utilizzata è *la porta 21*, la 22 se si vuole usufruire del SFTP (FTP sicuro), in quanto in origine il protocollo non implementava nessuna misura di sicurezza.

Il protocollo FTP, **utilizza il protocollo TCP** come protocollo di rete.

Il client chiede di aprire una connessione, stabilendo una sessione di lavoro e, dopo una prima fase di autenticazione, viene stabilita una **connessione detta di controllo**.

Dopo aver visto quali file sono disponibili l'utente esegue il comando GET (o il comando PUT se vuole caricare file) e una volta **dato il comando, l'effetto è quello di aprire una seconda connessione (aperta dal server)**, sulla porta 20, detta **connessione dati**.



Quando tutti i dati sono stati inviati, il server chiude la connessione dei dati.

Ma la connessione di controllo rimane aperta, cosicché il client possa mandare nuovi comandi.

La connessione di controllo viene chiusa dal client attraverso un apposito comando.

La connessione per i dati è non persistente, la connessione di controllo è persistente.

Perché ci sono due connessioni?

Nel caso in cui si utilizzasse una sola connessione, se si volesse abortire un'operazione (upload o download) questa richiesta del comando verrebbe processata alla fine del trasferimento dei dati; quindi, risulterebbe quasi inutile (in quanto verrebbe processata alla fine dell'esecuzione del comando che si vuole annullare).

Il termine tecnico è che: il controllo è *out of band*.

Un po' come se fossimo in autostrada, il canale di controllo è la corsia di emergenza, sempre libera e disponibile.

FTP è statefull: le informazioni sulle directory rimangono, per dove scarico/carico i file.

Web mail

Un'altra applicazione di rete è la posta elettronica, la quale esiste dagli albori di Internet ed è anch'essa un'applicazione client-server. Utilizza, come facilmente intuibile, il protocollo TCP per il trasferimento dei dati.

Il web mail si basa su tre componenti fondamentali:

- **User agent**, il programma che l'utente utilizza per scrivere e ricevere la posta elettronica
- **Il server di posta elettronica**
- Il protocollo che utilizzano i mail server per scambiarsi i messaggi di posta elettronica (quello del mittente e quello del destinatario)

Il servizio è asincrono, l'utente non si aspetta di ricevere un messaggio e il mittente non pone vincoli temporali sull'arrivo del messaggio.

In questo caso il server non fa solo da server.

SMTP

È un protocollo utilizzato per il web mail: Simple Mail Transfer Protocol (utilizza la porta 25).

Come funziona?

- Il *client manda la mail al proprio mail server*, il quale riceve il messaggio.
- Il mail server del mittente (dopo aver controllato nella coda dei messaggi in uscita se ci sono messaggi da mandare e controllato l'indirizzo) *contatta il mail server del destinatario e gli accoda sulla mailbox* (uno spazio presente sul mail server, privato per ogni utente, nella quale sono presenti i messaggi da leggere) il messaggio di posta elettronica.

Il trasferimento è diretto tra client server e “server” server.

Esempio di invio e ricezione messaggi:

1. Alice scrive il messaggio e l'indirizzo di posta elettronica di Bob.
2. Lo **user agent di Alice manda il messaggio al suo mail server**, nel quale sarà accodato nella coda dei messaggi in uscita.
3. La **parte client del web server di Alice vede il messaggio nella coda dei messaggi in uscita e apre una connessione TCP con il mail server di Bob** (attraverso il three-way handshake).
4. Dopo aver stabilito la connessione, il mail server di Alice manda il messaggio al mail server di Bob
5. Il mail server di Bob riceve il messaggio nella mailbox
6. Bob, quando aprirà la mailbox, troverà il messaggio inviato da Alice.

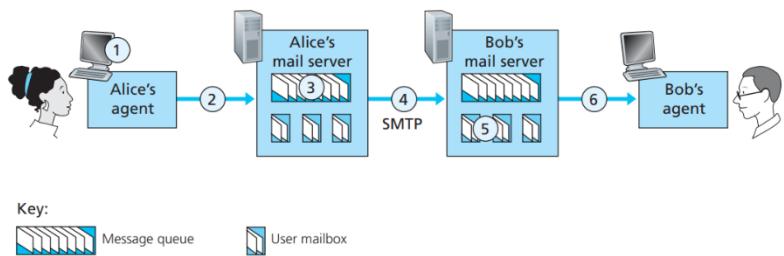


Figure 2.17 ♦ Alice sends a message to Bob

SMTP e HTTP a confronto: sono entrambi persistenti, entrambi strutturati in ASCII, ma mentre nel primo è il client che manda i dati, nel secondo è il server che gli manda.

Inoltre nel protocollo http ogni oggetto è in encapsulato nel proprio messaggio di risposta, nel caso del SMTP invece possiamo avere più oggetti in un unico pacchetto.

Limiti del SMTP: tutto il messaggio deve essere formattata in ASCII, il che come possiamo capire, con l'evoluzione avrebbe potuto portare a dei limiti importanti per il web mail.

MIME (Multipurpose Internet Mail Extensions)

Aggiunge due linee di intestazione

Content-Transfer-Encoding: *in quale codifica è codificato un oggetto*

Content-type: *il tipo di contenuto, video, pdf, immagine ecc.*

POP3 & IMAP (protocolli di accesso)

Abbiamo fatto finora una supposizione semplificatrice: ovvero che il mail server girasse sulla stessa macchina dello user agent (quindi dell'utente).

Agli albori di Internet era vero e il protocollo SMTP è stato pensato per il trasferimento da un host ad un altro, però con l'introduzione dei Personal Computer non è stato più possibile.

Il mail server gira sempre su una macchina che è sempre attivo (che è diversa dalla macchina dove gira lo user agent). Il provider mette a disposizione un mail server, quando l'utente invia il messaggio, questo viene trasferito sul mail server. Comunicano sempre attraverso SMTP. Il mail server del mittente comunica con il mail server del destinatario attraverso l'SMTP.

Infine, il destinatario non può accedere alla mailbox usando il protocollo SMTP, in quanto esso è un protocollo di "push"!

Si usano protocolli di accesso.

POP3 (Post Office Protocol)

È un protocollo per l'accesso alla posta elettronica: utilizza il protocollo TCP e serve per scaricare i messaggi di posta elettronica.

Deve garantire che l'utente possa accedere alla sua mailbox (e solo a quella).

POP3 è composto di 3 fasi:

1. Autenticazione, attraverso l'uso di usernamee password
2. Transazione (recupero dei messaggi), attraverso le operazioni che possono essere
 - a. Scarica e mantieni
 - b. Scarica e cancella
3. Aggiornamento, salvataggio delle modifiche fatte (se email cancellate o meno) e chiusura della connessione TCP

Il POP è stato implementato in CHIARO! Per questo motivo non è più la porta 25, altrimenti sarebbe ancora in chiaro.

Limi del POP3: eccetto la parte di autenticazione, è **stateless!** Questo implica il fatto che non mantenga le informazioni sulla gestione delle cartelle (ad esempio se accedo da più dispositivi).

IMAP (Internet Mail Access Protocol)

La manipolazione di cartelle e messaggi sulla macchina locale pone problemi agli utenti che vogliono accedere da più dispositivi alla stessa mailbox, in quanto preferirebbero mantenere una gerarchia di cartelle su un server remoto cui accedere da differenti host. Ciò non è possibile con **POP3, dato che questo protocollo non fornisce all'utente alcuna procedura per creare cartelle remote e assegnare loro i messaggi.**

Per risolvere questi e altri problemi è stato messo a punto il protocollo IMAP.

From: alice@mail.com

To: bob@mail2.com

Subject: Hello World!

MIME-Version: 1.0

Content-Transfer-Encoding: base64 (come è codificato)

Content-Type: image/jpeg (cos'è e il formato)

Data

DNS

Un host può essere identificato in vari modi, attraverso il proprio *hostname* o attraverso il proprio *indirizzo IP* (ricordiamo che questo è una sequenza di 32 bit).

Le persone ovviamente preferiscono il primo metodo, specialmente quando devono navigare su Internet e vogliono raggiungere siti come www.facebook.com o www.google.com; i router invece, essendo dei computer preferiscono gli indirizzi IP.

Ricordiamo inoltre che l'IP deve essere univoco per ogni host e, di conseguenza, anche il nome.

Ma come facciamo a mappare gli uni negli altri?

Ci pensa il **Domain Name Service**: è un servizio che permette di mappare gli *hostname* in indirizzi IP.

Questo perché è basato su un protocollo (DNS) il quale permette di interrogare un database distribuito sui DNS server.

Il protocollo DNS utilizza UDP (in quanto è molto importante la velocità di risposta, dato che è un servizio usato molto frequentemente, anche da altri protocolli a livello applicazione, e si vuole evitare che questo entri a far parte del ritardo).

Servizi offerti dal DNS (oltre alla risoluzione dei nomi)

- **Host aliasing:** un host dal nome complicato può avere uno o più sinonimi (alias). Per esempio, *relay1.west-coast.enterprise.com* potrebbe avere, diciamo, due sinonimi quali *enterprise.com* e *www.enterprise.com*.

In questo caso, si dice che il nome *relay1.west-coast.enterprise.com* è un **nome canonico**.

I sinonimi, se presenti, sono generalmente più facili da ricordare rispetto ai nomi canonici.

- **Mail server aliasing:** per ovvi motivi è fortemente auspicabile che gli indirizzi di posta elettronica siano facili da ricordare. Per esempio, se Bob ha un account Yahoo, il suo indirizzo di posta elettronica potrebbe essere semplicemente *bob@yahoo.com*.

Tuttavia, l'hostname del server di posta Yahoo è molto più complicato e assai meno facile da ricordare rispetto a *yahoo.com*. Per esempio, il nome canonico potrebbe assomigliare a *relay1.west-coast.yahoo.com*.

Un'applicazione di posta può invocare il DNS per ottenere il nome canonico di un sinonimo fornito, così come l'indirizzo IP dell'host.

- **Distribuzione del carico di rete** (load distribution): il DNS viene anche utilizzato per distribuire il carico tra server replicati, per esempio dei web server.

I siti con molto traffico vengono replicati su più server, ognuno eseguito su un host diverso con un indirizzo IP differente. Nel caso di web server replicati, va dunque associato a ogni nome canonico un insieme di indirizzi IP. Il database DNS contiene questo insieme di indirizzi.

Quando i client effettuano una query DNS per un nome associato a un insieme di indirizzi, il server risponde con l'intero insieme di indirizzi, ma ne varia l'ordinamento a ogni risposta. Dato che generalmente un client invia il suo messaggio di richiesta HTTP al primo indirizzo IP elencato nell'insieme, la rotazione DNS distribuisce il traffico sui server replicati.

Abbiamo detto che il DNS si basa sul DNS server: ma perché abbiamo detto che questo deve essere distribuito? Non potrebbe essere centralizzato?

In quel caso i client dirigerebbero tutte le richieste al singolo server e quest'ultimo risponderebbe loro direttamente. **Sebbene tale semplicità progettuale sia attraente, sarebbe inappropriata per l'attuale Internet, dotata di un vasto e sempre crescente numero di host.**

Tra i problemi legati a uno schema centralizzato:

- **Un solo punto di fallimento.** Se il DNS server si guastasse, Internet non funzionerebbe (se non attraverso l'suo degli indirizzi ip per raggiungere gli altri host!).
- **Volume di traffico.** Un singolo DNS server dovrebbe gestire tutte le richieste e potrebbe diventare un collo di bottiglia.
- **Database centralizzato distante.**
- **Manutenzione.** Il singolo DNS server dovrebbe contenere record relativi a tutti gli host di Internet. Non solo tale database centralizzato sarebbe vasto, ma dovrebbe essere aggiornato frequentemente per tener conto di ogni nuovo host.

Notiamo dunque che i problemi sarebbero tanti e importanti: il DNS utilizza un grande numero di **server, organizzati in maniera gerarchica e distribuiti nel mondo.**

In prima approssimazione, esistono tre classi di DNS server:

- **Root server:** esistono 400 root server, dislocati in tutto il mondo e gestiti da 13 organizzazioni. I root server forniscono gli indirizzi IP dei server TLD.
- **Top-level domain (TLD) server:** questi server si occupano dei domini di primo livello quali *com, org, net, edu e gov*, e di tutti i domini di primo livello relativi ai vari paesi, come uk, fr, ca e it.
- **Server autoritativi:** ogni organizzazione dotata di host pubblicamente accessibili tramite Internet (quali web server e mail server) deve fornire record DNS pubblicamente accessibili che associno i nomi di tali host a indirizzi IP.

Un'organizzazione può scegliere di implementare il proprio server autoritativo o di pagare un fornitore.

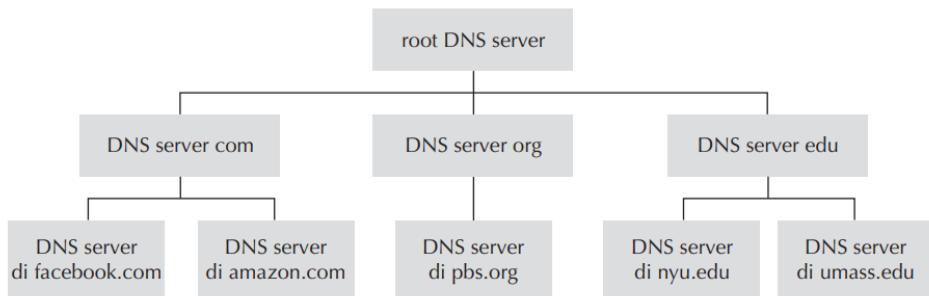


Figura 2.17 Gerarchia parziale di server DNS.

Local DNS Server (Default Name Server).

È una sorta di proxy per il client che deve fare la risoluzione dei nomi e sta solitamente nella rete locale oppure è fornito dal ISP.

Il Default Name Server, non appartiene strettamente alla gerarchia di server, ma che è comunque centrale nell'architettura DNS.

Per non saturare inutilmente la rete, il Local DNS Server si memorizza l'indirizzo IP corrispondente a un hostname.

Dato che gli host e le associazioni tra nome e indirizzo IP non sono in alcun modo permanenti, i DNS server invalidano le informazioni in cache dopo un periodo di tempo fissato (in genere di 2 giorni).

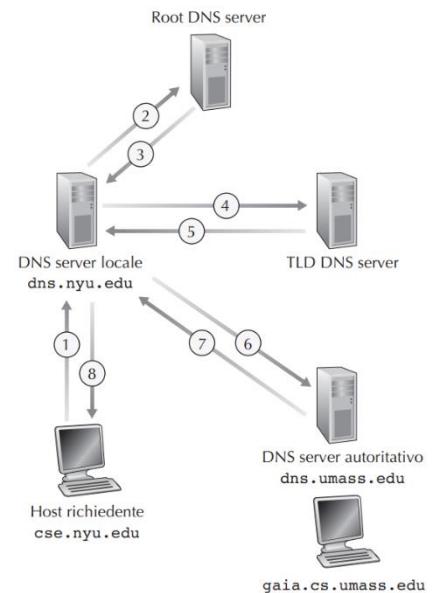
Funzionamento: approccio iterativo

Nell'approccio iterativo, il client contatta il Local DNS server, il quale (nel caso di ignoranza dell'IP) contatta a sua volta i vari DNS Server che lo indirizzeranno vero il DNS server che conosce l'IP, così da poterlo contattare e poter restituire al client l'IP cercato.

Esempio:

Supponiamo che l'host a `xxx.iet.unipi.it` voglia l'IP address per `gaia.umass.edu`:

- Il client contatta il *Local DNS server*: se conosce l'IP, glielo restituisce; altrimenti, *contatta un root DNS server*;
- Il root DNS Server dice al Local DNS di contattare edu DNS server all'indirizzo ip x.x.x.x
- Edu DNS server non è autoritative per umass, e fornisce l'ip del Umass DNS server;
- Il Local DNS, contatta allora umass DNS.
- Umass DNS Server, fornisce l'ip.
- Il Local DNS server può ora rispondere al Client fornendo l'IP.



Notiamo come il Local DNS Server, comportandosi come un proxy, abbia fatto sia da Server che da Client.

In questo caso, per quanto riguarda il DNS caching: *il DNS server*

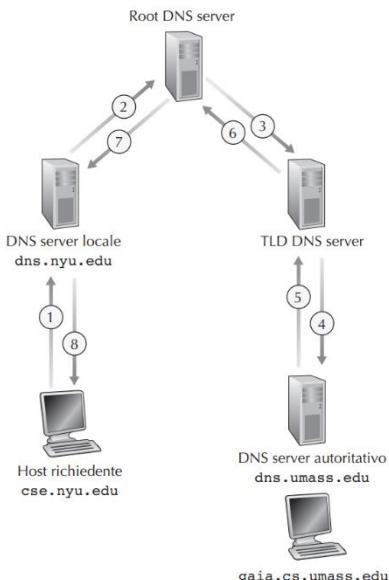
locale può, inoltre, memorizzare in cache gli indirizzi IP dei TLD server, consentendogli di aggirare i root server nella catena di richieste.

Funzionamento: approccio ricorsivo

Nell'approccio ricorsivo invece:

- Il client contatta il Local DNS Server
- Il Local DNS Server contatta root Server
- Il Root Server contatta Top Level DNS server
- Il Top Level DNS server si rivolge a Authoritative DNS server, il quale sa la risposta.
- La risposta poi viaggia a ritroso.

Per l'host richiedente non è cambiato niente, cambia per il Local DNS server, in quanto anche lui ora invia richiesta e aspetta.



L'approccio di default è quello iterativo.

Formato dei record nei DNS Server

Il formato dei Record del Database DNS è:

(*Name, Value, Type, TTL*)

- TTL è il tempo di vita del record: determina quando una risorsa deve essere eliminata dalla cache.
- Tipo e nome dipendono:
 - Se **Type=A**, address, il name è **il nome canonico di un host** e il value è l'ip che gli corrisponde.
 - Se **Type=CNAME**, canonicale name, **nome è l'alias, value è il nome canonico.**
 - Se **Type=NS**, il **nome è un dominio** (.edu, .gov, .com), il value è l'hostname del DNS server authoritative per quel dominio
 - Se **Type=MX**, mail exchange, **il nome è un indirizzo di posta elettronica** (la seconda parte @bob.com), il campo value è il nome canonico del mail server (@yahoo.com)

Cosa significa registrare un nuovo dominio?

Significa **registrare due record nel Database del TLD DNS server**:

(sito.it, dns1.sito.it, NS) → il nome del DNS Server authoritative per quel dominio

(dns.sito.it, 212.212.212.1, A) → l'IP del DNS Server authoritative di quel dominio

Poi, **nel DNS authoritative registrare i record**:

(sito.it, 182.23.41.2, A) → Per associare al server web l'indirizzo IP

(mail.sito.it, yahoo.com, MX) → Per il Mail Exchange, per associare il mail server

DNS e sicurezza

Abbiamo visto che il DNS è un componente critico dell'infrastruttura di Internet, in quanto molti servizi importanti, tra cui il Web e la posta elettronica, non possono farne a meno.

Ci chiediamo qual è il livello di sicurezza e come sia possibile attaccare il DNS.

Il primo attacco che viene in mente è quello DDoS contro i DNS server.

Per esempio, un attaccante può tentare di inondare di pacchetti ciascun root server, in modo che molte delle richieste DNS legittime rimangano senza risposta: generalmente i DNS server sono protetti da sistemi di filtraggio dei pacchetti, configurati per bloccare i messaggi ICMP ed inoltre, la presenza di molti DNS server locali che mantengono in cache gli indirizzi IP dei server di livello più alto, consente spesso al processo di richiesta di non dover interagire con i root server, fungendo anche da meccanismo di protezione verso i root e TLD Server.

Un attacco DDoS potenzialmente più efficace contro il DNS sarebbe quello di inondare di richieste DNS i server di primo livello, per esempio tutti i server che gestiscono il dominio .com.

Con i server DNS di primo livello sarebbe più difficile filtrare le richieste DNS e il loro uso non può essere evitato facilmente come nel caso dei root server.

Tuttavia, la gravità di questo attacco sarebbe mitigata dalle cache dei DNS server locali.

Teoricamente, **il DNS può essere attaccato in altri modi**. In un attacco di tipo **man-in-the middle** l'aggressore si sostituisce al DNS server.

In sintesi, DNS si è dimostrato sorprendentemente robusto contro gli attacchi.

Applicazioni P2P

Come sappiamo nelle applicazioni che utilizzano l'architettura P2P non è necessario che ci sia un server sempre acceso e sempre disponibile (con IP fisso ad esempio), inoltre la comunicazione non è diretta client-server, quindi non c'è questa direzionalità, ma anzi i peer comunicano tra loro.

Dovendo sviluppare un'applicazione per distribuire un file a N host, quale architettura è migliore?

Risposta: *in base a cosa?* Non ci è stato specificato quale criterio vogliamo ottimizzare: il ritardo totale, il risparmio energetico per il gestore, il controllo dei contenuti?

A noi in realtà interessa il tempo di distribuzione (l'upload).

Confrontando le due architetture:

Partiamo dal caso client-server:

Supponiamo che i server sia connesso con una banda molto elevata di u_s b/s.

Vogliamo calcolare il tempo di distribuzione.

Il server per inviare (lo fa in maniera sequenziale) ci mette NF/u_s (N numero di cliente, F dimensione del File, u_s banda di upload). Almeno! Quindi è possibile che ci metta di più.

Il client ci mette F/d_i (d_i banda in download) per scaricare.

Per fare una stima: si prende il massimo tra questi due valori, dove per F/d_i si prende quello con valore di download d_i minimo.

Il primo termine cresce linearmente, il secondo (in teoria potrebbe esserci un host che possiede d_i molto piccolo, ma potrebbe essere raro) viene trascurato.

Ergo il tempo di attesa cresce linearmente.

Ora analizziamo il caso p2p:

Ogni peer ha capacità di upload u_i e di download d_i .

Il server deve dare la copia del file ad un peer, quindi tempo: F/u_s (meno di quello non ci si può mettere).

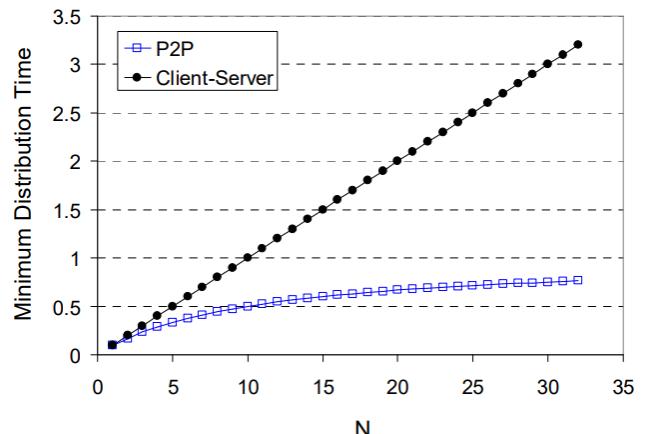
Il client ci mette sempre F/d_i per il download.

Il tempo più veloce di upload è $u_s + \text{sommatoria degli } u_i$.

Ora il tempo di ritardo è = max {tempo upload server, tempo download client, $N^*F/\text{tempo più veloce upload}$ }.

Per N che cresce e diventa molto grande?

Il primo termine non dipende da N , il secondo è il discorso di prima tra parentesi, trascurabile il suo variare. Il terzo termine dipende da N : sotto l'ipotesi (falsa!!) che u_i siano tutti uguali \rightarrow denominatore diventa u_s+u , e N viene semplificato.



Ricerca dei contenuti

Quando si vuole scaricare un file attraverso un'applicazione P2P, è necessario sapere dove si trova questo file. Perciò è necessario avere un indice, un database, nel quale ci sarà memorizzato il contenuto da cercare e l'indirizzo dell'host in cui questo contenuto è disponibile (ovvero a chi ci si deve rivolgere).

Come si gestisce questo database?

- **Soluzione centralizzata.**

L'indice sarebbe client-server, ma l'applicazione rimarrebbe p2p, solo la fase preliminare della ricerca dei contenuti utilizzerebbe l'architettura client-server.

È presente una **fase di registrazione peer che vuole condividere** manda una notifica al server, indicando file e proprio indirizzo IP.

Il peer richiedente manda al server una richiesta del tipo: mi serve questo file, a chi mi devo rivolgere? E **il server restituisce l'IP del peer condividente il contenuto cercato.**

Ora i peer stabiliscono una connessione TCP e il peer può scaricare il file.

Difetti della soluzione centralizzata:

- Single Point of failure;
- Il server diventa un bottle-neck.

Entrambi i problemi si possono risolvere duplicando i server.

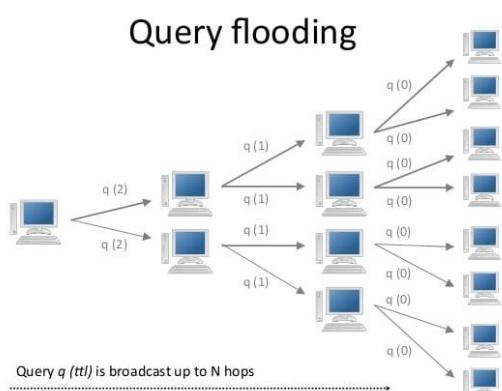
- Il gestore del Server è perseguitabile penalmente e civilmente nel caso in cui nell'applicazione si svolgessero attività illecite, come violazione del copyright.

- **Query Flooding**

Sarebbe una soluzione completamente distribuita.

Non ci sono i due problemi legati alla centralità del Server (ma come abbiamo visto, i problemi sarebbero stati risolti) e, a livello legale, non si può fare causa a tutti per la violazione del copyright.

L'indice è completamente distribuito tra i peer che mettono a disposizione dei contenuti, realizzando la parte di indici relativa ai suoi contenuti.



Ovviamente non è più necessaria la fase di registrazione: so dov'è l'elemento.

Un peer però **non è** collegato con tutti gli altri peer della comunità, ma solo con i vicini (ovvero i peer con i quali è collegato attraverso una connessione TCP e non è stabilita con tutti).

Se un peer è interessato a un certo contenuto x , il peer prepara una query e manda questa query a tutti i suoi vicini (ma non tutti i peer). Se i vicini posseggono l'elemento glielo restituiscono, altrimenti, inoltrano questa query a tutti i loro vicini (ovvero si fa **una query a cascata**).

Ottenuto l'IP del peer che possiede il contenuto si

stabilisce una sorta di connessione client-server tra peer.

Svantaggi:

- Si genera tanto traffico di controllo (serve solo a cercare l'ip di chi possiede il contenuto) a seguito delle query a cascata.
- Il tempo di risposta non è più predicibile, dipende dal numero di passi che si fanno

Per ridurre il traffico di controllo:

- **si pone** una sorta di TTL, ovvero **un limite massimo al numero di hope** (quante volte posso ripetere la query, e ogni volta che raggiungo un peer aumento che questo valore): se da una parte riduco il traffico di controllo, d'altra parte potrei non trovare sempre il contenuto.

• Hierarchical Overlay:

Una via di mezzo tra il totalmente centralizzato e il totalmente distribuito.

L'idea su cui si basa è che i peer su cui si basa la comunità non sono tutti uguali: esistono i **super-peer**. Questi non sono stati selezionati a priori, ma semplicemente hanno caratteristiche particolari.

Ad esempio, sono collegati con dei link con un'elevata banda e sono connessi sempre (o quasi), si pensi ai dispositivi che fanno parte di un'organizzazione, come un'università.

E poi sono presenti gli altri peer comuni.

L'indice viene organizzato in maniera distribuita SOLO sui super-peer. Quindi distribuito, ma solo su un numero limitato di peer.

Fase di registrazione: se consideriamo un peer ordinario che vuole mettere a disposizione dei contenuti, lo deve comunicare al "suo super-peer", il quale salva queste informazioni sul suo indice (ovvero: contenuti a,b,c,d disponibile presso x).

Fase successiva, quella di ricerca: supponiamo che un peer sia interessato a un certo contenuto: chiede al super-peer:

- se il super-peer lo sa, inoltra al peer-comune l'indirizzo IP del peer presso il quale è disponibile la connessione.
- Se il super-peer non ha nel proprio indice il contenuto che gli è stato richiesto: manda una **query a cascata agli altri super-peer**.

Una volta che il peer ottiene l'IP, apre una connessione TCP con il peer che mette a disposizione il contenuto cercato.

• Distributed Hash Table

Si usa una tabella hash distribuita: **ad ogni peer viene assegnato** un id compreso tra $[0, 2^N - 1]$ ovvero **una stringa di N bit** e **anche la chiave per la ricerca del contenuto deve essere rappresentato su N bit**.

Per ottenere questa stringa di N bit, ad esempio al titolo di un file, si applica una funzione hash.

La **coppia key-value** (contenuto-peer) dove va memorizzato? In un peer "qualsiasi".

O meglio c'è un criterio di scelta, nel caso del **Circle Distributed Hash Table** è quello del nodo immediato successore, inteso come **il peer che ha l'id più vicino al valore della chiave**.

Es: key = 4, la coppia si assegna al peer con id=5. (ovviamente non è detto che i peer ci siano tutti e che quindi il 5 sia presente, in quel caso si assegna al 6 o 7 ecc).

NOTA: in questo approccio un peer è collegato solo con 2 peer: l'immediato predecessore e l'immediato successore. Ergo per fare una ricerca si può chiedere solo ai suoi vicini; quindi, si chiede solitamente al suo immediato successore.

Ovviamente questo metodo è dispendioso dal punto di vista del tempo ma posso ovviare, spendendo di più, aggiungendo delle shortcut, quindi aggiungendo collegamenti. Quindi passando da $O(N)$ a $O(\log(n))$.

Distribuzione dei file

La distribuzione di un file può avvenire tramite il download da un singolo peer (Napster) o da più peer (nel caso di BitTorrent).

Nel primo caso, che è quello che si è preso sempre in considerazione finora, una volta stabilita la connessione TCP abbiamo una sorta di peer client e peer server.

È possibile scaricare un file da più peer in contemporanea?

Supponiamo ci sia un file che si vuole condividere con un certo numero di utenti attraverso l'uso del **protocollo BitTorrent**, il quale è un protocollo per la condivisione di file in architettura P2P. È così chiamato perché l'insieme dei peer che sono coinvolti nella distribuzione di file è chiamato Torrent.

C'è un Torrent per ogni file che si vuole condividere.

C'è anche dunque un'entità che supervisiona e controlla il processo di condivisione, il tracker, il quale contiene la lista di peer che fanno parte del torrent in un certo istante, il quale è dinamico: si consideri il fatto che un peer che scarica un file si aggiunge al torrent, ma quando termina il download, il peer si "toglie" dal torrent.

I peer non sono tutti uguali tra di loro:

- Esistono dunque dei **peer che non hanno scaricato niente del file**;
- I **leecher**, i quali hanno scaricato una parte del file
- I **seeder**, i quali sono peer che hanno scaricato una copia completa del file e rimangono nel torrent per condividere il file scaricato.

Operazioni per poter scaricare un file:

1. Il peer deve contattare il **torrent server**, il quale **risponderà** con un file .torrent contenente, tra le varie cose, anche *l'ip del tracker*.
2. Ottenuto l'IP e contatto *il tracker*, questo **registra il peer come parte del torrent**.
3. *Il tracker fornisce la lista dei peer che in questo momento fanno parte del torrent*.
4. Il peer inizia a **contattare gli altri peer nella lista**, allo scopo di stabilire delle connessioni con i peer che sono presenti nella lista.

(Perché sono connessioni TCP? Deve avvenire un trasferimento dei file, quindi si vuole un trasferimento affidabile).

In generale non riesce a stabilire connessione con tutti i peer della lista, ma solo con un sottoinsieme (ad esempio sono tanti i peer, o alcuni peer potrebbero non rispondere perché magari nel frattempo qualche peer ha abbandonato il torrent, ma è ancora nella lista perché non è aggiornatissima).

I peer con cui si stabilisce la connessione TCP è detto vicino.

5. Poiché il suo scopo è ottenere il file, il peer comincia a chiedere ai vicini il file, ma non tutto intero: il file viene spedito in **chunk** ovvero parti di file.

I chunk possono essere spediti in maniera non sequenziale: quindi l'importante non è l'ordine, l'importante è che arrivi tutto il file.

Il peer periodicamente chiede ai vicini di mandargli i chunk che loro hanno già.

Quale chunk si scarica per primo? Il chunk più raro.

Perché il più raro? Perché se è poco presente, se chi possiede quel chunk abbandonasse il torrent, aumenterebbero i tempi di download, perché quel chunk non sarebbe più disponibile e non si potrebbe più completare il download del file.

Se un peer riceve tante richieste, non può rispondere a tutti: dovendo scegliere, chi scegli?

Il criterio di scelta è il Tit-for-tat (Dente per dente): cerco di privilegiare, chi mi tratta meglio.

Ovvero cerco di servire prima quello da cui ricevo più velocemente.

Il peer ogni 10 secondi fa la graduatoria dei peer vicini che gli mandano i dati in base a quanti chunk o byte per secondo gli sono arrivati dai peer.

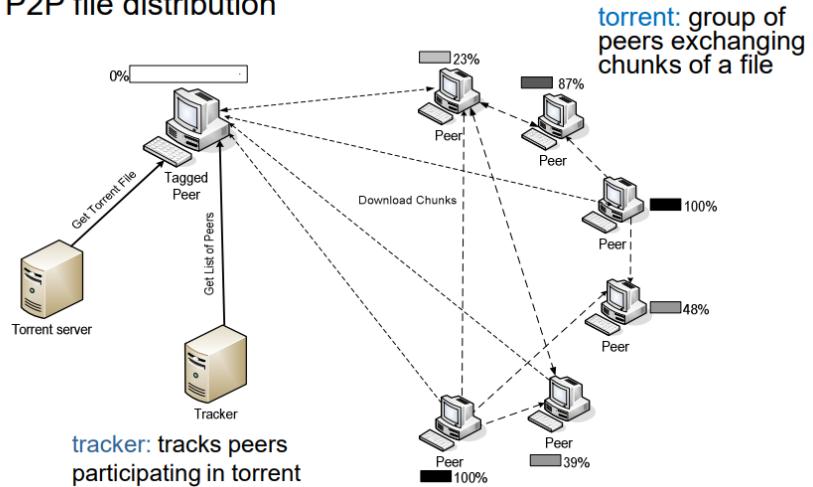
Dopodiché, sceglie i primi 4, e manda il chunk solo a quelli (risponde solo a quelli).

I peer veloci, comunicano con i peer veloci e i peer lenti si devono accontentare di comunicare con i peer lenti.

C'è una categoria di peer chiamata **free-rider**: ovvero, *chi riceve chunk ma non ne manda mai, ovvero non collabora*. Il meccanismo di BitTorrent scoraggia i free-rider.

Se però si facesse Tit-for-tat secco, si rischierebbe di fossilizzarsi, in quanto potrebbero esserci peer più veloci dei migliori 4: si prova a mandare dei chunk in maniera casuale, questo viene fatto ogni 30 secondi. Se questo peer risponde ed è più veloce di qualcuno della top 4 bene, altrimenti si è fatto un buco nell'acqua e non entra in classifica.

□ P2P file distribution



In questa maniera si dimezzano i tempi, ma soprattutto non si crea una connessione tipo client-server.

Il peer mentre scarica si comporta sia da client e da server.

È presente il problema della sicurezza: qualche peer potrebbe immettere una copia fasulla o corrotta o dei virus.

Layer 1: Physical

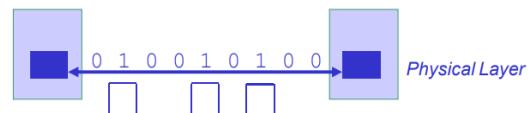
Supponiamo di avere due computer direttamente collegati tra loro: affinché possano comunicare è necessario avere una codifica dell'informazione. L'informazione è rappresentata attraverso i bit.

I bit sono codificati attraverso segnali elettrici, elettromagnetici o luminosi e sono spediti attraverso il link fisico, ai capi del quale sono collegati un trasmettitore e un ricevitore.

Nella realtà potrebbe verificarsi fenomeni di attenuazione e dunque, il messaggio potrebbe essere recepito in maniera diversa rispetto a come è stato inviato.

Ovviamente l'attenuazione dipende:

- Dalla potenza con cui il messaggio è stato inviato
 - Dalla distanza
 - Dal mezzo fisico



Oltre l'attenuazione, tra i fattori da tenere in considerazione ci sono i disturbi:

- Disturbo elettromagnetico
 - Disturbo termico



010010100 011010110 Direct Connection Networks

Il ricevitore conosce la frequenza con la quale viene trasmesso il messaggio ed ogni tot unità di tempo deve campionare il segnale: una volta fatto questo, deve confrontarlo con il valore di soglia.

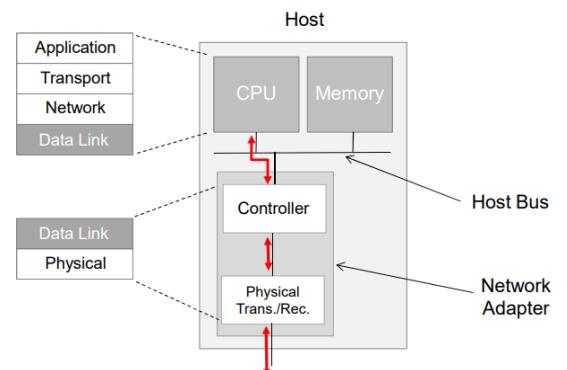
Il mezzo di comunicazione può essere **half-duplex** (link monodirezionale) o **full-duplex** (link bidirezionale).

Layer 2: Data-Link

Idealemente ci piacerebbe avere un canale affidabile, dunque dobbiamo aggiungere alcune funzionalità per poterne ottenere uno utilizzando il canale che per suo natura è inaffidabile.

Dove vengono realizzate le funzionalità messe a disposizione dal livello Data-Link?

Una parte in software e una parte in Hardware, all'interno della **scheda di rete**. Le funzionalità per il controllo dell'errore sono implementate in hardware dal controllore.



Servizi messi a disposizione

Framing: anziché gestire una lunga sequenza, si suddivide l'informazione in frame. I frame sono costituiti da un **campo dati**, nel quale è inserito il datagramma, e da **vari campi di intestazione**.

La struttura del frame è specificata dal protocollo.

Nell'header sono inserite ad esempio varie informazioni di controllo, come il numero del pacchetto in modo tale da poterlo riordinare.

Nel Trailer invece sono presenti i bit di ridondanza per l'error Detection.

Error Detection: facendo delle operazioni sul frame ricevuto, *il ricevitore* deve essere in grado di stabilire se il frame che ha ricevuto è corretto o meno.

Error correction: *in alcuni casi*, gli algoritmi più sofisticati di Error Detection, **sono in grado di identificare i bit errati**. Segue che se il ricevitore sa quali sono i bit errati, essendo bit (0-1), il ricevitore può correggere

l'errore.

Ovviamente l'error correction ha un costo.

Affidabilità della trasmissione: nel caso in cui non si voglia usare l'error correction a causa dei costi, o perché non è possibile identificare gli errori, è necessario avere un riscontro e una ritrasmissione da parte del trasmittitore.

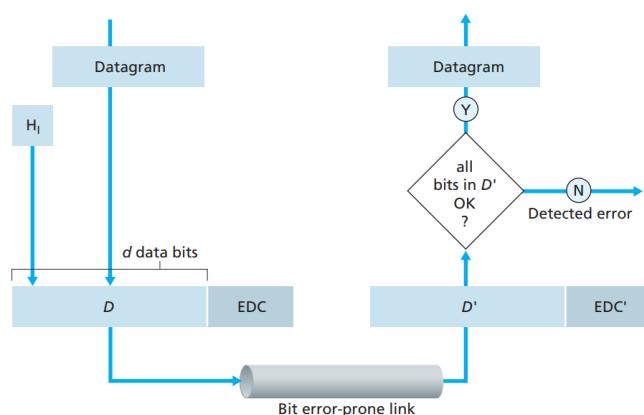
Controllo del flusso: serve ad **evitare che il trasmittitore invii dati a una velocità superiore** a quello con cui il ricevitore riesce a riceverli.

Error Detection

Il ricevitore deve analizzare il frame ricevuto e deve stabilire se il frame è corretto o meno: come fa?

Trasmittitore invia **oltre ai dati, R bit di ridondanza**, i quali sono i risultati dell'algoritmo di error detection.

Il ricevitore, all'arrivo del frame, utilizza lo stesso algoritmo usato dal trasmittitore per calcolare R: se ottiene lo stesso risultato del trasmittitore, il frame è arrivato integro (in linea teorica).



Questo perché, come possiamo notare dalla figura, **il mezzo di comunicazione è inaffidabile per sua natura**, dunque potrebbe corrompersi, oltre ai dati, anche i bit di ridondanza: se per una serie di sfortunati eventi R' (i bit di ridondanza arrivati) è uguale all' R calcolato dal ricevitore, esso pensa che il risultato sia corretto quando invece il pacchetto potrebbe essere corrotto! Inoltre, **gli algoritmi di error detection non sono affidabili al 100%**.

Parity check

Il trasmittitore aggiunge **un singolo bit** di ridondanza: si mette **in modo tale da far tornare pari/dispari il numero di 1 nella stringa** (la parità può essere sia pari che dispari, ovvero: voglio il numero di 1 pari oppure dispari).

Il costo è nullo: il circuito che conta è semplice, inviare un bit in più ha un costo trascurabile.

Problemi:

- **si accorge solo di un numero dispari di errori;**
- **R si può corrompere facilmente.**

Usato solo all'interno dei calcolatori.

Checksum

Dato un pacchetto di byte, lo si divide in pacchetti da 16 bit (esempio) e se ne fa la somma bit a bit di ogni pacchetto.

Se i due valori coincidono (la checksum ricevuta e quella calcolata) si assume che i dati ricevuti siano corretti, ma **problema:** *Non abbiamo la sicurezza al 100%.*

Nota: Il checksum si può fare offline, e si può inserire nell'header e non nel trailer.

A livello Data Link non si usa la checksum, **viene usata a livello del tcp perché si presuppone che i controlli più specifici siano già stati fatti a livello data link.**

Cyclic Redundancy Check (CRC)

Dato un blocco di D bit da mandare, si vuole calcolare il blocco di bit di ridondanza.

I dati si vedono come un numero binario: si deve scegliere **un generatore G** (noto a trasmettitore e ricevitore) tale che i bit di ridondanza siano calcolati così: $D * 2^r \text{ XOR } R = nG$

Il ricevitore, dunque, esegue l'operazione con lo XOR e divide per G: se il resto che ottiene è diverso da 0, è presente un errore o nei dati o nei bit R.

Questo algoritmo si usa a livello Data Link come algoritmo di error detection.

Error correction

Gli algoritmi di error correction (in realtà sono di error detection e correction) presentano un costo più elevato in quanto a elaborazione e sono in grado di correggere solo un numero limitato di errori.

Conviene usarli? Dipende, in generale, da soli no in quanto non affidabili al 100%: conviene usarli insieme ad un protocollo di trasferimento affidabile.

Bit di parità bidimensionale

Si ha un bit di parità sulle righe e uno sulle colonne: si è così in grado di determinare dove si è verificato l'errore.

In figura possiamo notare un esempio.

No errors	Correctable single-bit error
1 0 1 0 1 1	1 0 1 0 1 1
1 1 1 1 0 0	1 0 1 1 0 0
0 1 1 1 0 1	0 1 1 1 0 1
0 0 1 0 1 0	0 0 1 0 1 0

Parity error
Parity error

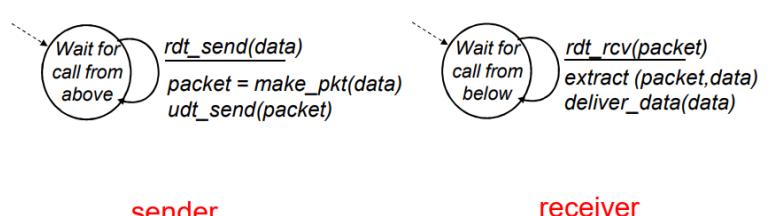
Reliable Data Transfer (Reti a Connessione diretta)

Il livello data link di fatto fornisce un link virtuale affidabile.

Trasferimento dati affidabile: è un problema presenti su più livelli dello stack, sia a livello data link sia a livello trasporto.

RdT 1.0

In questa versione del protocollo non c'è bisogno di fare i controlli quando si invia/si riceve il pacchetto: siamo nel caso del modello con livello fisico ideale.



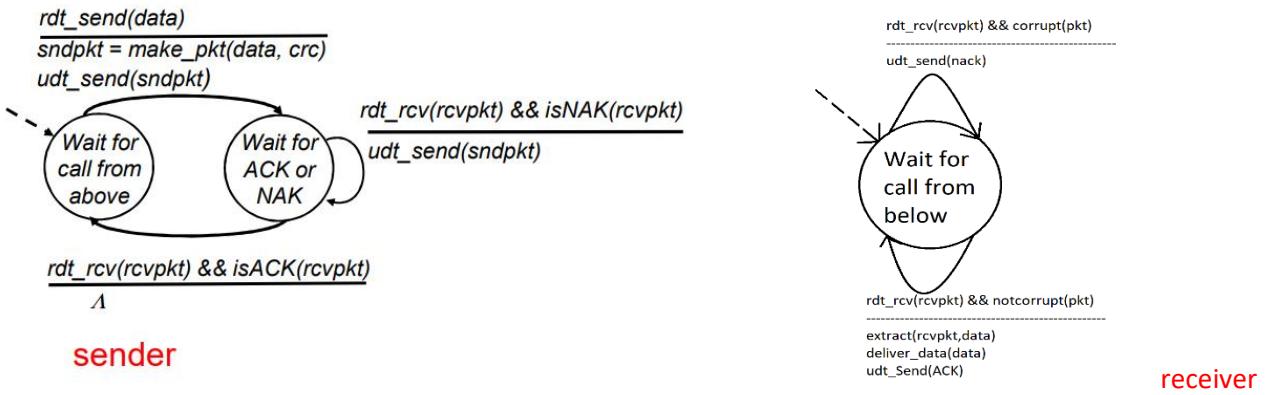
RdT 2.0

Si toglie l'ipotesi del mezzo di trasporto ideale: il mezzo introduce errori, ergo bisogna fare error correction da parte del ricevitore: *CRC* (Data Link), *CheckSum* (Transport Layer).

Questa versione è basata sulla metodologia **stop and wait**: il trasmettitore manda il pacchetto e aspetta finché non riceve risposta.

Che risposta deve attendere? **Il ricevitore dopo aver ricevuto il messaggio manda:**

ACK (Acknowledgement) o NACK (Negative Ack).



Per questo protocollo si è fatta un'assunzione: che il mezzo sia inaffidabile solo per i dati, mentre il mezzo è lo stesso e si possono corrompere anche ACK e NACK.

RdT 2.1

Problema da risolvere: Se l'ACK o il NACK si corrompono?

Soluzioni:

a) introdurre error correction solo su ack e nack che corregga eventuali errori. I bit sono meno rispetto ai dati. Difetto: bisogna inserire bit di ridondanza: non è detto che funzioni sempre e ha un costo.

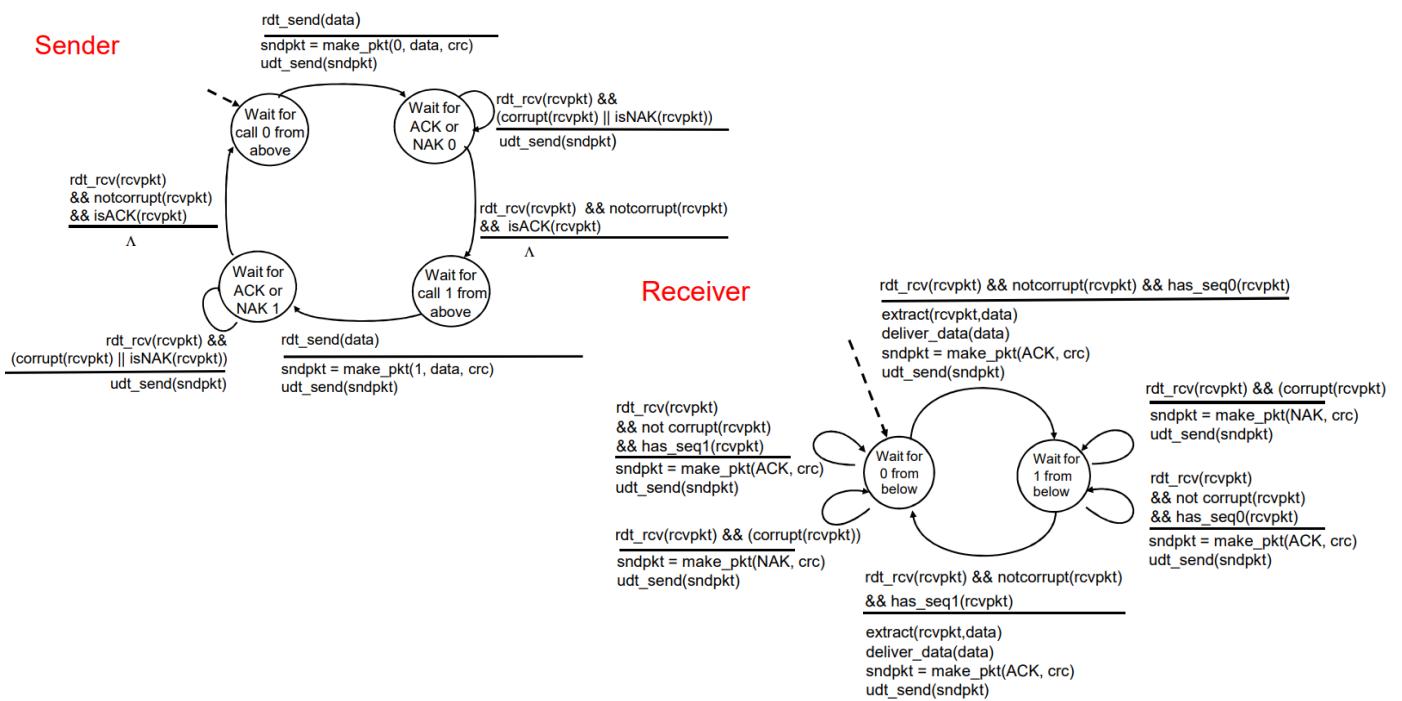
b) Fare in modo che ack e nack possano corrompersi e fare solo error detection, non correction.

Nel caso in cui si riceva un **ack o nack corrotto: si chieda che si ritrasmetta tutto il pacchetto per sicurezza, come se fosse stato ricevuto un nack**. Eventuali duplicati devono essere eliminati, perché non devono essere inseriti tutti.

Il ricevitore riconosce un duplikato inserendo un campo che indica il numero di sequenza del pacchetto.

Dopo aver controllato che sia corretto il pacchetto ricevuto, si controlla se il numero di sequenza è uguale o no, se è uguale si scarta il pacchetto.

Il numero di sequenza in realtà, essendo un approccio stop & wait è un solo bit, in quanto viaggia un solo pacchetto per volta.



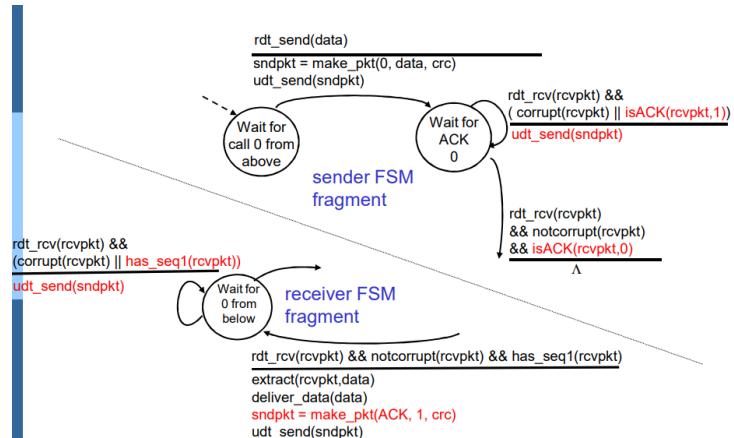
Posso non usare ack e nack?

Posso usare solo ack (Nack free): se il ricevente ha ricevuto correttamente il pacchetto manda un ack, se ha ricevuto il pacchetto corrente con errori, manda un ack relativo al pacchetto precedente: ovvero:

In rosso sono evidenziate le differenze rispetto a prima: notiamo come nel pacchetto con ack, mandiamo anche il numero di sequenze del pacchetto al quale ci riferiamo.

E se i pacchetti non arrivano?

Nel collegamento tra due computer non è molto frequente, ma a livello trasporto è frequente, ci sono i router, si possono creare code ecc ecc.



Come determinare una perdita di pacchetti?

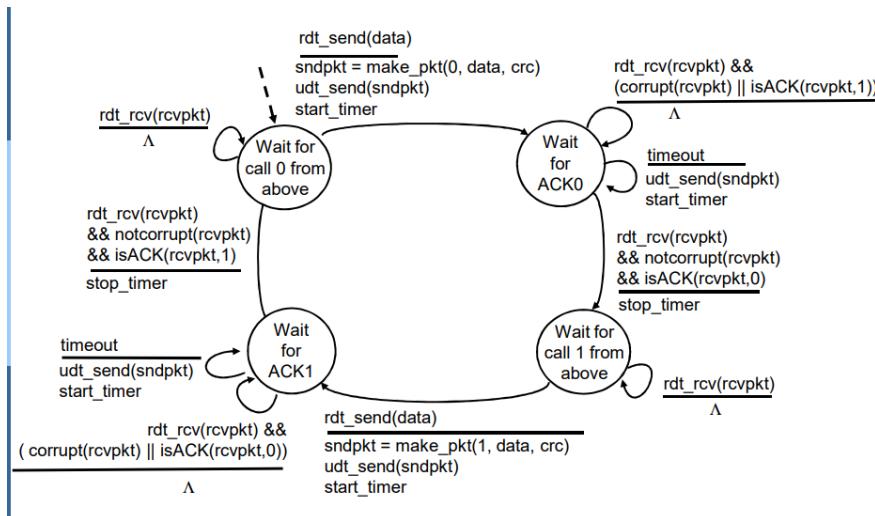
Il trasmettitore deve aspettare una quantità di tempo ragionevole prima di far scadere il time-out.

Cosa si fa nel caso in cui scada il time out? Si rimanda il pacchetto (si assume che non arriverà mai l'ack o il pacchetto).

Quando deve essere il time-out: almeno un RTT, ma possiamo calcolarlo? Beh, collegamento punto a punto, non abbiamo ritardi di accodamento, tempi di elaborazioni trascurabili, il tempo di propagazione (distanza/rate) e di trasmissione la sappiamo (Grandezza pacchetto/rate).

Come si modifica il protocollo quando assumiamo le perdite?

RdT 3.0



Sender:

è sbagliata azione evento ore 2 e suo diametralmente opposto: è *udt_send(sndpkt)* e non Λ .

Receiver rimane uguale a 2.1

Performance: a causa dell'approccio stop & wait, il fattore di utilizzo del mezzo è pessimo. È ricavato calcolando prima il ritardo di trasmissione dt (L/R) e poi facendo:

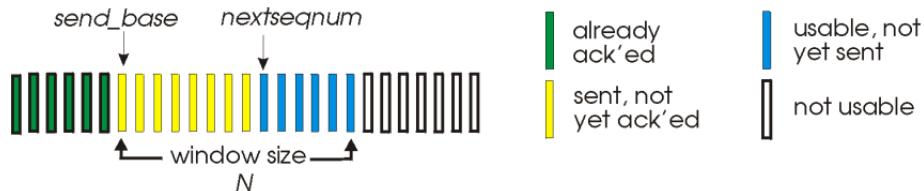
$$U = dt / (RTT + dt)$$

Pipeline

Il protocollo (non reale) visto prima è inefficiente, in quanto utilizza un approccio stop & wait, dunque dovendo mandare un messaggio alla volta non sfrutta al meglio le capacità del link di connessione.

Se si utilizzasse un approccio nel quale possono essere inviati più pacchetti prima di ricevere gli ACK, allora il fattore di utilizzo migliorerrebbe.

Ma il problema ora diventa: il ricevitore deve bufferizzare i pacchetti che arrivano fuori ordine e il trasmettitore deve bufferizzare quelli di cui non ha ricevuto ack.

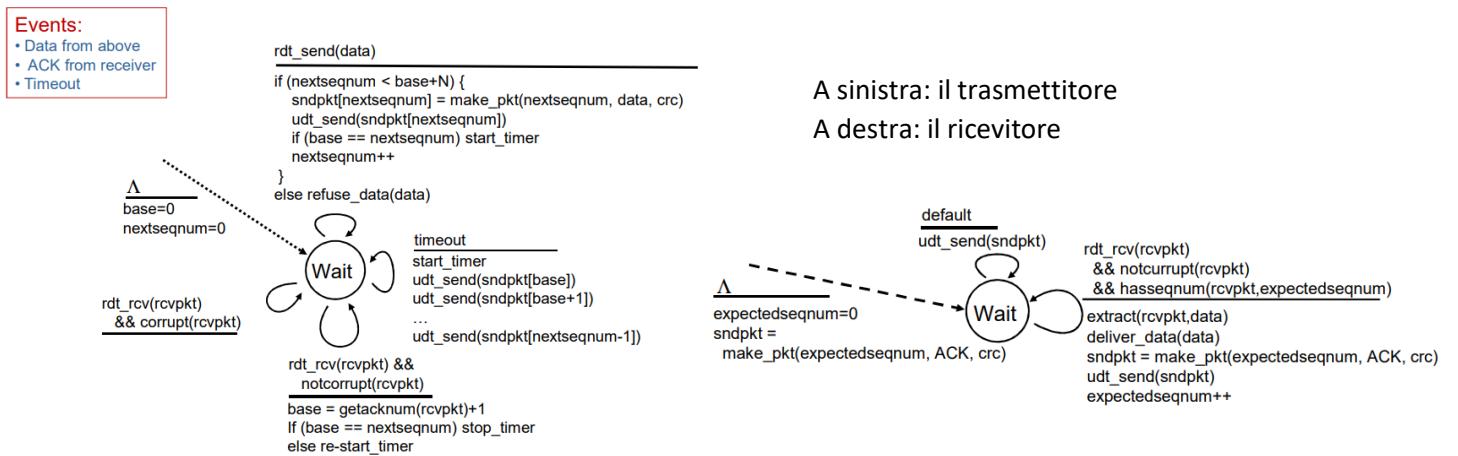


Ci sono due strategie:

Go-back-N

Può mandare fino a N pacchetti unACKed.

Poiché il ricevitore può mandare solo ACK cumulativi non bufferizza niente (nessun pacchetto fuori ordine), dunque il trasmettitore mantiene un timer per il pacchetto unACKed più antico: se scade il timeout, rispedisce tutti i pacchetti dal "send_base" in poi.

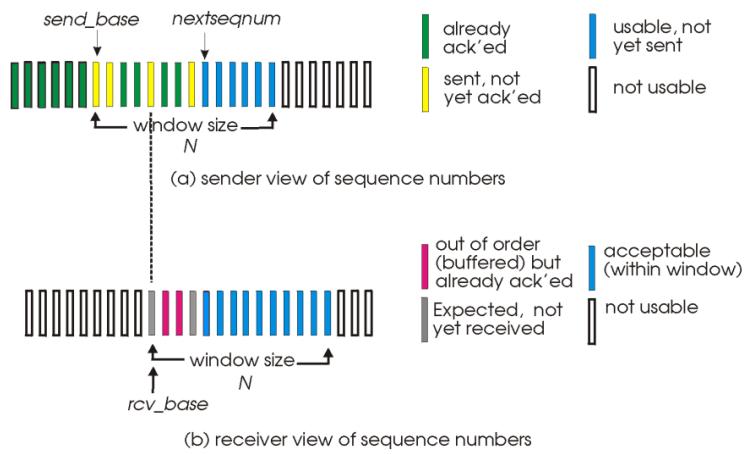


Limiti del go back n: i pacchetti sono riscontrati in maniera cumulativa, lato destinatario non vengono bufferizzati: *quando avviene un timeout si dà luogo a ritrasmissioni non necessarie*.

Selective Repeat

Può mandare fino a N pacchetti unACKed.

Poiché il ricevitore manda un ACK per ogni pacchetto e i pacchetti vengono bufferizzati, il trasmettitore: rimanda pacchetti solo per gli ack non ricevuti e tiene timer per ogni unACKed packet.



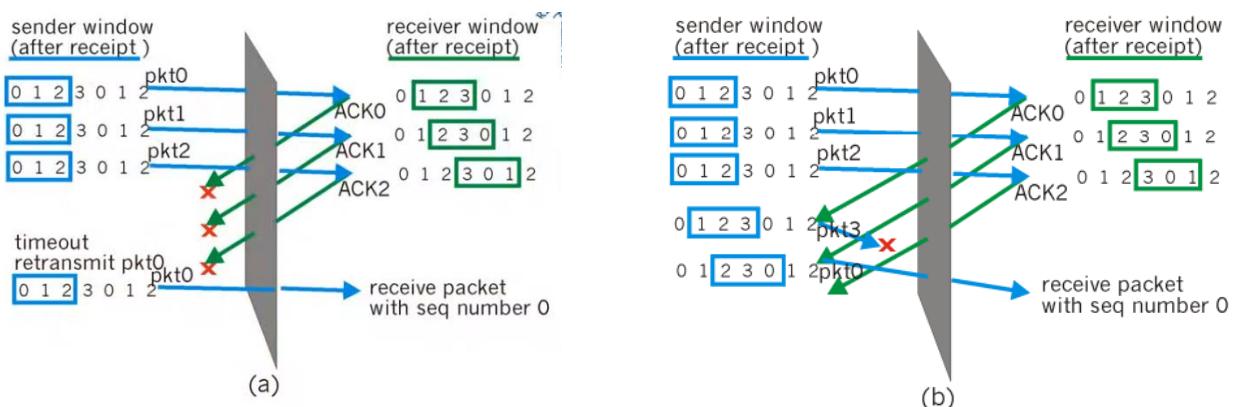
Il trasmettitore deve:

- Se il prossimo numero di sequenza disponibile è all'interno della finestra, spedisce il pacchetto
- Tiene un **timer per ogni pacchetto**, se questo scade prima di avere ricevuto un ack, rispedisce il pacchetto e fa ripartire il timer
- Se **riceve un ACK**, se è di un messaggio unACKed lo segna come ACKed e **se questo pacchetto era il più piccolo all'interno della finestra, avanza la finestra alla prima sequenza di pacchetti unACKed**.

Il ricevitore deve:

- Spedire l'**ACK del pacchetto ricevuto**: se non è in ordine, lo bufferizza, altrimenti consegna il pacchetto e gli eventuali bufferizzati e avanza la finestra.
- **Se riceve una copia di un pacchetto già ricevuto manda un ACK**

Eventuali problemi, possono essere legati alla dimensione della finestra. Notiamo come, nell'esempio che segue, il ricevitore non noti alcuna differenza.



Ma la differenza in realtà c'è: nel primo caso, il ricevitore interpreterà come nuovo, la copia di un pacchetto che ha già ricevuto, nel secondo caso invece il pacchetto è effettivamente nuovo.

Il numero di sequenza deve essere almeno il doppio della dimensione della finestra.

Ma la finestra non può essere grande a piacere: se la finestra è grande N, significa che devo bufferizzare, nel caso peggiore N-1: **la dimensione della finestra è vincolata dalla dimensione del buffer**.

PPP: Point to Point Protocol

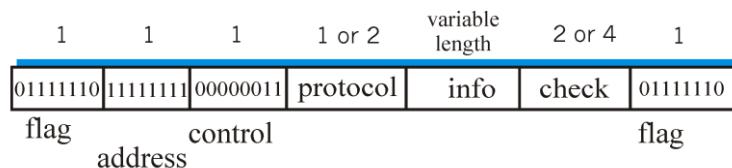
I protocolli visti finora di Reliable Data Transfer con o senza Pipeline non sono protocolli reali.
Il Point to Point Protocol è un protocollo reale del livello Data Link.

Il protocollo offre i seguenti servizi:

- Packet framing
- Bit transparency: deve portare qualsiasi tipo di dati
- Error detection
- Connection liveness: rileva e segnala collegamenti caduti al network layer
- Network layer address negotiation: permette di negoziare parametri livello data-link e network

Mancano: *error correction, flow control, ordine dei pacchetti*, non c'è supporto per comunicazione per punto-multipunto (le prime tre sono tutte delegate ai protocolli di livello più alto, come TCP).

Data Frame PPP:



Il primo e l'ultimo byte servono a segnalare rispettivamente l'inizio e la fine della comunicazione.

Se nel frame fosse presente **una sequenza di bit = al byte di flag**, il ricevitore può pensare che il frame sia finito, quando in realtà c'è altro da ricevere, motivo per il quale **viene posto una byte di escape prima del byte**.

Ma se si volesse trasmettere un byte = al byte di escape?

Si mette un byte di escape al byte = al byte di escape.

Questa tecnica si chiama: **byte stuffing o byte unstuffing**.

Configurazione dei parametri

Nel protocollo PPP ci sono dei parametri che si possono configurare:

- Si può negoziare il non utilizzo dei campi address e control
- Si può negoziare la max length del campo data (il campo info può avere dimensione compresa tra 0 e 1500 Byte)
- Eventuale procedura di autenticazione (protocollo nato senza)

Per poter negoziare questi parametri sono stati definiti dei protocolli di configurazione: **Link Control Protocol (LCP)**, utilizzati per negoziare i parametri di controllo a livello Data Link.

Nota: per negoziazione dei parametri livello rete (nella fattispecie nel caso di indirizzo IP) si usa protocollo IPCP (IP Control Protocol).

Attivazione di una connessione PPP:

1. Il PC chiama il router del provider via modem
2. Il modem del provider risponde
(Si stabilisce un collegamento fisico tra PC e router del provider)
3. Negoziazione dei parametri di link (protocollo LCP)
4. Negoziazione parametri di rete
5. Viene assegnato un indirizzo IP al PC
6. Il PC è ora collegato a Internet

Chisura di una connessione TCP

1. Protocollo IPCP:
 - a. Rilascio dell'indirizzo IP
 - b. Rilascio della connessione di livello rete
2. Protocollo LCP: Rilascio della connessione di livello Direct
3. Viene rilasciato il collegamento telefonico

I passi in grigio sono stati lasciati per motivi storici, ormai superati.

Multiple Access

Attraverso il collegamento punto-punto, *avendo n computer ci sarebbe bisogno di n(n-1)/2 collegamenti*: non è una buona soluzione, in quanto non scala (anzi scala quadraticamente, che è più o meno la stessa cosa).

Un approccio differente è quello di utilizzare un unico mezzo condiviso: utilizzando ***un link di tipo broadcast*** tutti i computer collegati a quel collegamento ricevono la stessa informazione.

È un approccio economico (un unico link, tanti computer), si può usare per scale geografiche non troppo elevate, ma presenta il problema delle collisioni.

I Multiple Access Protocols sono protocolli per gestire l'accesso multiplo.

Idealità che ricercano questi protocolli:

- Piena utilizzazione: se c'è un solo nodo che vuole trasmettere, deve farlo con rate R
- Parità: se m nodi devono trasmettere, ogni nodo deve trasmettere in media a R/M rate
- Completa decentralizzazione: ***no nodi speciali*** per coordinare la trasmissione e ***no sincronizzazione di clock***
- Semplicità

Possiamo classificare questi protocolli in 3 categorie:

1. **A partizionamento di risorse**: in cui si divide la risorsa, che viene usata dalla coppia trasmettitore-ricevitore. La risorsa potrebbe essere il tempo o ad esempio la frequenza.
2. **Random Access**: il canale non è suddiviso: *i nodi competono per l'utilizzo del canale*. Nel caso si verifichino collisioni si recupera dalla collisione.
3. **A turno**: i nodi si mettono d'accordo e trasmettono a turno.

Mac Protocols a Partizionamento di risorse: TDMA

Accesso multiplo basato sulla divisione di tempo.

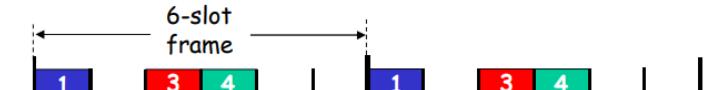
Ogni slot temporale è assegnato in qualche maniera ad un nodo, il quale può trasmettere solo in quella finestra temporale.

Pro:

- Fair

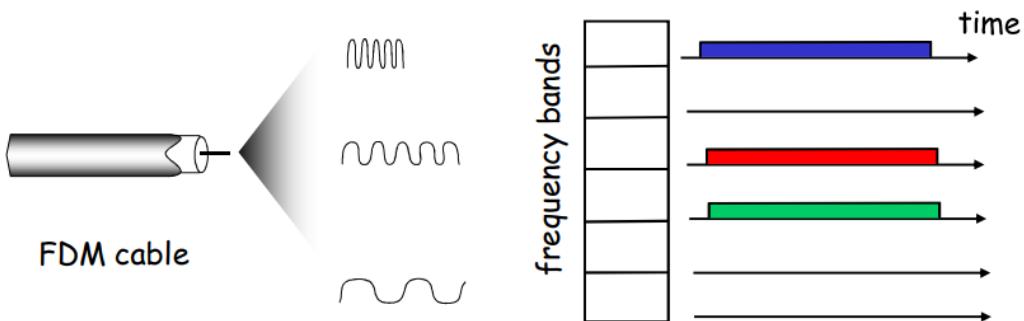
Contro:

- Inefficiente con bassi carichi, se c'è un solo nodo non può usare il mezzo al 100%



MAC Protocols a Partizionamento di risorse: FDMA

Accesso multiplo basato sulla divisione della frequenza.



MAC Protocols a Partizionamento di risorse: CDMA

Code Division multiple access: i due interlocutori si mettono d'accordo sul codice da utilizzare, il quale è unico per ogni coppia (il ricevitore riconosce solo quel codice e così riesce a decodificare quello indirizzato a lui).

Usato per lo più in ambito militare e telefonia cellulare.

MAC Protocols Random Access

In questi tipi di protocolli ogni nodo trasmette in maniera asincrona, può iniziare a comunicare in qualsiasi momento e perciò si possono verificare delle collisioni: per quanto possibile si cerca di evitare che si verifichino le collisioni, ma bisogna comunque essere pronti al recovery da esse.

Pro:

- Efficiente con bassi carichi, se c'è un solo nodo trasmettitore può usare il mezzo al 100%

Contro:

- Collision overhead con alti carichi

Slotted Aloha

Assunzioni:

- tutti i frame sono della stessa dimensione
- il tempo è diviso in slot di dimensioni uguali
- i nodi iniziano a trasmettere solo all'iniziare dello slot
- Se due o più nodi trasmettono allo stesso tempo, tutti i nodi si accorgono della collisione.

Come fa un nodo trasmettitore a sapere se si è verificata una collisione? Un ACK, ma la collisione non è l'unico dei problemi: il pacchetto si potrebbe essere corrotto.

Il trasmettitore in entrambi i casi non riceve indietro un ACK: il trasmettitore assume che si sia verificata una collisione: è una scelta più conservativa, in quanto non sa per quale motivo l'ACK non sia arrivato.

Supponiamo **ciascuno nodo trasmetta** con in uno slot (che si sia verificata una collisione o meno) con **probabilità p** e che i restanti $N-1$ nodi restino inattivi, dunque la probabilità che i rimanenti nodi non trasmettano è $(1-p)^{N-1}$.

Quindi la **probabilità di successo** di un dato nodo è **$p(1-p)^{N-1}$** .

Poiché ci sono N nodi, la probabilità che un nodo arbitrario abbia successo è **$Np(1-p)^{N-1}$** .

Di conseguenza, con N nodi attivi, l'efficienza dello slotted ALOHA è $Np(1-p)^{N-1}$.

Per ottenere la massima efficienza con N nodi attivi bisogna trovare il valore p^* che massimizza questa espressione, che risulta essere basta fare il lim per $N \rightarrow \inf$ di $Np^*(1-p^*)^{N-1}$ che è circa uguale a **1/e**.

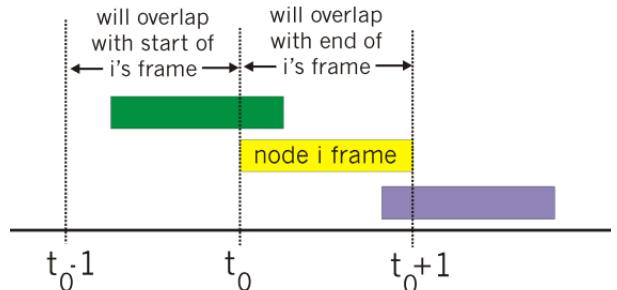
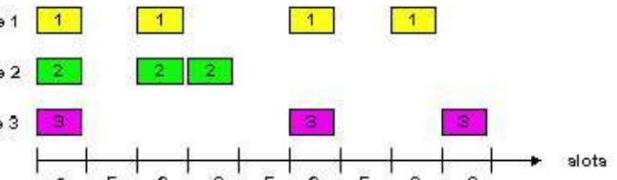
Contro: necessità di sincronizzazione del clock, e ha al massimo un'efficienza del 37%!

Unslotted Aloha

Più semplice rispetto a Slotted Aloha in quanto **non richiede sincronizzazione**: quando un trasmettitore ha qualcosa da trasmettere, trasmette. Come ovvio le **collisioni aumentano**.

Un frame trasmesso a t_0 , può collidere con un altro pacchetto mandato tra t_{0-1} e t_{0+1} .

Supponiamo **ciascuno nodo trasmetta con probabilità p** e che i restanti $N-1$ nodi non trasmettano, negli intervalli $[t_{0-1}, t_0]$ e $[t_0, t_{0+1}]$, con probabilità **$(1-p)^{N-1}$** , dunque la **probabilità di successo** di un dato nodo è **$p(1-p)^{2N-1}$** . Che per N nodi, trovando il valore p^* e al limite per N che tende a ∞ si ottiene che l'efficienza massima è circa $1/2e$.



La probabilità che un nodo trasmetta con successo è del **18%** → la metà rispetto Slotted Aloha

CSMA

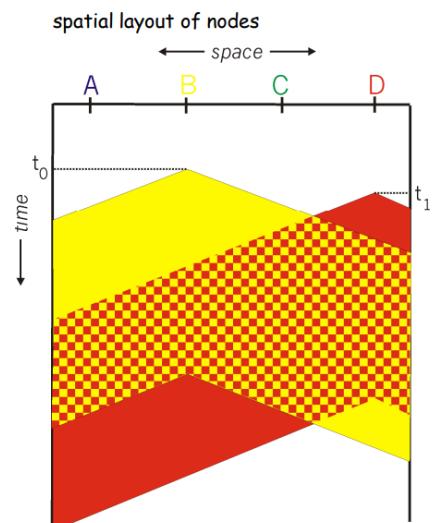
Carrier Sense Multiple Access: Protocolli che ascoltano prima di parlare, ovvero il problema principale di ALOHA, non ascoltava.

Se il mezzo fisico è occupato si differisce la trasmissione, se invece il mezzo è libera inizia a trasmettere.

Le collisioni possono ancora accadere.

CSMA/CD

In figura notiamo come B, in t_0 vede che il mezzo è libero ed inizia a trasmettere ("a sinistra e a destra", ovvero agli altri nodi), ma essendo **presente del ritardo di propagazione** D all'istante t_1 vede il mezzo ancora libero, in quanto non gli è ancora arrivato il messaggio da B, ergo inizia a trasmettere, ma come vediamo: avviene una collisione.



Meccanismo per fare collision detection, in modo tale da evitare di sprecare tempo, perché si manderebbero pacchetti e il ricevitore non li capirebbe: **il trasmettitore misura la potenza del segnale: se è significativamente maggiore** (c'è da tener conto del rumore presente) **rispetto a quella inviata, significa che qualcun altro sta trasmettendo e c'è una collisione.**

Una volta che un nodo ha scoperto la collisione, invia un **segnale di JAM**, cioè un segnale limitato che ha potenza elevata, così da far sì che tutti i nodi si accorgano della collisione.

Questo passaggio è necessario perché sulle lunghe distanze il segnale potrebbe essere attenuato a tal punto da non permettere alle altre stazioni di rilevarlo, generando quindi un'inconsapevole collisione.

MAC Protocols Protocolli a turno

Polling

Uno dei nodi, designato come **principale**, interpella “a turno” gli altri. In particolare, **il nodo principale invia un messaggio al nodo 1, comunicandogli che può trasmettere fino a un dato numero massimo di frame**.

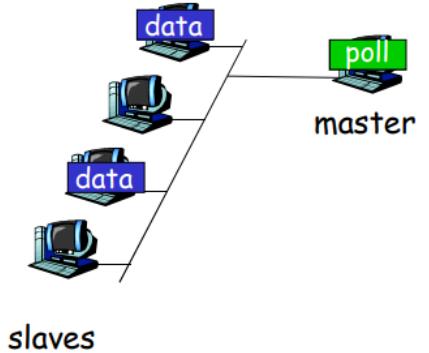
Dopo che il nodo 1 ha trasmesso dei frame, il nodo principale avvisa il nodo 2 che può iniziare a inviare un dato numero massimo di frame. **Il nodo principale può determinare che un nodo ha terminato di trasmettere, osservando la mancanza di segnale sul canale**. La procedura prosegue in questo modo, con il nodo principale che interpella in modo ciclico tutti gli altri nodi.

Pro:

- Elimina le collisioni
- Fair

Contro:

- Ritardo di polling: il tempo richiesto per notificare a un nodo il permesso di trasmettere
- Single point of failure: ma se il nodo principale cade, se ne elegge uno nuovo
- Se ci fosse un solo nodo, non può usare il 100% del mezzo trasmisivo, in quanto ciclicamente il nodo principale deve mandare il segnale di polling



Bluetooth utilizza il polling. Chi è il nodo master? Quello che ha iniziato la rete.

Passaggio del token

In questo protocollo può trasmettere solo chi possiede il token. Il token è un pacchetto: se un nodo non ha niente da trasmettere o se ha finito, lo passa al nodo vicino.

Pro:

- Decentralizzato

Contro:

- Guasto di un nodo o problemi nella trasmissione del token
- Single point of failure: ma se il nodo principale cade, se ne elegge uno nuovo

Indirizzamento fisico

Per poter fare una **comunicazione unicast su un mezzo broadcast è necessario** un indirizzo per identificare gli host.

Questo indirizzo detto **physical address o MAC address** (Media Access Control MAC), è **univoco per ogni dispositivo ed è impresso nella scheda di rete** (NIC): è un indirizzo di 48 bit, i primi 24 dei quali identificano il costruttore della scheda di rete e i 24 meno significativi sono assegnati dal costruttore.

Dunque quando si manda un messaggio si deve specificare l'indirizzo, ma siamo su un mezzo broadcast: il ricevitore vede, se il messaggio non è indirizzato a lui, scarta il messaggio, altrimenti lo consegna al livello superiore.

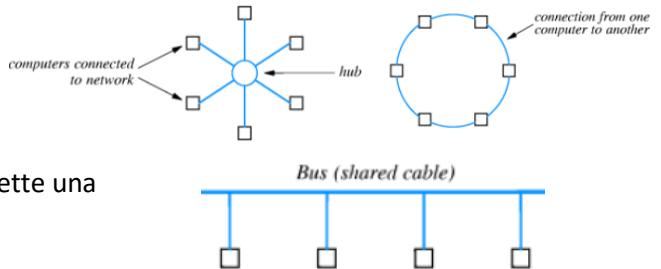
Per fare una comunicazione broadcast è necessario specificare l'indirizzo formato da tutti 1: FF-FF-FF-FF-FF.

Local Area Network (LAN)

Son reti che hanno copertura geografica locale (un edificio o parte di esso, un insieme di edifici) e sono caratterizzate da un mezzo di tipo broadcast e da un alto bit rate.

La topologia della rete locale non è sempre la stessa:

- A stella (con un hub nel centro)
- Ring, ad anello
- Bus, alla fine del bus il segnale si riflette
(bisogna evitarlo, ad esempio in Ethernet si mette una resistenza da 50 Ohm)



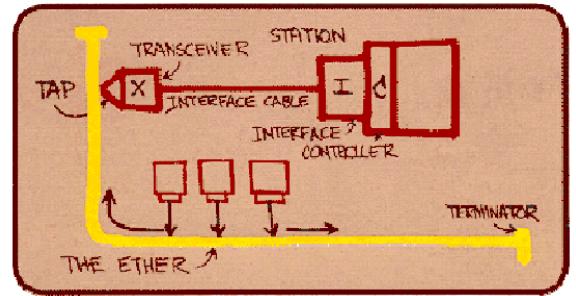
Una particolare tipologia di rete locale è Ethernet.

Ethernet

È la tecnologia di rete locale wired più diffusa.

Nella figura vediamo lo schema originale di questa metodologia: il transceiver fa parte dell'interfaccia oggigiorno e il terminatore è semplicemente una resistenza.

La proposta originale era quella di una rete con una topologia a bus, ma lo standard IEEE prevede due topologie:



- A bus, il quale costituisce l'unico dominio di collisione, in quanto il mezzo è unico, dunque se due nodi trasmettono collidono.
- A stella, anche in questo caso i nodi rappresentano un unico dominio di collisione, in quanto l'hub non fa niente: è solo un amplificatore.

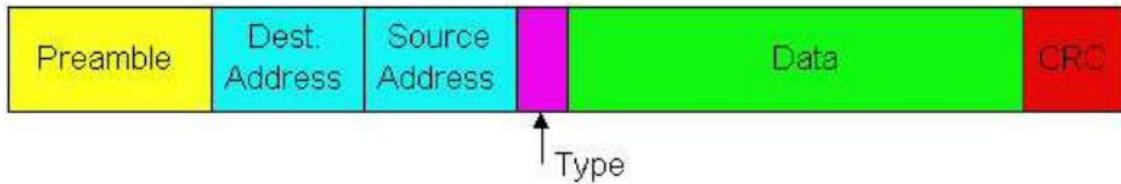
L'**Hub**, essendo un amplificatore (un dispositivo elettronico) opera solo a *livello fisico* e non a livello Data Link, dunque non “vede” i frame e i protocolli: per questo motivo si è pensato di sostituire l'hub con un dispositivo “intelligente”, o meglio con un dispositivo che opera a livello superiore: lo switch.

Lo switch opera a livello Data Link: avendo la visione dei frame, può bufferizzare i frame (store-and-forward), controlla l'indirizzo ed instrada i frame verso la linea di uscita corrispondente.

Inoltre, eccetto casi eccezionali (*broadcast e tabella di switching vuota*), esso non trasmette i frame su tutte le linee di uscita.

Infine, lo switch permette anche comunicazioni parallele tra dispositivi diversi della stessa rete, in quanto non vanno in conflitto poiché le connessioni verso lo switch sono dedicate.

Frame Ethernet



- Il preambolo ha una struttura particolare, sono **8 byte**:
 - I primi 7 byte sono tutti 10101010...10
 - L'ultimo byte è 10101011

Il preambolo serve a sincronizzare trasmettitore e ricevitore, cosicché il ricevitore possa sincronizzare il proprio clock su quello del trasmettitore. ***Si usa la codifica Manchester, ovvero una codifica che mette insieme informazioni e clock.***

- Il campo destinatario e sorgente sono da **6 byte ciascuno**
- Il tipo indica il protocollo di livello superiore (tipicamente IP), **2 byte**
- I dati, massimo **da 46 a 1500 byte**
- CRC, per fare error detection, **4 byte**

Nota: la codifica Manchester si ottiene facendo lo XOR dei dati con il clock: quando si trasmette uno 0 si fa una variazione da un livello basso a uno alto, quando si trasmette l'1 la situazione è capovolta.

Il ricevitore usando il circuito anello ad aggancio di fase, riesce ad estrarre il clock del trasmettitore.

Il preambolo non fa parte del frame ethernet, in quanto riguarda informazioni del livello fisico.

La dimensione minima di un frame ethernet è 512 bit (64 byte).

Servizi offerti da Ethernet

- Ethernet è un servizio non affidabile, in quanto non fa error correction e nemmeno ritrasmissione: **i pacchetti corrotti non vengono passati al livello superiore.**
- È senza connessione, non ci sono fasi di hand-shaking.

Il protocollo di accesso è il CSMA/CD, dato che il mezzo è un mezzo condiviso.

Algoritmo CSMA/CD per Ethernet

- Il trasmettitore attende che il canale di comunicazione sia libero** per **96 bit** (ovvero il tempo necessario a trasmettere 96 bit, ipotizzando che il mezzo trasmissivo abbia un rate di 10Mbps, quindi equivale a 9.6 micro secondi).
- Come fa ad accorgersi che il canale è libero: misura la potenza del segnale, se è sotto una certa soglia molto vicina allo zero (ci sono i disturbi da considerare) allora il mezzo è libero
- Mentre trasmette, il trasmettitore ascolta, ovvero misura la potenza del segnale in entrata: **se questa è maggiore rispetto a quella del segnale in uscita allora è avvenuta una collisione invia un segnale di JAM per 48 bit.**
- Nel caso in cui sia avvenuta una collisione, quando ritrasmette?

Ogni dispositivo che ha rilevato una collisione genera un tempo di backoff. Il dispositivo deve aspettare questo tempo.

Dopo la prima collisione, il dispositivo genera un numero tra 0 e 1 e moltiplica poi per 51.2 microsecondi (k512 bit time).

Dopo la n-esima collisione, genera **un numero tra 0,1,2... (2^n)-1.**

Dopo 10 volte, la finestra non viene più incrementata. Quindi il massimo è tra 0 e 1023 ($(2^{10})-1$).

Dopo 17 collisioni consecutive, il pacchetto viene buttato via.

Perché l'algoritmo è esponenziale? Perché il nodo non sa quanti sono quelli coinvolti nella collisione, in modo tale da diminuire la probabilità di collisione.

Perché non scegliamo da subito una finestra grande? Si diminuiscono le probabilità di collisione, ma rischiamo di aspettare tanto inutilmente (hanno colliso in due) e di lasciare inutilizzato il mezzo.

Reti basate su packet switching

Lo switch è un dispositivo trasparente, gli host non sono consapevoli di essere collegati ad uno switch.

Lo switch è un dispositivo di tipo plug-and-play: non c'è bisogno di configurazione, questo perché è un dispositivo self-learning.



Lo switch permette comunicazioni contemporanee tra più nodi, questo perché i nodi hanno collegamenti dedicati nodo-switch.

Lo switch, dunque, garantisce un throughput n-volte superiore rispetto ad un hub.

Lo switch mantiene al suo interno una **switching table**, con gli indirizzi **di destinazione – linee di uscita**, la quale viene creata automaticamente, in quanto il dispositivo è **self-learning**.

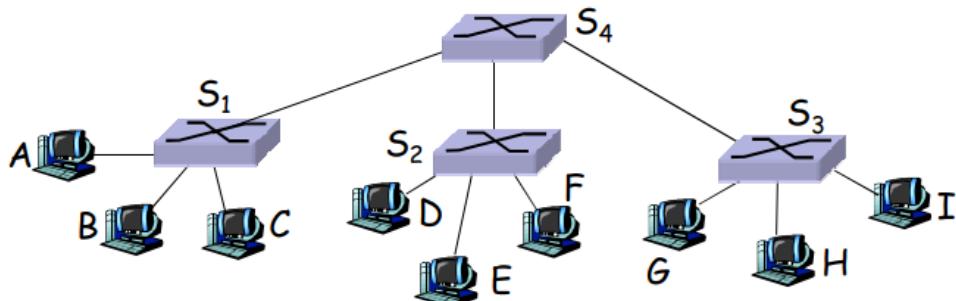
Switch utilizzati in Ethernet

Sono switch normali, ma dal punto di vista fisico hanno le caratteristiche dello standard IEEE.

Ogni switch ha un numero limitato di porte: e se si vogliono connettere un numero di dispositivi maggiore rispetto al numero di porte?

Si interconnettono gli switch tra loro, attraverso una struttura ad albero di switch.

La struttura ad albero si adatta bene al cablaggio di un edificio: ad esempio si può usare uno switch per edificio.

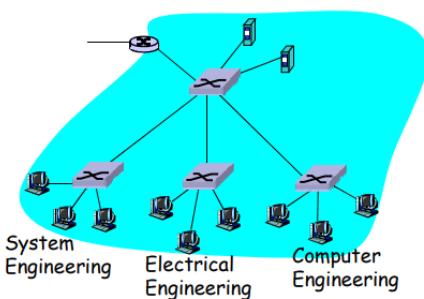


Nelle reti istituzionali solitamente ci possono essere host che devono essere accessibili dall'esterno, come web server e mail server, che per motivi di sicurezza e per motivi di velocità, si tende a collegare "direttamente" questi server allo switch radice.

Proprietà dello Switched Ethernet

- Si elimina l'unico dominio di collisione
- Con lo switch possono essere **supportati collegamenti eterogenei** tra di loro, ad esempio in una stessa rete ci possono essere collegamenti o host con **bitrate** molto diversi tra loro. Lo switch può autoregolare la propria velocità
- **Facilità di gestione dell'errore:** se una porta non funziona più, basta isolare quella porta e la rete continua a funzionare
- **Sicurezza:** fare lo **sniffing** è più **difficile** (la comunicazione non è più broadcast)

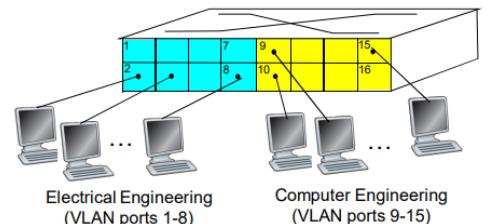
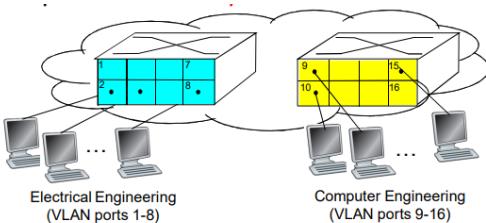
VLANs



Cosa non va con questa configurazione di rete? È vero che non abbiamo un unico dominio di collisione (e va bene), ma abbiamo un unico dominio di broadcast (il che può essere un male), il che può portare ad alcuni problemi di sicurezza; quindi, le comunicazioni broadcast attraversano tutta la rete.

Altro problema: ci potrebbe essere un uso inefficiente dello switch.

situazioni sono le **Virtual Lan**: si utilizza un singolo switch sul quale si definiscono più LAN virtuali, assegnando le porte alle VLAN.
L'assegnamento delle porte è dinamico, una porta può essere assegnata a una VLAN piuttosto che un'altra in base alle esigenze.

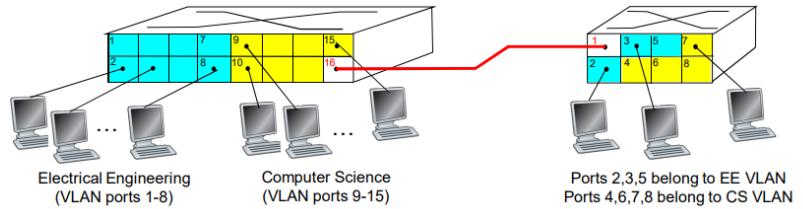


In entrambi i casi dal punto di vista funzionale, funzionano come **due switch separati**.

Quindi se si vuole inviare un frame da una VLAN all'altra bisogna necessariamente passare dal router!

Nel caso di VLAN però, il router è interno e non esterno!

Oltre ad avere più LAN virtuali su un unico switch, si possono definire anche delle LAN virtuali su più switch fisici: in questo caso è necessario un link che collega gli switch, quindi bisogna "sacrificare" una porta, detta **trunk port**, per switch.



Bisogna aggiungere *informazioni* quando si inoltra un frame: a quale VLAN è destinato il frame, **VLAN id**.
Questa informazione non fa parte del frame ethernet: bisogna aggiungere dei campi nel frame ethernet.

Original Ethernet Frame

Destination MAC	Source MAC	Length or Type	Data	Original FCS
-----------------	------------	----------------	------	--------------

802.1Q Tagged Frame

Destination MAC	Source MAC	TAG	Length or Type	Data	New FCS
-----------------	------------	-----	----------------	------	---------

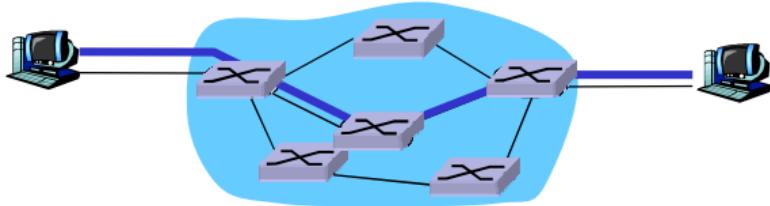
Type	Priority	Token Ring Encapsulation	VLAN ID
------	----------	--------------------------	---------

Questo nuovo frame non può essere consegnato direttamente all'host! Ci deve rimettere mano lo switch di destinazione.

Wide Area Network Switched Ethernet

Nel caso in cui, in una rete a copertura geografica, si tenesse la stessa topologia ad albero degli switch sarebbe un problema: è presente un unico point of failure.

Per questo motivo è meglio scegliere una topologia a maglia.



In questo caso, per identificare i dispositivi sulla rete, è necessario un indirizzo fisico (**NON MAC address, non è un link broadcast**).

Servizi:

- Connectionless, le informazioni possono essere inviate senza avere la connessione. Anche chiamato **datagram service**. **Switched Ethernet** è basato su questo.
- **Connection**, le informazioni prima di essere inviati, deve essere stabilita una connessione (connessione virtuale, non necessariamente fisica).

Per le WAN possono essere utilizzati entrambi i servizi.

ATM

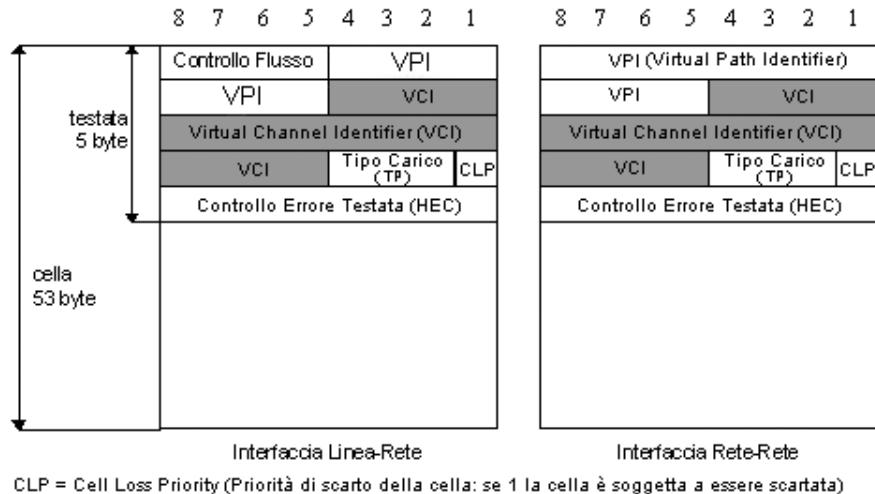
Asynchronous Transfer Mode, ATM è una tipologia di rete progettata agli inizi degli anni novanta e lanciato con una fortissima spinta dai grandi operatori di telecomunicazioni in quanto avrebbe dovuto soddisfare le esigenze di networking unificando varie tipologie di traffico o servizi (voce, dati, TV via cavo, telex, ecc.), all'interno di un sistema unico integrato, soddisfando i requisiti di integrità dei dati.

Basata su packet switching, con pacchetti di dimensione fissa, ma non troppo elevata a causa della necessità di trasmettere la voce, la quale non può avere troppo ritardo.

Tipologie di traffico:

- Constant Bit Rate, questo per la voce, si produce un bit rate fisso.
- Variable Bit Rate, le sequenze video hanno piccole variazioni solitamente, ma capita anche che le variazioni siano tante. Ad esempio si può codificare la differenza rispetto al quadro precedente o al pixel precedente nello stesso quadro.
- Available Bit Rate, per le applicazioni che hanno bisogno di un throughput minimo e poi si prendono "quello che c'è", se il throughput è maggiore molto meglio, altrimenti "pazienza".
- Unspecified Bit Rate, nessuna garanzia, l'equivalente del best effort di Internet.

Formato cella ATM



- Il VPI (*Virtual Path Identifier*) identifica il *path* virtuale su cui il circuito virtuale è stato attivato.
- Il VCI (*Virtual Circuit Identifier*) identifica il canale virtuale su cui il circuito virtuale è stato attivato.
- HEC, equivalente dei bit di risonanza, ma si fa error detection solo sull'header: perché? Queste reti sono pensate per collegamenti in fibra ottica (con error-rate molto bassi).

Circuito Virtuale VC

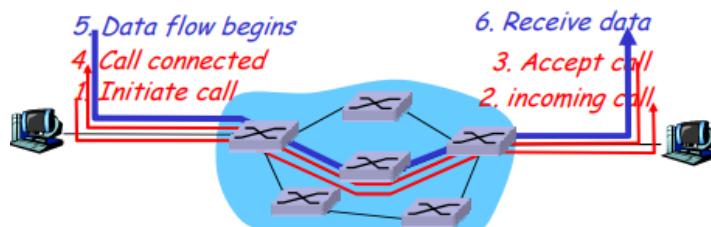
Un *VC* è normalmente un servizio orientato alla connessione cioè con una modalità di trasferimento a circuito fisso dedicato tra due nodi.

Un Circuito Virtuale è composto da:

1. Un **percorso**, formato da switch tra sorgente e destinazione
2. Un **id**
3. Entrate nella forwarding table di ogni switch

Per stabilire un circuito virtuale

- **Prima che i dati possano fluire**, bisogna stabilire una connessione e fare una *call setup*. Il sorgente specifica l'indirizzo del destinatario e aspetta per la risposta di accettazione della connessione. Viene definito il percorso attraverso gli switch, i quali settano anche l'id della VC per quella connessione. Durante il setup, può anche essere deciso di riservare risorse lungo il percorso da parte degli switch, come ad esempio la banda.



- Stabilita la connessione, possono iniziare **a fluire i dati**: ogni pacchetto trasporta il VCI e non l'indirizzo del destinatario: sulla connessione ci sono solo loro due, funziona come il collegamento telefonico.
- **Tutti le celle, una volta stabilito il VC, seguono lo stesso percorso.**
- Ogni switch sul percorso deve mantenere lo "stato" della connessione: informazioni sulla connessione virtuale e sulla porta di ingresso uscita.

Il **VIC** è *indipendente su ogni switch*: quindi uno **stesso canale può avere tanti id diversi**, a seconda dello switch, arriva con un id e riparte con un altro.

Incoming interface	Incoming VC #	Outgoing interface	Outgoing VC #
1	12	3	22
2	63	1	18
3	7	2	17
1	97	3	87
...

Se si fosse deciso di avere un **VIC** a comune lungo tutto il percorso, questo avrebbe comportato uno scambio di pacchetti per negoziare questo numero, il che avrebbe comportato un incremento importante dell'overhead.

- La **connessione viene chiusa** e vengono rimossi i record dalla tabella di forwarding degli switch riguardanti la connessione.

Il VC ha il vantaggio di garantire la QoS, ma lo svantaggio di introdurre overhead per stabilire la connessione e di non essere scalabile.

Layer 3: Network (o IP level)

Internetworking: introduzione

Cos'è un inter-rete? Una inter-rete è una rete di reti, dove le reti sono eterogenee: hanno mezzi diversi, diversi formati del frame e diversi schemi di indirizzamento.

È necessario, dunque, un interprete per fare comunicare tra di loro le reti: questo interprete è un dispositivo, il router.

Il router, a differenza dello switch, **lavora a livello 3**, il livello di rete.

Ma tutta questa infrastruttura, la rete di reti, deve risultare trasparente all'utente: deve apparire come un singolo sistema.

È necessario un protocollo affinché l'internetworking funzioni: **l'Internet Protocol (IP)**.

Questo protocollo **ha lo scopo di creare l'astrazione dell'inter-rete e di permettere il trasporto trasparente dei pacchetti** (chiamati *datagram* a questo livello) attraverso l'inter-rete.

Questo protocollo deve lavorare su tutti gli host e su tutti i router intermedi.

Routing vs Forwarding

Forwarding: il processo attraverso il quale, quando un datagram arriva a un *router*, il quale legge l'indirizzo di destinazione del pacchetto, si decide verso quale collegamento instradare il pacchetto.

Routing: processo che serve per decidere qual è il percorso meno costoso per andare dall'host sorgente a quello destinatario.

Come viene fatto il forwarding?

Il router possiede una tabella, detta **tabella di forwarding**, che il router consulta prima di instradare un pacchetto.

Chi riempie la tabella di forwarding? Il processo di routing.

L'IP è **connectionless** e l'**instradamento si fa per ogni singolo datagram**, andando a guardare l'indirizzo di destinazione e consultando la tabella di forwarding.

Vantaggi:

- Non c'è da stabilire la connessione o da chiudere la stessa, come nelle reti ATM
- I router devono mantenere le minime informazioni indispensabili per fare l'instradamento, quindi ad esempio la tabella di forwarding.

Internet (*datagram*) vs ATM (*virtual circuit*)

La prima nasce come rete per gestire dati generati da computer, la seconda nasce come l'evoluzione della rete telefonica: i due tipi di traffico sono estremamente diversi, nel secondo caso il flusso di informazioni è continuo.

Nel caso di internet, gli host sono dei dispositivi intelligenti, nell'ATM invece i dispositivi sono "stupidi": questo porta ad approcci differenti, infatti nell'ATM la filosofia è "tieni la complessità nel core della rete", mentre in *Internet* la filosofia è "*tieni la complessità ai margini*", così da mantenere il core semplice.

L'ultima porta anche ad avere approcci differenti: nell'Internet è più funzionale il connectionless mentre nell'ATM è più funzionale il connection (perché i terminali sono "stupidi").

Nel caso di Internet però non si garantisce nulla, non si garantisce consegna affidabile, in ordine e nemmeno una banda minima, un massimo ritardo (servizi che nell'ATM sono garantiti).

La QoS (Quality of Service) di Internet è Best Effort.

Il router

Il router svolge due funzioni essenziali:

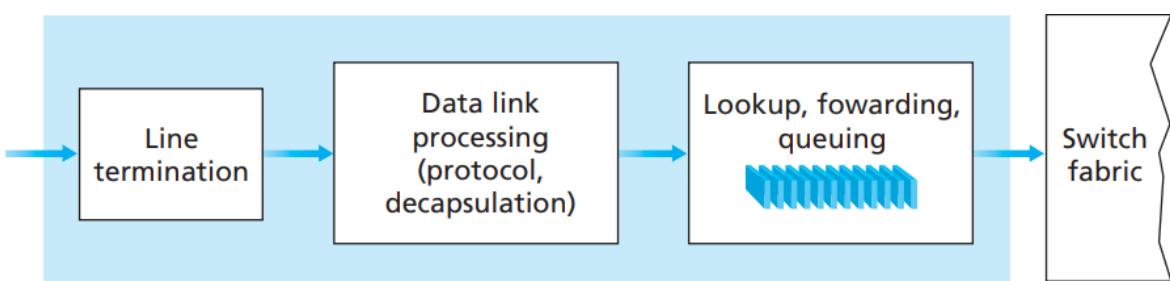
- Esegue il forwarding tra i link in ingresso e in uscita
- Esegue gli algoritmi e i protocolli di routing (RIP, OSPF, BGP)

Il router ha un certo numero di porte in ingresso e di uscite (le porte sono di tipologia full-duplex) e da un sistema di commutazione, ovvero un sistema che è in grado di commutare un pacchetto dalla porta di ingresso alla porta di uscita.

Funzionalità delle porte di ingresso

Le porte di ingresso svolgono diverse funzioni chiave;

- Svolgono **le funzionalità del livello fisico e del livello data link** (protocolli e decapsulamento) in modo tale da interagire con il data-link layer dall'altra parte del collegamento.
- **Lookup-function:** consulta la tabella di forwarding (è presente una copia in ogni singola porta dopo essere stata elaborata e aggiornata dal processore) per decidere in quale porta di uscita deve essere instradato il pacchetto arrivato.
- Viene gestito l'accodamento
- *I pacchetti di controllo (quelli che portano informazioni riguardanti il routing ad esempio) sono inoltrati al routing processor.*



Logica di commutazione

La logica di commutazione può essere implementata in diversi modi:

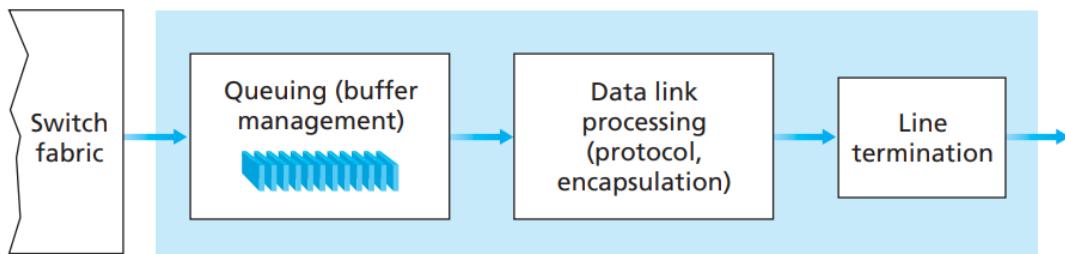
- Attraverso una **memoria condivisa** (caso più semplice): le porte d'ingresso e di uscita sono da immaginare come dei processori che fanno tutte le operazioni, con della memoria condivisa, così da potere realizzare la logica di commutazione attraverso la memoria condivisa.
Il dover fare diversi accessi alla memoria condivisa rallenta il processo di commutazione.
- Attraverso un **bus**, il quale collega le varie porte: basta trasmettere sul bus.
È più veloce ma c'è un limite: il bus è unico, problema dell'accesso multiplo.
- **Crossbar switch**: si possono fare più instradamenti contemporanei tra una porta di ingresso e una di uscita (purché ingresso e uscita siano differenti).
È la soluzione più veloce in quanto si possono fare più trasferimenti parallelamente.
È anche la soluzione più costosa.

È presente una rete dentro il router!

Funzionalità delle porte di uscite

Una volta commutato, il datagram arriva alla porta di uscita: questo potrebbe non essere il primo, ergo è presente un buffer in cui si crea una coda.

Lo scheduling della coda di uscita non necessariamente è FCFS, ad esempio si può fare dello scheduling basato sulla priorità basata sulla QoS del datagram che si deve inoltrare (ad es: Weighted for Queueing).



Protocollo IPv4

È un protocollo di interconnessione di reti (Inter-Networking Protocol), classificato al livello di rete, nato per **interconnettere reti eterogenee** per tecnologia, prestazioni, gestione, pertanto implementato sopra altri protocolli di livello data link.

È un protocollo a pacchetti **connectionless e di tipo best effort**, che non garantisce cioè alcuna forma di affidabilità della comunicazione in termini di controllo di errore, controllo di flusso e controllo di congestione, che può essere invece realizzata dai protocolli di trasporto di livello superiore, come TCP.

Il protocollo IP fornisce le seguenti funzionalità:

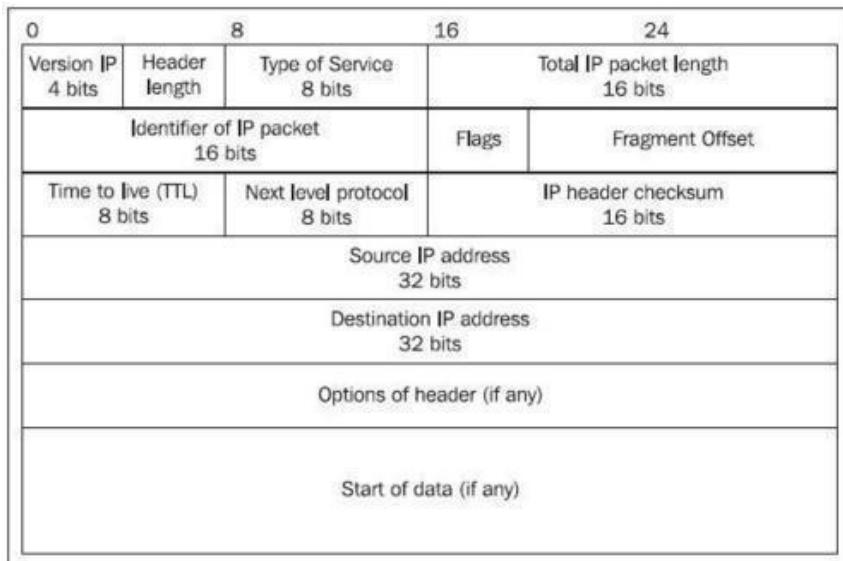
- Definisce il formato dei datagram
- Definisce lo schema di indirizzamento sull'internet
- Si occupa dell'inoltro dei pacchetti

Formato dei datagram

Poiché il protocollo IP lavora a livello di internetworking e serve a collegare reti eterogenee tra di loro, è necessario stabilire un nuovo formato dei pacchetti (datagram). Questo perché, essendo le reti eterogenee, potrebbero implementare protocolli di livello 2 differenti tra di loro.

Da questa stessa esigenza nasce un problema:

Problema dell'incapsulamento: i *payload di livello inferiore* (Ethernet, Wi-Fi, ecc) hanno dimensione diverse tra loro e rispetto all'IP (max 64Kbyte, Ricordiamo che: Ethernet 46-1500 Byte, Wi-Fi: 0-2312 Byte): si risolve con la frammentazione.



- **IP version:** primi 4 bit, IPv4
- **Lunghezza dell'header:** 4 bit, perché alcuni campi sono opzionali. Dici quante parole da 32 bit contiene l'intestazione: al minimo sono 5 parole da 32 bit, quindi 20 byte.
- **Tipo di servizio:** 8 bit, tipo di dati.
- **Lunghezza totale del pacchetto IP:** 16 bit, indica la dimensione totale del datagram (intestazione + dati): al **massimo** può essere 2^{16} , ovvero **64 Kbyte**.
- **Identificatore del pacchetto IP:** serve per la frammentazione del datagram, ogni datagram è identificato da un numero (se avessi più datagram da inviare alla stessa destinazione e provenienti dalla stessa sorgente, nella fase di riordino se questo campo non ci fosse, rischierei di mischiare i datagram)
- **Flags:** Uno dei tre flag serve per sapere se il frammento che arriva è l'ultimo o meno (**FragFlag** = 1 se ci sono segmenti successivi, 0 se è l'ultimo pacchetto)
- **Fragment Offset:** ordinamento dei frammenti, un po' come segnarvi sopra 1°, 2°... In realtà c'è scritto l'offset del primo byte del frammento rispetto all'inizio del datagram. Se il datagram fosse grande 64Kbyte, mi aspetto che gli ultimi frammenti, abbiano un offset vicino a 64 Kbyte: per risolvere questo problema, l'offset non è esattamente uguale al primo byte ma lo si divide per 8, in modo tale da far entrare l'offset in 13 bit.
- **TTL:** 8 bit, inizializzato a tutti 1 (decimale: **255**): inteso come quanti hope può fare il datagram (ovvero salti attraverso i router).
- **Protocollo di livello superiore:** 16 bit, a quale protocollo di livello superiore bisogna consegnare il payload? TCP o UDP, tipicamente.
- **Checksum dell'header:** 16 bit, viene calcolato prima di essere spedito in fase di invio, e ad ogni instradamento, perché cambia il TTL!
Si fa solo sull'header per due motivi: non introdurre troppo ritardo (calcolare il checksum su tutto il datagram farebbe impiegare troppo tempo) e perché l'IP ha un approccio best effort.

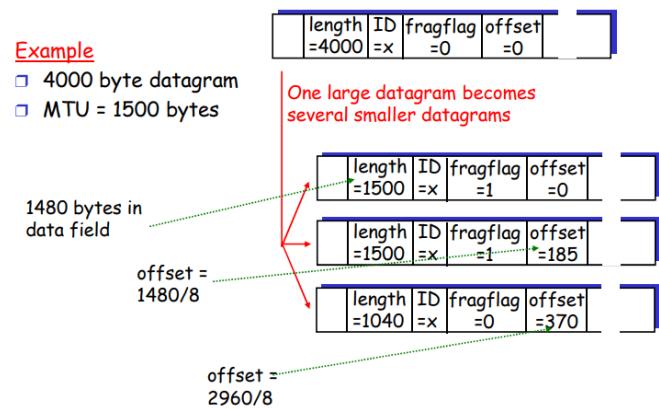
- **Source IP address:** 32 bit, nelle reti locali l'ordine era invertito (prima destination e poi source address), perché? Perché era necessario sapere se il pacchetto fosse destinato all'host che riceve il pacchetto per copiare o scartare il pacchetto.
- **Destination IP address:** 32 bit
- **Opzioni:** non sono sempre presenti.
- **Data:** lunghezza variabile, solitamente un segmento TCP o UDP.

Incapsulamento

Il problema principale dell'incapsulamento non è tanto il fatto di dover rispettare una dimensione massima (**Max Transfer Unit, MTU**) per la dimensione del datagram IP: il problema è che lungo il percorso da host a host questa dimensione cambia poiché ogni sottorete può usare un protocollo di livello data-link differente.

Il secondo problema sta anche nella questione: dove vengono riassemblati i pacchetti?

Per restare coerenti con il principio di tenere la complessità ai bordi della rete, questi vengono riassemblati alla destinazione finale, ma questo complica sia i router che gli host finali, in quanto i primi si trovano costretti talvolta a dover frammentare i pacchetti e i secondi a doverli riassemblare. Inoltre, se un pacchetto non arriva, poiché il servizio IP è di tipo best effort, l'host finale è costretto a scartare il pacchetto e non consegnarlo ad un livello superiore.



Infine, la segmentazione può essere usata per creare attacchi di tipo DoS: l'hacker potrebbe iniziare a mandare pacchetti piccoli e strani, ad esempio nell'attacco Jolt2 l'attaccante manda tanti frammenti ma tutti con fragFlag=1, cosicché il destinatario collassi aspettando di ricevere l'ultimo frammento, che non arriverà mai.

Indirizzamento IPv4

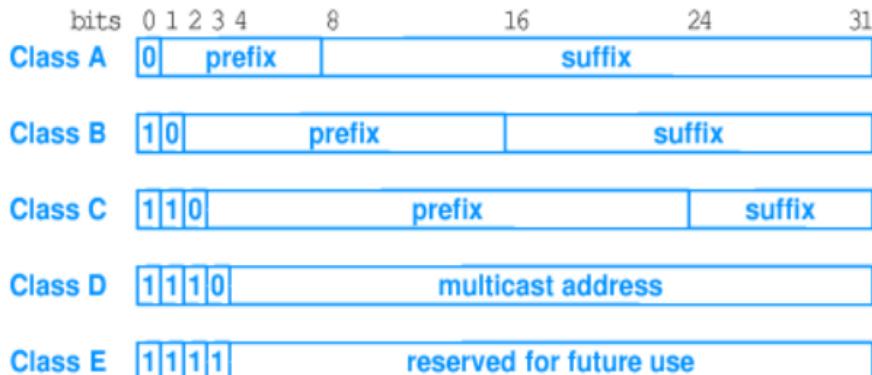
L'indirizzo IPv4 è un indirizzo a 32 bit che serve ad identificare una singola interfaccia di rete dell'host (o del router).

L'indirizzo IP è un indirizzo strutturato: è presente una parte che identifica la sottorete e una che identifica l'host.

Nota: sottorete è un insieme di interfacce che posseggono la parte che indica la sottorete uguale.

Ogni interfaccia collegata alla sottorete può raggiungere le altre senza passare dal router.

Un tempo, l'indirizzo IP era class-based:



Class	Range
A	0 - 127
B	128 - 191
C	192 - 223
D	224 - 239
E	240 - 255

Qual era il problema principale: supponiamo di avere una rete da più di 254 host: un indirizzo di classe C non basta, ma servirebbe un indirizzo di classe B: ma questo può contenere fino a 16Ki host! Si sprecerebbero indirizzi. Oppure sarebbe necessario avere più reti di classi C. Ma aumentando il numero di host connessi in rete, il numero di indirizzi IP disponibili iniziava a diminuire. Così, per ovviare a questi due problemi si sono introdotti: **l'indirizzamento classless** e il **protocollo NAT**.

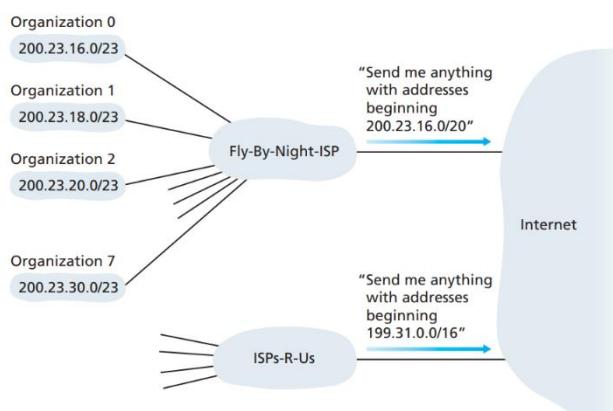
Classless InterDomain Routing (CIDR)

Il formato degli indirizzi IP cambia e diventa:

x.x.x.x/n

dove n è il numero di bit che identifica la sottorete, è la subnet mask!

L'approccio classless più efficiente per lo spazio degli indirizzi e per i router che si trovano nel core in quanto diminuisce il numero di entrate che vanno inserite nella tabella di forwarding.



L'approccio classless infatti permette anche **l'address aggregation**: questo permette di riferirsi a più reti, come visto nell'esempio, le reti 200.23.x.x/2x possono essere riferite in prima approssimazione con 200.23.x.x/20

Ottenere un indirizzo IP: organizzazione

Per ottenere un blocco di indirizzi da usare all'interno della sottorete della propria organizzazione, essa deve contattare il proprio ISP, il quale avrà allocato un certo numero di indirizzi IP. L'ISP andrà a guardare tra quelli disponibili ed in base alle esigenze dell'organizzazione darà un certo indirizzo di rete con una certa subnet.

Ma a sua volta l'ISP può:

- Dato un indirizzo, dividerlo in blocchi di uguali dimensioni → ad esempio un blocco/20 può farne 8 /23, ovviamente ognuna di queste reti potrà indirizzare meno host
- Deve chiedere gli indirizzi all'ISP di livello superiore fino ad arrivare all'ICANN (Internet Corporation for Name and Number)

Ottenere un indirizzo IP: host

Un host in una rete può avere un indirizzo IP statico o dinamico.

IP statico:

- Va inserito manualmente nella configurazione dell'interfaccia di rete
- Bisogna stare attenti a non assegnare più volte lo stesso indirizzo a host differenti (bisogna ricordarsi dunque nel DHCP di togliere gli IP statici dal pool di indirizzi)
- Non può avvenire il riciclo degli indirizzi: se un host si disconnette dalla rete, il suo IP, finché lui è scollegato, teoricamente può essere usato da un altro.

IP dinamico:

- Plug and Play: quando un dispositivo si connette alla rete, può iniziare a navigare perché ha un IP
- Riciclo degli utilizzati e nessuno spreco
- Necessità di un nuovo protocollo: DHCP

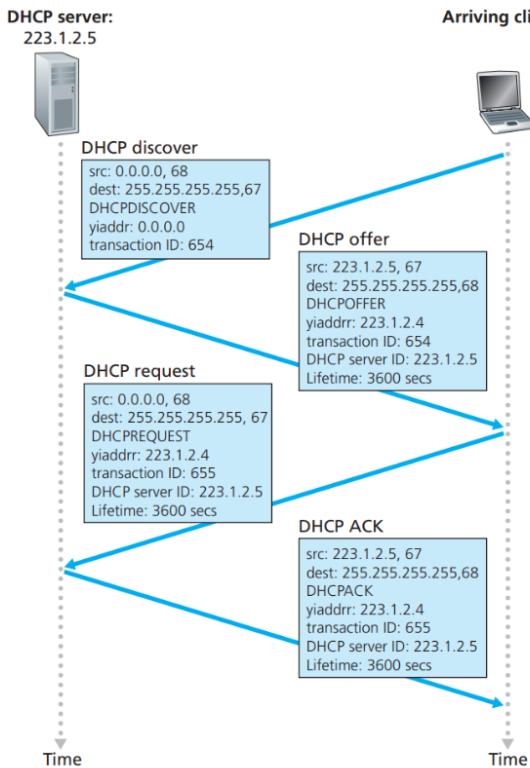
Dynamic Host Configuration Protocol

Il DHCP è un **protocollo applicativo** (*ausiliario all'IP livello 3*) che permette ai dispositivi di una certa sottorete di ricevere automaticamente la configurazione IP necessaria per potersi connettere alla rete, ovvero:

- Indirizzo IP
- Subnet Mask
- Indirizzo di rete, di broadcast (impliciti) e default gateway

È un protocollo di tipo client-server, è dunque necessario un **server DHCP**, questo server DHCP risponde alle richieste, fornendo gli indirizzi, prelevandoli dal proprio pool di indirizzi.

Possono esserci più server DHCP all'interno di una stessa sottorete.



- **DHCP DISCOVER**: l'host appena collegato alla rete non conosce e non possiede né l'indirizzo della sottorete né il suo indirizzo; quindi, manda una richiesta per ottenere un indirizzo IP in broadcast (se configurato al DHCP)
- Il server risponde con un'offerta di indirizzo IP (**DHCP OFFER**), risponde sempre in broadcast, in quanto l'host non ha ancora l'IP. Nella risposta è presente anche un lifetime, ovvero per quanto è valido l'IP fornito.
- L'host risponde con un **DHCP REQUEST sempre in broadcast**, per richiedere "formalmente" l'indirizzo IP proposto
- Il server risponde con un **DHCP ACK**, sempre in broadcast, in quanto ancora l'host non ha l'IP e non solo.

Notare come il **Transaction ID** serve **per identificare a chi sta rispondendo il server** (dato che sono comunicazioni in broadcast, poiché i client non hanno ancora un indirizzo), viene generato casualmente dall'host: è molto, molto difficile che due client generino stesso transaction ID.

Inoltre, le **ultime comunicazioni sono in broadcast**, affinché tutti i DHCP server sappiamo quale IP è stato concesso e accettato (dall'host: questo perché potrebbero esserci più DHCP server sulla rete e tutti devono togliere l'indirizzo IP assegnato dal pool di indirizzi assegnabili).

Il fatto che un indirizzo abbia un tempo di vita permette di riutilizzare gli indirizzi quando un dispositivo si scollega.

L'uso del DHCP non preclude l'uso di indirizzi IP statici e permanenti: anzi, in caso di server o dispositivi condivisi come la stampante è meglio utilizzare questo tipo di approccio.

Indirizzi riservati

Network Number	Host Number	Description	Notes
all 0s	all 0s	"this node"	Used at startup
x	All 0s	Network Address	Identify network x
x	all1s	Broadcast Address	datagram sent to all nodes of network x
all 1s	all1s	Restricted Broadcast Address	datagram sent to all nodes of the local network
127	--	Loopback Address	Used when developing applications

Restricted Broadcast: broadcast sulla rete locale.

Network Address Translation

È un approccio complementare rispetto al DHCP: consiste nello sfruttare la caratteristica degli indirizzi IP, alcuni sono pubblici e altri sono privati. I pacchetti con campo src/dst un indirizzo IP privato non possono circolare nella rete.

In breve, grazie al NAT, ad una rete locale può essere associato un solo indirizzo IP pubblico, a prescindere dalla dimensione della rete locale stessa, sfruttando la manipolazione dell'IP e della porta sul datagram.

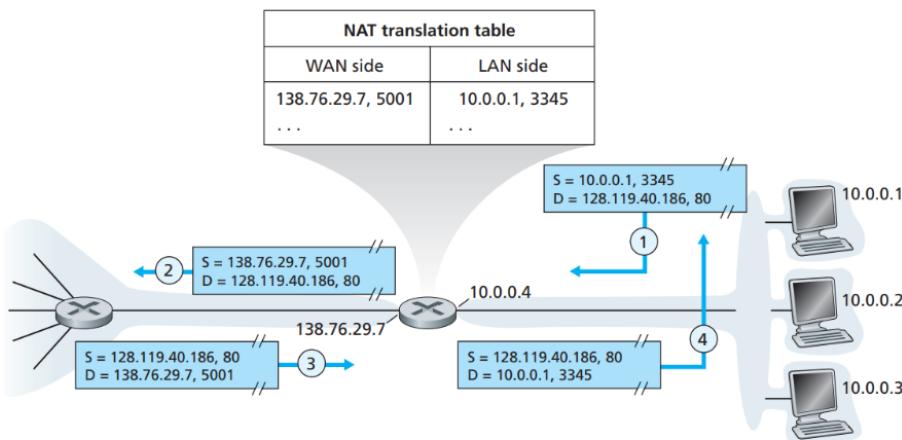
Sostanzialmente il router viene visto dal mondo esterno come un singolo dispositivo con un singolo indirizzo IP pubblico (da dove prende questo IP il router? Dal DHCP server dell'ISP).

Vantaggi per una sottorete:

- Non c'è bisogno di un range di indirizzi forniti dall'ISP, **basta un indirizzo IP**. Risparmio di indirizzi IPv4 a livello globale.
- **Si possono cambiare gli indirizzi ip degli host all'interno della rete senza comunicarlo al di fuori della rete**
- Si può **cambiare ISP senza modificare gli indirizzi**

È necessario un NAT server: nelle reti domestiche, solitamente il “modem” è un computer special purpose che svolge i compiti di: modem, router, switch, DHCP server, NAT server.

Nel router-NAT è presente la NAT translation table.



- Il **client** manda una richiesta al router indirizzata ad un host esterno, **mettendo nel campo src del datagram il proprio IP (privato) e un numero di porta** (casuale, associato ad un socket)
- Il router **modifica il datagram**: pone come **IP src l'IP pubblico**, e **cambia il numero di porta** con uno non utilizzato al momento.
(L'IP sarà sempre lo stesso per tutti i router, **la porta si cambia perché potrebbe capitare che due host generino lo stesso numero di porta** e non si potrebbe poi più riconoscere a quale host della rete privata è indirizzata la risposta).

Il fatto che il numero di porta sia su 16 bit ci permette in linea teorica di supportare fino a 64 Mila comunicazioni in contemporanea.

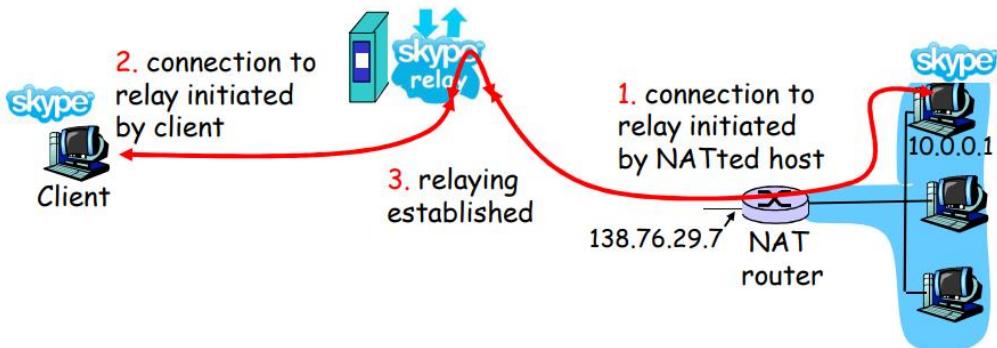
- Il router memorizza l'associazione sulla NAT table
- Il server esterno risponde ponendo nel datagram come IP dst quello pubblico e la porta modificata dal NAT
- Il router esegue la traduzione controllando la NAT table e recapita il pacchetto all'host corretto.

Come abbiamo visto la **NAT table si riempie ogni volta che un host della sottorete esegue una richiesta.**

Ma se ci fosse un server nella sottorete?

Il NAT non saprebbe a quale host privato è relativa la richiesta arrivata dalla rete pubblica, in quanto non è presente un'entrata nella NAT table e scarterebbe la richiesta.

1. È necessario dunque **inserire staticamente un'entrata** della NAT table (e assegnare un indirizzo statico al server).
2. UPnP: **Universal Plug and Play**, gli host imparano il proprio IP pubblico e chiedono al router di riservargli una porta (un esempio classico è BitTorrent che utilizza porta 5001)
3. **Relaying** (usato in skype)



Forwarding

- **Nei router:**

<u>SubnetNumber</u>	<u>SubnetMask</u>	<u>NextHop</u>	<u>Interface</u>
128.96.34.0	255.255.255.128	Router R1	interface 0
128.96.34.128	255.255.255.128	Router R3	interface 1
128.96.33.0	255.255.255.0	Router R3	interface 1
...

```
DHost=Destination IP Address
For each entry [i] in Table {
    DNet=(SubnetMask[i] & Dhost)
    If(DNet==SubnetNumber[i])
        then deliver datagram to NextHop[i] through Interface[i]
}
```

- **Negli host:**

```
SubnetNum=MySubnetMask & Dest_IP_Addr
If(SubnetNum ==MySubnetNum)
    then deliver datagram to Dest_IP_Addr directly
else forward datagram to default router
```

Address Resolution Protocol: ARP

Quando bisogna inviare un **datagram bisogna incapsularlo a livello data link** (come dipende dal protocollo livello data link) per poi mandarlo al default router, ad un altro router o ad un altro dispositivo.

Ma l'indirizzo a livello data link del destinatario, a prescindere da chi spedisce a chi, **come si ottiene?**

I router, o in generale, il livello rete, conosce l'indirizzo IP e non quello fisico.

1. L'host sorgente invia un **ARP request in broadcast**:
“chi ha questo indirizzo IP mi può fornire il proprio indirizzo hardware? Mi fornite l'ind. hardware?”
2. Chi ha quell'indirizzo IP ***risponde*** con il MAC address.
3. **L'host salva nella ARP table l'associazione IP-MAC.**

ARP è plug and play: gli host creano la loro ARP table senza l'intervento dell'admin di rete.

Il problema della **ARP table** memorizzata negli host è che funziona come una cache: dopo un po' di tempo ***potrebbe diventare obsoleta*** (un dispositivo cambia MAC o IP address), per questo motivo l'associazione “ha una data di scadenza” tipicamente di ***20 minuti***.

Nel router, che collega due reti, sono presenti due ARP table, una per ogni sottorete.

ICMP

Un protocollo ausiliario è ICMP che ***serve ad inviare notifiche***, le quali sono solitamente notifiche di errore.

Usato dagli *host e dai router per comunicare a livello di rete riguardo informazioni dello stesso livello*.

Per ***ogni messaggio ICMP ci sono un tipo e per ogni tipo possono esserci più codice***. Ad esempio:

Tipo | Codice

3	0	dest network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable (quell protocollo non è gestito nella rete di destinazione)
0	0	echo reply ping
8	0	echo request ping

Per i Test di connettività Ping e traceroute basati su ICMP si veda la dispensa sulla parte di laboratorio.

Routing

Quando un datagram viene trasmesso da un host sorgente a una destinazione al di fuori della propria rete, il problema principale è quello dell'instradamento, ovvero dopo essere passato per il default gateway, qual è il **next-hop**? Qual è il prossimo router che porta il datagram a percorrere il percorso migliore? Quello che gli fa fare meno strada? Quello che ci mette di meno?

Solitamente si rappresenta la rete come un grafo nel quale dobbiamo trovare il percorso di costo minimo, ma il costo da ottimizzare potrebbe non essere uno solo e, nel mondo reale, potrebbe esserci dei problemi legati **alle policy delle organizzazioni**.

- **Algoritmi di routing globali:** i router hanno la conoscenza di tutta la rete, sono detti **Link State Algorithms**
- **Algoritmi di routing decentralizzati:** ovvero hanno una conoscenza parziale sulla rete e sono basati sui vettori della distanza, chiamati infatti **Distance Vector Algorithms**.
- **Algoritmi statici:** si basano sul fatto che la rotta non cambi spesso, nel senso, sempre solito traffico, sempre solito link.
- **Algoritmi dinamici:** le rotte cambiano rapidamente, aggiornamenti periodici in risposta ai cambi dei costi della rete.

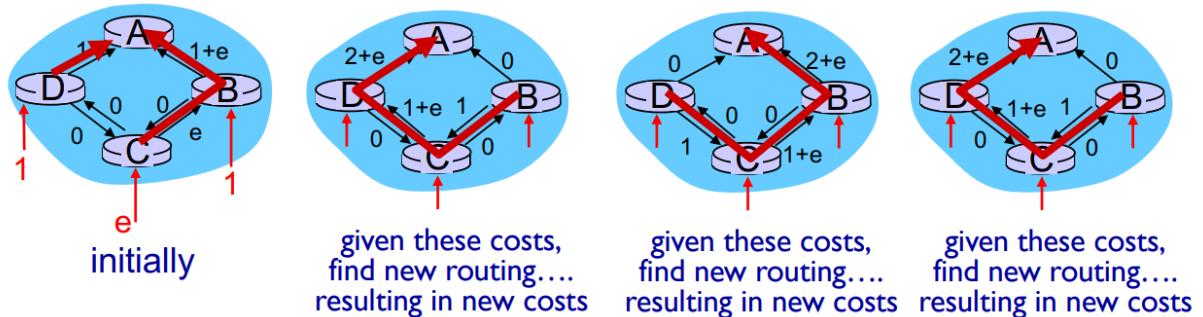
Link State Algorithm

Si usa l'**algoritmo di Dijkstra**: algoritmo ottimo che permette di trovare il costo minimo da un nodo ad un altro.

Avviene attraverso un messaggio di broadcast iniziale, che chiede a tutta la rete per avere informazioni.

Algoritmo di complessità **$O(n^2)$** , se più efficiente si può fare $O(n \log n)$.

Se i costi sugli altri sono dinamici si può verificare il problema dell'oscillazione:



Vale la pena usare l'**algoritmo di Dijkstra** quando le reti sono statiche, ovvero quando il livello di congestione cambia talmente poco che non è conveniente rieseguire Dijkstra.

Distance Vector Algorithms

Bellman-Ford equation (dynamic programming)

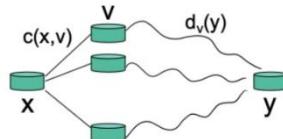
let

$$d_x(y) := \text{cost of least-cost path from } x \text{ to } y$$

then

$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

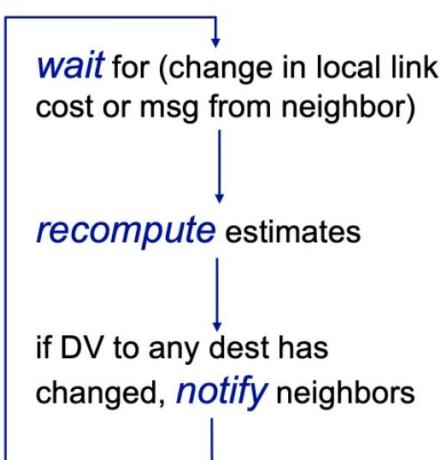
min cost from v to destination y
 cost to neighbor v
 min taken over all neighbors v of x



Sia $D_x(y)$ una stima del costo minimo per arrivare da x a y .

- Ogni nodo X conosce i costi $c(x,v)$ per arrivare ai suoi vicini e per ogni vicino un vettore di stime del costo minimo da X a tutte le destinazione y.
- Quando cambia un costo su un ramo verso un vicino o quando cambia un vettore delle stime, ciascun nodo spedisce il suo vettore di stima delle distanze ai vicini.
- Quando un nodo X riceve un nuovo vettore delle stime, aggiorna il suo vettore usando algoritmo Bellman-Ford usando le informazioni ottenute.
- Se l'esitameted distance vector verso un qualche nodo cambia, lo notifica a tutti i suoi vicini e si riniza da capo.

$$D_x(y) \leftarrow \min_v \{c(x,v) + D_v(y)\}, \forall y \in N$$



Sotto condizioni, la stima $D_x(y)$ l'algoritmo converge all'effettivo costo minimo $d_x(y)$.

Algoritmo iterativo asincrono: ogni iterazione casuata da: un costo locale cambiato, da un aggiornamento di DV (stima del nodo V).
è un algoritmo distribuito: ogni nodo manda le modifiche del proprio distance vectore solo ai propri vicini.

Programmazione dinamica: si basa su dividere il problema in sottoproblemi per trovare una sequenza di soluzioni in sottoproblemi che porti all'ottimo.

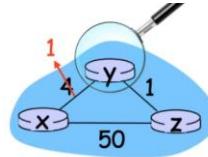
L'algoritmo però non lavora sulla conoscenza esatta del vettore della distanza, ma su una sua stima, ma sappiamo che questo algoritmo sotto determinate condizioni converge.

Problema con algoritmo Bellman- Ford, Distance Vector è **Count to Infinity**: è un problema legato al fatto che un nodo ha delle informazioni parziali sulla topologia della rete.

Se un Distance Vector migliora in generale non ci sono problemi, ma se uno peggiora ce ne potrebbero essere e anche grossi: potrebbe viaggiare sulla rete un'informazione falsa che si propaga a macchia d'olio.

link cost changes:

- ❖ y detects local link cost change
- ❖ updates routing info, recalculates distance vector
- ❖ if DV changes, notify neighbors



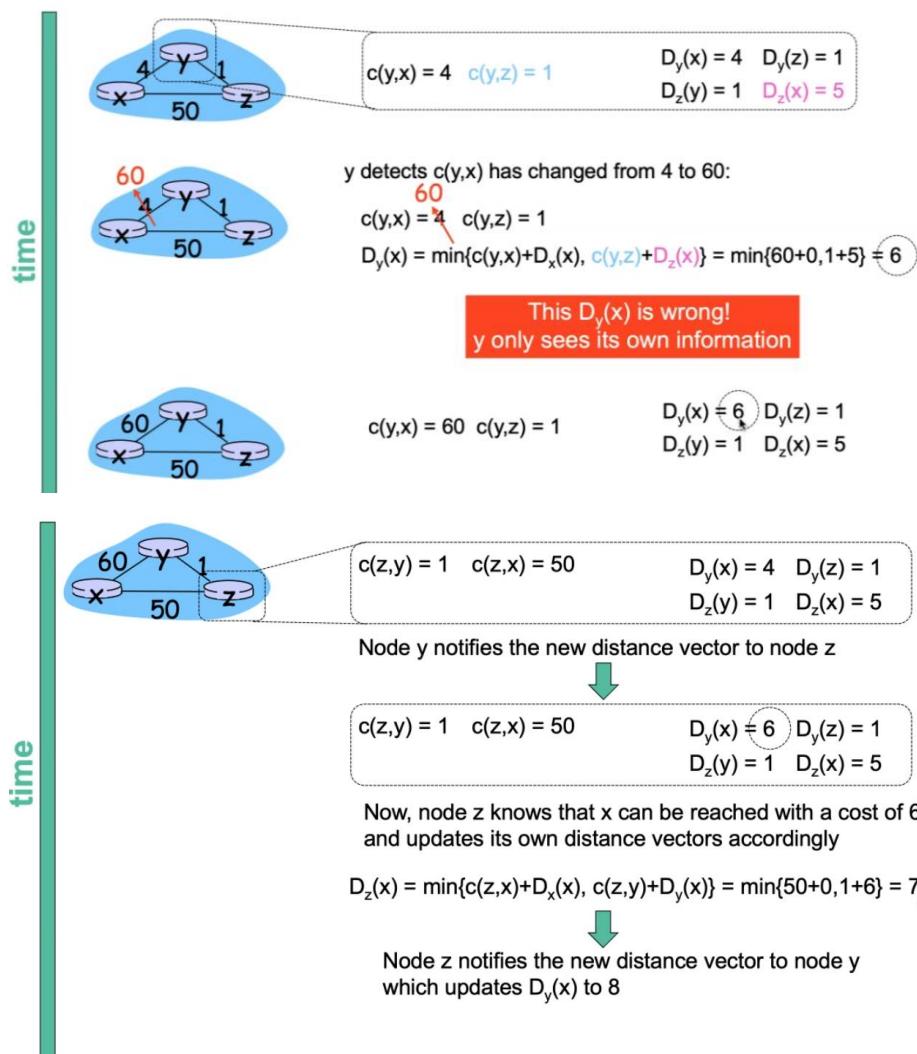
“good news travels fast”

t_0 : y detects link-cost change, updates its DV, informs its neighbors.

t_1 : z receives update from y, updates its table, computes new least cost to x, then sends its neighbors its DV.

t_2 : y receives z's update, updates its distance table. y's least costs do not change, so y does not send a message to z.

Problema legato alla conoscenza parziale: obsoleta.

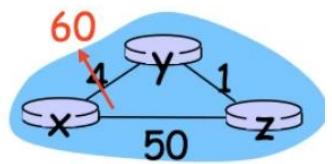


Poisoned reverse: modo per cercare di risolvere il problema dell count to infinity.

Se un router z passa attraverso y per andare a x, lui dice a y che *il suo percorso da z a x costa infinito*, in modo tale che y non passi da z.

Questo meccanismo non sempre funziona o comunque non sempre porta dei benefici importanti.

Nella rete vengono utilizzati dei meccanismi più avanzati.



Il problema del count to infinity è che aumenta il numero di messaggi scambiati sulla rete! Quindi la rete viene congestionata da messaggi che non hanno senso di essere scambiati.

Comparison of LS and DV algorithms

message complexity

- ❖ **LS:** with n nodes, E links, $O(nE)$ messages sent
- ❖ **DV:** exchange between neighbors only
 - convergence time varies

speed of convergence

- ❖ **LS:** $O(n^2)$ algorithm requires $O(nE)$ messages
 - may have oscillations
- ❖ **DV:** convergence time varies
 - may be routing loops
 - count-to-infinity problem

Robustness

what happens if router malfunctions?

LS:

- node can advertise incorrect *link cost*
- each node computes only its own table

DV:

- DV node can advertise incorrect *path cost*
- each node's table used by others
 - errors propagate through the network

Problema di robustezza:

- **LS:** se un link è troppo congestionato, rischia di saperlo solo il router che ci è collegato e gli altri no, per i link state algorithm. (Si sistema solo rireseguendo Dijkstra, dopo comunicazione broadcast)
- **DV:** possono propagare stime sbagliate sulla rete, ad esempio posso pubblicizzare un cammino di costo minimo sbagliato, ad esempio di un qualcosa che è diventato saturo

Routing ereditario

Routing ereditario: con oltre 600 milioni di destinazioni, non si può dire che tutto Internet funzioni tutto a Dijkstra o a Distance Vector.

Soluzione: raggruppare insiemi di router e definire il concetto di sistema autonomo.

Un **sistema autonomo** è un insieme di router appartenenti ad esempio alla stessa organizzazione.

All'interno di ciascuno sistema autonomo ci può essere un qualsiasi algoritmo di routing (Ed è lo stesso per tutti i router all'interno di uno stesso AS).

Infatti sono presenti algoritmi di routing INTRA-AS e INTER-AS.

All'interno degli Autonomous System le forwarding table sono popolate sia dagli algoritmi INTRA-AS sia da quelli INTER-AS.

Se un router deve spedire un pacchetto al di fuori del suo autonomous system deve prima sapere qual è il percorso per uscire più velocemente (hot-potato-routing), ovvero deve trovare qual è il border-router che lo fa uscire dalla rete. Come lo fa a saperlo? Quando i border router hanno saputo come si raggiunge un certo AS lo fanno sapere a tutti gli altri router della stessa AS.

IGP (Interior Gateway Protocols)

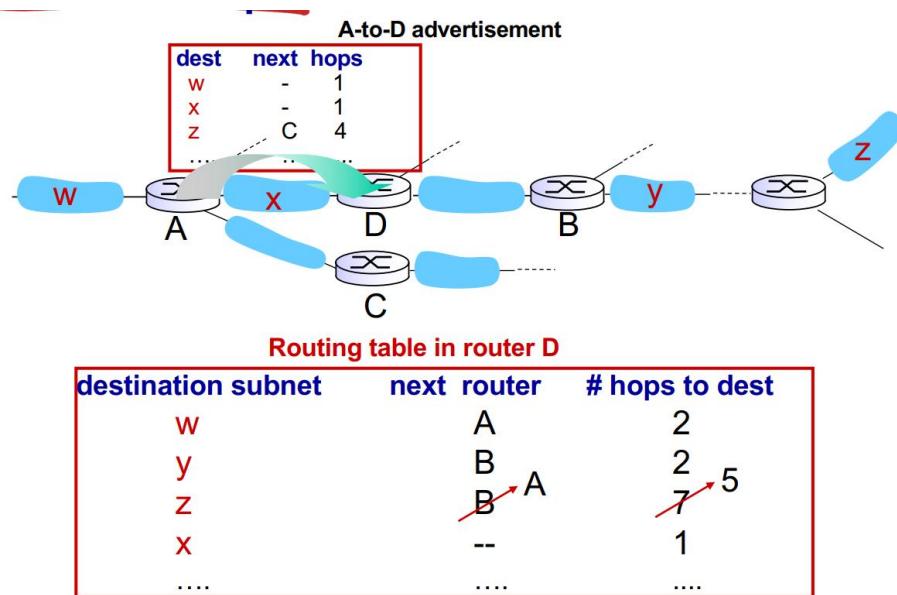
Sono i protocolli di Intra-AS.

RIP (Routing Information Protocol)

È un **Distance-Vector algorithm** che cerca di minimizzare il numero di salti dalla sorgente alla destinazione.

Il costo su ogni ramo è 1.

Il messaggio di advertisements che contiene i distance vector vengono scambiati ogni 30sec con i vicini. Per quale motivo? In quanto la topologia di rete può cambiare.



OSPF (Open Shortest Path First)

È un **algoritmo link state** che si basa sulla dissimnezione di pacchetti link state, ovvero riguardanti la tipologia della rete (siamo all'interno di un AS).

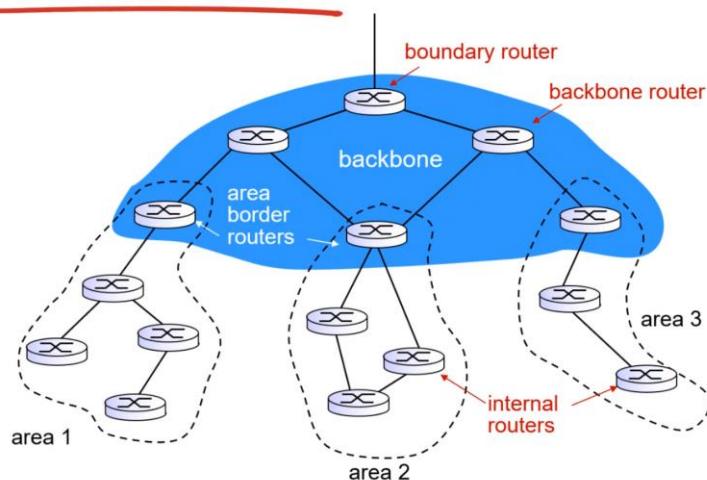
La computazione dello shortest path è fatta utilizzando l'algoritmo di Dijkstra.

OSPF advertisement porta un'entra per ogni vicino.

Se la rete è molto grande però

Routing gerarchico: è basato sul divere i router appartenenti ad un AS in più famiglie, creando una gerarchia

Hierarchical OSPF



Se la rete è molto grande, si decide di definire ***più OSPF area***, altro non sono che una suddivisione dell'AS (nel disegno 3 aree) dove all'interno delle varie aree si può usare un algoritmo intra-as, solitamente link-state (nel nostro caso proprio OSPF). ***Così facendo si diminuisce il volume di messaggi di flooding.***

L'area blu è chiamata ***area 0 o backbone***, area di collegamento tra le varie aree OSPF.

In ogni area è presente un area border router noto ai router della area stessa, i quali devono inoltrare i datagram a lui per far sì che questi possano uscire al di fuori dell'area.

Gli algoritmi della backbone usano l'algoritmo OSPF.

Questa è una metodologia per partizionare AS molto grandi.

Cosa succede se si vuole uscire dall'AS? Si passa dal boundary router.

BGP (Border Gateway Protocol), algoritmo inter-AS

È il collante di Internet.

Il protocollo fornisce a ciascuno AS un mezzo per:

- **eBGP**, external BGP: ottiene le informazioni di raggiungibilità dagli AS vicini.
Si preoccupa di capire quali AS vicini portano quali prefissi di rete esterni.
- **iBGP**, internal BGP: si occupa di propagare le informazioni ai router interni. Es, se il border router scopre tramite eBGP, che una certa rete è raggiungibile, propaga l'informazione agli altri router dell'AS.

Determinare una buona rotta sulla base delle informazioni di raggiungibilità della altre reti e sulla base della policy.

Permette inoltre alle sottoeti di avvertire della loro esistenza al resto di Internet.

Policy

Questioni politiche, per esempio:

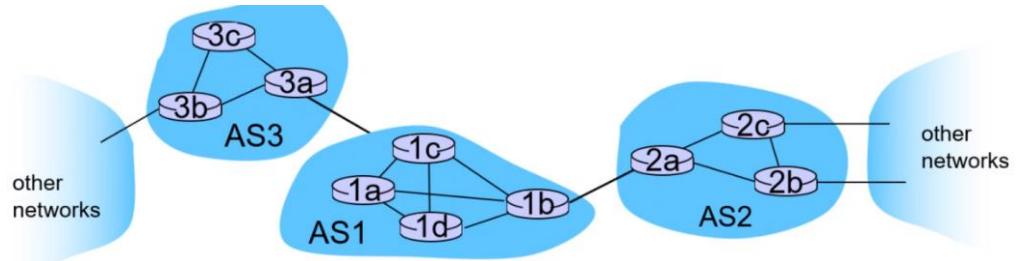
- Un AS aziendali potrebbero non essere disposti a trasportare i pacchetti generati da un AS estraneo o concorrente anche se si trova sullo shortest path.
Potrebbe però essere disposto a farlo per gli AS che hanno ***pagato il servizio di transito***.
Infatti le aziende sono molto contente di farlo per i propri clienti, ma non per quelli dei concorrenti.
- ***Nessun traffico commerciale su reti di ricerca***
- Es: il traffico da/verso Apple non deve passare attraverso Google

Le politiche di routing sono proprietarie e individuali e servono a decidere quale traffico può fluire su quali linee.

BGP basi

BGP è un protocollo path vector che si basa sullo scambio di advertisements i quali contengono un percorso, ma a livello di AS.

Supponiamo di essere in questa situazione:

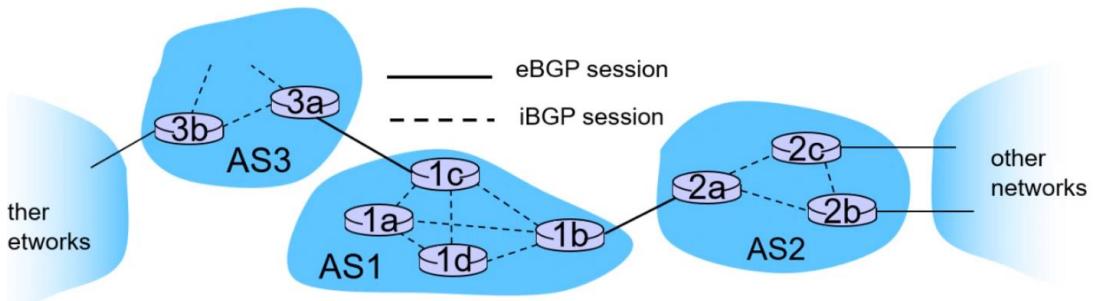


Quando AS3 avvisa di un prefisso a AS1:

- AS3 promette che inoltrerà i datagram per quel prefisso (o per qui prefissi, possono essere anche aggregati).

Cosa succede?

- C'è una sessione **eBGP** nella quale un AS informa l'altro della conoscenza di un percorso verso un determinato prefisso
- Il router di frontiera del secondo AS informa gli altri componenti dell'AS con una sessione di **iBGP**.



Gli advertisements sono una coppia (prefisso, attributi)

❖ **Prefix: 138.16.64/22 ; AS-PATH: AS3 AS131 ;
NEXT-HOP: 201.44.13.125**

- Il prefisso è la destinazione
- Attributi:
 - **AS-PATH:** contiene tutti gli AS tramite i quali l'advertisement è passato, ad esempio AS67, AS17
Così ripercorrendolo all'indietro c'è il path
 - Next-hop: indica qual è il prossimo router da contattare per raggiungere la destinazione. Se è una sessione eBGP il router sarà il border router che deve contattare l'altro border router; se la sessione è iBGP il router sarà sempre un border router, ma sarà quello che devono contattare i router interni per uscire dall'AS.

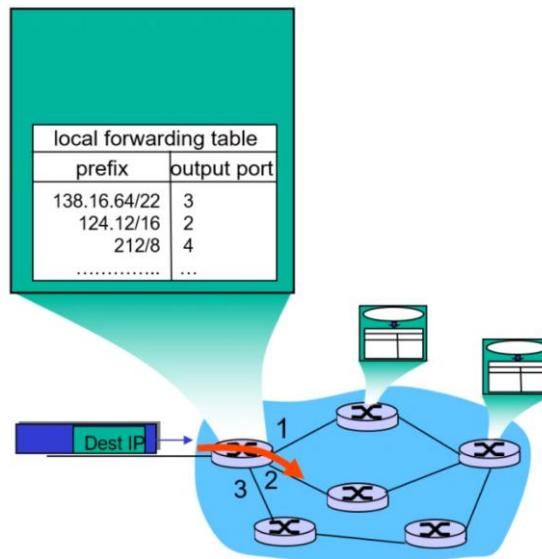
I router potrebbero imparare più di una rottura per un AS destinazione, la rottura è selezionata in base a:

- Decisioni politiche (infatti il border router potrebbe anche decidere di non accettare l'offerta di una rottura e non farlo sapere ai router interni)
- Shortest AS-PATH
- Next-hop più vicino
- Altri criteri

Il problema dell'avere però il path determinato da quanti salti tra gli AS si fanno potrebbe portare il router a non prendere la scelta migliore: potrebbero esserci molti hop all'interno di un AS che sta nello shortest path.

Quindi come viene determinata la porta di uscita da un router interno?

1. Si usa il BGP route selection basato su shortest AS-Path
2. Vengono usati i protocolli OSPF per determinare la migliore strada intra-AS (**hot potato routing**)
3. Viene scelta identificata la porta di uscita per quella rottura



scale:

- ❖ hierarchical routing saves table size, reduced update traffic

performance:

- ❖ intra-AS: can focus on performance
- ❖ inter-AS: policy may dominate over performance

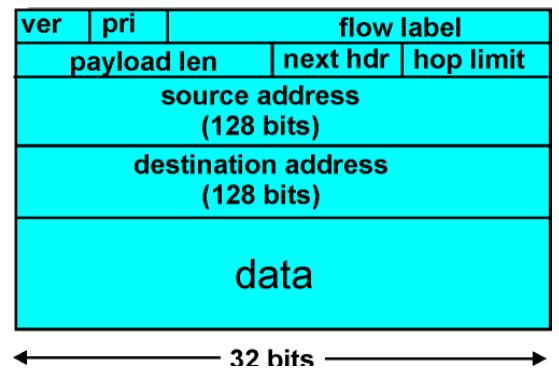
IPv6

Quali sono le motivazioni che hanno spinto ad “aggiornare” il protocollo IPv4 in IPv6:

- **IPv4 ha solo 2^{32} indirizzi**, 4Gi indirizzi, i quali con il diffondersi di Internet e dell’IoT risultano essere insufficienti.
- **Aumentare la velocità di processazione e di forwarding** del datagram: *la lunghezza variabile dell’intestazione e il checksum da ricalcolare ogni volta a causa del TTL aumentava questi tempi con i datagram IPv4*
- **Header IPv6 supporta la qualità del servizio**: nonostante il protocollo rimanga best-effort, il formato del pacchetto supporta la QoS
- **Eliminata frammentazione, il checksum e le opzioni** per aumentare la velocità di processazione e forwarding

Formato del datagram IPv6

- **ver**: versione, 4 bit
- **pri**: 8 bit, indica la priorità
- **flow label**: identifica un datagram nello stesso flusso, 20 bit
- **payload len**: 16 bit, indica la lunghezza del payload
- **next hdr**: 8 bit, indica il protocollo al quale deve essere consegnato questo pacchetto, se TCP o UDP
- **hop limit**: 8 bit, come il TTL dell’IPv4

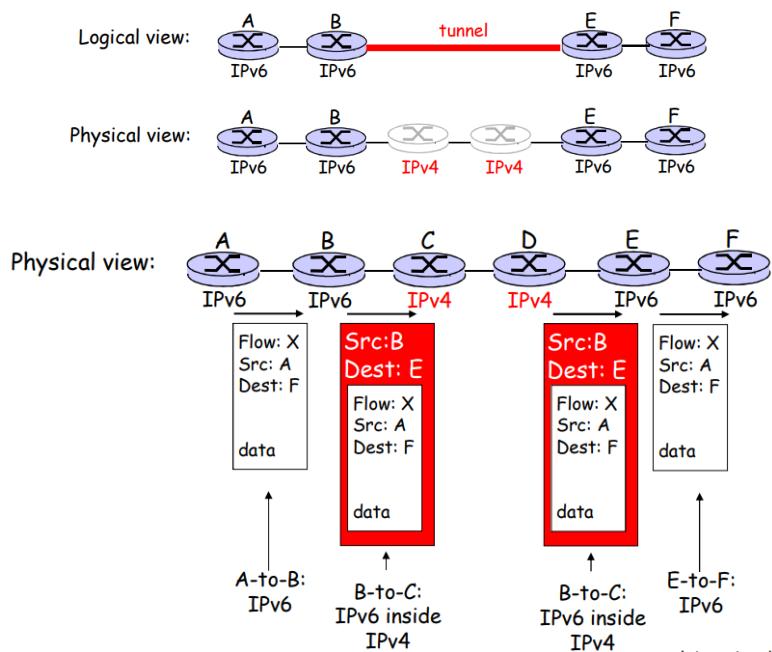


Transizione dall’IPv4 all’IPv6

La transizione da IPv4 a IPv6 al momento in cui si sta scrivendo questa frase non è ancora terminata: questo perché nonostante il protocollo sia adottato ufficialmente dal 2004 non tutti i router/dispositivi possono supportare questo protocollo e non tutti i dispositivi possono essere sostituiti così a cuor leggero.

Infatti, la maggior parte dei router attualmente supportano entrambi i protocolli: il problema si pone quando lungo il tragitto si incontra un router che non supporta IPv6.

Si utilizza allora la tecnica del tunneling.



Layer 4: Transport

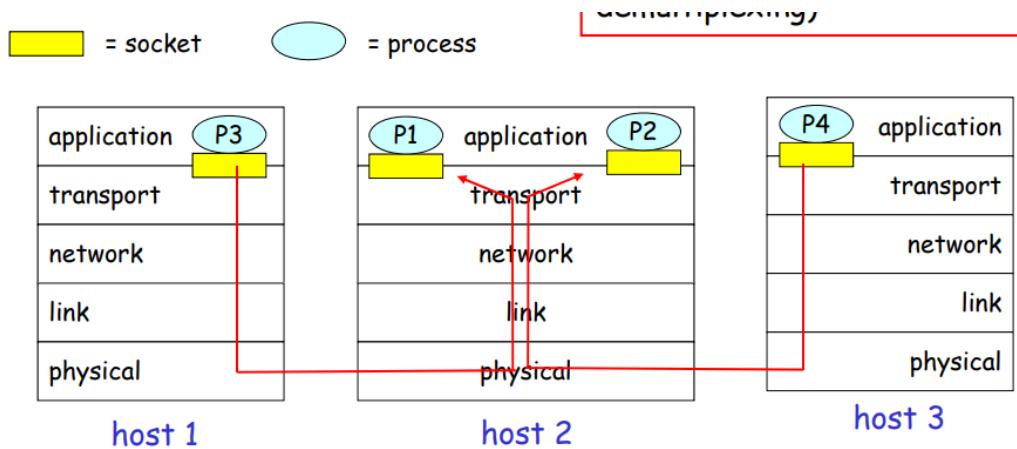
Fornisce una comunicazione logica tra i processi svolgendo il compito di consegnare i messaggi ad essi, oltre al compito di multiplexing e demultiplexing.

I protocolli di questo livello dividono i messaggi in **segmenti** passandoli poi al livello rete.

Multiplexing e Demultiplexing

Multiplexing: consiste nel ricevere dati da più socket e incapsularli.

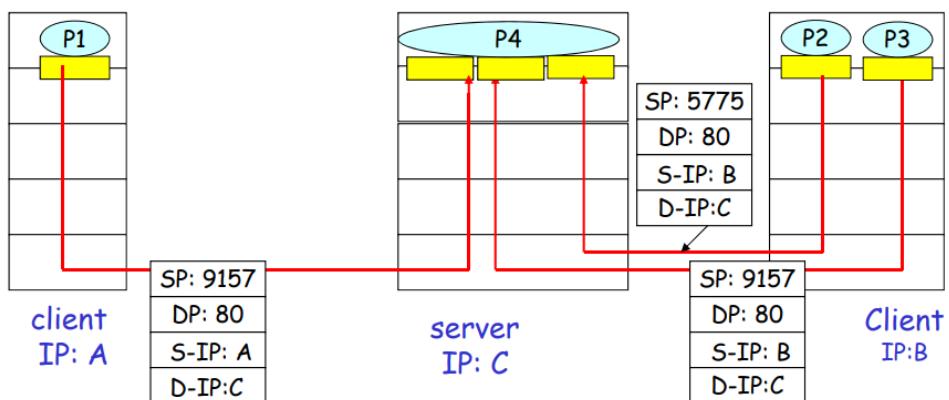
Demultiplexing: consiste nel consegnare i segmenti ricevuti al socket corretto.



In base al tipo di servizio, connectionless o connection oriented (rispettivamente UDP e TCP), il modo di fare demultiplexing cambia:

- **connectionless:** avviene la **ricerca della porta di destinazione e consegna al socket con quella porta e quell'indirizzo IP.**
Datagrams con IP e/o porta sorgente differenti vengono consegnati allo stesso socket, perché il **socket UDP è identificato solo da IP e porta destinatario.**
- **Connection oriented:** **il socket TCP è identificato oltre che dall'IP e dalla porta destinatario anche da quelli sorgenti.**

Per questo motivo i server host possono supportare anche molti TCP socket simultaneamente.



Come sappiamo, il processo server P4 può essere uno solo, che serve tante richieste a divisione di tempo o tanti, figli di un processo che sta in ascolto e che fa gestire le comunicazioni ai processi figli.

UDP

User Datagram Protocol, è un protocollo livello trasporto, semplice e veloce, in quanto l'unica operazione che compie è quella di **demultiplexing**.

Non si occupa di stabilire una connessione attraverso un 3-way-handshake come il TCP, ed ha un approccio di tipo best effort.

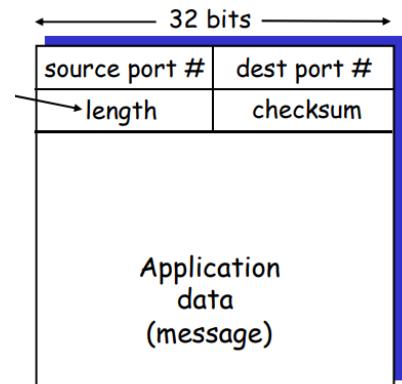
Anche se è inaffidabile è utilizzato da diverse applicazioni e protocolli come il DNS e il RIP o le applicazioni di tipo real time o da quelle applicazioni che preferiscono svolgere i controlli riguardanti l'affidabilità della comunicazione a livello applicazione (ad esempio quelli del TCP sono troppi o troppo pochi).

UDP fa comunque error detection utilizzando una checksum: se corrotto, butta via il pacchetto.

La UDP checksum (ma anche nel TCP) consiste nel dividere la stringa di d bit in stringhe di 16 bit, farne la somma bit a bit (eventualmente portando l'overflow all'inizio e sommandolo) e sommare con una nuova stringa di 16 bit.

Nel campo checksum del pacchetto però va il risultato negato, cosicché rifacendo tutti i calcoli e sommandoci il campo checksum il risultato venga una stringa di 1.

La lunghezza include gli 8 byte di intestazione.

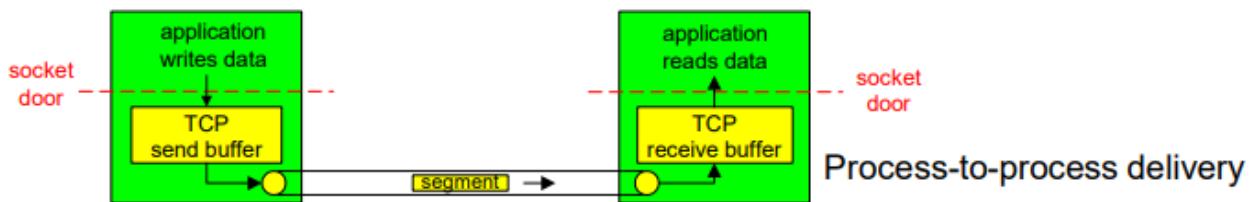


TCP

Offre un servizio di **trasporto tipo connection oriented**: prima di poter comunicare, trasmittitore e ricevitore devono stabilire la connessione: necessita di un three-way-handshake.

La connessione ricorda il circuito virtuale: ma i router non mantengono informazioni di stato (gli switch lo facevano per il VC), qui invece la connessione riguarda solo il livello di trasporto, e non ai dispositivi del core, loro non hanno visione della connessione dato che non implementano il livello trasporto.

Una sorta di Pipe virtuale, ci sono solo informazioni di stato alle due estremità: **come se ci fosse un canale logico point to point**: dal suo punto di vista, il processo mittente manda un messaggio e questo viene ricevuto dall'altra parte, come se:

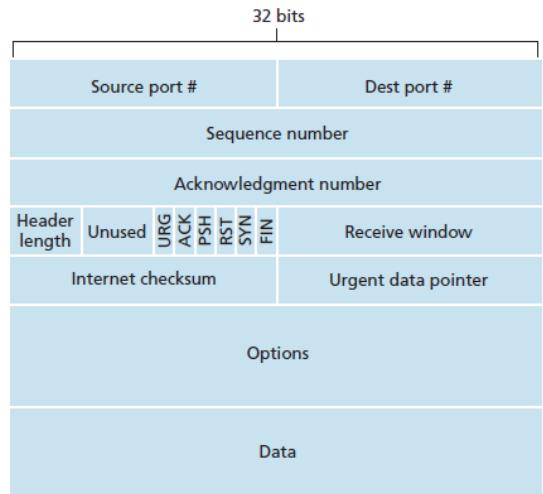


Il **flusso è full duplex ed è controllato**: il mittente non può sovrapporre il destinatario, ovvero se il rate di invio è più elevato del rate di lettura, se non ci fosse il controllo di flusso, il buffer destinatario saturerebbe e si perderebbero dei dati.

Infine, garantisce che i pacchetti arrivino tutti ed in ordine.

Struttura del segmento TCP

- Porta sorgente e destinataria, insieme agli IP contenuti nel datagram IP, servono per esercitare la funzionalità di full duplex
 - Il numero di sequenza e di ACK sono relativi ai byte: l'ack number non è sempre valido, dipende dal valore del bit di ACK: quando a uno, questo segmento è un ACK, se il bit =0, il campo acknowledgement number non ha significato.
- Quanto vale ackNumber?
- È relativo all'ultimo byte ricevuto correttamente e in ordine: anzi, nel campo non scrive il numero di sequenza ricevuto e in ordine, ma scrive il prossimo byte che si aspetta di ricevere.
- I flag RSF: si usano in fase di apertura e chiusura della connessione, S: SYN, F: FIN
 - **Receive window:** è qui che il ricevitore scrive quanto spazio è rimasto libero nel buffer del ricevitore.

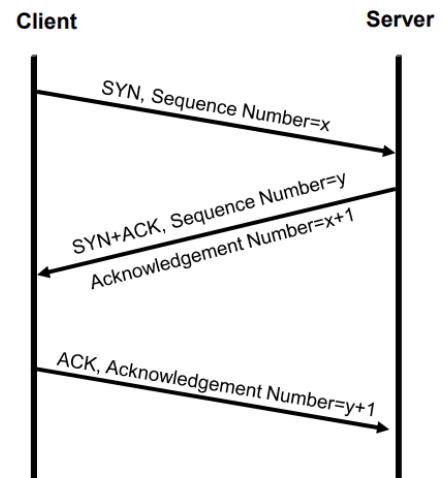


Stabilire una connessione

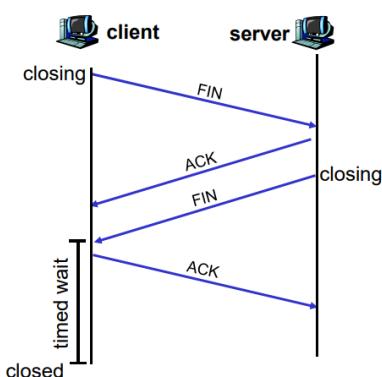
A cosa serve? Ad allocare le strutture dati, come ad esempio contatore num di sequenza, contatore num pacchetti ACK, buffer invio e ricezione ed altre strutture dati.

3 way handshake

- Inizia il client con la connect, che invia un segmento **SYN**, con un numero di sequenza: x (scelto a caso), in linea di principio anche 0 potrebbe andar bene, ma per precauzione: supponiamo di aprire e chiudere subito una connessione, e di aprirne un'altra subito dopo sarebbe possibile scambiare pacchetti della vecchia connessione rimasti nel buffer per pacchetti della nuova connessione).
- Il server risponde con un segmento di tipo **SYN+ACK**: sequence number = y e ack number = x+1;
- Il client risponde con un ACK con numero y+1.
Questo pacchetto di ACK può contenere dati, nel caso in cui sia http, tipicamente contiene la richiesta http.



Chiusura di una connessione



Finché anche il server non manda FIN, il client rimane solo nella fase di chiusura, ma non chiude subito dopo l'invio dell'ACK: in teoria potrebbe ma, se l'ACK si perde (o il primo FIN non arriva)?

Il server pensa che il client non abbia chiuso la connessione quindi se entro un certo timeout non riceve l'ACK, il client rimanda il FIN.

Quindi il client non chiude subito, potrebbe arrivare un altro fin, come se al server non fosse arrivato l'ACK di risposta al FIN del server, e quindi il client dovrebbe rispedire l'ACK. Questo perché? Perché se il server non deallocasse le strutture poi rischierebbe di finire le risorse, come nell'attacco *SYN flooding*.

Reliable Data Transfer

Il TCP utilizza una tecnica a Pipeline, vedi capitolo due, paragrafo 3.

Servizio di affidabilità: tutti i dati devono arrivare a destinazione e integri, si usa un protocollo di tipo reliable data transfer: unica differenza rispetto al collegamento punto-punto: qui non c'è un link fisico ma una connessione, una pipe virtuale. Come se ci fosse un link punto-punto virtuale: in realtà è la Pipe con le risorse allocate agli estremi e nessuna info di stato sui router.

Siccome è basato su ACK TIMEOUT e RITRASMISSIONE: quanto setto il timeout?

Non posso calcolarlo come facevo prima:

- ritardi di accodamento (unpredictable, e si verifica su ogni router) e non sappiamo nemmeno il percorso
- Ritardo di propagazione: non sappiamo il percorso e stessa cosa per il ritardo di trasmissione non conoscendo il rate dei link

Provo a stimarlo con la media esponenziale:

$$\begin{aligned} \text{SampleRTT} &:= RTT \\ \text{EstimatedRTT} &:= ERTT \end{aligned}$$

Alpha solitamente è
0.125.

$$\alpha < 1$$

$$ERTT_1 = RTT_0$$

$$ERTT_2 = \alpha \cdot RTT_1 + (1 - \alpha) \cdot RTT_0$$

$$ERTT_3 = \alpha \cdot RTT_2 + \alpha(1 - \alpha) \cdot RTT_1 + (1 - \alpha)^2 \cdot RTT_0$$

....

$$ERTT_{n+1} = \alpha \cdot RTT_n + \alpha(1 - \alpha) \cdot RTT_{n-1} + \alpha(1 - \alpha)^2 \cdot RTT_{n-2} + \dots + (1 - \alpha)^n \cdot RTT_0$$



$$ERTT_{n+1} = \alpha \cdot RTT_n + (1 - \alpha) \cdot [\alpha \cdot RTT_{n-1} + \alpha(1 - \alpha) \cdot RTT_{n-2} + \dots + (1 - \alpha)^{n-1} \cdot RTT_0]$$

$$ERTT_{n+1} = \alpha \cdot RTT_n + (1 - \alpha) \cdot ERTT_n$$

Scelta del timeout

Algoritmo di Karn-Partridge

In questo algoritmo non viene considerata la ritrasmissione dei pacchetti: se un pacchetto si perde e viene ritrasmesso e viene ricevuto un ACK non so dire a quale dei due pacchetti identici si riferisca: quindi per evitare errori, in quanto cambierebbe il valore dell'RTT, non si considerano i pacchetti ritrasmessi.

Va bene come approssimazione se la percentuale dei pacchetti ritrasmessi è limitata.

$$\text{Timeout} = 2 * \text{Estimated_RTT}$$

Algoritmo di Van Jacobson – Karel

Nel caso in cui i pacchetti ritrasmessi fossero tanti si utilizza questo metodo, basato sempre su media esponenziale mobile: si fa una stima della deviazione standard.

$$\text{DevRTT}(n+1) = (1-\beta) * \text{DevRTT}(n) + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

Con β circa 0.25

$$\text{Timeout} = \text{Estimated_RTT} + 4 * \text{DevRTT}.$$

Algoritmo TCP reliable data transfer

Questo particolare metodo non può essere definito né go-back-N ma nemmeno selective repeat: gli ACK risultato essere cumulativi, ma quando deve essere ritrasmesso un segmento, viene ritrasmesso solo il più vecchio.

Perché? **Perché i pacchetti potrebbero essere arrivati in disordine, ma l'ACK che viene mandato è relativo solo a quelli arrivati integri ed in ordine.**

Versione semplificata senza ACK cumulativo

Eventi per far scattare l'invio:

```

NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

loop (forever) {
    switch(event)

    event: data received from application above
        create TCP segment with sequence number NextSeqNum
        if (timer currently not running)
            start timer
        pass segment to IP
        NextSeqNum = NextSeqNum + length(data)

    event: timer timeout
        retransmit not-yet-acknowledged segment with
            smallest sequence number
        start timer

    event: ACK received, with ACK field value of y
        if (y > SendBase) {
            SendBase = y
            if (there are currently not-yet-acknowledged segments)
                start timer
        }
    } /* end of loop forever */

```

- ricezioni di un messaggio dal livello applicazione: bisogna prepara un pacchetto con un numero di sequenza opportuno.
Il numero di sequenza non è relativo ai segmenti ma ai byte. E si deve far partire il timer se questo non è già attivo.
Il timer è relativo al segmento più vecchio in attesa di ACK. Il timer lo “carico” con il valore del timeout tramite la stima.
- Timeout: si ritrasmette il segmento che ha causato il timeout. Un solo segmento, quello per cui è scattato il timeout! Si fa ripartire il timer: perché rimane sempre il segmento più vecchio in attesa di ACK.
- Ricezione di un ACK da parte del ricevitore: questo fa avanzare la finestra: alcuni segmenti che prima erano in attesa di ACK ora sono “acknowledgiati”. Si deve aggiornare il campo ACK. Si deve modificare al timer cosicché esso faccia riferimento al pacchetto più vecchio in attesa di ACK.

Il TCP usa un singolo timer di trasmissione.

Quando si ritrasmette? Ogni volta che scatta il timeout oppure quando il TCP mittente si accorge che un pacchetto si è perso (anche prima che scatti il timeout): ovvero fa una fast retransmission. Come si fa a sapere che un pacchetto si è perso?

Con gli ACK duplicati: dopo 3 ACK duplicati si fa una fast retransmission.

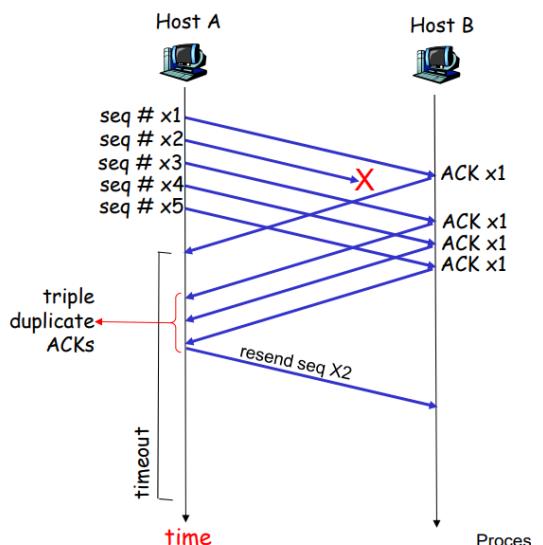
```

event: ACK received, with ACK field value of y
if (y > SendBase) {
    SendBase = y
    if (there are currently not-yet-acknowledged segments)
        start timer
}
else {
    increment count of dup ACKs received for y
    if (count of dup ACKs received for y = 3) {
        resend segment with sequence number y
    }
}

```

a duplicate ACK for
already ACKed segment

fast retransmit

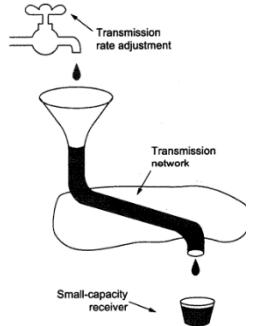


Generazione dell'ACK

Evento riscontrato dal ricevitore	Azione del ricevitore
Arriva il segmento con numero di sequenza atteso. Tutti i pacchetti precedenti sono già stati ACKnwoledgiati.	ACK ritardato. Se nei seguenti 500 ms arriva un nuovo segmento, allora viene spedito un ACK cumulativo.
Arriva il segmento in ordine mentre un altro ha un ACK pendente (entro i 500ms)	ACK cumulativo dei due pacchetti in ordine.
Arriva un segmento fuori ordine, genera un GAP	IMMEDIATAMENTE mandato un duplicate ACK <i>con seq number uguale a quello del pacchetto che ci si aspettava</i>
Arriva un segmento che parzialmente o interamente colma il GAP	Immediatamente spedito un ACK

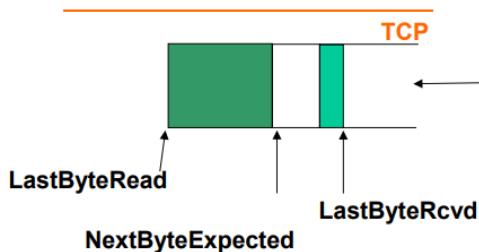
TCP flow control

L'obiettivo di tale controllo è evitare che il mittente invii una quantità eccessiva di dati che potrebbero, in alcune situazioni, mandare in overflow il buffer di ricezione del destinatario generando una perdita di pacchetti e la necessità di ritrasmissione con perdita in efficienza.



Come ricordiamo, nel segmento TCP è presente un campo contenente la dimensione della finestra di ricezione, ma come viene calcolata?

Supposto che sia presente un GAP, ma anche qualora non lo fosse la situazione non cambierebbe,

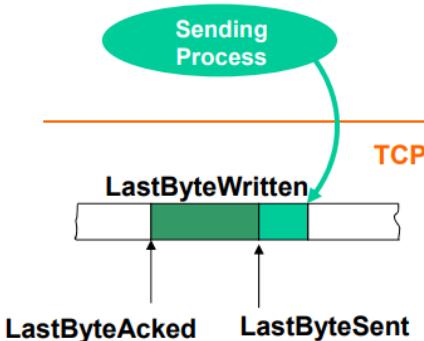


la grandezza dello **spazio occupato** misurato è: ultimoByteRicevuto - Ultimo Byte letto
e la dimensione della finestra risulta essere: dimensioneBuffer - spazioOccupato

Perché misurato e non effettivo? Perché è un'istantanea, nel momento in cui si calcolano questi valori potrebbero arrivare nuovi pacchetti o esserne letti di nuovi.

Il mittente, dunque, non può mandare tanti byte quant'è la dimensione della finestra, perché?

Perché nel caso migliore, sono arrivati tutti quelli ho mandato eccetto il primo unACKed, quindi quelli stanno già in memoria: quindi se ne mandassi tanti quanto la dimensione effettiva perché traboccherebbe di sicuro: questo perché la fotografia del buffer del ricevitore è stata fatta un po' di tempo fa (ritardo di trasmissione, di propagazione e semplicemente, mentre il ricevitore manda l'ACK io trasmettitore ho mandato altri byte).



Quindi la **finestra percepita dal trasmettitore** è:
CampoFinestraSegmento – (Ultimo ByteInvia – UltimoByteACKed)

Cosa succederebbe se lo spazio disponibile si riducesse a 0?
Il trasmettitore si ferma e periodicamente il trasmettitore manda un **segmento da 1 byte** per stimolare una reazione del destinatario, così da capire quanto spazio ha liberato (poiché se lo spazio disponibile è andato a 0, significa che il buffer del ricevitore si è saturato). Perché deve fare così? Perché se non mandasse un segmento periodicamente, il ricevitore non manderebbe più nessun feedback al trasmettitore e si andrebbe in stall, in quanto l'ACK viene mandato quando il pacchetto arriva e non quando viene letto dal buffer.

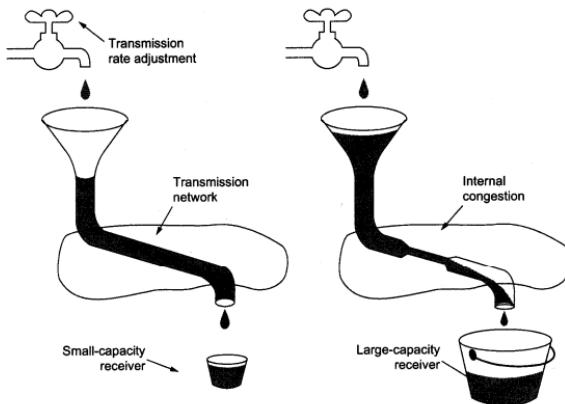
Congestion Control

Definizione informale di **congestione**: troppi mittenti mandano troppi dati troppo velocemente affinché la rete possa gestirli.

Se il tasso di ingresso è maggiore del tasso di uscita sul router, si crea una coda, una congestione.

Si può fare un'analogia tra la congestione nei router e il traffico ad una rotonda

Flow Control vs. Congestion Control



La congestione si verifica su un router nel percorso tra sorgente e destinatario: significa che quel router sta ricevendo più pacchetti di quelli che riesce a gestire perciò si verifica ritardo e nei casi peggiori perdita dei pacchetti.

Costi della congestione: se si perde un datagram, va ritrasmesso e va ritrasmesso attraverso tutti i router dove è passato! Significa che i router hanno fatto lavoro inutile. Bisogna evitare che si verifichino congestioni presso ai router.

Se un **pacchetto si perde non significa necessariamente che c'è congestione** (magari mezzo inaffidabile, è caduto un link lungo strada, ecc): il protocollo TCP però assume che ogni volta che un pacchetto si perde allora si è verificato congestione.

Come si può gestire?

Se tutti gli host diminuissero il rate di trasmissione, la situazione migliorerebbe (o comunque, sicuramente non peggiorerebbe).

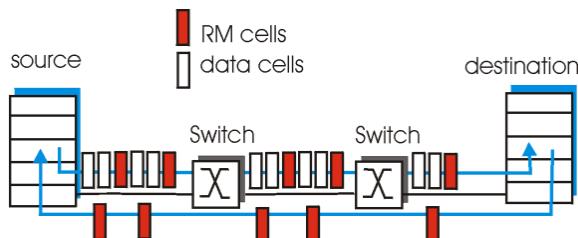
Ma come si fa a sapere che si è verificata una congestione?

Ci sono due approcci: quello **network-assisted** (i router forniscono un feedback agli host e in alcuni casi possono anche suggerire a quale rate mandare i pacchetti, questo approccio però non viene usato) oppure con un approccio **end-to-end** (non c'è feedback esplicito, ma l'host assume ci sia congestione quando si verifica una perdita, approccio usato dal TCP).

Congestion control nell'ATM

Prendiamo in considerazione l'ATM ABR (Available Bandwidth Rate): qui i dispositivi provano a prendere il massimo della banda disponibile e dispositivi del core mandano feedback, quindi è un approccio network-assisted.

Con questo approccio tra le celle dei dati vengono mandate le **resource-management cells (RM)** le quali sono necessarie per regolare il rate di trasmissione.



ABR fornisce tre meccanismi per segnalare le congestioni:

- **EFCI bit.** Ogni cella dei data contiene un bit EFCI (*Explicit Forward Congestion Indication*) bit. Uno switch in una rete congestionata setta questo bit a 1 per segnalare la congestione all'host destinatario. Il destinatario, controlla l'EFCI bit in ogni data cell e quando un RM cell arriva a destinazione, se il data cell più recente aveva EFCI a 1, allora setta CI (Congestione indication) a 1 nell'RM cell e la rimanda indietro al mittente.
- **RM cell.** Queste celle posseggono **NI (no increase)** e **CI (congestion indication)** bit. Questi bit possono essere modificati opportunamente dagli switch del core e una volta arrivati a destinazione, il destinatario rimanda queste celle al mittente il quale può così regolarsi.
- **ER setting.** Ogni cella RM può contenere anche un'indicazione esplicita sul rate.

Congestion control in Internet: TCP congestion control

L'obiettivo della trasmissione di dati è cercare di trasmettere il più velocemente possibile cercando di evitare che si verifichino congestioni.

Ma se tutti cercano di trasmettere al rate possibile più alto è certo che si verifichino congestione: quindi è **necessario** un controllo, più precisamente **un auto regolazione**, non essendoci nessuno che può dare indicazioni (i dispositivi del core non mandano indicazioni, approccio end-to-end).

- Come si fa a variare il rate?

Se si prende come riferimento una finestra (un intervallo di tempo) uguale al RTT: variando il numero pacchetti spediti durante questa finestra, varia il rate.

La quantità di dati da spedire è detta **congestion window**: la quale è definita come il numero max di byte che il trasmettitore può mandare prima di fermarsi per aspettare il round trip time.

La finestra non è fissa, ma varia in base alla congestione.

Il trasmettitore ha un limite sul **numero di pacchetti** da inviare: **il minimo tra la dimensione della congestion window e il numero di byte rimanenti da inviare**.

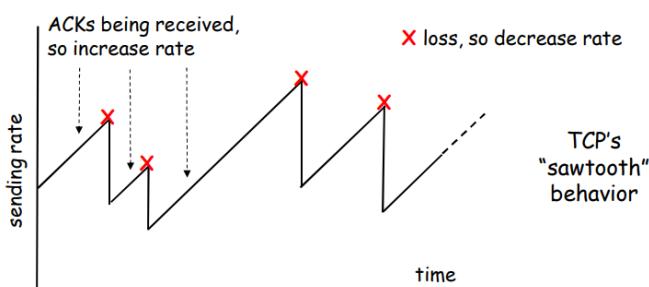
- Come fa il trasmettitore a percepire che si è verificata congestione?

Tutte le volte che un pacchetto si perde (ovvero se si ricevono 3 ACK duplicati di fila o se scade il timeout) il TCP suppone che si sia verificata una congestione, anche se non è necessariamente vero (ad esempio, nei collegamenti wireless si perdono tanti pacchetti a causa dell'inaffidabilità del mezzo).

- Come deve variare il rate?

Il trasmettitore prova ad ottenere il massimo della banda: ogni volta che il trasmettitore riceve un ACK, prova ad aumentare linearmente il rate.

E si riparte da capo.

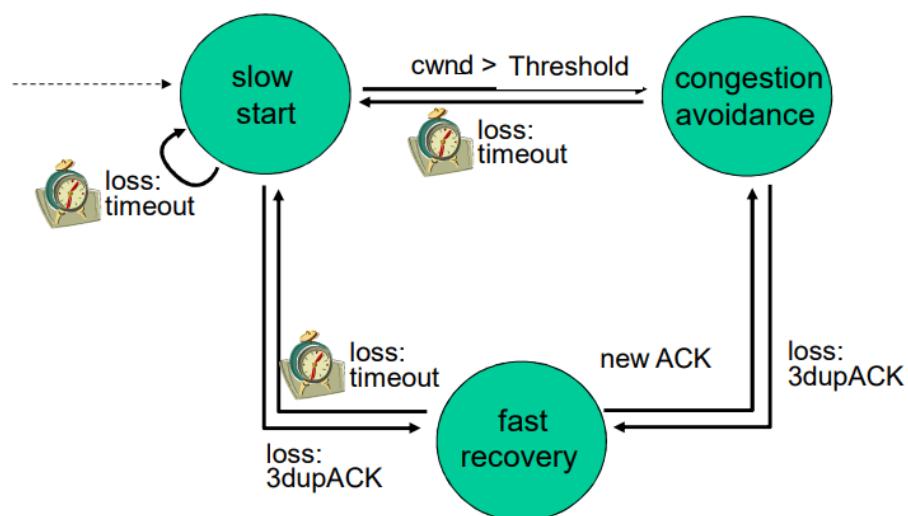


Dalla partenza in poi sono presenti 3 fasi:

- **Slow Start**: L'inizio è lento: si invia un solo segmento di dimensione massima: [**Maximum Segment Size: Payload Data Link – Header IP -TCP header (MSS)**]; **ricevuto l'ACK, si raddoppia la finestra di congestione.**
- L'incremento esponenziale avviene fino ad una certa soglia (**Threshold**), dopo questa soglia si esce dalla fase di slow start e si entra nella fase di **congestion avoidance**: in questa fase avviene un incremento della finestra lineare.

Ma quanto vale Threshold? Inizialmente vale **64Kbyte**.

- Se si verifica **un triplice ACK** la soglia viene **dimezzata rispetto al valore della congestion window prima che si verificasse la congestione.**
La **congestion window** invece cambia come $cwd=[(cwd/2) +3]$ per tenere conto dei 3 ack duplicati
Quando **si riparte si riparte dalla fase di congestion avoidance. (Fast recovery)**
- Se si verifica un Timeout la **finestra di congestione** si pone **uguale a 1**.
La soglia viene **dimezzata rispetto al valore della congestion window prima che si verificasse la congestione.**
E si riparte dalla fase di slow start.

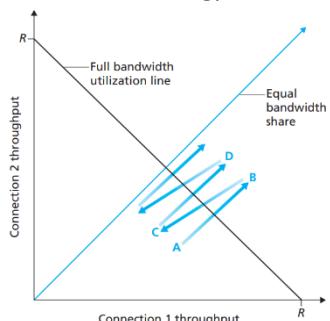


Fast Retransmit e Fast Recovery: due algoritmi (implementati sempre in coppia) specificatamente progettati per gestire le perdite singole

- il segmento considerato perso viene subito ritrasmesso (fast retransmit)
- la CWND non viene chiusa eccessivamente (fast recovery)

Il TCP che utilizza fast-recovery è noto come TCP RENO, mentre il TCP senza

Il TCP è un protocollo fair, supposto di avere due link con stesse caratteristiche, il comportamento delle due finestre e del throughput varia circa come segue:



Tra UDP e TCP invece, UDP è avvantaggiato in quanto non fa controlli di flusso o di congestione e perciò manda tutto quello che può mandare e alla velocità che vuole.

Considerando che UDP viene usato spesso nelle comunicazioni real time è anche corretto che siano gli altri a diminuire il rate.

Network Security

I rischi della rete

In rete sono presenti diverse minacce potenziali, poiché esistono utenti malintenzionati che possono attaccare le reti, attaccando gli host presenti per creare danni agli utenti.

Ma è sempre stato così? I protocolli implementano un qualche tipo di sicurezza?

Non è sempre stato così, poiché inizialmente Internet era utilizzato principalmente esclusivamente da accademici e si basava sul presupposto che tutti gli utenti collegati fossero brave persone, motivo per il quale i protocolli non implementavano (e tutt'ora) molta sicurezza, in quanto la visione era appunto quella di un gruppo di utenti che si fidano uno dell'altro collegati ad una rete trasparente.

Quali sono le minacce?

I rischi che corriamo quando ci colleghiamo in rete sono diversi:

- **Packet sniffing**

Può avvenire nei link di tipo broadcast, *a volta viene fatto a posta per analizzare le prestazioni della rete*, a volta per scopi non buoni: ad esempio con le interfacce di rete promiscue.

Ad esempio, uno sniffer può copiare i pacchetti e venire in possesso delle informazioni altrui.

- **IP spoofing**

Ovvero consiste nel mandare pacchetti con un IP falso, ovvero modifica il datagram

- **DoS o DDoS**

Si fa in modo che il server collassi, risorse o banda, cosicché il server sia impegnato a servire richieste fasulle invece che servire richieste vere.

- **Record-and-playback:**

L'hacker fa sniffing, registra il pacchetto e lo riutilizza dopo qualche tempo, il problema sta quando ci sono password o ad esempio se ci sono richieste di denaro

Gli attacchi non coinvolgono necessariamente sempre un intermediario su un mezzo comune. Molte volte è lo stesso utente che inconsapevolmente introduce nel proprio dispositivo un **malware**.

I malware possono entrare **da virus, worm e trojan horse** (con la collaborazione dell'utente che di sua spontanea volontà scarica un'applicazione o visita un sito inconsapevole che quella cosa sia infetta o che scarichi un virus): esempi di malware: **spyware**, ad esempio possono registrare le attività dell'utente come i tasti che digita e i siti che frequenta, oppure **arruolare l'host per creare una botnet** così da arruolarlo per un DDoS.

I malware spesso sono autoreplicanti.

Prima forma di prevenzione è la consapevolezza.

Sicurezza

Il termine sicurezza assume diversi significati all'interno della rete:

- **Confidenzialità:** segretezza della comunicazione. I dati che vengono scambiati tra due o più persone devono rimanere all'interno di quello scambio di dati.

Nasce l'esigenza della **crittografia**.

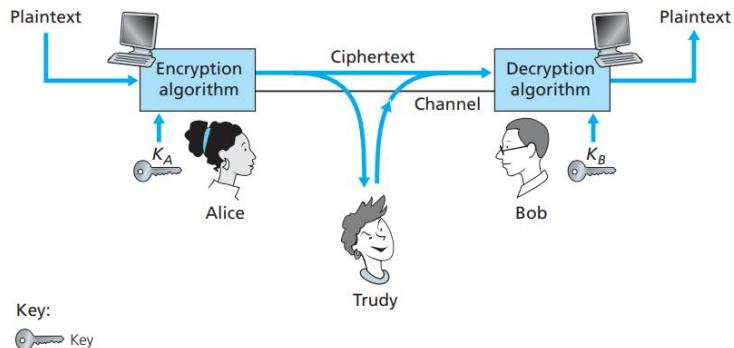
- **Integrità:** il destinatario deve ricevere interamente e non alterato il messaggio originario.

- **Autenticazione End – To – End:** i due che hanno intenzione di comunicare devono essere in grado di riconoscere l'altro come tale. C'è bisogno che dunque i due siano in grado di assicurarsi che il messaggio arrivi **esattamente dall'altra persona**, e non da un intruso.

- **Accesso e Disponibilità dei Servizi.**

Confidenzialità e crittografia

Per garantire tutte queste cose (eccetto l'ultima) si utilizza la crittografia:



Introduciamo un po' di terminologia:

- Chiameremo **messaggio in chiaro** (*plaintext*) il messaggio originale non cifrato che A vuol mandare a B
- Chiameremo **crittogramma, o messaggio cifrato**, (*ciphertext*) il messaggio che viaggia sul canale di comunicazione e che risulta illeggibile ad ogni intruso.
- Ovviamente le due parti devono riuscire a mettersi d'accordo su un certo **algoritmo di cifratura e decifratura**.

Crittografia a chiave simmetrica

Cifrario di Cesare

Il cifrario più antico e semplice: le lettere dell'alfabeto vengono traslate di un certo valore k , ad esempio:

plaintext: abcdefghijklmnopqrstuvwxyz
↓ ↓
ciphertext: ghijklmnopqrstuvwxyzabcdef

Semplice, ma facilmente decriptabile dall'esterno anche senza conoscere la chiave.

Cifrario monoalfabetico

Molto simile al cifrario di Cesare, ma l'alfabeto non viene traslato: esiste una corrispondenza tra una lettera ed un'altra.

plaintext: abcdefghijklmnopqrstuvwxyz
↓ ↓
ciphertext: mnbvcxzasdfghjklpoiuytrewq

È vero che sono presenti $26!$ Combinazioni, ma non viene modificata la struttura del messaggio, quindi anche questo è decriptabile dall'esterno anche senza conoscere la chiave.

Crittografia polialfabetica

Si fanno n passi di cifratura con n cifrari monoalfabetici.

Esistono in questo tipo di crittografia due tipi principali di cifrari:

- 1) **Cifrari di tipo "Stream":** che si occupano di cifrare bit per bit le informazioni
- 2) **Cifrari a Blocchi:** che dividono il messaggio in blocchi di k bit e cifrano blocco per blocco

Concentriamoci sui cifrari a blocchi. Ce ne sono diversi, e la prima cosa da fare è capire "come valuto se un cifrario è robusto?".

La soluzione sta nel **provare a farlo**.

Fu il caso di **DES (Data Encryption Standard)**, basato su cifratura a blocchi di 64 bit con una chiave stringa da 56 bit.

Ci fu una competizione, e con l'algoritmo a forza bruta si riuscì a forzare il cifrario in meno di un giorno.

Nacque così **AES (Advanced Encryption Standard)**, che era ora basato su blocchi di 128 bit cifrati usando chiavi di 128, 192 o 256 bit. Per forzare AES ci vorrebbero 149 trilioni di anni.

Scambio della chiave

Nella crittografia a chiave simmetrica, la chiave di Bob e di Alice sono identiche e segrete.

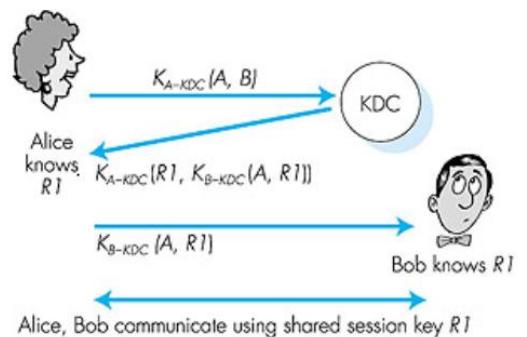
Come fanno Alice e Bob a condividere la chiave?

- scambio diretto, di persona
- distribuzione della chiave (un ente fidato fa da intermediario e aiuta a distribuire la chiave)
- usare la crittografia a chiave pubblica.

Key Distribution Center

Vediamo come funziona:

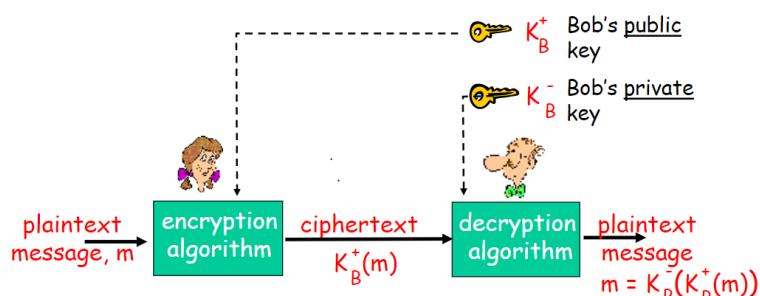
- 1) Alice e Bob vanno di persona a un KDC fisico e si registrano. Il KDC consegna a loro una chiave che permette la comunicazione in chiave simmetrica col KDC stesso. Chiameremo la chiave Alice – To – KDC K_{A-KDC} , mentre chiameremo la chiave Bob – To – KDC K_{B-KDC} .
- 2) Utilizzando una comunicazione a chiave simmetrica, Alice comunica a KDC che vuole parlare con Bob
- 3) Il KDC, utilizzando la stessa comunicazione (e quindi utilizzando la chiave K_{A-KDC}) manda ad Alice una chiave di sessione **R1**. Questa sarà la chiave che Alice dovrà utilizzare per parlare con Bob
- 4) Ma come fa Bob a ricevere questo R1? Nel messaggio precedente il KDC invia un messaggio cifrato **utilizzando la chiave di Bob** che quindi Alice non riesce a comprendere. Lei prende questo messaggio e lo spedisce a Bob. Bob, che viceversa questo messaggio può leggerlo, scopre che è un messaggio del KDC che dice "per parlare con Alice usa R1"
- 5) Ora Alice e Bob possono comunicare usando la chiave simmetrica di sessione R1



Crittografia chiave pubblica

L'idea della chiave pubblica risolve in generale il problema dello scambio della chiave, perché permette una comunicazione sicura senza alcuno scambio. In particolare **ogni host ha una coppia di chiavi**:

- 1) Una **chiave pubblica**, che è nota a tutti, anche agli intrusi.
- 2) Una **chiave privata**, che è nota solo all'host



Quando Alice vuole mandare un messaggio m a Bob, applica come chiave di cifratura di un algoritmo eventualmente pubblico la chiave pubblica di Bob.

Dopodiché Bob decifra il messaggio utilizzando invece la sua chiave privata.

La coppia di chiavi è pensata per essere applicata in qualsiasi ordine per criptare / decriptare un messaggio.

$$K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$$

use public key first, followed by private key

use private key first, followed by public key

In generale il meccanismo della chiave pubblica potrebbe essere utilizzato per comunicarsi qualsiasi cosa, ma ha dei problemi:

- **Chiunque** può mandare messaggi criptati a Bob pretendendo di essere Alice, perché sia Alice che gli intrusi utilizzerebbero la stessa chiave pubblica di Bob
- Il sistema è soggetto ad attacchi di tipo chosen – plain – text, ovvero l'attaccante può decidere una frase in chiaro e criptarla nello stesso modo di come Alice cripterebbe la stessa frase. Ciò significa che se l'intruso manda un messaggio a Bob e scopre che è la codifica è la stessa del messaggio che "vede passare", l'intruso scopre automaticamente cosa c'è scritto nel messaggio.
- È **computazionalmente più complicato comunicare in questo modo**, per questo motivo di solito si utilizza il meccanismo della chiave pubblica per scambiarsi la chiave simmetrica (detta **chiave di sessione**) e comunicare poi secondo l'altro meccanismo

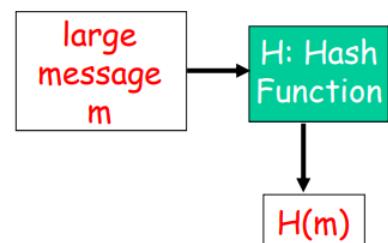
Integrità del messaggio

Message Digest:

Una funzione di hash prende in input una stringa di lunghezza variabile e restituisce una stringa di lunghezza fissa.

Le proprietà desiderabili da questa funzione di hash sono:

- Irreversibilità
- Resistenza alle collisioni
- Restituire un output che sembri randomico



Esempi di funzioni hash sono MD5 e SHA-1.

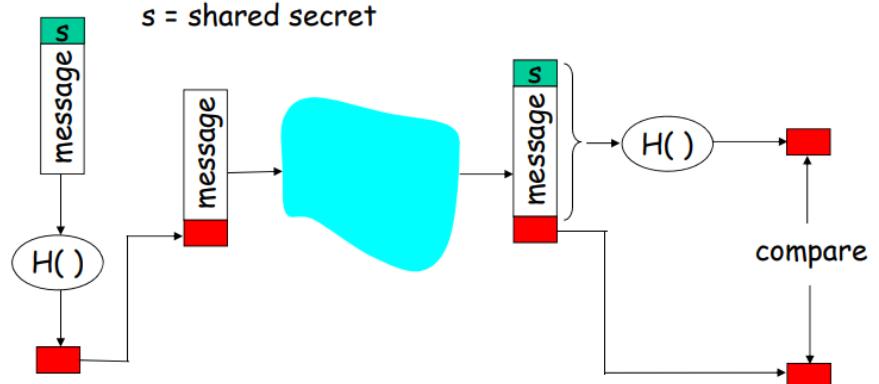
Message Authentication Code

Utilizzato per garantire l'integrità del messaggio.

Il MAC Message Authentication Code è un codice che si mette insieme al messaggio: per generare il MAC si utilizza una funzione HASH (MD5 o SHA-1).

A e B devono condividere un segreto, non è una chiave: qui non si fa cifratura e decifratura, anche se è una stringa di bit, Alice prende il segreto lo concatena al messaggio e lo passa alla funzione Hash, viene generato un digest: questo è il MAC, preso il messaggio e ci appende il MAC e lo manda a BOB, separa e ripete la stessa funzione di hash e confronta i MAC

Oltre all'integrità del messaggio, questo garantisce anche l'autenticazione del mittente.



Dove viene usato MAC? Nell'algoritmo OSPF (routing). Perché serve? Perché si potrebbero mandare informazioni fasulle (tipo un certo link non funziona o funziona ma non è vero) cosicché il traffico sia spedito su un router attaccato alla rete solo per attaccare la rete e conoscerne il traffico.

OSPF, quindi, prevede 3 tipi di autenticazione:

- Nessuna
- Con una password condivisa che viene inserita tutte le volte nel campo del pacchetto e viene posta in chiaro
- Autenticazione più elevata con hash crittografato, con MD5, quindi viene usato un Message authentication code

Questo per essere sicuri che il router che manda messaggi sia fidato.

Il MAC non garantisce da attacchi di tipo record and replay.

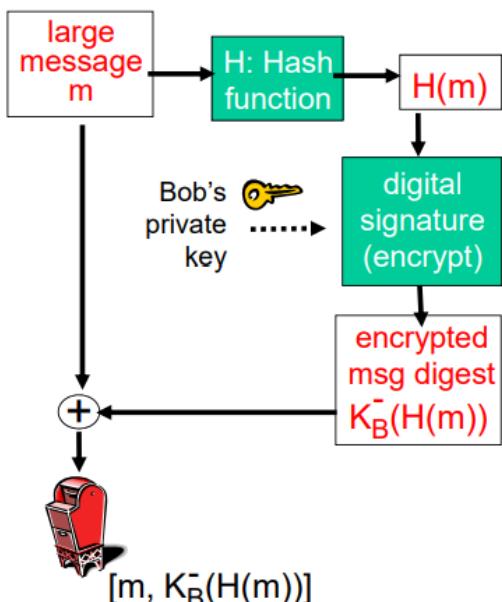
La firma digitale per assicurare l'autenticazione

Deve essere equivalente a quella autografa, deve essere perciò:

- verificabile (si deve poter verificare che la firma l'abbia messa bob e non caio)
- non falsificabile
- non può essere ripudiata (nel senso una volta che abbiamo firmato un documento non può più essere ripudiato)
- integrità del messaggio (il mittente può provare che lui ha firmato il messaggio m e non m').

Si utilizza la crittografia a chiave pubblica:

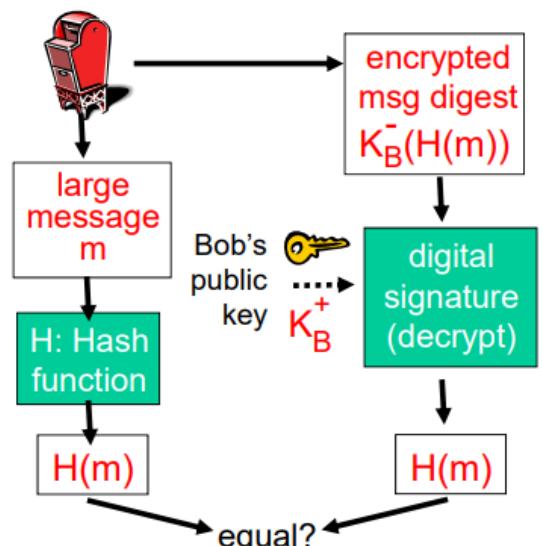
Bob sends digitally signed message:



1. Bob passa il *messaggio ad una funzione di hash*
2. Il **digest viene criptato con la chiave privata di Bob**
3. Viene spedito il messaggio e il digest criptato

Alice verifies signature and integrity of digitally signed message:

$[m, K_B^-(H(m))]$



4. Alice riceve il messaggio e il digest criptato: **decripta il digest con la chiave pubblica di Bob** e recupera il digest
5. **Applica la stessa funzione di Hash al messaggio ricevuto**: ora può confrontare il digest calcolato con quello ricevuto

Questo è possibile grazie alle proprietà dell'RSA.

Se i digest sono uguali possiamo concludere che il messaggio è arrivato integro ed è stato firmato da Bob.

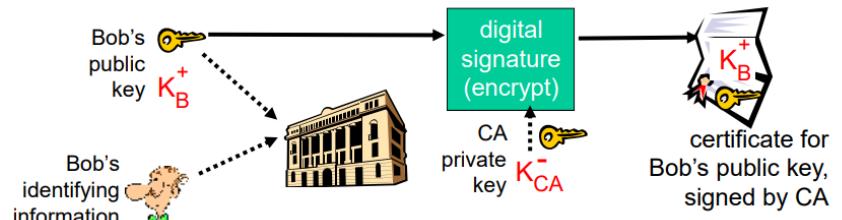
Come fa Alice a conoscere la chiave pubblica di Bob?

Non si può chiedere direttamente in quanto nel caso in cui rispondesse Trudy invece che Bob, potrebbe rispondere con la sua chiave pubblica e far sì che la comunicazione avvenga tra lei e Alice spacciandosi per Bob.

Necessitiamo del **Certification authority**, ovvero di un ente.

Bob comunica la propria chiave pubblica ad un ente e si fa identificare.

L'ente crea un **certificato** che lega l'ente con la chiave pubblica di Bob e critta il tutto con la sua chiave privata.



Il certificato, dunque, è un messaggio

firmato con la firma digitale della Certification Authority, che può essere mandato anche da Bob verso Alice per garantire l'autenticazione di Bob.

Quando Alice vuole ottenere la chiave pubblica di Bob deve ottenere il certificato (anche da Bob) e poi deve decriptarlo con la chiave pubblica dell'ente. Ottiene così la chiave pubblica di Bob.

Si necessita però della chiave pubblica della certification authority: chi mi assicura che sia effettivamente CA e non Trudy?

La CA è un'entità fidata, la chiave è nota a tutti e non è falsificabile.

Cosa accade se si chiede direttamente la chiave pubblica all'altra persona?

Si rischia attacco di man in the middle, a meno che non venga fornito sotto forma di certificato autenticato.

Proteggersi da attacchi record and playback

Sono possibili diverse soluzioni:

- Inserire un **timestamp nel messaggio**, se ci arriva un messaggio dopo tot, è troppo vecchio: scartato.
Quanto piccolo deve essere il timestamp? Se sniffato e rimandato entro lo scadere del timestamp non abbiamo risolto il problema.
- **OTP, One Time Password**: le due parti conoscono un *algoritmo pseudo random per la generazione di una password*, mittente e destinatario conoscono quei numeri, ma al di fuori di loro due sembrano caratteri casuali.
Fondamentale che l'algoritmo sia conosciuto solo dai due attori in gioco.
- Mittente manda un **nonce** alla ricevente, che genera digest sulla base della chiave, del messaggio e del nonce, il quale garantisce la freschezza del messaggio.
In alcuni casi si utilizzano due canali diversi per ricevere il **nonce**.

Dove implementare i protocolli di sicurezza

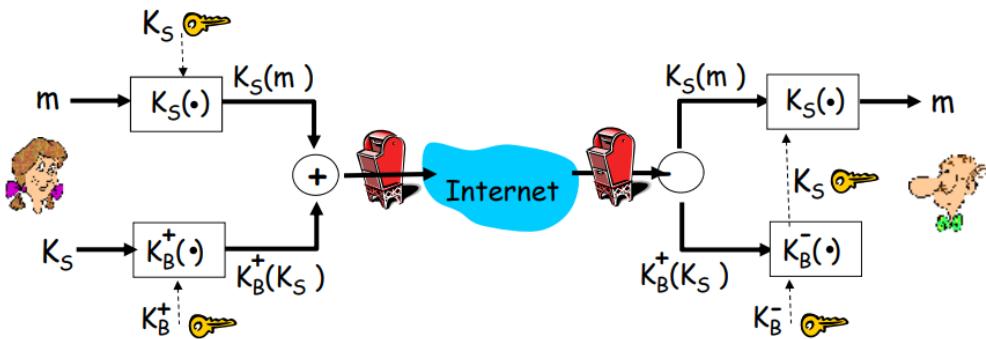
Intrinsecamente sarebbe possibile implementare i protocolli di sicurezza a qualsiasi livello della pila, ma non c'è un modo unico. Perché non si implementa, ad esempio, la sicurezza solo a livello rete?

Primo perché non si potrebbe fornire sicurezza a livello utente, ad esempio, un sito di commercio non può implementare la sicurezza basandosi solo sull'IP.

In secondo luogo è più semplice (e più “veloce”) implementare un servizio di sicurezza a livello applicazione, è il caso del PGP (Pretty-good-privacy)

Secure Email e PGP

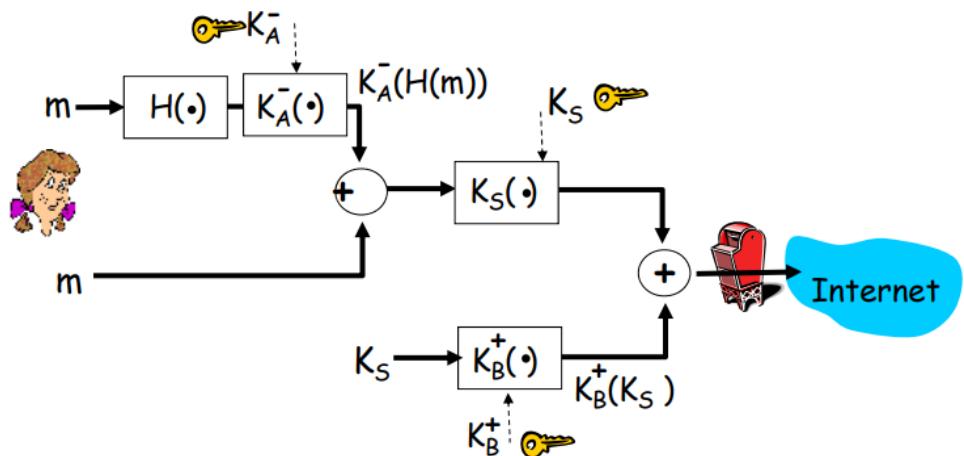
Per garantire la confidenzialità:



Alice: genera una chiave simmetrica per velocizzare la cifratura del messaggio, lo cifra e poi cifra la chiave simmetrica con la chiave pubblica di Bob così da potergliela comunicare.

Bob: decripta la chiave simmetrica con la sua chiave privata. Ottenuta la chiave simmetrica, può decriptare il messaggio.

Per garantire l'autenticità basta la firma digitale, per ottenere autenticazione, segretezza e integrità del messaggio (PGP) si concatenano i due passaggi:



SSL (Secure Sockets Layer)

È un protocollo crittografico del livello presentazione, ovvero a livello **middleware**.

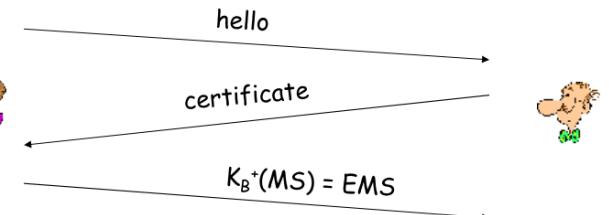
Non sarebbe efficiente se fosse implementato come PGP, perché solitamente nella posta elettronica una sessione riguarda un messaggio, mentre a livelli sotto il livello applicazione transitano stream di byte.

Obiettivi del SSL:

1. Autenticazione del Server
2. Negoziazione: accordo sugli algoritmi di cifratura
3. Stabilire una chiave
4. (Opzionale) autenticazione client

In prima approssimazione possiamo considerare il protocollo con queste 4 fasi:

- Handshake
- Derivazione delle chiavi
- Trasferimento dati
- Chiusura della connessione



Perché servono delle chiavi? E quante chiavi servono?

■ MS = master secret

■ EMS = encrypted master secret

Sono ottenute basate sul **Master secret** e sull'utilizzo della key derivation function:

- K_c chiave di cifratura (e decifratura) per i dati spediti da client a server (session key)
- K_s chiave di cifratura (e decifratura) per i dati spediti da server a client (session key)
- M_c segreto da utilizzare nella generazione del MAC per i dati spediti da client a server
- M_s segreto da utilizzare nella generazione del MAC per i dati spediti da server a client

Ma dove viene messo il MAC generato? E a cosa serve?

Si considera il MAC a **livello di record ponendolo alla fine di ogni record**.

Garantisce integrità ma non attacchi di tipo record e playback.

Un attaccante potrebbe ad esempio cambiare l'ordine di sequenza ll'interno dello straemnig: come possiamo evitarlo?

Si considera un numero di sequenza, un number ID, il quale viene considerato insieme ai dati per generare il MAC.

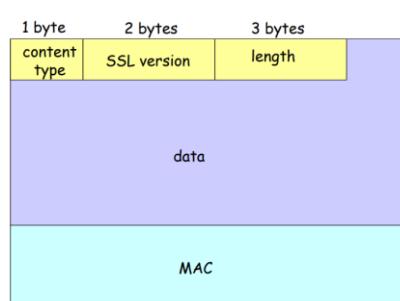
Non ci dà sicurezza sugli attacchi di tipo connection replay. Per questi vengono introdotti dei **nonce**.

Altro attacco che si può verificare è **truncation**: ad esempio viene mandato un TCP di chiusura, e fa sì che la connessione viene chiusa temporaneamente e la transazione rimane in uno stato inconsistente.

Quindi nel campo di intestazione si mette un tipo e che dice se è un record normale o di terminazione.

Solo perché la chiusura prima deve essere segnalata.

Il mac rimane comunque in fondo, ma ci viene concatenato nella sua generazione anche il tipo di messaggio. **MAC=MAC(M_x, sequence // type // data).**



Il protocollo SSL non è così semplice, infatti:

1. Il client manda una **suite crittografica** (client e server devono concordare su: algoritmo a chiave pubblica, algoritmo a chiave simmetrica, algoritmo MAC). **Il client manda la lista egli algoritmi che lui supporta.**
Poiché questo pacchetto viaggia in chiaro (non si è ancora deciso come crittografare) potrebbe essere soggetto ad attacchi **man-in the-middle** tali che possano modificare il contenuto del messaggio eliminando gli algoritmi più robusti e lasciando solo quelli più facilmente decriptabili. Per evitare questo si eseguono i passi 5-6.
Inoltre vengono aggiunti i nonce per evitare connection reply.
2. Il **server sceglie dalla lista I vari algoritmi** e manda indietro il messaggio con le sue scelte insieme al suo certificato ed ad un nonce.
3. Il client verifica il certificato, estrae la chiave pubblica del server, **genera un Pre-Master Secret (PMS) e lo cifra con la chiave pubblica del server.** Manda il PMS crittografato al server.
4. Usando la stessa chiave di cifratura, client e server generano indipendentemente le 4 chiavi utili per la comunicazione.
5. Il client manda un MAC di tutti i messaggi di handshake.
6. Il server manda un MAC di tutti i messaggi di handshake.

Gli ultimi due passaggi servono a proteggere da attacchi di tipo man in the middle, infatti client e server possono così accorgersi di eventuali intrusi.

Sicurezza a livello rete: IPsec

Le istituzioni e le organizzazioni spesso vorrebbero avere reti private, ma creare una nuova infrastruttura di rete costa. E fare delle operazioni “private” collegandosi all’Internet è pericolo in quanto si è soggetti ad attacchi e soprattutto non si è sicuri al 100% che i dati arrivino integri o che non siano sniffati da nessuno. Con **una VPN (Virtual Private network)** però si risolve, il traffico viaggia su Internet ma è criptato prima di mandare dati sulla rete.

Si utilizza il modello **IPsec** il quale offre i servizi di:

- Confidenzialità
- Prevenzione da attacchi record and playback
- Autenticazione dell’origine
- Integrità

Esistono due protocolli IPsec:

- **Autenticathion Header (AH)** che garantisce tutti i servizi tranne la confidenzialità
- **Encapsulation Security Protocol (ESP)** che garantisce tutti i servizi

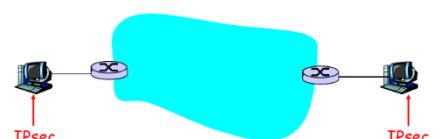
IPsec è connection oriented: la connessione si chiama **security assocation.**

Esistono tre modi differenti per garantire la sicurezza:

IPsec Transport Mode

Transport Mode

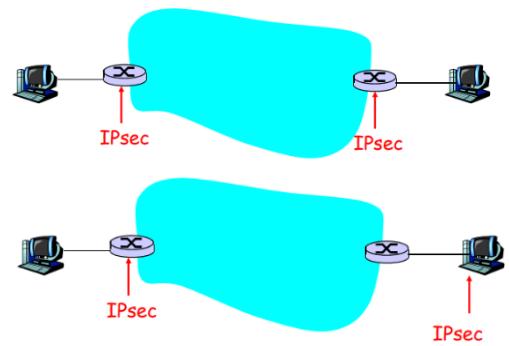
I datagram viaggiano da host a host nel formato IPsec, viene stabilita la connessione tra i due endpoint.



Tunneling Mode

Esistono due tipi di tunneling mode, nella rete "fidata" (ad esempio dell'headquarter) i datagram viaggiano sempre con protocollo IPv4 ma quando escono/entrano viaggiano in formato IPsec.

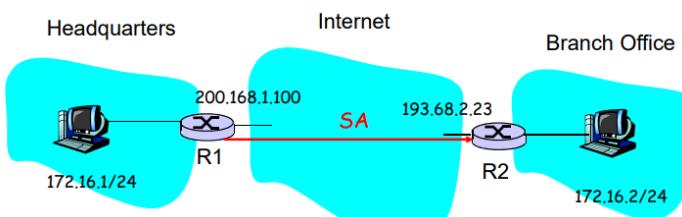
La differenza sta se la connessione è stabilità tra i due border router o tra dispositivo e router (si utilizza il secondo caso quando ad esempio un dispositivo che vuole connettersi ad un altro è collegato ad una rete pubblica).



ESP

Le Security Associations sono connessioni virtuali **non duplex**.

Quindi, per avere una normale comunicazione, sono necessarie due SA.



I due end-point (siano router o host) devono mantenere delle informazioni di stato riguardanti la SA.

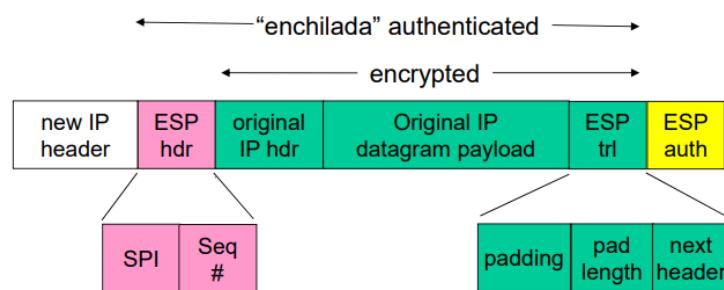
Le informazioni da memorizzare sono:

- 32-bit identifier for SA: **Security Parameter Index (SPI)**
- L'interfaccia di origine
- L'interfaccia di destinazione
- Tipo di cifratura usata e chiave di cifratura
- Tipo di controllo dell'integrità
- Chiave di autenticazione

Tutte queste informazioni vengono memorizzate nel **Security Association Database (SAD)**.

Come viene realizzato un datagram IPsec con ESP?

- Si "appende" un **ESP trailer al datagram originale**
 - Si **cifra il risultato** usando algoritmo e chiave specificata dalla SA
 - Si **appende al risultato l'ESP header, creando "enchilada"**. L'ESP header contiene il Security Parameter Identify e un numero di sequenza.
- Ogni volta che un datagram è spedito si aumenta il numero di sequenza per prevenire attacchi di tipo record and playback.**
- Viene creato il **MAC di tutta l'enchilada** e viene messo alla fine della stessa
 - Viene creato **un classico header IPv4** cosicché i router non interessati e gli sniffer vedano un semplice pacchetto IPv4



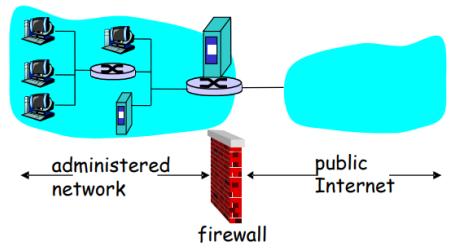
È presente anche un **Security Policy Database (SPD)**, il quale dà **informazioni all'end-point sul cosa fare** con i datagram in arrivo (**SAD** dava informazioni sul **come** fare).

Firewall

Il firewall è un **sistema hardware o software** che opera a livello di **rete o di applicazione** che controlla le connessioni di ingresso uscita.

Meccanismo di protezione che un host della rete può utilizzare: servono a vietare determinati accessi o a permetterne altri.

Esistono tre tipi di firewall: stateless packet filter, statefull packet filter, application gateway.



Come vengono implementate le politiche di sicurezza?

Attraverso le **ACL (Access Control List)**: sono tabelle di regole, applicate dall'alto verso il basso su pacchetti che arrivano ad ogni interfaccia.

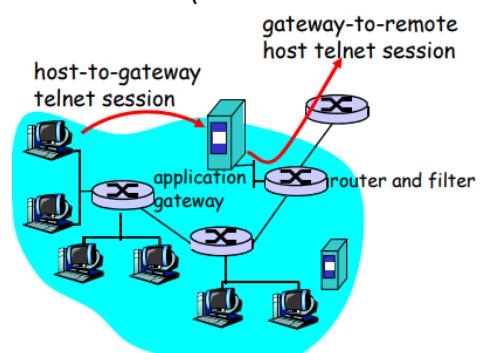
Si possono avere approcci restrittivi (inclusivi) o permissivi (esclusivi).

Nell'approccio statefull aggiunto un campo all'ACL: il pacchetto deve fare parte di una connessione attiva? Si o no.

In entrambi i casi le decisioni che si possono prendere sono a livello di rete e non utente (firewall che lavora a livello rete)

Application gateway: non si opera più a livello datagram ma a livello applicazione.

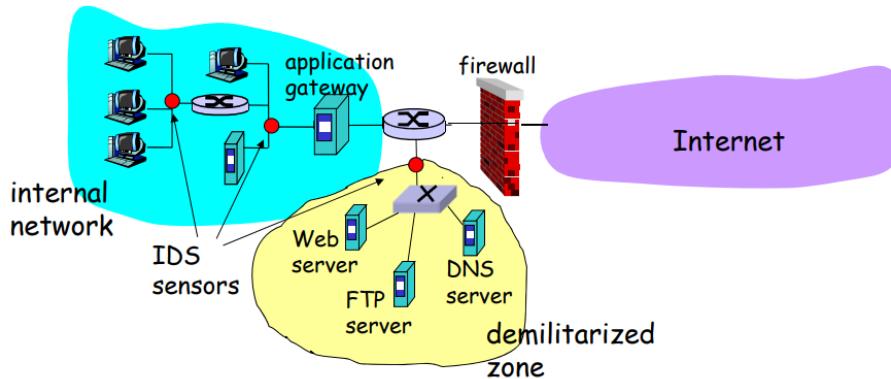
Un po' come quando la richiesta viene deviata verso il proxy server, stavolta la connessione viene deviata verso l'application gateway ed è poi l'application gateway che controlla se ci sono le condizioni, nel caso è l'app gateway stesso che apre una sessione "telnet ad esempio" con il server stesso. Tutto ciò è trasparente all'utente.



Alcune limitazioni del firewall: **ad esempio non riesce a risolvere i problemi di IP spoofing.**

Quindi **insieme ai firewall solitamente si usa IDS (Intrusion Detection Systems)**: ispezione profonda dei pacchetti: guardano al contenuto dei pacchetti o le sequenze dei pacchetti (ad esempio se si scoprono più parecchi "strani", ovvero con determinati caratteri noti per attacchi di tipo SQL injections, si fa la scansione delle porte ecc, e questi pacchetti arrivano in sequenza significa che qualcuno sta provando ad effettuare un attacco).

Se all'interno della rete sono presenti dei Server, allora si utilizza una **DMZ: non si mettono in atto particolari misure di sicurezza. Ovviamente non è a protezione zero.**



Sarebbe troppo lento accedere ai server nella zona non demilitarizzata e non sarebbe in grado di gestire tutto il traffico (motivo per il quale si mettono più IDS).

Wireless e Mobile Network

Nelle reti Wireless gli *host* sono tipicamente *laptop*, *smartphone* e possono essere anche dispositivi non mobili.

Sono presenti delle base station che possono essere stazioni radio base o access point.

Nei collegamenti wireless la potenze del segnale degrada almeno con il quadrato della distanza: è presente la path loss, questo perché il segnale può rimbalzare sugli oggetti. La comunicazione wireless è più soggetta ad interferenze o a collisioni.

La Bit error rate: è dunque elevata dipende da quanto è disturbato il canale, dalla codifica che si utilizza anche (come viene codificata l'informazione nel segnale elettromagnetico); alcune sono più robuste ma con una bit rate minore, altre hanno una bit rate maggiore ma anche una bit error maggiore.

Quando le condizioni lo consentono si cerca di fornire il bit rate più elevato, se invece il segnale diventa molto disturbato, allora il sistema automaticamente si configura su una codifica più robusta.

Tipi di reti Wireless

Classificazione in base all'area di copertura

Diversi tipi di reti: Reti cellulari, WLAN, WPAN (pensate per connettere dispositivi di uso personale, ad esempio pc a mouse ecc), WBAN (come sopra, Bluetooth).

Le Ban servono a connettere dispositivi wearable, come smartwatch. (Body Area Network).

Clasasificazione in base all'infrastruttura e al numero di hop

Le reti wireless si possono classificare in base al fatto che usino o meno infrastruttura.

Reti con infrastruttura: reti cellulari, c'è un'infrastruttura fissa, ci sono le stazioni radio base che offrono connettività. O anche le reti **wifi**, con gli access point, che danno accessibilità ai dispositivi all'interno dell'area.

Senza infrastruttura: i **nodi** possono essere **statici o mobili**, ma non c'è alcuna infrastruttura: **i nodi**

comunicano tra di loro solo perché sono dotati di interfaccia wireless: come il bluetooth, o come la rete che usa la protezione civile.

Le reti possono essere single o multi hop.

	single hop	multiple hops
infrastructure (e.g., APs)	host connects to base station (WiFi , WiMAX , cellular) which connects to larger Internet	host may have to relay through several wireless nodes to connect to larger Internet: mesh nets , sensor nets
no infrastructure	no base station, no connection to larger Internet (Bluetooth , ad hoc nets)	no base station, no connection to larger Internet. May have to relay to reach other a given wireless node MANET , VANET

Problema del nodo nascosto



È un problema che si verifica solo nelle reti wireless e non nelle reti wired.

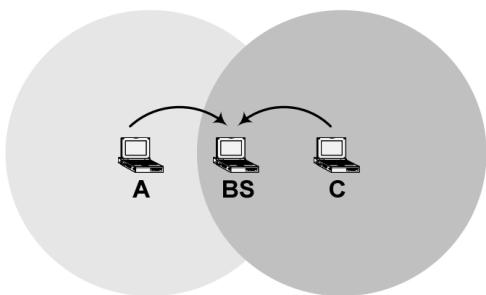
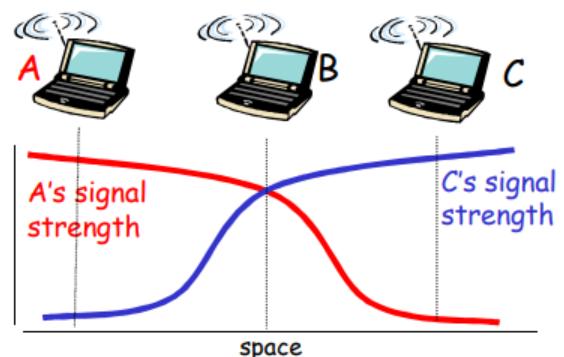
Immaginiamo la situazione della figura: sono presenti tre host: A, B e C.

Tra A e C è presente un'ostacolo, A e C non possono accorgersi di una comunicazione reciproca, A non si accorge di comunicazioni iniziate da C e viceversa.

Se C trasmette a B, A non se ne accorge e viceversa.

Il nodo C ascolta: nessuno trasmette, inizia a trasmettere, ma stava trasmettendo A verso B e quindi si verifica collisione, perché B riceve sia i messaggi di A e C.

Si può presentare anche in modo diverso: **il problema si può verificare anche a causa della distanza a causa dell'attenuazione di segnale:** supponiamo ci sia una certa distanza tra A e C, allontanandosi da A, il suo segnale viene interpretato come rumore, non come segnale (uguale per C).



Supponiamo che A stia trasmettendo a B, e B lo riceve bene perché è abbastanza vicino da ricevere con potenza relativamente alta, ma C è distante e interpreta il segnale come rumore e dal suo punto di vista il canale è libero e inizia a trasmettere a B.

B riceve entrambi i segnali: si verifica una collisione.

Problema del nodo esposto

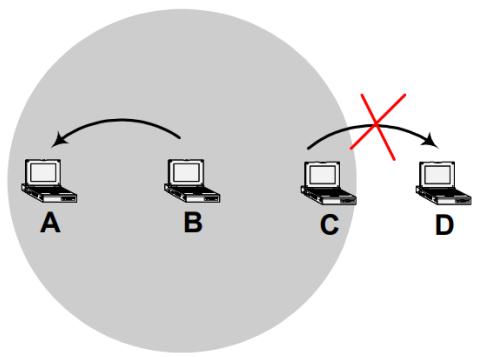
Partiamo da una precisazione: in entrambi i problemi l'area di copertura non è esattamente un raggio: il raggio non esiste, il raggio varia dinamicamente ed è diverso nelle varie direzioni.

Il nodo C vorrebbe trasmettere a D: ma non può perché sente B trasmettere, ma D sta fuori dal raggio di B.

Il messaggio di C non arriverebbe ad A, al massimo a B, ma non è un problema.

Quindi ad A e D non si verificherebbero collisioni.

C però si astiene dal trasmettere perché il protocollo è così. Problema del nodo esposto.



WiFi: standard IEEE 802.11

Le reti WiFi sono **reti con infrastruttura:** **ogni Access Point è sostanzialmente un bridge con due interfacce:** da una parte interfaccia ethernet (attraverso la quale si connette all'infrastruttura di rete) e dall'altra interfaccia WiFi.

Area di copertura: Hotspot o BSS (Basic Service Set).

Ci sono procedure di associazione (magari un host si trova tra due access point e deve sceglierne uno): **access point vicini devono operare su frequenze diverse, in modo tale da non avere interferenze.**

In generale un nodo che arriva non sa se c'è un access point ed eventualmente a quale frequenza.

- Host si mette su una frequenza e fa una scansione
- L'access point periodicamente (ogni 100 ms) manda il suo SSID e il suo MAC address
- Il nodo che fa la scansione, se non riceve segnali dopo un po' di tempo cambia canale

Dopo la fase di associazione si prosegue con l'eventuale fase di autenticazione e con l'ottenimento dell'IP tramite DHCP.

La comunicazione è di tipo **broadcast**. L'idea è che il nodo trasmetta all'acces point e viceversa, ma in realtà ascoltano tutti.

Che protocollo di accesso si utilizza? **Non** si può usare CSMA/**CD** usato in Ethernet motivi:

1. Il nodo dotato di un interfaccia wifi è dotato di una sola antenna, quindi se sta trasmettendo non può fare cd, perché non può ricevere (poco male, ci mettiamo due antenne e risolveremmo)
2. Problema serio: **nodo nascosto**.

CSMA/CA (Collision Avoidance)

Il tempo è suddiviso in tanti slot, ma lo slot non ha la durata come in slotted aloha, ha una durata molto piccola e servono solo per dire dove devono iniziare le operazioni (come se il tempo fosse continuo in realtà).

I nodi si chiamano stazioni.

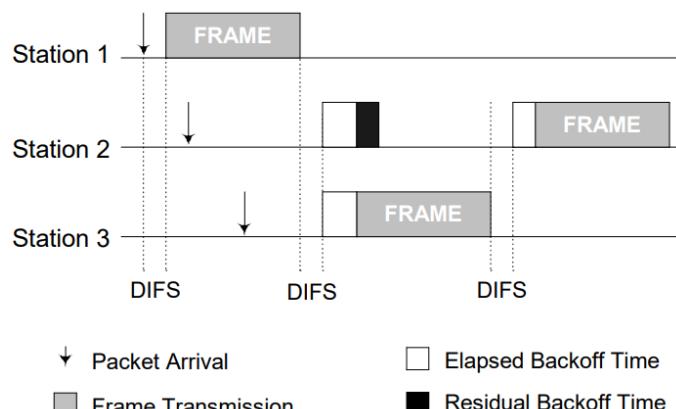
- Il nodo sorgente deve ascoltare il mezzo: se il mezzo è **libero** per un certo intervallo di tempo chiamato **DIFS** (maggiore della durata di trasmissione dell'ACK) allora il nodo sorgente può trasmettere.
- Il nodo destinatario aspetta un certo **SIFS** che deve essere inferiore del DIFS, per dare priorità al nodo destinatario, può mandare l'ACK

Per prevenire la collisione: si sparpagliano i tentivi, dopo il periodo di canale occupato, gli si fa fare una "corsa":

- dopo che un nodo ha rilevato che il mezzo è occupato, genera un intervallo di tempo casuale espresso in numero di slot, e caricano il timer di backoff: a differenza di ethernet qua si fa prima di collidere, con Ethernet invece si fa dopo la collisione. Approccio preventivo.
- Mentre il timer va, ascoltano il mezzo: se si accorgono che il mezzo diventa occupato, congelano il timer.
- Quando diventa di nuovo libero, si fa ripartire e poi se libero si fa trasmettere.

Si evitano tutte le collisioni? NO, supponiamo di non avere nodo nascosto

- Se due stazioni generano lo stesso intervallo di backoff, ci sarà una collisione.
- Se un altro genera un backoff uguale al residuo di un altro nodo, alla fine della corsa collidono.



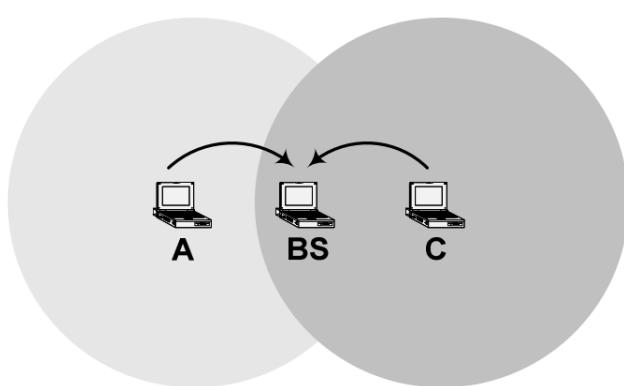
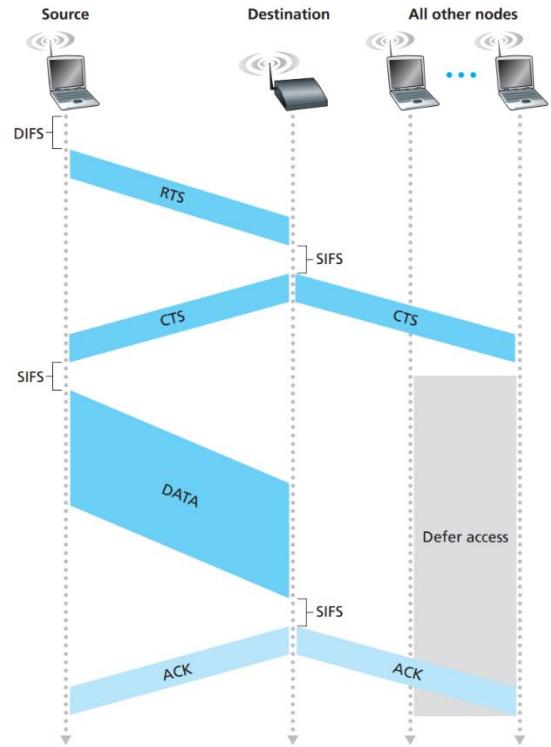
Scelta del tempo di backoff:

- Intervallo di backoff è scelto casualmente tra [0, CW-1], dove CW è la Contention Window
- Inizialmente CW=CWmin, ma ad ogni ACK mancato CW=2CW, fino a CWmax
- CWmin e CWMax sono parametri dipendenti dal mezzo fisico.

Virtual Carrier Sensing

Tra il problema del nodo nascosto e quello del nodo esposto il più critico è il primo: si verificano delle collisioni, quindi vi è consumo energia, perdita del throughput e di tempo (il tempo anche nel nodo esposto). Quindi per ovviare si fa un Virtual Carrier Sensing:

- Nodo sorgente che vuole mandare un pacchetto: fa Physical Carrier Sensing.
Ascolta il mezzo per un DIFS, se libero manda un RTS (Request To Send)
- Il frame viene trasmesso alla Base Station, ma viene sentito da tutti: siccome all'interno del RTS è presente un campo duration, gli altri host sanno che per tutto l'intervallo di duration il mezzo è occupato.
Duration va dalla fine della trasmissione del frame alla fine della trasmissione dell'ACK
- Base Station risponde con CTS (Clear To Send): anche questo frame contiene un campo duration, stavolta decrementato di SIFS più durata del CTS.



C non vede l'RTS di A, ma vede il CTS di risposta del BS: quindi gli altri dispositivi devono settare un timer uguale alla durata del duration del campo del pacchetto che vedono. Questo significa che C non trasmette!!!

Una volta caricato il NAV (il timer) viene fatto partire immediatamente: un nodo non può trasmettere finché il suo NAV è attivo, anche se il mezzo risulta libero

(prendendo il caso di C, se C ascoltasse e non aspettasse il NAV, colliderebbe sicuramente).

Ecco il virtual CS: si va ad “ascoltare” il mezzo, ovvero vado a vedere se il NAV è attivo.

Le collisioni non sono totalmente evitate: ci possono essere collisioni su RTS!

La finestra di RTS però è molto piccola, quindi è poco probabile che avvengano e, qualora avvenisse, non sarebbe grave in quanto bisognerebbe ritrasmettere pochi byte.

Costo: scambio di RTS e CTS, diminuisce il throughput!

Non conviene sempre: se il frame è piccolo quanto l'RTS conviene mandare direttamente il frame in modo tale da eliminare l'overhead.

Frame WiFi

2	2	6	6	6	2	6	0 - 2312	4
frame control	duration	address 1	address 2	address 3	seq control	address 4	payload	CRC

- Address 1: MAC host o access point
- Address 2: MAC sorgente
- Address 3: MAC del router, perchè l'AP è l'ultimo Hop, ma essendo un bridge non sa fare switching e quindi bisogna specificarglielo direttamente
- Address 4: usato nella modalità ad hoc

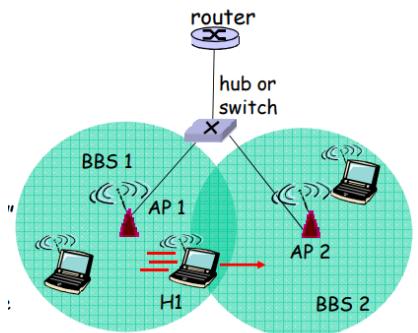
Mobilità

Può accadere che l'utente si sposti e che rimanga collegato sempre allo stesso hotspot e rimanga nell'area di copertura dello stesso access point. A livello Data link non viene percepito questo spostamento. L'access point di riferimento è sempre lo stesso.

Possibile che si sposti l'host e che cambi access point di riferimento: la mobilità viene sempre gestita a livello data link: i due access point di mettono d'accordo: c'è un momento in cui c'è la porcedura di associazione con l'access point 2.

E lo switch come fa a saperlo? Manda un pacchetto all'access point 1. Ogni tanto lo switch ripulisce le tabelle e l'Access point 2 manda un messaggio di broadcast per comunicare che l'host ora sta in quell'hotspot.

Ci possono essere perdite durante l'handoff.



Power Management

Problema energetico, gli host mobili hanno solitamente poca autonomia, a casua della batteria. Quindi si deve cercare di limitare il consumo della batteria: la rete è uno degli elementi che consuma di più.

Standard IEEE 802.11 incorpora un meccanismo di power management: i nodi non sono costretti a stare sempre accessi, ma possono andare a dormire per un periodo: chi dorme, non riceve bit. L'ideale sarebbe dormire e ricevere.

Se un host dorme, la trasmissione non è un problema, basta che l'host si svegli, ma il problema rimane ricevere.

L'access point emette periodicamente dei Beacon: per la sincronizzazione (occorre periodicamente risontinazzare i clock: nel beacon c'è scritto il clok dell'access point così che tutti gli host si allineino), ma non solo: utili per gestire il meccanismo di power management.

- I nodi vanno a dormire, informano l'access point: l'access point ne prende nota
 - Se arrivano dei frame per il nodo dormiente, non vengono inoltrati, ma vengono mantenuti dall'access point
 - I nodi dormienti si svegliano un attimo prima di ricevere il Beacon
 - Dopodichè: l'access point quando manda il Beacon, manda a dire anche quali sono i nodi per cui ci sono pacchetti in attesa.
 - Se un nodo dormiente ha dei pacchetti di attesa, si sveglia e chiede esplicitamente i pacchetti in attesa: l'access point riceve ed inoltra i pacchetti.
- Altrimenti il nodo torna a dormire

Picolo degrado di prestazione, ma lo sleep delay è bassissimo, se il Beacon viene mandato uno ogni 100ms, si tollera.

Rete cellulare

Sono delle reti wireless con infrastruttura, costituita dalle **mobile switching center**, centrali per la rete mobile, che (ciascuna di queste) gestiscono delle stazioni radio base: ognuna di queste gestisce alcuni dispositivi nell'area. Ognuno di questi dispositivi si collega a una di queste stazioni radio base, che si collega allo switching center che si collega all'infrastruttura fissa.

Ognuna di queste aree gestite dalla stazione radio base si chiamano celle. Le celle sono esagonali, solo nei disegni perché nella realtà sono circolari più o meno... in realtà nemmeno così, perché il mondo non è perfetto. Come per le wan, le aree di copertura si modifcano, si allargano e si restringono. Il mezzo è broadcast: si utilizza come protocollo di accesso multiplo un mix di tdma, fdma, e altre cose.

Diverse generazioni: dalla prima generazione che era analogica, 2G (nota come GSM) pensata solo per la voce ma digitale, combo di TDMA e FDMA: voce codificata a 13kbps fino alla 5G pensata per applicazioni real time mobili.

Addressing and routing per utenti mobili

Internet è stata progettata per host fissi, infatti si suppone che quando c'è una perdita di pacchetti questa sia dovuta ad una congestione. Il protocollo IP ad esempio assume implicitamente che gli host siano stazionari: si può gestire in prima maniera con il DHCP. In alcuni casi non può andar bene.

Cosa si intende per mobilità dal punto di vista di Internet?

L'host è mobile oppure no dal punto di vista di IP? Ci sono diversi tipi di mobilità. **Non è da confondere dalla mobilità dal punto di vista dell'utente: non sempre combaciano!**

Un host è mobile quando cambia punto di accesso alla rete.

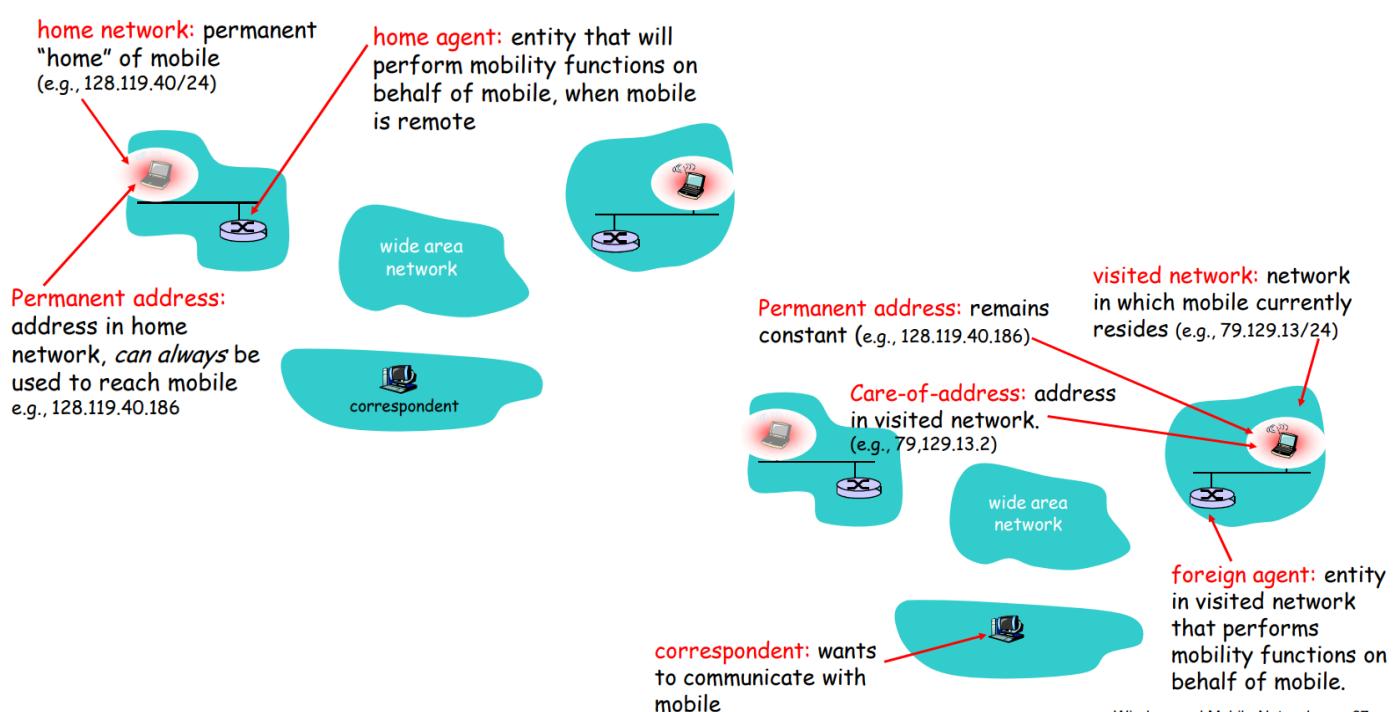
Se cambio access point l'host non è considerato mobile in quanto l'IP è lo stesso!

In diverse applicazioni (Mail, ad esempio) non è un problema la mobilità e si può gestire con DHCP.

Nello streaming invece, se utilizza il protocollo TCP, quando ci si sposta da una sottorete ad un'altra, cambio indirizzo IP: le connessioni TCP sono caratterizzate da 4 parametri (IP dest e src, porta IP src e dst) e se uno di questi cambia, la connessione viene meno.

Come si fa a mantenere la continuità del servizio? Cercando di mantenere la continuità dell'IP, nonostante il fatto che l'utente si sposti.

Partiamo da un po' di nomenclatura:



- Quando l'host mobile arriva sulla rete visitata si deve registrare: gli dice chi è e da dove viene e qual è l'home agent.
- Il foreign agent deve contattare l'home agent per comunicargli che l'host mobile ora risiede in quella rete
- L'home agent sa ora a chi deve inoltrare i pacchetti destinati all'host mobile.

Il corrispondente però come fa? Ci sono due approcci

- **Approccio indiretto:**

- il corrispondente non sa niente e manda al solito indirizzo
- l'home agent intercetta il pacchetto e lo manda al foreign agent (perché lo sa lui, gli è stato comunicato prima).
- Quando l'host mobile risponde risponde direttamente al corrispondente.

Piccolo **inconveniente**: si fa la triangolazione del routing. Supponiamo che il corrispondente sia nella stessa rete "foreign network": il corrispondente ha dovuto mandare il pacchetto fuori dalla rete per poi farlo rientrare e questo perché non lo sapeva. Si perde in efficienza ma è un approccio trasparente al corrispondente host.

Se l'host è molto mobile e cambia nuovamente rete si deve registrare presso la nuova foreign network: il foreign agent nuovo riceve le nuove informazioni sull'home agent ecc, il foreign agent contatta 'home agent il quale ora sa dove si trova l'host mobile', non più sulla foreign network 1 ma sulla foreign network2, e così via.

Quindi ogni volta che si sposta l'host mobile, l'home agent lo sa perché i foreign agent lo contattano. **Ma a causa del ritardo di propagazione, e al ritardo dovuto a spostamento, registrazione e comunicazione tra agent c'è un periodo di tempo in cui l'home agent pensa ancora che l'host mobile si trovi sulla foreign network 1.**

Cosa succede ai network instradati verso questa rete dopo che l'host mobile si è spostato? Saranno recapitati alla foreign network 1, ma non possono essere consegnati: si perdono i pacchetti, ci sarà dunque una piccola interruzione di servizio.

- **Approccio diretto:**

- Il corrispondente manda un datagram all'IP permanente dell'host mobile
- l'home agent sa che si è spostato: risponde quindi al corrispondente dicendogli la nuova posizione dell'host mobile: "guarda devi mandarlo a questo IP: xxxx";
- il corrispondente manda questo datagram verso la foreign network, dove il foreign agent inoltra il pacchetto all'host mobile.

Questo metodo anche se supera la triangolazione, non è trasparente rispetto al corrispondente agent, perché il corrispondente sa che l'host mobile non è "in sede" e sa dove si trova ora (privacy).

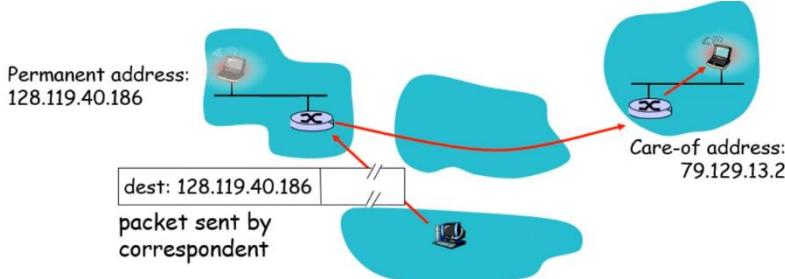
Cosa succede se l'host mobile cambia di nuovo posizione?

Il corrispondente manda il pacchetto alla solita foreign network che lui conosce, ma l'host mobile si è spostato, ma così facendo questi pacchetti si perderanno: c'è modo per garantire la continuità della comunicazione? Quando l'host mobile si sposta nella nuova rete, si registra al foreign agent2: comunica con l'home agent (e per i corrispondenti successivi va bene, ma non per quelli precedenti, in quanto quelli sanno ancora che l'host si trova nella rete visitata 1): il nuovo foreign agent allora deve comunicare al vecchio foreign agent che ora l'host mobile si è spostato nella sua rete: così che tutti i datagram per l'host mobile che arrivano al vecchio foreign agent devono essere inoltrati dal foreign agent vecchio alla nuova foreign network. Ma se si sposta ancora l'host mobile? Si viene a creare una catena. Si salva la situazione ma più ritardo e più overhead.

Mobile IP

Il metodo utilizzato per il routing è quello indiretto.

Supponiamo di essere in questa situazione, in cui il correspondent agent manda un pacchetto all'host mobile e lo manda dunque al permanent address.



L'home agent intercetta il pacchetto e sa che quel pacchetto è per l'host mobile che sta in un'altra rete.

Come lo ridirige?

Facendo incapsulamento: **IP over IP**.

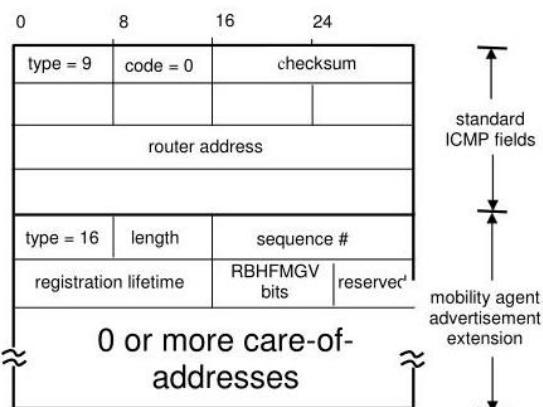
Nel payload di questo nuovo datagram, c'è l'intero datagram ricevuto, ma:

- nel campo destination address viene messo il care of address
- il source address rimane lo stesso
- Nel **campo di protocollo superiore c'è scritto IP**, di fatto il livello rete quando riceve questo datagram lo riconsegna a sé stesso.

packet sent by home agent to foreign agent: a packet within a packet



Agent discovery

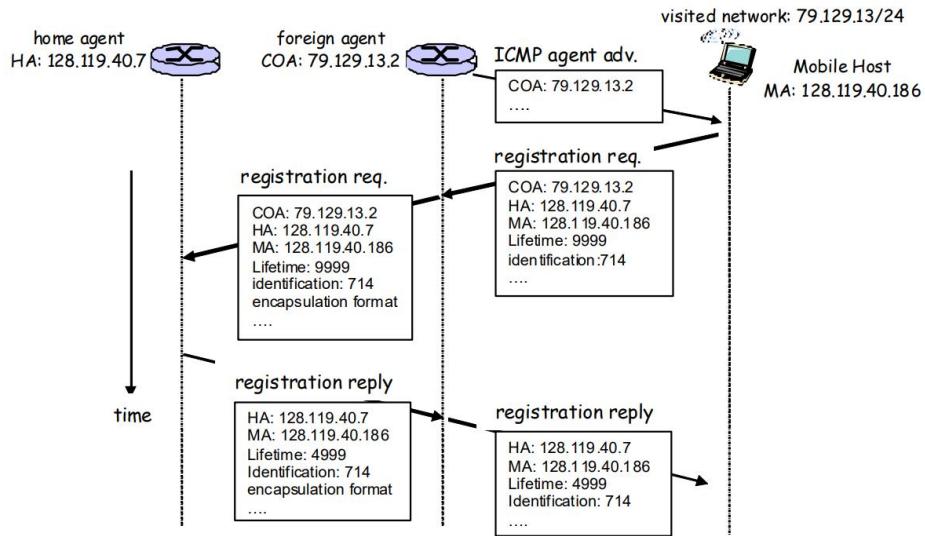


Periodicamente l'home/foreign agent manda un **pacchetto ICMP type 9 e code=0**, nel formato:

Da notare i flag:

- H/F, ad indicare se l'agente è un home o foreign (solitamente è entrambi contemporaneamente)
- R, il bit registration required

Registrazione dell'host mobile ad una foreign network



La registrazione avviene in 5 passi:

1. Host mobile si mette in ascolto e riceve l'advertisment message da parte dell'agent
2. Ricevuto il care of address si assegna questo indirizzo ed invia come messaggio al foreign agent:
 - a. il care of address
 - b. il proprio indirizzo permanente
 - c. l'inidirizzo del proprio home agent
 - d. il lifetime (per quanto tempo pensa di rimanere nella rete)
 - e. Identification (che serve a identificare la risposta come nel DHCP)
3. Il foreign agent riceve il messaggio, si scrive le informazioni e le inoltra all'home agent. Manda anche informazioni sull'encapsulation (registration request)
4. L'home agent risponde con registration reply, (ad esempio può anche abbassare il lifetime)
5. Il foreign agent riceve, aggiorna le informazioni e inoltra il registration replay all'host mobile

Impatto della mobilità

A causa della mobilità si possono perdere dei pacchetti (handoff tra access point, cambio della rete da parte dell'host mobile, ecc) e il protocollo TCP assume implicitamente che ci sia congestione, abbassa il rate e il throughput diminuisce e quando l'host mobile arriva alla nuova rete, il throughput rimane basso a causa del controllo di congestione e le performance sono diminuite. Stessa cosa accade nei collegamenti WiFi. Si devono prendere accorgimenti:

- recupero locale, così da assorbire le perdite a livello data-link, spesso viene usato questo metodo.
- Modificare TCP cosicchè il TCP possa distinguere se le pedite sono dovute a congestione o meno, attraverso un feedback dal router.
- Split connection: spezzare la connessione tcp in due tronconi: la prima per la parte wired e la seconda per la parte wireless, interrompendo la connessione ad esempio al router wifi.
Usando ad esempio un altro protocollo per la parte wireless.
Split connection è usato nelle applicazioni di streaming, ma affinchè funzioni è necessario che il server sia vicino al client.

Reti senza infrastrutture: ad hoc network.

Sono reti costituite solo da nodi con interfaccia wireless, come ad esempio le reti bluetooth.

Sono reti:

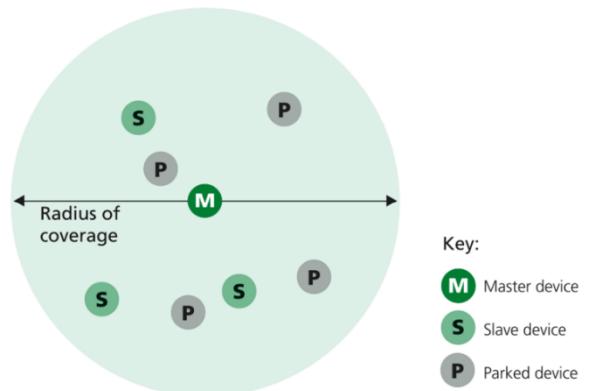
- **low power**, con un basso dispendio energetico
- **low-cost**
- **short-range** (max 10m).

Bluetooth

Inizialmente la rete bluetooth era stata pensata come cable replacement (tastiera, mouse, stampante) non si hanno cavi, ma si ha il problema della batteria.

I dispositivi di una rete bluetooth devono organizzarsi da soli:

- chi inizia la conversazione solitamente è il **nodo chiamato master**
- possono essere presenti al **massimo 8 nodi attivi**. I nodi non master sono detti **slave**.
- Possono essere **presenti fino a 255 parked devices**. Questi dispositivi non possono comunicare finché il loro stato non viene cambiato da parked ad attivo dal nodo master.
- La rete che si viene a creare è chiamata **Piconet**



Anche il Bluetooth opera nel range di frequenza 2.4 GigaHertz ed utilizza diverse frequenze (78).

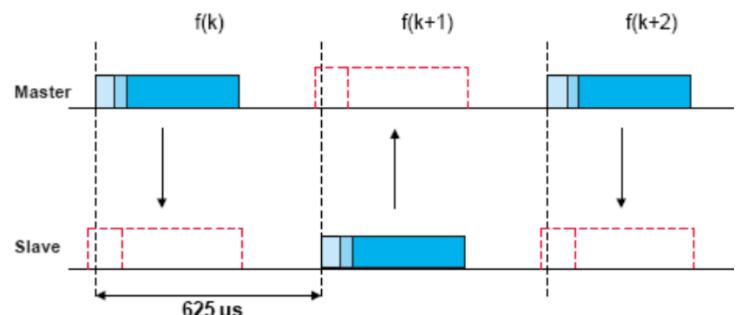
Anche il WiFi opera nella stesso range ma ha 11 canali.

Essendo una banda molto affollata potrebbero crearsi delle interferenze e il segnale potrebbe riultare distrubato: si opera un **frequence opting**, ovvero le **78 frequenze vengono usate ciclicamente secondo sequenza pseudo random che conoscono tutti i nodi**.

In questo modo si rende più robusta la comunicazione: cambiando continuamente la frequenza, quello che può accadere è che in un piccolo intervallo di tempo effettivamente il segnale sia disturbato, ma poi nelle frequenze successive almeno aumenta la probabilità di successo della trasmissione.

Il protocollo di accesso è time division duplexing (gli slot sono lunghi 625 micro secondi), ed è utilizzato un **protocollo di polling**.

- **Gli slot pari sono usati dal master**, per decidere chi deve comunicare mandando il **pacchetto di poll**
- **Gli slot dispari sono usati dagli slave**, che ricevuto il pacchetto di poll possono mandare un pacchetto o, se non hanno nulla da dire, mandare un pacchetto di NULL.



Talvolta si posso trasmettere **pacchetti più lunghi di uno slot, ma sempre in numero dispari**, così da non alterare la cosa sequenza.

Durante la trasmissione non si può cambiare frequenza.

Laboratorio

Indirizzamento riepilogo

Un host per poter essere connesso alla rete deve avere:

- Un indirizzo IP
- Un netmask
- Un indirizzo di default gateway
- Indirizzo DNS server (anche se non strettamente necessario)

L'indirizzo IPv4 è una sequenza di 32 bit nella quale i primi K-bit identificano la rete e i restanti 32-k bit identificano l'host.

Rete	Host
------	------

Come fa però il router o l'host a identificare l'indirizzo della rete e l'indirizzo di broadcast? Utilizza la netmask: una sequenza di 32 bit, in cui i primi k-bit che identificano la rete sono tutti posti a 1.

Per identificare *l'indirizzo di rete: and bit a bit IP – netmask*; per indirizzo di broadcast *or bit a bit con netmask negata*.

Quando un host deve mandare un pacchetto (un datagram a livello rete) controlla se la rete di destinazione è la stessa di appartenenza: se sì lo invia direttamente, altrimenti lo inoltra al default gateway.

Mostrare e configurare interfacce: comando ip

`$ ip addr show {up}` Mostra le interfacce {attive}

Per abilitare e configurare un'interfaccia è necessario avere i privilegi di admin.

`# ip link set eth0 { up | down}` Abilita/disabilita un'interfaccia

`# ip addr add 192.168.1.21/24 broadcast 192.168.1.255 dev eth0` Assegna indirizzo interfaccia

`# ip addr flush dev eth0` Rimuove ip interfaccia

LE CONFIGURAZIONI COME QUESTE, CHE NON SONO PRESENTI NEL FILE DI CONFIGURAZIONE NELLA PARTE *auto* VENGONO ANNULLATE AL RIAVVIO DELLA MACCHINA.

Il file di configurazione è `/etc/network/interfaces`.

iface eth0 inet static
address 192.168.1.2
netmask 255.255.255.0
broadcast 192.168.1.255

Comandi ***ifup*** e ***ifdown*** abilitano/disabilitano le interfacce con le configurazioni presenti nel file.

ifup –a abilita tutte le interfacce presenti nella parte *auto* ed è eseguito all'avvio automaticamente.

Gateway

Per aggiungere il default gateway, nel **file di configurazione** basta aggiungere all'interfaccia "**gateway 192.168.1.1**".

Per visualizzare la tabella di routing: ***ip route show***

per aggiungere rotte:

- Stessa rete: `# ip route add 192.168.1.0/24 dev eth0`
- Default gateway: `# ip route add default via 192.168.1.1`

DNS

Il Domain Name Service è quel servizio di livello applicazione che ci permette, tra le altre cose, di associare un nome di dominio ad un indirizzo IP e viceversa.

DNS statico

Nel file **/etc/hosts** basta associare: *IPv4* *dominio*

DNS Server

Nel file **/etc/resolv.conf** basta porre: nameserver 8.8.8.8.

Come scegliere e quale scegliere? Ovviamente la configurazione statica è la peggiore tra le due: se un dominio cambio IP o quell'IP non è raggiungibile, non è possibile collegarsi a quel dominio, oltre al fatto che il file dovrebbe avere dimensioni molto grandi.

Quindi si usa il **Name Service Switch**: nel file **/etc/nsswitch.conf** si scrive:

hosts: files dns

In questo caso, per risolvere un nome, prima si va a cercare sul file /etc/host e poi si contatta il DNS server.

Vedere se il DNS è configurato correttamente: \$ nslookup dominio.

DHCP

Sappiamo che la configurazione delle interfacce non viene fatta a mano dall'utente o dal gestore della rete, ma viene utilizzato il Dynamic Host Configuration Protocol. È necessario dunque avere un DHCP server da contattare.

DHCP Server

1. Installare il DHCP Server: **# apt-get install isc-dhcp-server**
2. Andare su **/etc/default/isc-dhcp-server** e impostare come interfaccia INTERFACES = "eth0"
3. Andare su **/etc/dhcp/dhcpd.conf** e scrivere:

```
option domain-name-servers 192.168.0.2, 8.8.8.8;
option routers 192.168.0.1;
default-lease-time 3600;          #tempo di durata dell'ip
subnet 192.168.0.0 netmask 255.255.255.0 {
    range 192.168.0.10 192.168.0.100;      #ip assegnabili dal DHCP server
}
```

4. Eseguire: **#systemctl restart isc-dhcp-server.service**

Client DHCP

Nel file **/etc/network/interfaces** basta impostare l'interfaccia di interesse del tipo:

iface eth0 inet dhcp

Test di connettività

Utilizzano il protocollo ICMP, il quale è un protocollo ausiliario al protocollo IP di livello Rete.

Ping

Tra i vari pacchetti ICMP, si possono usare ad esempio, i pacchetti ping, che permettono di svolgere un test di connettività.

1. Il pingante manda un **ICMP echo request**
2. Il destinatario risponderà con **ICMP echo reply**

Il **comando ping**, eseguito da A, invia a B **una serie di pacchetti Echo Request (per default, uno al secondo)** e **calcola la percentuale di pacchetti ricevuti e il Round Trip Time (RTT)**.

Possibili errori: Network unreachable; 100% packet loss; Unknown host.

Traceroute

Mostra il percorso (gli indirizzi IP dei router attraversati) che un pacchetto IP effettua per raggiungere un host destinatario.

Il campo TTL (time-to-live) del datagram IP specifica il numero di apparati di rete che il pacchetto può attraversare prima di «scadere».

Ogni router che riceve un pacchetto, prima di inoltrarlo, diminuisce TTL di 1.

Se TTL=0, invia al mittente un messaggio di errore ICMP di tipo Time Exceeded con l'indirizzo del router che ha portato TTL a 0.

Funzionamento:

1. Invia all'host destinatario una serie di **terne di pacchetti UDP con TTL crescente**.
La prima terna con TTL=1, la seconda con TTL=2, e così via
2. Tiene traccia degli indirizzi IP all'interno dei messaggi ICMP Time Exceeded ricevuti finché l'ultimo pacchetto della serie raggiunge l'host destinatario
3. Il destinatario, quando viene raggiunto, non manda ICMP Time Exceeded, ma un ICMP Destination Unreachable: UDP ha bisogno di una **porta**. I pacchetti verso il destinatario necessitano di una porta in ascolto. **Scegliendone una alla cieca, è improbabile beccarne una in ascolto**, ecco perché l'host invia **Destination Unreachable (con code=3, cioè port unreachable)**.
4. Traceroute ordina gli IP in base al TTL crescente e ricostruisce il percorso. Fornisce anche i round-trip dei pacchetti.

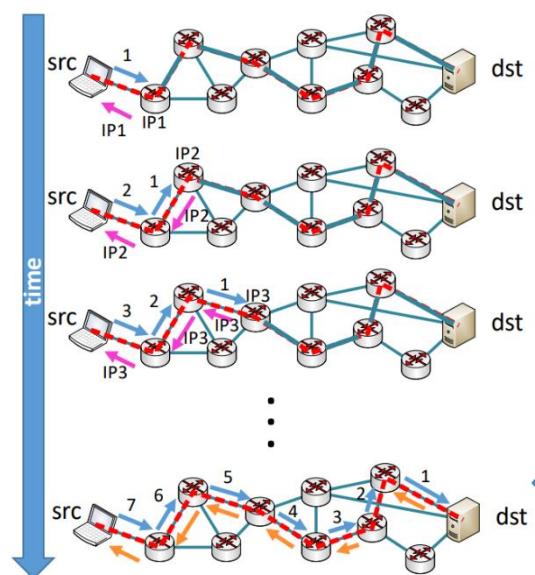
Tracerout come output fornisce dunque:

Domino ip RTT1 RTT2 RTT3

Ma potrebbe fornire come output: *****

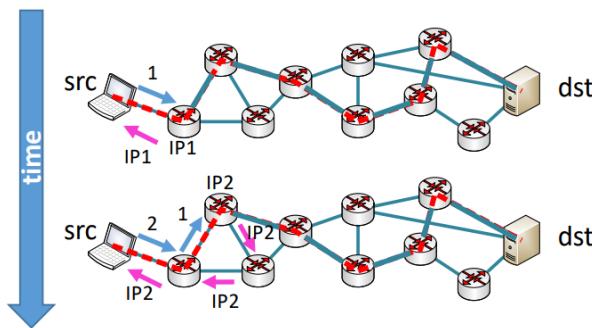
In quel caso significa **che è scaduto il timeout (default 5 sec)**: non è stato ricevuto il messaggio di risposta per alcun pacchetto della terna.

Possibili cause: device non configurato per rispondere a traffico ICMP/UDP; **pacchetti scartati per motivi di rete (es: traffico bloccato da un firewall)**.



Problema:

- I pacchetti possono seguire più percorsi, quindi gli IP ricavati possono riferirsi a più percorsi.
- Se i pacchetti e i messaggi ICMP seguono percorsi diversi, il calcolo del round-trip time è inaffidabile.



Firewall

Meccanismo di protezione che un host della rete può utilizzare: servono a vietare determinati accessi o a permetterne altri.

Il firewall è un sistema hardware o software che opera a livello di rete o di applicazione che controlla le connessioni di ingresso uscita.

Network Layer Firewall: operano a livello TCP/IP, **analizzando header IP, TCP e UDP**.

Application Layer Firewall: analizzano tutto il pacchetto, header e payload, e possono analizzare tutto il traffico fino a livello 7 dello stack ISO-OSI, sono più efficaci ma usano più risorse

Level	Shallow Packet Inspection	Medium Packet Inspection	Deep Packet Inspection	ISO/OSI
7				Application
6				Presentation
5				Session
4				Transport
3				Network
2				Data Link
1				Physical

Firewall a filtraggio di pacchetto stateless

Ogni firewall contiene una **tabella delle regole**, ogni regola ha

- un indice progressivo
- criteria (insieme di campi):
 - IP sorgente
 - Porta sorgente
 - IP destinatario
 - Porta destinatario
- Azione: può avere valori: DROP o ACCEPT

Indice	IP sorgente	Porta sorgente	IP destinatario	Porta dest.	Azione
1	111.11.11.0/24		222.22.0.0/24		BLOCCA
2	111.11.0.0/16		222.22.22.0/24		ACCETTA
3	0.0.0.0		0.0.0.0		BLOCCA

L'ultima regola è la default rule: se non soddisfi nessun criterio, o **scarto tutto (firewall inclusivo)** o **accetto tutti (firewall esclusivo)**.

Per ogni pacchetto: scorre le regole (in ordine crescente di indice) e quando ne trova una i cui criteri corrispondono ai criteri del pacchetto, applica quella sola regola, poi analizza altri pacchetti.

UNA REGOLA PER PACCHETTO, TROVATA E APPLICATA SI PASSA AL PROSSIMO.

Firewall Inclusivo l'ultima regola blocca tutto, sicuro ma scomodo, senza definire regole non si può accedere a nulla.

Firewall esclusivo l'ultima regola consente tutto, comodo ma insicuro, devo prevedere e inserire manualmente tutte le regole che ritengo utili.

Netfilter e iptables

Netfilter: componente kernel di linux che offre le funzionalità di packet filtering e NAPT

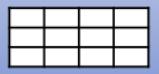
iptables: programma per configurare le tabelle delle regole (filter e nat)

Iptables

Lavora sulle tabelle filter e nat: ogni tabella ha 3 *chain*, ognuna con regole da applicare a una diversa categoria di pacchetti.

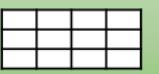
La tabella filter è quella che si occupa del filtraggio dei pacchetti, ha i seguenti 3 chain:

INPUT



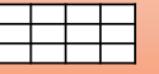
per i pacchetti in ingresso destinati ai processi locali

OUTPUT



per i pacchetti in uscita dai processi locali

FORWARD



per i pacchetti in transito, cioè da inoltrare ad altri host

- **# iptables [-t table] -L [chain]**
per visualizzare le regole di un determinato chain, di default la tabella è filter e i chain tutti
- **# iptables [-t table] -A chain rule-specification**
per inserire una regola in fondo alla catena
- **# iptables [-t table] -I chain [num] rule-specification**
per inserire una regola alla riga specificata: di default è 1, quindi viene messa in testa
- **# iptables [-t table] -D chain rule-specification**
iptables [-t table] -D chain num
Per eliminare una regola
- **# iptables [-t table] -F [chain]**
Per rimuovere tutte le regole da una o più catene
- **# iptables [-t table] -P target**
Per cambiare la policy (ovvero il comportamento di default)

Tipi di regole:

- `-p <protocollo>` protocollo (TCP, UDP, ICMP, ...)
- `-s <address>` indirizzo IP sorgente
- `-d <address>` indirizzo IP destinazione
- `--sport <port>` porta sorgente
- `--dport <port>` porta destinazione
- `-i <interface>` interfaccia di ingresso
- `-o <interface>` interfaccia di uscita
- `-j <target>` azione (DROP/ACCEPT)

Esempio:

```
# iptables -A OUTPUT -p tcp -d 10.0.5.4 --dport 80 -j DROP
```

LE REGOLE NON VENGONO SALVATE!

Necessario salvarle e ricaricarle:

- `# iptables-save > file`
- `# iptables-restore < file`

Tabella NAT

La tabella NAT ha 3 catene:

- PREROUTING, fa destination NAT (D-NAT), cioè altera indirizzo/porta di destinazione dei pacchetti in arrivo
- OUTPUT fa destination NAT (D-NAT) dei pacchetti in uscita dai processi locali, prima del routing
- POSTROUTING fa source NAT (S-NAT), cioè altera indirizzo/porta sorgente dei pacchetti in uscita

Stesse regole della filter, il comando infatti è lo stesso.

Ovviamente non abbiamo le stesse rule-specification ma:

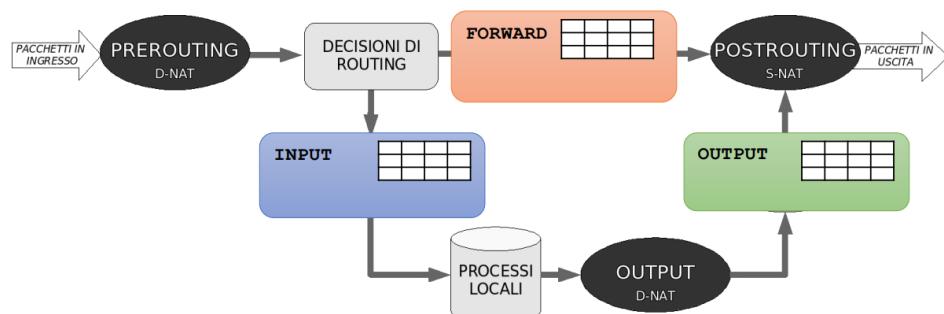
- `--to`
- `--to-source`

Esempi:

```
# iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -j SNAT --to-source 151.162.50.1:4001-4100
```

```
# iptables -t nat -A PREROUTING -d 151.162.50.2 -j DNAT --to 192.168.0.2
```

Con il PAT la porta caratterizza non un processo ma un host (come faceva l'IP).



Ripasso sul C

Allocazione e rilascio della memoria

Allocazione normale

```
void* punt;  
punt = malloc(dimensione);  
free(punt);
```

Allocazione con cast

```
double* punt;  
punt = (double*)malloc(sizeof(double)*dimensione);  
free(punt);
```

String

```
#include <string.h>
```

Sono array di char che terminano con '\0'.

strlen(char*) → restituisce la lunghezza, ma non conta '\0'!

strcmp(char*,char*) → restituisce int, <0 se str1 minore di str2 (come ASCII), >0 se viceversa, 0 se uguali.

strncpy(str1,str2,n) → copia al più n caratteri da str2 su str1

strncat(str1,str2,m) → concatena m caratteri di str2 con str1

Attenzione: *strncat e strcpy non si occupano della gestione di '\0' e della dimensione*; quindi, ad esempio, n in strncpy è l'ideale che sia sizeof(str1).

File

```
#include <stdio.h>
```

```
FILE* fd;  
fd = fopen(file , mode );
```

Mode può essere:

- r, sola lettura
- w, sola scrittura
- a, append
- r+, lettura e scrittura
- a+, lettura e append

Se il file non esiste e si apre in scrittura o append, viene creato.

fopen potrebbe fallire: il file non esiste (in lettura), voglio scrivere, il file non esiste ed è impossibile allocare nuova memoria.

fscanf(fd,"%d",&n)	--> leggere dal file
fprintf(fd, "%s",str)	--> scrivere sul file

Nota:(f)printf/scanf restituiscono il numero di caratteri scritti/letti.

Comunicazione tra processi: socket

Programmazione distribuita in C:

Andiamo ad analizzare prima il modello client-server.

Possiamo vedere i due processi in esecuzione rispettivamente su client e server in maniera stratificata:

- Livello applicativo
- Livello di trasporto (TCP/UDP)
- Livello di rete:
 - IP
 - Interfaccia

Ogni livello aggiunge informazioni: quando avviene la comunicazione tra un client e un server si scende e si rissale attraverso questa pila, un'analogia può essere la matrioska o una caramella con molti strati: per poter spedire i dati bisogna impacchettare tutti gli strati e per poter ricevere bisogna scartare tutti gli strati, dove però su ogni strato ci sono delle informazioni.

Affinché all'utente e al programmatore questa *comunicazione risulti orizzontale*, senza doversi preoccupare di frammentazione, handshake, ecc., **c'è bisogno dei socket** (letteralmente presa): **sono un'astrazione che il Sistema Operativo offre attraverso le primitive**:

- Per creare il socket
- Assegnarli un indirizzo IP (e una porta)
- Connettersi a un altro socket
- Accettare/richiedere una connessione
- Chiudere una connessione

Un socket è l'estremità di un canale di comunicazione fra due processi in esecuzione su macchine connesse in rete.

Creazione del socket

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
int socket(int domain, int type, int protocol);
```

domain: famiglia dei protocolli che utilizziamo: *AF_LOCAL*, *AF_INET* rispettivamente per comunicazione locale e comunicazione con protocolli IPv4

type: tipologia di socket: *SOCK_STREAM*, *SOCK_DGRAM* rispettivamente per TCP E UDP

protocol: per noi vale sempre 0

Restituisce il descrittore del socket (di file): (*in caso di errore restituisce -1*) cioè un numero che rappresenta il socket aperto dal processo e sul quale può fare operazioni

Attenzione: dopo la chiamata alla primitiva socket, questo non è associato nè ad un IP nè ad una porta

Struct per gli indirizzi

```
#include <sys/socket.h>
#include <netinet/in.h>
```

La prima struttura ci permette di associare ad una porta: la famiglia (o dominio), la porta e l'indirizzo (ma in formato numerico big endian)

```
struct sockaddr_in{
    sa_family_t    sin_family;      // permette di specificare la famiglia, AF_INET (o AF_LOCAL)
    in_port_t      sin_port;        // Permette di specificare della porta
    struct in_addr sin_addr;       // Indirizzo, guarda sotto
}

struct in_addr{
    uint32_t s_addr;               // indirizzo, espresso in network order, un int unsigned su 32 bit
}
```

Nota: usiamo i tipi opachi perché così siamo sicuri che funzionino su tutte le architetture.

Conversione dell'endiannes

```
#include <arpa/inet.h>
```

Funzioni di conversione da little a big endian.

```
uint32_t htonl(uint32_t hostlong)      //host to network long
uint16_t htons(uint16_t hostshort)     //host to network short
uint32_t ntohl(uint32_t netlong)       //network to host long
uint16_t ntohs(uint16_t netshort)      //network to host short
```

Conversione del formato dell'indirizzo

Gli indirizzi possono rappresentati come:

- **Formato presentazione:** decimale puntata, es **192.168.1.10**
- **Formato numerico:** rappresenta l'indirizzo come un intero unsigned su 32 bit

Conversione presentazione → numerico:

```
int inet_pton(int af, const char* src, void* dst);
```

- af: famiglia, **AF_INET**
- src, indirizzo in formato presentazione
- dst, area di memoria dove memorizzare il risultato

Conversione numerico → presentazione:

```
const char* inet_ntop(int af, const void* src, char* dst, socklen_t size);
```

- af: famiglia, **AF_INET**
- src: puntatore **in_addr**
- dst: stringa di destinazione
- size: Come valore, si può usare **INET_ADDRSTRLEN**

Creazione ed inizializzazione di un socket

```
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main () {
    /* Creazione socket */
    int sd = socket(AF_INET, SOCK_STREAM, 0);

    /* Creazione indirizzo del socket */
    struct sockaddr_in indirizzo;

    memset(&indirizzo, 0, sizeof(indirizzo)); // Pulizia
    // Address family (IPv4)
    indirizzo.sin_family = AF_INET;
    // Port inizializzata a 4242, in network order
    indirizzo.sin_port = htons(4242);
    // IP address "192.168.4.5", convertito in numeric format
    inet_pton(AF_INET, "192.168.4.5", &indirizzo.sin_addr);

    // to be continued...
}
```

memset: puliamo la zona di memoria dove memorizzeremo l'indirizzo, buona programmazione.

Assegnare indirizzo a un socket (lato server)

Lo si fa attraverso la primitiva bind(): specifica **IP e porta dove il server riceve richieste di connessione**, ovvero il socket di ascolto.

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)

- sockfd: descrittore del socket
- addr è puntatore alla struttura creata prima (occorre un cast, in quanto noi abbiamo creato `sockaddr_in`)
- restituisce: 0 se ha successo, -1 se si verifica un errore

Mettere in ascolto un socket

Primitiva listen: il socket che stiamo costruendo è un socket passivo: ovvero un socket che resta in attesa di richieste e che non prende l'iniziativa.

Solo i socket SOCK_STREAM possono restare in attesa.

int listen(int sockfd, int backlog)

- sockfd: descrittore del socket
- backlog: numero max di richieste che possono essere pendenti

La primitiva restituisce 0 se ha successo, -1 se c'è un errore

Se il server serve una richiesta alla volta, è un server iterativo.

Programmazione distribuita su Server

Server deve fare nell'ordine queste cose

1. `socket()`: serve per creare un nuovo socket
2. `bind()`: serve per associare un indirizzo e una porta ad un socket
3. `listen()`: serve per mettere in ascolto il server su quel socket
4. `accept()`: serve per accettare una richiesta che è arrivata sul socket. **Ha senso solo sui socket socket stream**
5. `send()/recv()`: scambio di dati, non sullo stesso socket di ascolto
6. `close()`: chiudere la connessione

Stiamo parlando di **socket di tipo bloccante** (*di default sono bloccanti i socket*): significa che ci sono dei momenti in cui l'applicazione, sia il processo client che quello server si fermano.

La primitiva bloccante è la `accept` e non la `listen`!

La `listen`, è solo una syscall che permette di creare la coda di attesa e che permette di dire quel socket è in ascolto sulla porta x. NON BLOCCANTE.

```
int accept (int socfd, struct sockaddr *addr, socklen_t *addrlen)
```

- `socfd`: descrittore del socket in ascolto
- `sockaddr*`, ci viene salvato l'indirizzo del client (chi lo fa? L'`accept`)

Restituisce il descrittore di un NUOVO socket.

Il socket sul quale ascolta il server, non è quello su cui transitano i dati.

```
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main () {
    int ret, sd, new_sd, len;
    struct sockaddr_in my_addr, client_addr; // Strutture per gli IP
    /* Creazione socket */
    sd = socket(AF_INET, SOCK_STREAM, 0);

    /* Creazione indirizzo socket*/
    memset(&my_addr, 0, sizeof(my_addr)); // Pulizia
    my_addr.sin_family = AF_INET ;
    my_addr.sin_port = htons(4242);
    inet_pton(AF_INET, "192.168.4.5", &my_addr.sin_addr);
    // In alternativa: my_addr.sin_addr.s_addr = INADDR_ANY;

    ret = bind(sd, (struct sockaddr*)&my_addr, sizeof(my_addr));
    ret = listen(sd, 10);
    len = sizeof(client_addr);
    new_sd = accept(sd, (struct sockaddr*)&client_addr, &len);
    //...
}
```

Nota: andrebbero fatti i controlli sulla variabile `ret`.

Oltre a restituire -1, le primitive settano una variabile chiamata **`errno`**.

Funzione perror()

```
perror("Error: "); //automaticamente ci viene messo il valore di errno, fa solo una stampa
```

Programmazione distribuita su Client

Client deve fare nell'ordine queste cose

1. `socket()`: serve per creare nuovo socket
2. `connect()`: consente a un socket sul client di inviare a una richiesta di connessione a un socket remoto
3. `send()/receive()`
4. `close()`

Il cliente non deve fare bind:

non deve agganciare nessun indirizzo al socket: perché? **Non ha bisogno di farla ESPLICITAMENTE**, perché la fa il sistema operativo lo fa al posto suo, usando le **porte effimere** (le porte non riservate, sono quelle che durano poco).

Il client non deve fare listen:

non deve ascoltare, ha un ruolo attivo e non passivo.

`int connect(int sockfd, const struct sockaddr * addr, socklen_t addrlen)`

- `sockfd`: descrittore di client locale
- `addr`: indirizzo socket remoto

Restituisce un intero: 0 ok, -1 non è andata bene.

È una primitiva bloccante! (Se socket bloccante).

Si blocca in attesa della accept: si aggancia sul socket di comunicazione.

NOTA: Il client crea SOLO il socket di comunicazione! Il server ha il socket di ascolto e di comunicazione.

Scambio di dati

La comunicazione è sempre un stream di byte e deve essere trattata così.

I dati devono trasferiti da un buffer di applicazione a un buffer di sistema: il primo è creato nel codice dell'applicazione, ad esempio un array.

Il programmatore può solo sapere che da livello applicazione sta mandando dei dati verso il livello di trasporto che utilizza un protocollo affidabile e quindi i dati saranno recapitati ed in maniera corretta. Ma **non può dire il programmatore che chiamando la send e il messaggio viene mandato tutto in una volta**: NO! Perché?

Questo dipende dal buffer di trasmissione associato al socket. Se i dati non entrano nel buffer di trasmissione del socket (troppo piccolo o terminato lo spazio): **la chiamata a send restituisce il numero di byte che è riuscita a scrivere nel buffer di trasmissione del socket a livello del kernel**.

- Se questo valore è uguale alla dimensione del messaggio che io voglio spedire: allora messaggio interamente copiato, e sarà inviato interamente (prima o poi) in maniera corretto.
- Se restituisce invece un numero minore della dimensione del messaggio ad inviare, allora nel buffer di trasmissione del socket non c'era spazio a sufficienza e ne è stata copiata solo una parte: il resto? **Il resto ce la copia il PROGRAMMATORE. Come? Richiama la send! Ma gli passa solo la parte di messaggio ancora non scritta nel buffer del kernel.**

Stesso concetto vale anche per chi riceve, ma al contrario ovviamente: non si ha la certezza che con un'unica chiamata alla receive riesca a trasferire tutto il messaggio dal buffer di ricezione del socket al buffer dell'applicazione: perché magari non è arrivato tutto il messaggio (ad esempio).

Send e receive sono due primitive bloccanti.

Send()

Invia un messaggio attraverso un socket **CONNESSO**. Messaggio termine generico, come pacchetto, se non ci poniamo a un livello dello stack.

Ssize_t send (int sockfd, const void* buf, size_t len, int flags)

- Sockfd, descrittore del socket
- buf, puntatore al buffer dell'applicazione contenente il messaggio da inviare
- len
- flags

La funzione restituisce -1 in caso di errore, o il numero di byte copiati.

L'errore non è che non è riuscito a copiare: in quel caso restituisce 0.

Funzione bloccante se i socket sono bloccanti: *si blocca se non ha scritto tutto il messaggio*.

Recv()

Ssize_t recv (int sockfd, const void* buffer, size_t len, int flags)

- sockfd: descrittore del socket
- buf: puntatore al buffer in cui salvare il messaggio
- len: dimensione in byte del messaggio desiderato
- flags: per settare le opzioni

La funzione restituisce il numero di byte ricevuti, -1 su errore, 0 se il socket remoto si è chiuso (vedi più avanti).

La funzione si blocca finché non ha letto qualcosa: qualcosa cosa? **Un byte**. Minima quantità di informazione, essendo uno stream di byte.

Perché un byte?

Così piano piano svuoto il buffer (educazione), oppure magari parte dei 500 byte che devono arrivare, qualcosa può servire all'applicazione per preparare per quando arriva tutto.

Flag= MSG_WAITALL: si sblocca solo quando sono arrivati tutti byte e non uno.

Close()

Fatta da entrambi: client/server → serve per chiudere un socket.

int close(int fd)

Quando si chiude non può più essere usato: l'host remoto riceverà 0 nella recv: se sto facendo receive e ricevo 0 → chiudo il socket

Server Concorrenti

Se avessi solo server iterativi, saremmo limitati nella gestione delle richieste.

Server iterativo: *serve una richiesta alla volta* e per ogni richiesta accettata (accept()), il processo server la elabora e accoda le richieste che sopraggiungono.

Server concorrente: *serve più richieste alla volta* e per ogni richiesta accettata (accept()) il processo server crea un processo che la elabora (detto processo figlio.)

Il server padre si occupa del socket di ascolto, quando arriva una richiesta, la fa servire dal processo figlio che si occupa del socket di comunicazione.

fork()

pid_t fork()

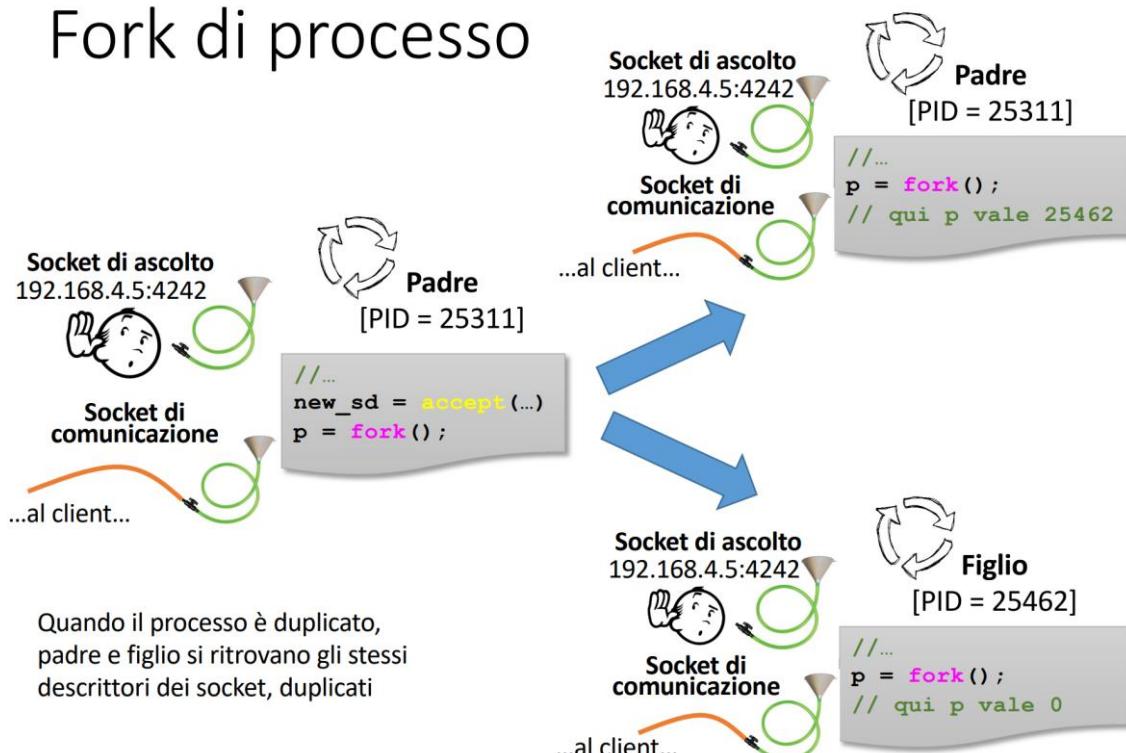
La system call restituisce il Process ID:

- 0 al processo figlio
- Del figlio al processo padre (-1 in caso di errore)

Essendo un nuovo processo, **il processo figlio possiederà uno spazio di indirizzamento privato, ma condividerà il codice con il padre.** Inoltre, oltre condividerne il codice, **eredita una copia delle aree dati globali, stack e heap.** Il processo figlio, quando andrà in esecuzione, riprenderà dall'istruzione successiva alla fork().

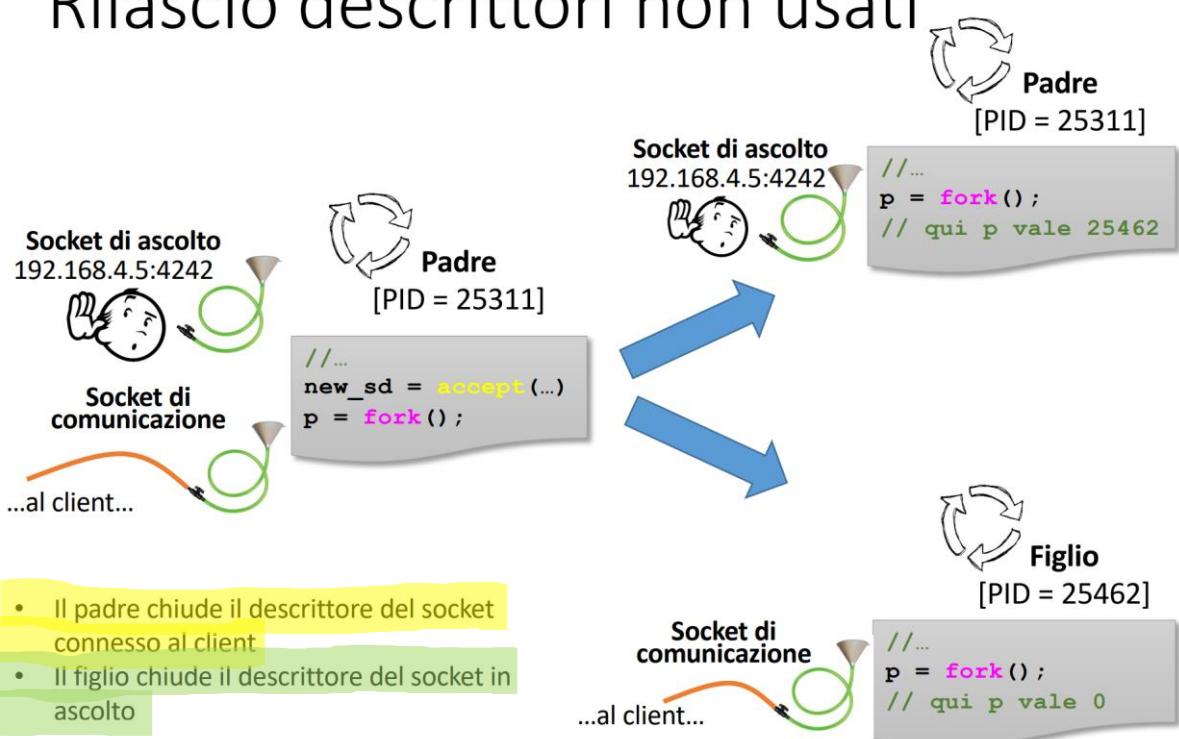
Situazione subito dopo la fork()

Fork di processo



Al figlio però non serve il socket di ascolto e al padre non serve quello di comunicazione.

Rilascio descrittori non usati



```
#include ...
int main () {
    int ret, sd, new_sd, len;
    struct sockaddr_in my_addr, cl_addr;
    //...
    pid_t pid;
    sd = socket(AF_INET, SOCK_STREAM, 0);
    /* Creazione indirizzo */
    ret = bind(sd, (struct sockaddr*)&my_addr, sizeof(my_addr));
    ret = listen(sd, 10);
    len = sizeof(cl_addr);
    while(1) {
        new_sd = accept(sd, (struct sockaddr*)&cl_addr, &len);
        pid = fork();
        if (pid == -1) /* Gestione errore */
        if (pid == 0) { /* Sono nel processo figlio */
            close(sd);
            /* Gestione richiesta (send, recv, ...) */
            close(new_sd);
            exit(0);
        }
        // Sono nel processo padre
        close(new_sd);
    }
}
```

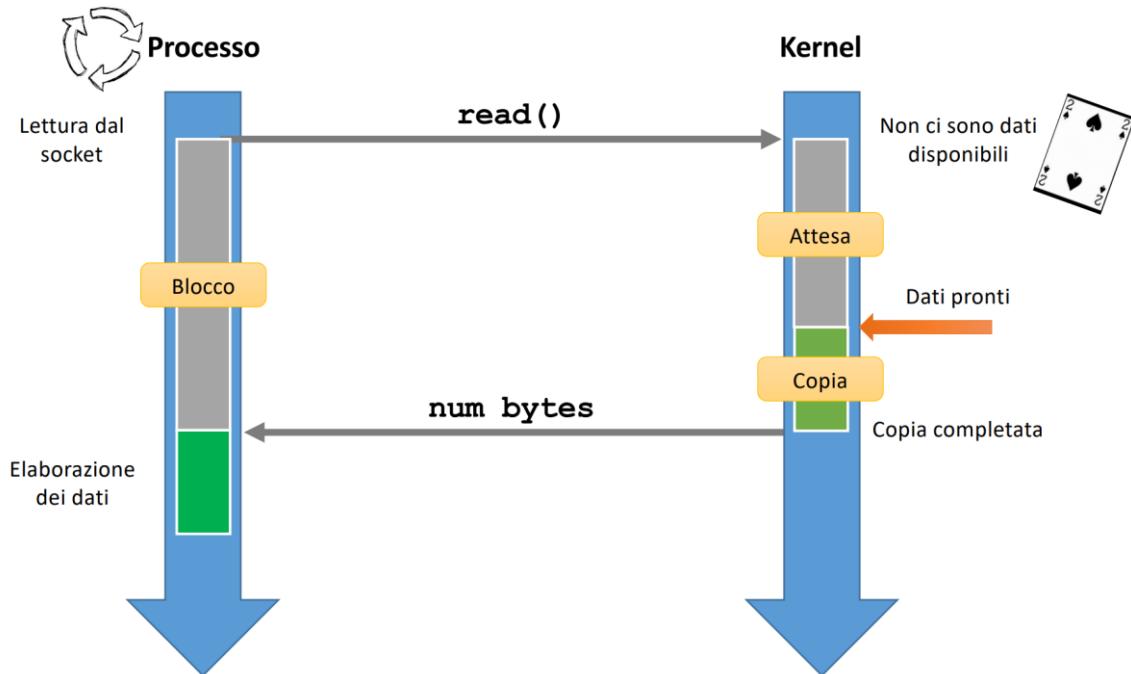
Il figlio, servita la richiesta può terminare.

Socket non bloccanti

I socket di default sono bloccanti, ma possono non esserlo.

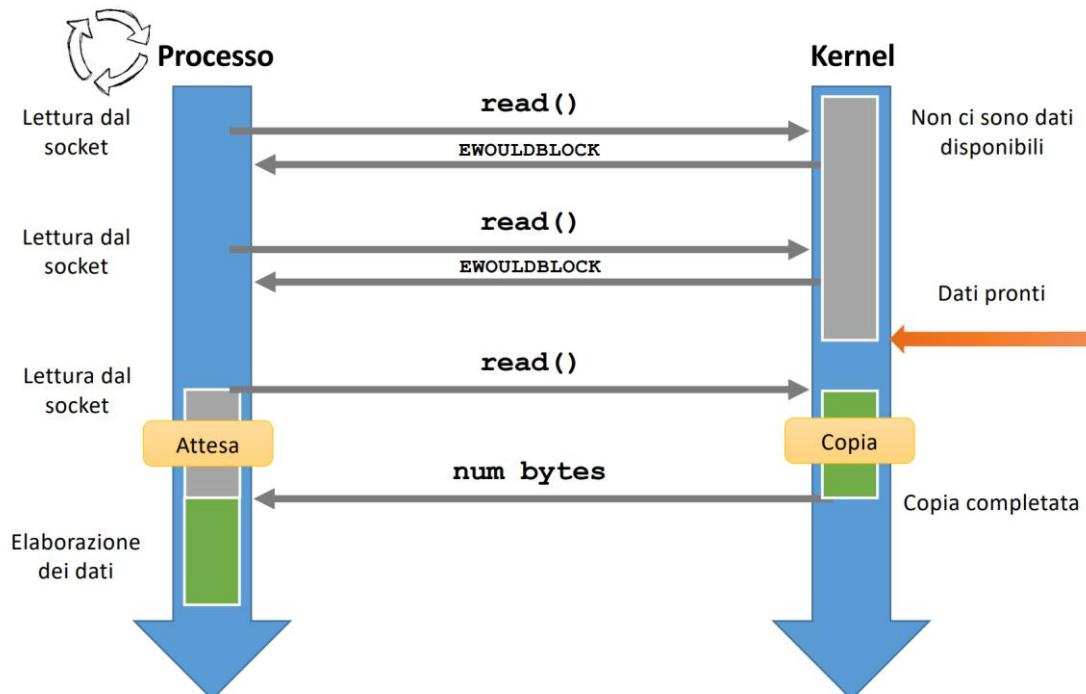
Socket bloccante

↓ timeline



La copia consiste in una copia dei dati nel buffer di sistema, restituiti i num bytes, il processo server poi copia quei dati nei buffer dell'applicazione.

Socket non bloccante



Per rendere un socket non bloccante:

```
socket(AF_INET, SOCK_STREAM | SOCK_NONBLOCK, 0);
```

La differenza, oltre che nei tempi di attesa, sta nel fatto che le primitive non sono più bloccanti:

- `accept()`, se non ci sono richieste, restituisce -1 e setta `errno` a `EWOULDBLOCK`
- `connect()`, se non ci si può connettere, restituisce -1 e setta `errno` a `EINPROGRESS`
- `send()`, se non può inviare tutto il messaggio, restituisce -1 e setta `errno` a `EWOULDBLOCK`
- `recv()`, se non ci sono messaggi, restituisce -1 e setta `errno` a `EWOULDBLOCK`

L'attesa è minore e si sposta sul processo, non nel kernel

IO multiplexing

Finora abbiamo ignorato la presenza di un problema: se faccio operazioni su un socket bloccante, non posso controllarne altri: necessitiamo dunque di controllare più descrittori/socket allo stesso tempo.

Attraverso la primitiva select si controllano più socket e si utilizza il primo pronto:

Un socket è pronto in lettura se:

- C'è almeno un byte da leggere
- Il socket è stato chiuso (read() restituirà 0)
- È un socket in ascolto, e ci sono connessioni effettuate
- C'è un errore (read() restituirà -1)

Un socket è pronto in scrittura se:

- C'è spazio nel buffer per scrivere
- C'è un errore (write() restituirà -1)
Se il socket è chiuso, `errno` vale EPIPE

Un insieme di descrittori (detto set) si rappresenta con una variabile di tipo `fd_set`

```
/* Aggiungere un descrittore "fd" all'insieme "set" */
void FD_SET(int fd, fd_set* set);

/* Controllare se un descrittore "fd" è nell'insieme "set" */
int FD_ISSET(int fd, fd_set* set);

/* Rimuovere un descrittore "fd" dall'insieme "set" */
void FD_CLR(int fd, fd_set* set);

/* Svuotare l'insieme "set" */
void FD_ZERO(fd_set* set);
```

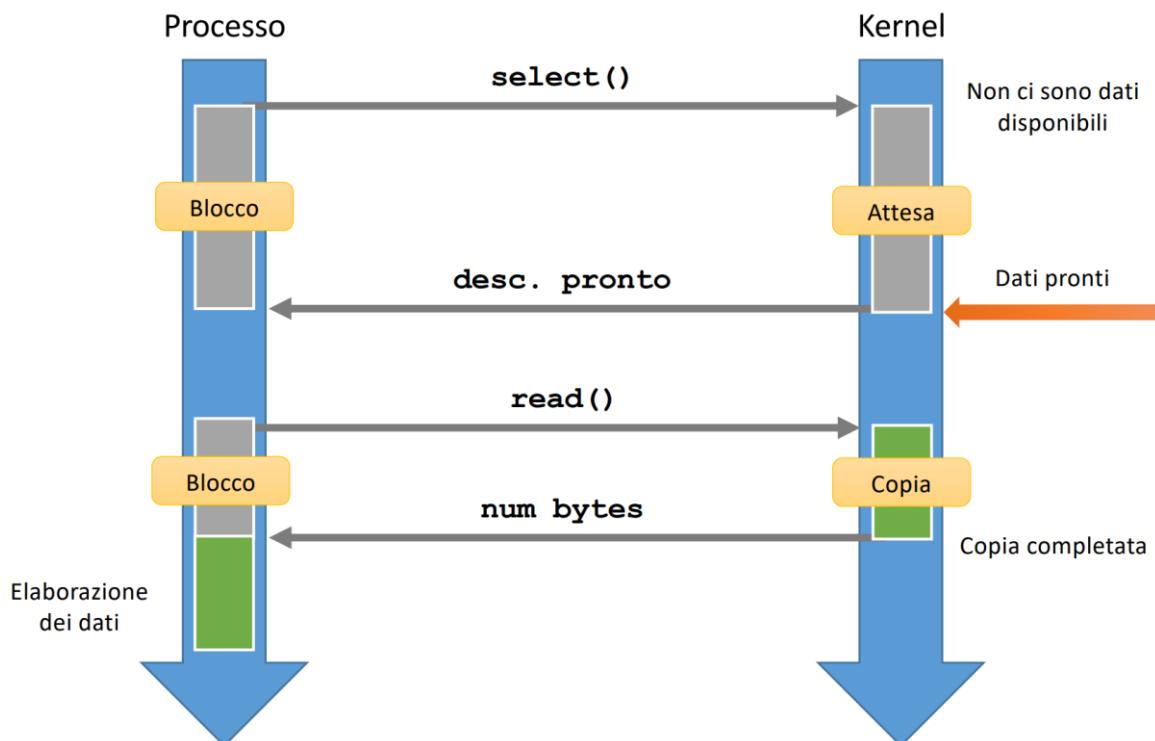
Select()

```
int select(int nfds, fd_set* readfds, fd_set* writefds,  
          fd_set* exceptfds, struct timeval* timeout);
```

- **nfds**: numero del descrittore più alto tra quelli da controllare, +1
- **readfds/writefds**: lista di descrittori da controllare per la lettura/scrittura
- **exceptfds**: lista di descrittori da controllare per le eccezioni (non ci interessa)
- **timeout**: intervallo di timeout:
 - NULL → Attesa indefinita, fino a quando un descrittore è pronto
 - timeout = { 10; 5; } → Attesa massima di 10 secondi e 5 microsecondi
 - timeout = { 0; 0; } Attesa nulla, controlla i descrittori ed esce immediatamente
- **Restituisce il numero di descrittori pronti** (-1 in caso di errore)

È bloccante: si blocca finché uno dei descrittori controllati non è pronto, o finché non scade il timeout (in questo caso, restituisce 0, cioè 0 descrittori pronti).

La select modifica i set che controlla: elimina dal set i socket non pronti! Quindi la select va fatta su una copia del set da controllare.



Quando scade il timeout ritorna 0: ovvero 0 descrittori pronti.

Qui è sempre il processo padre che fa tutto, ma a “divisione di tempo”, non è che lo fa fare al figlio.

Esempio:

```
int main(int argc, char* argv[]){
    fd_set master;                      /* Set principale gestito dal
                                            programmatore con le macro */

    fd_set read_fds;                    /* Set di lettura gestito dalla
                                            select */

    int fdmax;                          // Numero max di descrittori

    struct sockaddr_in sv_addr;        // Indirizzo server
    struct sockaddr_in cl_addr;        // Indirizzo client
    int listener;                      // Socket per l'ascolto
    int newfd;                         // Socket di comunicazione
    char buf[1024];                   // Buffer di applicazione
    int nbytes;
    int addrlen;
    int i;
    /* Azzero i set */
    FD_ZERO(&master);
    FD_ZERO(&read_fds);

    listener = socket(AF_INET, SOCK_STREAM, 0);
    sv_addr.sin_family = AF_INET;
    // INADDR_ANY mette il server in ascolto su tutte le
    // interfacce (indirizzi IP) disponibili sul server
    sv_addr.sin_addr.s_addr = INADDR_ANY;
    sv_addr.sin_port = htons(20000);

    bind(listener, (struct sockaddr*)& sv_addr, sizeof(sv_addr));
    listen(listener, 10);
    // Aggiungo il listener al set dei socket monitorati
    FD_SET(listener, &master);
    // Tengo traccia del maggiore (ora è il listener)
    fdmax = listener;
    for(;;){
        read_fds = master;      // read_fds sarà modificato dalla select
        select(fdmax + 1, &read_fds, NULL, NULL, NULL);
        for(i=0; i<=fdmax; i++) {          // f1) Scorro il set
            if(FD_ISSET(i, &read_fds)) { // i1) Trovato desc. pronto
                if(i == listener) {      // i2) È il listener
                    addrlen = sizeof(cl_addr);
                    newfd = accept(listener,
                        (struct sockaddr *)&cl_addr, &addrlen)
                    FD_SET(newfd, &master); // Aggiungo il nuovo socket
                    if(newfd > fdmax){ fdmax = newfd; } // Aggiorno fdmax
                }
                else { // Il socket connesso è pronto
                    nbytes = recv(i, buf, sizeof(buf));
                    //... Uso i dati
                    // Chiudo il socket connesso
                    close(i);
                    // Tolgo il descrittore del socket connesso dal
                    // set dei monitorati
                    FD_CLR(i, &master);
                }
            } // Fine if i1
        } // Fine for f1
    } // Fine for(;;
    return 0;
}
```

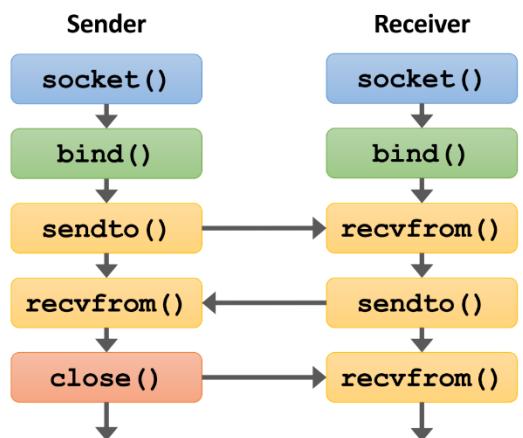
Socket UDP

È connectionless non usa operazioni per creare una connessione:
nessuna operazione preliminare per creare un canale virtuale.
UDP è inaffidabile.

UDP usa le primitive **sendto** e **recvfrom** le quali ogni volta necessitano di sapere da/a chi si sta comunicando.

Sendto()

Cosa vuol dire inviare qualcosa sul socket? Significa copiare le cose da mandare nel **buffer del kernel**.



ssize_t sendto(int sockfd, const void* buf, size_t len, int flags, const struct sockaddr* dest_addr, socklen_t addrlen);

- **Sockfd:** il descrittore del socket
- **Buf:** il buffer applicativo
- **Len:** la lunghezza del buffer
- **Flags:** opzioni
- **Dest_addr:** puntatore alla struttura con indirizzo destinatario
- **Addrlen:** lunghezza ind destinatario
- **RESTITUISCE IL NUMERO DI BYTE INVIATI**

È bloccante!

Recvfrom()

ssize_t recvfrom(int sockfd, const void* buf, size_t len, int flags, struct sockaddr* src_addr, socklen_t addrlen);

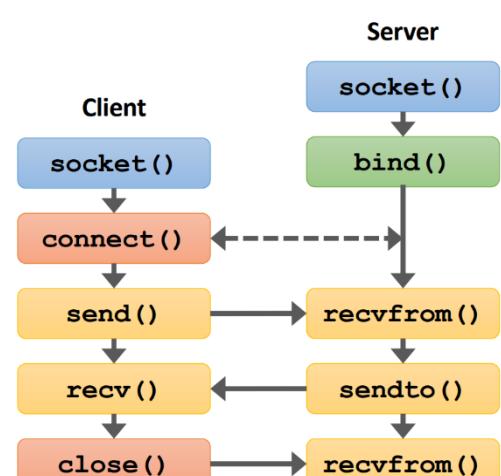
- **Sockfd:** il descrittore del socket
- **Buf:** il buffer applicativo
- **Len:** la lunghezza del buffer
- **Flags:** opzioni
- **src_addr:** puntatore alla struttura con indirizzo sorgente
- **Addrlen:** lunghezza ind sorgente
- **RESTITUISCE IL NUMERO DI BYTE INVIATI**

È bloccante! Si Blocca finché non ha letto un byte!

Socket UDP “connesso”

Un client può usare connect() per associare il socket UDP a un indirizzo remoto (quello del server): invierà (riceverà) pacchetti solo da/a quell’indirizzo così da non dover specificare l’indirizzo del destinatario o (mittente) ogni volta.

Il CLIENT può quindi usare recv e send. Non è presente una connessione, è solo una comodità: il protocollo rimane UDP.



Protocolli Text e Binary

Protocolli text

Vantaggi dei protocolli text:

- Semplici

Svantaggi:

- *overhead* della **codifica** e decodifica;
- *sniifabili*, se osservato, si può capire il pattern del protocollo;
- **possono occupare più spazio rispetto alla codifica binary**: ad esempio un intero su 5 cifre, binario bastano 4 byte, ASCII necessita di 5 byte.

Server

```
char buffer[1024];

// Dichiarazione
struct temp{
    int a;
    char b;
};

// Istanziazione
struct temp t;

// Conversione a stringa
sprintf(buffer, "%d %c", t.a, t.b);
```

Client

```
// Istanziazione
struct temp t;

// Ricezione
...

// Parsing e memorizzazione nei campi
sscanf(buffer, "%d %c", &t.a, &t.b);
```

Protocolli binary

Vantaggi:

- *i dati in codifica ASCII possono occupare più spazio rispetto alla codifica Binary*
- *difficilmente sniffabili, se non conosco la struttura* è difficile rimettere insieme i pezzi

Problemi:

- *conoscenza della struttura dei dati* che si stanno manipolando
- serializzazione e deserializzazione

```
struct temp{
    uint32_t a;
    uint8_t b;
};
struct temp t;
...
// Convertire in network order prima dell'invio
t.a = htonl(t.a);

// Spedire i campi sul socket 'new_sd'
ret = send(new_sd, (void*)&t.a, sizeof(uint32_t), 0);
...
ret = send(new_sd, (void*)&t.b, sizeof(uint8_t), 0);
```

```
struct temp t;
...
ret = recv(new_sd, (void*)&t.a, sizeof(uint32_t), 0);
if (ret < sizeof(uint32_t)){
    // Gestione errore
}
// Convertire in host order il campo 'a'
t.a = ntohl(t.a);

ret = recv(new_sd, (void*)&t.b, sizeof(uint8_t), 0);
if (ret < sizeof(uint8_t)){
    // Gestione errore
}
```

Nei protocollo binari bisogna essere sicuri che *i dati che si vogliono comunicare occupino lo stesso spazio: si usano i tipi opachi.*

Si definiscono messaggi con una struttura ben precisa, con campi che rappresentano l'informazione da scambiare.

Se qualora il protocollo possa prevedere una qualche campo a lunghezza variabile, va indicata la lunghezza max.

Server Apache

È un server web integrato all'interno del sistema operativo.

Su Debian 8 versione 2.4: **implementato dal programma apache2**. Non si può invocare direttamente: va manipolato con dei comandi particolari.

CI VOGLIONO I PRIVILEGI DI ROOT: perché? Il privilegio di root viene immediatamente rilasciato una volta che il server web va in esecuzione

Configurazione

apache2ctl (ctl sta per control)

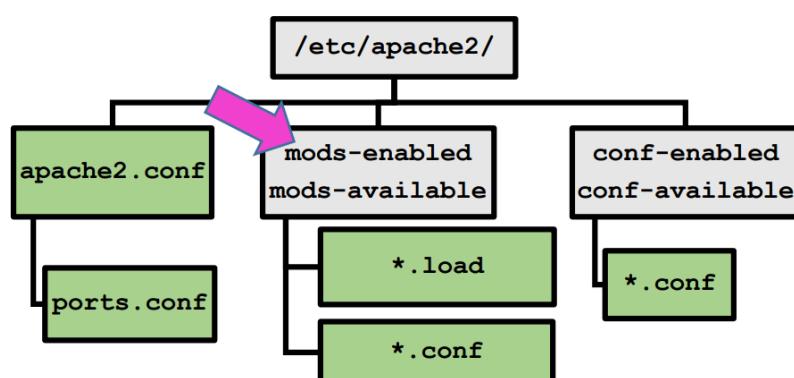
ci permette di fare diverse cose, si da un comando dopo apache2ctl

- **apache2ctl start**
- **apache2ctl restart**
- **apache2ctl status**
- **apache2ctl configtest**

Altrimenti si può fare: **service apache2 <comando>** → service è “declinabile”, utilizzabile in diversi contesti

Quando in esecuzione, si può aprire un browser e accedere a <http://localhost/> la pagina mostrata è /var/www/html/index.html

Si possono accettare più richieste contemporaneamente e si può impostare il modello I/O da usare.



/etc/apache2/

Il file principale di config è apache2.conf

è un file modulare: contiene delle direttive e delle direttive contenitore (cioè nomi di proprietà e un valore, le direttive contenitore contengono più impostazioni contemporaneamente).

Il file di configurazione può usare **anche direttiva Includ per incoppare parti di configurazione**.

Ports.conf è una parte di configurazione che viene importata attraverso la direttiva includ. Le includ rendono il file di configurazion altamente modulare.

/etc/apache2/conf-available/

Una sorta di recipiente per tutte le parti di configurazioni che si possono includere in apache2.conf. **Essere disponibili non significa che siano incluse**.

I file vengono abilitati con **a2enconf <nome_file>** (apache2 enable configuration).

Perché per poterla includere, oltre ad essere available, una direttiva deve essere enbled.

Quel nome di file è una delle conf available.

Viene creato un soft link nella diretctory conf-enabled.

All'avvio, tutte le config abilitate vengono automaticamente incluse.

Un file si disabilita con **a2disconf <nomefile>**

Ogni volta che si modifica il file di configurazione di apache per rendere le modifiche effettive, bisogna riavviare il server.

/etc/apache2/mods-available/ e /etc/apache2/mods-enabled/

I moduli sono parti di configurazione complesse.

Per abilitare un modulo: **a2enmod** e per disabilitare **a2dismod**

Direttive globali

A differenza di quelle locali che sono valide su un singolo virtual host, le direttive globali hanno effetto su tutto il server e sono:

- ServerRoot
- KeepAlive e KeepAliveTimeout
- ErrorLog
- Listen

ServerRoot /etc/apache2

Specifica la directory principale del file di configurazione. Su Debian, la direttiva server root è settata automaticamente all'avvio del servizio **apache2ctl**.

Infatti nel file apache2.conf la direttiva è commentata,

KeepAlive on, specifica se si vogliono o meno le connessioni persistenti

KeepAliveTimeout 5 quanto bisogna aspettare una seconda richiesta sulla stessa connessione prima di chiuderla

Il valore non deve essere troppo alto (potrebbero bloccare inutilmente un processo che serve un client lento) o troppo basso: risorse e connessioni da rifare

KeepAliveTimeout non è obbligatoria, c'è un valore di default.

Listen numeroDiPorta

Il processo può ascoltare su più porte scrivendo più volte la direttiva, ad esempio Listen 80 Listen 8080.

è obbligatoria.

La direttiva è presente [/etc/apache2/ports.conf](#) ed è incluso automaticamente da apache2.conf.

ErrorLog path

Direttiva che ci dice dove vanno i finire i log che vengono generati.

Si può configurare la forma di come vengono scritti i log.

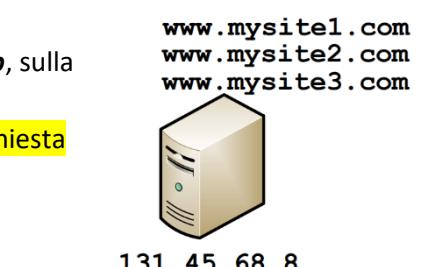
Direttivi per i siti web e virtual host

Host singolo: nel caso più semplice un server Web è in esecuzione su una macchina con un indirizzo IP, e ospita un solo sito Web, ciò porta ad un'elevata richiesta di risorse (per ogni sito, occorre un server Web...).



Con il **Virtual host**: si possono **configurare più siti sullo stesso server Web**, sulla stessa macchina, con lo stesso indirizzo IP.

Il server discrimina le richieste dei client in base al campo 'Host' della richiesta http.



Come accade per parti di configurazione e moduli, i virtual host sono in una directory dedicata:

/etc/apache2/sites-available

Non fruibili: available ma non enabled!

Per abilitare i siti: **a2ensite nome_file**

Per disabilitare i siti: **a2dissite nome_file**

Anche in questo caso, bisogna riavviare il server per ricaricare la configurazione.

Quanti virtual host ci posso essere? Tante

L'unico virtual host abilitato di default è uno in 000-default.conf.

Nel file di config, ci sarà:

```
<VirtualHost *:80>
    ServerName www.example.com
    ...
    DocumentRoot /var/www/html
</VirtualHost>
```

VirtualHost: direttiva contenitore che necessita di IP:porta.

ServerName è il nome simbolico del sito.

DocumentRoot è la directory del file del sito (dove sono contenuti css, JS, ecc): **attenzione** alla differenza alla differenza tra ServerRoot e DocumentRoot

Multi-Processing Modules

Apache accetta e serve più richieste contemporaneamente e lo fa tramite moduli **Multi-Processing Module** (MPM) occupandosi della:

- gestione dei socket;
- binding delle porte;
- processazione delle richieste usando processi figli e thread

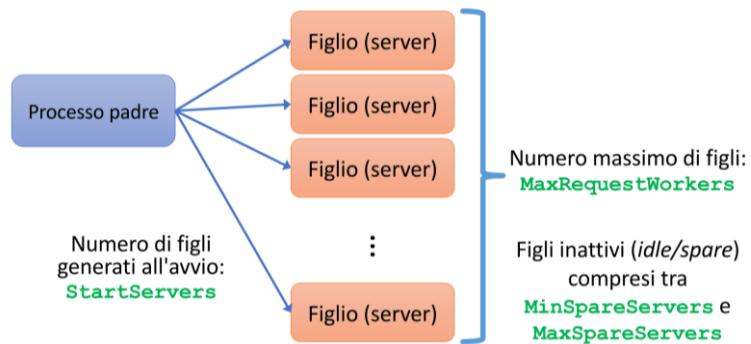
Su Unix, si può scegliere tra gli MPM: prefork, worker ed event.

Prefork

Il modulo prefork implementa un server multi-processo, **senza thread**.

La prefork *viene fatta all'inizio: il server si prepara sulla base delle conoscenze del carico delle richieste che riceverà*: istanzia un pool di figli che saranno già disponibili per gestire le richieste, così da non dover fare la fork nel momento in cui arriva la richiesta. *I figli stanno in ascolto, accettano le connessioni e le servono (detti worker).*

Quando hanno servito una connessione, rimangono disponibili e sono allocabili per altre richieste.



Ogni figlio viene riciclato per `MaxConnectionsPerChild` connessioni poi viene terminato, **per evitare memory leak accidentali**.

Vantaggi:

- molto **compatibile**, alcuni moduli Apache non supportano i Thread.
- Molto **stabile**: *se un processo crasha interrompe una sola connessione*.

Svantaggi:

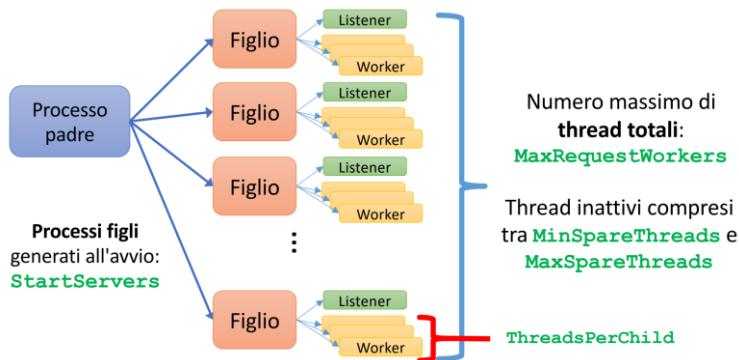
- *occupazione di memoria*
- *Complessità del tuning* (regolare bene le impostazioni): troppi processi inattivi occupano inutilmente in memoria; troppi pochi causano più overhead da fork().

Worker

Implementa un server multi processo e multi thread.

Il processo padre **genera un pool di figli iniziali**: però, ogni processo figlio genera:

- **Un thread listener** che accetta/smista le connessioni
- **Pool di thread worker** che servono le richieste (il listener smista a loro le connessioni che arrivano)



Ogni processo figlio viene riciclato per `MaxConnectionsPerChild` connessioni.

Il numero massimo di figli è necessariamente **`MaxRequestWorkers/ThreadsPerChild`**.

Vantaggio: overhead ridotto grazie al preforkin e risparmio di memoria grazie ai thread.

Event

È una versione migliorata del worker: oltre ad accettare le connessioni, ***il listener gestisce le connessioni temporaneamente inattive.***

Questo perché con worker, un thread, non si rende conto se una connessione sta tardando quindi le connessioni lente o non responsive vanno a bloccare risorse che potrebbero essere usate per soddisfare altre richieste.

L'idea chiave di event è l'evento: quando si verifica l'evento di cattivo funzionamento, rilevarlo e metterlo in pratica un giusto trattamento.

In particolare, ad esempio, si revoca le risorse alle connessioni divenute non responding.

Aumenta il numero di connessioni servibili contemporaneamente a parità di numero di thread, eliminando i tempi morti.