

RETI DI CALCOLATORI

Prof. Roberto Canonico – A.A. 2024/25

Luca Maria Incarnato

INDICE DEGLI ARGOMENTI

PROTOCOLLI APPLICATIVI

1. MODELLI DI SWITCHING E I TIPI DI NETWORK (p. 3)
2. INTERNET (p. 8)
3. IL MODELLO ISO-OSI E I PROTOCOLLI (p. 10)
4. IL PROTOCOLLO HTTP (p. 17)
5. IL SISTEMA DNS (p. 27)
6. PROTOCOLLI APPLICATIVI: FTP E SMTP (p. 33)
7. CONTENT DELIVERY NETWORKS (p. 40)
8. APPLICAZIONI PEER TO PEER (p. 43)

IL LIVELLO RETE E I PROTOCOLLI AUSILIARI

9. IL LIVELLO RETE E IL PROTOCOLLO IP (p. 50)
10. L'ASSEGNAZIONE DEGLI INDIRIZZI IP (p. 58)
11. I ROUTER (p. 64)
12. NETWORK ADDRESS TRANSLATION (p. 68)
13. GLI INDIRIZZI MAC E I PROTOCOLLI ARP, RARP E DHCP (p. 71)
14. ICMP: PING E TRACEROUTE (p. 77)
15. INTRODUZIONE AL ROUTING (p. 78)
16. ROUTING DISTANCE VECTOR (p. 81)
17. ROUTING LINK-STATE E L'ALGORITMO DI DIJKSTRA (p. 85)
18. PROTOCOLLI DI ROUTING (p. 89)
19. IL ROUTING GERARCHICO E GLI AUTONOMOUS SYSTEMS (p. 94)
20. BROADCAST E MULTICAST (p. 99)

IL LIVELLO TRASPORTO

21. FUNZIONI DEL LIVELLO TRASPORTO (p. 108)
22. IL PROTOCOLLO UDP (p. 112)
23. TRASMISSIONE DI FLUSSI MULTIMEDIALI IN INTERNET (p. 113)
24. RTP/RTCP e DASH (p. 115)
25. TECNICHE DI TRASMISSIONE AFFIDABILE DEI DATI (p. 118)
26. IL PROTOCOLLO TCP (p. 124)
27. CONTROLLO DI FLUSSO E DI CONGESTIONE IN TCP (p. 135)

IL LIVELLO DATA LINK

28. IL LIVELLO DATA LINK E I PROTOCOLLI AD ACCESSO MULTIPLO (p. 144)
29. RETI LOCALI ETHERNET (p. 152)
30. INTERCONNESSIONE DI LAN: HUB E BRIDGES (p. 157)
31. RETI VLAN E WLAN (p. 161)

APPLICAZIONI CLIENT-SERVER E LA SICUREZZA IN RETE

32. INTRODUZIONE ALLA COMUNICAZIONE CLIENT SERVER CON API (p. 169)
33. COMUNICAZIONE CLIENT-SERVER CON TCP E UDP IN PYTHON (p. 172)
34. LE TECNICHE CRITTOGRAFICHE (p. 178)
35. INTEGRITÀ DEI MESSAGGI E PROTOCOLLI DI AUTENTICAZIONE (p. 184)

PROTOCOLLI APPLICATIVI

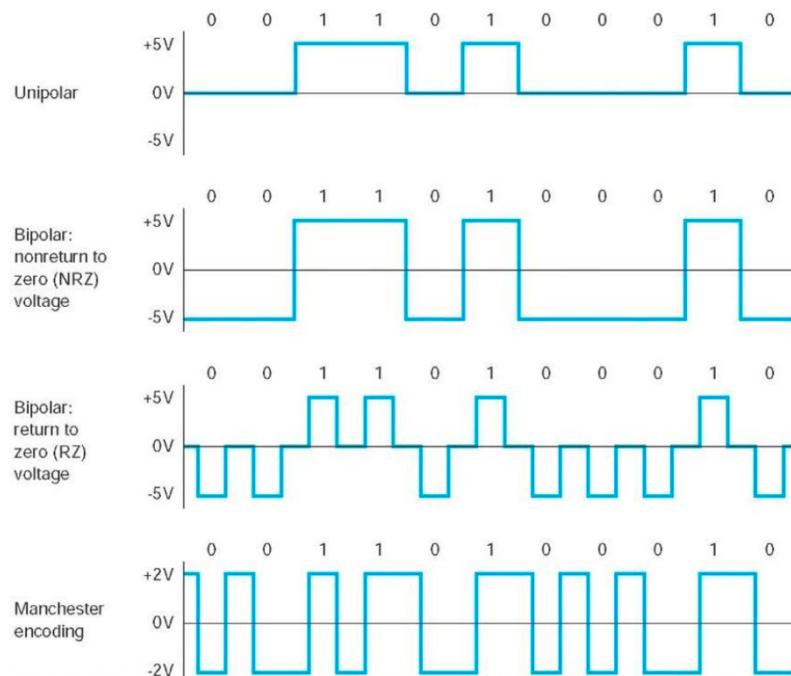
MODELLO DI SWITCHING E I TIPI DI NETWORK

Per rete di calcolatori (o computer network in inglese) si intende un insieme di dispositivi di elaborazione che sono connessi (in svariati modi) con lo scopo di comunicare e condividere risorse; le infrastrutture di questo tipo sono eterogenee, essendo composte da diversi tipi di terminali con diversi mezzi di trasmissione, diverse tecnologie di comunicazione, diversi possessori di server e diversi servizi messi a disposizione.

In ottica futura, è necessario distinguere:

- **Terminali** (o host o end-system), ovvero PC, server, periferiche singole, smartphones, sensori, ecc...
- **Dispositivi intermedi**, ovvero attori posti nel mezzo del processo di comunicazione (a differenza dei terminali, che sono dispositivi di frontiera) e assumono nomi diversi sulla base del loro ruolo (hub, switch, router, modem, access point, ecc...)
- **Connessioni** (o links), ovvero cavi fisici e mezzi di comunicazione wireless che permettono il passaggio di informazioni tra i vari dispositivi.

Tutti questi sono **dispositivi fisici**, non si è ancora menzionato alcun protocollo o software ma solo gli **agenti materiali che permettono la comunicazione terminale**. Per tecnica di trasmissione digitale si intende il processo che permette la trasmissione e la ricezione di sequenze di simboli binari (bits) su un canale di comunicazione; esistono diverse tecniche di modulazione che permettono di trasmettere un simbolo binario tramite l'associazione del valore ad un livello di un segnale o di una sua variazione:



Una qualsiasi **trasmissione digitale** è caratterizzata da un **bit rate** (o **data rate**) che determina il **numero di bits che possono essere trasmessi in un'unità di tempo** (solitamente un **secondo**); di conseguenza, per determinare il tempo **T** necessario per inviare una quantità di bit **L** ad un **bit rate R** si può determinare con la seguente relazione:

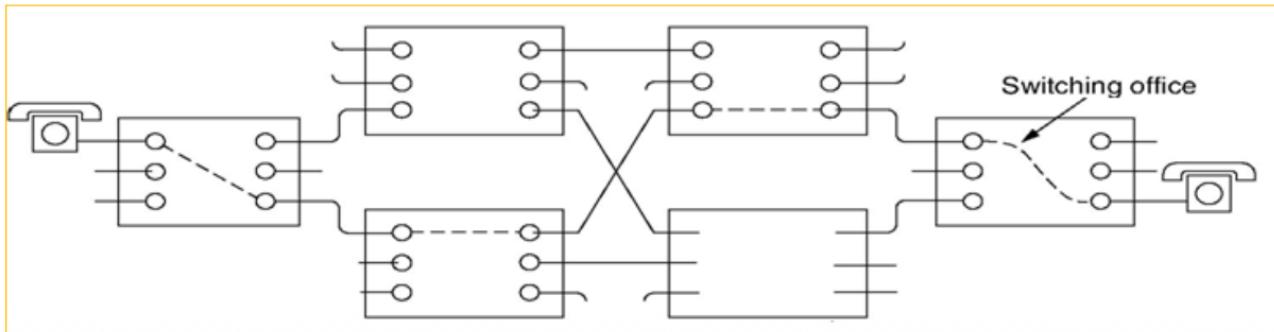
$$T = \frac{L}{R} = \left\lceil \frac{\text{bit}}{\frac{\text{bit}}{s}} \right\rceil = [s]$$

I link connettono sistemi terminali e dispositivi intermedi e possono essere distinti, come già parzialmente evidenziato, in due categorie:

- **Connessioni via cavo (wired links)**, coinvolgono la trasmissione di segnali digitali tramite un medium fisico (twisted pairs, cavi coassiali, fibra ottica, ecc...);
- **Connessioni senza cavo (wireless links)**, coinvolgono la trasmissione di segnali digitali modulando onde radio (appartenenti allo spettro infrarosso) propagate nell'aria e, a loro volta, si dividono in:
 - **Spettro sotto licenza**, la trasmissione è permessa solo ad entità autorizzate;
 - **Spettro libero da licenza**, la trasmissione è permessa anche senza licenza ma sotto limitazioni di potenza.

Più del 95% delle connessioni mondiali è via cavo.

La rete di computer opera secondo il **modello a commutazione di pacchetto (packet switching)**, a differenza del **sistema telefonico** (considerato un suo antenato) che opera secondo il **modello a commutazione di circuito (circuit switching, o PSTN)**. Nel modello **PSTN** i **terminali** (i telefoni) sono connessi tramite una serie di uffici di smistamento; quando viene effettuata una chiamata, un circuito è fisicamente stabilito tra i due telefoni tramite una serie di collegamenti circuituali lungo uno predeterminato tra uffici di smistamento:



Secondo questo modello, **un circuito è dedicato ad una singola telefonata** e, pertanto, **la sua capacità di trasmissione è assegnata ad una chiamata anche quando nessuno dei due interlocutori parla**. Questo **sistema di connessione** include **tre fasi**:

1. **Determinazione del circuito** (è determinato il percorso, i vari link necessari sono stabiliti e le risorse allocate);
2. **Chiamata** (o il trasferimento dati);
3. **Tear down** (i link utilizzati sono scollegati e le risorse deallocate).

La prima e la terza fase coinvolgono lo scambio di informazioni sia tra terminale e switching office che tra due switching offices.

Il **modello PSTN** prevede che gli uffici di smistamento siano gerarchicamente organizzati: i collegamenti che connettono gli uffici devono poter trasmettere più chiamate allo stesso tempo e, pertanto, la loro capacità dovrà essere divisa in diversi canali per gestire adeguatamente il traffico aggregato. Nella pratica, questa divisione veniva fatta utilizzando diverse tecniche di

multiplexing, come quella a **base temporale** (TDM, Time Division Multiplexing) o a **base frequenziale** (FDM, Frequency Division Multiplexing): entrambe **partizionano la capacità di collegamento in canale di dimensione predeterminata**, in modo che le singole chiamate possano essere trasmesse ad un bit rate di **64kb/s** e assegnandogli un canale in fase di determinazione del circuito.

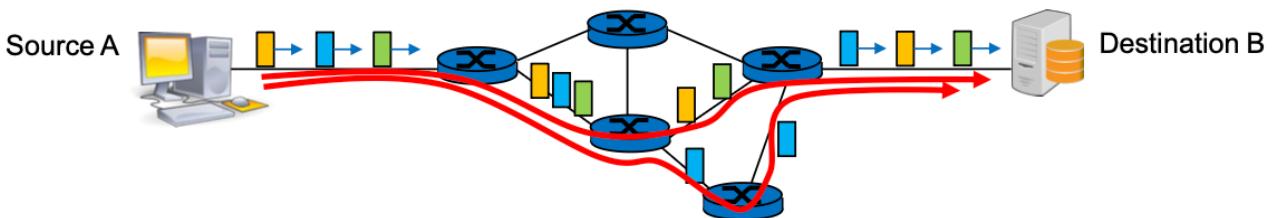
Le reti di computer lavorano, invece, con il **modello a commutazione di pacchetto**, ideata negli anni 60 da Paul Baran. L'informazione è trasmessa in **pacchetti** (unità di trasmissione), **formati da un header**, che contiene **informazioni di controllo** (inclusi l'indirizzo del destinatario), e **un payload** (detto anche **parte utile**). La divisione tra **header e payload**, così come tra più pacchetti, può essere effettuata o dichiarando a priori la lunghezza o usando una sequenza di caratteri speciali che ne demarca la fine; quest'ultima opzione, però, sprecherebbe una quantità di bit non trascurabile, rendendo più conveniente la specifica della dimensione nell'header.

I sistemi intermedi, in questo modello, operano tipicamente secondo un etica **store-and-forward**: ogni pacchetto è ricevuto nella propria integrità, ispezionato per eventuali errori, e ritrasmesso lungo il percorso di destinazione, con tanto di **buffering** e di accodamento del pacchetto nel caso in cui i pacchetti arrivino più velocemente di quanto possono essere processati; in questo modo, ogni canale è occupato solo durante la trasmissione di un pacchetto, dopodiché è disponibile per il traffico esterno. In particolare, l'**host** prende il messaggio dal terminale e lo divide in pacchetti di **L bits di lunghezza**, trasmessi in rete tramite un **transmission rate R** (detto anche link bandwidth, o banda del collegamento), che produce un **delay D** quantificabile dalla relazione:

$$D = \frac{L}{R} = \left[\frac{\text{bit}}{\frac{\text{bit}}{\text{s}}} \right] = [\text{s}]$$

Il packet switching ha due possibili implementazioni: in **rete a datagrammi** e in **Virtual Circuit Networks**.

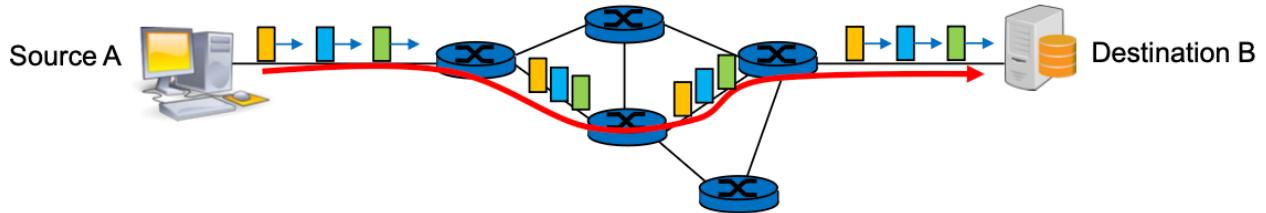
In una rete a **datagrammi** ogni pacchetto è indipendentemente indirizzato verso la propria destinazione, senza seguire percorsi prestabiliti: ogni volta che un pacchetto raggiunge un device **intermedio** lungo gli strati della rete, viene deciso quale sarà il prossimo device a cui il pacchetto verrà trasmesso. Ciò significa che **pacchetti contigui**, inviati da una stessa sorgente A e diretti verso una stessa destinazione B, **possono non solo essere diretti verso percorsi diversi ma anche arrivare in un ordine diverso da quello iniziale**. Sebbene questa implementazione **non renda necessario alcun setup di connessione** (per una **complessità più bassa**), **non impedisce ad un pacchetto di perdere lungo il percorso da A a B**.



In un **Virtual Circuit Network** un percorso da A a B viene predeterminato e bloccato in modo che i pacchetti abbiano già una rotta da seguire e che arrivino a destinazione nello stesso ordine di invio (cosa che può essere implementata con opportuno multiplexing, tipo TDM - Time Division Multiplexing, per il quale ogni slot di un intervallo di tempo è assegnato in maniera fissa ad un certo flusso di pacchetti); tuttavia, a differenza dell'implementazione precedente, è necessaria una fase di

setup preliminare (signalling) e delle risorse possono potenzialmente essere messe da parte per lo stream A→B in ogni device intermedio.

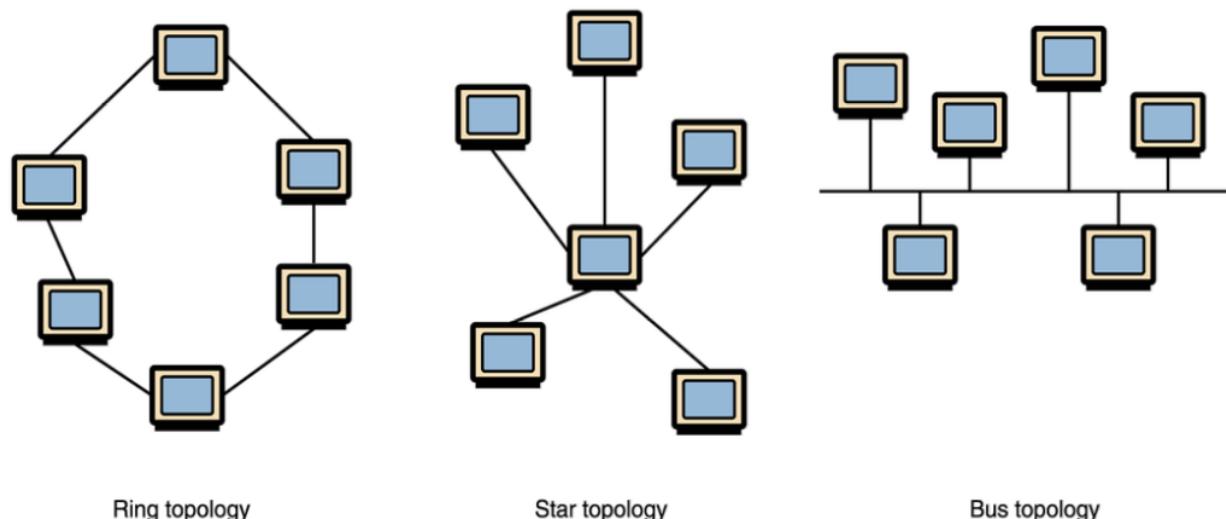
Si noti una certa somiglianza con il modello a commutazione di circuito, nonostante si stia sempre parlando di commutazione di pacchetto.



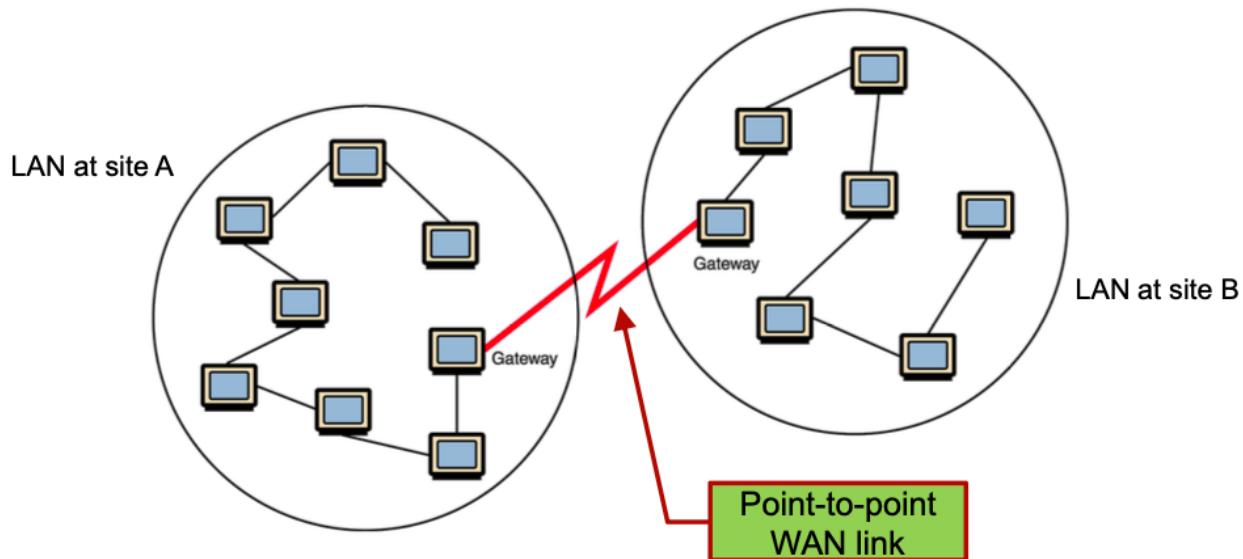
I tipi di reti si distinguono sulla base della loro estensione geografica:

- **Local-Area Network (LAN)**, connette un numero relativamente piccolo di terminali in un'area geografica relativamente limitata;
- **Wide-Area Network (WAN)**, connette due o più LAN a distanze geografiche relativamente larghe;
- **Metropolitan-Area Network (MAN)**, per connettere infrastrutture lungo larghe città.

L'Internet, così come è conosciuto oggi, è essenzialmente la più grande WAN esistente, estesa in tutto il globo terrestre. Le reti WAN sono realizzate, tipicamente, interconnettendo diverse reti LAN (la pratica di connessione di più reti è detta internetworking).

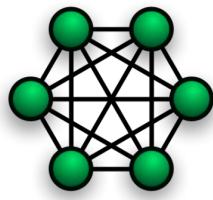


Quando due o più LAN, in due siti diversi, sono interconnesse, un particolare nodo ad ogni rete è impostato per servire come gateway alla gestione di tutte le comunicazioni tra la LAN e il resto della rete esterna; nell'Internet, i gateway sono sempre riferiti come routers.

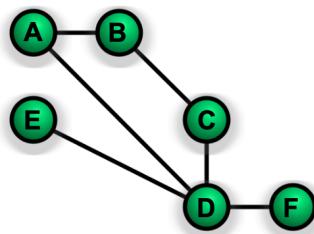


Si consideri un **internetwork** di N siti in cui ognuno è connesso a $N-1$ altri siti (full mesh topology o topologia magliata); il numero di link bidirezionali è:

$$\frac{N \cdot (N - 1)}{2}$$



Internetwork di grandi scale (come l'Internet) non possono avere una topologia full mesh, non solo per ovvi motivi logistici ma anche perché la maggior parte dei collegamenti non verrebbe utilizzata. Questo tipo di reti hanno, tipicamente, una **partially connected mesh topology**:



Non tutti i link sono uguali, alcuni hanno una **capacità maggiore** di altri e possono occuparsi di un carico di informazioni maggiore per unità di tempo. Si ricorre a questa soluzione perché se due nodi non sono direttamente collegati, possono comunicare tramite un percorso alternativo che attraversa altri nodi intermediari; ad esempio, A può comunicare con F seguendo i percorsi:

$$A \leftrightarrow D \leftrightarrow F$$

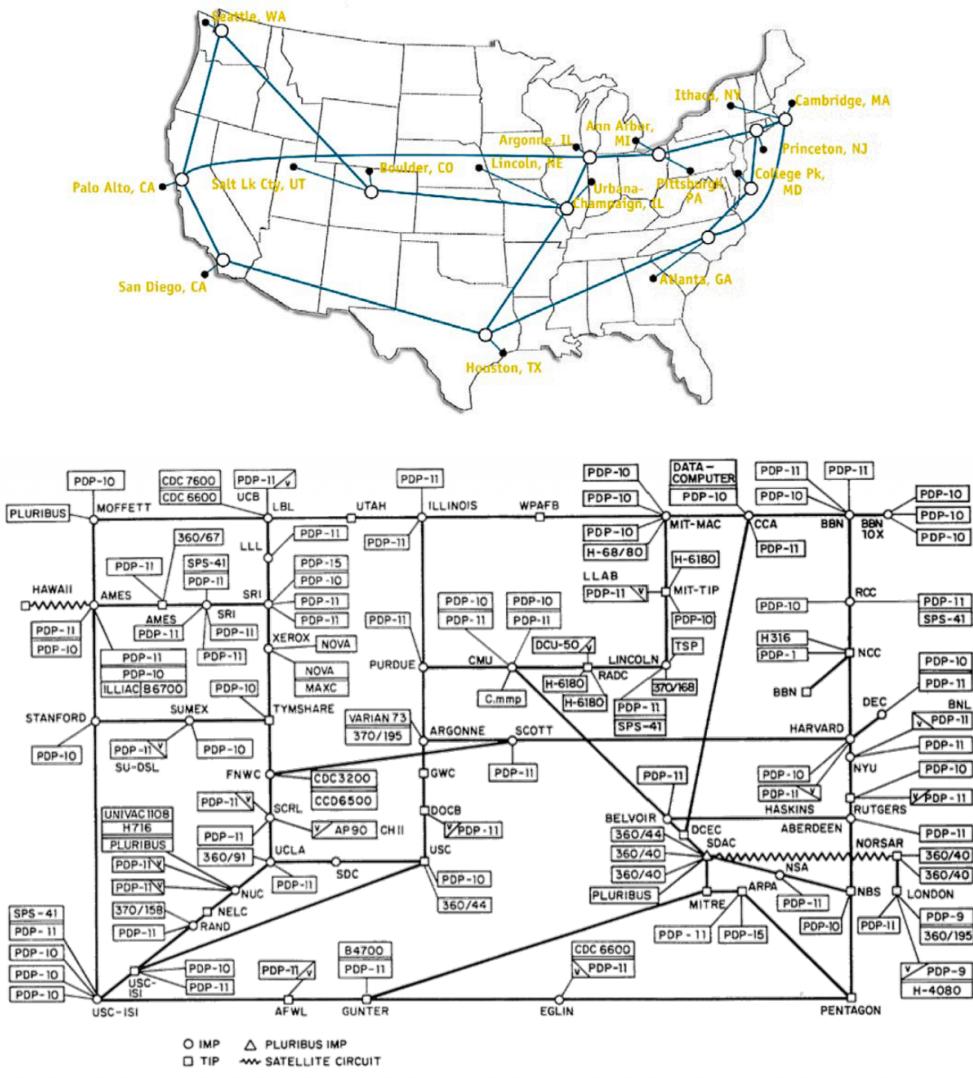
$$A \leftrightarrow B \leftrightarrow C \leftrightarrow D \leftrightarrow F$$

Spesso si nota che la parte più interna della rete (detta backbone) ha una topologia magliata, garantendo la trasmissione anche nel caso in cui certi collegamenti non siano più disponibili, mentre la parte periferica ha una topologia parzialmente stellata.

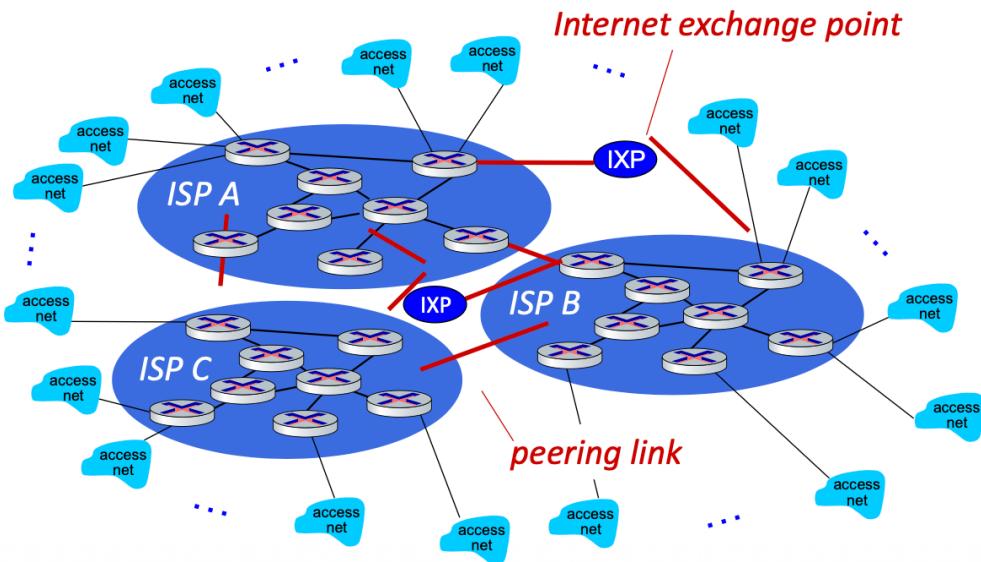
INTERNET

La più grande WAN è chiamata **Internet** ed è stata inizialmente creata come **DARPNET** per iniziativa di **DARPA** (Defense Advanced Research Projects Agency Network), quindi per uso militare, negli anni 60 dal dipartimento di difesa degli USA (DoD) con lo scopo di condividere ricerche e risorse computazionali tra i vari istituti governativi e accademici. Il nome Internet, giunto successivamente, richiama al concetto di interconnessione di reti (inter-network), inteso come rete di reti singolarmente gestite da istituti indipendenti e aziende. Prima di giungere al nome oggi conosciuto, **DARPNET** è divenuto **ARPANET** e **NSFNET** ed è stato anche il primo ad implementare la suite di protocolli TCP/ICP. Il primo messaggio mandato tramite ARPANET è partito il 29 Ottobre 1969 dalla UCLA e diretto verso SRI, conteneva la stringa "login" ma il sistema è crashato prima che il messaggio potesse essere completamente inviato.

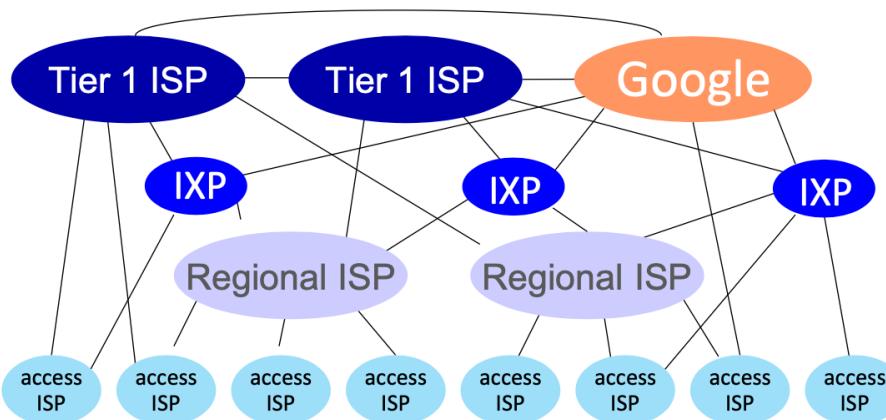
NSFNET T3 Network 1992



Internet è una rete di reti che opera tramite ISP, Internet Service Providers; ogni ISP opera in una porzione dell'Internet e comunica con altri ISP tramite IXP, Internet Exchange Points:

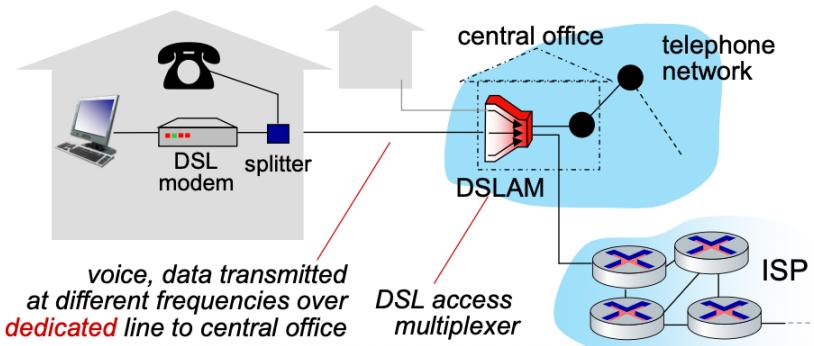


Al “centro” di tutto si trovano poche ma ben connesse reti, come Tier-1 ISP (con coperture nazionali o internazionali, tipo AT&T, NTT, ecc...) e content provider networks (reti private che connettono i propri data center all'Internet bypassando i Tier-1 ISP, tipo Google, Facebook, ecc...), che giungono ai Regional ISP o smistano direttamente agli Access ISP:



Gli ISP regionali si occupano di portare la connettività ad Internet agli utenti terminali; nel corso degli anni questo compito è stato assolto ricorrendo a diverse tecnologie:

- Fino agli anni 90 era in uso un sistema di **dial-up connection** per il quale veniva usata la preesistente **POTS** (Plain Old Telephone Service) in rame:
 - Un **modem** (MOdulator/DEModulator) era usato per **trasmettere il flusso di dati binari lungo la linea telefonica usando le stesse frequenze del servizio**, garantendo un **data-rate di 56kbps**;
- Per la fine degli anni 90 fu sviluppata la **tecnologia ADSL** (Asymmetric Digital Subscriber Line) che permetteva a particolari **modem** di **trasmettere lungo la linea telefonica usando frequenze differenti** (garantendo data-rates di 24Mbps in downstream e 1Mbps in upstream con le varie evoluzioni che si sono susseguite nel tempo, da cui il termine xDSL):
 - **xDSL** si affida all'infrastruttura telefonica e, pertanto, **necessita di uno splitter** che **separa il segnale telefonico da quello per i dati provenienti da Internet**.



Per raggiungere **data-rates più importanti** a livello consumer, è stata introdotta la **fibra ottica** nella rete di accesso agli ISP. Dal 2010 sono stati implementati diversi modelli di distribuzione della fibra ottica:

- **Fiber To The Cabinet**, FTTC;
- **Fiber To The Building**, FTTB;
- **Fiber To The Home**, FTTH.

Il modello FTTC (o **mixed copper**) usa rame dagli edifici degli utenti fino al cabinato nelle strade e poi fibra ottica fino al Central Office dell'ISP; inoltre, utilizza modulazioni avanzate (come VDSL, Very-high-bit-rate Digital Subscriber Loop fino a 55Mbps in downstream e 3Mbps in upstream, o VDSL2, 200Mbps in downstream e 100Mbps in upstream) per raggiungere alti data-rates.

Il modello FTTB porta la fibra ottica fino alle fondamenta degli edifici degli utenti e il modello FTTH fino alle utenze vere e proprie, terminando in un CPE (Customer Premises Equipment) router.

IL MODELLO ISO-OSI E I PROTOCOLLI

La comunicazione tra computer richiede soluzioni tecniche complesse e articolate riguardanti una serie di problematiche:

- Ricezione e trasmissione fisica;
- Controllo degli errori;
- Controllo di flusso;
- Conversione dei dati;
- Crittografia e sicurezza;
- Sincronizzazione.

Un approccio logico comune per la soluzione di questo tipo di problematiche consiste nell'analizzarle singolarmente, **“Divide et impera”**. Nelle reti di calcolatori, ciò ha portato all'ideazione dei cosiddetti **modelli “a strati”**; infatti, la suddivisione delle funzionalità secondo un **modello a strati agevola**, come sarà chiaro a breve, la **gestione della complessità**.

Ciascun strato (o livello o layer) è responsabile di un sottoinsieme definito e limitato di compiti, funziona in maniera lasciamente accoppiata con gli altri strati, interagisce solo con gli strati immediatamente superiore ed inferiore, fa affidamento sui “servizi” forniti dallo strato immediatamente inferiore e fornisce “servizi” allo strato immediatamente superiore. Alcuni

strati, in particolare quelli più bassi, sono realizzati a livello hardware, mentre quelli tendenzialmente superiori a livello software. I vantaggi di questo approccio sono i seguenti:

- L'indipendenza tra gli strati permette la sostituzione di uno strato con un altro di pari livello che offre i medesimi servizi allo strato superiore ma che funziona meglio;
- Limitare le funzionalità di uno strato ne semplifica la realizzazione e minimizza i rischi di errori o malfunzionamenti.

Mentre, per quanto riguarda gli svantaggi:

- L'eccessivo numero di strati può portare ad inefficienze o ad una scarsa leggibilità della struttura.

Storicamente, l'**Organizzazione Internazionale per la Standardizzazione (ISO)** ha stabilito il **modello OSI (Open Systems Interconnection)**, basato su sette layers:

- I layer dal 1 al 3 sono implementati sia nei terminali che nei gateway (più in particolare, 1 e 2 sono gli effettivi strumenti di collegamento, il 3 è implementato solo nei router);
- I layer dal 4 al 7 sono implementati nei sistemi terminali.

7	Application layer
6	Presentation layer
5	Session layer
4	Transport layer
3	Network layer
2	Data Link layer
1	Physical layer

Ad oggi, questo modello è usato solo a scopi didattici ma le sue evoluzioni sono ancora impiegate a livello commerciale. Volendo approfondire le funzioni di questi layer:

- **L1 – Fisico**

Si occupa di **trasmettere sequenze binarie sul canale di comunicazione**. A questo livello si specificano le **caratteristiche elettriche dei segnali**, le **tecniche di codifica e decodifica**, le **caratteristiche dei mezzi trasmissivi e i tipi di connettori**. Il livello fisico è nel **dominio dell'ingegneria elettronica**, della descrizione elettro/meccanica dell'interfaccia, ed è, chiaramente, realizzato a livello hardware.

- **L2 – Data Link**

Ha come scopo la **trasmissione affidabile** (nel senso di “garanzia di inoltro”) di pacchetti di dati (frame) ed accetta come input i frame (poche centinaia di byte) e li trasmette sequenzialmente; inoltre, verifica la presenza di errori di trasmissione aggiungendo delle informazioni aggiuntive di controllo (FCS, Frame Control System) e può gestire meccanismi di correzione di errori tramite ritrasmissione.

- **L3 – Rete**

Questo livello **gestisce l'instradamento dei messaggi e determina quali sistemi intermedi devono essere attraversati da un messaggio per giungere a destinazione**. In sostanza, quindi, il terzo livello gestisce delle tabelle di instradamento per ottimizzare il traffico sulla rete.

- **L4 – Trasporto**

Fornisce servizi per il trasferimento dei dati da terminale a terminale (end-to-end) indipendentemente dalla rete sottostante. In particolare, il quarto livello può frammentare i pacchetti in modo che abbiano dimensioni idonee per L3, rilevare e correggere errori, controllare il flusso e, quindi, anche le congestioni.

- **L5 – Sessione**

È responsabile dell'organizzazione del dialogo e della sincronizzazione tra due programmi applicativi e del conseguente scambio di dati; si occupa, cioè, di stabilire la sessione.

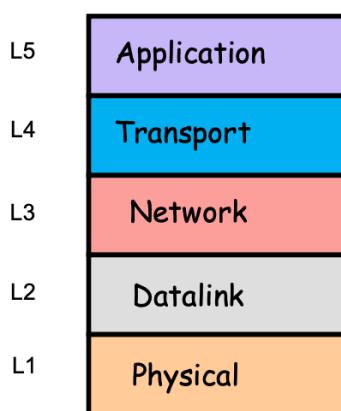
- **L6 – Presentazione**

Gestisce la sintassi dell'informazione da trasferire, la quale è rappresentata in modi diversi su elaboratori diversi (ASCII, EBCDIC, ecc...).

- **L7 – Applicazione**

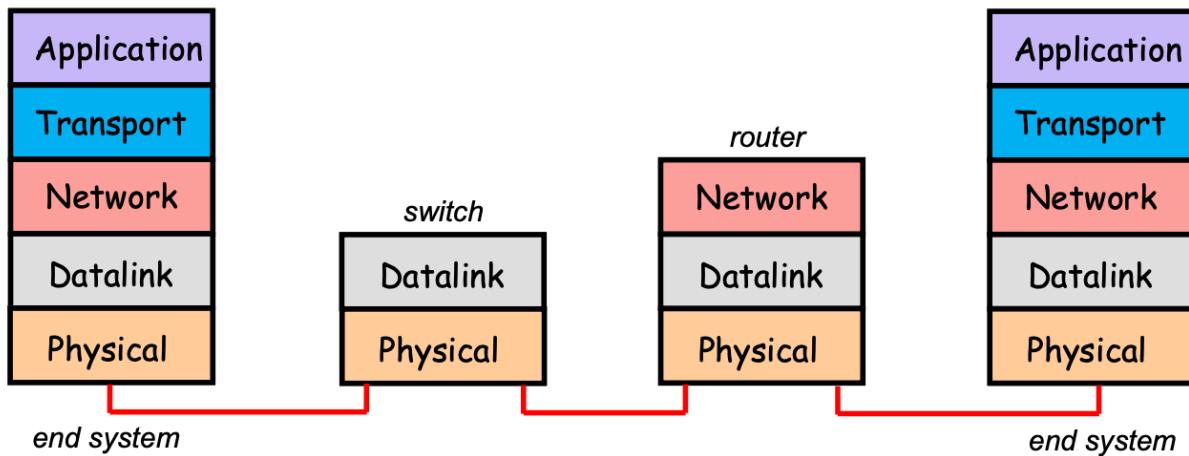
È il **livello dei programmi applicativi**, cioè quei programmi appartenenti al sistema operativo o scritti dagli utenti, attraverso i quali l'utente finale utilizza la rete.

L'Internet è stato progettato considerando la presenza di **cinque livelli in un modello a pila**; a differenza del modello ISO-OSI, **le funzioni di L5 e L6 non sono esplicitamente assegnate ad un layer specifico** (sono assegnate al **layer di applicazione**, se necessario, spesso nominato L7 come nel modello ISO-OSI):

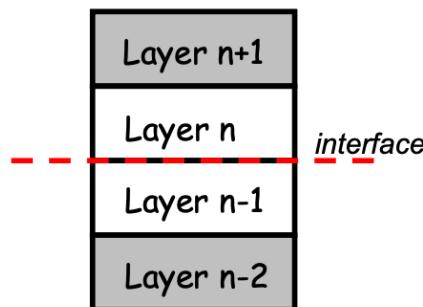


Nella maggior parte dei network, **due sistemi terminali sono interconnessi da un numero arbitrario di dispositivi intermedi**; questi, implementano solo i livelli più bassi, come:

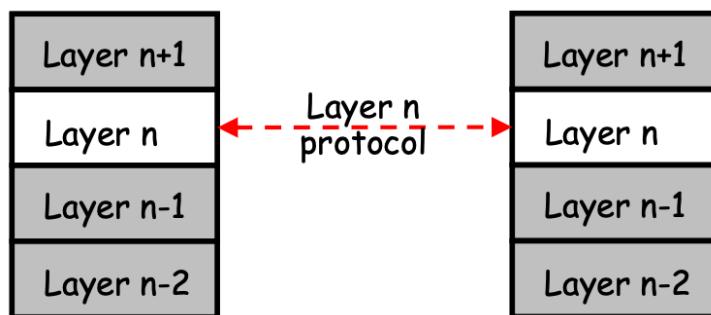
- **Ripetitori**, implementano solo **L1**;
- **Switches**, implementano solo **L1 e L2**;
- **Routers**, implementano solo **L1, L2 e L3**.



All'interno di ciascuno dispositivo di rete, **lo scambio di informazioni tra due strati adiacenti avviene attraverso un'interfaccia**, che definisce i servizi offerti dallo strato inferiore a quello superiore:



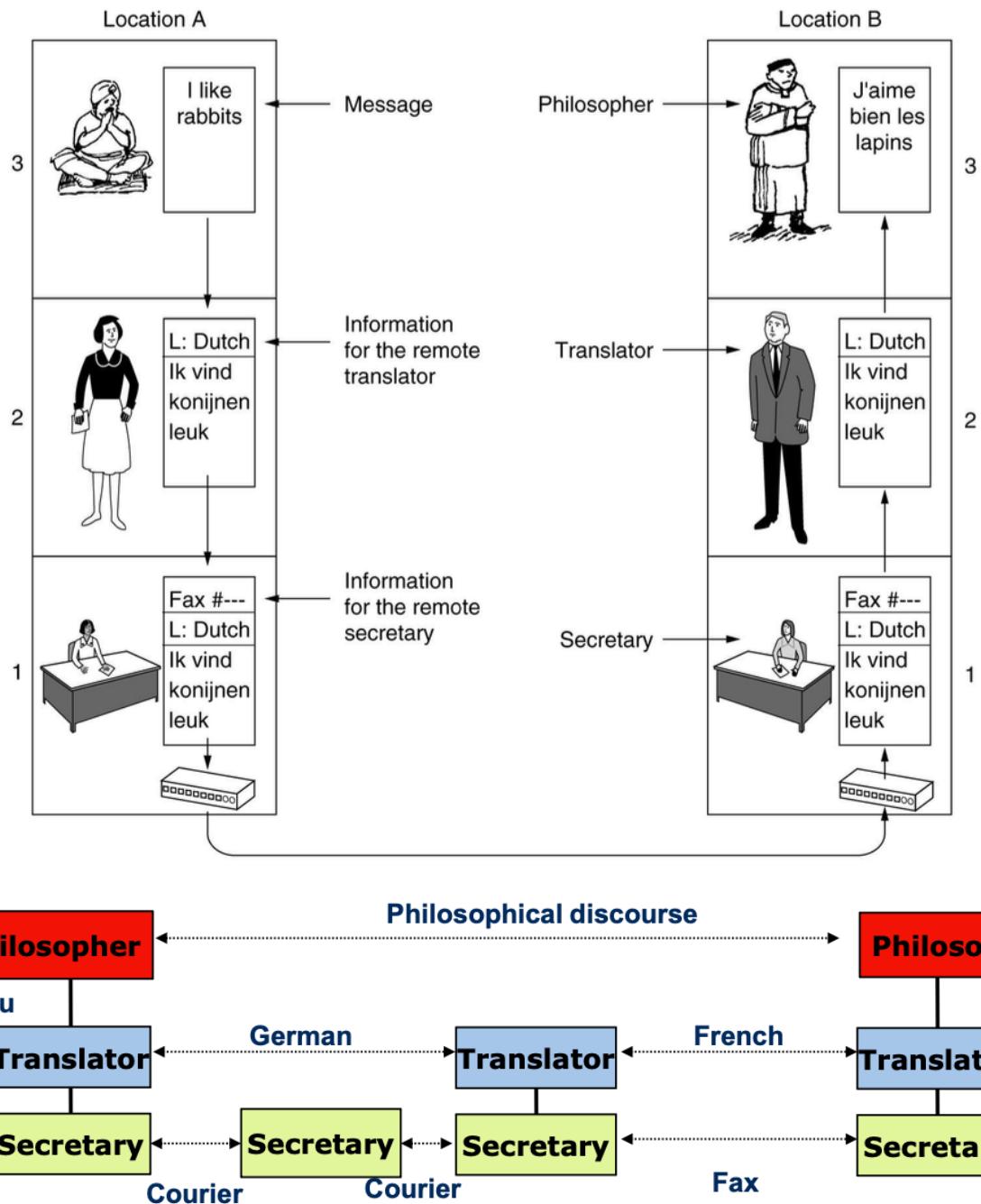
Lo strato n-esimo di un dispositivo comunica sempre con lo strato n-esimo di un'altra entità, seguendo il protocollo assegnato:



Si consideri che **l'interazione tra due livelli diversi** (quindi necessariamente dello **stesso dispositivo**) è **un'interazione locale**, mentre **quella tra due pari livelli** (quindi necessariamente di **due dispositivi diversi**) non necessariamente.

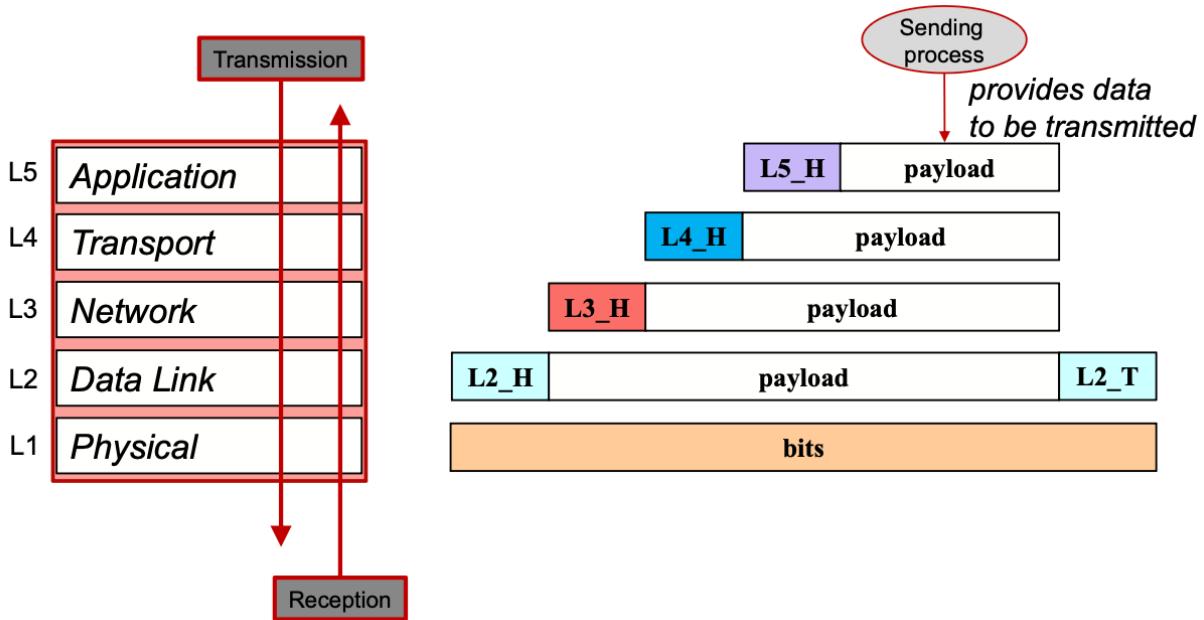
Per **protocollo di comunicazione** si intende un **insieme di regole** che permette la corretta **instaurazione, mantenimento e terminazione di una comunicazione** di qualsiasi tipo tra due o più entità. Un protocollo di comunicazione **definisce il formato e l'ordine dello scambio di messaggi** tra le entità comunicanti e, nelle reti di calcolatori, **regola la comunicazione tra entità di pari livello esistenti in due dispositivi della rete tra loro comunicanti**; in questo ambito, **un notevole sforzo è stato compiuto per definire protocolli standard**, allo scopo di consentire l'integrazione di reti differenti. Dal momento in cui ogni layer ha i propri protocolli, spesso è utilizzata

una **protocol stack**; pertanto, è necessario il rigore relativo a queste specifiche, in modo che **tutte le interazioni in un protocol stack si sviluppino in maniera automatica**, nonostante spesso si rendano i protocolli corposi.



Si può chiaramente intuire come i **protocolli vadano**, implicitamente, a definire la struttura e la sintassi che un messaggio deve avere per poter essere inviato e ricevuto. Tale sintassi viene descritta in notazioni formali come ABNF (Augmented Backus-Naur Form).

In una pila a livelli di protocolli, **ogni layer riceve un payload dal livello superiore e forma un PDU (Protocol Data Unit)**, fatto di **header e payload**, per inviarlo al layer inferiore come **payload stesso**. Così come nel sistema postale, il contenuto di un collo va messo in un pacco, il quale va indirizzato, l'header del PDU contiene le informazioni di controllo come l'indirizzo del destinatario; quando, poi, un PDU è ricevuto, il payload è estratto e passato al livello superiore.



Spesso capita che, nel momento in cui avviene un’interazione tra due terminali, essa inizi tipicamente da un’entità di livello massimo, applicativa, rendendo necessaria la comunicazione con i livelli sottostanti. Ciò viene realizzato discendendo lo stack in ordine decrescente (trasmissione) ed accrescendo il messaggio di informazioni di controllo nell’header relative al livello in cui sono inserite. Una volta ricevuto il messaggio, il terminale dovrà portare il messaggio al livello applicativo (ricordando che i livelli possono comunicare solo con i propri omologhi), facendo un unwrapping ascendente lungo i vari livelli soprastanti (ricezione).

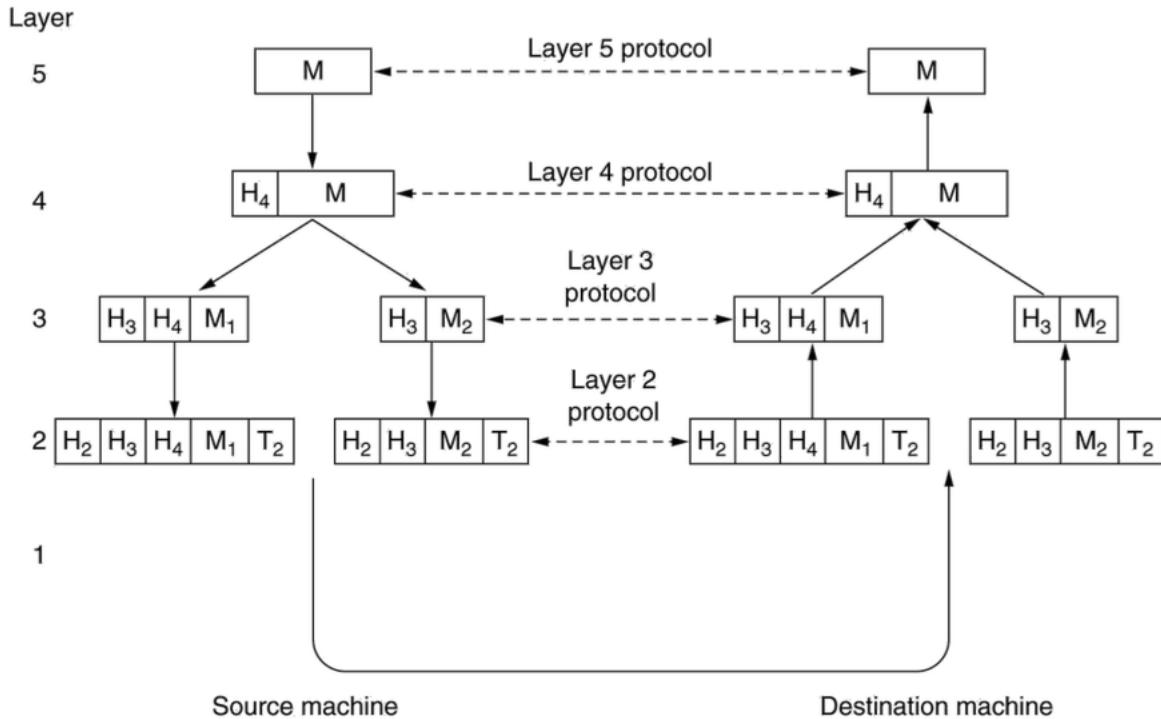
L’interazione tra un livello ed uno ad esso adiacente avviene in punti detti Service Access Point (SAP), mentre l’unità di trasmissione gestita da ogni livello viene detta Protocol Data Unit (PDU), chiamata segmento al livello trasporto e applicativo, datagramma al livello rete e frame al livello link, mentre a livello fisico si parla direttamente di bit.

Layer	PDU name
Application	Message
Transport	Segment
Network	Datagram
Data Link	Frame
Physical	Bit

Dunque, analizzando la struttura di un messaggio, si può osservare come l’header più esterno appartenga al primo livello, quello immediatamente successivo al secondo livello e così via fino all’header adiacente al payload, che appartiene al livello applicativo. Tuttavia, sebbene sia possibile fare questa distinzione dall’esterno, ogni livello non vede altro che il proprio header (i primi N bit del messaggio) e un payload, che poi si sa essere un insieme di altri header e un payload generale.

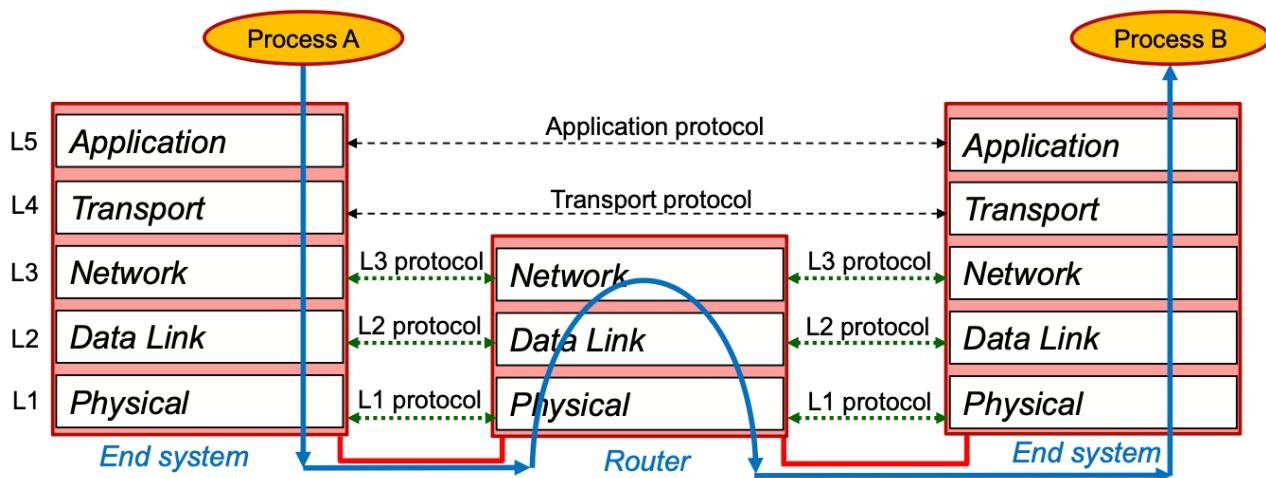
Ad ogni livello della pila, può capitare che il payload sia troppo largo per poter essere incluso in un singolo PDU (si può notare come ad ogni livello la grandezza del messaggio aumenta). In queste circostanze, è possibile separare il payload in una sequenza di pacchetti (frammentazione), che

poi verrà ricostruito in fase di ricezione (riassembaggio), e apporre ad ognuno di essi un header per comunicare eventuali istruzioni di riassembaggio:

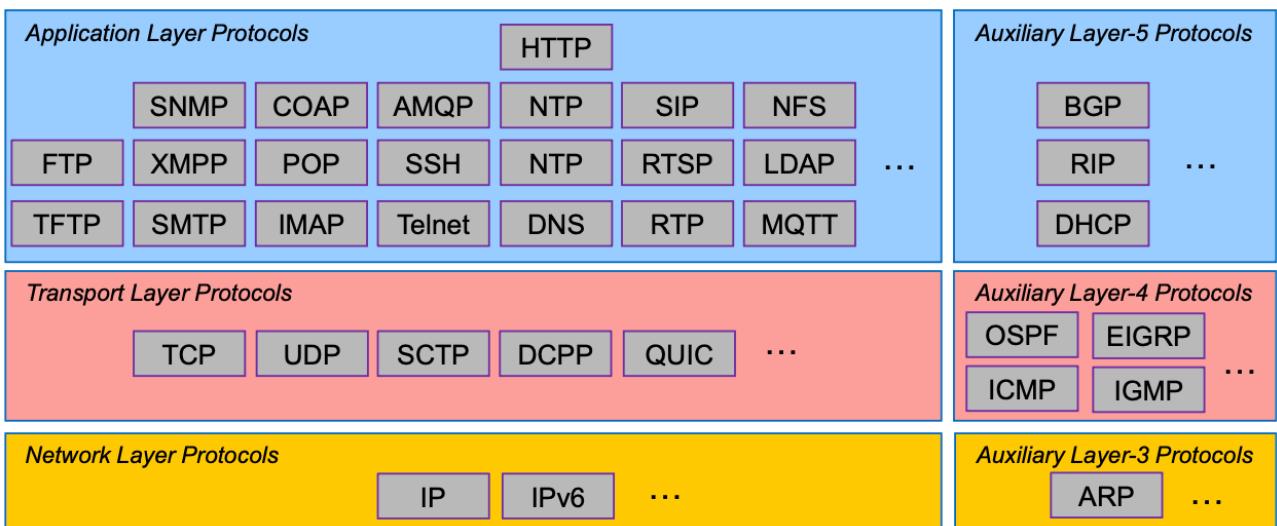


Si consideri che per **payload** si intende l'insieme del payload originale del messaggio e degli header dei livelli successivi e, pertanto, anch'essi saranno soggetti a frammentazione. Questo, però, in seno al principio di incapsulamento precedentemente illustrato, non costituisce un problema in fase di ricostruzione.

Ricapitolando, la comunicazione end-to-end attraverso un sistema intermediario avviene secondo il seguente schema:



Per **Internet Protocol Suite** si intende un insieme di protocolli ad oggi utilizzati attraverso l'Internet, conosciuti anche come **TCP/IP protocol stack**; la maggior parte di questi protocolli sono stati definiti dall'IETF e sono “**open standards**”. L'IPS non considera layer sotto il livello Network perché il protocollo IP può adattarsi ad ogni tecnologia di layer 2.



Per standard si intende un framework di specificazioni che è stato o approvato da un'organizzazione riconosciuta (de-iure) o è generalmente accettata e ampiamente utilizzata nell'industria (de-facto). L'adattarsi agli standard è necessario per ottenere l'interoperabilità tra prodotti di differenti produttori, andando a favorire la competizione e a incentivare l'innovazione. Nello sviluppo delle ICT (information and communications technology) hanno particolare rilevanza gli **open standards**, definiti come quegli standard che sono facilmente accessibili da tutti i lettori e da tutti gli utenti e che sono sviluppati tramite un processo collaborativo aperto (come progetti open source); ad oggi, però, si è ancora in dibattito circa il fatto che un vero e proprio open standard debba essere royalty-free o meno.

IL PROTOCOLLO HTTP

Il protocollo HTTP è stato concepito per la trasmissione di hypertexts da Tim-Berners Lee e fu implementato per la prima volta al CERN. Si basa su TCP e prevede che il client apra un socket verso il port TCP 80 del server (se non diversamente specificato), il server accetta la connessione ed il client manda una richiesta per uno specifico oggetto (un file o un insieme di file), identificato tramite URL (Uniform Resource Locator), e il server risponde e chiude la connessione. Il protocollo HTTP è stateless: né il server né il client mantengono, a livello HTTP, informazioni relative ai messaggi precedentemente scambiati.

La sintassi per un **URL HTTP** è la seguente (secondo la RFC 2396):

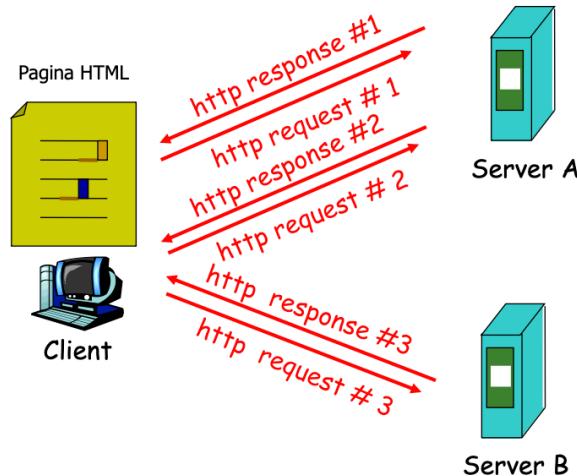
[http://host\[:port\]/path\[#fragment\] \[?query\]](http://host[:port]/path[#fragment] [?query])

Dove:

- **host** identifica il sever e può essere sia un nome simbolico che un indirizzo IP in notazione dotted decimal;
- **port** è opzionale, di default è 80;
- **path** identifica la risorsa sul server (come, ad esempio, images/sfondo.gif);
- **#fragment** indica un punto preciso all'interno di un oggetto;
- **?query** è usato per passare informazioni dal client al server (come, ad esempio, i dati inseriti in un form).

Tipicamente, **una pagina web è descritta da un file testuale in formato HTML** (HyperText Markup Language) ed è identificata tramite un **indirizzo**, detto **URL**. Un file HTML può contenere riferimenti ad altri oggetti che arricchiscono la pagina con elementi grafici (come immagini, uno sfondo, audio, ecc...), ciascuno dei quali è identificato dal proprio URL (infatti, possono anche trovarsi su server o pagine web diversi). Una volta **ricevuta la pagina HTML**, il browser estraie i riferimenti agli altri oggetti, che devono essere prelevati e li richiede attraverso una serie di connessioni HTTP.

Un esempio di richiesta di una pagina contenente immagini prevede:



1. Il client apre una connessione TCP sul port 80 verso l'indirizzo www.unina.it

→

2. Il server è in ascolto sul port 80 ed accetta la connessione

←

3. Il client invia un messaggio di richiesta HTTP della home page

→

4. Il server analizza la richiesta HTTP, prepara la risposta HTTP, e la invia al client

↓

5. Il server chiude la connessione TCP

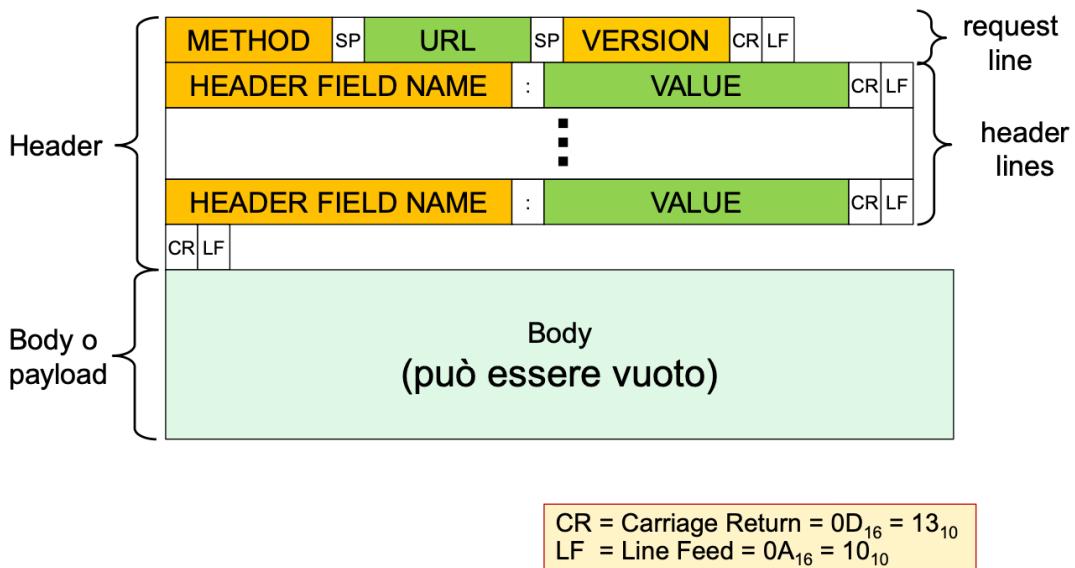
←

6. Il client effettua il parsing del documento (ad esempio, HTML) contenuto nel messaggio di risposta HTML, ne fa il rendering sullo schermo e rileva che, all'interno della pagina, sono presenti tre collegamenti ad immagini

↓

7. Per ciascuna delle immagini, vengono ripetuti i passi da 1 a 5

HTTP è un protocollo testuale, i messaggi sono costituiti da una **sequenza di byte**, ognuno dei quali identifica **un carattere secondo la tabella ASCII**. Ciò **garantisce che il payload dei messaggi possa essere comunque in formato binario** (come un'immagine, un video, ecc...).



Una richiesta HTTP è caratterizzata dal campo **Method** nella **Request Line** e, in HTTP/1.1, può assumere i seguenti valori (sebbene in questa sede verranno approfonditi solo i primi quattro):

- GET;

Il metodo **GET** è usato per richiedere una risorsa identificata da un URL (se disponibile, il server la invia nel body del messaggio di risposta) ed è un **metodo sicuro ed idempotente**. È il metodo **più frequentemente usato e quello attivato in browser quando si fa click su un link ipertestuale** in un documento HTML o si inserisce un URL nella **barra degli indirizzi del browser**; inoltre, GET **può essere**:

- **Assoluto**, la risorsa viene richiesta senza altre specificazioni;
- **Condizionale**, si richiede la risorsa se è soddisfatto un criterio indicato negli header (if-match, if-modified-since, if-range, ecc...);
- **Parziale**, si richiede una sottoparte di una risorsa memorizzata.



- HEAD;

Il metodo **HEAD** è simile a GET: è un **metodo sicuro e idempotente** e prevede che si richieda la **validità di una risorsa identificata da un URL** ma, a differenza di GET, la **risorsa non viene inviata nel messaggio di risposta**. HEAD serve al client per verificare:

- **La validità di un URL e per apprendere le caratteristiche della risorsa** rappresentate nell'header del messaggio di risposta;
 - **Accessibilità di un URL**, cioè per verificare se l'accesso ad una risorsa non sia vincolato ad un'autenticazione;
 - **Verifica di coerenza di una copia della risorsa già disponibile** nella cache del browser rispetto all'originale sul server.
- **POST;**

Il metodo **POST** è usato per trasmettere informazioni dal client al server senza la creazione di una nuova risorsa; l'URL presente nel messaggio di richiesta è associato ad un programma eseguito nel server che riceve come input le informazioni inviate dal client nel body del messaggio di richiesta. POST è un metodo non sicuro e non idempotente e il server può rispondere positivamente in tre modi:

- **200 Ok**, dati ricevuti e sottomessi alla risorsa specificata, è stata data risposta;
- **201 Created**, dati ricevuti ma la risorsa non esisteva ed è stata creata;
- **204 No content**, dati ricevuti e sottomessi alla risorsa specificata, non è stata data risposta.

In un'applicazione web, l'invio di dati da un client ad un server può avvenire sia usando il metodo **GET** che **POST**; ad esempio, un form HTML:

```
<form action="/action_page.php" method="post">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <label for="lname">Last name:</label>
  <input type="text" id="lname" name="lname"><br><br>
  <input type="submit" value="Submit">
</form>
```

- **PUT;**

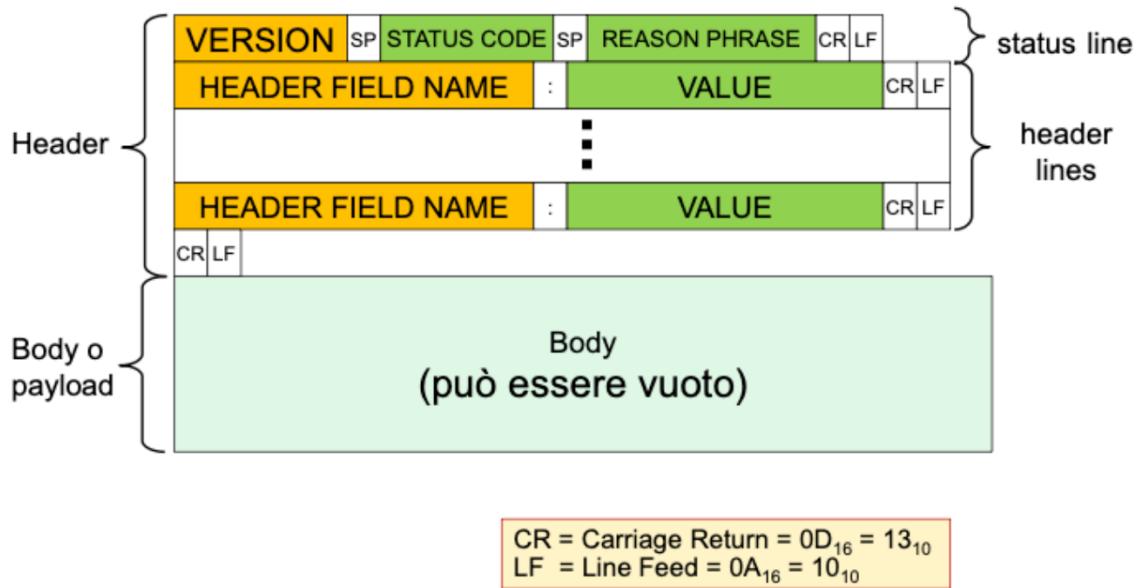
Il metodo **PUT** serve per trasmettere delle informazioni dal client al server, creando o sostituendo la risorsa specificata dall'URL; in caso di risposta positiva ad una richiesta PUT, i dati inviati nel body del messaggio PUT dovrebbero costituire il valore della risorsa che il server restituisce ad una successiva richiesta GET per la stessa URL. Il metodo **PUT** è non sicuro ma idempotente e, per motivi di sicurezza, scarsamente utilizzato.

- **DELETE;**
- **TRACE;**
- **OPTIONS;**

Un metodo HTTP può essere:

- **Sicuro**, nel senso che non genera cambiamenti allo stato interno del server (GET e HEAD sono sicuri);
- **Idempotente**, nel senso che l'effetto sul server di più richieste identiche è lo stesso di quello di una sola richiesta (tutti i metodi HTTP tranne POST sono idempotenti).

Il formato di un **messaggio di risposta** è il seguente:



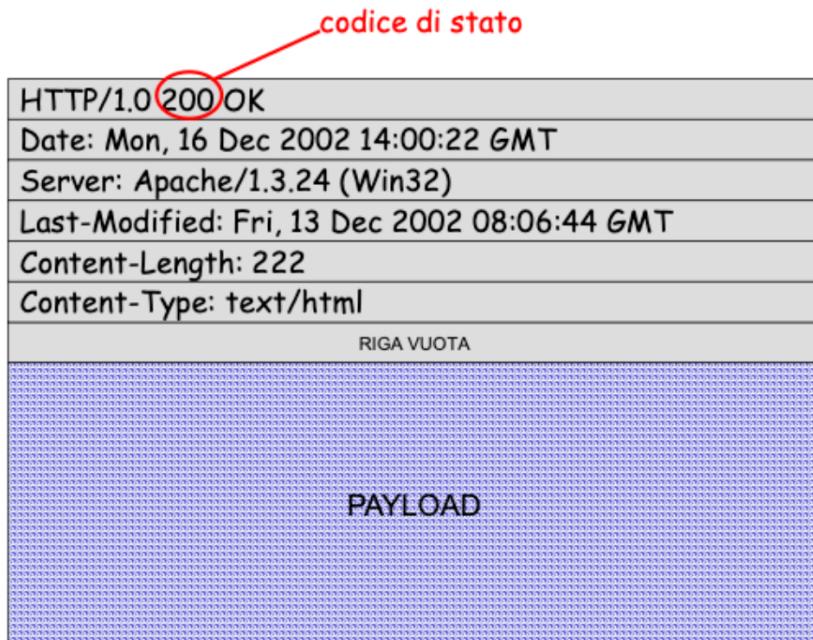
Lo **status code** è un **numero di tre cifre**, di cui la **prima** indica la **classe della risposta** e le altre **due** la **risposta stessa**. Esistono le seguenti **classi di risposta**:

- **1xx Informational**, una risposta temporanea alla richiesta, durante il suo svolgimento;
- **2xx Successful**, il server ha ricevuto, capito ed accettato la richiesta;
- **3xx Redirection**, il server ha ricevuto e capito la richiesta ma sono necessarie altre azioni da parte del client per portare a termine la richiesta;
- **4xx Client error**, la richiesta del client non può essere soddisfatta per un errore da parte del client (errore sintattico o richiesta non autorizzata);
- **5xx Server error**, la richiesta può anche essere corretta ma il server non è in grado di soddisfare la richiesta per un problema interno.

Alcuni esempi di status code completo sono:

- **100 Continue**, se il client non ha ancora mandato il body;
- **200 Ok**, GET con successo;
- **201 Created**, PUT con successo;
- **301 Moved permanently**, URL non valida, il server indica la nuova posizione (**Location:**);
- **400 Bad request**, errore sintattico nella richiesta;
- **401 Unauthorized**, manca l'autorizzazione;
- **403 Forbidden**, richiesta non autorizzabile;
- **404 Not found**, l'oggetto richiesto non è presente sul server, URL non valido;
- **500 Internal server error**, tipicamente un programma in esecuzione sul server ha generato errore;
- **501 Not implemented**, metodo non conosciuto al server;
- **505 HTTP Version not supported**, la versione del protocollo HTTP usata non è supportata dal server.

Un esempio di **messaggio di risposta** è il seguente:



Gli **header generali** si applicano solo al messaggio trasmesso e sia ad una richiesta che ad una risposta (non necessariamente alla risorsa trasmessa). Sono composti da:

- **Date**, data ed ora della trasmissione;
- **MIME-Version**, la versione MIME usata per la trasmissione (sempre 1.0);
- **Transfer-Encoding**, il tipo di formato di codifica usato per la trasmissione;
- **Cache-Control**, il tipo di meccanismo di caching richiesto o suggerito per la risorsa;
- **Connection**, il tipo di connessione da usare (Keep-Alive, tenere attiva dopo la risposta, oppure Close, chiudere dopo la risposta, ecc...);
- **Via**, usato da proxy e gateway.

Si sottolinea che, da **HTTP/1.1** tutte le connessioni sono di base Keep-alive ed è, quindi, necessario specificare **Close** se si vuole chiudere una specifica connessione. Nonostante ciò, **HTTP rimane stateless**, perché alla chiusura di una connessione nessuna informazione permane. Ulteriormente, queste sono solo **richieste che si inviano al server**, che possono essere rispettate o meno anche senza informare il client (tant'è che spesso un server chiuderà una connessione, prima o poi, per risparmiare risorse).

Gli **header di risposta**, invece, sono posti dal server per specificare informazioni sulla risposta e su sé stesso al client. Sono composti da:

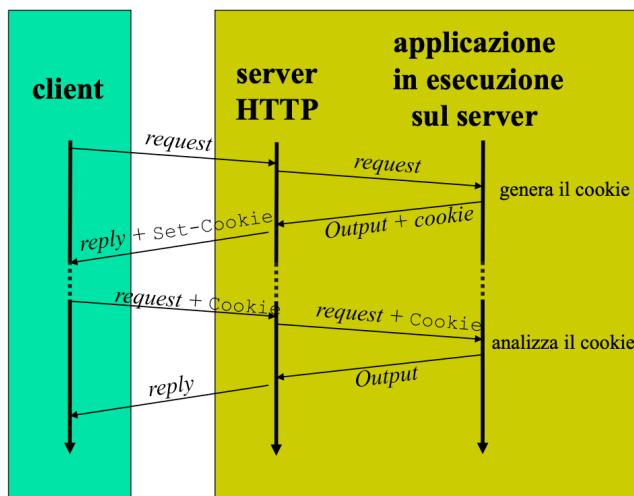
- **Server**, una stringa che descrive il server (tipo, Sistema Operativo e versione);
- **Accept-ranges**, specifica che tipo di ragni può accettare (valori previsti: byte e none).

Gli **header dell'entità** danno informazioni sul body del messaggio o, se questo è assente, sulla risorsa specificata. Sono composti da:

- **Content-Type**, oggetto/formato (ogni coppia costituisce un tipo MIME dell'entità acclusa) è obbligatorio in ogni messaggio che abbia un body e specifica se è un testo, un'immagine GIF, un'immagine JPEG, un suono WAV, un filmato MPG, ecc...
- **Content-Length**, la lunghezza in byte del body ed è obbligatorio, soprattutto se la connessione è persistente;

- **Content-Base**, Content-Encoding, Content-Language, Content-Location, Content-MD5, Content-Range, ovvero l'URL di base, la codifica, il linguaggio, l'URL della risorsa specifica, il valore di digest MD5 e il range richiesto dalla risorsa;
- **Expires**, una data dopo la quale la risorsa è considerata non più valida (e quindi va richiesta o cancellata dalla cache);
- **Last-Modified**, la data e l'ora dell'ultima modifica (serve per decidere se la copia posseduta è ancora valida o meno ed è obbligatoria se possibile);

Come già anticipato, **HTTP è stateless**, il server non è tenuto a mantenere informazioni su connessioni precedenti. Per **cookie**, si intende una breve informazione scambiata tra server e client e tramite il quale il client mantiene lo stato di precedenti connessioni, mandandolo al server di pertinenza ogni volta che viene richiesto un documento; ad esempio, tramite un cookie viene riproposto il proprio username all'atto di accesso ad un sito per la posta. I cookie sono stati definiti originariamente nel 1997 nel documento RFC 2109 (attualmente in RFC 2965 "HTTP State Management Mechanism") ma non sono menzionati nei documenti che definiscono HTTP.



Il meccanismo dei cookies definisce **due nuovi possibili header**, uno per la risposta ed uno per le richieste successive:

- **Set-Cookie**, header della risposta con il quale il client può memorizzarlo (se vuole) e rispedirlo alla prossima richiesta;
- **Cookie**, header della richiesta con il quale il client decide se spedirlo sulla base del nome del documento, dell'indirizzo IP del server e dell'età del cookie.

Un browser può essere configurato per accettare o rifiutare i cookie ed alcuni siti web richiedono necessariamente la capacità del browser di accettare i cookies.

In aggiunta ad un nome ed un valore, un cookie può avere degli attributi che ne specificano la durata o lo "scope" e sono specificati solo nei messaggi di risposta del server, non sono inviati dal browser. Gli attributi che definiscono la durata di un cookie sono:

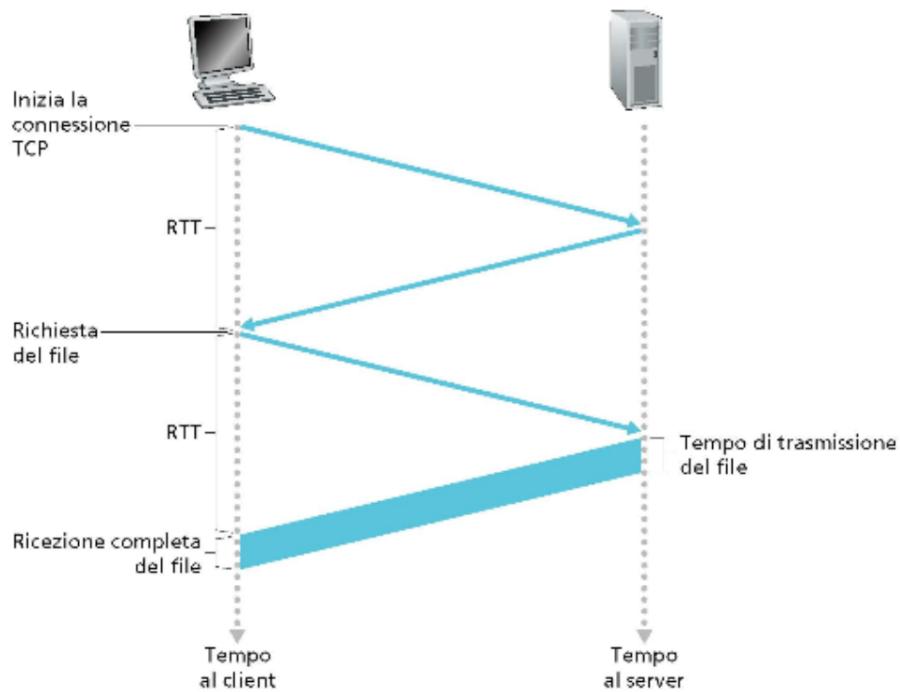
- **Expires**, specifica una data oltre la quale il browser deve eliminare i cookie;
- **Max-Age**, specifica dopo quanto tempo (in secondi), rispetto al momento della creazione il browser dovrà eliminare il cookie (non è supportato in Internet Explorer).

Un cookie per il quale non è specificata la durata è detto **session cookie** e dura finché il processo del browser non è terminato.

L'attributo **Domain**, invece, **indica i domini DNS a cui si applica il cookie**. Se non è specificato, il browser invia il cookie a tutte le successive richieste fatte al server dal quale il cookie è stato ricevuto; ad esempio, se il server `docs.foo.com` invia il cookie `Set-Cookie: HSID=AYQEvn...DKrdst; Domain=foo.com` il cookie sarà inviato a tutte le successive richieste fatte ai server del dominio `foo.com`, altrimenti solo al server `docs.foo.com`. Invece, l'attributo **Path** indica a quali oggetti (URL) si deve applicare il cookie (se `Path=/`, il cookie si applica a tutti gli oggetti).

Quando non specificato altrimenti, **un cookie è valido nel dominio e path di creazione**, mentre se **un cookie è valido per un qualsiasi dominio**, **è valido anche per tutti i suoi sottodomini ma non in quelli di dominio superiore** (così come se è valido per un path, lo è per tutti i percorsi ad esso innestati ma non per quelli in cui è innestato); è possibile porre il campo **Domain** (e similmente il campo **Path**) **per domini di livello maggiore e minore rispetto a quello di origine** ma con una limitazione sul TLD.

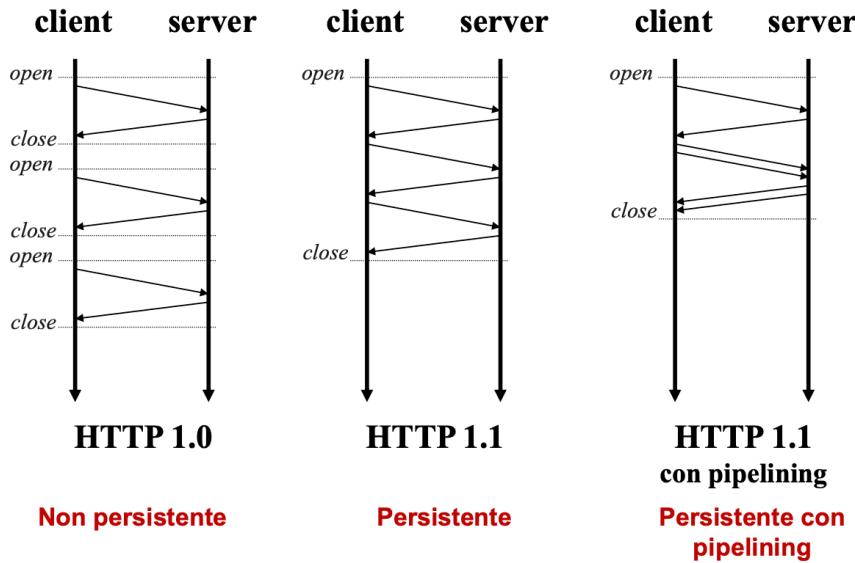
Per **Round Trip Time**, o **RTT**, si intende **il tempo che intercorre tra l'inizio della connessione TCP e l'istante in cui il client è abilitato a richiedere il file** (cioè dopo che ha ricevuto la conferma di avvenuta connessione dal server):



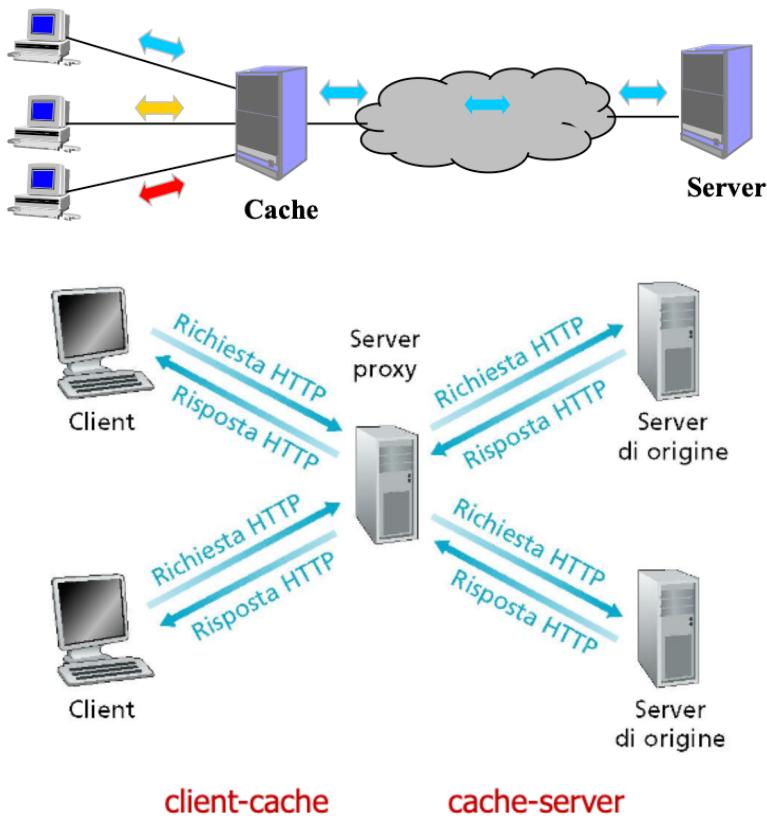
Il protocollo HTTP permette anche di **distinguere connessioni persistenti e non persistenti**:

- **Connessione non persistente**, HTTP/1.0 (RFC 1945), il server analizza una richiesta, la serve e chiude la connessione, richiedendo due RTT per ciascuna richiesta e facendogli subire uno slow-start TCP;
- **Connessione persistente**, HTTP/1.1 (RFC 2068, RFC 2116), il server mantiene attiva una connessione TCP fino allo scadere di un timeout, mentre il client può usare una connessione TCP precedentemente creata con il server per inviare richieste HTTP consecutive, avendo meno RTT.

Esiste anche una **versione con parallelismo della connessione persistente**, ottenuta tramite **pipelining**.



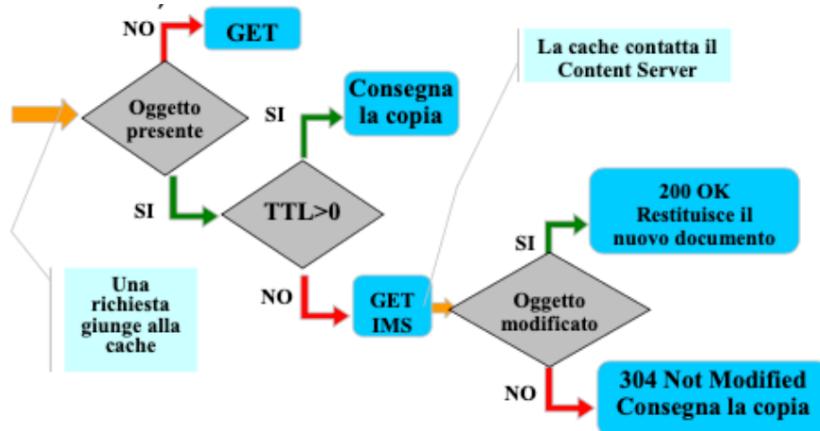
Si parla genericamente di **Web caching** quando le richieste di un determinato client non raggiungono il Web Server ma vengono intercettate da una cache. Tipicamente, un certo numero di client di una stessa rete condivide una stessa Web cache, posta nelle loro prossimità (ad esempio, nella stessa LAN) e, se l'oggetto richiesto non è presente nella cache, questa lo richiede in vece del client conservandone una copia per eventuali richieste successive, servite ovviamente più rapidamente. Con il Web caching si definiscono due nuovi tipi di interazione HTTP: **client-cache** e **cache-server**.



Nel caso in cui la cache fa parte della rete privata in cui il client si trova, si parla di proxy, ma tipicamente nell'utilizzo di ogni browser si ritrova un tipo di cache che fa parte del client stesso.

Il Web caching pone un problema: cosa succede se l'oggetto presente nel server è aggiornato? La copia in cache deve essere aggiornata per mantenersi uguale all'originale. HTTP fornisce due meccanismi per la gestione della coerenza:

- **TTL** (Time To Live), il server quando fornisce un oggetto comunica anche quando quell'oggetto “scade” (header Expires), ovvero quando TTL<0, ma in tal caso non è detto che in realtà l'oggetto sia realmente modificato;
- **Il client può fare un ulteriore controllo mediante una GET condizionale (If-Modified-Since).**



Nella trattazione è stato diverse volte menzionato il **browser**, un'applicazione costituita da un'interfaccia grafica e un browser engine che permettono ai client di effettuare richieste nella rete. Un **browser engine**, a sua volta, è composto da un **layout engine**, che decodifica e visualizza il documento HTML e gli oggetti multimediali in esso contenuto tenendo in considerazione le indicazioni contenute in un file CSS (che ne determinano l'aspetto grafico e lo stile), e un **JavaScript engine**, che esegue il codice JavaScript encapsulato nel documento HTML o contenuto in altri file esterni (file .js). Nel corso degli anni sono stati sviluppati diversi **browser engine**, tra cui:

- **WebKit**, usato da Apple Safari e Chrome (fino alla versione 27);
- **Blink**, usato da Google Chrome (da v.28) e deriva da WebKit;
- **Gecko**, usato da Firefox;
- **Trident**, usato da Internet Explorer;
- **EdgeHTML**, usato da Microsoft Edge e derivato da Trident.

Tranne gli ultimi due, tutti i browser engine sono open source.

In varie situazioni, il ruolo di client HTTP può essere assunto da programmi che devono interagire con un server per ottenere informazioni tramite il protocollo HTTP. Un client HTTP attivabile da command-line è wget:

```
wget <URL> [OPZIONI]
```

Ci sono opzioni che permettono di scaricare ricorsivamente tutti i file puntati dal primo URL (crawling), in modo da ricavare localmente una copia di un intero sito web.

HTTP non è utilizzato solo per il Web, altre applicazioni del protocollo sono i **Web Services** e **SOA** (Service Oriented Architecture), il **Video streaming** (come DASH, Dynamic Adaptive Streaming over HTTP) e il **Peer-to-Peer**.

IL SISTEMA DNS

È stato mostrato come **una risorsa disponibile nell'Internet sia raggiunta da un dispositivo terminale ad essa non fisicamente collegata tramite indirizzi IP e protocolli HTTP**, facilmente comprensibili da macchine computazionali perché sequenze di bit; tuttavia, **all'utente finale queste informazioni non sono pervenute**, i siti in questione **sono raggiunti tramite nomi più facilmente localizzabili come www.google.com, www.unina.it**, ecc... Nomi di questo tipo, però, **non danno informazioni ai dispositivi intermedi, come i router, circa la dislocazione sul territorio della macchina che si desidera contattare, rendendo impossibile un'eventuale instradamento verso tali locazioni.**

Non volendo rinunciare ai nomi simbolici che gli utenti della rete utilizzano, è **stata necessaria la progettazione di un sistema di risoluzione dei nomi simbolici in indirizzi IP**; questo sistema, **associa un identificativo univoco (www.unina.it) ad un indirizzo IP (143.22.229.3)**, permettendo così **di raggiungere la macchina di destinazione conoscendo solamente l'identificativo (umanamente comprensibile)**. Il servizio in questione prende il nome di **Domain Name System (DNS)**, è definito nei documenti RFC 1034 e RFC 1035, ha origine nel 1984 da parte di Paul Mockapretis e **si basa sullo scambio di messaggi UDP sul port 53**.

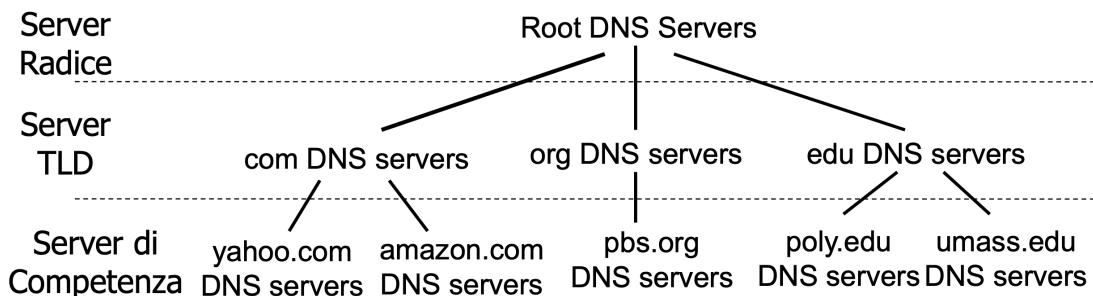
Quella appena illustrata, tuttavia, **non è l'unica funzione offerta da un DNS**, si riconosce anche la possibilità di **specificare alias per gli hostname**, ovvero **si garantisce che una macchina con un nome complicato possa essere raggiunta anche con un "soprannome" più semplice da ricordare** (rcsn1.roma.rai.it → www.rait.it), ma anche per i **server postali**, permettendo di **associare un server di posta ad un dominio postale per facilitarne la memorizzazione dell'indirizzo** (luca.incarnato@unina.it identifica l'utente luca.incarnato sul server mailsrv1.cds.unina.it, con l'associazione @unina.it → mailsrv1.cds.unina.it fatta dal servizio DNS), o **la distribuzione dei carichi**, in uso quando **un server gestisce un carico troppo elevato e si vuole replicare il suo contenuto su molte macchine differenti**, in tal caso **il servizio DNS distribuisce tale carico tra le macchine rilasciando ciclicamente indirizzi appartenenti all'intero pool, senza che gli utenti si accorgano di nulla** (www.domain.com indirizza gli IP address seguenti: 1.2.3.4, 1.2.3.15, 1.2.4.200, 1.2.15.121, 1.5.34.12).

Si può pensare di **posizionare nel globo terrestre, in un unico punto geografico, un'unica macchina che si occupi di realizzare la risoluzione di tutti i nomi presenti nell'Internet**. Questa soluzione, **sebbene teoricamente realizzabile**, presenta diverse criticità:

- Rappresenterebbe un **Single Point of Failure**;
- Dovrebbe **gestire un volume di traffico spropositato**;
- In alcuni luoghi si avrebbe **un database troppo distante geograficamente**;
- Sarebbe richiesta una **manutenzione specifica e frequente**.

Abbandonando l'idea di un DNS centralizzato, si può pensare di adottare un **servizio DNS distribuito**, grazie al quale **si distribuiscono e informazioni tra varie entità server**, ciascuna delle quali **con la responsabilità di raccogliere, gestire, aggiornare e divulgare le informazioni che la riguardano**. Questo tipo di approccio, in particolare, è di **tipo gerarchico: gli elementi più alti nella gerarchia contengono molte informazioni poco dettagliate e gli elementi più bassi nella gerarchia contengono poche informazioni molto dettagliate**. Attraverso un colloquio concertato tra le entità (di cui gli utenti non hanno percezione) si riesce a fornire il servizio di risoluzione.

In cima alla gerarchia si trovano i **root server**, sotto i quali si individuano i **Top Level Domain (TLD) server**, che si occupano di domini di alto livello, e ancora più sotto i vari **server di competenza** (amazon.com, yahoo.com, ...), detti **server autoritativi**.



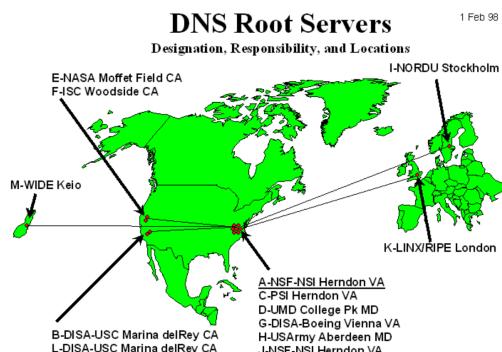
Volendo fare un **esempio pratico**, si supponga che un **client** richieda l'IP di www.amazon.com:

1. **Il client contatta uno dei root server** per avere la lista degli indirizzi IP dei TLD per il dominio com;
2. **Il client contatta uno dei TLD server** che gli restituisce l'indirizzo IP del server autorizzato per amazon.com;
3. **Il client contatta il server di competenza** per amazon.com che gli restituisce l'indirizzo IP di www.amazon.com.

Gli **attori coinvolti in un servizio DNS** sono:

- **Resolver**, il client da cui parte la richiesta di risoluzione al sistema DNS ed è una funzionalità user-level del Sistema Operativo del dispositivo terminale;
- **Registry**, titolare della risoluzione di un determinato namespace ed è l'organizzazione abilitata a fare modifiche al database dei nomi di un determinato dominio, mantenendo in esercizio i server autoritativi per un determinato dominio;
- **Registrar**, è l'agente che sottmette al registry le richieste di modifica di risoluzione per conto del registrant;
- **Registrant**, è l'entità che “possiede” l'uso di un determinato dominio.

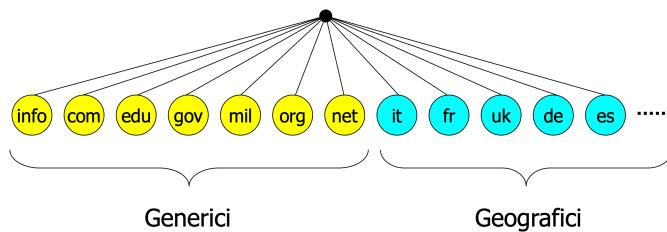
Per quanto riguarda i **Root Name Server**, si parla di **13 server logici in Internet** (etichettati dalla A alla M) **i cui indirizzi IP sono ben noti alla comunità**, sebbene si parli di **376 diversi server fisici ai quali viene fatto riferimento dai Local Name Server** quando non può essere soddisfatta immediatamente una richiesta di risoluzione (**il Local Name Server si comporta come client DNS e invia una richiesta di risoluzione al Root Name Server**).



Ciascun operatore installa nella propria rete un **Local Name Server** e configura gli host con l'**indirizzo del DNS server locale** (in modo che richiedono a questo server il servizio di risoluzione);

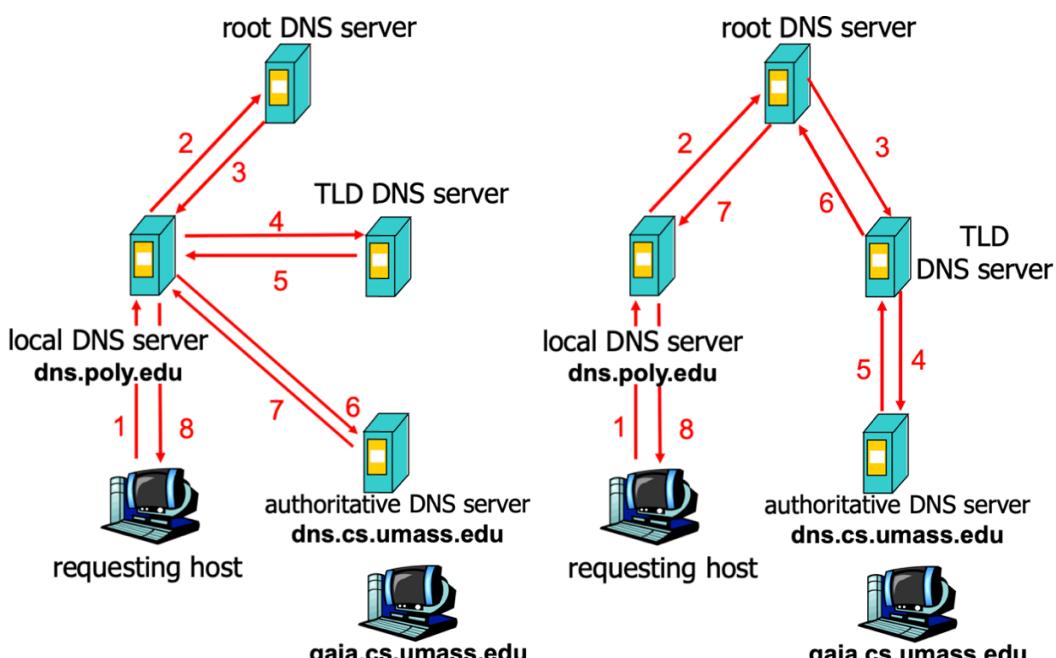
il Local Name Server non opera, in questo meccanismo, **se non da proxy per inviare la query alla gerarchia di server DNS (alla quale non partecipa) e per restituire ai client le risposte finali**. Questo meccanismo di server DSN locali **permette ai client di fare una sola query DNS verso di essi** (il local DNS server interroga i server della gerarchia secondo una sequenza che sarà più chiara a breve e che chiarirà anche il perché vengono chiamati recursive name servers). Infine, alcuni operatori “over-the-top” gestiscono server pubblici con funzione di “DNS server locale” (come Google, per 8.8.8.8 e 8.8.4.4, e Cloudflare, per 1.1.1.1 e 1.0.0.1)

Sono stati già menzionati i Top Level Domain server (TLD), coloro che si occupano dei domini di alto livello (20 generici gTLD e 248 geografici ccTLD).



Per **Autoritative Name Server (Assoluto)** si intende un server dei nomi capace di risolvere tutti i nomi all'interno di un determinato dominio (un server dei nomi Assoluto per il dominio unina.it deve saper risolvere anche nomi tipo studenti.unina.it e simili). Ad esso si riferiscono i Name Server TLD quando devono risolvere un indirizzo del dominio e può essere mantenuto dall'organizzazione che ha titolo all'uso del dominio o da un provider che gestisce il servizio di risoluzione dei nomi per conto del proprietario del dominio.

La risoluzione di un nome tramite servizi DNS può avvenire con **due tipologie di query**, iterative e ricorsive. Quando si utilizzano **query iterative**, un host chiede ad un server DNS locale della rete poly.edu l'indirizzo IP di gaia.cs.umass.edu ed il server contattato risponde con il nome del server da contattare secondo una logica “non conosco questo nome ma conosco qualcuno che lo conosce”. Invece, quando si usano **query ricorsive** si chiede al server contattato di fornire la **risoluzione completa** (questa soluzione comporta un pesante overhead sui server più in alto nella gerarchia).



Il vantaggio dei primi risiede in un carico inferiore per i singoli server, al costo di un tempo di risoluzione potenzialmente maggiore, mentre il caso ricorsivo è certamente più veloce ma richiede ai singoli server un carico di lavoro maggiore.

Può capitare che **il TLD non contatti necessariamente l'Autoritative Name Server finale ma un Autoritative Name Server intermediario**; in tal caso, sarà quest'ultimo a fornire il nome del server di competenza, facendo aumentare il numero di messaggi DNS.

Prima di interrogare il sistema DNS, **il DNS resolver presente nei dispositivi terminali controlla un'eventuale corrispondenza indirizzo-IP nome** presente nel file di sistema **hosts** (in Unix/Linux /etc/hosts, in Windows C:\Windows\System32\Drivers\Etc\hosts). Inoltre, per esigenze di efficienza, **sia il client DNS presente nei dispositivi terminali che il server DNS memorizzano localmente un certo numero di corrispondenze in una memoria cache (caching dei nomi)**, svuotata dopo una certa quantità di tempo (tipicamente un giorno) per evitare che informazioni non aggiornate rimangano nella rete. Questa soluzione è **particolarmente efficace quando**, ad esempio, **un server locale, che ha memorizzato le associazioni IP-nome non di sua competenza e/o gli indirizzi dei server TLD, può aggirare eventuali query a server root**, che possono **richiedere un tempo non trascurabile**.

Navigata a fondo l'infrastruttura esterna del servizio DNS, **ci si chiede cosa un server DNS memorizzi per poter lavorare correttamente**. Si parla, in questa ottica, di **Resource Records (RR)** o **formato RR** facendo riferimento alla **tupla**:

(Nome, Valore, Tipo, TTL)

Dove:

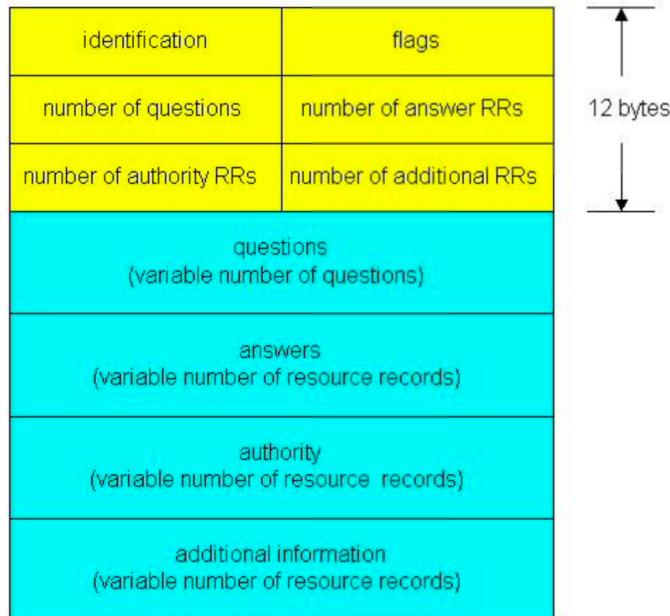
- **TTL** è il tempo residuo di un record scaduto e prossimo ad essere eliminato dalla cache;
- **Tipo**:
 - **A**:
 - Nome, hostname;
 - Valore, indirizzo IPv4;
 - **AAAA**:
 - Nome, hostname;
 - Valore, indirizzo IPv6;
 - **NS**:
 - Nome, dominio (ad esempio, unina.it)
 - Valore, indirizzo IP dell'Autoritative Name Server;
 - **CNAME**:
 - Nome, alias per il nome canonico;
 - Valore, nome canonico;
 - **MX**:
 - Nome, dominio di posta;
 - Valore, nome dell'host mailserver associato a Nome.

Ad esempio, ignorando il TTL:

- relay.bar.foo.com, 145.37.93.192, A;
- www.bar.com, 2001:734:3403:ffffa::7, AAAA;
- foo.com, dns.foo.com, NS;

- foo.com, relay.bar.foo.com, CNAME;
- foo.com, mail.bar.foo.com, MX.

Il protocollo per mandare messaggi DNS prevede che le richieste e le risposte vengano entrambe mandate con lo stesso formato:



Con:

- **Identification**, un numero di 16 bit univoco per ogni coppia richiesta-risposta;
- **Flags**, possono essere:
 - Risposta a richiesta;
 - Ricorsione desiderata;
 - Ricorsione disponibile;
 - Risposta autoritativa;
- **Questions**, nome e tipo per una richiesta;
- **Answers**, RR in risposta ad una richiesta;
- **Authority**, records per server autoritative.

Per **BIND** (Berkley Internet Name Domain) è **una particolare implementazione dei protocolli DNS liberamente re-distribuibile e composta dai seguenti componenti:**

- Un **server DNS** (named);
- Una **libreria per la risoluzione dei nomi di dominio**;
- **Strumenti di diagnostica**.

Questa particolare implementazione è **la più usata in Internet su sistemi Unix-like**. Un esempio di configurazione BIND per un file di zona è il seguente:

```

$TTL 3600
@ IN SOA grid.grid.unina.it. root.grid.grid.unina.it. (
    2004020901      ; Serial
    10800            ; Refresh
    3600             ; Retry
    604800           ; Expire
    86400            ; Minimum TTL
; Machine Name
localhost A 127.0.0.1

vesuvio A 143.225.229.1
grid A 143.225.229.3
honolulu A 143.225.229.111
comicserver A 143.225.229.112
...
; Aliases
www CNAME grid
ftp CNAME grid
news CNAME grid
tesisti CNAME vesuvio
www.tesisti CNAME vesuvio

; MX Record
MX 10 grid.grid.unina.it.

```

* SOA: Start of Authority

Dove:

- **Serial** è un numero seriale progressivo utilizzato per rilevare aggiornamenti del file (di solito si usa il formato aaaammggxx);
- **Refresh** è l'intervallo in secondi tra due successivi prelievi del file di zona da parte di un server DNS;
- **Retry** è l'intervallo in secondi tra tentativi successivi di recupero di una zona in caso di fallimento;
- **Expire** è l'intervallo in secondi che deve trascorrere per ritenere scadute le informazioni di una zona che non si riesce ad aggiornare;
- **Minimum TTL** è il tempo di durata di default delle singole entry del file di zona.

Mentre la configurazione del Reverse DNS:

```

$TTL 3600

@ IN SOA grid.grid.unina.it. root.grid.grid.unina.it. (
    2004020901      ; Serial
    10800            ; Refresh
    3600             ; Retry
    604800           ; Expire
    86400            ; Minimum TTL
; DNS Servers
NS grid.grid.unina.it.

; Machine Name
1 PTR vesuvio.grid.unina.it.
3 PTR grid.grid.unina.it.
111 PTR honolulu.grid.unina.it.
112 PTR comicserver.grid.unina.it.

```

Il file named.root:

```

.
A.ROOT-SERVERS.NET.      3600000   IN   NS    A.ROOT-SERVERS.NET.
                           3600000   A     198.41.0.4
;
; formerly NS1.ISI.EDU
;
.
B.ROOT-SERVERS.NET.      3600000   NS    B.ROOT-SERVERS.NET.
                           3600000   A     128.9.0.107
;
; formerly C.PSI.NET
;

...
.
L.ROOT-SERVERS.NET.      3600000   NS    L.ROOT-SERVERS.NET.
                           3600000   A     198.32.64.12
;
; housed in Japan, operated by WIDE
;
.
M.ROOT-SERVERS.NET.      3600000   NS    M.ROOT-SERVERS.NET.
                           3600000   A     202.12.27.33
; End of File

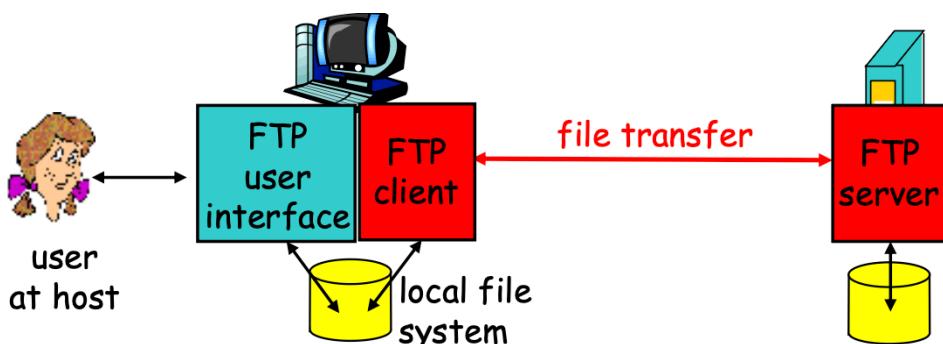
```

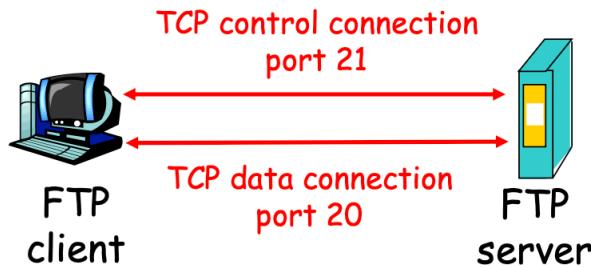
PROTOCOLLI APPLICATIVI: FTP E SMTP

Internet oggi si presenta come una rete ad estensione globale che connette molti milioni di macchine sparse su tutto il globo. Spesso, **sorge l'esigenza di copiare un file da una macchina ad un'altra** per poterlo utilizzare localmente (si pensi ad un documento di Office, un file eseguibile, ecc...); tuttavia, **questa necessità si può presentare sia tra macchine fisicamente connesse**, presenti nello stesso locale, **che tra macchine molto distanti tra di loro**. Proprio per questo è stato definito (in RFC 959) il **File Transfer Protocol (FTP)**, con il quale è possibile **trasferire uno o più file di qualsiasi tipo tra due macchine, indipendentemente dalla loro posizione relativa, utilizzando due connessioni: una connessione dati e una connessione di controllo (out of band)**.

Il principio di funzionamento di FTP si basa sul **modello client/server**: il **client** è l'entità che dà luogo al trasferimento (sia in un senso che nell'altro), mentre il **server** è l'entità remota che è in continua attesa di connessioni FTP da parte di altre entità. Nella pratica, il **client FTP contatta il server FTP sul port 21, aprendo due connessioni parallele**:

- **Connessione di controllo**, dedicata allo scambio di comandi o messaggi di risposta tra il client e il server (controllo “out of band”, fuori banda) e distingue FTP da altri protocolli applicativi (in cui le informazioni di controllo sono trasmesse nello stesso canale di comunicazione, come HTTP);
- **Connessione dati**, dedicata al trasferimento dei file che fluiscono dal client al server o viceversa.





I comandi vengono inviati come testo ASCII sulla connessione di controllo (port 21), così come eventuali messaggi di risposta o richiesta, mentre i dati sono trasferiti mediante un canale apposito. Sapendo che il server mantiene uno stato (come, ad esempio, la directory corrente o i dati dell'autenticazione), si possono fare alcuni esempi di comandi e di codici di stato:

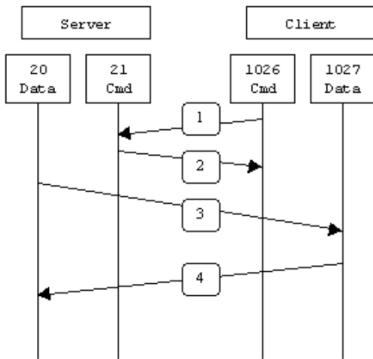
- **Comandi:**
 - USER username;
 - PASS password;
 - LIST, restituisce la lista dei file presenti nella directory corrente;
 - GET filename, preleva il file dalla macchina remota;
 - PUT filename, invia il file alla macchina remota;
- **Codici di stato:**
 - 331 Username OK, password required;
 - 125 Data connection already open, transfer starting;
 - 425 Can't open data connection;
 - 452 Error writing file.

Non è possibile per un client stabilire una connessione FTP verso una qualsiasi macchina, il paradigma adottato (client/server) presuppone, infatti, che il server debba essere stato opportunamente configurato per accettare connessioni. Solitamente, per questioni di sicurezza, le macchine non sono configurate per accettare connessioni di tipo FTP e, pertanto, se si tenta di stabilire una connessione verso una macchina non abilitata, la sessione fallisce e nessun trasferimento risulta possibile. I client FTP sono, invece, disponibili pressoché su tutti i Sistemi Operativi.

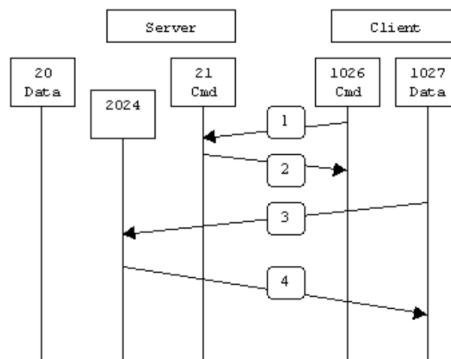
Esistono **due tipologie di FTP**:

- **Active FTP**, per il quale il client apre una connessione di controllo verso l'indirizzo del server sul port 21 utilizzando un ephemeral port e il server una connessione dati dal port 20 verso il client;
- **Passive FTP**, il client apre una connessione di controllo verso l'indirizzo del server sul port 21 usando un ephemeral port e il server sceglie un ephemeral port per la connessione dati, comunicata poi al client, il quale apre la connessione dati sul port indicato dal server.

Active FTP



Passive FTP

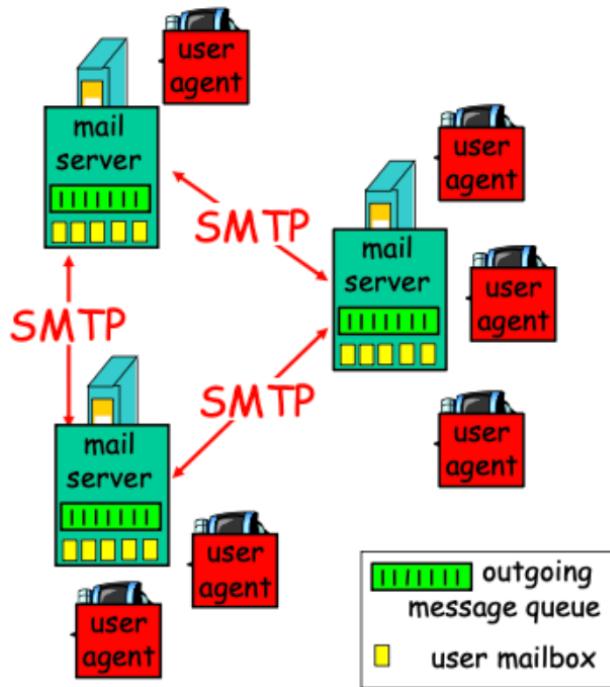


Nel caso del **passive FTP**, è il client ad aprire entrambe le connessioni verso il server, sia per il canale di controllo che di dati. Si noti che l'**active FTP** non è funzionale in scenari moderni, dal momento in cui **difficilmente un ephemeral port** di un client è direttamente raggiungibile dall'esterno; per questo motivo, su molti client FTP moderni c'è la possibilità di passare in modalità **passiva** (mediante apposito comando PASSV), necessaria per far uso oggi giorno di FTP.

Una volta creato, il canale di controllo è fisso e può essere usato per aprire e chiudere molteplici canali dati, che invece hanno durata temporanea per ogni trasmissione voluta; ciò permette ad un server FTP di mantenere uno stato con varie informazioni sulla connessione, come i dati dell'autenticazione o la directory in cui avviene il trasferimento, rendendo FTP un protocollo stateful (a differenza di HTTP). I principali comandi FTP sono USER (per l'username), PASS (per la password), LIST (o LS su Windows, per restituire la lista di file presenti nella directory corrente), GET e PUT (entrambi per il filename ma, rispettivamente, per prelevare e inviare il file alla macchina remota); ad ogni comando inviato, in maniera simile ad HTTP, si ricevono dei codici di stato. Si noti che c'è una differenza tra quali comandi provvede il client e quali sono previsti dal protocollo.

Per far uso di una connessione FTP è innanzitutto necessario avere un account sulla macchina remota e, come presupposto dal paradigma client-server, che il server sia opportunamente configurato per accettare connessioni (solitamente non lo sono per motivi di sicurezza). Sotto queste ipotesi, quando si vuole fornire accesso pubblico ad un filesystem, è possibile ricorrere all'username anonymous privo di password.

Una volta che **una e-mail è stata scritta, è necessario mandarla al destinatario**, il quale potrebbe **non essere disponibile all'immediata consegna del messaggio** (l'utente è impegnato in altre attività o il computer è spento). **La posta elettronica sfrutta degli intermediari per il trasferimento delle e-mail tra le parti**, alla stregua degli uffici postali che ospitano pacchi in attesa che il destinatario venga a ritirarli; questo sistema di intermediari non sarebbe possibile senza il **Simple Mail Transfer Protocol (SMTP)**, definito in RFC 821 e **appositamente sviluppato**, come suggerisce il nome, **per gestire il trasferimento dei messaggi di posta elettronica**.



In gioco ci sono tre entità: gli user agents, i mail servers e il protocollo stesso. Per user agent, o mail reader, si intende il responsabile della composizione, modifica e lettura dei messaggi e dell'entrata e uscita dei messaggi immagazzinati sul server tramite il protocollo (che può essere usato anche tra user agent e server durante l'invio di una mail). Per mail server si intende la mailbox contenente messaggi in entrata (non letti) per l'utente e la coda dei messaggi in uscita, contenente messaggi non ancora recapitati; SMTP è usato principalmente per l'interazione tra due mail server:

- “client”, il mail server mittente;
- “server”, il mail server destinatario.

Un mail server funge in momenti diversi da “client” e da “server” a seconda del ruolo che ricopre nello scambio del messaggio.

Prima di inviare un messaggio, un client deve far uso di un user agent che si occupa di inoltrare tale messaggio alla coda del mail server mittente, il quale, poi, si occupa di trasferire i messaggi dalla propria coda alla mailbox del mail server destinatario. SMTP usa il protocollo TCP (sul port 25) per consegnare in modo affidabile messaggi dal “client” al “server” con un trasferimento diretto dal server mittente al server destinatario **in tre fasi**:

1. **Handshaking**;
2. **Trasferimento** del messaggio;
3. **Chiusura** della connessione.

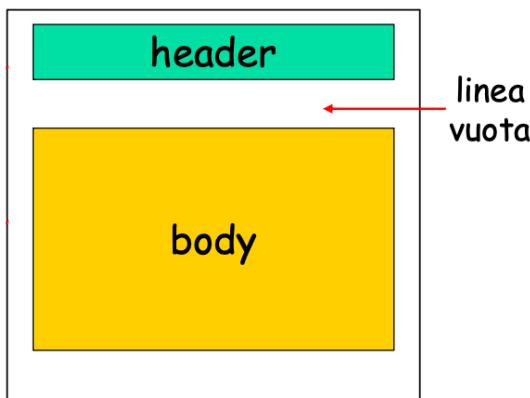
L’interazione avviene secondo il modello comando/risposta in formato, rispettivamente, testo ASCII e codice di stato e descrizione (quest’ultima, facoltativa), e con un tipo di connessione persistente. In entrambi i casi, i messaggi (eventualmente comprensivi di contenuto) sono codificati con caratteri ASCII a 7 bit. Alcune combinazioni di caratteri, però, non sono ammesse (come CRLF, usato per determinare la fine un messaggio), quando appaiono, il messaggio va opportunamente codificato.

Di seguito è proposto un esempio di interazione client → server:

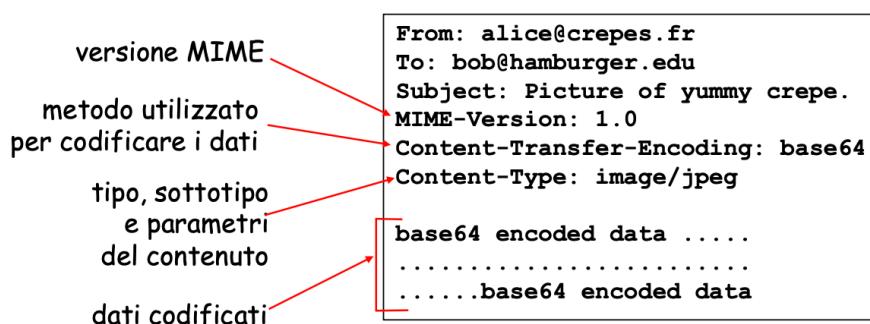
```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Il formato di un messaggio SMTP prevede un **header** ed un **body**, separati da una riga vuota, in cui:

- **Header:**
 - To;
 - From;
 - Subject;
 - ... (differenti da comandi SMTP);
- **Body:**
 - Il “messaggio” vero e proprio in soli caratteri ASCII.



Le **righe aggiuntive** nell'intestazione **informano la presenza di un body MIME** (Multipurpose Internet Mail Extensions, definito in RFC 2045-2056), utilizzato per inserire file o far uso di codifiche diverse:



Una **codifica spesso usata nei MIME** per fare in modo che si possa formattare il testo in maniera differente è **HTML**, con il quale è necessario inviare due messaggi: un primo messaggio di solo testo (plain text) ed un secondo codificato in HTML mediante MIME; in questo modo, client che non riescono a decodificare HTML possono comunque visualizzare il messaggio di solo testo, senza dover visualizzare un messaggio HTML incomprensibile all'utente. Di base una parte MIME permette di codificare un solo tipo di dato, nonostante sia possibile la codifica di più tipi di dato ricorrendo ad un marcatore di separazione per dividere in più parti MIME il messaggio.

Esempi di tipi MIME sono:

- Testo:
 - Sottotipi: plain, html;
- Image:
 - Sottotipi: jpeg, gif;
- Audio:
 - Sottotipi: basic (8-bit mu-law encoded), 32kadpcm (32 kbps coding);
- Video:
 - Sottotipi: mpeg, quicktime;
- Application:
 - Altri dati che devono essere processati da specifiche applicazioni;
 - Sottotipi: msword, octet-stream.

Di seguito è proposto un esempio di mail MIME:

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789

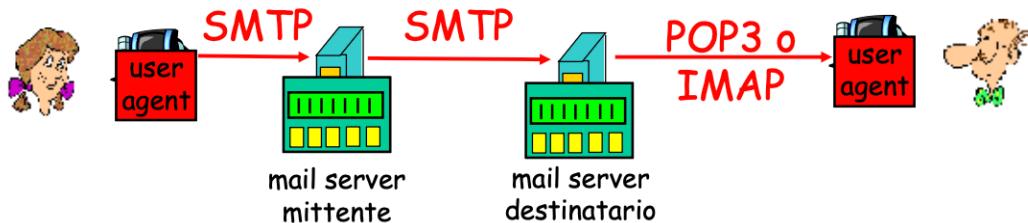
--98766789
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain

Dear Bob,
Please find a picture of a crepe.
--98766789
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.....
.....base64 encoded data
--98766789--
```

Fino ad ora è stato mostrato come sia possibile trasferire messaggi tra i vari mail server ma non è stato ancora menzionato il meccanismo per il quale un utente possa, in un momento qualsiasi, accedere alla propria casella di posta elettronica per leggere i propri messaggi. Per questo tipo di operazione **SMTP non è un protocollo sufficiente** e bisogna introdurre un nuovo protocollo, **POP3** (Post Office Protocol – versione 3), definito in RFC 1939. Si tratta sempre di un **protocollo**

client/server, con l'user agent che gioca (ancora una volta) **il ruolo di client POP e il mail server quello di server POP**. In totale, la catena dei protocolli per la posta elettronica assume la seguente forma:



Con:

- **Consegna di messaggi:**
 - SMTP;
- **Protocolli di accesso alla mail e recupero dei messaggi dal server:**
 - POP:
 - Autorizzazione (agent ↔ server) e download;
 - IMAP (Internet Mail Access Protocol, definito in RFC 2060):
 - Più complicato e potente;
 - Manipolazione avanzata dei messaggi sul server;
 - HTTP:
 - Gmail, Hotmail, Yahoo! Mail, ecc...

Un **dialogo POP3** ha la seguente **forma**:

- **Autorizzazione:**
 - **Comandi del client:**
 - user: specifica l'username;
 - pass: specifica la password;
 - **Il server risponde:**
 - +OK;
 - -ERR;
- **Fase di scambio:**
 - **Comandi del client:**
 - list: visualizza la lista dei messaggi;
 - retr: preleva il messaggio per numero;
 - dele: elimina il messaggio dal server;
 - quit: chiude la sessione.

```

S: +OK POP3 server ready
C: user alice
S: +OK
C: pass hungry
S: +OK user successfully logged on

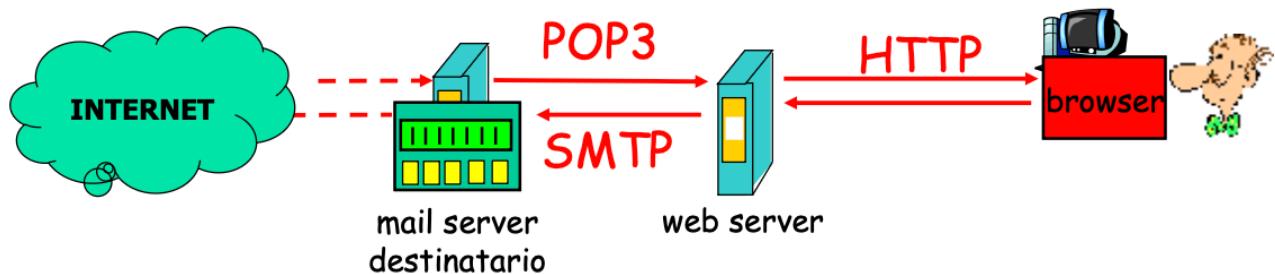
```

```

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off

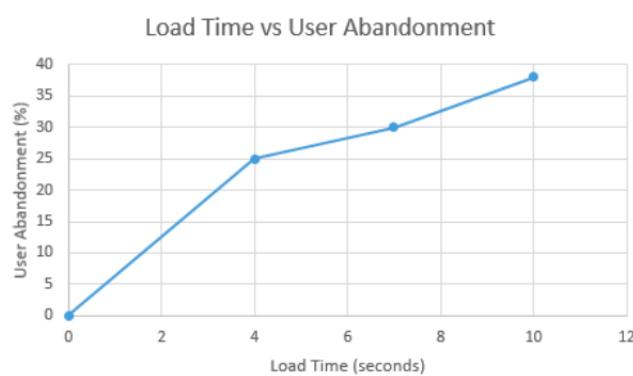
```

Molti siti web forniscono accesso alle proprie caselle di posta (Gmail, Hotmail, Yahoo!, ecc...), in modo da **non aver bisogno di un user agent installato e correttamente configurato** per ricevere ed inviare posta, è sufficiente disporre di un browser. In queste situazioni, è il **web server** che si occupa di fare da mail server per l'utente.



CONTENT DELIVERY NETWORKS

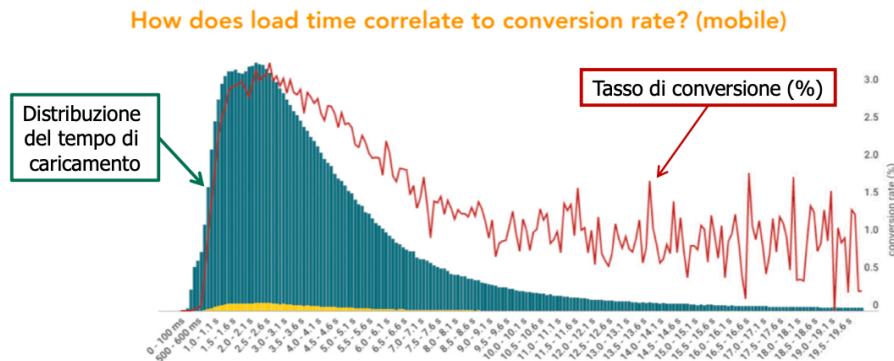
Per un sito web, un parametro particolarmente importante è il **Time To Interact (TTI)**, ovvero il tempo che impiega la pagina web per rendere i suoi contenuti fruibili all'utente; secondo una ricerca di Radware, il **57% dei consumatori online abbandona una pagina web di un sito di e-commerce che richieda più di 3 secondi per caricarsi**, mentre se il tempo di caricamento di una pagina web (generica) supera i 10 secondi, circa il 40% degli utenti rinuncia a proseguire la navigazione.



Il tasso di conversione (o conversion rate) di un sito di e-commerce è il rapporto percentuale di visite al sito che si traduce in una transazione commerciale (una vendita):

$$CR = \frac{\#Sales}{\#Visits}$$

Il tasso di conversione è fortemente influenzato dal tempo di caricamento delle pagine web del sito, in particolare per l'accesso da terminali mobili (Amazon stima che un aumento della latenza media di 100ms comporti una perdita di ricavi dell'1%):



Per **Content Delivery Network** si intende un'infrastruttura creata per distribuire efficacemente agli utenti di Internet i contenuti dei siti web più popolari, basata sulla **distribuzione di repliche** dei contenuti dal server principale del “Content Provider” ad una molteplicità di server disposti sulla rete da un “Content Delivery Operator”. Si presenta come un servizio a pagamento del quale usufruiscono i gestori dei siti web commerciali più popolari.

Una CDN ha i seguenti obiettivi:

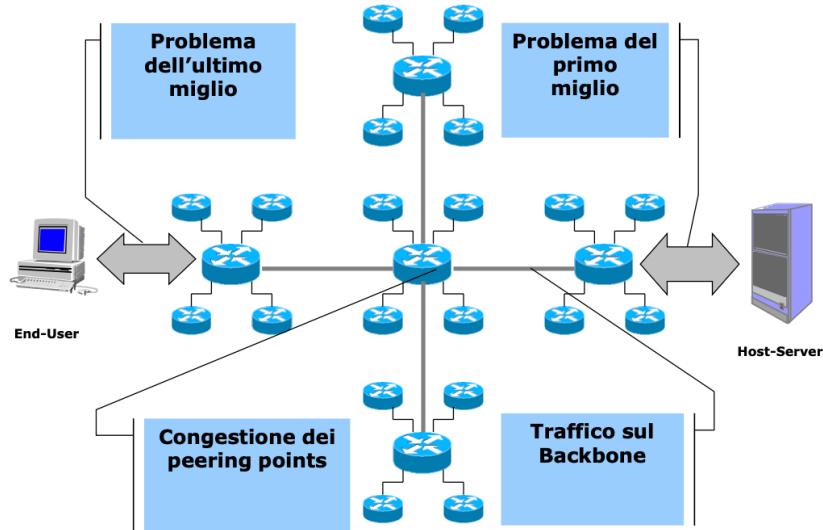
- **Alleviare il server web “master” dal carico degli utenti**, in particolare proteggerlo da picchi di traffico improvvisi (flash crowds);
- **Offrire i contenuti ai singoli utenti tramite server collocati in prossimità degli utenti** (alla periferia della rete);
- **Rendere il sistema di distribuzione dei contenuti più affidabile e robusto ai guasti**.

Le interazioni che determinano il tempo di accesso ad una pagina web sono di seguito riassunte:



- User enters www.xyz.com
 - Browser requests IP address for www.xyz.com
 - DNS returns IP address
- Browser requests HTML
- Content provider's web server returns HTML
- Browser obtains IP addresses for hostnames listed in URLs of objects embedded on page
- Browser requests embedded objects
- Content provider's web server returns embedded objects

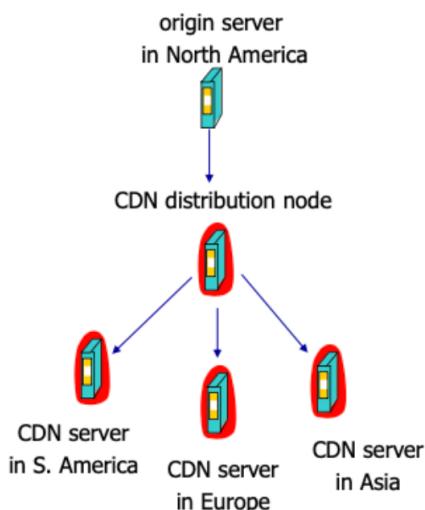
Questo procedimento avviene, come noto, **non direttamente ma su una rete distribuita**, sia per l'uso di routing gerarchico tra più sistemi autonomi (AS) sia per la natura distribuita dei DNS. Un approccio centralizzato, cioè senza l'impiego di CDN, porta con sé **diversi limiti**, tra cui:



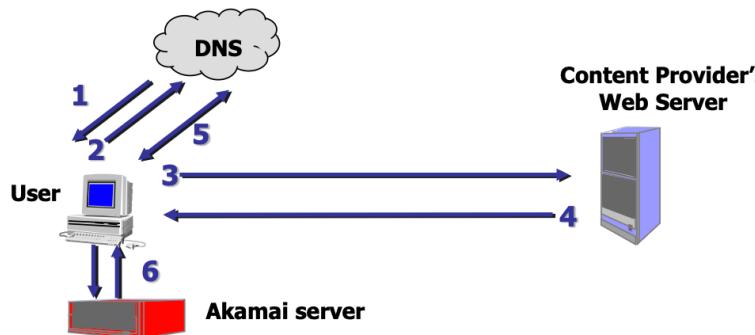
Tramite un'infrastruttura, spesso privata, le **CDN distribuiscono in maniera capillare i contenuti di uno specifico Content Provider**, usando forme proprietarie di caching basate su una **complessa gestione del DNS**, caratterizzata, tra l'altro, dalla **conoscenza dell'indirizzo IP del Client**. Le **CDN offrono ai Content Provider la possibilità di raggiungere**, con una certa QoS, **una vasta utenza e, d'altra parte, propongono a ISP di medie e grandi dimensioni di collaborare**, spesso gratuitamente, **alla loro struttura**.

In questo modo, le **CDN ottimizzano l'uso di risorse di Internet avvicinando il contenuto all'utente**, beneficiando gli utenti, gli ISP e i Content Provider tramite bassa latenza, bassi costi, raggiungibilità e protezione da flash events. **Nel 2014 il mercato delle CDN era stimato valere circa 3,71mld\$**.

Come già anticipato, le **CDN funzionano tramite replicazione ed aggiornamento dei contenuti**: la CDN replica il contenuto del proprio customer (ovvero il Content Provider) sui propri CDN server, aggiornati quando il provider aggiorna il contenuto.

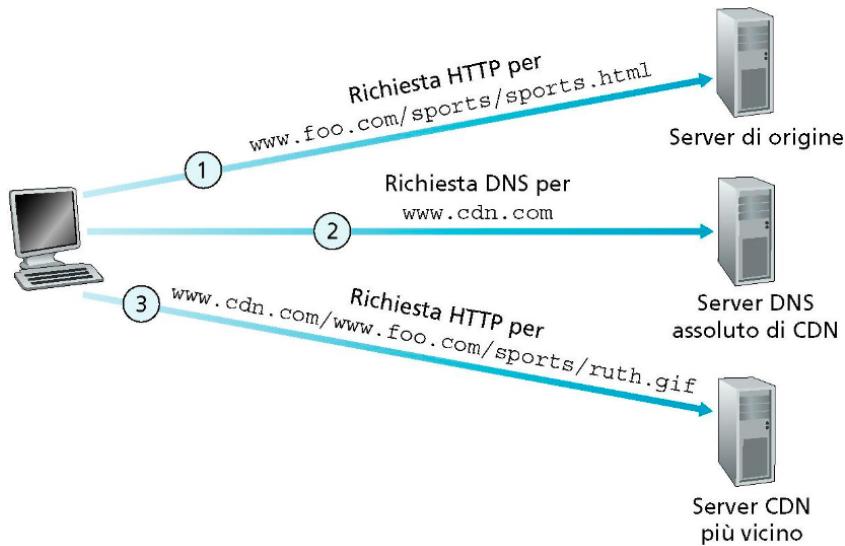


Un esempio di funzionamento di un CDN è quello di **Akamai** (azienda fondata nel 1998 da Tom Leighton e Danny Lewin, ricercatori del MIT, e leader nel settore delle CDN con più di 240.000 server in più di 130 paesi per più di 1700 network):



- User enters www.xyz.com and browser requests IP address for www.xyz.com
- DNS returns IP address
- Browser requests HTML
- Content provider's web server returns page with **Akamaized URLs**
- Browser obtains IP address of optimal Akamai server for embedded objects
- Browser obtains objects from optimal Akamai server

Il routing delle richieste, passando dal server origine al nodo, **prevede che la CDN crei una mappa** che indichi le distanze tra i vari ISP e i propri nodi: quando arriva una query al DNS aut, si determina l'ISP che ha originato la query e si usa la mappa per la scelta del server CDN più vicino.



Il progetto **Google AMP** (Accelerated Mobile Pages Project) è un progetto reso pubblico ad ottobre 2015 e **costruito su un framework** (basato su tecnologie web esistenti) **che consente di creare pagine web leggere e visualizzabili rapidamente da dispositivi mobile**. Il progetto si basa su **tre componenti principali**:

- **AMP HTML**, un HTML con alcune restrizioni;
- **AMP JS**, una libreria che consente l'esecuzione di codice JavaScript con delle limitazioni al fine di ottenere un rendering veloce delle pagine;
- **AMP Cache**, un sistema di web cache che può essere usato per servire pagine AMP HTML precedentemente memorizzate (attualmente, ci sono due provider AMP Cache: Google AMP Cache e Cache AMP Cloudflare).

Poiché anche il browser lavora per ridurre il TTI, mostrando, ad esempio, una pagina web nonostante gli oggetti della stessa non siano stati caricati correttamente, un modo efficace con cui una CDN può lavorare consiste nel fare in modo che l'HTML richiesto dal browser del client (dopo aver ottenuto l'indirizzo IP del web server dal DNS) faccia riferimento non al web server, bensì a server più vicini al client e gestiti dal CDN.

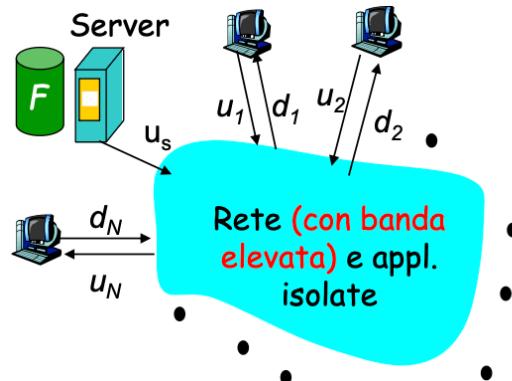
APPLICAZIONI PEER TO PEER

P2P (Peer to Peer) è un modello alternativo al client/server per il quale non esistono server sempre connessi (always-on servers) e neanche alcuna differenziazione funzionale tra client e server, gli end system (peer) comunicano direttamente, sono connessi ad intermittenza e cambiano il proprio IP.

Ci si chiede quale sia il tempo necessario a distribuire un file da un server a N peer (tempo di distribuzione, D) nel caso client/server, sapendo che u_s è la bandwidth del server in upload, u_i la bandwidth dei peer in upload e d_i in download. Il server invia N copie del file, con un fattore limitante che può essere u_s o d_i : se il fattore limitante è u_s , allora il tempo richiesto è inversamente proporzionale a tale parametro, altrimenti è inversamente proporzionale al più piccolo dei d_i . In particolare:

$$D_{CS} = \max \left\{ N \frac{F}{u_s}, \frac{F}{\min_i d_i} \right\}, \forall i \in (0, N]$$

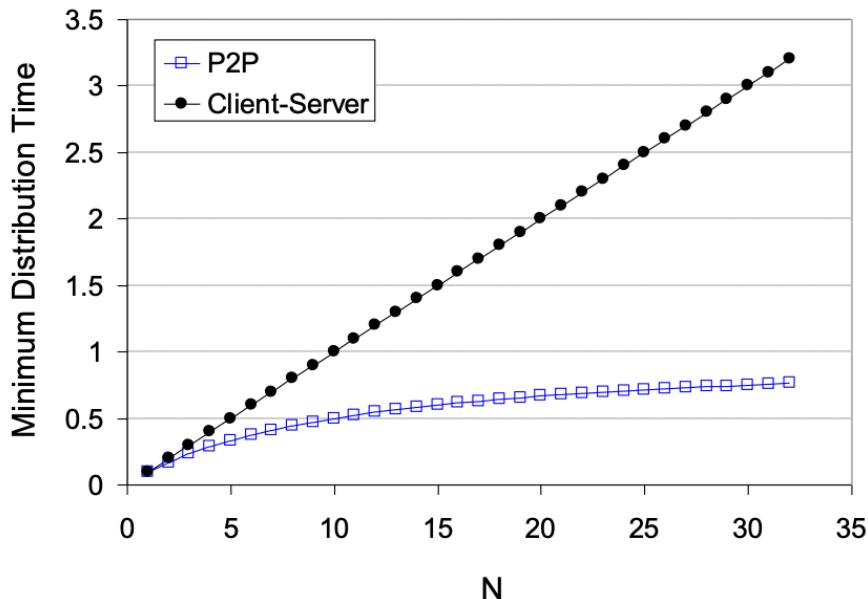
Chiaramente, per N elevato (quindi una quantità crescente di peer a cui distribuire il file), il termine NF/u_s è, a priori, predominante, rilevando un incremento lineare al crescere di N.



Si noti che il termine relativo al tempo di download più basso fra i client è lo stesso nel caso P2P, in cui il tempo dipende dal numero di peer coinvolti ma anche dall'ordine con cui il contenuto è trasferito, il server deve inviare almeno una copia, quindi, il tempo è al meno F/u_s , il client più lento impiega F/d_{min} , mentre NF bit totali devono essere scaricati a velocità massima di upload $u_s + \sum u_i$, per un tempo di distribuzione:

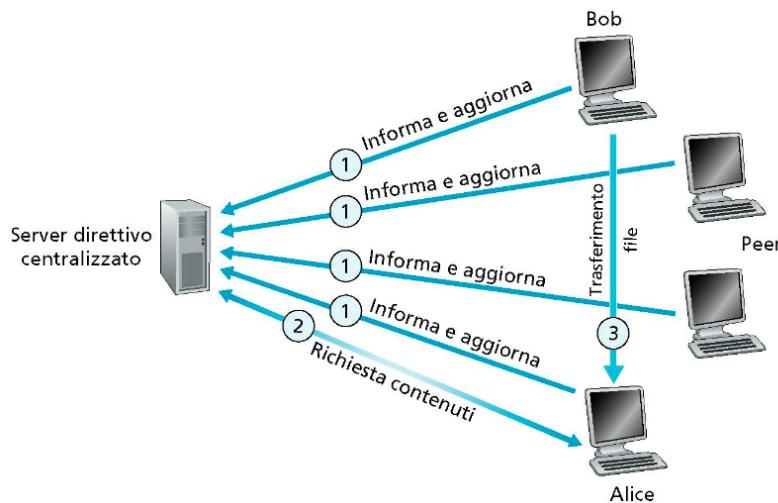
$$D_{P2P} = \max \left\{ \frac{F}{u_s}, \frac{F}{\min_i d_i}, \frac{NF}{u_s + \sum u_i} \right\}$$

Sotto le ipotesi di rate di upload costante per tutti i peer e con $F/u = 1h$, $u_s = 10u$ e $d_{min} \geq u_s$:



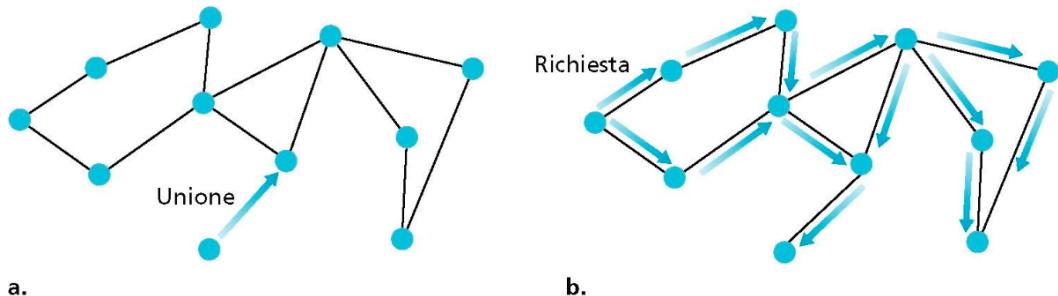
Secondo il **modello P2P**, tutti i peer possono essere server, garantendo un'architettura altamente scalabile. Ad esempio, un utente A è connesso ad Internet e ha lanciato la sua applicazione di file sharing P2P sul proprio PC (A non è perennemente connesso ad Internet, non ha un hostname e, pertanto, il suo indirizzo IP cambia ad ogni connessione), richiede una copia di un file e l'applicazione gli restituisce tutti gli altri peer che hanno una copia di quel file. Dopo che A ha scelto uno dei peer, l'utente B, il file è trasferito (copiato) dal PC di B al proprio usando HTTP ma, mentre effettua il download, altri peer possono prendere altri file da A, fungendo sia da client che da server.

Un modello alternativo di P2P è quello con **directory centralizzata** (usato, ad esempio, da Napster): quando un peer si connette alla rete, si collega ad un server centralizzato fornendo il proprio indirizzo IP e il nome degli oggetti resi disponibili per la condivisione, mentre in server sono raccolte le informazioni (aggiornate dinamicamente) sui peer attivi.



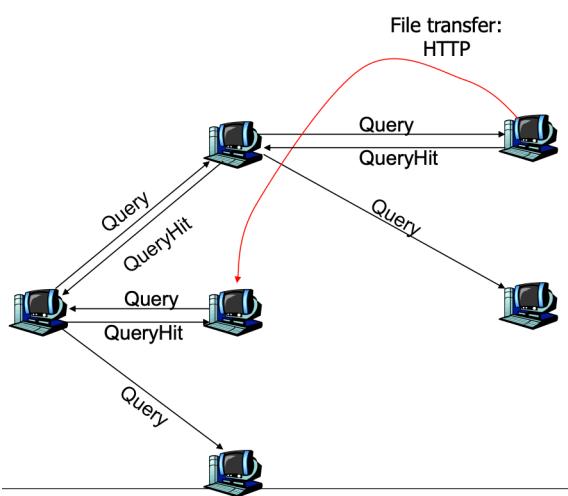
Questo approccio, però, presenta diverse criticità: **dispone di un Single Point Failure**, rappresenta un **collo di bottiglia per le prestazioni** e rischia di violare le leggi sul diritto d'autore in alcuni domini (come è accaduto a Napster). Il trasferimento dei file è decentralizzato ma la localizzazione dei contenuti è pesantemente centralizzata.

Il P2P con architettura decentralizzata (usato, ad esempio, da Gnutella) è realizzato con un'architettura completamente distribuita (non ci sono server centrali) e viene realizzata con una rete sovrapposta (overlay network) fatta da connessioni TCP tra peer, con una struttura paritetica. Nonostante la rete possa avere centinaia di migliaia di peer, ognuno è connesso al massimo a dieci altri peer nell'overlay. Questo modello presenta, però, due problemi: il come viene costruita e gestita la rete di peer e il come un peer localizza il contenuto.

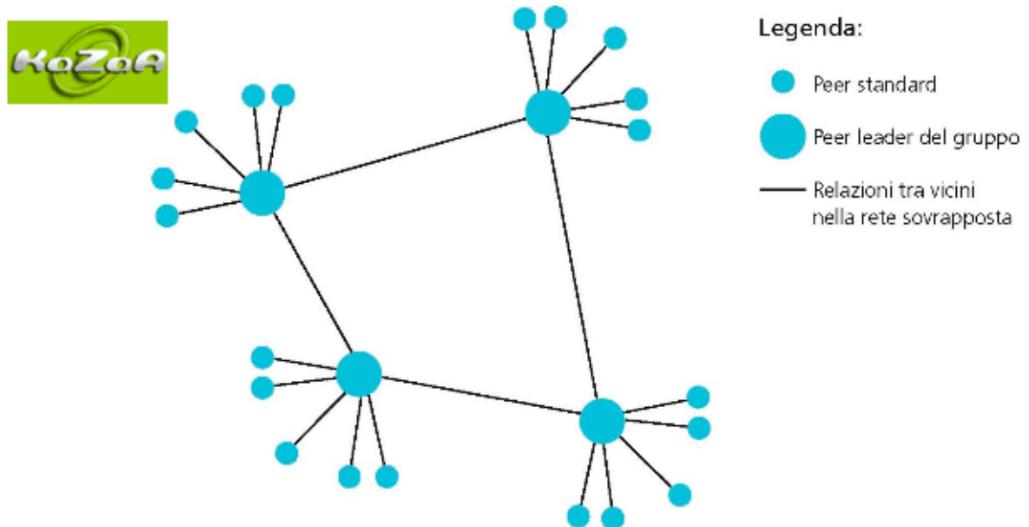


I peer, una volta unitisi alla rete, inviano richieste mediante la tecnica del flooding (inondazione), detta anche **query flooding**, a raggio limitato (ovvero con uno scope limitato a 7 peer, ad esempio), basato su Query e QueryHit.

Il peer X deve trovare altri peer già parte dell'overlay, mantenendo una lista di IP o contattando un sito Gnutella contenente la lista; dopo l'accesso alla lista, X tenta di impostare una connessione TCP con i peer della lista, fermandosi quando si connette ad un peer Y. X spedisce, poi, un ping Gnutella a Y, che lo inoltra finché il contatore non si azzera; tutti i peer che ricevono un messaggio di ping, rispondono con un messaggio di pong contenente l'indirizzo di chi ha inviato il pong, il numero di file in condivisione e la dimensione totale. Quando, infine, X riceve i messaggi di pong, avendo l'IP, può impostare una connessione TCP con alcuni di essi per prelevare il file desiderato. Secondo questo approccio, possono coesistere più fasi di bootstrap in parallelo.



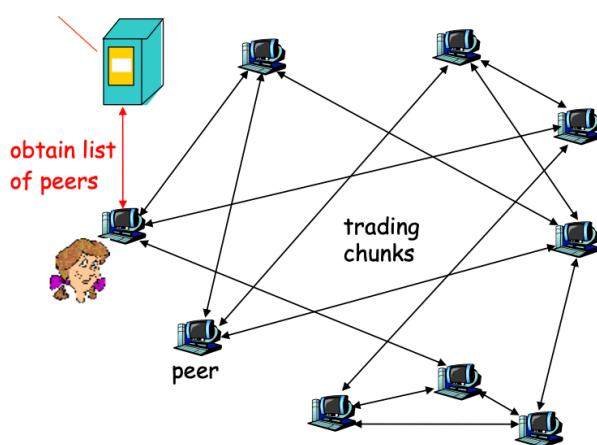
Un altro tipo di P2P con directory decentralizzata (usata, ad esempio, da KazaA) sfrutta quelli che sono i punti di forza delle reti di Napster e di Gnutella per far in modo che ogni peer sia associato ad un **group leader** (che funga da **mini hub**), il quale è esso stesso un peer. Il group leader memorizza le informazioni in condivisione dei figli ed è in grado di interrogare altri group leader, con un query flooding che si applica ad una quantità minore di nodi.



In fase di bootstrap, un peer che si connette deve essere associato ad un (o designato) group leader, mentre l'overlay è costituita da connessioni TCP tra peer e group leader e tra coppie di group leader. Ogni file possiede un identificatore hash e un descrittore. I peer (client) spediscono le query al proprio group leader, il quale risponde per ogni richiesta con l'indirizzo IP del detentore della risorsa, l'hash, e dei metadati ad essa associati, per poi selezionare la risorsa file per il download e inviare una richiesta HTTP al detentore della risorsa usando come identificatore l'hash. In tutto ciò, il group leader inoltra sia le richieste sia le eventuali risposte da parte di altri group leader.

Per migliorare le prestazioni, è possibile effettuare l'accodamento delle richieste e la limitazione del numero degli upload simultanei, inserire un sistema di priorità di incentivo e permettere il download parallelo di parti dello stesso file da più utenti.

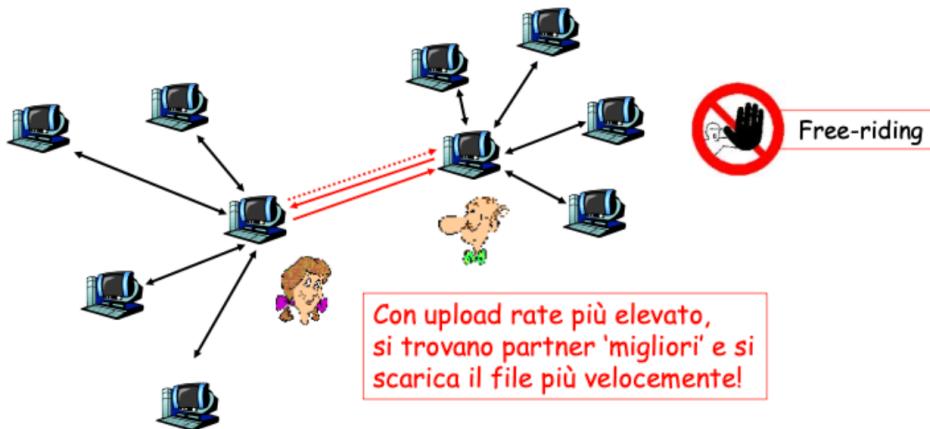
BitTorrent, invece, usa un P2P file distribution strutturato attorno a tracker e swarm: un tracker è un nodo della rete che si occupa di tenere traccia dei peer che compongono un torrent, mentre un swarm è un gruppo di peer che si scambiano porzioni (chunk, 256KB) dello stesso file.



Quando un peer si aggiunge ad uno swarm si registra presso il tracker per avere la lista dei peer e si connette ad un sottoinsieme di tali peer (neighbours). Durante il download, il peer esegue l'upload di chunk verso altri peer, i quali possono disattivarsi o attivarsi dinamicamente. Una volta scaricato l'intero file, il peer può (egoisticamente) abbandonare o (altruisticamente) rimanere nello swarm. Un nodo, quindi, può:

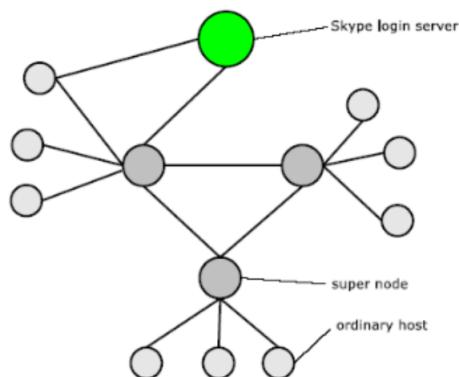
- **Prelevare un chunk** (tecniche rarest first):
 - Peer differenti posseggono differenti sottoinsiemi di chunk di un file;
 - Periodicamente, un peer chiede a tutti i neighbours la lista dei chunk in loro possesso ed invia le richieste per i propri chunk mancanti;
- **Invio di chunks** (tecniche tit-for-tat):
 - L'idea di fondo è quella di dare priorità a chi fornisce dati al rate più alto;
 - Un peer invia chunk ai quattro neighbours attualmente più veloci (che gli inviano chunk al rate più elevato), ricalcolati ogni 10 secondi;
 - Ogni 30 secondi si seleziona in maniera casuale un altro peer e si inizia ad inviargli chunk;
 - Il peer appena scelto può essere aggiunto ai top 4;
 - Questo procedimento è detto optimistically unchoke.

Ad esempio, per la tecnica tit-for-tat, un utente A effettua l'unchoke ottimistico di un utente B, diventa un top-four provider per B, il quale “ricambia” diventando uno dei top-four di A.



Skype è un'applicazione P2P VoIP sviluppata da KazaA nel 2003 che **supporta instant messaging e conferencing**, con un **protocollo proprietario**, e che impiega una rete overlay con tre tipi di host:

- **Host ordinari**, Skype users;
- **Super nodi**, Skype users con sufficiente CPU, memoria, banda, ...;
- **Server di login** per l'autenticazione.



Ciascun client Skype mantiene una **lista di indirizzi IP di super nodi conosciuti**, inizialmente vuota, e, attraverso uno di essi, si connette alla rete. I super nodi sono **responsabili della localizzazione**

degli utenti, del routing delle chiamate e del mantenimento delle informazioni circa gli host connessi alla rete Skype.

Per la connessione ai super nodi si distinguono:

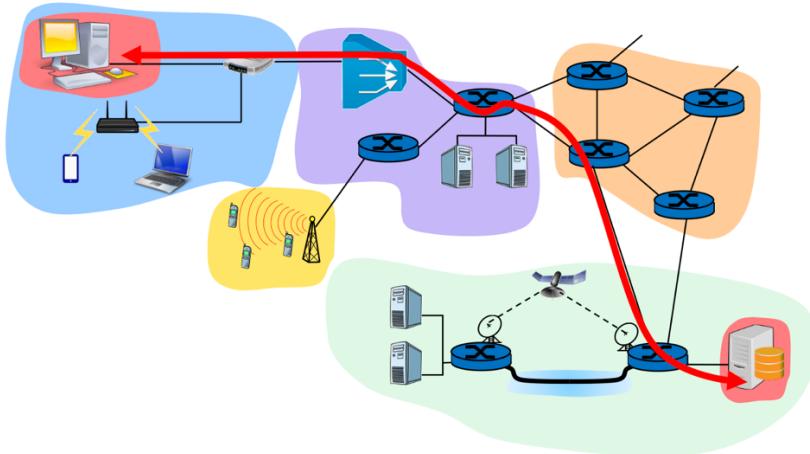
- **Primo login:**
 - Alla prima esecuzione dopo l'installazione, un client Skype comunica con il server centrale (skype.com), durante la quale la cache dell'host è popolata di 7 indirizzi IP di super nodi da usare per il bootstrap;
 - A questo punto, l'host può contattare uno di questi super nodi per il join e far partire parte la fase di autenticazione con username e password con il server Skype;
 - L'host viene periodicamente aggiornato con indirizzi IP di nuovi super nodi;
- **Login successivi:**
 - Un client sceglie uno degli indirizzi dei super nodi e stabilisce la connessione.

Il P2P si può applicare anche per le cosiddette P2P IPTV, con ogni peer sia viewer che server (alcuni dei quali ricevono il flusso direttamente dalla sorgente originale, il server IPTV, altri invece solo attraverso altri peer).

IL LIVELLO RETE E I PROTOCOLLI AUSILIARI

IL LIVELLO RETE E IL PROTOCOLLO IP

In una rete di computer ottenuta tramite l'interconnessione di reti distinte (internetwork), **il compito del livello di rete è quello di definire i percorsi dei pacchetti nel loro transito da host mittente a host destinatario.**



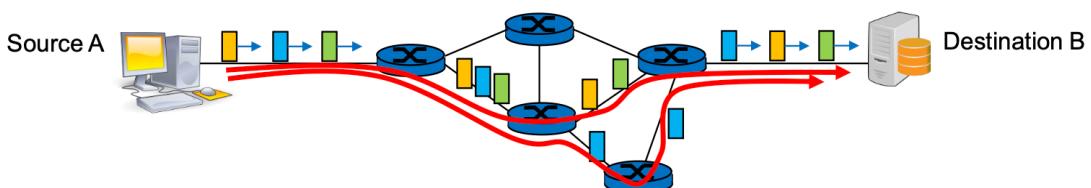
Oltre agli end systems, **a livello di rete operano i router**, dispositivi progettati per realizzare l'internetworking in sé e per sé.

Le reti di calcolatori così ipotizzate **operano secondo il modello del packet switching** (o commutazione di pacchetto), per la quale **l'informazione è trasmessa sotto forma di pacchetto** formato da **intestazione** (header) e **contenuto** (payload), **il primo contiene informazioni di controllo** (come indirizzo di destinazione). In uno stack protocolare (protocol stack), **ciascun layer aggiunge il proprio header al pacchetto** (L2 si sa che aggiunge anche un trailer), risultando in un insieme di header di protocolli diversi che **incapsulano un payload** (sapendo, però, che **ogni header non sa che c'è un header prima e dopo di sé stesso**).



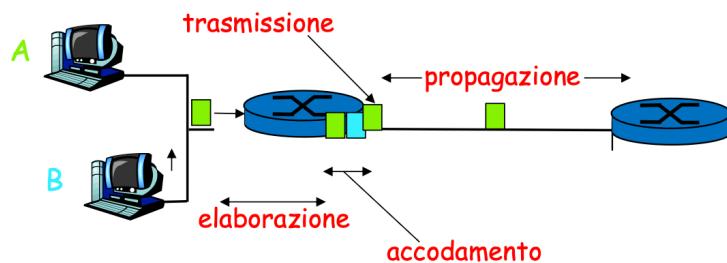
I dispositivi intermedi che operano a livello di rete funzionano in una modalità detta **store-and-forward**: **ogni pacchetto è ricevuto interamente, se ne controlla l'assenza di errori e se ne opera la ritrasmissione su un link di uscita, conservandoli in un buffer gestito con una coda nel caso di eventuale traffico.**

In una rete a commutazione di pacchetto basata sul **modello a datagrammi**, **ciascun pacchetto è inoltrato verso la sua destinazione indipendentemente dagli altri**: ogni volta che **un pacchetto arriva ad un router, il dispositivo lo inoltra verso un successivo intermediario** (o verso il destinatario finale qualora questo sia direttamente raggiungibile). **I pacchetti inviati da un terminale A verso un terminale B in momenti successivi possono seguire percorsi differenti nella rete e, quindi, arrivare a destinazione in un ordine diverso da quello di partenza** (o non arrivare proprio).



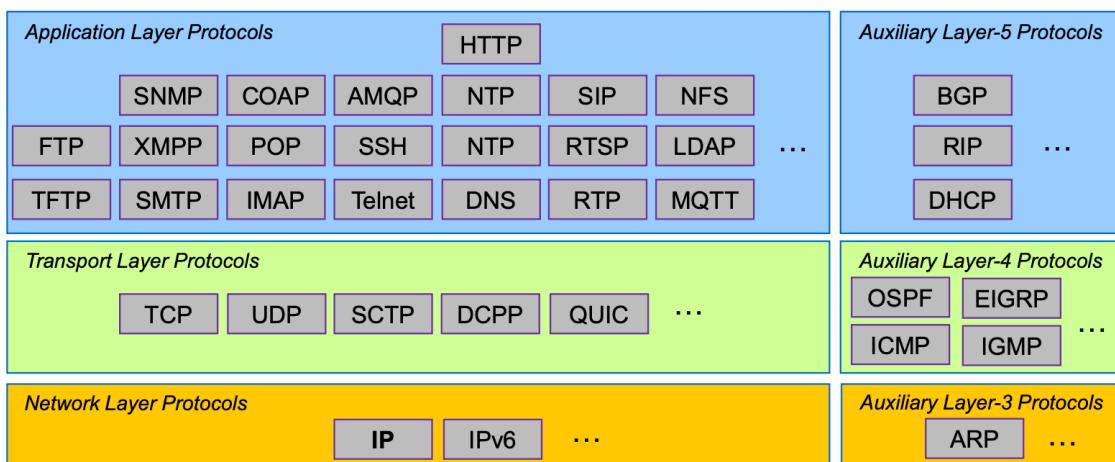
Il servizio offerto da questo tipo di rete consiste nel **recapitare pacchetti da un qualunque terminale mittente ad un qualunque terminale destinatario** ma la sua **Qualità del Servizio (QoS)** viene misurata sulla base di una molteplicità di “**indici di valutazione**”, quelli più comunemente utilizzati sono:

- **End-to-end delay**, definito come il ritardo nella consegna dei pacchetti [s] e determinato da:
 - Tempo di elaborazione del nodo (controllo di errori, determinazione di link di uscita, ecc...);
 - Tempo di trasmissione su ciascun link (Lunghezza in bit / velocità in bps);
 - Tempo di attesa nelle code dei router (variabile);
 - Tempo di propagazione sulle linee (Lunghezza della linea / velocità del segnale).



- **Packet delay variation (PDV)**, definita come la variazione temporale del ritardo one-way (spesso indicata anche con il termine packet jitter);
- **Throughput**, definito come la quantità di bit al secondo che la rete è in grado di trasferire tra i due terminali [b/s];
- **Loss rate**, definito come la probabilità che un pacchetto non venga consegnato a destinazione.

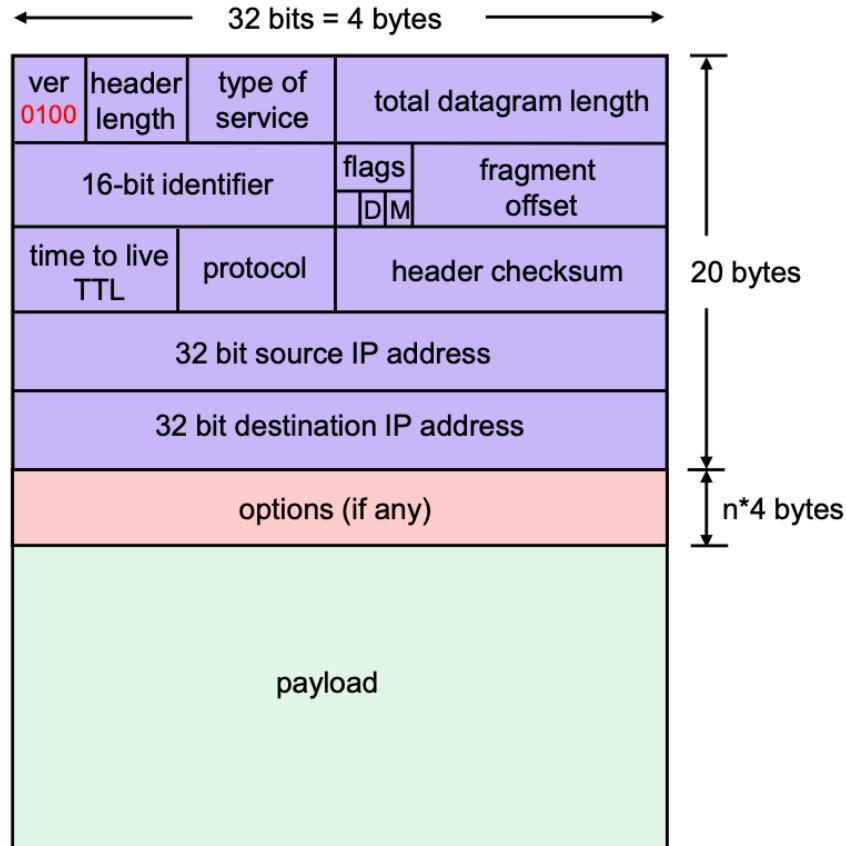
Nella rete Internet, la funzione principale del livello di rete è svolta dal protocollo IP, ancora oggi prevalente la versione 4 nonostante molti dispositivi si stiano adattando alla 6. La caratteristica principale del protocollo IP è quella di **offrire un servizio di consegna elementare e senza garanzie (best effort) di pacchetti**, il che la rende **adattabile ad un'ampia varietà di tecnologie di livello inferiore**.



Al di sopra di IP, nello stack TCP/IP (detta Internet Protocol Suite), operano i protocolli di livello trasporto (UDP e TCP). Il protocollo IP gestisce indirizzamento, frammentazione, riassemblaggio e multiplexing dei protocolli, ed è implementato sia nei terminali che nei router.

Inoltre, è responsabile della scelta dell'interfaccia sulla quale un pacchetto deve essere trasmesso per arrivare a destinazione.

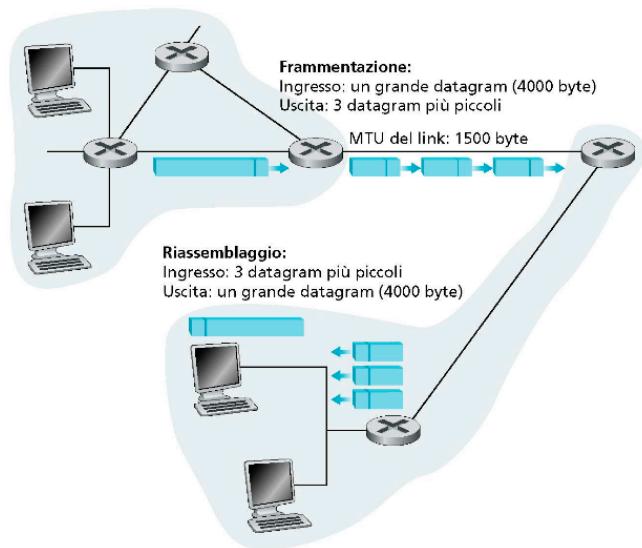
Un **datagramma** (o pacchetto) IPv4 può avere una **dimensione massima di $2^{16} - 1$ byte** (65535 byte) ed è **costituito da un header ed un payload**:



- L'header è costituito da una parte a struttura fissa (20 byte) ed una opzionale:
 - **Campo versione** (4 bit), pari a 0100 per la versione 4;
 - **IP header lenght** (4 bit), lunghezza dell'header in multipli di 32 bit (max 60 byte);
 - **Type-of-Service** (8 bit), specifica il tipo di servizio che si richiede alla rete ed è usato, in pratica, per scopi differenti;
 - **Total lenght** (16 bit), indica la lunghezza in byte dell'intero pacchetto (header + payload);
 - **Time-to-live** (TTL, 8 bit), è un numero residuo di router attraversabili che viene decrementato di 1 ad ogni router, in modo che a 0 il pacchetto venga scartato, e serve in caso di percorsi circolari (loop) ad evitare che un pacchetto resti perennemente in circolo;
 - **Protocol** (8 bit), indica il protocollo di livello superiore associato al payload (6 indica TCP, 17 UDP) e serve al demultiplexing dei pacchetti a destinazione;
 - **Header checksum** (16 bit), serve a verificare l'integrità dell'header IP;
 - **Source IP Address** (32 bit), indirizzo IP del nodo mittente del pacchetto;
 - **Destination IP Address** (32 bit), indirizzo IP del nodo destinatario del pacchetto;
 - **Identification** (16 bit), Flags (3 bit), Fragment Offset (13 bit), sono usati in caso di frammentazione del pacchetto da parte di un router e permettono al nodo destinatario di ricostruire il pacchetto originario;

- La necessità di frammentare un pacchetto si presenta quando la dimensione del pacchetto supera la Maximum Transmissible Unit (MTU) sul link di uscita e dipende dalla tecnologia usata al livello 2 (si può intuire, quindi, che le frammentazioni lungo il percorso possono essere anche multiple);
- Identification, 16 bit identificativi del datagramma e servono ad associare diversi frammenti ad un unico pacchetto originario;
- Flags, 3 bit di cui il bit D (don't fragment) indica se il pacchetto può essere frammentato e M (more fragments) se il pacchetto è l'ultimo frammento;
- Fragment offset, 13 bit che identificano la posizione del frammento all'interno del pacchetto;
- Il payload è creato di norma da un protocollo di trasporto ma, in circostanze particolari, può contenere anche un altro pacchetto IP (pratica detta incapsulamento IP in IP), mentre alcuni protocolli ausiliari (non intesi a supportare la comunicazione di applicazioni eseguite nei terminali) inviano i loro messaggi inserendoli direttamente in un payload IP (ICMP, IGMP, OSPF, ecc...).

Se un pacchetto di dimensione N arriva ad un router e deve essere trasmesso su un link di uscita con MTU M<N, il pacchetto viene frammentato ed ogni frammento è trasmesso come singolo pacchetto IP, con lo stesso ID number, e le cui dimensioni devono essere un multiplo di 8 byte (così come i payload di tutti i frammenti, tranne l'ultimo). Essendo la dimensione massima di un datagramma 65535 byte, ci possono essere al più 8192 (65535 / 8) frammenti per ogni datagramma. Come si è potuto osservare andando ad approfondire l'header di un pacchetto IPv4, la posizione del payload di un frammento rispetto al payload del pacchetto originario è espressa mediante un offset di 13 bit.



Si consideri un N pari a 4000 e un M pari a 1500, è necessaria la frammentazione del pacchetto in tre frammenti, ciascuno con un header di 20 byte:

1. Payload 1480, offset 0;
2. Payload 1480, offset $(1480) / 8 = 185$;
3. Payload 1020, offset $(1480 + 1480) / 8 = 370$

Si noti che $20 + 480 + 1480 + 1020 = 4000$. Oppure:

Original IP Datagram					
Sequence	Identifier	Total Length	DF May / Don't	MF Last / More	Fragment Offset
0	345	5140	0	0	0

IP Fragments (Ethernet)					
Sequence	Identifier	Total Length	DF May / Don't	MF Last / More	Fragment Offset
0-0	345	1500	0	1	0
0-1	345	1500	0	1	185
0-2	345	1500	0	1	370
0-3	345	700	0	0	555

Il compito di riassemblaggio è oneroso, il destinatario deve collezionare tutti i frammenti del pacchetto originario prima di consegnare il payload al livello superiore e, **se non termina entro un determinato tempo, tutti i frammenti arrivati devono essere scartati** (altrimenti potrebbe essere sfruttata come tecnica per attaccare un host bersaglio). Per evitare la frammentazione dei pacchetti lungo il percorso, talvolta si effettua un path MTU discovery, ovvero si determina la più piccola MTU lungo il percorso da un host A ad un host B, in modo che A eviti del tutto la frammentazione se invia pacchetti di dimensione minore a tale valore, auto-limitandosi; ad esempio, A invia un pacchetto ICMP echo request a B di massima dimensione con flag D = 1 e, se il pacchetto incontra sul percorso un router che non riesce a trasmetterlo, viene ricevuto un messaggio ICMP “Destination unreachable: Fragmentation needed”, A dimezza la dimensione e ritrasmette, ripetendo il procedimento finché non è assente il messaggio di errore.

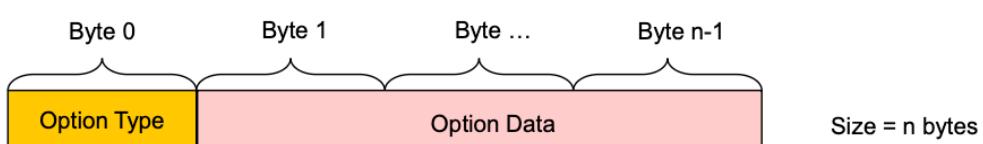
L'header IP può essere esteso con dei campi “Opzione” mediante i quali si intende chiedere una collaborazione speciale del pacchetto da parte dei router. I campi in questione sono:

- Security;
- Source routing;
- Route recording;
- Stream identification;
- Timestamping.

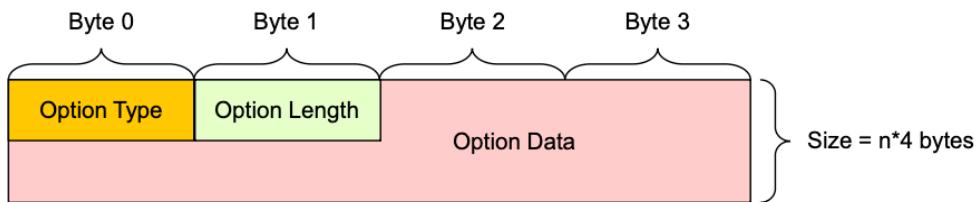
Per la presenza delle opzioni, l'header IP può essere di lunghezza variabile, perciò è inserito un campo header lenght, e se l'opzione non occupa 4 byte (o un suo multiplo), vengono inseriti dei bit di riempimento (tutti zero). Nei router in cui il dataplane è implementato in hardware, l'elaborazione di questi campi non è effettuata in hardware (fast path) ma in software (slow path), oppure questi campi sono ignorati. Gli attacchi DoS di tipo “Christmas Tree” consistono nel trasmettere pacchetti IP con diverse opzioni inutili nell'header al fine di sovraccaricare i router.

Il formato del header option non è sempre lo stesso ma può cambiare da caso a caso:

- **Single byte case**, le opzioni hanno una lunghezza di n byte implicitamente definita dal valore di Option Type (ad esempio, “End of Option List” Type = 0, “No Operation” Type = 1, ecc...);



- **Multiple byte case**, le opzioni hanno una lunghezza multipla di 4 byte esplicitamente indicata nel campo Option Length;



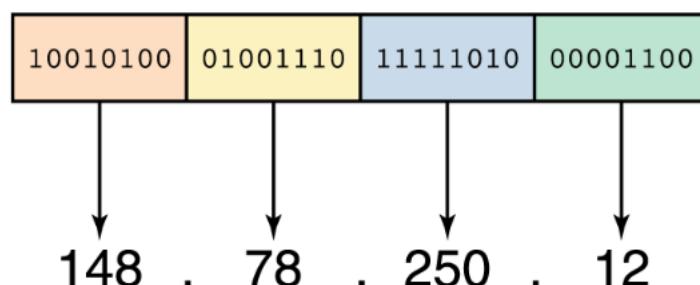
I numeri di IP Option Type standard sono registrati in una lista gestita da IANA:

Option Type byte		
Subfield Name	Size (bits)	Description
Copied	1	If 1: Option to be copied in all fragments If 0: Option only kept in first fragment
Option Class	2	0: Control Options 1: Unused 2: Debugging/Measurement 3: Unused
Option Number	5	Up to 32 different Options for each class

IP non garantisce di prevenire pacchetti duplicati, consegna ritardata o fuori ordine, corruzione dei dati o perdita di pacchetti; la consegna affidabile può avvenire grazie a meccanismi di controllo realizzati nei protocolli di livello superiore (negli end-system). Ogni router che riceve un pacchetto IP decide a quale altro nodo inoltrarlo, sulla base dell'indirizzo destinazione contenuto nel pacchetto, in maniera indipendente rispetto agli altri router e agli altri pacchetti passati in precedenza per lo stesso router. Il protocollo IP è stato progettato per realizzare un servizio best-effort, ovvero un servizio che non fornisce alcuna garanzia sulla consegna di un pacchetto ma non discrimina un pacchetto rispetto ad altri (proprietà di network equity).

Un indirizzo IP è una sequenza di 32 bit che, in forma testuale per un utente umano, è rappresentato nella notazione dotted decimal:

- I 32 bit sono decomposti in 4 byte, il valore di ciascuno dei quali è riportato in decimale come numero naturale tra 0 e 255;
- I quattro numeri decimali sono, poi, scritti in sequenza e separati dal punto.



Raggiunta la consapevolezza dell'esaurimento dello spazio di indirizzi supportati da IPv4, per aumentarlo fu avviato lo sviluppo del protocollo IPv6; nel frattempo, però, furono adottati sistemi per ritardare l'esaurimento, ormai inevitabile, come il subnetting, gli IPv4 privati e i NAT, ma, all'arrivo dello switching, l'impiego di IPv6 non avvenne in maniera rapida per via della

pervasività dell'adozione di indirizzi IPv4, sia nei software che negli hardware di router ed end system. L'estensione della capacità di indirizzo è effettuata facendo uso di 16 byte (128 bit), richiedendo un header di dimensioni maggiori; per contenere questo aumento, si è semplificato il formato, ottenendo un header grande “solo” il doppio di quello previsto in IPv4, aggiungendo però ulteriori funzioni, come il supporto al QoS.



Inizialmente è presente un campo versione, un campo Traffic Class (simile a quello Type of Service dell'IPv4) ed un campo Flow Label (che associa il pacchetto ad un dato flusso, nuova funzionalità di IPv6), per poi esserci un campo Payload Length (che rimpiazza il Total Length dell'IPv4), un campo Next Header ed uno Hop Limit. Non viene effettuato più il controllo della checksum.

Il supporto alla frammentazione dei pacchetti e le opzioni non sono più presenti nell'header dell'IPv6, ma sono previste in altra maniera mediante Extension Header, cioè header ulteriori posti sullo stesso livello dell'IPv6; il campo Next Header indica, quindi, quale sia l'Extension Header successivo oppure, se successivamente è presente il payload, si ha il protocollo del livello di trasporto presente (ad esempio, può essere Next Header = TCP). Un Next Header di valore 4 indica un incapsulamento IP in IP, che può essere usato per contenere un pacchetto IPv4 in uno IPv6.

Si sottolinea, però, che con IPv6 solo gli host sono in grado di frammentare i pacchetti, nel caso in cui un pacchetto supera l'MSS di un router, tale router scarta il pacchetto e richiede tramite ICMPv6 all'host di frammentare il pacchetto prima di inviarlo.

Per rappresentare i 128 bit si può usare una notazione esadecimale, in cui ciascun gruppo di 2 byte è rappresentato con quattro cifre esadecimali, per un totale di 8 gruppi. Valgono due regole di semplificazione della notazione: uno o più zero all'inizio di un gruppo di 4 cifre possono essere omessi, così come i campi nulli o una sequenza di essi, sostituiti quindi da due doppi punti. Così come per IPv4, anche in IPv6 lo spazio di indirizzamento si divide mediante netmask (ad esempio, un indirizzo IPv6 che inizia con i bit 1111111011 corrisponde all'utilizzare una netmask FE80::/10).

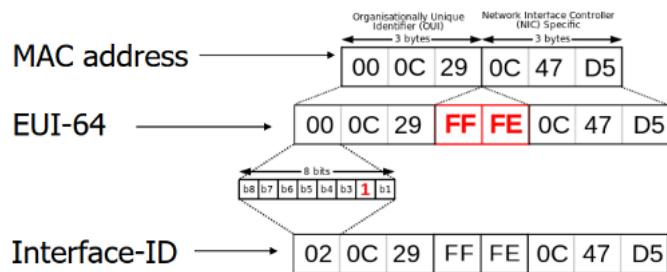
L'IPv6 supporta la trasmissione unicast, multicast, ed anycast (una trasmissione da uno al più vicino scelto secondo una certa selezione) ma non è possibile la trasmissione broadcast. Esistono tre tipi di indirizzi IPv6 unicast (che identificano una sola interfaccia nella rete):

- **Global unicast**, simili agli indirizzi IPv4 pubblici, che hanno un prefisso 2000::/3 (001 in binario) e sono costituiti da una parte subnet ID che identifica la sottorete e da una interface ID che identifica l'interfaccia nella sottorete;

- **Link local**, usati per l'invio di pacchetti nella sottorete locale, cioè per comunicare con altri host o router direttamente raggiungibili, ed hanno prefisso FE80/10 (qualunque interfaccia di rete per cui è abilitata IPv6 è sempre configurata con: un indirizzo IPv6 link-local con prefisso detto, 54 bit nulli (quindi in generale un FE80/64) e un insieme di 64 bit detto Interface-ID);
- **Unique local**, simili agli indirizzi IPv4 privati, ed hanno prefisso FD00::/8.

Si distinguono **due diversi ambiti**: **link**, che comprende tutti i dispositivi in comunicazione diretta tramite LAN o point-to-point, e **site**, che comprende un insieme di link gestiti da un'unica entità ma **deprecato per ambiguità**.

Un **interface-ID** è prodotto in vario modo a seconda del sistema operativo, o **a partire dal MAC address**, o **in maniera pseudo-randomica**. La possibilità di generarli **a partire da un MAC address** non garantisce necessariamente l'**univocità**, perché al giorno d'oggi un **end-system** è in grado di possedere un numero indefinito interfacce virtuali con dei dati **MAC address** e generabili all'occorrenza ma, in caso si potesse, si ottengono creando prima un **identificativo a 64 bit** detto **EUI-64**, inserendo la sequenza di 16 bit FF:FF tra i primi 24 bit e i secondi 24 bit del **MAC address**, e invertendo poi il settimo bit da sinistra dell'**EUI-64**. Alcuni indirizzi IPv6 speciali sono quelli di loopback ::1/128, analogo a 127.0.0.1 di IPv4, e l'assenza di indirizzo, ::/128.



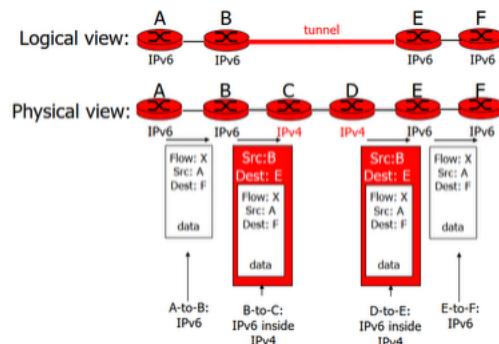
Esistono **varie modalità per attribuire un indirizzo IPv6 ad una interfaccia di host**, c'è la possibilità di assegnare **in maniera statica** da parte dell'amministratore, o **in maniera dinamica**. Per quanto riguarda quest'ultima, i metodi sono due: **a generazione automatica**, in cui un indirizzo è generato dal Sistema Operativo e i restanti sono determinati con la comunicazione di messaggi ICMPv6, o **generazione da parte di un server DHCPv6**; la differenza tra i due metodi sta nel fatto che **nel secondo caso il server DHCPv6 conosce gli indirizzi delle macchine**, mentre **nel primo gli indirizzi sono ignoti**. Inoltre, **nel primo caso è necessario che un host invii dei messaggi per il neighbor discovery affinché le altre macchine possano determinare i propri indirizzi IP**; i messaggi inviati dall'host sono del tipo Router Solicitation, comportando che i router direttamente raggiungibili rispondano con un messaggio Router Advertisement. È quest'ultimo tipo di messaggio che contiene i prefissi usati per determinare se un altro indirizzo condivide lo stesso link, oltre ad altre informazioni.

Per **determinare il MAC address di un sistema al quale vuole inviare un pacchetto** si fa sempre uso di **messaggi ICMPv6**, dove **l'host o router invia un messaggio Neighbor Solicitation**, e l'informazione voluta è data da un Neighbor Advertisement. **Non si fa quindi uso di ARP come nell'IPv4, non essendoci in IPv6 un indirizzo broadcast**.

Ci sono indirizzi IPv6 compatibili con IPv4, ossia indirizzi IPv6 con un prefisso di 96 bit nulli e i restanti 32 pari ad un indirizzo IPv4, quindi **rappresentabile come 0:0:0:0:0:X:Y, o 0:0:0:0:0:w.x.y.z o anche ::w.x.y.z**. Lo scopo di questi indirizzi è quello di **permettere la comunicazione di router IPv4 con router IPv6**.

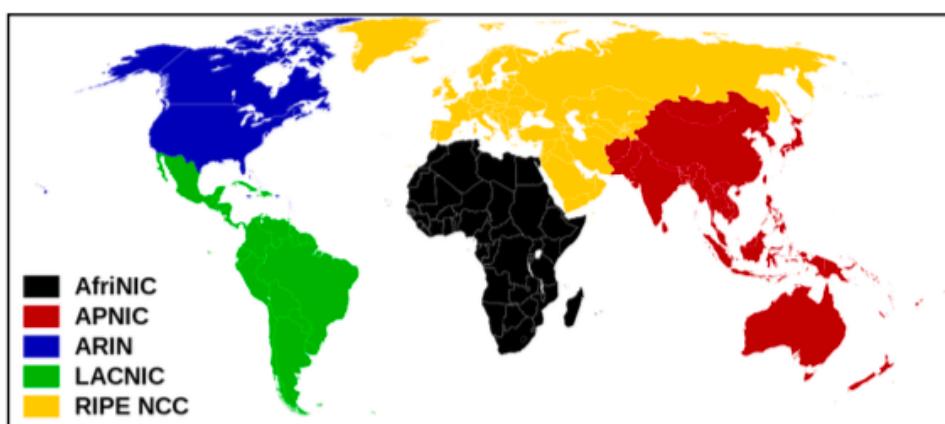
Un caso particolare è quello di **indirizzi IPv6 che incorporano un indirizzo IPv4** e di **indirizzi IPv6 mappati su IPv4**, dove i primi 80 bit sono nulli, 16 sono pari ad uno, e i 32 bit successivi sono un indirizzo IPv4; si rappresentano con **::FFFF:X:Y oppure ::FFFF:w.x.y.z** e sono usati per la comunicazione da router IPv6 verso router IPv4. Se la destinazione è nel formato precedente, infatti, il mittente genera un pacchetto IPv4.

Nel tunneling i pacchetti IPv6 vengono trasportati come payload all'interno di datagrammi IPv4, tra router IPv4. Questa tecnica si usa quando tra router IPv6 ci sono router IPv4, e fa in modo che **un router IPv4 possa trasmettere un pacchetto IPv6 nonostante non lo supporti**, ricevendolo imbustato all'interno di un pacchetto IPv4.



L'ASSEGNAZIONE DEGLI INDIRIZZI IP

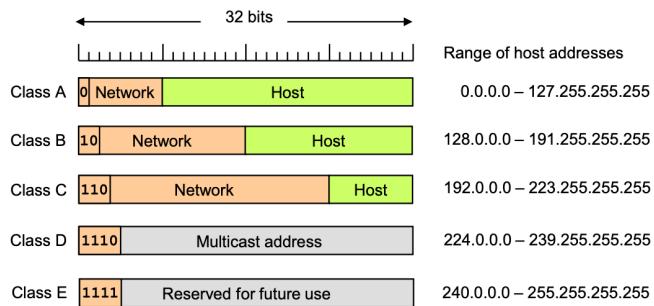
In una rete IP (ad esempio, Internet) **un indirizzo serve ad identificare univocamente un'interfaccia di rete di un dispositivo**. Un **end system** può avere **una sola interfaccia di rete**, un **router almeno due** (i terminali moderni hanno diverse interfacce di rete, multi-homed, e dunque diversi indirizzi IP, interfaccia Ethernet, WiFi, Bluetooth, ecc...). **L'assegnazione degli indirizzi IP avviene attraverso un sistema gerarchico di autorità**, dove il gestore globale dell'intero spazio di indirizzamento è IANA (Internet Assigned Numbers Authority, in origine una persona sola: Jon Postel), delegate poi a cinque autorità regionali RIR (Regional Internet Registries), in Europa il RIPE NCC. **I registry regionali assegnano blocchi di indirizzi agli Internet Service Provider (ISP)** ed alle grosse organizzazioni che, a loro volta, sono **responsabili dell'assegnazione unica degli indirizzi di loro pertinenza ai singoli dispositivi delle proprie reti**.



I router entrano in gioco quando è necessario collegare più reti WAN tra di loro ma all'interno di una rete stessa è comunque possibile l'intercomunicazione grazie al protocollo IP. Questo tipo

di comunicazione interna, per il **significato di rete, non necessita l'ausilio di un router** perché tutti gli host di una stessa rete possono comunicare direttamente al livello Data Link; pertanto, risulta necessario in un indirizzo IP specificare sia un campo che identifica la rete di appartenenza e sia un campo che identifica il terminale all'interno della rete.

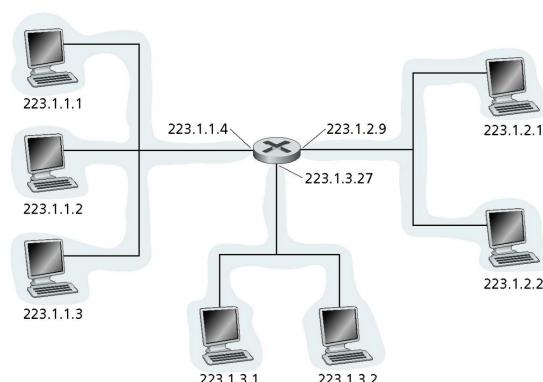
Inizialmente, nella rete Internet si adottò una gestione degli indirizzi per classi, per la quale la demarcazione tra i due campi è fissa e determinata dal valore dei primi bit.



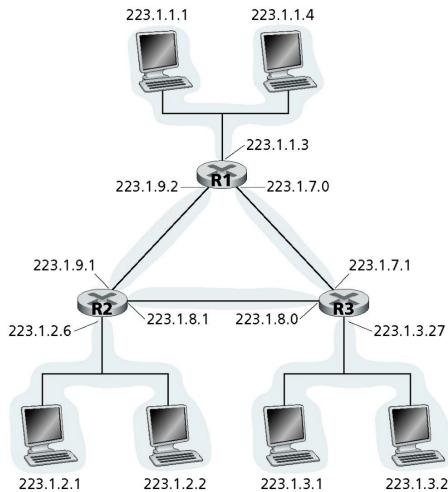
Un **indirizzo IP di classe A** usa il primo byte per identificare la rete ed i restanti per l'host, mentre una rete di classe A è un blocco di 2^{24} ($\approx 16.777.216$) indirizzi consecutivi e ne esistono solo 256 distinte (2^7 possibili configurazioni perché il MSB è sempre nullo). Un **indirizzo IP di classe B** usa i primi due byte per identificare la rete ed i restanti due per l'host, mentre una rete di classe B è un blocco di 2^{16} (≈ 65.536) indirizzi consecutivi ed esistono altrettante reti distinte. Un **indirizzo IP di classe C** usa i primi tre byte per identificare la rete ed i restanti per l'host, mentre una rete di classe C è un blocco di 2^8 (≈ 256) indirizzi consecutivi e ne esistono 2^{24} distinte. Un **indirizzo di classe D** (nel range 224.0.0.0-239.255.255.255) è usato per identificare gruppi di trasmissione multicast (RFC 1112) e possono essere usati solo come indirizzo destinazione, mentre gli **indirizzi di classe E** (240.0.0.0-255.255.255.255) sono stati riservati per usi futuri e mai utilizzati.

Per l'assegnazione di indirizzi IP alle interfacce di rete, si considerino tre reti fisiche associate a tre diverse reti di classe C:

- 223.1.1.X è il prefisso per la rete a sinistra;
- 223.1.2.X è il prefisso per la rete a destra;
- 223.1.3.X è il prefisso per la rete in basso.



Oppure, considerando sei reti fisiche distinte associate a sei diverse reti di classe C:



Tra tutti i possibili indirizzi IP, ci sono **alcuni che hanno funzionalità particolari**. L'indirizzo **0.0.0.0** è usato per scopi speciali in vari contesti (come l'identificazione, all'interno di un host, del "qualunque indirizzo IP assegnato alle sue proprie interfacce"), mentre il blocco di indirizzi **0.X.Y.Z** è riservato e non può essere assegnato specificamente ad un'interfaccia. Gli indirizzi di rete **127.X.Y.Z** sono tutti associati ad un'interfaccia virtuale che è presente in qualunque sistema e che può essere usata per la comunicazione tra processi in esecuzione nella stessa macchina (interfaccia di loopback). L'indirizzo **255.255.255.255** (usato come destinazione) indica il broadcast a tutti gli host nella rete locale del mittente. L'indirizzo che ha tutti zero nel campo host serve a identificare la rete (ad esempio, 148.78.0.0), mentre quello che ha tutti uno nel campo host serve a identificare, come destinatario, tutti gli host della rete specificata nel campo network (broadcast diretto); ad esempio, un pacchetto con indirizzo 148.78.255.255 è consegnato a tutti i sistemi che hanno un'interfaccia nella rete 148.78.0.0. Data una rete qualsiasi, gli indirizzi che hanno nel campo host tutti zero e tutti uno sono considerati speciali e, quindi, non assegnabili a specifici host.

Tipicamente, un qualunque end system è configurato in modo da avere almeno un'interfaccia di rete virtuale (cioè non associata ad alcuna scheda di rete fisica) detta interfaccia di loopback, il cui scopo è quello di consentire la comunicazione tra processi attivi nello stesso end system mediante protocolli TCP/IP anche quando il sistema è fisicamente disconnesso da una rete. Nei sistemi Linux, questa interfaccia è denominata **1o0** e ad essa è assegnato staticamente l'indirizzo **127.0.0.1**, al quale è associato nel file di sistema **/etc/hosts** il nome **localhost**. È anche possibile creare ulteriori interfacce di loopback (**1o1, 1o2,...**) a cui sono assegnati indirizzi del tipo **127.0.0.n**. I dispositivi di rete scartano qualunque pacchetto che abbia un indirizzo del tipo **127.X.Y.Z** come indirizzo mittente o destinatario.

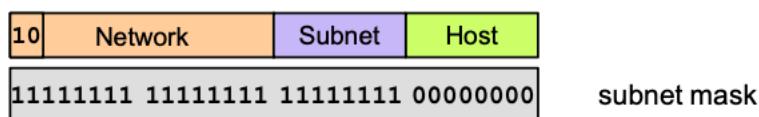
La gestione per classi degli indirizzi IP condusse ad un uso inefficiente dello spazio di indirizzamento e ad una conseguente difficoltà di assegnare indirizzi IP a nuove reti collegate ad Internet; infatti, una rete con più di 256 host necessita di un blocco di indirizzi di classe B che, però, comprende 65.536 indirizzi, che sono relativamente pochi. Nel 1992 una nuova tecnica di gestione degli indirizzi IP, **CIDR**, fu introdotta per la capacità di separare campo network e campo host all'interno di una rete tramite una tringa di 32 bit ausiliaria, detta **network mask** o **netmask**, ovvero una sequenza di k "1" in testa che identificano la parte di bit che rappresentano l'**identificatore di network**, ed una restante sequenza di $32-k$ "0" che identificano l'**host nella rete**. Le netmask si rappresentano o, anch'esse, con la notazione dotted decimal o con la notazione /k, con k il numero di "1" consecutivi in testa; ad esempio:

- 255.255.0.0 o /16;

- 255.255.128.0 o /17;
- 255.255.255.0 o /24;
- 255.255.255.240 o /28;
- 255.255.255.255 o /30.

Nella gestione CIDR (o gestione classless), ciascuna delle reti originariamente definite dalle classi è stata divisa in sottoreti, ovvero blocchi di indirizzi consecutivi identificati usando un campo subnet sottratto al campo host e la cui demarcazione tra campi subnet e host è realizzata mediante netmask. Anche per la gestione classless, vale il significato di rete, per cui gli host di una stessa sottorete comunicano direttamente a livello 2 senza l'ausilio di un router. Tutti gli host della stessa sottorete, poi, devono essere configurati con la stessa netmask.

Di seguito è proposto un **blocco di indirizzi di classe B ripartito in $2^8 = 256$ sottoreti da 256 indirizzi ciascuna:**



Questa subnet può contenere **fino a 254 host distinti**, perché **gli indirizzi che hanno tutti zero e tutti uno nel campo host sono usati per scopi speciali**.

IANA (con RFC 1918) ha riservato i seguenti tre blocchi di indirizzi per reti TCP/IP private (ovvero, reti non collegate a livello 3 alla rete Internet):

- 10.0.0.0-10.255.255.255 (10.0.0.0/8) un blocco di 2^{24} indirizzi;
- 172.16.0.0-172.31.255.255 (172.16.0.0/12) un blocco di 2^{20} indirizzi;
- 192.168.0.0-192.168.255.255 (192.168.0.0/16) un blocco di 2^{16} indirizzi.

Un'organizzazione può assegnare nella propria rete interna degli indirizzi specificati in RFC 1918 senza dover ricevere alcuna autorizzazione; questo, però, impedisce la possibilità di comunicare con host in Internet, a meno di non usare una soluzione di address translation (NAT) che verrà approfondita di seguito.

In RFC 5737 sono indicati tre blocchi di indirizzi che sono considerati riservati per l'uso in manuali e documentazione:

- 192.0.2.0/24 (TEST-NET-1);
- 198.51.100.0/24 (TEST-NET-2);
- 203.0.113.0/24 (TEST-NET-3).

I router di Internet sono configurati per eliminare (cioè non inoltrare) pacchetti aventi come indirizzo mittente o destinatario uno degli indirizzi riservati di RFC 1918 e RFC 5737.

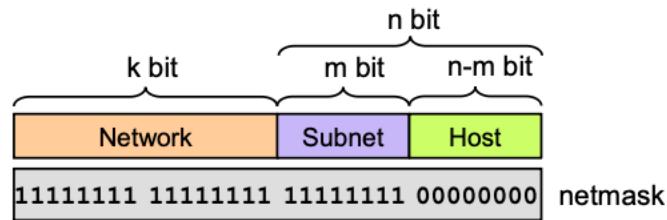
Esistono anche altri indirizzi IPv4 riservati per usi speciali ma sono elencati sul sito ufficiale di IANA.

Un blocco di $N = 2^n$ indirizzi consecutivi è identificato dal prefisso /k con $k = 32 - n$. Per **subnetting con fixed lenght subnet mask** (FLSM) si indica la **ripartizione di un blocco di N indirizzi consecutivi in M sottoinsiemi disgiunti**, ciascuno formato da N/M indirizzi consecutivi (**subnet**). In ciascun blocco di N/M indirizzi, **due indirizzi saranno riservati per scopi speciali**:

- L'indirizzo con tutti zero nel campo host indica l'intera subnet;
- L'indirizzo con tutti uno nel campo host indica il broadcast alla subnet.

Pertanto, solo $(N/M)-2$ indirizzi saranno attribuibili alle interfacce degli host che appartengono alla subnet e questo intervallo, pertanto, è detto **host range**. All'interno del blocco, ciascuna subnet sarà identificata da $m = \log_2(M)$ bit e tutte le interfacce dei dispositivi della rete saranno configurate con una netmask avente:

- $k+m$ bit “1” per identificare globalmente ciascuna subnet;
- $n-m$ bit “0” per identificare ciascun host all'interno di una subnet.



Si consideri un blocco di $N = 2^8 = 256$ indirizzi 192.168.20.0/24 e lo si voglia ripartire in $M = 8$ blocchi uguali (subnet) da $N/M = 32$ indirizzi ciascuno; in ciascuna subnet sono assegnabili agli host ed ai router al più 30 indirizzi, dal momento in cui si devono contare i due indirizzi riservati. Ciascuna subnet, poi, è identificata da $m = \log_2 8 = 3$ bit ed occorre usare, pertanto, una netmask con $24 + 3$ bit “1” e $8 - 3 = 5$ bit “0”, ovvero:

- 11111111.11111111.11111111.11100000 in binario;
- 255.255.255.224 in notazione dotted decimal;
- /27 come prefisso.

Subnet	Subnet Address	Host Range	Broadcast Address
0	192.168.20.0 /27	192.168.20.1 to 192.168.20.30	192.168.20.31
1	192.168.20.32 /27	192.168.20.33 to 192.168.20.62	192.168.20.63
2	192.168.20.64 /27	192.168.20.65 to 192.168.20.94	192.168.20.95
3	192.168.20.96 /27	192.168.20.97 to 192.168.20.126	192.168.20.127
4	192.168.20.128 /27	192.168.20.129 to 192.168.20.158	192.168.20.159
5	192.168.20.160 /27	192.168.20.161 to 192.168.20.190	192.168.20.191
6	192.168.20.192 /27	192.168.20.193 to 192.168.20.222	192.168.20.223
7	192.168.20.224 /27	192.168.20.225 to 192.168.20.254	192.168.20.255

Sia disponibile un blocco di $N = 2^n$ indirizzi consecutivi identificato dal prefisso /k, per **subnetting con variable lenght subnet mask** (VLSM) si indica la **ripartizione del blocco di N indirizzi in M sottoinsiemi disgiunti di differente dimensione**, ognuno dei quali con una **dimensione pari ad una potenza di due**. La ripartizione avviene in maniera gerarchica:

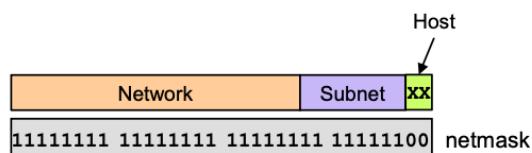
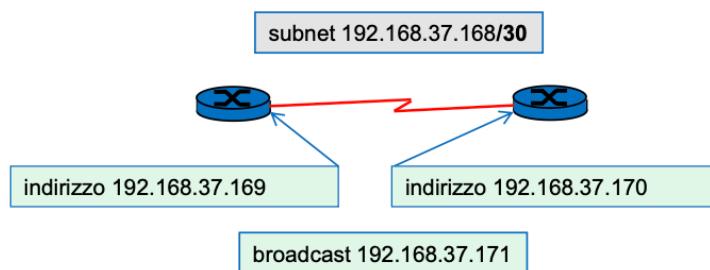
- Si ripartisce il blocco in M_1 blocchi “grandi” identificati da un prefisso di $m_1 = \log_2 M_1$ bit;
- Uno o più dei blocchi ottenuti sono suddivisi in M_2 blocchi più piccoli identificati da un prefisso di $m_2 = \log_2 M_2$ bit;
- La ripartizione può essere ulteriormente effettuata in blocchi ancora più piccoli, se necessario.

Ciascun blocco sarà associato alla propria netmask, con subnet associate ai blocchi “grandi” aventi netmask pari a $/k+m_1$ e subnet associate ai blocchi ottenuti dalla seconda suddivisione pari a $/k+m_1+m_2$ ecc...

Quando si usa la tecnica VLSM, alle reti associate ai link punto-punto che collegano due router conviene assegnare una subnet che comporti il minor spreco possibile di indirizzi IP; tale subnet deve comprendere quattro indirizzi IP consecutivi:

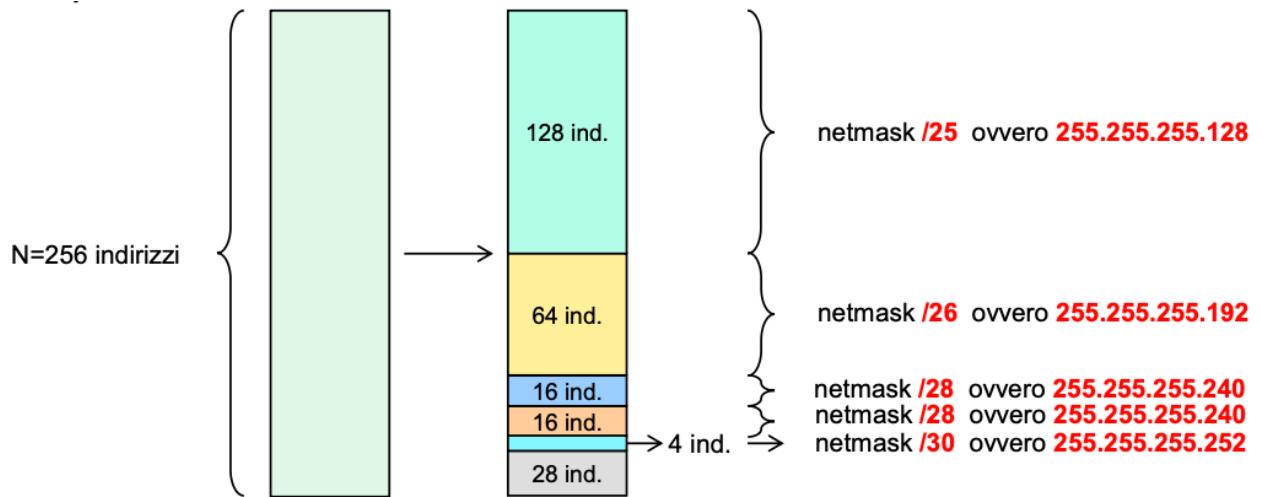
- **Un indirizzo che ha nel campo host la configurazione di bit 00 riservato per la subnet;**
- **Un indirizzo che ha nel campo host la configurazione di bit 11 riservato per il broadcast;**
- **Sono assegnabili alle interfacce dei due router gli indirizzi che hanno nel campo host le configurazioni di bit 01 e 10.**

La netmask da utilizzare per una tale subnet è, quindi, /30, ovvero 255.255.255.252.



Riprendendo lo stesso esempio di prima, cioè con un blocco di $N = 2^8 = 256$ indirizzi consecutivi identificato dal prefisso /24, si consideri una ripartizione in 5 sottoreti di dimensione: 128, 64, 16, 16, 4. In ciascun blocco, due indirizzi sono riservati e, pertanto, non assegnabili a host; quindi, il numero totale di indirizzi usati è $128+64+16+16+4=228$, con un avanzo di 28 indirizzi. La ripartizione è effettuata in blocchi di dimensioni crescenti:

Subnet	Subnet Address	Host Range	Broadcast Address
0	192.168.20.0 /25	192.168.20.1 to 192.168.20.126	192.168.20.127
1	192.168.20.128 /26	192.168.20.129 to 192.168.20.190	192.168.20.191
2	192.168.20.192 /28	192.168.20.193 to 192.168.20.206	192.168.20.207
3	192.168.20.208 /28	192.168.20.209 to 192.168.20.222	192.168.20.223
4	192.168.20.224 /30	192.168.20.225 to 192.168.20.226	192.168.20.227
Unused		192.168.20.228 to 192.168.20.255	



I ROUTER

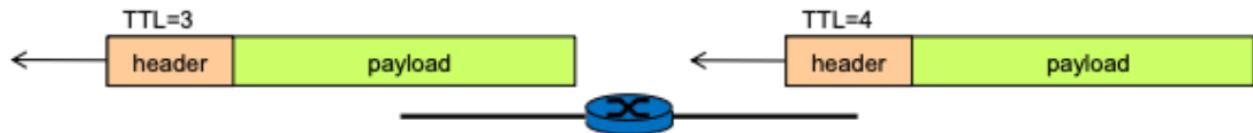
Un router è un dispositivo dotato di più interfacce di rete che serve a collegare due o più reti tra di loro ed ha il compito di inoltrare pacchetti nella rete verso la destinazione finale; pertanto, le due funzioni principali di un router sono il **forwarding** e il **routing**. Per **forwarding** si intende l'inoltro di ciascun pacchetto che entra da un'interfaccia verso un'altra interfaccia; l'azione di forwarding effettuata dai router deve essere coordinata, in modo da far sì che un pacchetto, generato da un qualunque host mittente, possa arrivare verso un qualsiasi host destinatario. Per **routing**, si intende la determinazione dei percorsi che un pacchetto ha da fare per giungere a destinazione.

Ogni indirizzo assegnato ad un'interfaccia di rete di un router è detto **indirizzo di gateway** e due router direttamente collegati tra di loro avranno, per le interfacce reciprocamente opposte, lo stesso **indirizzo di gateway** (lo stesso non vale se il collegamento è mediato da un altro dispositivo). I router operano secondo lo stesso modello **store-and-forward** finora menzionato; ciò significa che ogni pacchetto è ricevuto internamente, mantenuto in uno o più buffer di memoria gestiti come code, controllato per eventuali errori e infine, se quest'ultimo check dà successo, trasmesso su un link di uscita. Un router ha interfacce sia per connessioni **LAN** (ad esempio, Ethernet) che per connessioni **WAN**, oltre a porte seriali ed interfacce per connessioni di gestione.

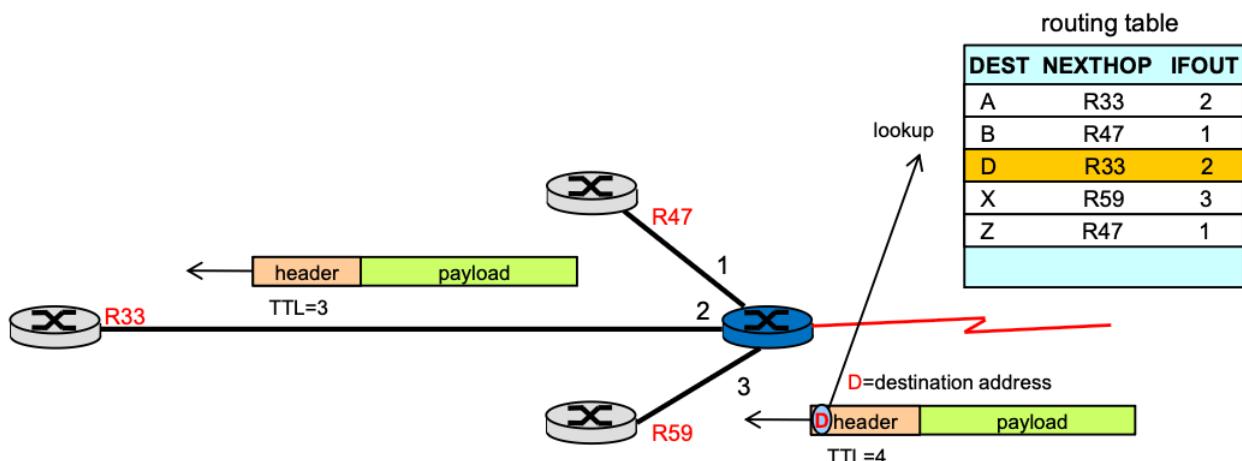
Le due funzioni principali in questione sono svolte contemporaneamente da **due distinte sezioni del router**: il **forwarding** dal **data plane** e il **routing** dal **control plane**. Il primo deve essere in grado di **operare alla velocità dei link** (infatti, la funzione di forwarding è tipicamente realizzata mediante hardware specifico), mentre il secondo può operare a **velocità più basse** (le scelte di percorso cambiano nell'ordine di secondi) e la funzione di routing è tipicamente realizzata mediante **software eseguito da CPU**.

Un router IP è un dispositivo dotato di più interfacce di rete che serve a collegare due o più reti tra di loro e ad ognuna di esse è assegnato un **indirizzo IP** appartenente alla subnet associata alla rete a cui l'interfaccia si collega; internamente, il router identifica le proprie interfacce mediante **degli identificatori locali** (come fa0, eth0, eth1, ecc...). La **funzione di forwarding** svolta da un router IP è la seguente:

- Per ciascun pacchetto, viene determinata l'interfaccia di uscita sulla base dell'indirizzo IP destinazione contenuto nel pacchetto;
- Prima della ritrasmissione, il campo TTL (time-to-live) nell'header del pacchetto viene decrementato di 1 (e viene verificato che non sia nullo);
- La modifica del TTL impone il ricalcolo del valore del campo header checksum;
- Il pacchetto è trasmesso sull'interfaccia di uscita.



La scelta dell'interfaccia verso la quale il router realizza la ritrasmissione è determinata dall'indirizzo IP del destinatario del pacchetto ed è operata sulla base delle regole di instradamento contenute in una tabella di routing: ogni volta che il router deve inoltrare un pacchetto, viene consultata la tabella di routing per determinare l'interfaccia di uscita del pacchetto, tramite un'operazione di ricerca nella tabella (lookup) per determinare la regola da applicare.



Nella tabella di routing c'è scritto, per ogni destinazione:

- L'indirizzo IP del nexthop router;
- L'identificativo locale dell'interfaccia tramite la quale si raggiunge il nexthop.

Non è plausibile avere una regola per ciascun possibile indirizzo IP di destinazione, sarebbero $2^{32} \approx 4$ miliardi di indirizzi (sufficiente per topologie piatte, per questo si parla di flat network addressing); occorrono, quindi, tecniche che consentano di compattare le regole nelle tabelle di routing in modo che tutti i blocchi di indirizzi consecutivi, che hanno lo stesso prefisso e lo stesso nexthop router, siano rappresentati nella tabella di routing da una sola regola (si parla di network addressing gerarchico), mentre l'operazione di lookup nella tabella viene effettuata con il criterio longest prefix match. Una regola di default è di solito presente e si applica a tutte le destinazioni per le quali non c'è una regola scritta nella tabella.

Il campo "destinazione" è sufficiente che sia l'identificatore di una rete, perché, ogni pacchetto indirizzato ad un host di una data rete passerà sempre per lo stesso nexthop, dopodiché l'indirizzamento verso l'host specifico della rete è risolto dal router del nexthop. Ciò permette di compattare molto una tabella di routing. Ogni rete è identificata da due indirizzi IP, uno di rete,

e uno della netmask, quindi una destinazione nella tabella di routing corrisponde ad una tale coppia di valori.

Con questo approccio, la ricerca dell'IP di destinazione di un datagramma all'interno della tabella di routing si riduce a verificare quale rete di destinazione nella tabella lo contiene. Si vedrà che, per via della tecnica NAT, gli indirizzi IP non sono necessariamente identificatori univoci per reti differenti; quindi, in altre parole, un indirizzo IP di destinazione può corrispondere a più regole nella tabella di routing. È per gestire questi casi che il lookup nella tabella di routing è effettuato con il criterio detto longest-prefix match, ossia si sceglie la regola la cui destinazione combacia per il maggior numero di bit con l'indirizzo di destinazione del pacchetto. In genere, come ultima voce della tabella si ha una riga di default, da usare nel caso in cui la destinazione indicata non rientra in nessuna di quelle presenti nella tabella.

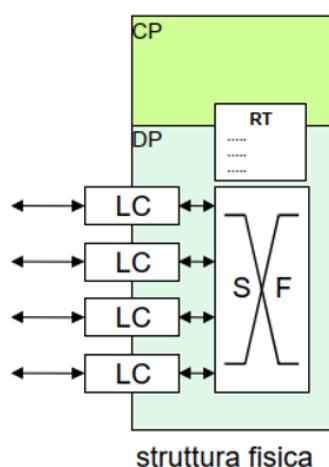
Ad esempio:

Destination Prefix (binary)	Destination Prefix (decimal)	Output Interface
11001000 00010111 00010	200.23.16.0/21	0
11001000 00010111 00011	200.23.24.0/21	2
11001000 00010111 00011000	200.23.24.0/24	1
default	default	3

Il pacchetto con destinazione 200.23.24.17 (ovvero **11001000 00010111 0001100 00010001**) ha longest prefix match con la terza regola quindi l'output interface è 1, mentre 200.23.25.11 (ovvero **11001000 00010111 00011001 00001011**) ha longest prefix match con la seconda regola quindi l'output interface è 2.

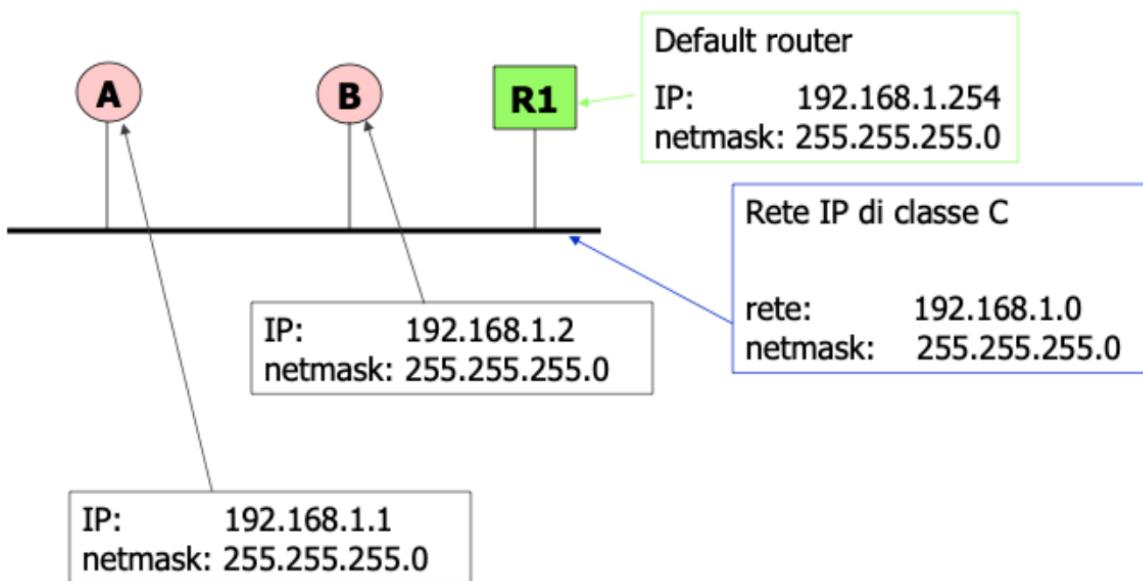
Un router esplica la funzione di forwarding dei pacchetti consultando, per ogni pacchetto processato, la tabella di routing, la cui costruzione è un compito che può essere svolto in due modi:

- **Routing statico**, per il quale l'amministratore di rete, conoscendo la topologia della rete, determina i percorsi tra qualunque coppia sorgente-destinazione e conseguentemente configura ciascun router con le opportune regole di inoltro;
- **Routing dinamico**, per il quale in ciascun router, nel control plane, opera un programma che, mediante lo scambio di informazioni con i router vicini, determina attraverso un algoritmo i percorsi verso qualunque destinazione e conseguentemente crea nella tabella di routing le regole corrispondenti;
 - Lo scambio di informazioni tra i router necessario all'esecuzione di routing è regolato da appositi protocolli di comunicazione, i protocolli di routing (o Address Resolution Protocol, ARP).



Un router, come detto, è costituito da un control plane e un data plane; le due parti interagiscono mediante la routing table, che è scritta dal primo e letta dal secondo. Nello specifico, il data plane è strutturato da circuiti hardware dedicati, la cui interfaccia con l'esterno è effettuata mediante le cosiddette line card, che operano in maniera bidirezionale.

Si considerino due terminali A e B, con A che ha intenzione di inviare un pacchetto a B; ci si chiede come faccia A a sapere se B si trova nella sua stessa sottorete IP. La risposta risiede nelle netmask; infatti, ogni computer ha un indirizzo IP e una netmask, quest'ultima usata proprio per individuare la propria sottorete IP (in Windows basta digitare il comando ipconfig/all). Per trovare l'indirizzo della sottorete a cui il computer B appartiene, per poi verificare se ci si trova nella stessa, viene fatta una AND bit a bit tra l'indirizzo IP destinazione e la propria netmask:



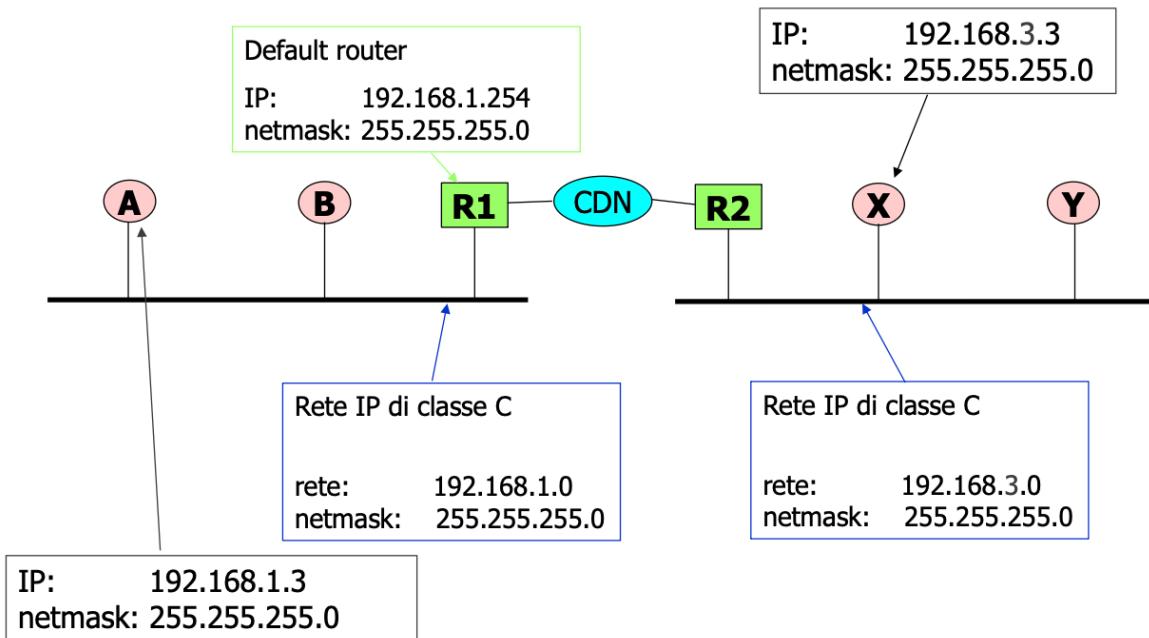
$$192.168.1.2 \text{ AND } 255.255.255.0 = 192.168.1.0$$

L'indirizzo della sottorete IP di B è lo stesso di A e, pertanto, la comunicazione può avvenire in maniera diretta: A manda un pacchetto ARP request in broadcast per conoscere il MAC address di B, specificando nel campo DEST IP il suo indirizzo.

Se A intende mandare un pacchetto ad un terminale X ma l'operazione di AND restituisce un risultato differente dal proprio indirizzo di sottorete, vuol dire che il destinatario non è nella stessa subnet IP del mittente (A):

$$192.168.3.3 \text{ AND } 255.255.255.0 = 192.168.3.0 \neq 192.168.1.0$$

In questo caso, pertanto, al primo hop il destinatario de livello 2 è l'interfaccia del router che appartiene alla subnet di A, questo prepara un pacchetto ARP in cui specifica come indirizzo IP DEST proprio l'indirizzo IP del router.

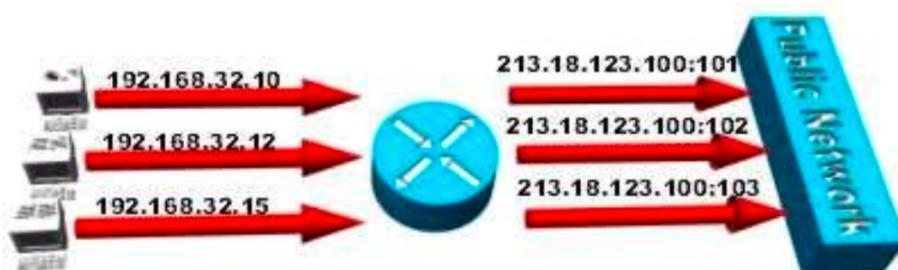


Quindi, secondo il protocollo ARP, quando un terminale vuole inviare un pacchetto viene fatta una AND logica tra l'indirizzo IP del destinatario e la propria netmask:

- Il risultato fornisce l'indirizzo della propria subnet:
 - Si invia una richiesta ARP per risolvere l'indirizzo della destinazione;
- Il risultato fornisce un indirizzo diverso da quello della propria subnet:
 - Il pacchetto deve essere inviato al router di default e, nel caso in cui il suo MAC address non sia noto:
 - Viene inviata una richiesta ARP per risolvere l'indirizzo IP del router.

NETWORK ADDRESS TRANSLATION

Per NAT (Network Address Translation) si intende una tecnica, definita in RFC 1631, che consente ad un dispositivo (router) di agire come intermediario tra Internet (rete pubblica) e una rete privata, in modo che un indirizzo IP possa rappresentare un gruppo di computer di una rete privata. L'uso più comune di NAT è quello con il quale è mappato un insieme di indirizzi privati su un unico indirizzo pubblico, utilizzando differenti port (di livello trasporto) per mantenere traccia dell'indirizzo privato di provenienza.



Quando il router riceve un pacchetto inviato da un computer della rete privata ad un computer esterno, salva in una tabella l'indirizzo e il port del mittente, oltre ai nuovi valori che esso assegna.

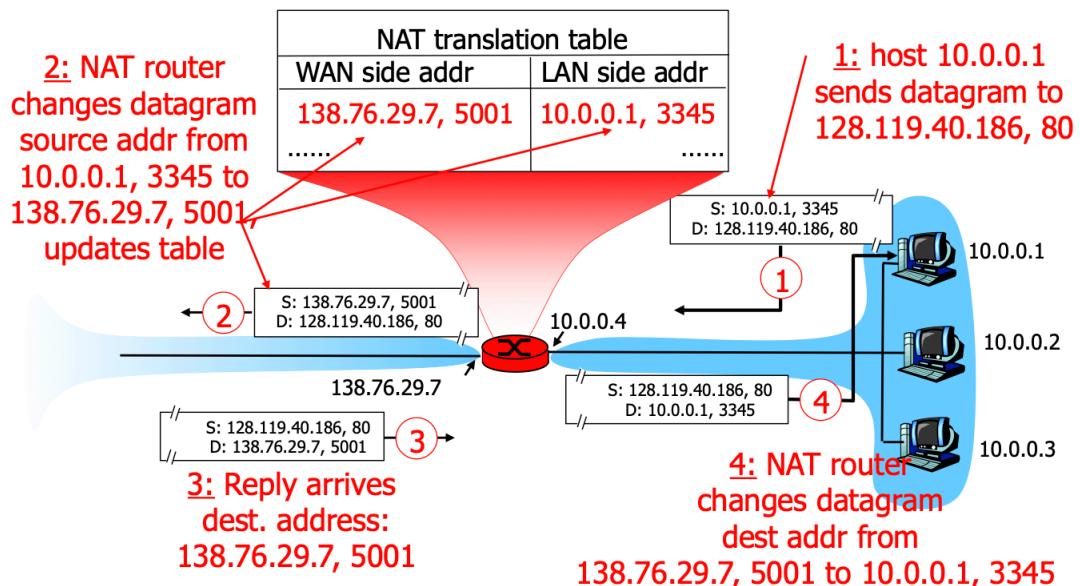
Tale tabella viene consultata anche quando il router riceve un pacchetto in entrata susseguente ad un pacchetto in uscita. Ad esempio, una tabella NAT per TCP appare come segue:

Source Computer	Source Computer's IP Address	Source Computer's Port	NAT Router's IP Address	NAT Router's Assigned Port Number
A	192.168.32.10	400	215.37.32.203	1
B	192.168.32.13	50	215.37.32.203	2
C	192.168.32.15	3750	215.37.32.203	3
D	192.168.32.18	206	215.37.32.203	4

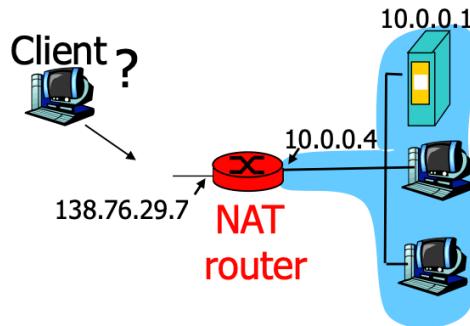
Un router NAT deve:

- **Sostituire nei pacchetti in uscita della rete privata** l'indirizzo IP privato del mittente A e numero di port sorgente P con l'indirizzo pubblico del NAT N e il nuovo numero di port sorgente X (l'host destinazione nella rete pubblica risponderà inviando pacchetti a questi ultimi);
- **Ricordare in una tabella le corrispondenze** indirizzo IP privato del mittente A e numero di port sorgente P – indirizzo pubblico del NAT N e nuovo numero di port sorgente X;
- **Sostituire nei pacchetti in entrata dalla rete pubblica** il nuovo numero di port sorgente X con l'indirizzo IP privato del mittente A e numero di port sorgente P.

Ad esempio:

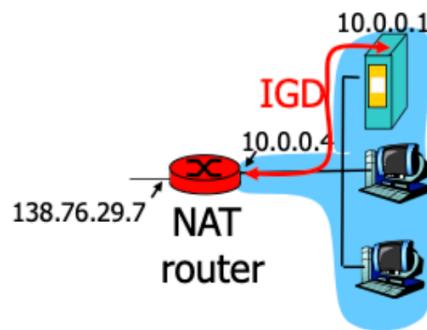


Il campo port-number è di 16 bit, permettendo 60.000 connessioni simultanee con un solo indirizzo LAN-side. NAT è controverso: i router dovrebbero processare fino al terzo livello, invece entrano in merito del livello trasporto, violano il principio di trasporto end-to-end, per cui un pacchetto dovrebbe arrivare inalterato a destinazione, e la carenza di indirizzi dovrebbe essere comunque risolta in IPv6, rendendo NAT potenzialmente non necessario.

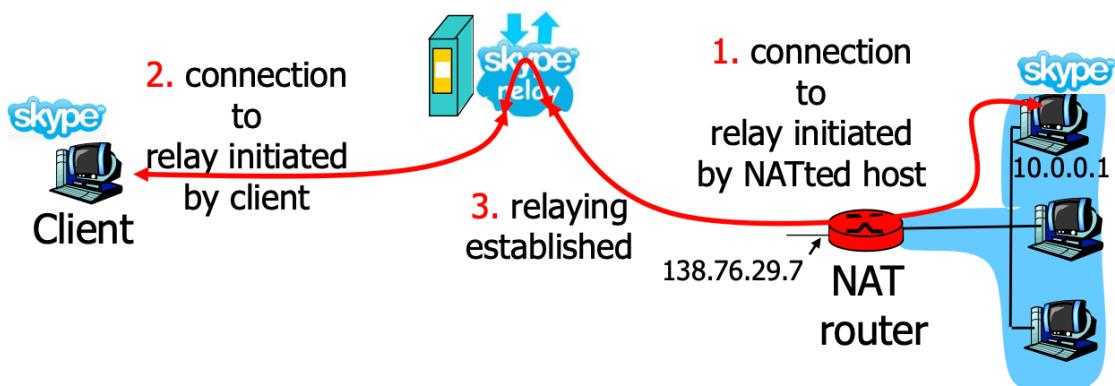


Si consideri un client che vuole connettersi ad un server con indirizzo 10.0.0.1, un indirizzo locale al LAN (il client non può usarlo direttamente come destination address) ma è solo visibile come indirizzo NAT 138.76.29.7. Per poter connettere i due terminali, possono essere adottate tre soluzioni:

1. Si configura, statisticamente, il NAT per inviare ogni richiesta di connessione in entrata ad un port con il server (ad esempio, 123.76.29.7 – port 2500 viene sempre indirizzato verso 10.0.0.1 – port 2500), pratica detta **port forwarding statico** ma non adatto in caso di web server distribuiti su più macchine locali (mediante una stessa porta pubblica, ogni client sarà associato sempre ad una sola macchina) ;
2. Si utilizza un UPnP (Universal Plug and Play) con un Internet Gateway Device (IGD) Protocol, per cui vengono aggiunti e rimossi i port mapping, in modo da permettere all'utente di connettersi al server corretto per una specifica durata (lease time), pratica detta **port forwarding dinamico**;

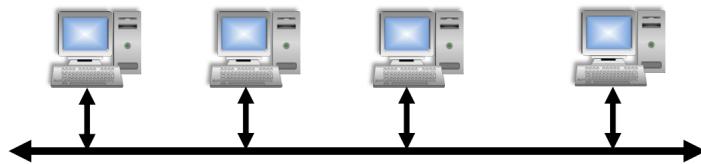


3. Si utilizza il **relying** (impiegato da Skype, ad esempio), per cui il NATed host stabilisce una connessione al relay, il client esterno si connette al relay e questo trasporta i pacchetti da un lato all'altro della connessione.



GLI INDIRIZZI MAC E I PROTOCOLLI ARP, RARP E DHCP

Quasi tutte le tecnologie di rete locale realizzano, in modi diversi, l'equivalente logico di un bus: la trasmissione di una stazione può arrivare direttamente ad una delle rimanenti N-1 stazioni e ciascuna stazione è collegata alla rete mediante una scheda (NIC, Network Interface Card, detta anche scheda di rete) identificata univocamente da un **indirizzo MAC** di 48 bit configurato dal costruttore nell'hardware della scheda e rappresentati comunemente in notazione esadecimale.

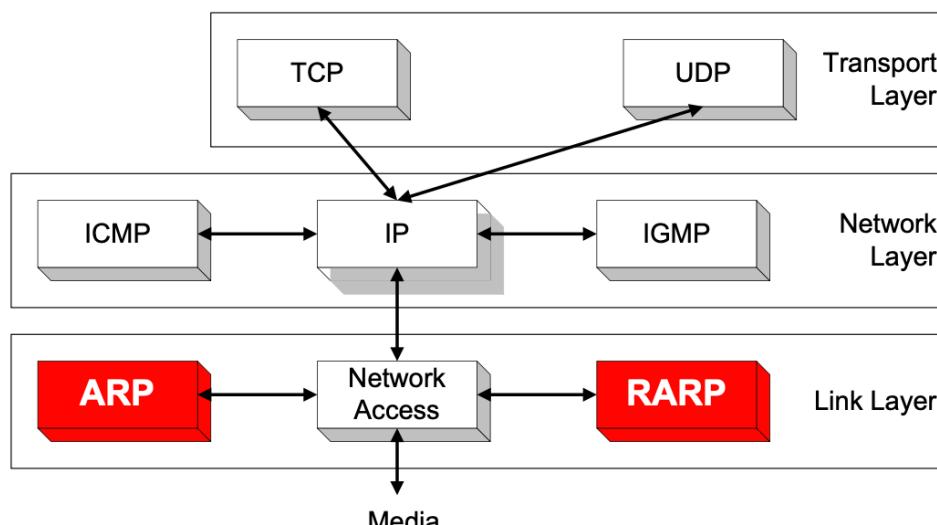


Una PDU di livello 2 (frame) Ethernet presenta nell'header:

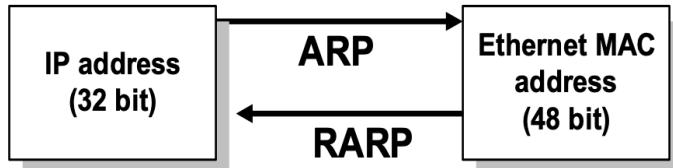
- **SA**, indirizzo MAC di 48 bit che identifica la **scheda** che ha trasmesso la frame;
- **DA**, indirizzo MAC di 48 bit che identifica la **destinazione della frame** e può essere:
 - **Unicast**, una specifica scheda collegata alla rete locale;
 - **Broadcast**, tutte le schede collegate alla rete locale (FF:FF:FF:FF:FF:FF);
 - **Multicast**, un sottoinsieme di schede collegate alla rete locale;
- **Type** (2 byte), indica il **protocollo del pacchetto trasportato dalla frame nella parte Data** (lunghezza variabile):
 - 0x0800, indica il protocollo IPv4;
 - 0x86DD, indica il protocollo IPv6;
 - 0x0806, indica il protocollo ARP;
 - Ecc...



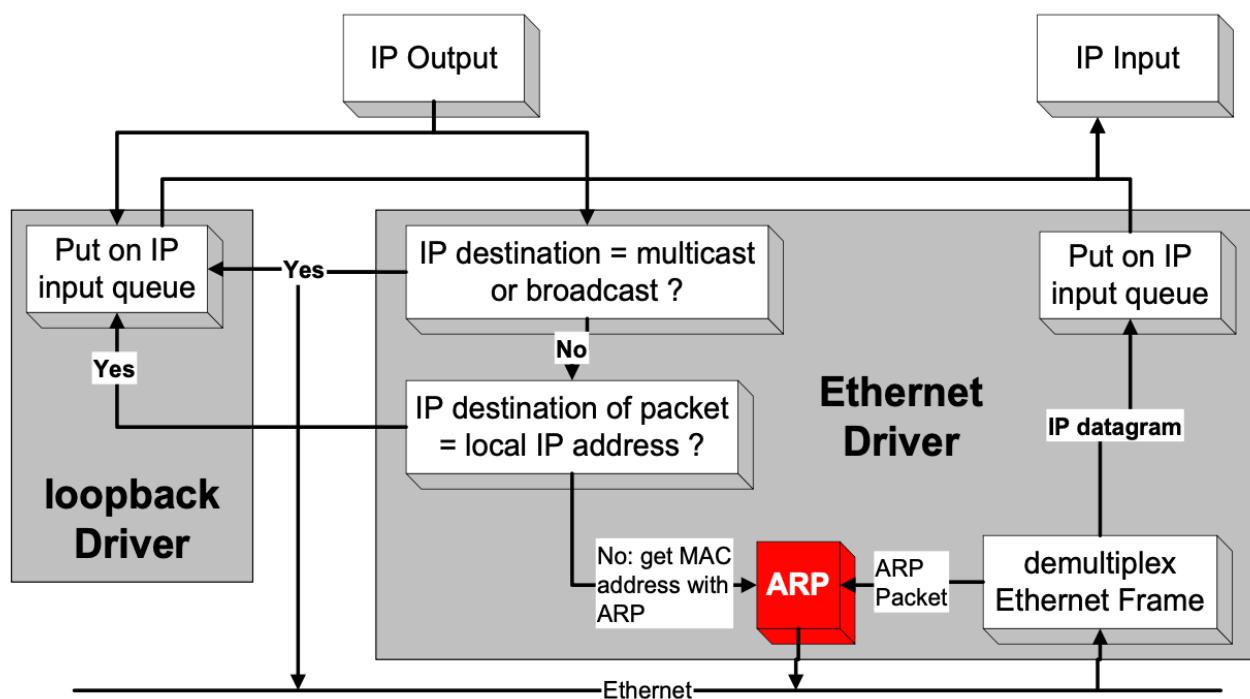
I protocolli ARP e RARP svolgono funzioni ausiliarie a supporto della trasmissione di datagrammi IP (ed in genere di un protocollo di livello rete) in reti locali con capacità di trasmissione broadcast (come, ad esempio, le LAN Ethernet).



Il protocollo ARP (definito in RFC 826) serve ad **associare un indirizzo di rete** (ad esempio, IP) al corrispondente **indirizzo MAC** e la sua funzione è necessaria quando **un host vuole trasmettere un pacchetto IP ad una certa destinazione presente sulla rete locale e non ne conosce il corrispondente indirizzo MAC**; conoscere un indirizzo MAC è necessario a costituire la frame secondo la struttura appena illustrata.

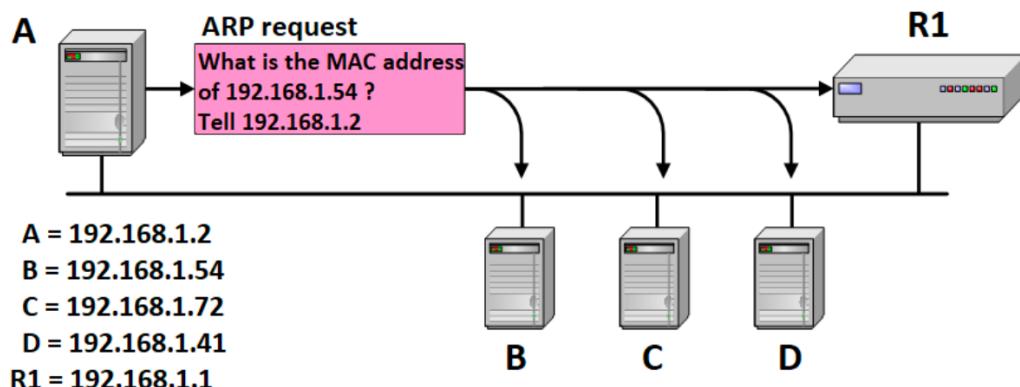


Lo schema generale di elaborazione di un pacchetto svolta dal driver è il seguente:

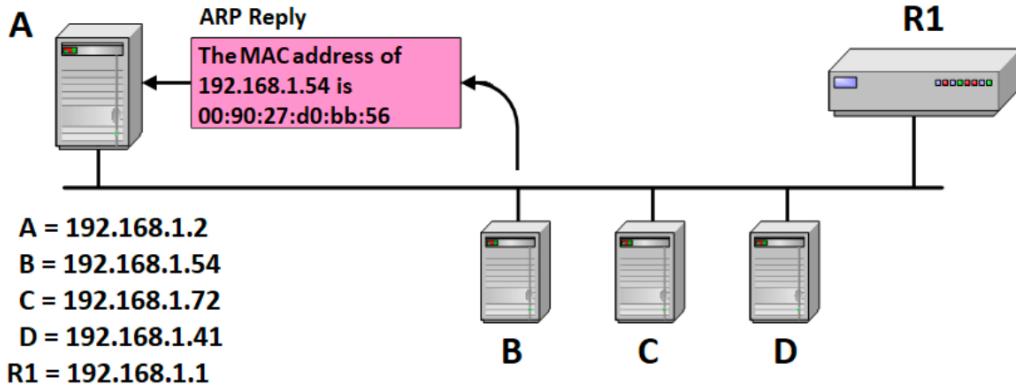


Per analizzare al meglio l'uso di ARP, si considerino i seguenti scenari:

1. A vuole trasmettere un pacchetto IP a B

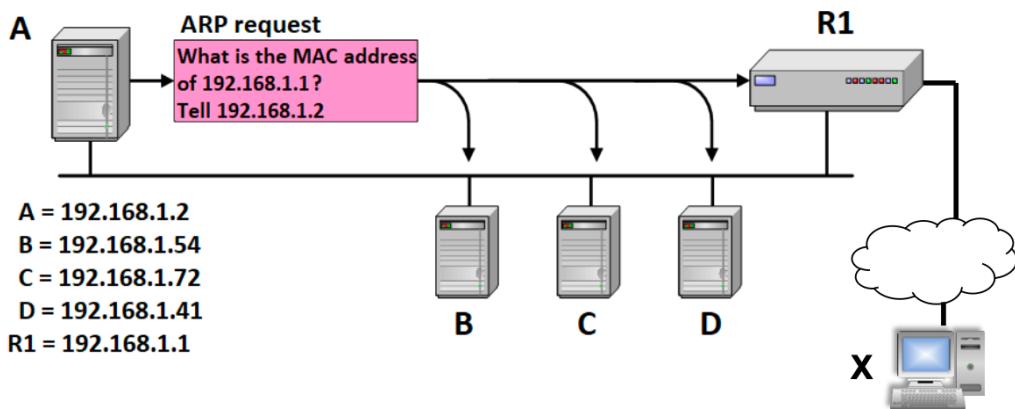


La destinazione è nella stessa LAN (subnet IP) del mittente; quindi, A chiede mediante ARP di conoscere il MAC address associato alla destinazione B (tramite target IP), direttamente raggiungibile. La richiesta ARP è, poi, trasmessa in una frame con indirizzo MAC destinazione broadcast FF:FF:FF:FF:FF:FF.

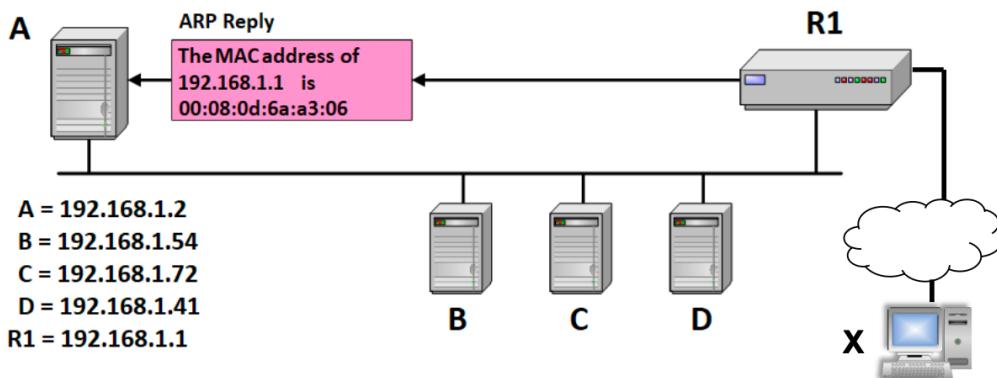


La risposta, poi, è inviata da B direttamente ad A.

2. A vuole trasmettere un pacchetto IP a X



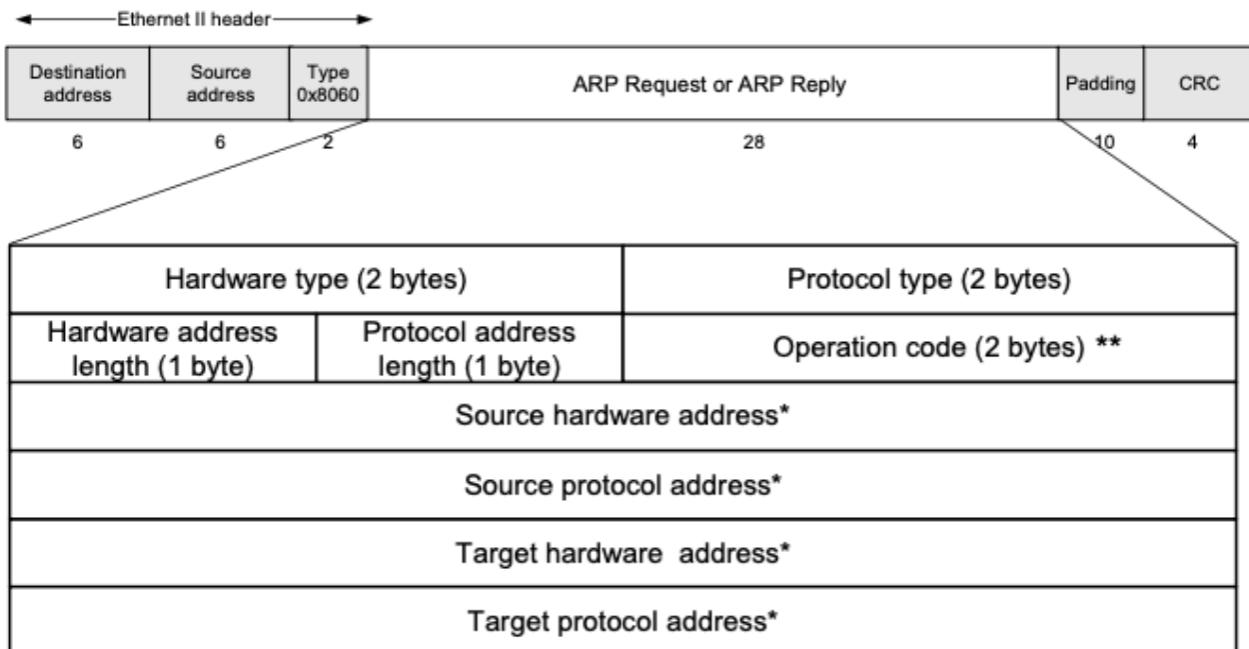
La destinazione è fuori dalla LAN (subnet IP) del mittente; in base alla tabella di routing, A determina l'indirizzo R1 del gateway associato alla destinazione X e chiede, mediante ARP, di conoscere il suo MAC address. La richiesta ARP è, poi, trasmessa in una frame con indirizzo MAC destinazione broadcast FF:FF:FF:FF:FF:FF.



Anche per la risposta, la destinazione è fuori dalla LAN (subnet IP) del mittente e la risposta ARP è inviata dal gateway R1 direttamente ad A.

Un caso non pensato durante la progettazione di ARP è quello per cui **un dispositivo invii una risposta ad una richiesta ARP nonostante non sia lui il terminale con l'indirizzo specificato**, iniziando così una **comunicazione con il terminale che ha inviato la richiesta**, casistica detta **ARP spoofing**.

Il formato di un pacchetto ARP è il seguente:



* Note: The length of the address fields is determined by the corresponding address length fields

** *OpCode = 0x0001 → request*

OpCode = 0x0002 → reply

La denominazione dei campi dei messaggi ARP usa i termini:

- **Protocol**, per riferirsi a indirizzi di rete (layer 3, tipicamente IPv4 a 32 bit/4 byte);
- **Hardware**, per riferirsi a indirizzi fisici (layer 2, tipicamente MAC a 48 bit/6 byte);
- **Source**, per riferirsi al mittente del messaggio ARP;
- **Target**, per riferirsi al destinatario del messaggio ARP.

Complessivamente, **un messaggio ARP contiene 4 indirizzi: source hardware, source protocol, target hardware e target protocol**. In un contesto nel quale ARP si usa per la risoluzione di indirizzi MAC rispetto ad indirizzi IP:

- **Source hardware address**, specifica il MAC address dell'host mittente;
- **Source protocol address**, specifica l'indirizzo IP del mittente;
- **Target hardware address**, contiene:
 - FF:FF:FF:FF:FF:FF, in una richiesta ARP “normale”;
 - 00:00:00:00:00:00, nei messaggi ARP Probe ed ARP Announcement;
 - L'indirizzo MAC del destinatario in una risposta;

- **Target protocol address**, contiene:
 - L'indirizzo IP di cui si desidera conoscere il MAC in una richiesta;
 - L'indirizzo IP del destinatario in una risposta.

Per ridurre traffico sulla rete e ridurre il tempo necessario all'invio di nuovi pacchetti IP, **ogni host mantiene in una cache le corrispondenze tra indirizzi logici e fisici precedentemente apprese** (pratica detta ARP caching); ciascuna corrispondenza viene **mantenuta per un tempo limitato** (alcuni minuti) e poi eliminata allo scadere di un timeout, mentre **per un eventuale rinnovo** di una corrispondenza prossima alla scadenza, **un host può inviare una richiesta ARP unicast al MAC address associato all'indirizzo IP** (comportamento definito Unicast Poll nella sezione 2.3.2.1 ARP Cache Validation di RFC 1122), **evitando così la trasmissione di una richiesta in broadcast**.

In quasi tutti i Sistemi Operativi, il comando **arp** consente di consultare le corrispondenze MAC-IP già presenti nella cache ARP dell'host, mentre **arp -f** si occupa di eliminarle (operazione di flush).

Interfaccia: 192.168.1.123 --- 0x15

Indirizzo Internet	Indirizzo fisico	Tipo
192.168.1.1	20-b0-01-1d-70-b0	dinamico
192.168.1.195	22-b0-01-1d-70-b9	dinamico
192.168.1.218	82-69-d7-6f-43-dd	dinamico
192.168.1.240	3c-52-82-08-ef-c2	dinamico
255.255.255.255	ff-ff-ff-ff-ff-ff	statico

Si osservi che l'**indirizzo IP speciale 255.255.255.255 usato per inviare un pacchetto IP in broadcast nella propria rete è associato al MAC address broadcast FF:FF:FF:FF:FF:FF**; questa associazione è considerata statica perché non ottenuta tramite ARP.

Per “**Gratuitous ARP**” (o GARP) si intende un **messaggio ARP reply** trasmesso in una frame con **MAC destinazione FF:FF:FF:FF:FF:FF** con:

- Source hardware address dell'host mittente;
- Source protocol address dell'host mittente;
- Target hardware address FF:FF:FF:FF:FF:FF;
- Target protocol address dell'host mittente.

Un GARP è un messaggio di risposta ARP unsolicited, cioè non generato per effetto di una precedente richiesta ma se:

- L'host desidera popolare le ARP cache degli altri host della rete locale (questo effetto potrebbe non essere raggiunto, dal momento in cui un host non è obbligato ad inserire le corrispondenze apprese mediante GARP);
- L'host ha modificato il proprio indirizzo IP o ha subito una disconnessione dalla rete e desidera aggiornare del cambiamento gli altri host (e switch) della rete locale.

Per “**ARP Probe**” si intende un **messaggio ARP request** trasmesso in una frame con **MAC destinazione FF:FF:FF:FF:FF:FF** con:

- Source hardware address dell'host mittente;
- Source protocol address 0.0.0.0;
- Target hardware address 00:00:00:00:00:00;

- Target protocol address indirizzo oggetto del “probe”.

Un ARP Probe può essere usato per verificare che nella rete locale non ci siano altri host con lo stesso indirizzo IP target e l’eventuale ricezione di una risposta al probe è sintomo del fatto che un altro host nella rete locale è già configurato con tale indirizzo. Usando ARP Probe, anziché un GARP, si evita di inserire una entry duplicata nelle cache degli altri host della rete locale, cioè si evita il verificarsi del problema della duplicazione di un indirizzo IP.

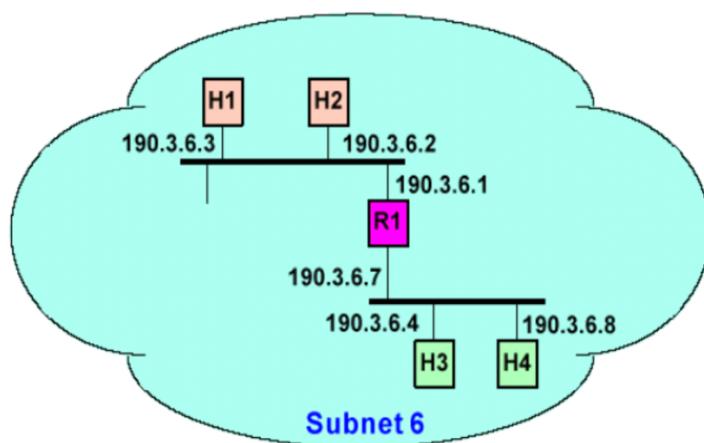
Un “**ARP Announcement**” si intende un **messaggio ARP request** trasmesso in una frame con **MAC destinazione FF:FF:FF:FF:FF:FF** con:

- Source hardware address dell’host mittente;
- Source protocol address dell’host mittente;
- Target hardware address 00:00:00:00:00:00 o FF:FF:FF:FF:FF:FF;
- Target protocol address dell’host mittente.

Un ARP Announcement è simile ad un GARP perché il Source Protocol Address corrisponde con il Target Protocol Address ma **differiscono per il fatto che il primo è un messaggio di richiesta e il secondo di risposta**; inoltre, **a differenza di un ARP Probe, un ARP Announcement trasmette un’associazione completamente definita tra un indirizzo MAC sorgente ed un indirizzo IP sorgente** (gli host della rete locale che ricevono l’announcement possono aggiornare le proprie ARP Cache).

I messaggi **ARP Anouncement** vengono **usati con le stesse finalità dei messaggi GARP** e possono anche essere **usati per verificare che nella rete locale non ci siano altri host con lo stesso indirizzo IP target**; infatti, l’eventuale ricezione di una risposta è sintomo del fatto che un altro host nella rete locale è già configurato con tale indirizzo.

Una **Proxy ARP** permette di **usare la stessa subnet su due o più reti fisiche diverse**:



Il protocollo RARP (Reverse ARP) svolge il ruolo opposto ad ARP, cioè **associa un indirizzo fisico (MAC) ad uno logico (IP)**, è usato nei sistemi diskless, come X terminal (diskless workstation) o sistemi che al boot non conoscono il loro indirizzo IP, e **presuppone l’esistenza nella rete locale di un server RARP che risponde ad eventuali richieste**. Questo protocollo, oggi, è superato da DHCP che, oltre ad assegnare un indirizzo IP, **consente di fornire all’host ulteriori parametri di configurazione**, come netmask, default gateway, server DNS locale, ecc...

Il protocollo DHCP è, di fatto, un **protocollo applicativo** basato sul **modello client-server**, con il quale **un client che si vuole connettere ad una data rete**, inizialmente privo di indirizzo IP per tale

rete, **interagisce con un server che può fornirgli un IP**. Oltre all'indirizzo IP sono forniti, in genere, altri campi opzionali, come una netmask, indirizzi di server DNS locali, ecc...

L'**interazione tra client e server si effettua mediante broadcast**, essendo gli indirizzi IP reciproci ignoti alle due parti. Il client DHCP invia un messaggio di **DHCP discovery**, un messaggio in broadcast che serve a scoprire se è presente un server DHCP, il quale, se presente, invia come **risposta un DHCP offer**, sempre in broadcast, in cui offre un dato IP; a questo punto, il client può seguire la comunicazione con una **DHCP request** in cui chiede l'indirizzo IP messo a disposizione e il server risponde, infine, con una **DHCP acknowledgement**.

Nel caso in cui **client e server sono in reti diverse**, tra di essi non è possibile lo scambio di messaggi **DHCP in broadcast**. Per risolvere tale problema, interviene un **DHCP relay agent**, un'entità che consente ad un client DHCP di comunicare con un server DHCP collocato in una diversa rete e il cui compito è quello di convertire pacchetti broadcast in unicast.

ICMP: PING E TRACEROUTE

ICMP, Internet Control Message Protocol, è **un protocollo ausiliario del livello trasporto che ha lo scopo di verificare lo stato della rete** (tramite echo request ed echo reply), **riportare anomalie** (come destination unreachable, time exceeded o parameter problem), **scoprire la netmask** (con mask request e address mask reply) e **migliorare il routing** (con operazioni di redirect). I messaggi ICMP sono trasportati in datagrammi IP con Protocol Type 0x01.

Un **messaggio ICMP** è individuato da **un tipo** e da **un codice**:

Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

Applicazioni di questo protocollo sono:

- **Ping**, utilizzato per verificare la connettività a livello rete tra due host, A e B:
 - A invia un pacchetto “echo request” e, alla ricezione di tale messaggio, B risponde con un pacchetto “echo reply” dopo un certo tempo di latenza;
- **Traceroute**, utilizzato per scoprire il percorso seguito per raggiungere una certa destinazione tramite l'invio di una serie di pacchetti con TTL a partire da 1 e via via crescente:
 - Il router che, decrementando il TTL, lo azzerà invierà indietro un messaggio di “Time exceeded”, permettendo, così, di determinare il percorso fino alla destinazione.

Un esempio di traceroute è il seguente:

```
>traceroute 8.8.8.8
```

Traccia instradamento verso dns.google [8.8.8.8]
su un massimo di 30 punti di passaggio:

1	12 ms	10 ms	10 ms	100.102.0.1
2	26 ms	18 ms	11 ms	172.31.185.252
3	1885 ms	35 ms	8 ms	ru-unina-l2-rx2-na1.na1.garr.net [193.206.130.9]
4	55 ms	16 ms	25 ms	r11-na01-rs1-ba01.ba01.garr.net [185.191.180.201]
5	24 ms	19 ms	21 ms	rs1-ba01-rs1-bo01.bo01.garr.net [185.191.180.13]
6	34 ms	29 ms	28 ms	rs1-bo01-re1-mi02.mi02.garr.net [185.191.180.57]
7	36 ms	41 ms	79 ms	142.250.174.46
8	43 ms	28 ms	32 ms	72.14.238.234
9	51 ms	37 ms	32 ms	142.251.235.177
10	92 ms	48 ms	55 ms	dns.google [8.8.8.8]

Traccia completata.

Il tempo di risposta non è sempre strettamente crescente, perché ogni router può essere impegnato in maniera differente. La prima colonna indica il numero di ciascun pacchetto, come anche il rispettivo TTL, l'ultima colonna indica l'IP dell'access point, mentre le altre tre colonne indicano che sono stati inviati tre pacchetti, ciascuno a cui sarà associato quindi un diverso tempo di risposta.

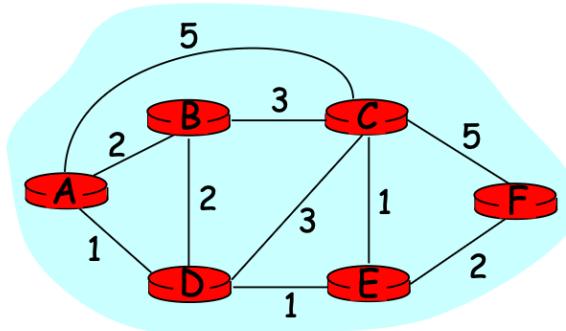
Talvolta un pacchetto può non giungere a destinazione, e ciò si indica con un asterisco *. Se nessuno dei (tre, di base) pacchetti inviati dal traceroute ottiene risposta da un access point, anziché dell'IP è presente il messaggio Richiesta scaduta. In genere ciò non significa che un tale router non è operativo, bensì che è configurato per non rispondere ad un traceroute. Si nota anche, talvolta, che il numero di access point è molto minore a quanto ci si potrebbe aspettare; ciò si può verificare, ad esempio, quando l'IP a cui si vuole accedere è servito da una CDN. Altra cosa da notare è che il primo access point è in genere sempre il router della rete di cui si fa parte; si può infatti notare la corrispondenza tra tale indirizzo IP e quello che si può ricavare dal comando ipconfig.

Come già anticipato, un traceroute invia di base pacchetti echo request, anche se in certi casi è possibile inviare pacchetti UDP con un port number casuale, dove è improbabile che ci sia un processo sulla macchina target; in questo modo tale macchina può rispondere con un messaggio “**ICMP port unreachable**” (per vari motivi di sicurezza, generalmente come indirizzi verranno restituiti degli asterischi, facendo morire il pacchetto nel router, senza alcun echo replay).

INTRODUZIONE AL ROUTING

Come già anticipato in precedenza, a livello di rete il lavoro da effettuare consiste nello scegliere il prossimo nodo della rete a cui mandare il pacchetto, con il fine di fargli raggiungere la destinazione. La rete può essere, infatti, modellata come un grafo in cui i nodi sono i router e gli archi i link fisici tra i router, ognuno dei quali implica un costo (ritardo, costo di trasmissione, congestione, ecc...); in questo contesto, a livello di rete bisogna scegliere il cammino tra i nodi a costo minimo (o basandosi su altre possibilità esplicitamente richieste, come un cammino calcolato

in base a specifici vincoli). Gli algoritmi per la gestione di una rete, come è facile intuire, sono basati tutti sulla **teoria dei grafi**.



I parametri del processo decisionale in fase di routing sono:

- **Bandwidth**, la capacità di un link (tipicamente definita in bit per secondo, o bps);
- **Delay**, il tempo necessario per spedire un pacchetto da una sorgente ad una destinazione;
- **Load**, una misura del carico di un link;
- **Reliability**, proprietà riferita, ad esempio, all'error rate di un link;
- **Hop count**, il numero di router da attraversare nel percorso dalla sorgente alla destinazione;
- **Cost**, un valore arbitrario che definisce il costo di un link (ad esempio, costruito come funzione di diversi parametri, come bandwidth, delay, packet loss, MTU, ecc...).

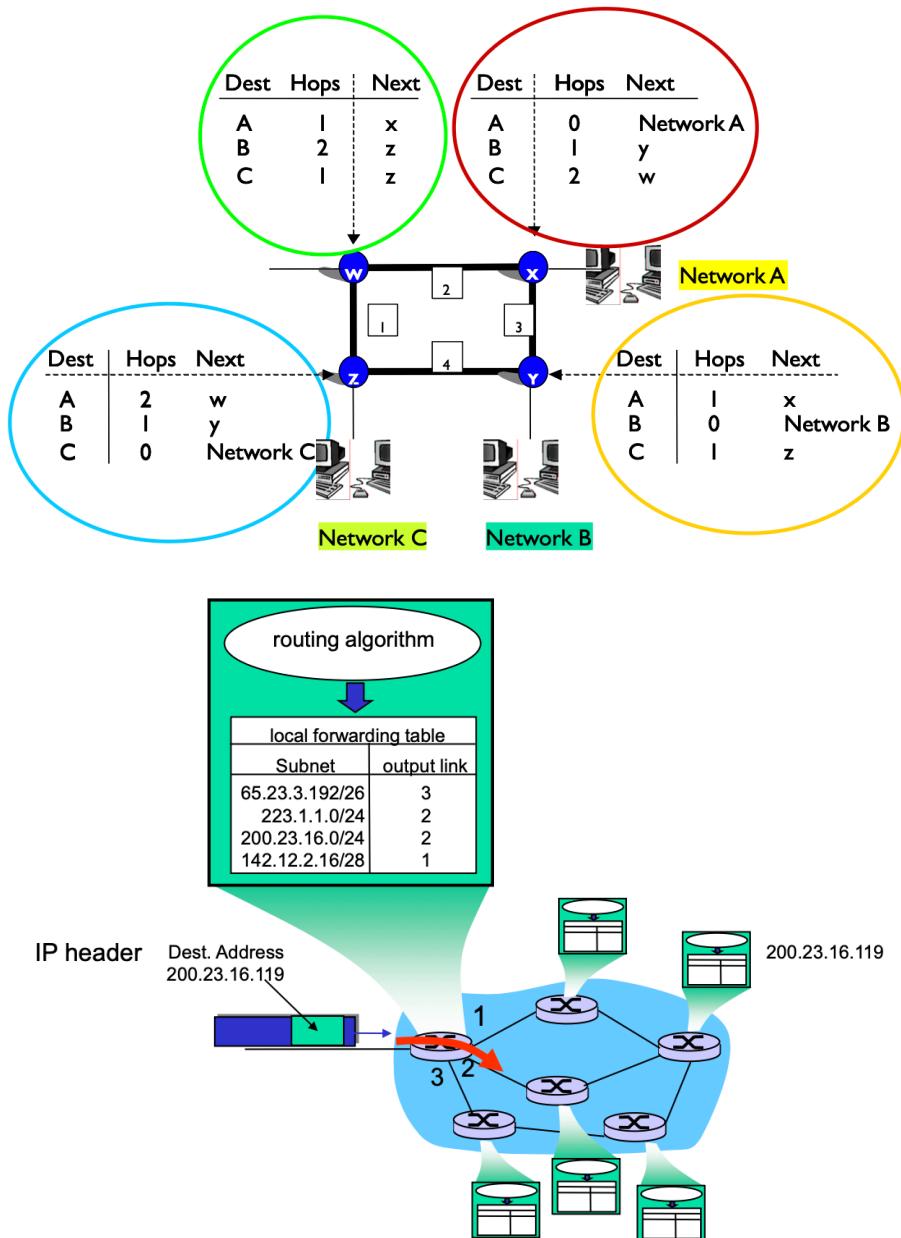
Un **algoritmo di routing** è un algoritmo che ha lo scopo di trovare il percorso con il costo minore considerando i parametri appena elencati.

Un **router esplica la funzione di forwarding dei pacchetti consultando**, per ogni pacchetto processato, **la tabella di routing**, creata come **scopo ultimo del processo di routing**. La **costruzione della tabella di routing** è un compito che può essere svolto, come già anticipato, in due modi:

- **Routing statico**, per il quale l'amministratore di rete, conoscendo la topologia della rete, determina una tantum i percorsi tra qualunque coppia sorgente-destinazione e conseguentemente configura ciascun router con le opportune regole di inoltro;
 - Qualunque modifica della topologia della rete richiede il ricalcolo dei percorsi e la riconfigurazione dei router;
- **Routing dinamico**, per il quale in ciascun router, nel control plane, opera un programma che, mediante lo scambio di informazioni con i router vicini, determina (attraverso un algoritmo) i percorsi verso qualunque destinazione e conseguentemente crea nella tabella di routing le regole corrispondenti.

Per il **routing dinamico**, si distinguono due implementazioni:

- **A controllo centralizzato**, per cui un'entità centralizzata (detta controller) acquisisce dai router informazioni circa la topologia e lo stato della rete e conseguentemente calcola i percorsi, configurando successivamente i router;
- **A controllo locale**, i router si scambiano informazioni circa lo stato della rete e, sulla base delle informazioni acquisite, ciascun router determina per ogni possibile destinazione il proprio next-hop.



Lo scambio di informazioni necessario all'esecuzione dell'algoritmo di routing è regolato da appositi protocolli di comunicazione, i **protocolli di routing**.

Il **processo di routing** è un **processo decisionale**, ogni entità che partecipa a questo processo mantiene delle informazioni, definisce il procedimento di instradamento verso le possibili destinazioni in base ad uno specifico algoritmo ed in funzione di determinate metriche ma può anche spedire informazioni di aggiornamento alle altre entità coinvolte, secondo diversi paradigmi operativi. La funzione principale di un **router** è quella di **determinare i percorsi** che i pacchetti devono seguire per arrivare a destinazione, partendo da una data sorgente: ogni router, quindi, si occupa del processo di ricerca di un percorso per l'instradamento di pacchetti tra due nodi qualunque di una rete.

In tutto ciò, ci sono dei **problematiche** da risolvere che possono essere riassunti nelle seguenti domande:

- **Quale sequenza di router deve essere attraversata?**
- **Esiste un percorso migliore** (più breve, meno carico, ...)?
- **Cosa fare se un link si guasta?**

- Come trovare una soluzione robusta e scalabile?

Alcune **tecniche di routing** sono di seguito riassunte:

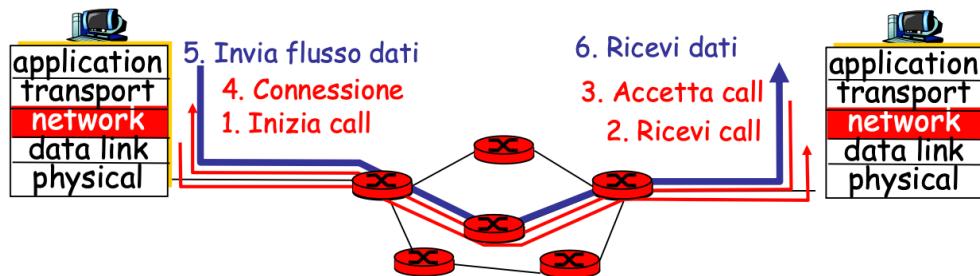
- **Routing by Network Address**

Ogni pacchetto contiene l'**indirizzo del nodo destinatario**, che viene usato come chiave di accesso alle tabelle di instradamento; è una tecnica usata tipicamente nei protocolli non orientati alla connessione (come IPv4 e IPv6, bridge trasparenti, OSI CLNP, ecc...).

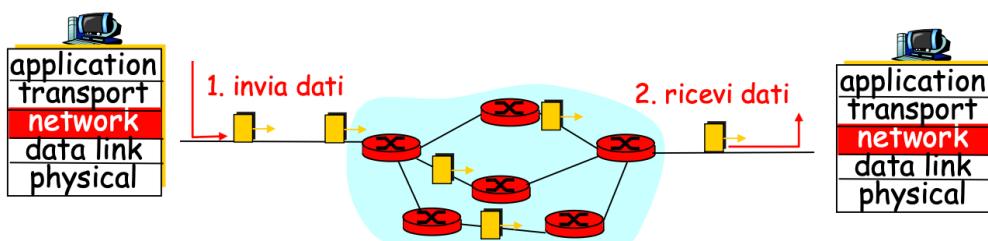
- **Label Swapping**

Ogni pacchetto è marcato con un **label** che identifica la connessione e che viene usato come chiave per determinare l'instradamento; è una tecnica generalmente usata nei **protocolli orientati alla connessione** (X.25, ATM, MPLS, ecc...).

Il routing nelle reti a circuiti virtuali avviene tramite una fase di call setup a livello rete, solo dopo la quale la connessione tra due nodi è stabilita e il trasferimento dei dati possibile:



In una **rete a datagrammi**, invece, la fase di call setup non è effettuata e non esiste il concetto di **connessione tra due router**; i pacchetti sono indirizzati usando un ID di destinazione e pacchetti fra la stessa coppia sorgente-destinazione possono seguire strade diverse:



Chiaramente, la **reliability** del primo si scontra con la **semplicità e rapidità** del secondo, il che ha fatto preferire storicamente le reti a **datagrammi** (che nel tempo hanno acquistato anche loro una certa reliability).

ROUTING DISTANCE VECTOR

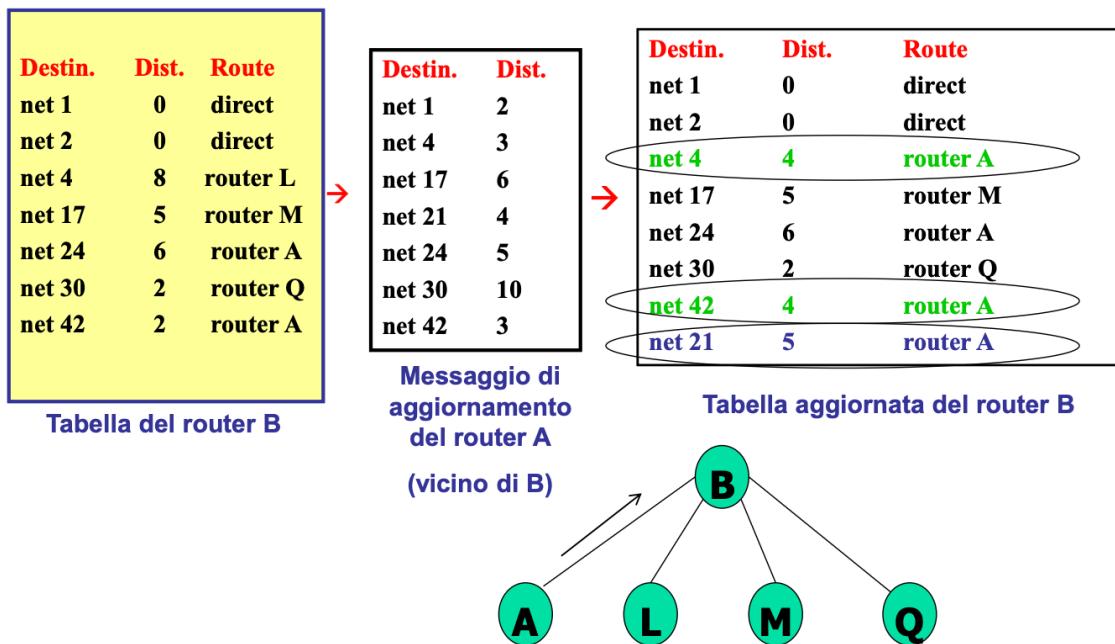
Per **routing Distance Vector** si intende un **algoritmo di routing** per il quale **ogni router** mantiene una tabella di tutti gli **instradamenti noti** (inizialmente, solo le reti a cui è connesso direttamente), formata da entry che indicano una rete raggiungibile, il **next hop** e il **numero di hop** necessari per raggiungere la destinazione, e per cui, periodicamente, **ogni router invia a tutti i vicini** (due

router sono vicini se sono collegati alla stessa rete fisica) un messaggio di aggiornamento contenente tutte le informazioni della propria tabella (vettore delle distanze, **distance vector**, da cui il nome dell'algoritmo); i router che ricevono tale messaggio aggiornano la tabella nel seguente modo:

- Eventuale modifica delle informazioni relative a cammini già noti;
- Eventuale aggiunta di nuovi cammini;
- Eventuale eliminazione di cammini non più disponibili.

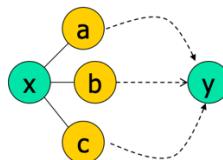
In altre parole, un router ricalcola le sue tabelle se cade una linea attiva o se riceve un distance vector, da un nodo adiacente, diverso da quello memorizzato. Se le tabelle risultano diverse da quelle precedenti, si invia ai nodi adiacenti un nuovo distance vector, mentre il calcolo di routing consiste nella fusione di tutti i distance vector delle linee attive.

Un esempio di tabella di routing per un algoritmo Distance Vector:



Definito $d_x(y)$ il costo del percorso (a costo minore) tra x e y, allora:

$$d_x(y) = \min_V \{c(x, V) + d_V(y)\}$$



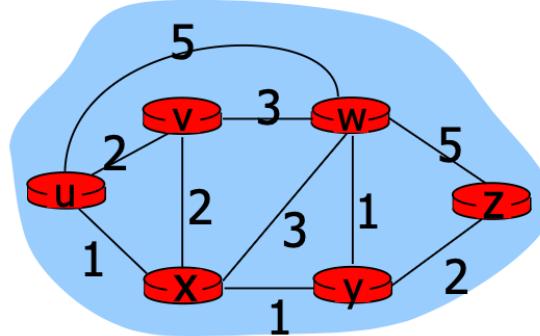
Con V nodo adiacente ad x. La relazione appena enunciata è detta **Equazione di Bellman-Ford** e ha come risultato la determinazione del vicino V che induca al costo minore tra tutti, determinando così il **next-hop**. Ad esempio, sapendo che:

$$d_v(z) = 5; d_x(z) = 3; d_w(z) = 3$$

Si può affermare:

$$d_u(z) = \min\{c(u, v) + d_v(z), c(u, x) + d_x(z), c(u, w) + d_w(z)\} = \min\{7, 4, 8\}$$

E concludere, quindi, **che il next-hop con il minor costo è x** (nonostante bisogna attraversare il nodo y).

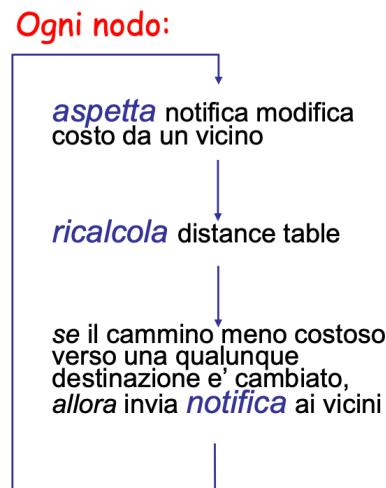


Sia $D_x(y)$ la stima del minimo costo da x a y e $D_x = [D_x(y) : y \in V]$ il distance vector di x , sapendo che il nodo x sappia il costo del link con ogni vicino ($c(x, V)$) e che mantenga i distance vector proprio e dei vicini ($\forall v \in V, x$ mantiene $D_v = [D_v(y) : y \in V]$), allora:

$$D_x(y) = \min\{c(x, v) + D_v(y)\}, \forall y \in V$$

Sapendo anche che **ogni nodo periodicamente invia il proprio distance vector stimate ai vicini** e che, quando un nodo x riceve un nuovo distance vector da un vicino, aggiorna il proprio distance vector usando la formula di Bellman-Ford. Sotto “condizioni naturali”, il distance vector stimato $D_x(y)$ converge all’attuale minor costo $d_x(y)$.

L’algoritmo Distance Vector è un algoritmo iterativo e asincrono, per cui **ogni iterazione è causata da un cambiamento di costo di un collegamento o dal messaggio di un vicino**, e distribuito, per cui **ogni nodo contatta i vicini solo quando un suo cammino di costo minimo cambia** (così, a loro volta, i vicini).



Ad ogni nodo x si ha:

1. Inizializzazione;
2. Per tutti i nodi adiacenti v :
 - a. $D_x(*, v) = \infty$ (con * che indica “per ogni riga”);
 - b. $D_x(v, v) = c(x, v)$;

3. Per tutte le destinazioni y:

a. Manda $\min_w D(y, w)$ ad ogni vicino.

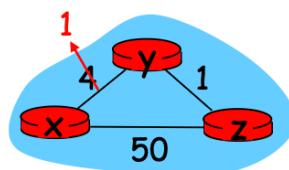
```

→ 8 loop
  9   aspetta (fino a quando vedo una modifica nel costo di un
  10    collegamento oppure ricevo un messaggio da un vicino v)
  11
  12   if (c(x,v) cambia di d)
  13     { cambia il costo a tutte le dest. via vicino v di d }
  14     { nota: d puo' essere positivo o negativo }
  15     per tutte le destinazioni y:  $D^X(y,v) = D^X(y,v) + d$ 
  16
  17   else if (ricevo mess. aggiornamento da v verso destinazione y)
  18     { cammino minimo da v a y e' cambiato }
  19     { V ha mandato un nuovo valore per il suo  $\min_w D^V(y,w)$  }
  20     { chiama questo valore "newval" }
  21     per la singola destinazione y:  $D^X(y,v) = c(x,v) + newval$ 
  22
  23   if hai un nuovo  $\min_w D^X(y,w)$  per una qualunque destinazione y
  24     manda il nuovo valore di  $\min_w D^X(y,w)$  a tutti i vicini
  25
  26 forever
```

Per questo algoritmo:

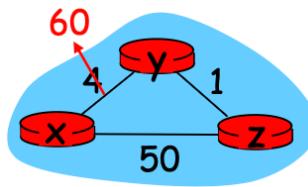
- **Vantaggi:**
 - È facile da implementare;
- **Svantaggi:**
 - Ogni messaggio contiene un'intera tabella di routing;
 - Si ha una lenta propagazione delle informazioni sui cammini, convergente alla velocità del router più lento;
 - Se lo stato della rete cambia velocemente, le rotte possono risultare inconsistenti e si possono innescare dei loop a causa di particolari variazioni della topologia;
 - È difficile capirne e prevederne il comportamento su reti grandi, nessun nodo ha una mappa della rete.

Per **convergence speed** (o velocità di convergenza) si intende la **velocità con cui i router imparano circa i cambiamenti di stato nella rete**. Con un **routing a Distance Vector**, buone notizie viaggiano velocemente e cattive notizie lentamente.



Si supponga di avere un **decremento nel costo dei link**: il nodo y individua il cambiamento di **costo locale**, aggiorna le informazioni di routing, ricalcola il **distance vector** e, se cambia, notifica i vicini all'istante t_0 . All'istante t_1 , z riceve l'aggiornamento da y e aggiorna la propria tabella,

ricalcola un nuovo least cost to x e invia il proprio distance vector ai propri vicini. All'istante t_2 , y riceve l'aggiornamento di z e aggiorna la propria distance table, pur non cambiando il proprio least cost to x; pertanto, y non invia alcun messaggio a z. Questo schema di evoluzione riassume il caso “**Buone notizie viaggiano velocemente**”.



Si supponga di avere un **incremento nel costo dei link**: all'istante t_0 il nodo y rileva il cambiamento, aggiorna il proprio costo fino a x a 6, dal momento in cui precedentemente z aveva comunicato a y un costo di raggiungimento pari a 5, ma:

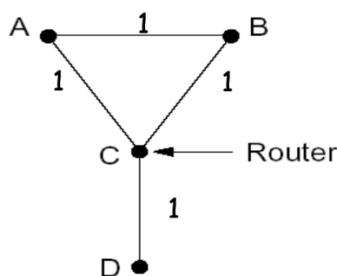
$$\min\{60 + 0, 5 + 1\} = 6$$

Si è raggiunto un **routing loop**: i pacchetti destinati a x da y fanno avanti e indietro tra y e z fino all'**infinito** (o finché il loop non si rompe). All'istante t_1 , z riceve l'aggiornamento da y e aggiorna il proprio cost to x a 7, dal momento in cui:

$$\min\{50 + 0, 1 + 6\} = 7$$

L'algoritmo avrà bisogno di diverse iterazioni per stabilizzarsi e, perciò, questo schema di evoluzione riassume il caso “**Cattive notizie viaggiano lentamente**”.

Una **possibile soluzione** consiste nel cosiddetto **poisoned reverse**, per il quale se z raggiunge x tramite y, dice a y che la sua distanza per x è infinita, in modo da **non andare a x tramite y ed ingoiare il boccone del link ad elevato costo pur di non finire in loop**. Questa soluzione non è priva di falle, esistono condizioni per cui anche il poisoned reverse fallisce; ad esempio:



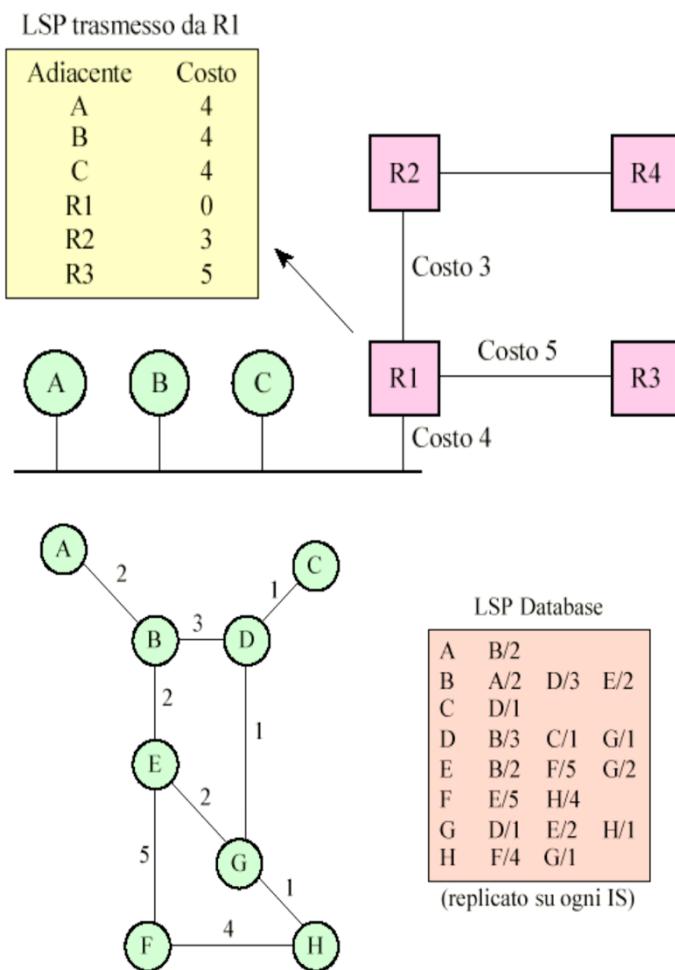
Il link tra C e D si interrompe, con C che imposta la sua distanza da D a ∞ ; però, A userà B per andare a D e B userà A per andare a D. Dopo questi update, sia A che B riporteranno un nuovo percorso da C a D (diverso da ∞).

ROUTING LINK-STATE E L'ALGORITMO DI DIJKSTRA

Per **algoritmo link-state** si intende un algoritmo per il quale **ogni router impara il proprio ambito locale** (linee e nodi adiacenti), **trasmette queste informazioni a tutti gli altri router della rete** tramite un **Link State Pocket (LSP)** e **memorizza le omonime trasmesse dagli altri router**, costruendo poi una **mappa della rete**; inoltre, calcola, in maniera indipendente, le sue tabelle di

intradamento applicando alla mappa della rete l'algoritmo di Dijkstra, noto come Shortest Path First (SPF). L'approccio appena mostrato è utilizzato nello standard ISO 10589 (protocollo IS-IS) e nel protocollo OSPF (adottato in reti TCP/IP).

Come già anticipato, ogni router genera un Link State Packet (LSP) contenente lo stato di ogni link connesso al router, l'identità di ogni vicino connesso all'altro estremo del link, il costo del link, il numero di sequenza per l'LSP, la checksum e il lifetime (la validità di ogni LSP è limitata nel tempo). Un LSP viene generato o periodicamente o quando viene rilevata una variazione nella topologia locale (nelle adiacenze), ossia se viene riconosciuto un nuovo vicino, se il costo verso un vicino è cambiato o se si è persa la connettività verso un vicino precedentemente raggiungibile (si noti che le possibili condizioni di rigenerazione di un LSP corrispondono ai parametri potenzialmente mutabili dell'LSP stesso). La trasmissione di un LSP avviene in flooding su tutti i link del router, mentre i pacchetti LSP memorizzati nei router formano una mappa completa e aggiornata della rete, detta Link State Database.



La rappresentazione del LSP-DB più appropriata per applicare l'algoritmo di Dijkstra è la seguente:

SORGENTE

↓

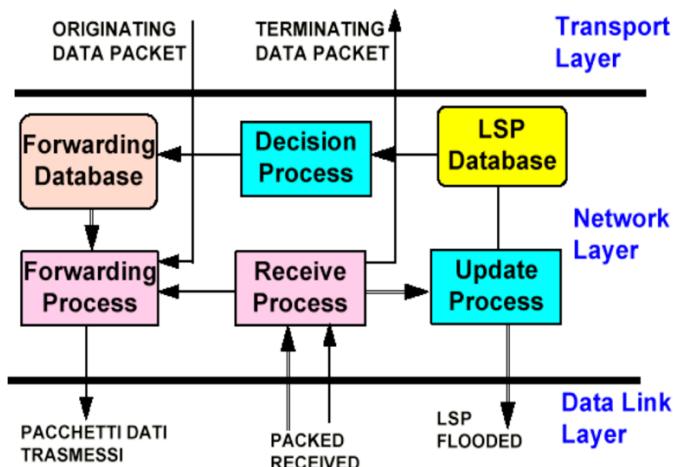
	A	B	C	D	E	F	G	H
A	0 2							
B	2 0		3 2					
C			0 1					
D		3 1	0			1		
E		2		0 5	2			
F				5 0		4		
G				1 2		0 1		
H					4 1	0		

DESTINAZIONE →

All'atto della ricezione di un LSP, il router compie le seguenti azioni:

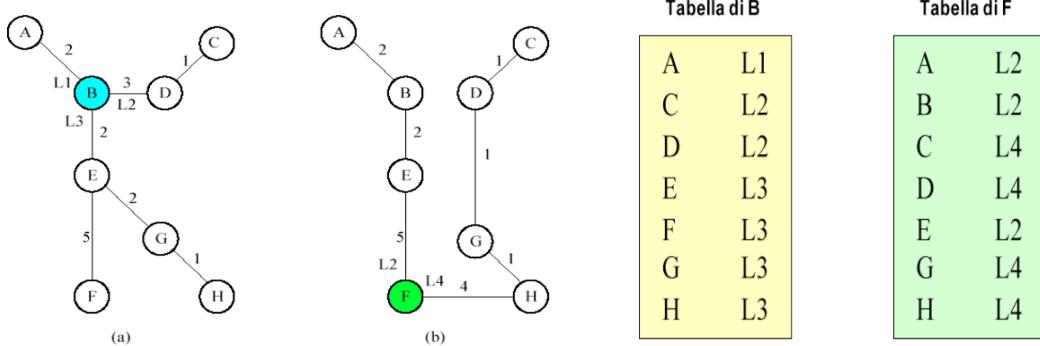
- Se non ha mai ricevuto un LSP da quel router o se l'LSP è più recente di quello precedentemente memorizzato, memorizza il pacchetto e lo ritrasmette in flooding su tutte le linee eccetto quella da cui l'ha ricevuto;
- Se l'LSP ha lo stesso numero di sequenza di quello posseduto, non fa nulla;
- Se l'LSP è più vecchio di quello posseduto, trasmette al mittente il pacchetto più recente.

Il router elabora il Link State Database per produrre il Forwarding Database: si pone come radice dello shortest-path tree il LSP-DB e si cerca lo shertes path per ogni nodo destinazione, memorizzando il vicino (o i vicini) che è sullo shortest path verso ogni nodo destinazione. Il Forwarding Database contiene l'insieme delle coppie (path, vicino) e la dimensione di tale insieme.



Per questo approccio:

- **Vantaggi:**
 - Possono essere gestite reti di grandi dimensioni;
 - Si ha una rapida convergenza;
 - Difficilmente viene generato un loop (e comunque si è in grado di identifierli e interromperli rapidamente);
 - È facile da capire: ogni nodo ha la mappa della rete;
- **Svantaggi:**
 - La realizzazione è molto complessa (la prima implementazione ha richiesto alla Digital 5 anni).



Ogni nodo ha a disposizione il grafo della rete (i nodi sono i router e gli archi, ad ognuno dei quali è associato un costo, le linee di collegamento tra router) ed **usa l'algoritmo di Dijkstra per costruire lo Shortest Path Tree del grafo**, ovvero l'albero dei cammini di costo minimo. Ad **ogni nodo è assegnata un'etichetta che rappresenta il costo massimo per raggiungere quel nodo**, mentre l'algoritmo **le modifica cercando di minimizzarne il valore e di renderle permanenti**. Formalmente:

- **La topologia della rete è nota a tutti i nodi:**
 - La diffusione è realizzata via “link state broadcast”;
 - Tutti i nodi hanno la stessa informazione;
- **Si calcola il percorso minimo da un nodo a tutti gli altri:**
 - L'algoritmo fornisce la tavola di routing per quel nodo;
- **Iterativamente un nodo**, dopo k iterazioni, **conosce i cammini meno costosi verso k destinazioni**.

La notazione è la seguente:

- $c(i, j)$, definito come il costo del collegamento da i a j (è, per definizione, ≥ 0 e commutativo, nel caso in cui non ci sia collegamento è infinito);
- $D(v)$, definito come il costo corrente del percorso, dalla sorgente al nodo v ;
- $p(v)$, definito come il predecessore (collegato a v) lungo il cammino, dalla sorgente a v ;
- N , definito come l'insieme di nodi per cui la distanza è stata trovata.

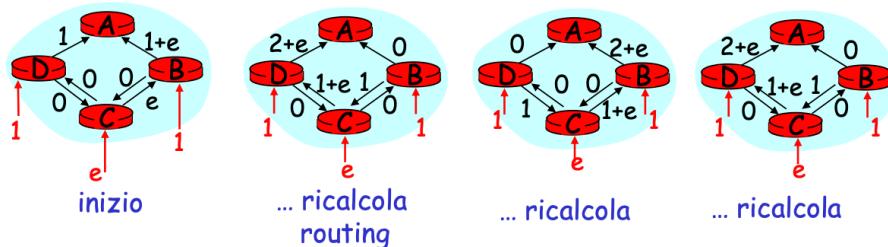
L'algoritmo può, così, essere enunciato in pseudocodice:

```

1 Inizializzazione:
2    $N = \{A\}$ 
3   per tutti i nodi  $v$ 
4     if ( $v$  è adiacente a  $A$ )
5       then  $D(v) = c(A, v)$ 
6       else  $D(v) = \infty$ 
7
8 Loop
9   sia  $w$  non in  $N$  tale che  $D(w)$  è minimo
10  aggiungi  $w$  a  $N$ 
11  aggiorna  $D(v)$  per ogni  $v$  adiacente a  $w$  e non in  $N$ :
12     $D(v) = \min(D(v), D(w) + c(w, v))$ 
13  {il nuovo costo fino a  $v$  è o il vecchio costo, oppure il costo del
      cammino più breve fino a  $w$  più il costo da  $w$  a  $v$ }
15 fino a quando tutti i nodi sono in N

```

L'algoritmo consiste in un passo di inizializzazione più un ciclo di durata pari al numero di nodi della rete; al termine si avranno i percorsi più brevi, dal nodo sorgente a tutti gli altri nodi. Se il costo di un link è proporzionale al traffico su quel link, allora sono possibili oscillazioni; per risolvere questo problema si può evitare la sincronizzazione nell'invio dei messaggi dei router.



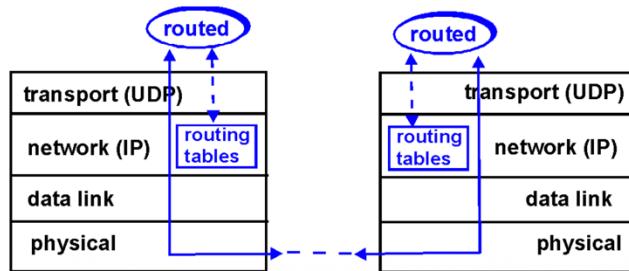
Si è visto che con l'algoritmo Link State ogni router necessita idealmente di avere prima un LSP Database completo, e quindi è necessario attendere la completa trasmissione di tutti i LSP, per poi applicare l'algoritmo di Dijkstra, che impiega un certo tempo di computazione. C'è onere temporale e computazionale, che aumenta ulteriormente se i collegamenti possono variare di costo, potenzialmente in maniera indefinita se il costo dipende dal traffico. L'algoritmo Distance Vector, invece, applica una sola equazione ad informazioni ottenute dall'ambito locale. È computazionalmente molto semplice ma c'è onere temporale proporzionale al numero di nodi, risponde rapidamente a riduzioni di costi ma molto lentamente ad aumenti, in maniera proporzionale, e alle rimozioni di nodi c'è il rischio di attesa indefinita.

PROTOCOLLI DI ROUTING

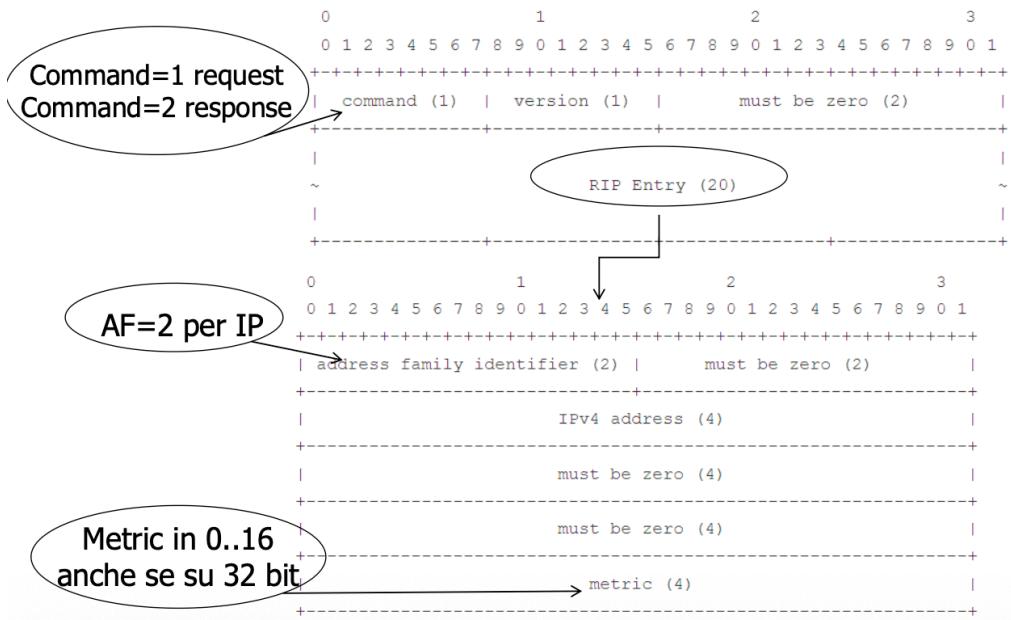
I protocolli di routing più conosciuti sono RIP e IGP. Il primo, più diffuso del secondo sebbene non sia necessariamente il migliore, risale al 1969 (definito in RFC1058 per la versione 1 e RFC 2453 per la versione 2) ed è implementato su tutti i sistemi UNIX (dal 1982) dal programma routed; inoltre, è basato sulla trasmissione broadcast, rendendolo adatto a reti broadcast (Ethernet) ma non a reti WAN, e implementa l'algoritmo distance vector.

RIP non fa distinzione formale tra reti ed host singoli, le routing entry possono puntare ad un singolo host (anche se è conveniente usare reti che aggregano insiemi di indirizzi) e divide le entità in attive e passive: le entità passive possono solo ricevere messaggi (ad esempio, gli host), le entità attive possono anche spedirli (ad esempio, i router). Le entità attive mandano un messaggio in broadcast ogni trenta secondi (messaggi RIP response), contenenti la tabella di routing e il numero di hop (unica metrica utilizzata), ognuno dei quali contiene fino a 25 reti destinazione; un host aggiorna una rotta solo se ne apprende una strettamente migliore (messaggio di advertisement), con ogni informazione che ha un timeout di 180 secondi oltre i quali il link è considerato morto e le route che attraversano quel vicino sono rese non valide, mandando un nuovo advertisement ai vicini, i quali propagano l'informazione (se le loro tavole cambiano) facendo diffondere rapidamente la notizia dell'interruzione a tutta la rete.

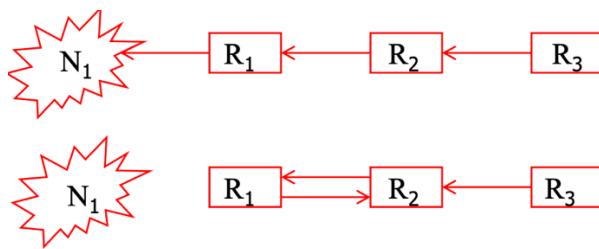
RIP è un protocollo di livello applicativo: le tavole di routing RIP sono elaborate da un processso a livello applicazione detto routed. I pacchetti sono ricevuti ed inviati utilizzando il protocollo UDP (in particolare, il port 520), dal momento in cui piccoli messaggi regolari non necessitano del meccanismo del windowing, di un meccanismo di handshaking o di ritrasmissioni.



Il formato dei messaggi RIP è il seguente:



Il protocollo non individua esplicitamente i cicli, assume che tutte le rotte pubblicate siano corrette. Inoltre, per prevenire inconsistenze, fissa una distanza massima di routing (ad esempio, 15, il che significa che la distanza 16 non è raggiungibile), anche se soffre di Slow Convergence, avendo aggiornamenti delle rotte che si propagano lentamente.



Ad esempio:

- Il collegamento tra R_1 e N_1 cade;
- R_2 invia la propria tabella a R_1 ;
- R_1 invia la propria tabella:
 - R_2 utilizza una nuova rotta lunga 4, passante per R_1 ;
- Si prosegue così fino a raggiungere distanza 16.

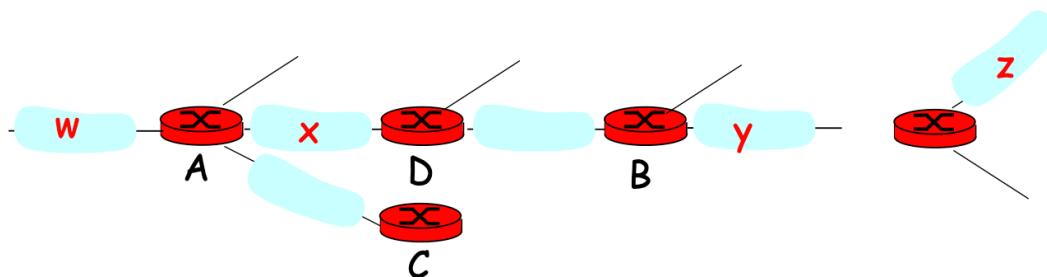
L'unica metrica usata è l'hop count, rendendo il routing indipendente dal traffico sulla rete e, pertanto, non adatto a gestire eventuali congestioni; inoltre, RIP crede a tutte le informazioni che

gli arrivano, con un router malizioso che può potenzialmente indurre gli altri a modificare le loro tabelle a proprio vantaggio. In quest'ottica, il protocollo è accettabile all'interno dello stesso AS (Autonomous System) ma non tra AS distinti.

Sono state studiate diverse tecniche per combattere la slow convergence (nessuna delle quali però risolve completamente il problema):

- **Split horizon** (obbligatorio), per il quale R_2 non invia a R_1 le rotte che passano per R_1 , prevenendo solo i loop tra due router;
- **Split horizon with poisoned reverse** (opzionale), per il quale R_2 dichiara ad R_1 a distanza infinita le reti che R_2 raggiunge attraverso R_1 stesso, producendo una più veloce eliminazione dei loop ma non eliminando del tutto la possibilità dei loop che si creano tra nodi adiacenti;
- **Triggered updates**, per il quale appena un router aggiorna la propria tabella di routing invia i distance vector aggiornati ai propri vicini;
- **Hold down**, per il quale R_2 , dopo aver ricevuto il messaggio di R_1 , ignora tutte le rotte per N_1 per un certo periodo di tempo (60 secondi), preservando i loop per tutta la durata dell'hold time.

Tabella di routing nel router D



Destination Network	Next Router	Num. of hops to dest.
W	A	2
Y	B	2
Z	B	7
X	--	1
....

Destination	Gateway	Flags	Ref	Use	Interface
127.0.0.1	127.0.0.1	UH	0	26492	lo0
192.168.2.	192.168.2.5	U	2	13	fa0
193.55.114.	193.55.114.6	U	3	58503	le0
192.168.3.	192.168.3.5	U	2	25	qaa0
224.0.0.0	193.55.114.6	U	3	0	le0
default	193.55.114.129	UG	0	143454	

Il contenuto della routing table, implementando il protocollo RIP, è:

- **Address/Destination**, indirizzo IPv4 dell'host o della rete destinazione;
- **Router/Gateway**, il primo router lungo la route per la destinazione;
- **Interface**, la rete fisica che deve essere usata per raggiungere il prossimo router;
- **Metric**, un numero che indica la distanza dalla destinazione (è la somma dei costi dei link che bisogna attraversare per raggiungere la destinazione);
- **Timers**, il tempo tra due update della stessa entry nella tabella;

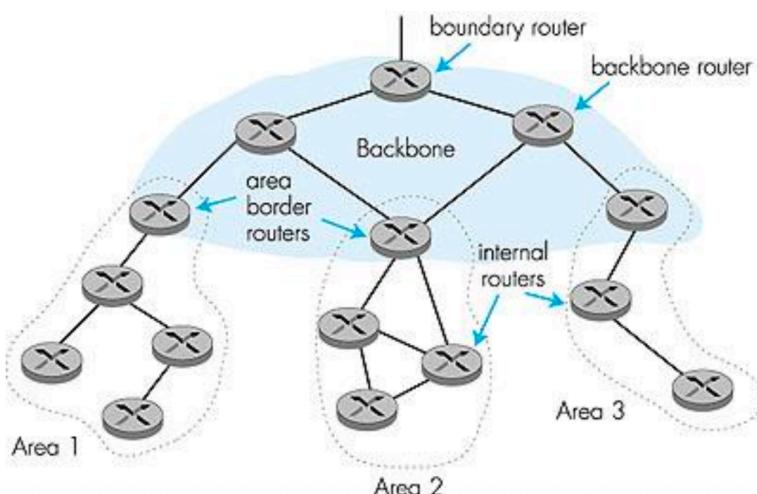
- **Flags:**
 - **U**, la rotta è disponibile;
 - **G**, la rotta utilizza un router intermedio (se non è presente, la destinazione è collegata direttamente);
 - **H**, la destinazione è un host e non una rete;
 - **D**, la rotta è creata da un redirect;
 - **M**, la rotta è modificata da un redirect.

RIP non gestisce le netmask, non consentendo di pubblicizzare rotte con subnetting e CIDR; per questo motivo, si è resa necessaria una seconda versione del protocollo, RIP2, modificando la struttura dei pacchetti RIP aggiungendo nuovi campi per la netmask, il next hop (eliminando anche il problema dei loop e della slow convergence) e utilizzando 0.0.0.0 per la rotta di default.

IGRP è un protocollo proprietario CISCO basato su Distance Vector e che usa diverse metriche di costo (ritardo, banda, affidabilità, ecc...), con tavole di routing scambiate tramite TCP solo quando si modificano i costi. IGRP supporta il multipath routing a costi differenziati: se esistono più rotte per la stessa destinazione, il carico è distribuito tra di esse proporzionalmente al costo delle rotte.

EIGRP (enhanced IGRP, le cui specifiche sono state rese pubbliche da CISCO nel 2013 e descritte in RFC 7868) è una versione “migliorata” del protocollo che supporta indirizzamenti classless con maschere di sottorete a lunghezza variabile (VLSM) e che fa uso di DUAL (Distributed Updating Algorithm), algoritmo di routing che garantisce assenza di cicli (loop free) congelando la tabella di routing dopo l’incremento di una distanza fino a quando tutti i nodi influenzati saranno informati del cambiamento.

OSPF (Open Shortest Path First) è un protocollo IGP (Interior Gateway Protocol, usati per il routing all’interno di AS) basato su tecnica link state, open (pubblicamente disponibile), descritto in RFC 1131 del 1989 e (v2) in RFC 2328 del 1998, che supporta il routing gerarchico (la rete è suddivisa in aree), per il quale i pacchetti Link State (LSP) sono trasmessi in flooding nella porzione di rete gestita da OSPF (area) e grazie a cui ogni nodo conosce la topologia della rete. Ogni link associato ha un costo configurabile dall’amministratore di sistema e, minore il costo, maggiore la probabilità che l’interfaccia è usata per inoltrare il traffico.

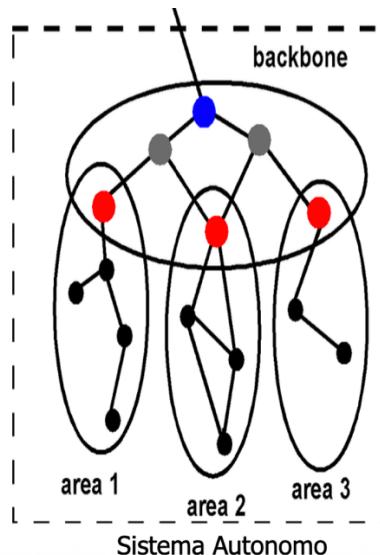


OSPF prevede quattro tipi di router:

- **Internal router**, in cui tutte le interfacce appartengono alla stessa area;

- **Area border router**, in cui le interfacce appartengono a due o più aree distinte;
- **Backbone router**, in cui almeno un’interfaccia appartiene all’area 0;
- **Autonomous system boundary router**, in cui almeno una delle interfacce utilizza un diverso protocollo di routing o appartiene ad un diverso AS.

La gerarchia ad aree prevista da OSPF è composta da local area e backbone (o area zero). Gli Advertisement Link-state non lasciano le rispettive aree e i nodi in ogni area, pur avendo una topologia dettagliata dell’area, conoscono solo la direzione verso reti in altre aree. Gli Area border router “riassumono” distanze a reti nell’area di competenza e le comunicano ad altri router di tipo Area border, mentre i Boundary router si connettono ad altri AS.



Per i principi di gerarchia di OSPF, **ogni area non deve conoscere la topologia delle altre aree; inoltre, il protocollo incapsula i propri messaggi direttamente in datagrammi IP con numero di protocollo 89, non usando un protocollo di trasporto** (scelta ben diversa da RIP e BGP e gestendo autonomamente i meccanismi di affidabilità della comunicazione). **Con OSPF è possibile anche pubblicizzare rotte apprese da altri AS.**

Per quanto riguarda la sicurezza, **tutti i messaggi OSPF sono autenticati** (per prevenire attacchi) **con un meccanismo di autenticazione semplice** (password in chiaro) **o con MD5** (viene trasmesso in ogni pacchetto il proprio hash a cui è stata aggiunta una chiave segreta non trasmessa ma nota a tutti i router). **Sono possibili cammini multipli** (con lo stesso costo), mentre **in RIP ne è possibile uno solo, garantendo il bilanciamento del carico tra percorsi multipli**, ed è supportato il **multicast integrato** (MOSPF, usa lo stesso database di OSPF).

Ogni router manda periodicamente un messaggio di HELLO ad ogni router direttamente collegato, verificando che sia raggiungibile; i router si scambiano informazioni sulla topologia della rete e le propagano con la tecnica del flooding (un pacchetto LSP, una volta ricevuto, se non è stato già ricevuto in precedenza, viene ritrasmesso su tutte le interfacce tranne quella da cui è arrivato) **e ognuno di essi periodicamente pubblica lo stato dei suoi link.**

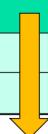
Tutti i pacchetti OSPF hanno un header comune:

VERSION(1)	TYPE	MESSAGE LENGTH
SOURCE IP ADDRESS		
AREA ID		
CHECKSUM	AUTHENTICATION TYPE	
AUTHENTICATION(ottetti 0-3)		
AUTHENTICATION(ottetti 4-7)		

Mentre il payload per un messaggio di HELLO:

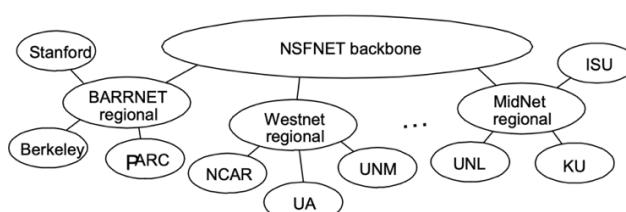
OSPF header		
network mask		
source IP address		
dead timer	hello inter	gway prio
designated router		
backup designated router		
neighbor 1 IP address		
neighbor 2 IP address		
...		

Un messaggio LSA, invece:

OSPF header	
Number of link status advertisements	
Link status advertisement 1	
Link status advertisement 2	
....	
	Link age
	Link type
	Link ID
	Advertising router
	Link sequence number
	Link checksum
	length

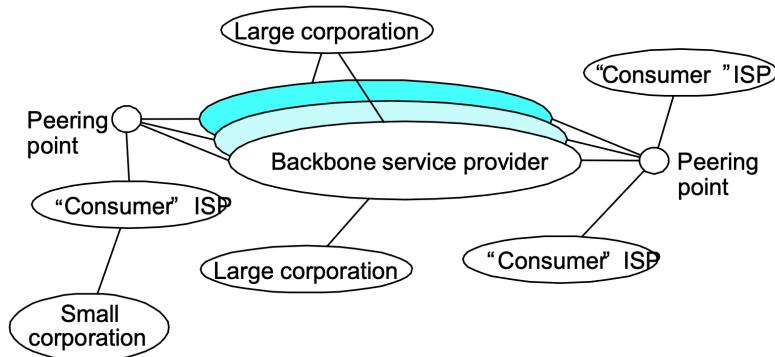
IL ROUTING GERARCHICO E GLI AUTONOMOUS SYSTEMS

Negli anni 80 l'architettura di Internet era ancora molto semplice: c'era un'unica rete backbone e ogni rete fisica era collegata alla backbone da un core router, ognuno dei quali conosceva le rotte per tutte le reti fisiche.

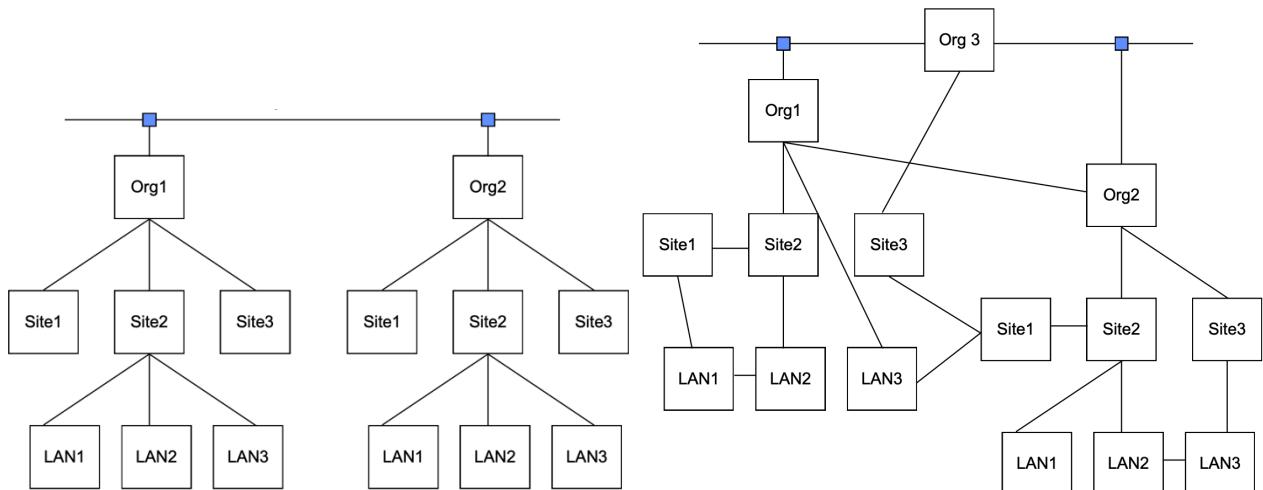


Al giorno d'oggi, non è accettabile la possibilità che ci sia un unico proprietario per la backbone di tutta la rete e non tutte le reti fisiche possono essere collegate direttamente a questa unica backbone; la soluzione non è scalabile, al crescere del numero di core router diventa impossibile mantenerli tutti aggiornati.

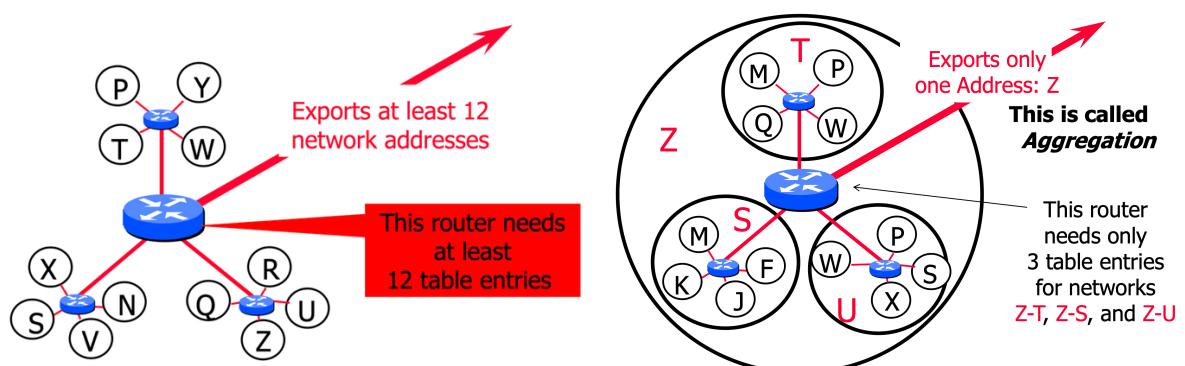
Per questo motivo, **ad oggi Internet è strutturato in reti con Peer Backbone** (o sistema di routing gerarchico): è prevista l'esistenza di diverse dorsali, i cui amministratori devono concordare una politica di routing per evitare la creazione di router, e con i core router delle diverse reti che devono scambiare informazioni sulle rotte reciprocamente.



Confrontando il prima e il dopo:



Dal punto di vista pratico, **la differenza tra il Flat Network Addressing e il Hierarchical Network Addressing risiede nella necessità, per i core router, di richiedere un minor quantitativo di entry nella tabella di routing**:



Perciò, nel IP forwarding, non è più necessario un lookup ad exact match ma può essere sfruttato un criterio di best match.

Internet, ad oggi, è strutturato come un insieme di Autonomous System, collezioni di un numero finito di reti amministrate da una singola entità, garantendo una gestione delle informazioni di routing all'interno dell'AS stesso più agevolata. Ogni AS, identificato da un AS number (un nome di 16 bit), è responsabile del routing all'interno delle proprie reti (pratica detta routing interno) ma tra di loro devono scambiarsi informazioni di raggiungibilità (pratica detta routing esterno), garantendo la sicurezza e la consistenza delle informazioni memorizzate nelle tabelle dei router.

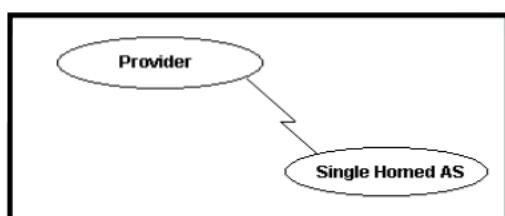
Le tabelle di routing interne di un AS sono mantenute dall'Interior Gateway Protocol (IGP): i messaggi IGP sono scambiati tra router appartenenti allo stesso AS e contengono solo informazioni sulle reti dell'AS (RIP, ovvero distance vector, OSPF, ovvero link state, IGRP, ovvero Interior Gateway Routing Protocol). Le tabelle di routing esterne di un AS sono mantenute dall'Exterior Gateway Protocol (EGP): i messaggi EGP sono scambiati tra router designati dai rispettivi AS, detti border router, e contengono informazioni sulle rotte conosciute dai due AS coinvolti. EGP è ormai obsoleto e gradualmente lo si sta sostituendo con BGP, Border Gateway Protocol, che usa un approccio path vector.

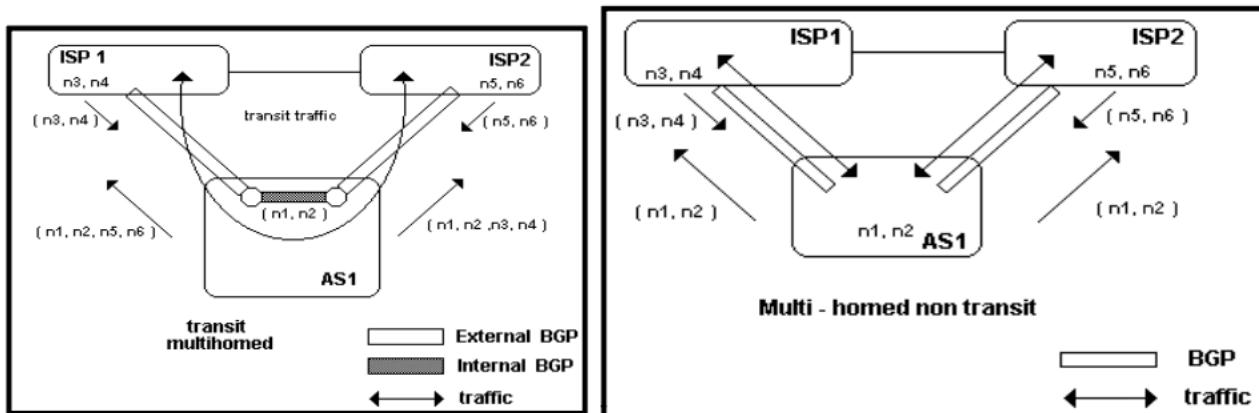
Per comprendere la differenza tra routing interno (o intra-AS) e routing esterno (o inter-AS) è possibile fare riferimento alla seguente tabella:

	Routing esterno	Routing interno
Politica	Si concentra su aspetti politici (quale provider scegliere o evitare)	Si applica all'interno dell'AS, la politica applicata è coerente e non dipende se non dall'organizzazione
Dimensioni	Si realizza un routing gerarchico	Si diminuisce il traffico per aggiornare le tabelle di routing
Prestazioni	Gli aspetti politico-amministrativi prevalgono	L'ottimizzazione delle prestazioni prevale

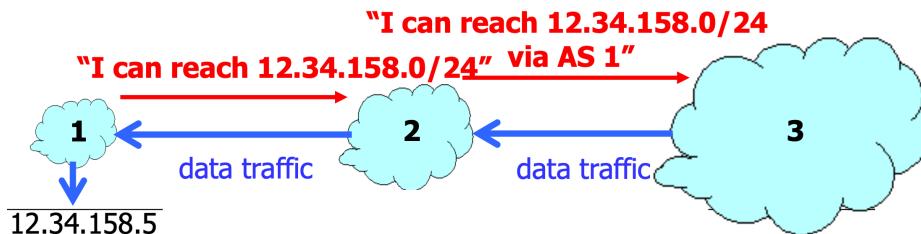
Esistono diversi tipi di AS:

- **Single border router** (stub o single-homed), sono utilizzati in piccole corporate;
- **Multiple border router** (multi-homed), a loro volta si dividono in:
 - **Transit** (provider), accetta di essere attraversato dal traffico diretto ad altri AS;
 - **Non transit** (grandi corporate), non accetta di essere attraversato dal traffico diretto ad altri AS.

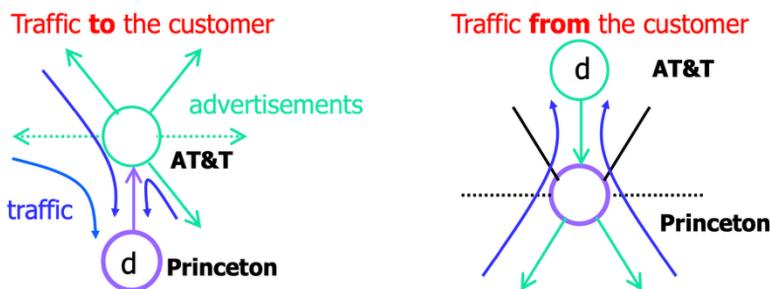




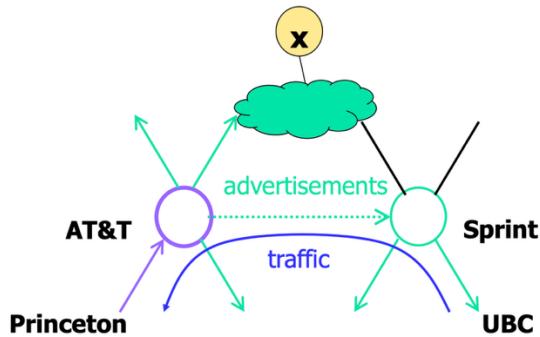
Gli AS scambiano informazioni di raggiungibilità sotto forma di prefissi IP (blocchi di indirizzi di destinazione) e AS path (sequenza di AS lungo il percorso); per quest'ultimo, le politiche sono configurate dagli operatori della rete, come la path selection (quale percorso utilizzare) e la path export (con quali vicini comunicare). Il tutto è gestito, come è stato già anticipato, dal protocollo BGP.



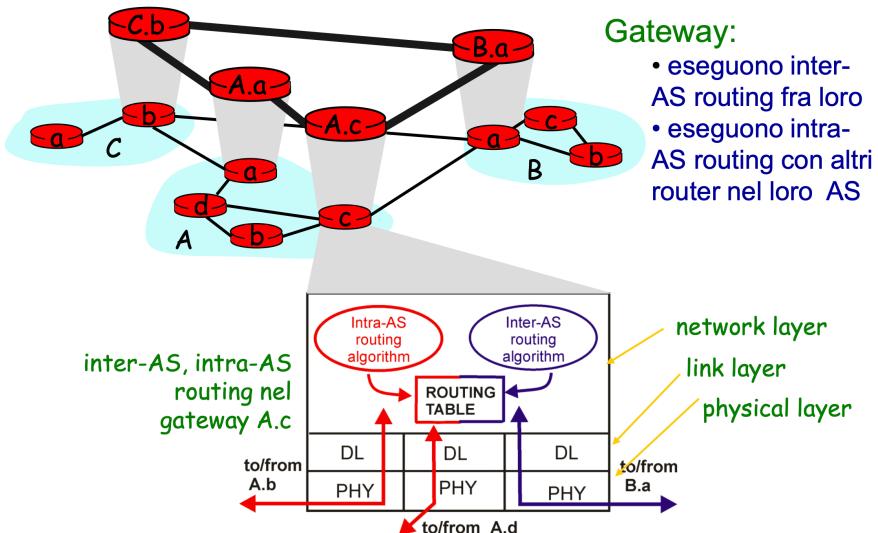
I clienti di un provider pagano per l'accesso ad internet; ovvero, affinché i provider stessi esportino le proprie rotte al di fuori della propria rete locale e affinché il cliente possa importarvi le rotte del provider. Il provider dovrebbe permettere anche ad altre parti della rete di raggiungere il cliente ma, dal momento in cui è il cliente a pagare per l'accesso ad Internet, il provider dovrebbe semplicemente garantire al cliente la maggior raggiungibilità possibile.



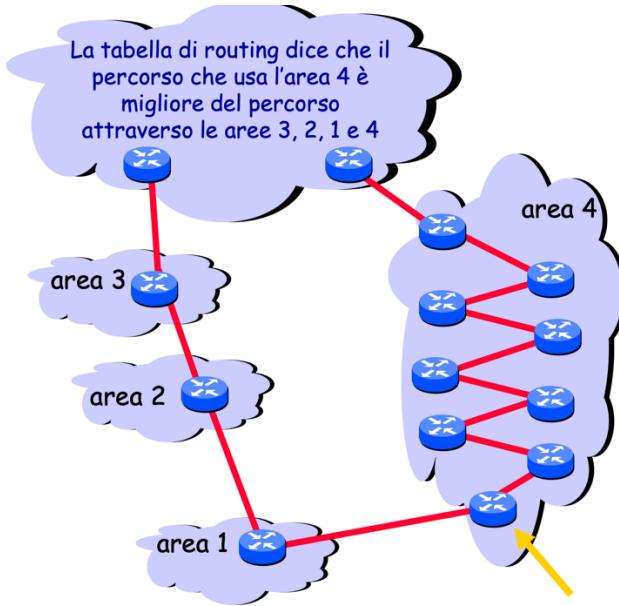
La relazione appena mostrata è detta Relazione Customer-Provider ma non è l'unico tipo di relazione che è possibile instaurare con un AS, esiste anche la relazione Peer-Peer, per la quale i Peer scambiano traffico tra clienti: gli AS esportano le rotte di un cliente solo ad un altro Peer e le rotte di un peer solo ai propri clienti. Ci si chiede, a questo punto, perché non si scambia tutta la tabella; il motivo di questa scelta risiede nel fatto che gli AS non vogliono provvedere a trasferimenti per conto dei propri peers, non essendo clienti paganti per quel lavoro.



I gateway router sono speciali router dell'AS che eseguono protocolli di routing intra-AS con altri router appartenenti all'AS e sono, inoltre, responsabili del routing verso destinazioni esterne al proprio AS, eseguendo un protocollo di routing inter-AS con altri gateway router. Su questi tipi di router sono pertanto attivi contemporaneamente sia protocolli di routing IGP (ad esempio, OSPF) sia protocolli di routing EGP (ad esempio, BGP). Si può facilmente intuire come i router di gateway siano gli unici a poter disporre, all'interno di un AS, di quest'ultimo tipo di protocolli.

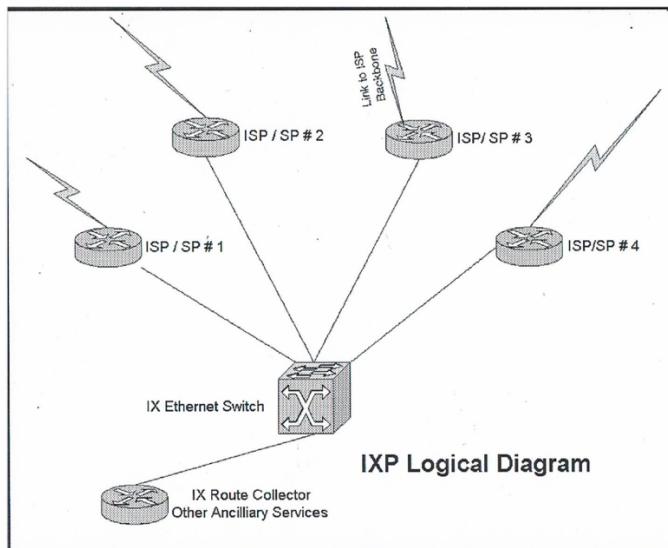


Come è facilmente intuibile a partire da quanto detto in precedenza, il routing gerarchico migliora notevolmente la scalabilità dell'Internet, garantendo, ad oggi, circa 150 milioni di destinazioni; non sarebbe possibile, impiegando il vecchio sistema a singola backbone, memorizzare in una tabella di routing tutte le destinazioni e, eventualmente, lo scambio di una quantità di informazioni di tale mole diminuirebbe notevolmente la banda utilizzata, limitando la quantità di dati scambiati. Il problema dell'approccio gerarchico risiede in un'unica falla: la possibilità di effettuare scelte sub-ottime; ad esempio:



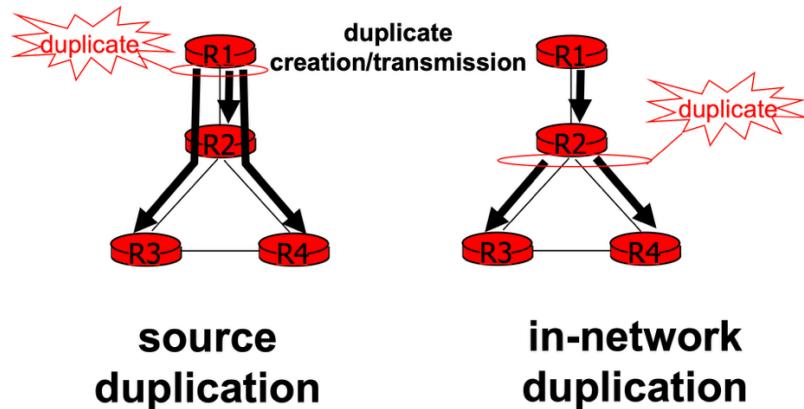
Nell'esempio in immagine, chiaramente il percorso ottimale sarebbe quello che attraversa gli AS 3, 2 e 1 (in ordine), piuttosto che quello che attraversa l'area 4, realmente configurato nella tabella di routing.

Per **Internet Exchange Points (IXP)**, o **Network Access Points (NAP)**, si intendono infrastrutture fisiche che permettono a diversi Internet Service Providers (ISP) di scambiare traffico Internet fra loro, interconnettendo i propri AS attraverso accordi di peering (generalmente gratuiti) con lo scopo di permettere agli ISP di risparmiare una parte della banda che comprano dai loro upstream provider e di guadagnare in efficienza e in affidabilità. Gli IXP operano, solitamente, realizzando connessioni L2 tra router BGP degli ISP.



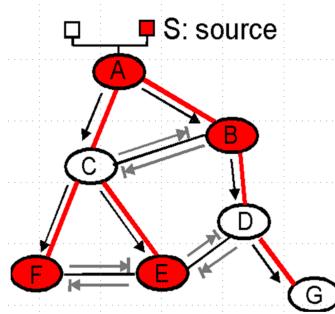
BROADCAST E MULTICAST

Il broadcast routing è quella pratica per cui i pacchetti sono consegnati dalla sorgente a tutti gli altri nodi e per la quale la source duplication è inefficiente:



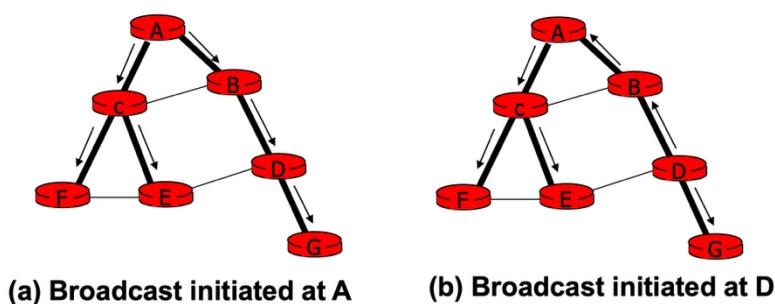
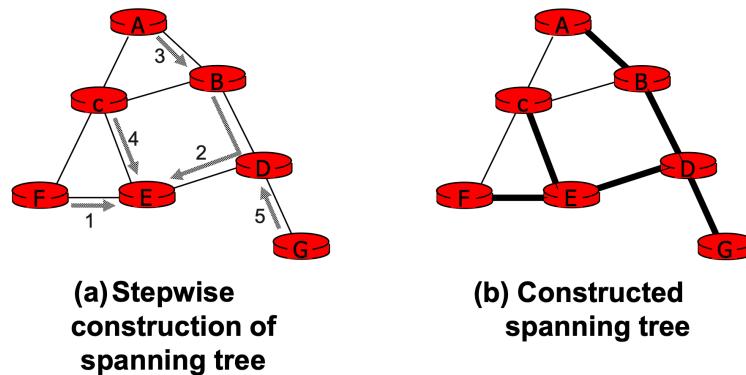
Il problema principale della source duplication risiede nell'inaccuratezza della valutazione, da parte della sorgente, del numero di nodi a cui inoltrare il pacchetto broadcast. La pratica dell'in-network duplication avviene tramite:

- **Flooding**, quando un nodo riceve un pacchetto broadcast ne manda una copia a tutti i vicini (con il rischio associato di cicli e broadcast storm);
- **Controlled flooding**, un nodo manda un pacchetto in broadcast solo se non ha ricevuto precedentemente lo stesso pacchetto:
 - **Sequence number**, il nodo tiene traccia dei pacchetti già mandati in broadcast;
 - Reverse path forwarding (RPF), inoltra un pacchetto solo se è arrivato dal percorso più breve tra nodo e sorgente;

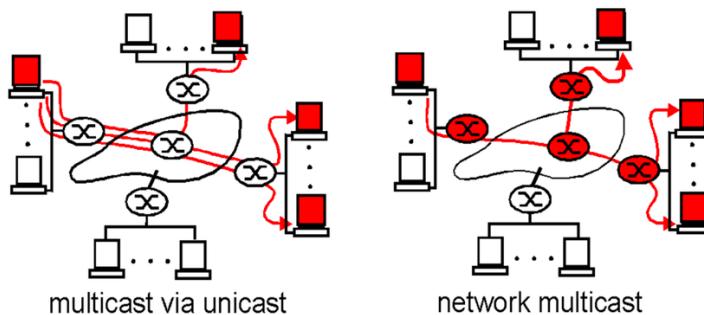


- **Spanning tree:**
 - Si costruisce un albero;
 - Si sceglie un Center Node (o Rendevouz Point);
 - Si fa in modo che ogni nodo invii una unicast join al CN (inoltrato finché il messaggio non arriva ad un nodo già parte dello spanning tree unicastly joinedo al CN);
 - Si fa in modo che un nodo inoltri un pacchetto solo lungo l'albero (un nodo non riceverà mai un pacchetto superfluo);

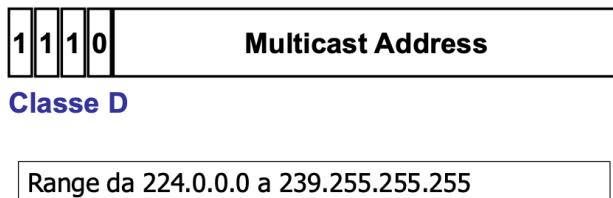
Si costruisce dapprima un albero, si sceglie un center node (o rendevouz point) e si fa in modo che un nodo inoltri un pacchetto solo lungo tale albero, in modo che un nodo **non riceva mai un pacchetto superfluo.**



Un numero sempre maggiore di applicazioni di rete richiedono la spedizione di pacchetti da uno o più sender ad un gruppo di receiver (come la trasmissione di un aggiornamento software di uno sviluppatore ai suoi utenti o la trasmissione live di una riunione aziendale o di una teleconferenza tra molti partecipanti distribuiti); per questo tipo di applicazioni, un'astrazione utile è la definizione di multicast: l'invio di un pacchetto da un sender a molti receiver con una singola operazione di spedizione.



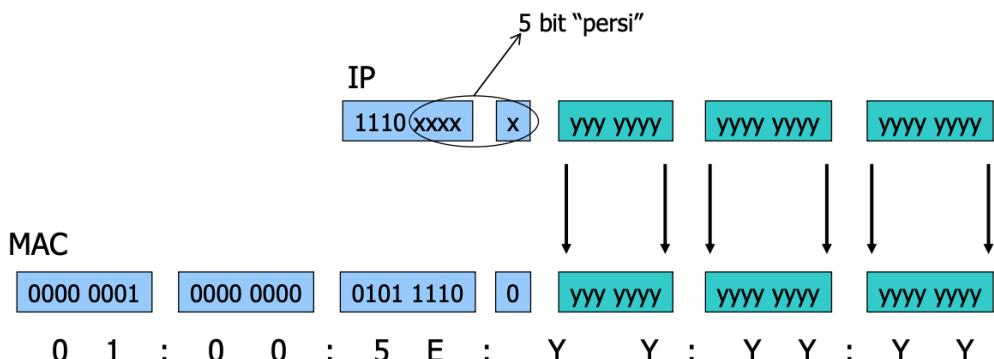
In una trasmissione multicast, sono due le domande da porsi per poter compiere l'operazione con successo: come identificare i ricevitori di un datagramma multicast? Come inviare un datagramma ai ricevitori, una volta identificati? La risposta ad entrambe le domande risiede nel cambio di paradigma da “indirizzo per destinazione” ad “indirizzo per evento” (Address Indirection): si utilizza un identificativo unico per il gruppo di ricevitori ed una copia del datagramma è inviata, utilizzando tale identificativo, a tutti i membri del gruppo. Ad ogni gruppo è associato un indirizzo multicast (indirizzo IP di classe D):



IANA ha assegnato, per usi riservati, indirizzi di classe D staticamente, da 224.0.0.0 a 224.0.0.255, mentre i seguenti sono assegnati dinamicamente:

- 224.0.1.0-238.255.255.255, **global scope**;
- 239.0.0.0-239.255.255.255, **limited scope**;
- 239.255.0.0/16, **site-local scope**;
- 239.192.0.0/16, **organization-local scope**.

Per la trasmissione di datagrammi IP multicast su reti LAN ethernet occorre mappare un indirizzo in classe D su di un indirizzo MAC multicast, il che risulta non possibile in maniera univoca, visto che il range degli indirizzi MAC multicast è da 01:00:5e:00:00:00 a 01:00:5e:7f:ff:ff; si ha quindi un indirizzo IP multicast con 28 bit liberi su uno MAC multicast ethernet con 23 bit liberi: il mapping non è 1:1, ci sono 32 gruppi multicast IP per ogni MAC multicast, inducendo la **possibilità di conflitti**. Per comprendere come avviene il mapping, si faccia riferimento allo schema di sotto:

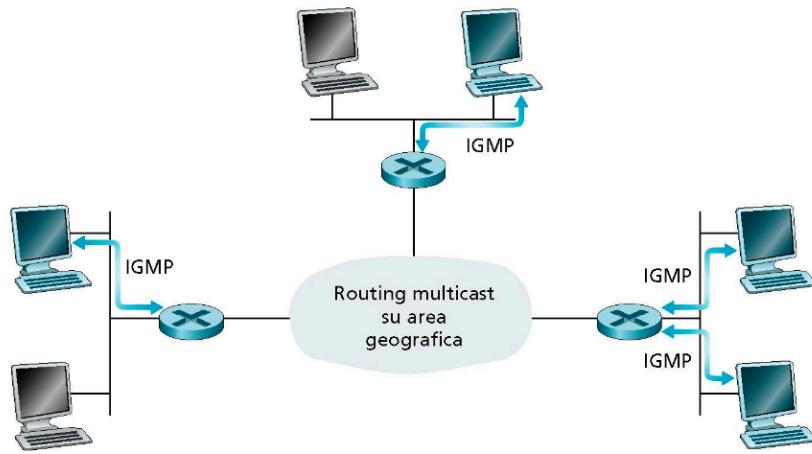


Per annunciare una sessione multicast e la sua descrizione si utilizza il protocollo SAP (Session Announcement Protocol), mentre per il trasferimento vero e proprio si impiega il protocollo UDP sul port 9875 e TTL 255; infine, per cancellare una sessione, si può ricorrere ad un Explicit Timeout (la durata è parte dell'annuncio), ad un Implicit Timeout (se non si riceve nulla per un intervallo prefissato) o ad una Explicit Delition.

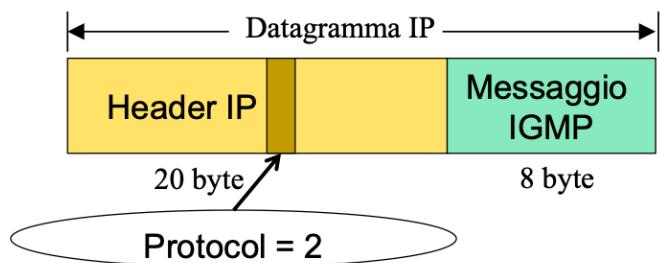
La gestione dei gruppi è di tipo dinamico: un host può unirsi o abbandonare un gruppo in qualsiasi momento e può appartenere contemporaneamente a più gruppi, non è necessario appartenere ad un gruppo per poter inviare ad esso dei datagrammi e i membri del gruppo possono appartenere alla medesima rete o a reti fisiche differenti.

Il multicast router è quel dispositivo che si occupa dello smistamento dei datagrammi multicast, in maniera trasparente riguardo agli host interessati. Ogni end system trasmette i datagrammi multicast sfruttando il meccanismo hardware messo a disposizione dalla rete locale su cui si trova e, se un datagramma giunge al multicast router, quest'ultimo si occupa, se necessario, di instradarlo verso le altre reti. IGMP (Internet Group Management Protocol) fornisce ad un host i mezzi per informare il multicast router ad esso più vicino che un'applicazione vuole unirsi ad un determinato gruppo multicast ed opera tra un host ed il router ad esso direttamente.

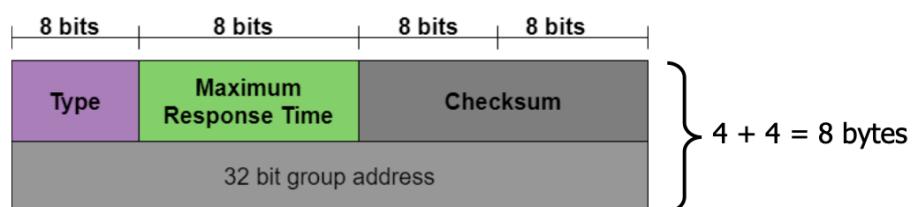
collegato, mentre **gli algoritmi per il multicast routing coordinano i multicast router al'interno della rete Internet**, per permettere l'instradamento dei datagrammi multicast.



IGMP serve a gestire in ambito locale (nel senso che i messaggi IGMP sono scambiati tra end system e router) **le informazioni di membership ai vari gruppi multicast e i suoi messaggi sono trasmessi in datagrammi IP il cui header ha il campo protocol con valore 2 e il campo TTL con valore 1.**



Quando un'applicazione chiede di entrare in un nuovo gruppo (operazione di join), l'host in cui gira l'applicazione invia un messaggio IGMP membership report: il messaggio è detto **unsolicited** perché non è generato come risposta ad un messaggio di membership query e gli eventuali multicast router presenti nella rete locale dell'host ricevono il messaggio e stabiliscono i meccanismi di routing atti a ricevere i pacchetti indirizzati al gruppo. Dovendo gestire i gruppi in maniera dinamica, i multicast router interrogano periodicamente (operazione di polling) gli host presenti nelle reti locali ad essi collegate, inviando messaggi di membership query a cui gli host rispondono inviando messaggi di membership report. Il formato di un messaggio IGMPv2 è il seguente:



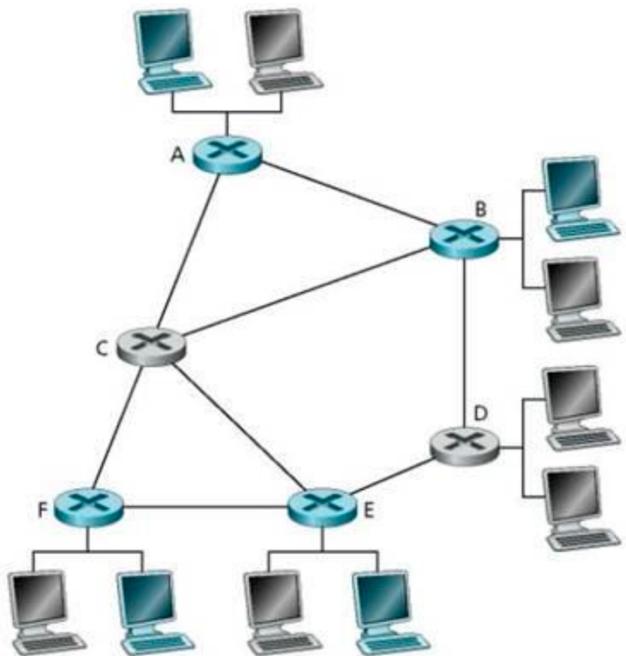
Mentre alcuni indirizzi IP destinazione dei messaggi IGMP sono:

IGMP message type	IGMP version	First octet value	Destination Address	
Membership Query (general)	ALL	0x11	224.0.0.1 ALL-SYSTEMS.MCAST.NET	Router asks all systems in the LAN to declare their memberships
Membership Query (group-specific)	v2, v3	0x11	group address G	Router queries for members of a group G in the LAN
Membership Report	v1	0x12	group address G	Join group G
Membership Report	v2	0x16	group address G	Join group G
Membership Report	v3	0x22	224.0.0.22 IGMP.MCAST.NET	Join group G
Source Specific Membership Report	v3	0x22	SSM group address	Join group G with source S
Leave Group	v1	N/A	N/A	N/A
Leave Group	v2	0x17	224.0.0.2 ALL-ROUTERS.MCAST.NET	Leave group G
Leave Group	v3	N/A	N/A	IGMPv3 uses membership reports instead of Leave msg

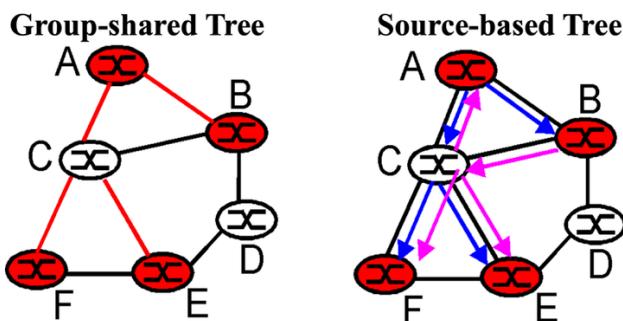
Per evitare il sovraccarico di una rete con un elevato numero di messaggi di membership report, alla ricezione di un messaggio membership query per un gruppo multicast al quale l'host ha fatto join, l'host stesso imposta un timer ad un valore D scelto casualmente e si mette in attesa: se prima dello scadere del timer, riceve un membership report inviato da un altro host, l'host proprietario del timer lo annulla e si astiene dall'inviare il proprio report; se il timer scade senza che siano arrivati altri report, l'host proprietario invia il proprio report. In questo modo, per ciascun gruppo attivo si genera un solo messaggio di report; tuttavia, questo meccanismo di Report Suppression esiste in IGMPv1 e IGMPv2 ma non in IGMPv3.

Il modello di Source-Specific Multicast (SSM, definito in RFC1112 e indicato come Any-Source Multicast ASM) prevede la possibilità per un host di manifestare l'intenzione a ricevere pacchetti inviati ad un gruppo G da uno specifico host sorgente S attraverso un'operazione di source-specific join realizzata mediante messaggi IGMPv3 di source-specific membership report. Gli indirizzi di gruppo IPv4 riservati all'uso di SSM sono 232/8 (ovvero da 232.0.0.0 a 232.255.255.255), mentre quelli IPv6 sono definiti dal prefisso FF3x::/32.

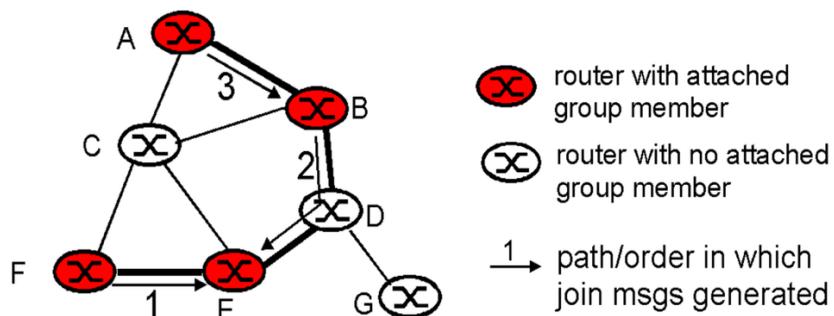
L'obiettivo di un instradamento multicast è di trovare un albero di link che colleghi tutti i router cui sono attaccati gli host che appartengono al gruppo multicast. I pacchetti multicast saranno, allora, instradati verso l'albero individuato dal sender a tutti gli host che vi appartengono; naturalmente, l'albero può contenere i router che non hanno collegati host appartenenti al gruppo multicast.



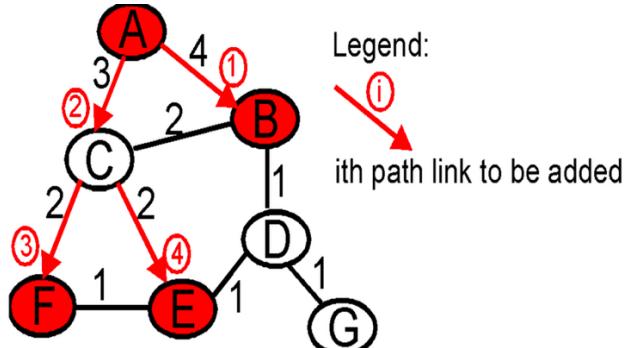
La rilevazione dell'albero di routing può avvenire seguendo due diversi approcci:



- **Group-shared tree**, per il quale è individuato un unico albero condiviso dal gruppo per distribuire il traffico di tutti i mittenti;
 - Questo approccio individua un Steiner Tree Problem, cioè al problema di ricerca dell'albero al costo minimo (problema NP-completo risolto grazie ad algoritmi che approssimano la soluzione ottima);
 - Può essere usato anche un approccio core-based, per il quale è individuato un nodo centrale come punto di rendezvous (o nucleo) e tutti i router collegati host multicast aderiscono al nucleo con messaggi unicast, mentre il percorso seguito definisce il ramo dell'albero;



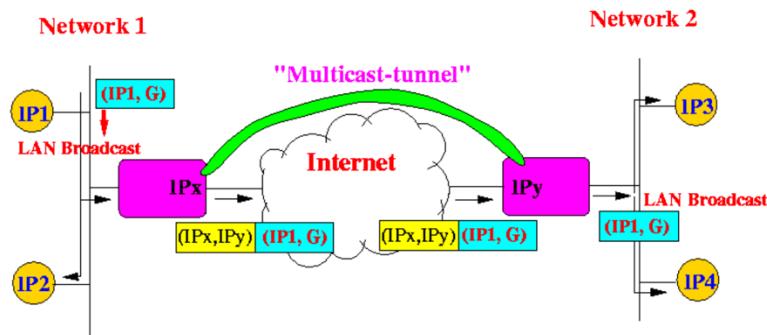
- **Source-based Tree**, per il quale è individuato un albero di instradamento per ciascun singolo mittente:
 - L’albero individuato da questo approccio può essere visto anche come un least unicast-cost path tree, cioè l’unione dei percorsi minimi dalla sorgente a tutte le destinazioni.



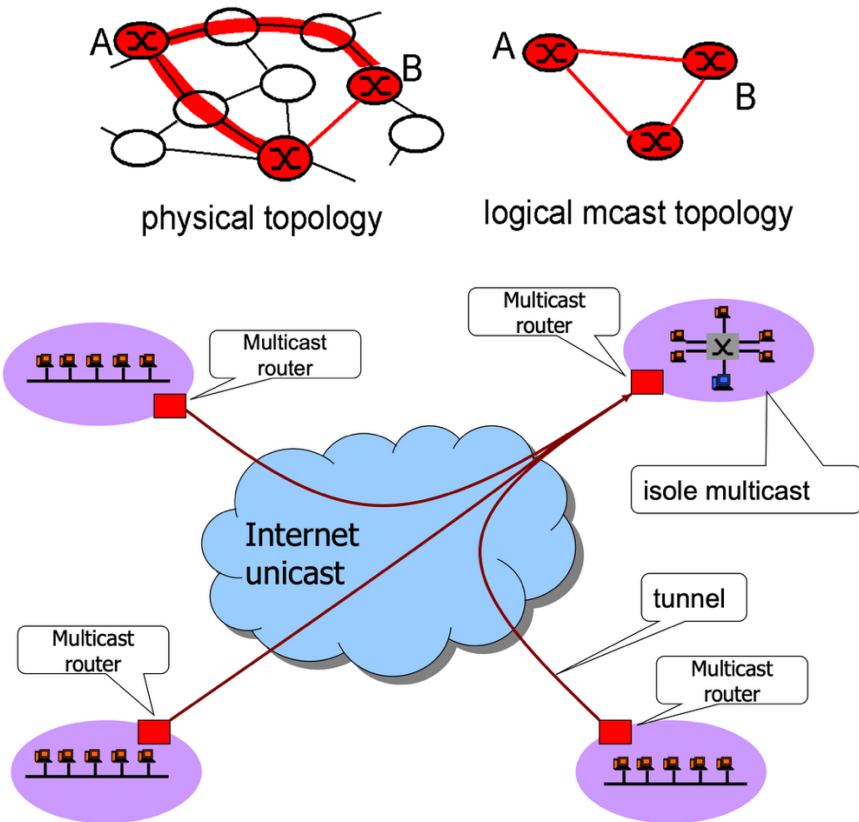
Analizzando le due soluzioni, si conclude che:

- Steiner Tree, è minimizzata la somma dei costi dei link dell’albero multicast;
- Least unicast-cost path tree, è minimizzato il costo dalla sorgente ad ognuna delle destinazioni.

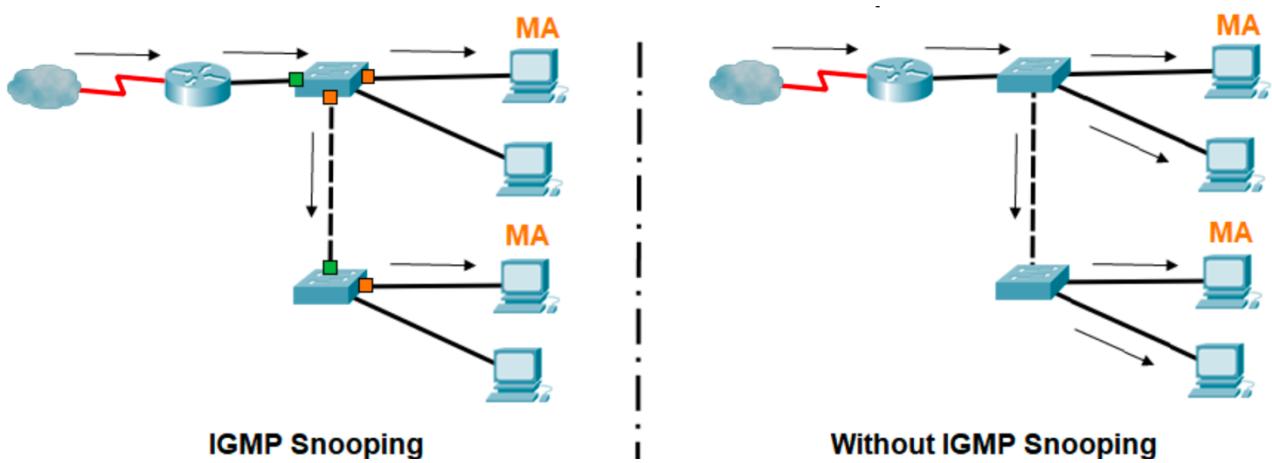
In passato era utilizzato MBone (Multicast BackBone) come banco di prova semi-permanente per il testing delle applicazioni e dei protocolli di routing multicast. Era una rete virtuale (overlay) composta da “isole” capaci di supportare il forwarding di pacchetti multicast IP collegate mediante link virtuali di tipo punto-punto, chiamati “tunnel”, e veniva usata per la trasmissione multicast di flussi multimediali descritti tramite il servizio “SDR” (Session Directory) basato sulla trasmissione multicast di annunci di sessione in formato SDP. I pacchetti IP multicast venivano incapsulati prima di essere trasmessi attraverso i tunnel, in modo da apparire, all’esame dei router e delle sottoreti intermediarie, come normali datagrammi multicast.



Un router multicast intenzionato a trasmettere un pacchetto all’altro capo di un tunnel deve aggiungere ad esso un ulteriore header IP in cui sia presente, come indirizzo destinazione, l’indirizzo unicast del router che si trova al capo opposto del tunnel (IP multicast in IP unicast). Il router situato all’estremo del tunnel deve, alla ricezione del pacchetto, eliminare l’header unicast che fungeva da capsula e smistare il pacchetto multicast nel modo appropriato.



Gli switch di rete (operando a L2) di default trattano i pacchetti multicast alla stregua di pacchetti broadcast e li inoltrano in flooding su tutte le porte, con sovraccarico conseguente della rete. La funzionalità di IGMP snooping consente agli switch di “imparare” (ascoltando i messaggi IGMP provenienti dagli host) quali porte devono essere interessate alla trasmissione multicast per ciascun gruppo. Tuttavia, questa funzionalità richiede che lo switch sia in grado di elaborare il payload della PDU di L2, dal momento che IGMP è un protocollo di L3.

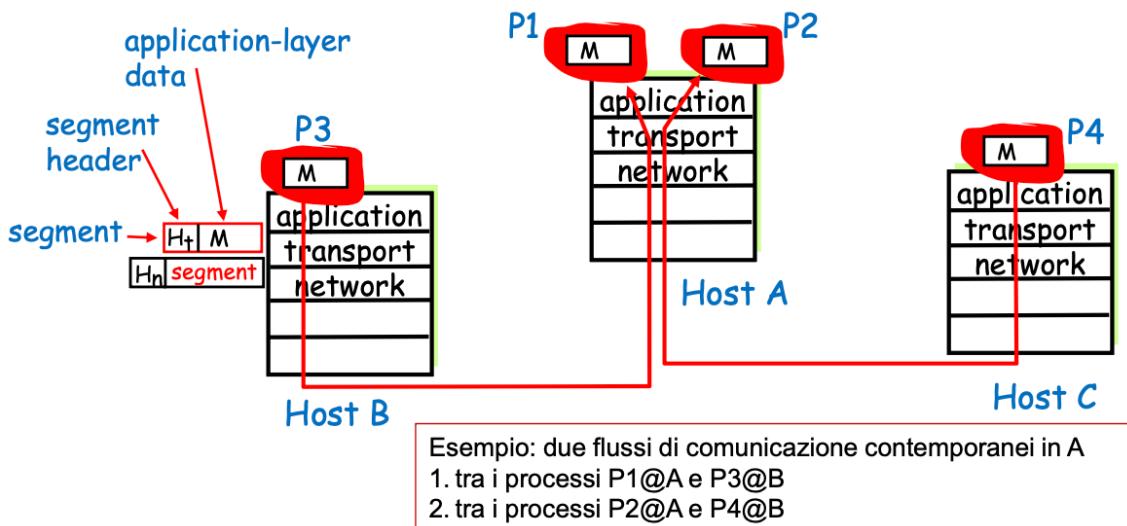


IL LIVELLO TRASPORTO

FUNZIONI DEL LIVELLO TRASPORTO

Il ruolo del livello trasporto è quello di consentire alle molteplici applicazioni eseguite in un end system lo scambio di informazioni attraverso il servizio offerto dal livello rete; pertanto, il livello trasporto agisce solo nei dispositivi terminali. In uno stesso momento, in un terminale possono essere attivi molteplici flussi di comunicazioni, ciascuno dei quali trattato direttamente dal livello trasporto.

Il compito del livello trasporto è quello di consegnare messaggi applicativi contenuti nei pacchetti che arrivano dal livello rete al processo (inteso nel senso dei Sistemi Operativi) al quale sono destinati. Questa funzione di smistamento (**demultiplexing**) è esplicata al livello trasporto tramite un'informazione ausiliaria contenuta nell'header dei pacchetti dei protocolli di livello trasporto, il **port number**.



Il livello di rete offre un servizio inaffidabile e senza garanzie di consegna; pertanto, il livello trasporto deve rimediare aumentando l'efficienza e l'affidabilità, effettuando il controllo di eventuali errori, ordinando la sequenza di pacchetti, controllandone il flusso e l'eventuale congestione. Questa pratica di miglioramento dell'affidabilità viene applicata per isolare i livelli superiori dai problemi dovuti all'uso di differenti tecnologie di rete e dalle loro (eventuali) imperfezioni. I protocolli di livello trasporto sono realizzati al di sopra di quelli del livello di rete, rendendo necessaria la gestione dell'apertura della connessione (fase di setup), della memorizzazione dei pacchetti all'interno della rete, del numero elevato di connessioni e molto altro; tutto ciò è realizzato mediante **multiplexing** e **demultiplexing**.

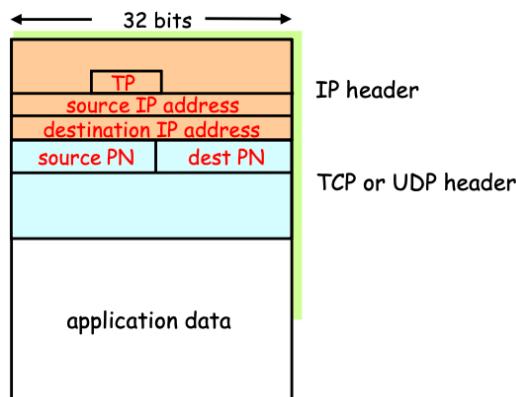
Definendo segment i dati che sono scambiati tra processi a livello trasporto, il **demultiplexing** si configura come la pratica dell'inoltro di segment ricevuti al corretto processo cui i dati sono destinati, mentre per **multiplexing** la pratica di raccogliere i dati provenienti dalle applicazioni e dell'imbusto dei dati con un header appropriato (per il demultiplexing).

Un pacchetto in arrivo ad un host è associato ad uno specifico flusso di informazione (segment) mediante la quintupla:

Transport_protocol, Source_IP_address, Source_port_number, Destination_IP_address, Destination_port_number

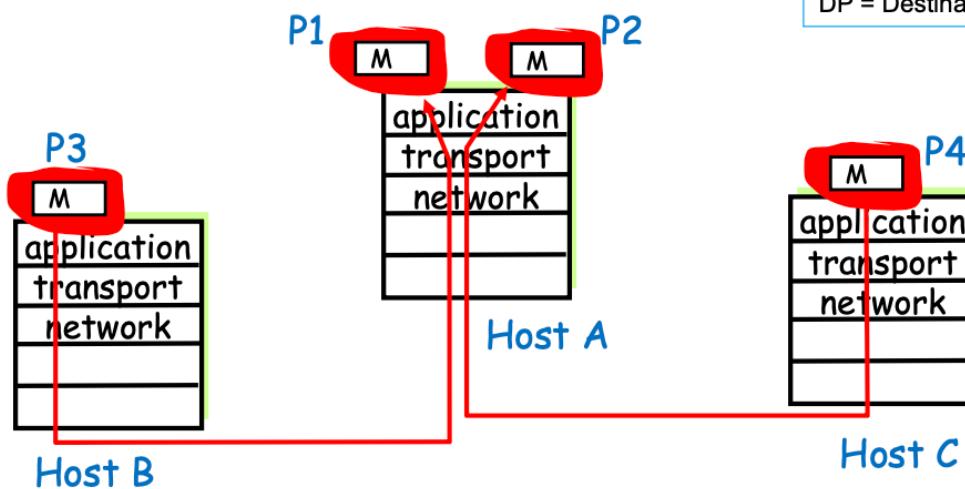
Dove:

- **Transport_protocol** (TP) è un numero naturale rappresentato su 8 bit che identifica il protocollo di livello trasporto ed è collocato nell'header di livello rete (IP), 6 per TCP e 17 per UDP (valori specificati in RFC 790);
- **Source_IP_address** e **Destination_IP_address** sono gli indirizzi IP (32 bit) del mittente e del destinatario del pacchetto;
- **Source_port_number** e **Destination_port_number** sono numeri naturali espressi su 16 bit che servono ad identificare il flusso di informazione e sono collocati nell'header TCP e UDP:
 - **Source_port_number** è scelto dall'host mittente;
 - **Destination_port_number** è scelto dall'host destinatario.



Esempio: due flussi di comunicazione contemporanei in A
 1. tra i processi P1@A e P3@B con TCP
 2. tra i processi P2@A e P4@B con TCP

Legenda:
 TP = Transport Protocol
 SA = Source IP Address
 DA = Destination IP Address
 SP = Source Port Number
 DP = Destination Port Number

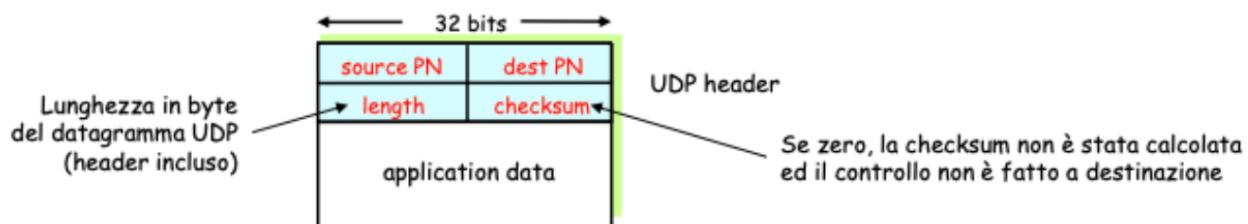


- Flusso P1@A → P3@B: TP=TCP; SA=A; DA=B; SP=80; DP=3127
- Flusso P3@B → P1@A: TP=TCP; SA=B; DA=A; SP=3127; DP=80
- Flusso P2@A → P4@C: TP=TCP; SA=A; DA=C; SP=80; DP=1984
- Flusso P4@C → P2@A: TP=TCP; SA=C; DA=A; SP=1984; DP=80

I due host B e C possono eventualmente scegliere lo stesso valore di Port Number, non si genera ambiguità perché i flussi entranti in A sono distinti in base al Source IP address.

- Flusso P3@B → P1@A: TP=TCP; SA=B; DA=A; SP=3127; DP=80
- Flusso P4@C → P2@A: TP=TCP; SA=C; DA=A; SP=3127; DP=80

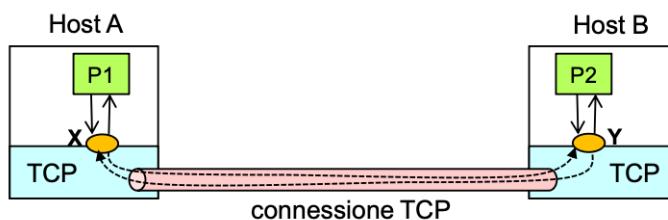
Il protocollo UDP offre alle applicazioni un servizio di consegna best effort ma senza garanzie di consegna o di rispetto dell'ordine di invio, forma un datagram IP per ciascun messaggio applicativo ricevuto dall'applicazione. In sostanza, aggiunge al servizio di IP solo il multiplexing attraverso il port number e, optionalmente, effettua un controllo di correttezza del datagramma ricevuto tramite una checksum (controllo con esito negativo corrisponde ad un datagramma UDP scartato). Le applicazioni che usano UDP devono, se lo ritengono opportuno, implementare meccanismi che adottino contromisure nel caso di perdita di datagrammi o di consegna fuori ordine. Infatti, questo tipo di applicazioni sono orientate a semplici interazioni richiesta-risposta.



TCP, invece, offre alle applicazioni un servizio di consegna senza perdite e con rispetto dell'ordine di uno stream di byte bidirezionale tra due endpoint; questo tipo di servizio è di tipo connection-oriented, cioè, prima di iniziare una trasmissione di dati stabilire una connessione tra i due endpoint attraverso una procedura di connection establishment, eliminata tramite una analoga procedura di connection teardown. Un pacchetto è associato ad una connessione TCP mediante la quintupla:

Transport_protocol, Source_IP_address, Source_port_number, Destination_IP_address, Destination_port_number

Negli end system, i processi si “agganciano” agli endpoint di una connessione TCP mediante opportuni strumenti di comunicazione previsti dal Sistema Operativo (socket).

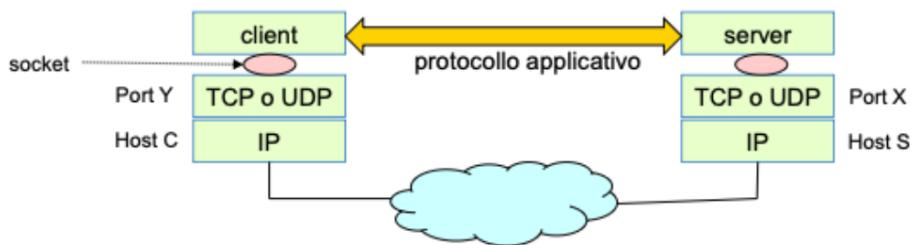


Il modello comportamentale a cui si ispira la maggioranza delle applicazioni commercialmente distribuite è il modello client-server:

- Un **processo server** è in esecuzione su un host S e si rende disponibile alla comunicazione, attraverso il protocollo TCP o UDP, ad un numero di porta X noto a priori ai client (well-known);
- Un **processo client** viene eseguito su un host C, acquisisce dal Sistema Operativo in esecuzione sull'host C la disponibilità d'uso del numero di porta Y e “chiede” al Sistema

Operativo di stabilire una comunicazione tramite TCP o UDP con il processo server, del quale conosce a priori l'indirizzo IP S ed il port number X.

L'interazione tra le componenti client e server delle applicazioni è governata da protocolli di livello applicativo, il quale definisce il formato dei messaggi ed i pattern di interazione.



La suite TCP/IP mette a disposizione delle applicazioni due servizi di livello trasporto: connectionless (senza garanzie, orientato ai messaggi e tramite UDP) e connection-oriented (con garanzie di consegna in ordine e senza perdite, tramite TCP). La scelta tra TCP o UDP dipende dai requisiti dell'applicazione:

- Per **applicazioni che si basano su semplici interazioni richiesta-risposta** è conveniente usare il **protocollo UDP**;
- Per **applicazioni che devono realizzare trasferimenti di dati significativi** tra due processi è conveniente usare il **protocollo TCP**;
- Per classi di **applicazioni con requisiti particolari** sono stati definiti **ulteriori protocolli di trasporto**:
 - Ad esempio, per le applicazioni che trasmettono flussi di dati time-sensitive è stato realizzato il protocollo di trasporto RTP, che è “implementato su UDP” (nel senso che l'header RTP è inserito nel payload di un datagramma UDP).

La definizione di protocolli applicativi standard consente di svicolare i servizi da una specifica implementazione; ad esempio, il servizio Word-Wide-Web è accessibile da una molteplicità di client (browser), ma lo stesso WWW può essere erogabile da una molteplicità di server (Apache Web Server, Microsoft IIS, NGINX, ecc...).

Uno sviluppatore che realizzi un'applicazione client/server è libero di definire ed implementare il proprio protocollo applicativo (purché sviluppi sia la componente client che quella server) **ma i servizi offerti tramite Internet si attengono a protocolli applicativi standard definiti dall'IETF in specifici documenti RFC**:

Applicazione	Protocollo applicativo	Protocollo di trasporto
world-wide web	HTTP	[RFC 2068,2616]
DNS	DNS	[RFC 1034,1035]
file transfer	FTP	[RFC 959]
e-mail (invio)	SMTP	[RFC 821]
E-mail (lettura)	POP3	[RFC 1939]
remote file server	NFS	[RFC 1813]
remote terminal access	TELNET	[RFC 854]
VOIP (segnalazione)	SIP	[RFC 3261]
VOIP (flussi multimediali)	-	RTP su UDP
network management	SNMP	[RFC 3416]

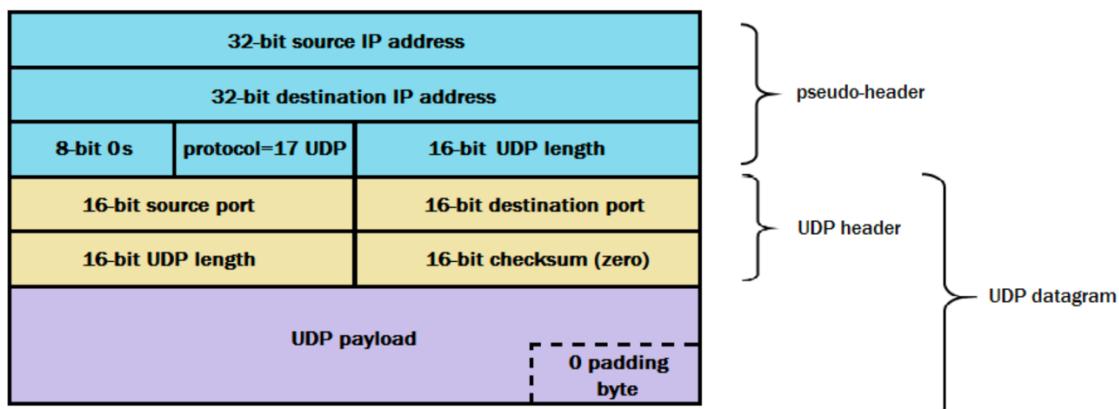
IL PROTOCOLLO UDP

Il protocollo UDP (User Datagram Protocol) **aggiunge poco ad IP**: offre un **servizio best effort** (i pacchetti UDP possono subire perdite, giungere a destinazione in ritardo, non arrivare affatto o giungere a destinazione non ordinati) **ma è un servizio connectionless** (non è prevista una fase di inizializzazione ed ogni segmento UDP è inviato indipendentemente dagli altri). **La vera forza di UDP risiede proprio nella sua essenza connectionless**: non è necessaria la fase di setup e sono evitati **delay inutili** (UDP è usato nei servizi DNS, migliorando la qualità del servizio all'utente finale); tuttavia, **hanno riscosso successo anche la sua semplicità** (sender e receiver non devono conservare informazioni di stato), **l'intestazione di dimensioni contenute** (basso overhead) e **il controllo della congestione assente** (le applicazioni possono inviare dati alla velocità da esse desiderata, anche se questa proprietà risulta utile per alcune applicazioni, è pur sempre rischiosa per la rete). **UDP è ampiamente utilizzato per applicazioni multimediali** (tolleranti alle perdite di pacchetti ma sensibili ai ritardi) **o in applicazioni come i servizi DNS, i NFS (Network File System)** **o i SNMP (Simple Network Management Protocol)**.

A questo punto, ci si chiede se sia possibile rendere UDP affidabile; la risposta risiede nella **possibilità di implementare un'adeguata gestione degli errori e la conferma di avvenuta ricezione**. Il primo milestone sta nella **rilevazione di errori** (bit alterati) nel segnale trasmesso:

- **Mittente:**
 - Tratta il contenuto del segmento come una sequenza di interi espressi su 16 bit;
 - Checksum, complemento ad 1 della somma (in complemento ad 1) dei numeri da 16 bit che costituiscono il segmento UDP (incluso uno pseudoheader che include gli indirizzi IP sorgente e destinazione);
 - Pone il valore della checksum nel campo checksum del segmento UDP;
- **Ricevente:**
 - Calcola la somma in complemento ad 1 dei campi del segmento ricevuto, compresa la checksum;
 - Risultato:
 - Tutti 1, nessun errore rilevato;
 - Alcuni 0, errore.

La condizione appena illustrata è necessaria ma non sufficiente a decretare la presenza di errori: se il risultato della checksum nel ricevente restituisce tutti 1, non è detto che non ci siano stati errori nella trasmissione. La checksum viene calcolata anteponendo al datagramma UDP uno **pseudo-header** che contiene gli indirizzi IP sorgente e destinazione e la lunghezza del datagramma e viene aggiunto ai soli fini del calcolo della checksum, non viene trasmesso.



In C, il calcolo della checksum può assumere la seguente forma:

```
unsigned short int udp_checksum(const void *buf, int len, in_addr_t src_addr, in_addr_t dest_addr)
{
    const unsigned short int *buf=buf;

    unsigned short int *ip_src=(void *)&src_addr, *ip_dst=(void *)&dest_addr;
    unsigned short int sum;
    size_t length=len;

    // Calculate the sum
    sum = 0;
    while (len > 1)
    {
        sum += *buf++;
        if (sum & 0x80000000)
            sum = (sum & 0xFFFF) + (sum >> 16);
        len -= 2;
    }

    // Add the padding if the packet lenght is odd
    if (len & 1)
        sum += *((unsigned char *)buf);

    // Add the pseudo-header
    ...
}
```

```
...
// Add the pseudo-header
sum += *(ip_src++);
sum += *ip_src;

sum += *(ip_dst++);
sum += *ip_dst;

sum += htons(IPPROTO_UDP);
sum += htons(length);

// Add the carries
while (sum >> 16)
    sum = (sum & 0xFFFF) + (sum >> 16);

// Return the one's complement of sum
return (unsigned short int)(~sum);
}
```

TRASMISSIONE DI FLUSSI MULTIMEDIALI IN INTERNET

Si pone un problema: **trasferire informazioni multimediali** (come audio e video) **da una sorgente ad uno o più ricevitori attraverso una rete**; per ridurre la quantità di informazioni trasferita sulla rete, **il trasmettitore opera una compressione mediante opportune tecniche** (MPEG 1-2-4, MJPEG, MP3, ...) **e permette il trasferimento in rete a pacchetti**. Il ricevitore, poi, **recupera l'informazione originaria dalla sequenza di pacchetti ricevuti**, mediante un'operazione inversa a quella di compressione, **e una successiva trasformazione in forma sonora o in forma video**.

Nel caso di informazioni in tempo reale (live), l'informazione è prodotta dalla sorgente mediante un apposito sistema di acquisizione (microfono + scheda audio, telecamera + video capture board), **opportunamente compressa** (in tempo reale) e **trasmessa sulla rete ai ricevitori**. Nel caso di informazioni preregistrate, l'informazione è già registrata in formato compresso (MPEG, MJPEG, MP3, ...) in un file memorizzato su memoria di massa; a questo punto, questo tipo di trasmissione si distingue in:

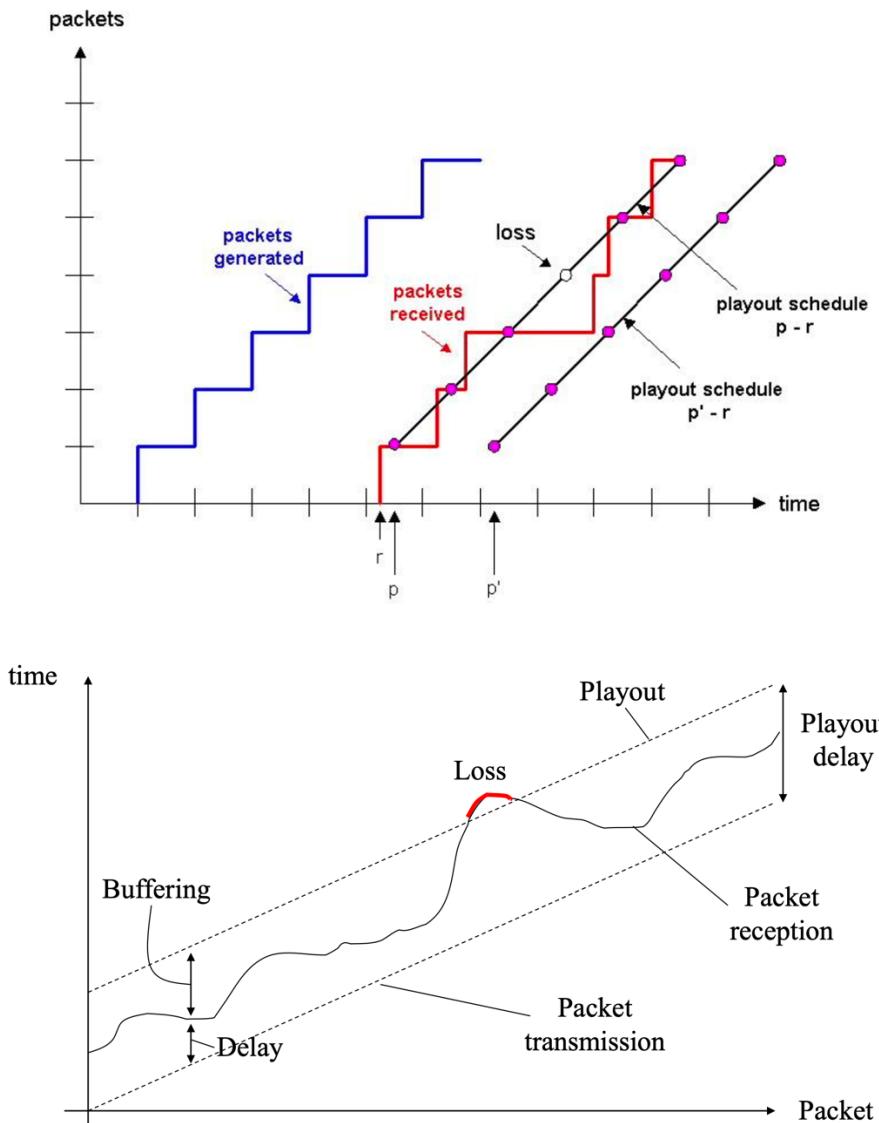
- **File transfer**, trasferimento dell'intero file da sorgente a ricevitore e successiva riproduzione, per la quale la riproduzione può iniziare solo al termine del trasferimento del file (con un ritardo proporzionale alla dimensione del file) e per cui è necessaria un'adeguata capacità di memorizzazione (su memoria di massa) da parte del ricevitore;
 - Soluzione idonea per documenti di piccole dimensioni;
- **Streaming**, riproduzione progressiva del contenuto multimediale durante il trasferimento dell'informazione, per la quale il ricevitore memorizza l'informazione ricevuta in un buffer (detto playout buffer), che viene continuamente alimentato dai dati ricevuti dalla rete e svuotato progressivamente e non dovendo memorizzare l'intero file, e per cui la riproduzione può iniziare non appena il buffer si è “sufficientemente” riempito;
 - Soluzione soggetta a sensibilità al jitter, la qualità della riproduzione può degradare se la rete non manitene la continuità temporale del flusso di informazioni trasmesso dalla sorgente.

Nel caso di informazioni live, la sorgente produce un flusso continuo di informazioni, eventualmente spezzato in pacchetti che sono trasmessi individualmente in rete: trasmissione in streaming. Il ricevitore riceve i pacchetti, recupera l'informazione originaria e la riconverte in forma audio e/o video. Il ricevitore riesce a recuperare la continuità del flusso di informazioni prodotto dalla sorgente se tutti i pacchetti arrivano a destinazione, con la stessa tempificazione relativa; la rete, però, può alterare la continuità temporale del flusso di informazioni in due modi:

- Facendo occasionalmente perdere dei pacchetti;
- Consegnando i pacchetti al ricevitore con una tempificazione diversa da quella con cui sono stati trasmessi (jitter).

Perché la rete possa effettivamente supportare la trasmissione di flussi multimediali occorre che alcuni parametri di QoS siano soddisfatti (percentuale di perdita di pacchetti, latenza, jitter, ...). Gli effetti di degradazione introdotta dalla rete sono diversi a seconda della natura del media, a seconda della tecnica di compressione utilizzata ed a seconda del grado di alterazione introdotto (nel caso di flussi audio, vengono percepiti dei disturbi, detti hiccups, mentre nel caso di flusso video, si hanno dei disturbi, detti glitches, che possono essere più o meno localizzati nel tempo e nello spazio). **Sia audio che video possono**, in generale, tollerare una parziale degradazione ma, quando si oltrepassano dei valori di soglia, l'informazione diventa inintelligibile.

Rispetto alla perdita occasionale di pacchetti, ci si difende mediamente con l'adozione di tecniche di compressione robuste, per le quali l'informazione audio/video ricostruita non è sensibilmente degradata quando occasionalmente si perde un pacchetto (in alcuni casi si adottano tecniche Forward Error Correction, FEC, o basate su ritrasmissione, idonee per lo streaming); **per limitare gli effetti del jitter**, invece, **si adotta una strategia di buffering**: un buffer in ricezione fa da volano e compensa (introducendo un ritardo extra) la variabilità del ritardo di attraversamento della rete. Questa pratica è detta Buffering con Ritardo di Riproduzione Costante:



Si noti che **non sarebbe necessario introdurre delle contromisure se la rete fosse in grado di offrire servizi a qualità garantita** (non servizi best-effort come quelli concretamente offerti).

RTP/RTCP e DASH

Il **trasferimento di informazioni multimediali su Internet** mediante la tecnica del **file transfer** è tipicamente realizzato adottando il **protocollo applicativo HTTP**, che si appoggia sul **protocollo di trasporto TCP**. Per la trasmissione in **streaming**, invece, sono adottati due approcci: o mediante un **protocollo ad-hoc (RTP)** su **UDP** o mediante **HTTP su TCP**.

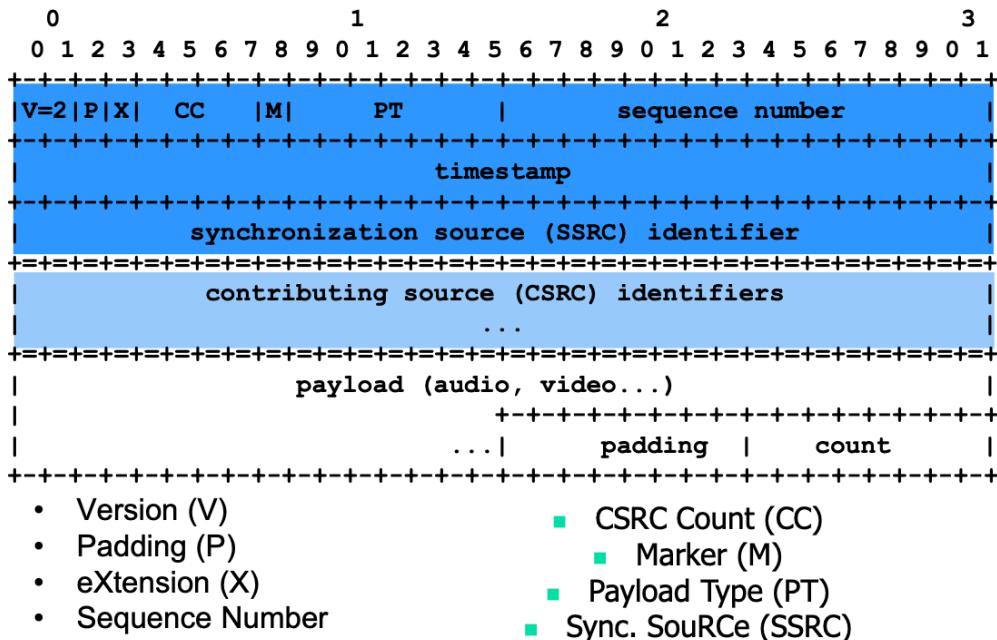
RTP sta per **Real-time Transport Protocol** (definito in RFC 1889 dal Working Group “Audio/Video Transport” dell’IETF) ed offre un servizio di livello trasporto specificamente progettato per i requisiti di flussi multimediali, con pacchetti incapsulati in datagrammi UDP (un protocollo di livello trasporto su un altro di livello trasporto). RTP è un protocollo concepito secondo il modello **Application Level Framing**, concepito per essere implementato direttamente nelle applicazioni, e non come uno strato aggiuntivo dello stack protocollare, e che offre funzionalità minimali richieste dalla trasmissione di flussi continui tipici delle applicazioni multimediali (infatti, è

neutrale rispetto alla codifica utilizzata). Inoltre, **RTP fornisce informazioni di tempificazione** (timestamp) per consentire la sincronizzazione intra-media (ricostruzione della corretta tempificazione della sequenza di pacchetti ricevuti) e inter-media (finalizzata a mantenere “al passo” flussi multimediali trasmessi separatamente, come audio e video per il lipsync), **supporta sia la trasmissione unicast che la trasmissione multicast**, i suoi meccanismi sono **scalabili rispetto al numero di appartenenti al gruppo multicast** e separa la trasmissione dei dati dalla trasmissione delle informazioni di controllo.

RTP è definito congiuntamente ad un protocollo di controllo, RTCP, utilizzato per scambiare informazioni di servizio e di controllo sulla qualità della trasmissione, garantendo la possibilità di combinare flussi di dati differenti mediante appositi mixer software.



Un pacchetto RTP è trasmesso in un datagramma UDP il cui header contiene i numeri di port sorgente e destinazione; RTP utilizza numeri di port destinazione pari ($2n$) per la trasmissione dei flussi di dati, mentre i numeri di port dispari adiacenti ($2n+1$) sono usati da RTCP per trasmettere le informazioni di controllo relative a quel flusso.



I campi di un header RTP sono:

- **Payload type**, 7 bit, specifica la codifica utilizzata per i dati (PCM, MPEG2 video, ...);
- **Sequence number**, 16 bit, serve ad identificare perdite di pacchetti;
- **Timestamp**, 32 bit, specifica il tempo di cambiamento del primo byte del payload e serve a rimuovere il jitter introdotto dalla rete mediante buffering;
- **Synchronization Source Identifier (SSRC)**, 32 bit, identifica la sorgente del flusso ed è scelto casualmente dalla sorgente stessa, introdotto per non dover fare affidamento sull’indirizzo IP per identificare la sorgente (sarebbero possibili conflitti);
- **Contributing Source Identifier List (CSRC)**, sequenza di n campi da 32 bit (con n positivo minore o uguale a 15), ciascuno dei quali identifica la sorgente originaria in un flusso prodotto

dalla “fusione” di flussi diversi mediante un mixer software (ad esempio, un’audio conferenza a più partecipanti: SSRC identifica il mixer e CSRC lo speaker corrente).

Per **sessione RTP** si intende **un’associazione tra un gruppo di entità che comunicano mediante RTP**; alcune applicazioni danno vita a sessioni RTP differenti per media differenti (ad esempio, audio e video) a meno che la tecnica di codifica adottata non effettui un multiplexing di flussi differenti in un singolo flusso di dati. **Sessioni RTP differenti** (ad esempio, audio e video) **vengono gestite da un ricevitore mediante il port number di livello trasporto (UDP)**.

Il valore di timestamp inserito in ogni pacchetto riferisce la tempificazione dei dati inseriti nel payload rispetto ad un clock specifico per il media trasportato; possono essere generati pacchetti RTP consecutivi con lo stesso timestamp. Il numero di sequenza identifica un pacchetto rispetto agli altri (non possono essere generati due pacchetti con lo stesso numero di sequenza), principalmente per consentire di identificare pacchetti persi.

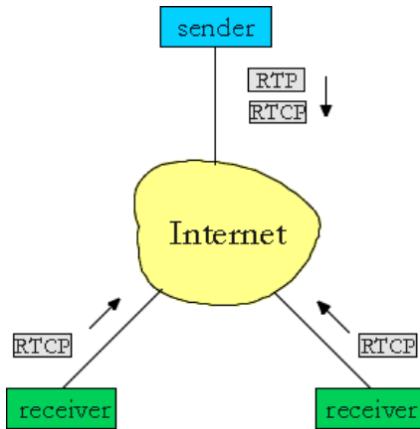
RTCP è un protocollo utilizzato congiuntamente ad RTP per la trasmissione di informazioni di controllo e i cui pacchetti vengono inviati con una certa periodicità, trasportando informazioni di varia natura: feedback sulla qualità della ricezione dei dati (percentuale pacchetti persi, ...), identificazione dei partecipanti ad una sessione RTP mediante un identificativo detto CNAME, ecc ... Nel caso di trasmissione RTP tra partecipanti ad un gruppo multicast, RTCP consente ad ogni partecipante di conoscere il numero di partecipanti.

Il protocollo RTCP definisce cinque tipi di messaggi:

- **Source Report (SR);**
- **Receiver Report (RR);**
- **Source Description (SD);**
- **BYE;**
- **APP.**

I messaggi di tipo report contengono statistiche sul numero di pacchetti inviati, numero di pacchetti ricevuti, percentuale di pacchetti persi, jitter dei tempi di interarrivo, ... e servono a monitorare la qualità della trasmissione. I messaggi di tipo description, invece, descrivono la sorgente del flusso (contengono, tra l’altro, il CNAME). BYE serve a notificare l’uscita da una sessione e APP è un tipo di messaggio le cui funzioni sono definibili dall’applicazione.

Nel caso di trasmissione multicast, ciascun ricevitore invia periodicamente (allo stesso gruppo multicast) i report RTCP. Per contenere il traffico di controllo, nel caso in cui il numero di membri del gruppo diventi molto elevato, si inserisce una minima forma di coordinamento: l’intervallo temporale tra due report è proporzionale al numero di partecipanti alla sessione, in modo che la banda consumata da RTCP non superi il 5% della banda usata dalla sessione.

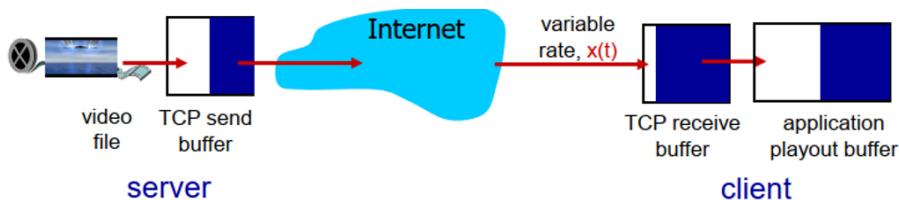


Nonostante l'esistenza di protocolli come RTP, i protocolli più usati per lo streaming sono congiuntamente HTTP e TCP. Questo perché, innanzitutto, HTTP è implementato in qualsiasi browser e, inoltre, i dati HTTP sono “trattati meglio” dai provider rispetto, ad esempio, a dati UTP, non sono coinvolti dai firewall associati perché ogni pacchetto HTTP è trattato come traffico ordinario. Per via del controllo di congestione che effettua TCP il tasso di trasferimento varia nel tempo, problema a cui si può ovviare adottando un ritardo di riproduzione maggiore.

L'uso di HTTP per lo streaming consiste, essenzialmente, nella richiesta da parte del client del media tramite HTTP GET, inviato tramite TCP. In aggiunta, secondo la tecnica detta DASH (Dynamic Adaptive Streaming over http), il media è suddiviso in molteplici pezzi dal server, ognuno dei quali sarà richiesto dal client con HTTP GET ed inviato individualmente e in continuazione dal server. Il client può individuare gli URL di ciascun pezzo da richiedere consultando un manifest file, generato dal server alla suddivisione, che contiene ciascun URL di ciascun pezzo; è analogo all'utilizzo in bitTorrent di un file Torrent per identificare i vari pezzi di un file. Ciascun pezzo ottenuto è, poi, mantenuto in una cache per un opportuno periodo di tempo.

Il client si occupa di misurare la qualità di trasmissione e può anche scegliere diversi tipi di codifica, quindi di risoluzione, mentre avviene la trasmissione; a risoluzione maggiore aumenta il numero di pezzi ricevuti.

Il vantaggio di DASH risiede nel fatto che tutta l’“intelligenza” (tutta la valutazione sulla qualità della trasmissione e su come ottenere ciascun pezzo) grava sul client e non sulla rete. Per via di DASH, l'uso di RTP è stato ridimensionato. Un suo utilizzo rimasto è il VOIP, la telefonia su IP; per il trasferimento di media, infatti, la VOIP usa l'RTP, mentre per la segnalazione fa uso del protocollo applicativo SIP (Session Initiation Protocol).

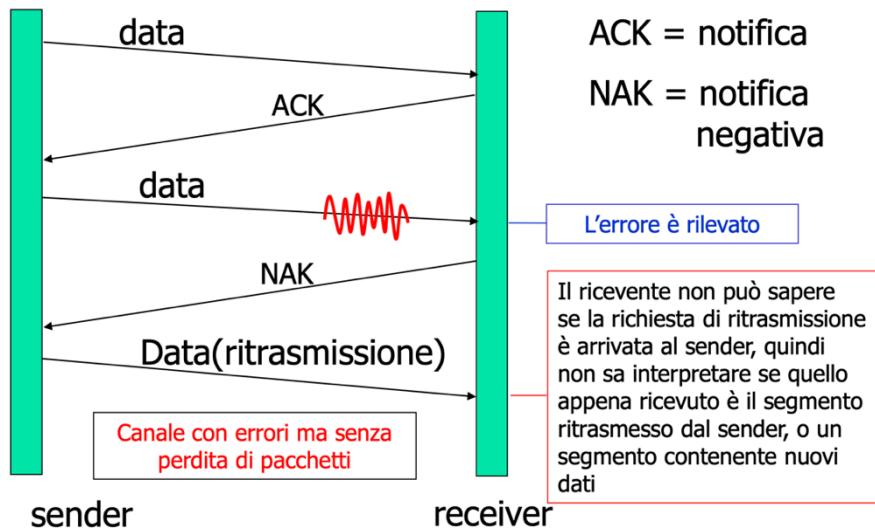


TECNICHE DI TRASMISSIONE AFFIDABILE DEI DATI

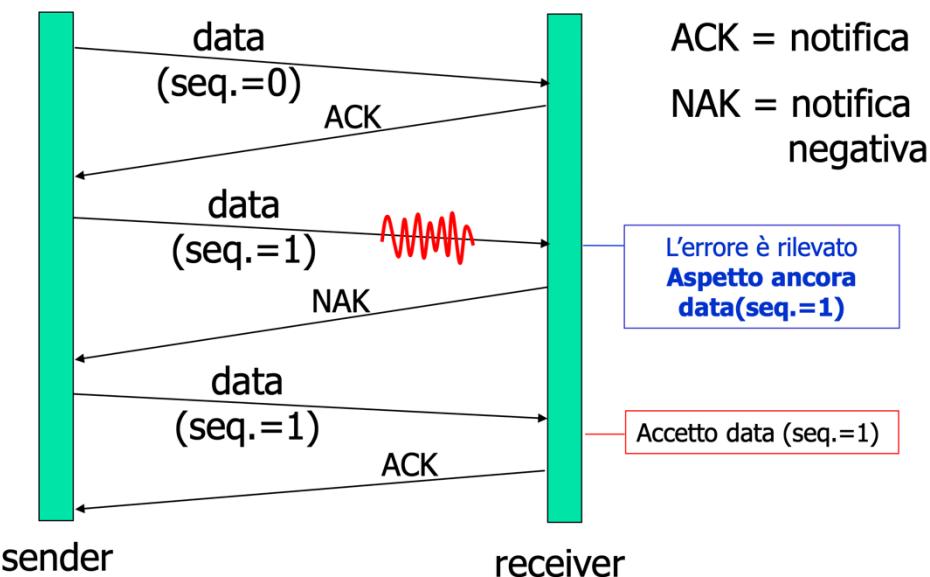
Se il livello rete è inaffidabile, nella comunicazione end-to-end si potrà verificare la presenza di errori, la perdita di pacchetti, l'assenza di garanzie sull'ordinamento dei pacchetti o la

duplicazione di pacchetto; il livello trasporto può farsi carico di rimediare a queste circostanze a favore delle applicazioni e può permettere la realizzazione di meccanismi che tengano in considerazione i buffer del computer ricevente a capacità limitata (controllo di flusso) e i buffer dei router a capacità limitata (controllo di congestione).

In caso di rete che presenta errori di trasmissione, il ricevente deve effettuare un rilevamento degli errori e, conseguentemente, una correzione o una notifica al mittente con richiesta di ritrasmissione. La prima soluzione introduce delle complicazioni, mentre la seconda possibili duplicazioni sulla rete che il ricevente non è in grado di interpretare ed una latenza potenzialmente maggiore.

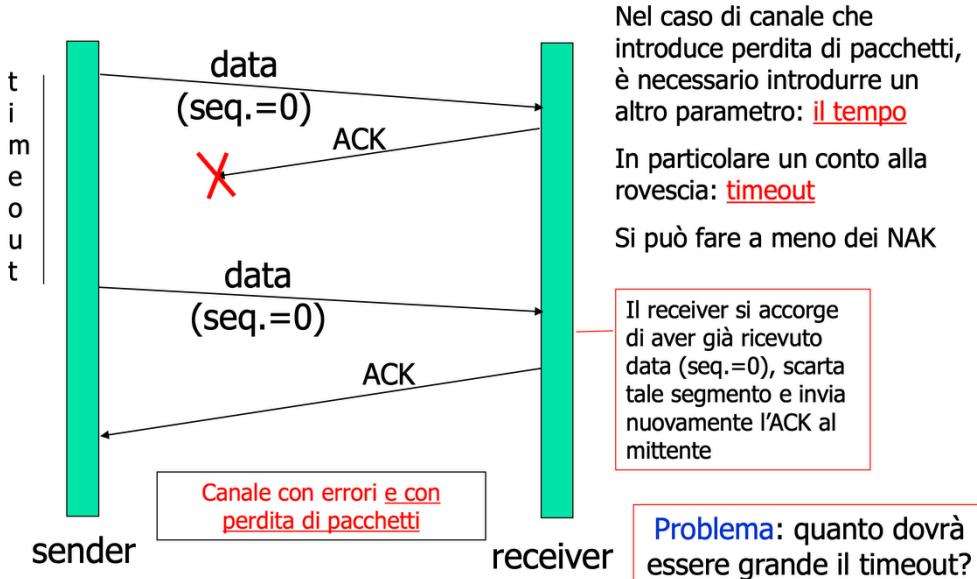


Per risolvere il problema di duplicazioni che il receiver non è in grado di interpretare, occorre inserire nell'header del segmento da inviare un'ulteriore informazione: il numero di sequenza. Nel caso di protocolli che inviano un messaggio, e che quindi aspettano un riscontro prima di ritrasmettere un nuovo messaggio (stop & wait), è sufficiente un numero di sequenza su un bit; ad esempio:

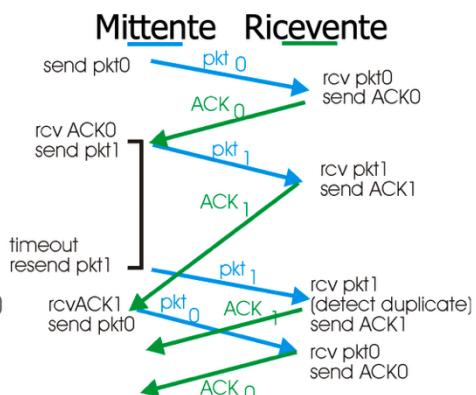
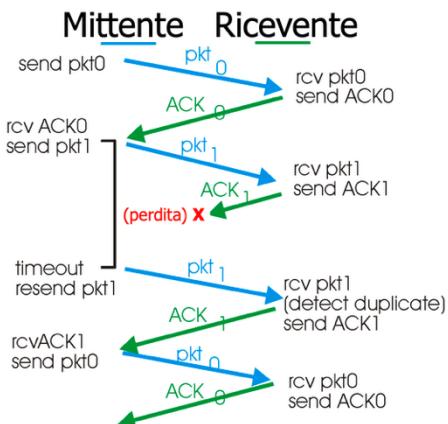
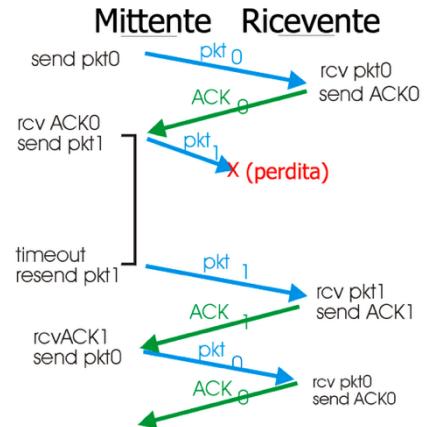
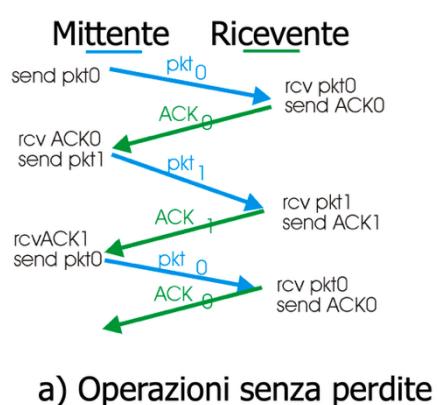


Pur mantenendo la funzionalità del sistema, è possibile evitare l'uso di NAK ma, invece, inviare un ACK per l'ultimo pacchetto ricevuto correttamente (il destinatario, però, deve includere

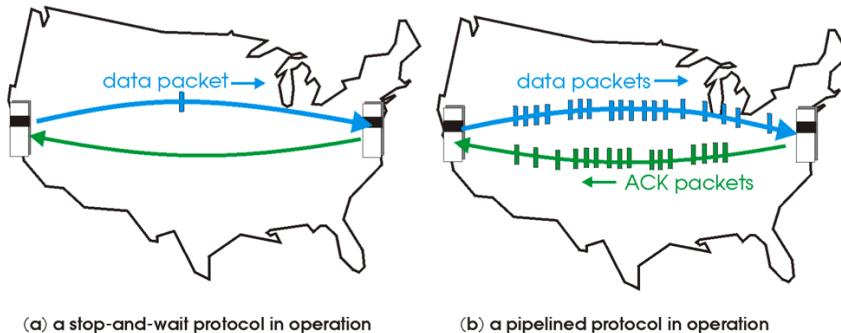
esplicitamente il numero di sequenza del pacchetto con l'ACK). Un ACK duplicato presso il mittente determina la stessa azione del NAK: ritrasmettere il pacchetto corrente.



Ricapitolando il sistema di stop & wait (detto anche “protocollo ad alternanza di bit”):



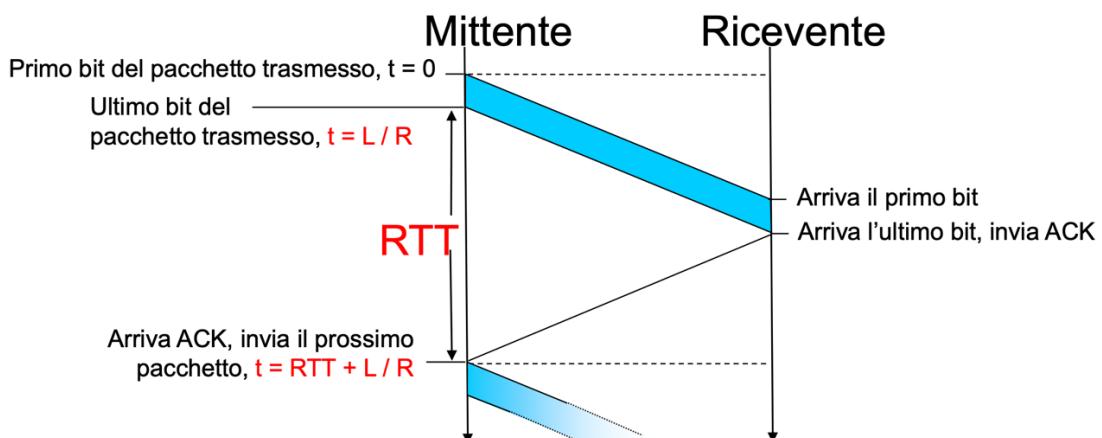
In alternativa a questo modello, è possibile implementare il **pipelining**, per il quale il mittente invia pacchetti prima di ricevere il riscontro dei precedenti, sebbene occorra aumentare l'intervallo dei numeri di sequenza e aggiungere buffer nel sender e/o nel receiver. Due alternative per il pipelining sono il go-Back-N e il selective repeat.



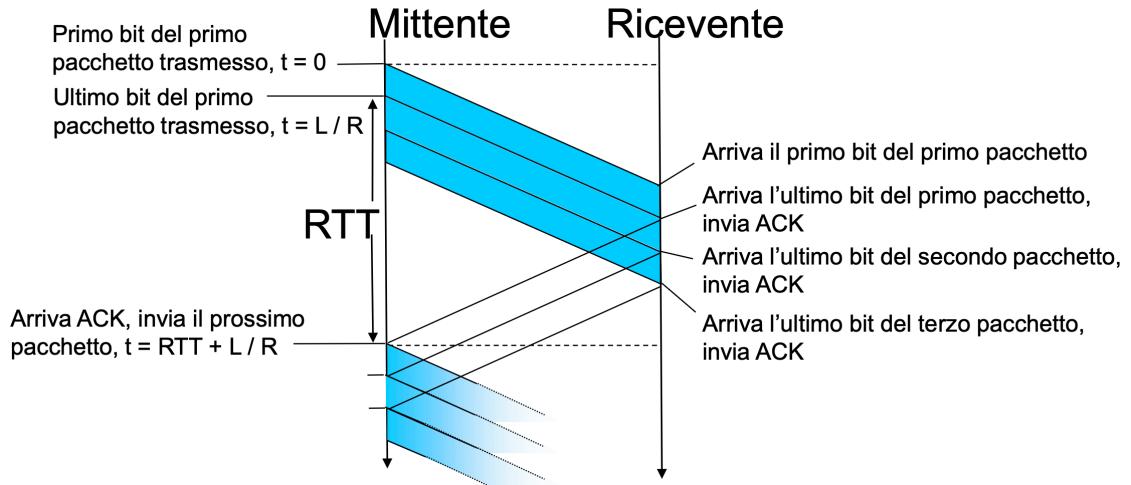
Il sistema stop & wait funziona ma le performances non sono ottimali: ad 1Gbps link, 15ms e-e delay di propagazione e 1KB di pacchetto:

$$T_{trans} = \frac{L \text{ (lunghezza pacchetto in bit)}}{R \text{ (data rate in bps)}} = \frac{8b/pkt}{10^9 b/s} = 8\mu s/pkt$$

Che è un tempo non trascurabile. Confrontando l'utilizzo del mittente per stop & wait e per il pipelining:



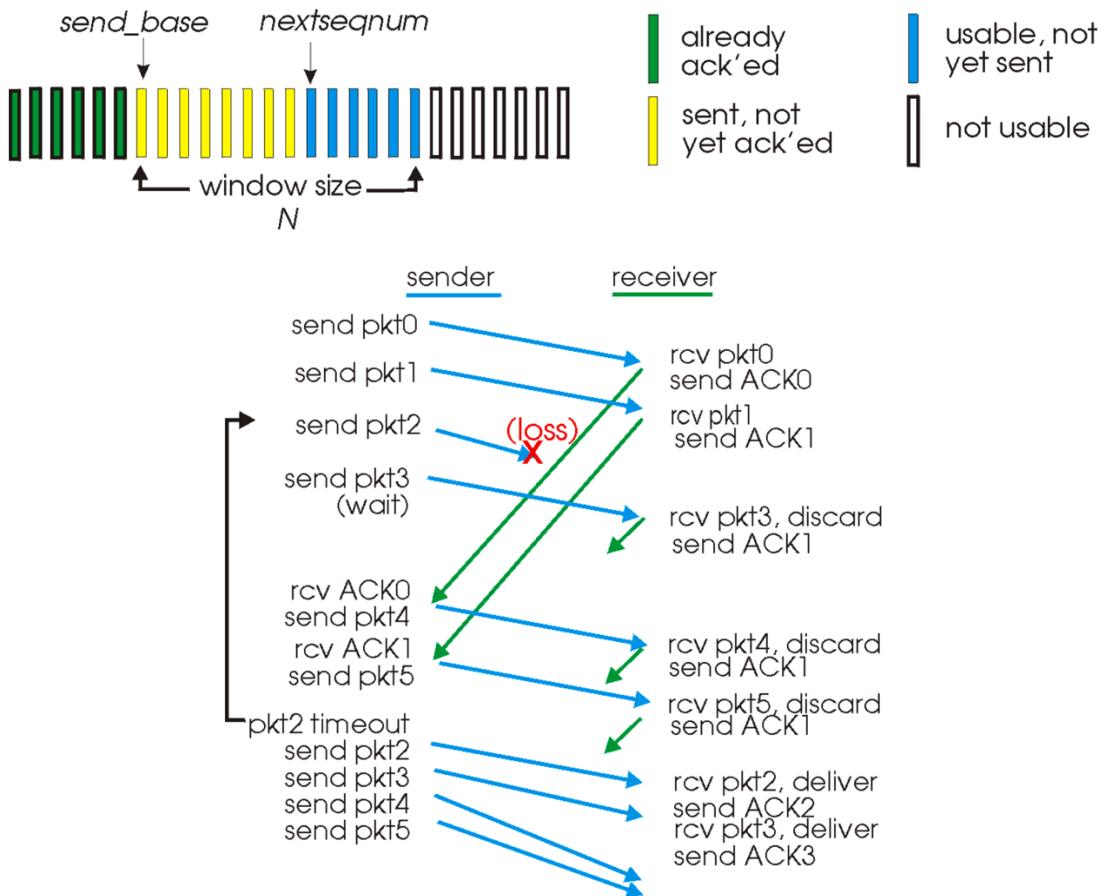
$$U_{mitt} = \frac{L/R}{RTT + L/R} = \frac{0.008}{30.008} = 0.00027 = 0.027\%$$



$$U_{mitt} = \frac{3 \cdot L/R}{RTT + L/R} = \frac{0.024}{30.008} = 0.0008 = 0.08\%$$

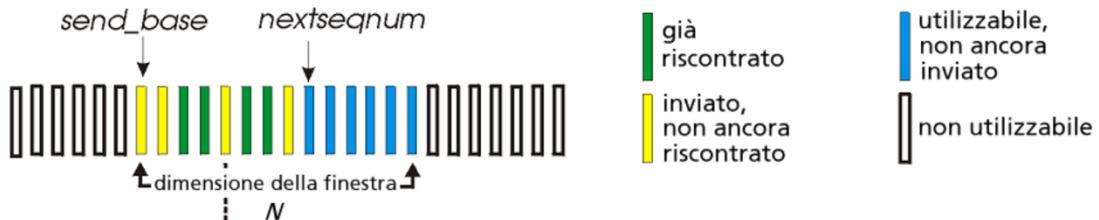
Con un aumento di un fattore di 3.

GO BACK N

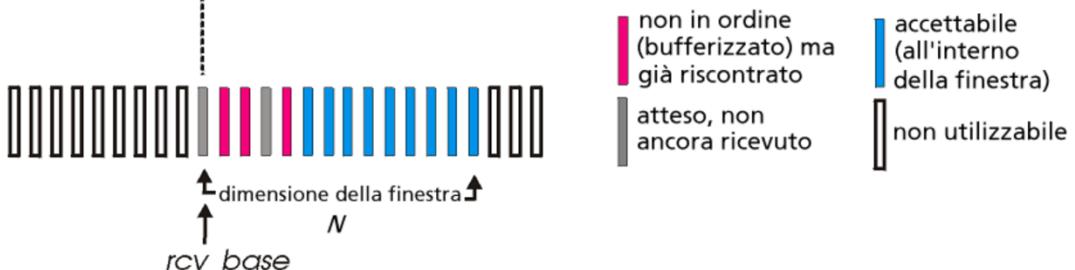


Secondo il modello di ripetizione selettiva, il ricevente invia riscontri specifici per tutti i pacchetti ricevuti correttamente, con buffer dei pacchetti (se necessario) per eventuali consegne in sequenza al livello superiore. In questo modo, il mittente ritrasmette soltanto i pacchetti per i quali non ha ricevuto un ACK (timer del mittente per ogni pacchetto non riscontrato) con una

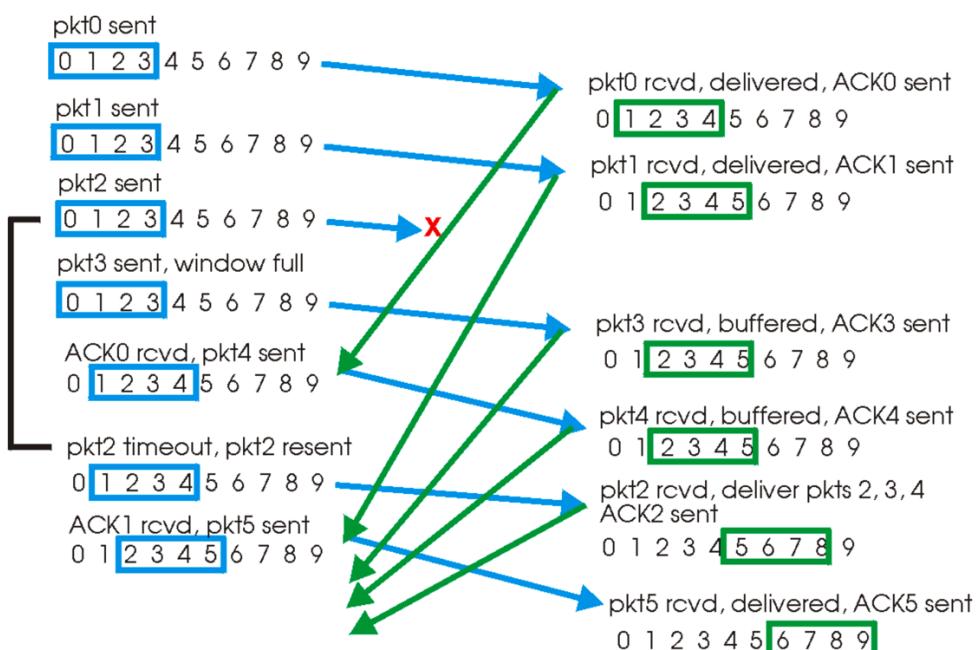
finestra di N numeri di sequenza consecutivi, pur limitando ancora i numeri di sequenza dei pacchetti inviati non riscontrati.



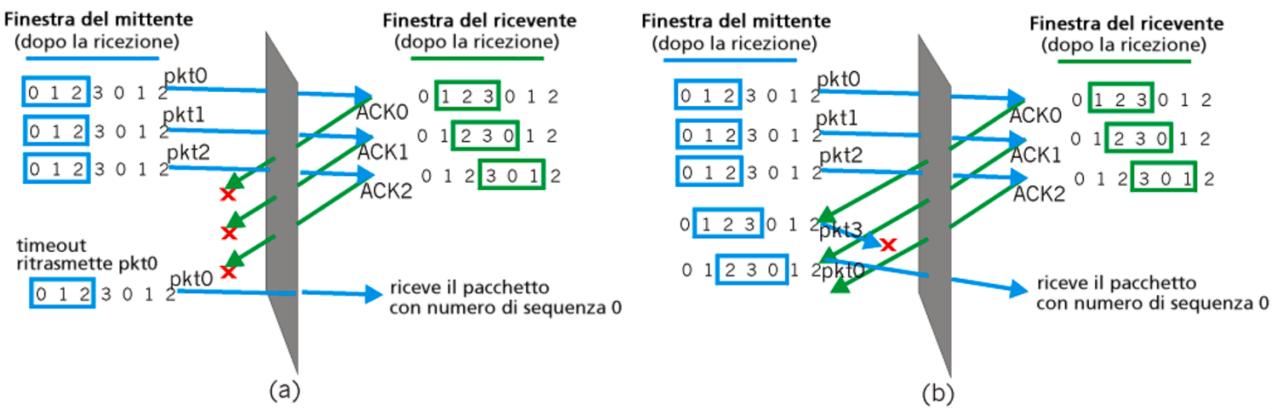
a) Visione del mittente sui numeri di sequenza



b) Visione del ricevente sui numeri di sequenza



Nello scenario seguente, con numeri di sequenza 0, 1, 2 e 3 e dimensione della finestra pari a 3, il ricevente non vede alcuna differenza fra i due scenari, passando erroneamente i dati duplicati come nuovi in (a):

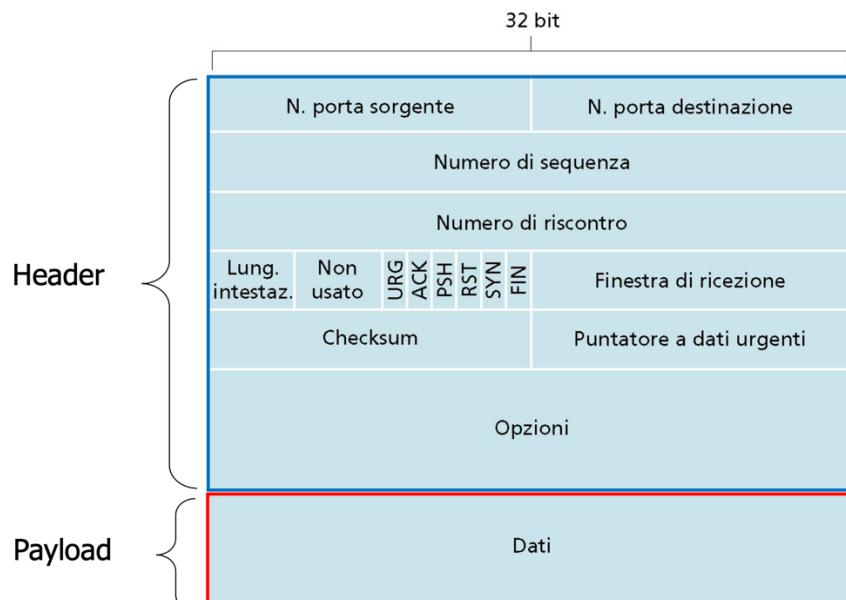


IL PROTOCOLLO TCP

TCP è un protocollo di trasporto connection-oriented, definito in RFC 793, che consente la trasmissione bidirezionale affidabile di un flusso di byte tra due endpoint, garantisce l'assenza di perdite e il rispetto nella sequenza di dati e realizza la propria affidabilità mediante controllo di sequenza e meccanismi di ritrasmissione (i dati sono mantenuti in appositi buffer ad entrambe le estremità dei flussi di dati). Le applicazioni interagiscono con TCP attraverso API che si avvalgono di punti di accesso al servizio detti socket.



La struttura di un **segmento TCP** è la seguente:



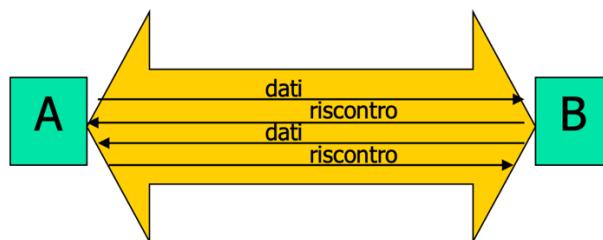
I numeri di porta sorgente e destinazione contengono, come nei protocolli analoghi, i numeri di port di protocollo TCP che identificano gli endpoint della connessione (per le operazioni di mux

e demux), mentre la lunghezza header (HLEN) è un numero di 4bit che contiene un numero intero che indica la lunghezza dell'intestazione TCP del datagramma in word da 32 bit, è un'informazione necessaria perché il campo opzioni può avere dimensione variabile (HLEN può andare da 5 a 15, codificando, rispettivamente, l'assenza di opzioni o un header massimo di 60 byte). Il numero di sequenza identifica, nello stream di byte del trasmettitore, la posizione dei dati nel segmento ed è riferito allo stream che fluisce nella medesima direzione del segmento, mentre il numero di riscontro si riferisce allo stream che fluisce nella direzione opposta. Quest'ultimo contiene il numero sequenziale di byte successivo a quello correttamente ricevuto dalla destinazione ed è valido solo nei segmenti di riscontro (cioè quelli che hanno ACK pari a 1); nel calcolo del numero di riscontro, oltre a considerare i bytes contenuti nel payload TCP, bisogna considerare anche la presenza di bytes SYN e FIN inviati, che valgono come un singolo byte. Il campo Flag è usato per identificare il tipo di informazione contenuta nel segmento e impiega i seguenti 6 bit di codice:

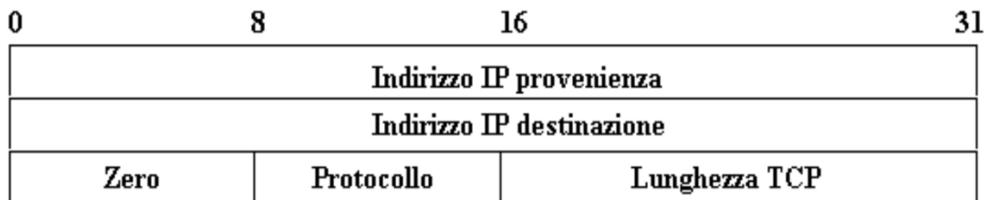
- **ACK**, il campo riscontro è valido;
- **RST**, effettua il reset della connessione;
- **SYN**, sincronizza i numeri di sequenza;
- **FIN**, il trasmettitore ha raggiunto la fine del suo stream di byte;
- **PSH**, questo segmento richiede una “spinta” a destinazione;
- **URG**, il campo puntatore urgente è valido.

Per quanto riguarda la finestra di ricezione, si tratta di un intero senza segno di 16 bit che specifica la dimensione del buffer che il TCP ha a disposizione per immagazzinare dati in arrivo, mentre il puntatore ai dati urgenti è il campo in cui è memorizzato (in caso di URG pari a 1) il puntatore alla posizione nello stream dei dati NON urgenti (ovvero, l'ultimo byte dei dati urgenti) perché TCP permette la trasmissione di dati informativi ad alta priorità che devono essere trasmessi il prima possibile. Checksum è un campo di 16 bit contenente un valore intero utilizzato dal ricevitore per verificare l'integrità del segmento ricevuto, visto che IP non prevede nessun controllo di errore sulla parte dati del frame, e marca il segmento per essere scartato in presenza di errori.

Durante lo stream di dati si alternano due fasi: una fase di scambio dati ed una di riscontro, per verificare che il segmento sia stato inviato correttamente, senza errori, e per verificare che non sia un duplicato. In fase di riscontro, l'informazione viaggia in normali segmenti TCP identificati dal valore 1 del flag ACK; per ogni connessione TCP tra due endpoint A e B, esistono due flussi dati distinti, quello da A a B e quello da B ad A:

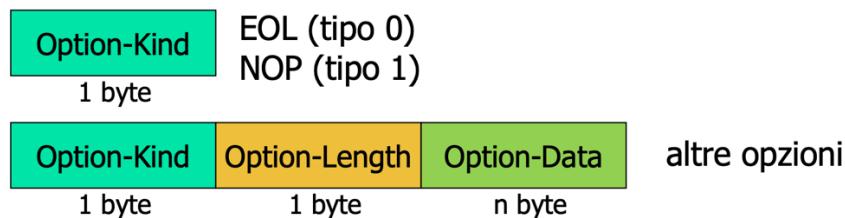


Un segmento inviato dall'host B all'host A può contenere o meno dati relativi al flusso da B ad A e può contenere o meno un'informazione di riscontro relativa al flusso dati da A a B. Se un segmento contiene sia dati che riscontro, si dice che il riscontro viaggia a cavalluccio (piggy-backing) dei dati. Ai soli fini del calcolo della checksum, TCP aggiunge fittiziamente al segmento effettivo uno pseudo-header costituito come segue:



In ricezione, TCP ricrea la pseudo-header interagendo con lo strato IP sottostante, calcola la **checksum** e verifica la correttezza del messaggio. Il mittente tratta il contenuto del segmento (e dello pseudo-header) come una sequenza di interi espressi su 16 bit e calcola la checksum come il complemento ad 1 della somma in complemento ad 1 dei “pezzi” da 16 bit che costituiscono il **segmento TCP e lo pseudo-header**, ponendo poi il valore di checksum nell'apposito campo del segmento trasmesso. Il ricevente, invece, **calcola la somma in complemento ad 1 dei campi del segmento ricevuto, compresa la checksum**, e, se il risultato non è composto da tutti 1, c'è un **errore**, altrimenti il controllo di checksum è superato.

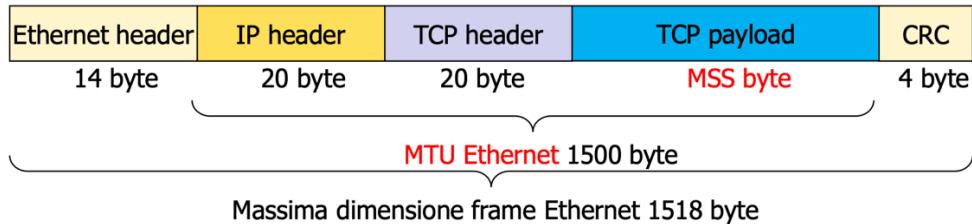
Le opzioni di un header TCP sono di lunghezza variabile, a patto che **la lunghezza dell'header totale sia un multiplo di 4 byte** (se necessario, **possono essere aggiunti zeri di padding** dopo l'ultima opzione). Sono previsti sette tipi diversi di opzioni: MSS, Window scale, Timestamp, Selective ACK permitted, Selective ACK, NOP e EOL, con un formato variabile tra due tipi:



La maggior parte delle opzioni è consentita solo nella fase di instaurazione della connessione (ovvero, per segmenti con flag SYN pari a 1). Andando ad approfondire le opzioni:

- **MSS**, durante la fase di configurazione ciascun endpoint annuncia (MSS announcement) la massima dimensione di payload (Maximum Segment Size) che desidera accettare;
- **Window Scale**, per negoziare un fattore di scala per la finestra, utile per connessioni a larga banda ed elevato ritardo di trasmissione (long-fat pipes);
- **Selective Repeat**, nel caso in cui un segmento corrotto sia stato seguito da segmenti corretti, introduce i NAK (Not ACK), per permettere al receiver di richiedere la ritrasmissione di quello specifico segmento;
- **Timestamp**, utilizzata per aiutare i due endpoint a determinare il RTT (tempo intercorso dalla trasmissione di un segmento alla ricezione del relativo ACK);
- **NOP**, No-Operation, separa opzioni diverse quando non allineate su multipli di lunghezza di quattro byte;
- **EOL**, End of Option List, indica la fine delle opzioni ed è necessario se la fine delle opzioni non coincide con la fine dell'header TCP.

Per MSS in TCP si intende la massima dimensione di payload consentita ed è determinata dall'MTU del link sul quale l'host trasmette; per un link Ethernet, MTU è 1500 byte e l'MSS è 1460 byte:



L'opzione MSS di TCP consente a ciascuno dei due endpoint di notificare all'altro la massima dimensione del segmento da utilizzare:

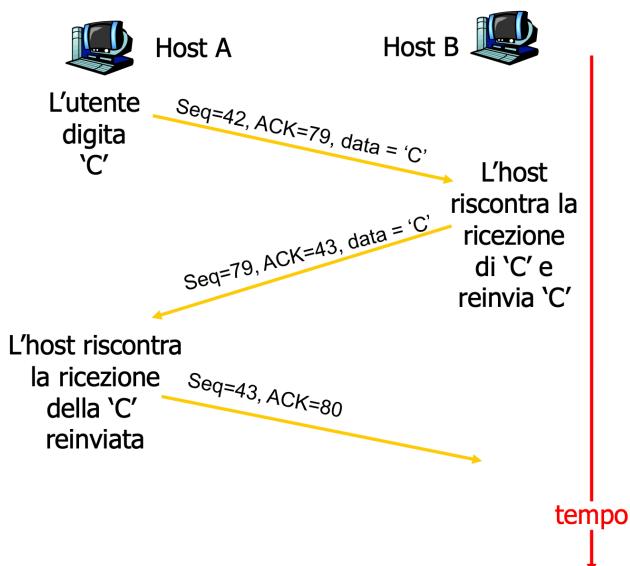


Se l'altro endpoint non ha trasmesso l'opzione MSS nel segmento SYN, RFC 1122 (Requirements for Internet Hosts) prescrive che MSS sia posto a 536 byte.

In TCP, **l'affidabilità della consegna dei dati è ottenuta tramite riscontro e ritrasmissione**, ovvero nella ritrasmissione di un segmento se non è giunta conferma entro un tempo massimo, e **Time-out**, un timer attivato al momento della trasmissione.

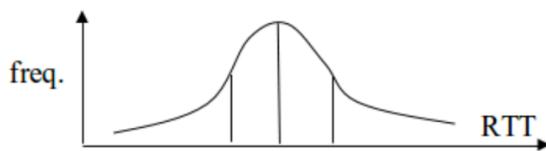
TCP vede i dati come un flusso di byte non strutturati ma ordinati, il campo “numero di sequenza” di un segmento TCP indica il numero di sequenza progressivo del primo byte nel segmento (flusso lungo 500.000 byte, MSS pari a 1000 byte, primo byte numerato con 0, TCP costruisce 500 segmenti, con numero di sequenza 0, 1000, 200, ...). **Per i numeri di riscontro**, vista la **natura full-duplex** della connessione TCP, **si ha che**, ad esempio, **A invia e contemporaneamente riceve da B; i segmenti B→A contengono un numero di sequenza relativo ai dati B→A e il numero di riscontro che A scrive nei propri segmenti è il numero di sequenza del byte successivo che A attende da B** (e lo può mandare anche inviando dati a B). TCP invia riscontri cumulativi.

Sapendo che **il numero di sequenza è il numero del primo byte del segmento nel flusso di byte e che ACK è il numero di sequenza del prossimo byte atteso dall'altro lato della comunicazione** (ACK cumulativo), **come fa il destinatario a gestire i segmenti fuori sequenza**, visto che la specifica TCP non lo dice (dipende da chi implementa il sistema)? Viene **utilizzata Telnet**, un'applicazione in cui il server invia l'eco dei caratteri digitati dal client.

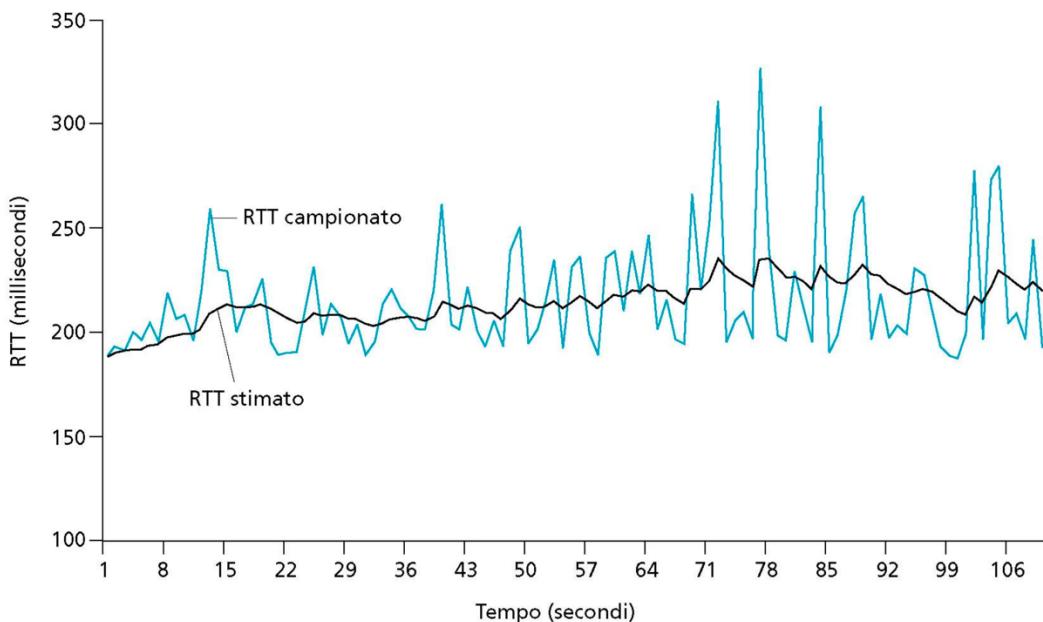


Nel datagramma di riscontro, la destinazione comunica quale byte dello stream si aspetta di ricevere successivamente, mentre i riscontri specificano il numero sequenziale del primo byte non ancora ricevuto; ad esempio, in uno stream di 1000 byte segmentato in blocchi di 100 byte, partendo da 0, il primo riscontro conterrà il numero sequenziale 100. Con questo metodo di riscontro cumulativo si ha il vantaggio che la perdita di un riscontro non blocca la trasmissione se confermato dal riscontro successivo.

A questo punto, ci si chiede a quale valore impostare il timeout; di sicuro il valore in questione deve essere maggiore del RTT. Tuttavia, se questo varia nel tempo e il timeout scelto è troppo breve, il timeout è prematuro, si hanno ritrasmissioni ridondanti e si abbassa l'efficienza, mentre se il timeout scelto è troppo lungo si causa scarsa efficienza nella gestione delle ritrasmissioni. Il problema, quindi, si riduce alla stima del RTT corrente e, a causa della sua variabilità, anche della sua varianza.



Una stima del RTT a confronto con il suo campionamento è simile ad una regressione della seconda curva:



La stima del RTT avviene con una media esponenziale pesata dei campioni (EWMA, Exponential Weighted Moving Average), per cui l'influenza di un singolo campione sul valore della stima decresce in maniera esponenziale, dando molta più importanza ad un campione recente:

$$RTT_{ext} = (1 - \alpha) \cdot RTT_{ext} + \alpha \cdot RTT_{sample}$$

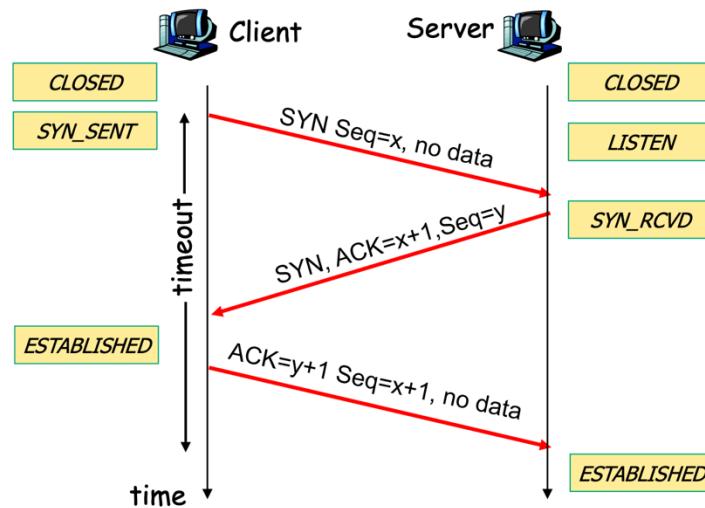
Con valori tipici di α pari a 0.125. Per quanto riguarda la deviazione di RTT:

$$RTT_\sigma = (1 - \beta) \cdot RTT_\sigma + \beta \cdot |RTT_{sample} - RTT_{ext}|$$

Con valori consigliati di β pari a 0.25. La stima in questione è un margine di sicurezza proporzionale alla variabilità della stima effettuata e una variabilità significativa di RTT_{ext} implica un margine più ampio; pertanto:

$$\text{Timeout} = RTT_{ext} + 4 \cdot RTT_\sigma$$

La procedura per il TCP connection establishment è detta **three-way-handshake** e inizia con il client che prende l'iniziativa inviando il primo segmento:

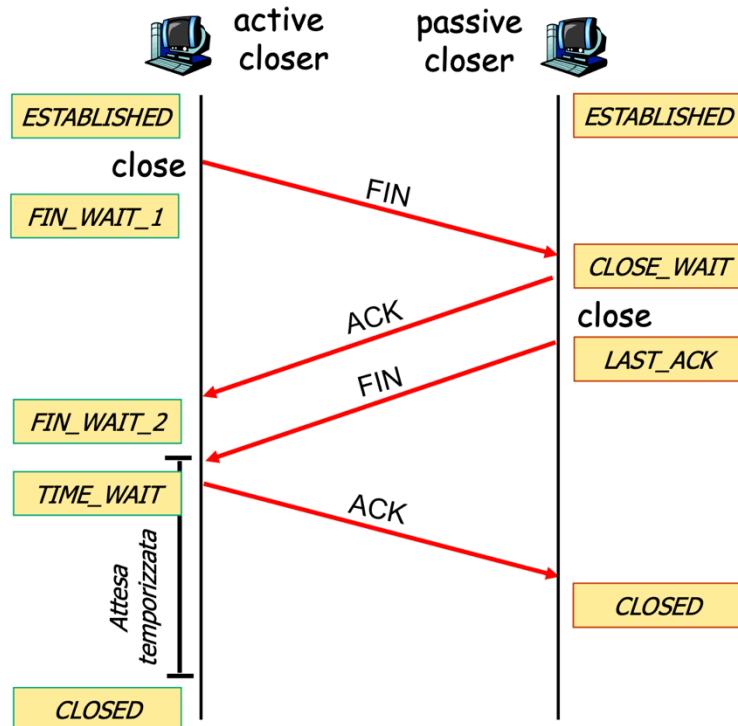


Nella fase principale, i due endpoint allocano i buffer per i dati e concordano sui valori iniziali dei numeri di sequenza da utilizzare per gli stream dati in entrambi i versi:

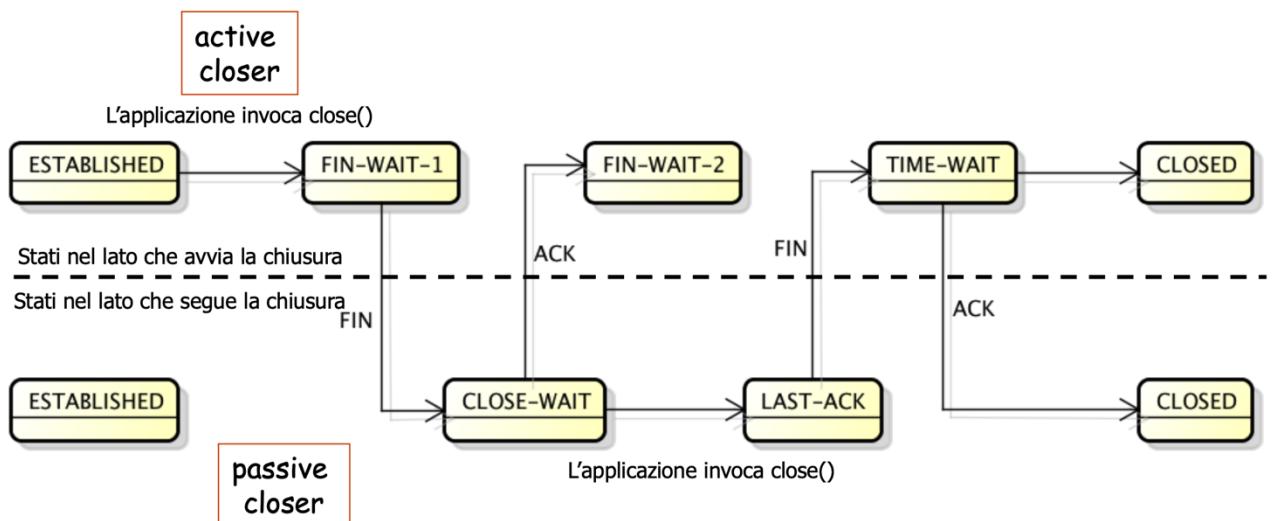
1. Il client invia il segmento di controllo TCP SYN al server:
 - a. Specifica il primo numero di sequenza;
2. Il server riceve SYN e risponde con un segmento di controllo SYN/ACK:
 - a. ACK del SYN ricevuto;
 - b. Alloca il buffer;
 - c. Specifica il primo numero di sequenza per la connessione server→client;
3. Il client riceve SYN/ACK e invia ACK al server:
 - a. Connessione instaurata.

In questa fase, inoltre, le due parti apprendono il valore iniziale di Recv Window e utilizzano le opzioni TCP per determinare il valore di MSS, RTT, ecc...

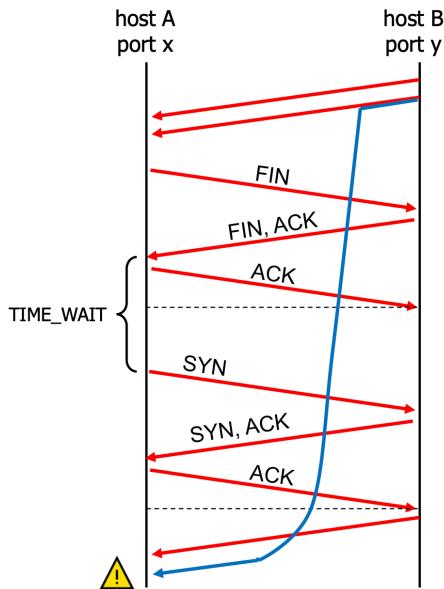
La connection termination, invece, è una procedura detta **four-way-handshake**, in cui la parte che manifesta per prima la volontà di chiudere la connessione è detta **active closer**, mentre l'altra **passive closer** (possono essere sia il client che il server, a seconda dei protocolli applicativi). Se non ha altri dati da inviare, il **passive closer** può inviare ACK e FIN nello stesso segmento, mentre l'**active closer** attende in uno stato **TIME_WAIT** (nel caso in cui l'ultimo ACK vada perso, e riceva un'ulteriore FIN dal passive closer), la cui durata dipende dal sistema operativo (tipicamente, 120 secondi).



Complessivamente, la sequenza di stati in fase di chiusura è la seguente:

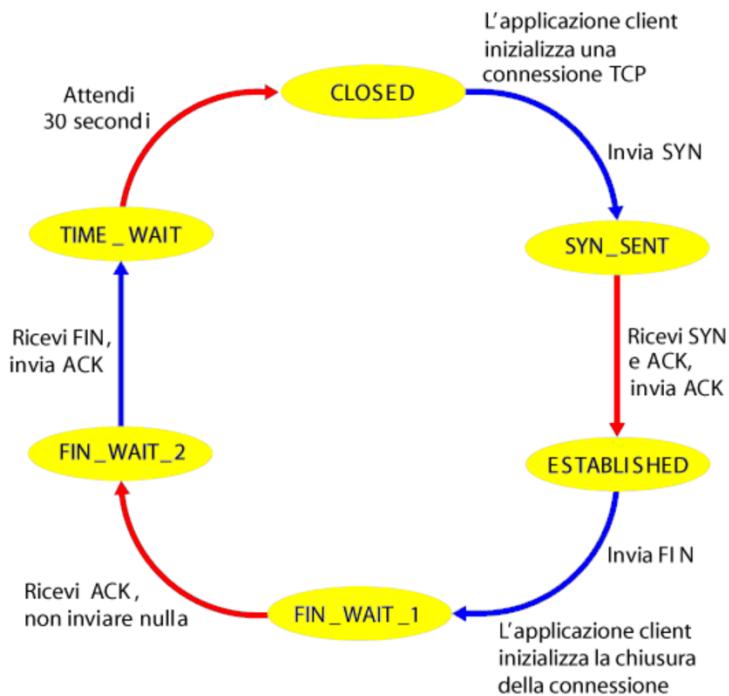


Lo stato **TIME_WAIT** è raggiunto dall'endpoint che ha preso iniziativa di chiudere la connessione TCP (active closer) e, finché è attivo, non è possibile creare una nuova connessione tra gli stessi endpoint. Ciò serve ad evitare lo scenario seguente:

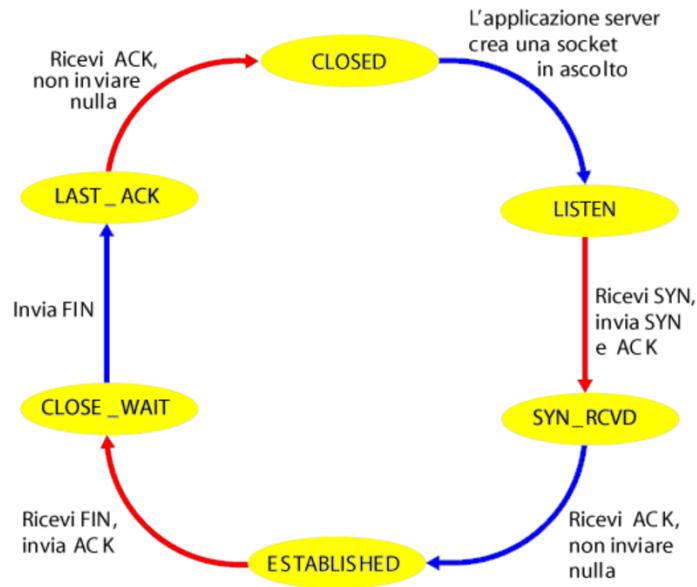


Un segmento dati duplicato prodotto da una precedente connessione tra (A,x) e (B,y) il cui numero di sequenza iniziale rientri nella finestra di ricezione corrente sarebbe erroneamente ritenuto un segmento “valido” per la nuova connessione.

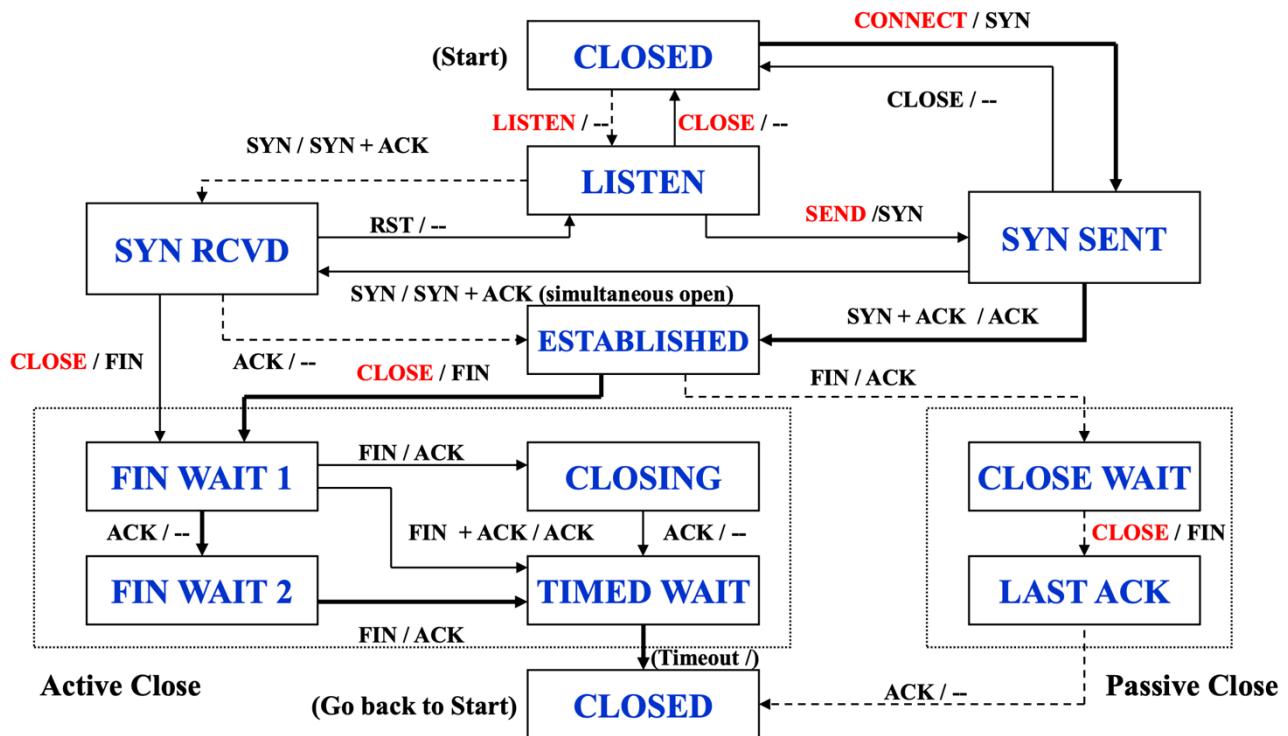
Tipicamente, la sequenza degli stati nel client è la seguente:



Mentre nel server:



Il diagramma degli stati completo del TCP, invece:



TCP crea un servizio di trasferimento dati affidabile sul servizio inaffidabile di IP tramite ACK cumulativi e un solo timer di ritrasmissione; questi ultimi sono avviati da eventi di timeout o ACK duplicati. Si consideri, inizialmente, un mittente TCP semplificato, ignorando ACK duplicati e ignorando il controllo di flusso e di congestione. Per quanto riguarda i dati ricevuti dall'applicazione, è creato un segmento con il numero di sequenza, numero del primo byte del segmento nel flusso di byte, ed è avviato il timer (se non già in funzione, bisogna pensare al timer come se fosse associato al più vecchio segmento non riscontrato) con intervallo di scadenza TimeOutInterval. Al timeout è ritrasmesso il segmento che lo ha causato ed è riavviato il timer, mentre agli ACK ricevuti, se è riscontrato un segmento precedentemente non riscontrato, si aggiorna ciò che è stato completamente riscontrato e si avvia il timer se ci sono altri segmenti da completare.

Un **sender semplificato** come quello appena illustrato appare come segue:

```

NextSeqNum=InitialSeqNumber
SendBase=InitialSeqNumber

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)
        break;

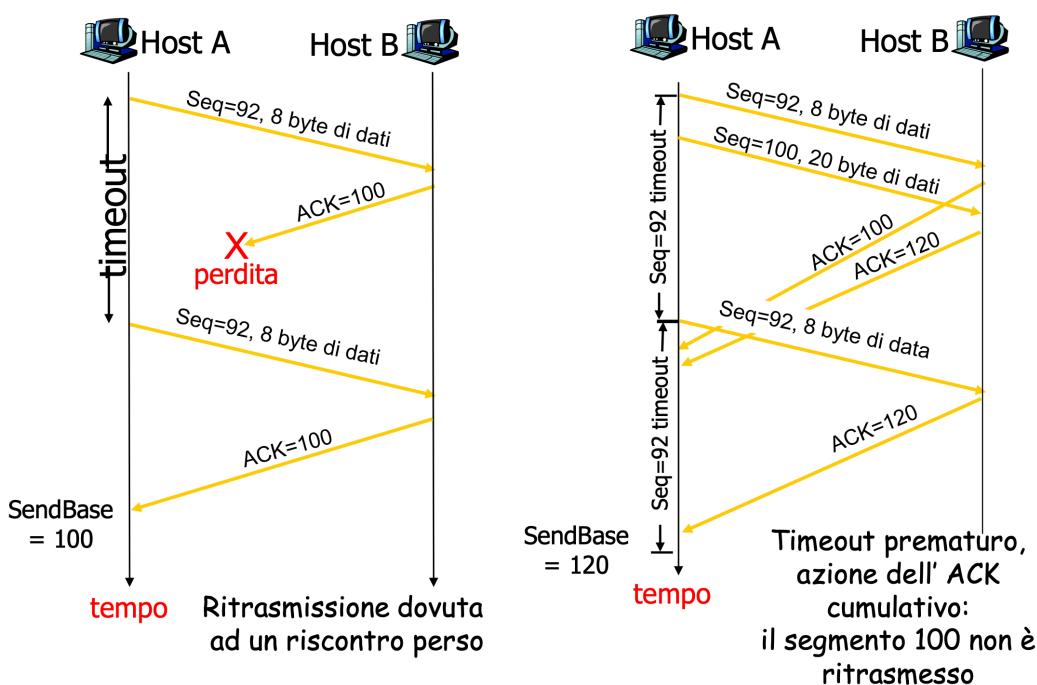
        event: timer timeout
        retransmit not-yet-acknowledged segment with
            smallest sequence number
        start timer
        break;

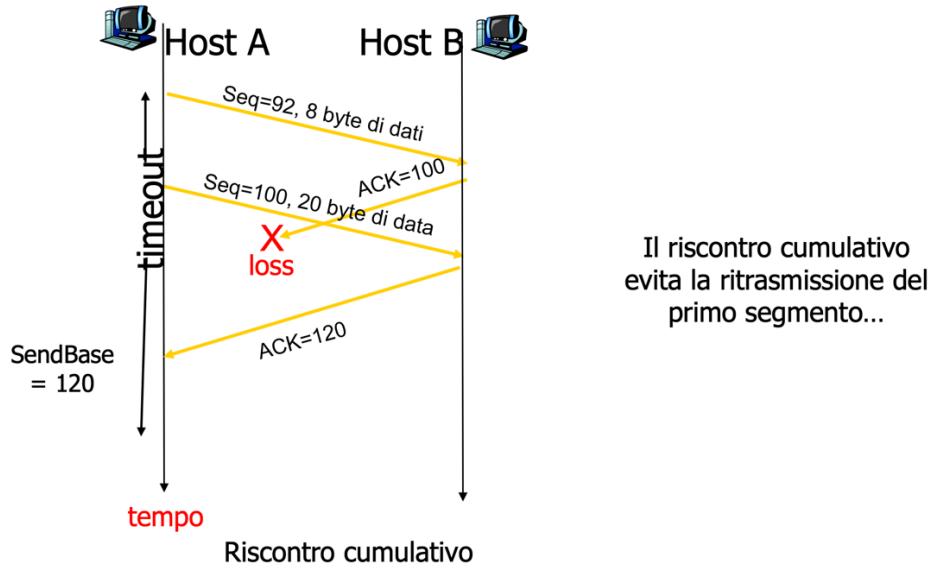
        event: ACK received, with ACK field value of y
        if (y > SendBase) {
            SendBase=y
            if (there are currently any not-yet-acknowledged
                segments)
                start timer
        }
        break;

    } /* end of loop forever */
}

```

Alcuni scenari di rilievo, invece, sono schematizzati di seguito:





Alcune modifiche tipiche del TCP sono:

- **Raddoppio dell'intervallo di timeout:**

- Allo scadere di un timeout si imposta il prossimo intervallo al doppio del valore precedente (invece che usare la stima di RTT), con una crescita esponenziale degli intervalli dopo ogni ritrasmissione;
- Quando un timer viene riavviato (ricezione di un ACK o di nuovi dati dall'applicazione), l'intervallo di timeout viene nuovamente configurato in funzione dei valori più recenti di RTT_{ext} e RTT_σ ;
- Questa soluzione fornisce una forma limitata di controllo della congestione, il mittente, nel caso supponga una situazione di congestione (ovvero, perdita di un segmento), ritrasmette ad intervalli sempre più lunghi;

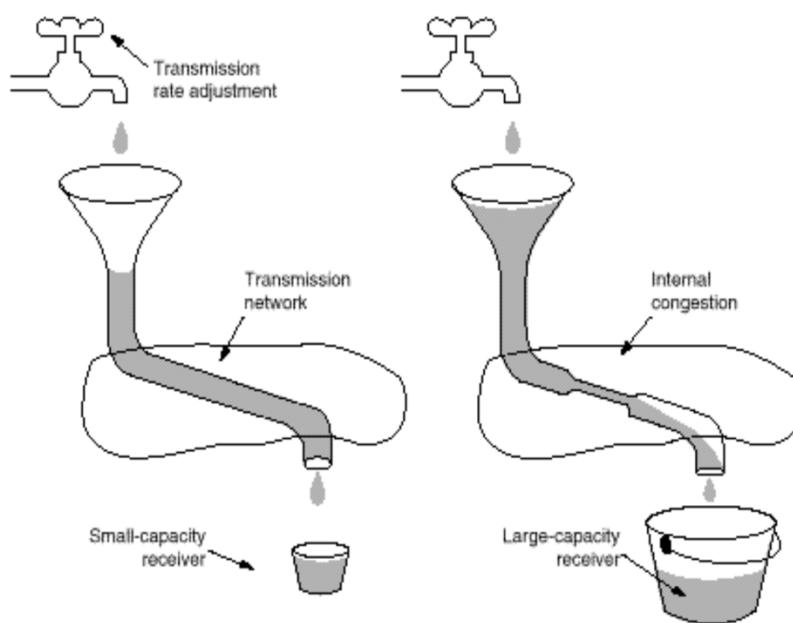
- **Ritrasmissione veloce:**

- ACK duplicati consentono di rilevare la perdita di un pacchetto prima del timeout, con un receiver che rileva un "buco" nei segmenti ricevuti (ricezione di un segmento con numero di sequenza maggiore di quello atteso) e l'invio di un nuovo riscontro per l'ultimo byte di dati che ha ricevuto correttamente;
- Poiché il mittente spesso manda molti segmenti contigui, se uno di tali segmenti si perde, ci saranno molti ACK duplicati contigui, un sender che riceve tre ACK duplicati per gli stessi dati assume che il segmento successivo a quello riscontrato tre volte è andato perso ed effettua, quindi, una ritrasmissione prima della scadenza del timeout.

Per RFC 1122, RFC 2581 e RFC 5681, la generazione di ACK può avvenire per:

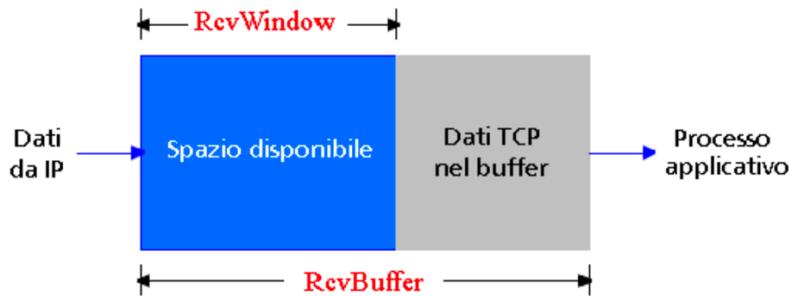
Evento nel destinatario	Azione del ricevente TCP
Arrivo ordinato di un segmento con numero di sequenza atteso. Tutti i dati fino al numero di sequenza atteso sono già stati riscontrati.	ACK ritardato. Attende fino a 500 ms l'arrivo del prossimo segmento. Se il segmento non arriva, invia un ACK.
Arrivo ordinato di un segmento con numero di sequenza atteso. Un altro segmento è in attesa di trasmissione dell'ACK.	Invia immediatamente un singolo ACK cumulativo, riscontrando entrambi i segmenti ordinati.
Arrivo non ordinato di un segmento con numero di sequenza superiore a quello atteso. Viene rilevato un buco.	Invia immediatamente un ACK (duplicato), indicando il numero di sequenza del prossimo byte atteso.
Arrivo di un segmento che colma parzialmente o completamente il buco.	Invia immediatamente un ACK, ammesso che il segmento cominci all'estremità inferiore del buco.

CONTROLLO DI FLUSSO E DI CONGESTIONE IN TCP



Adattando i due esempi, ci si chiede come gestire entrambi i tipi di controllo: receiver window, dipende dalla dimensione del buffer di ricezione, e congestion window, basata su una stima della capacità della rete. I byte trasmessi corrispondono alla dimensione della finestra più piccola.

Il lato ricevente della connessione TCP ha un buffer di ricezione, la cui lettura potrebbe rallentare il processo applicativo:



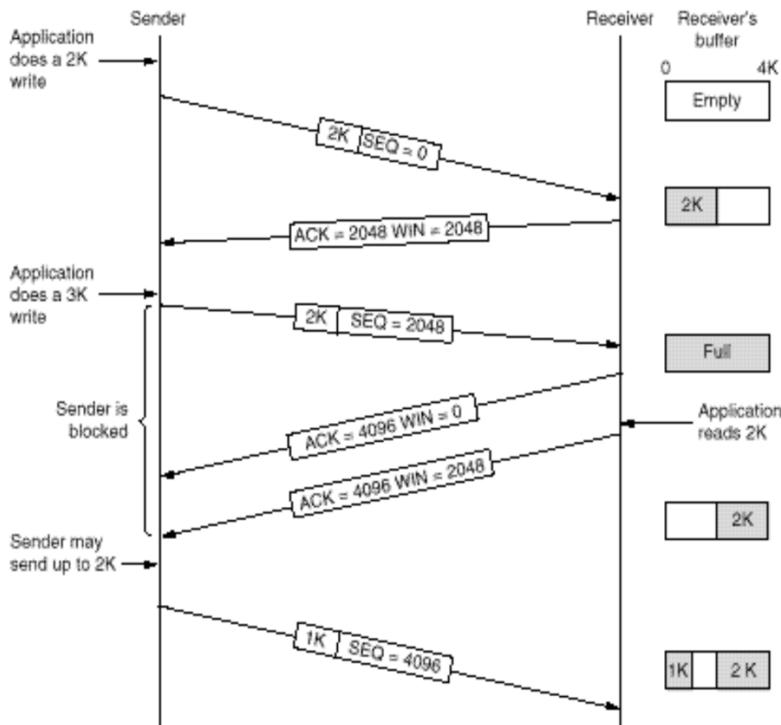
Il servizio di corrispondenza delle velocità prevede che la frequenza di invio corrisponda alla frequenza di lettura dell'applicazione ricevente (supponendo che il destinatario TCP scarti i segmenti fuori frequenza). Per **flow control** si intende quella pratica per la quale il mittente non dovrà sovraccaricare il ricevente inviando dati ad una velocità troppo elevata.

Sapendo che **RcvBuffer** è la dimensione del buffer receiver TCP e che **RcvWindow** è la quantità di spazio disponibile in un buffer, il ricevente comunica dinamicamente al mittente la dimensione corrente del buffer tramite il campo **RcvWindow** nel segmento TCP:

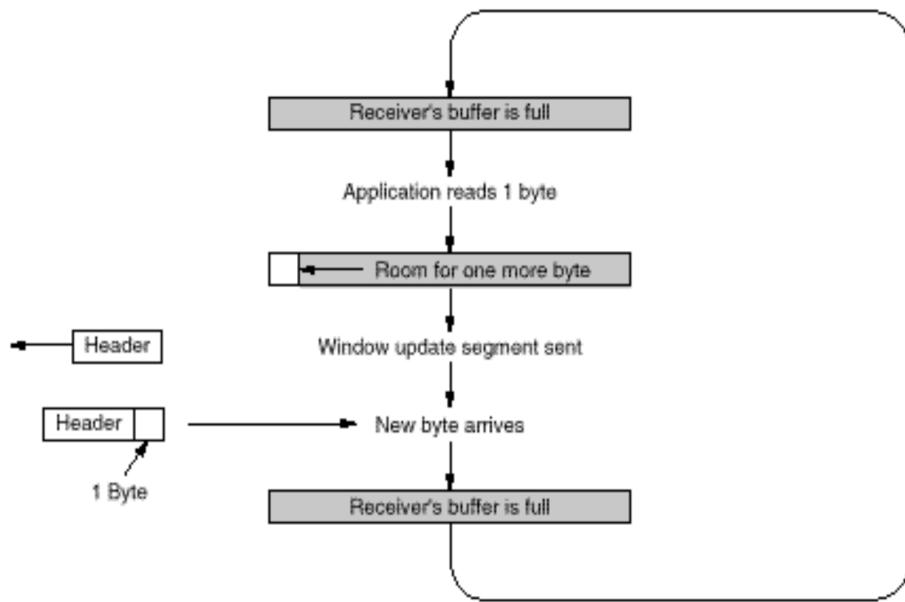
$$RcvWindow = RcvBuffer - [LastByteRcvd - LastByteRead]$$

Mentre il mittente conserva i dati già trasmessi ma non riscontrati e limita tale quantità all'ultima **RcvWindow** ricevuta:

$$LastByteSent - LastByteAcked \leq RcvWindow$$



Per **sindrome della Silly Window** al ricevitore si intende quella situazione che occorre quando il sender invia blocchi grandi ma il receiver li legge un byte alla volta:



Una possibile soluzione a questo problema prende il nome di **Soluzione di Clark** e prevede di impedire al receiver di aggiornare la finestra un byte alla volta; il ricevitore indica una finestra nulla finché il buffer di ricezione non si è svuotato per metà o per una porzione pari a MSS.

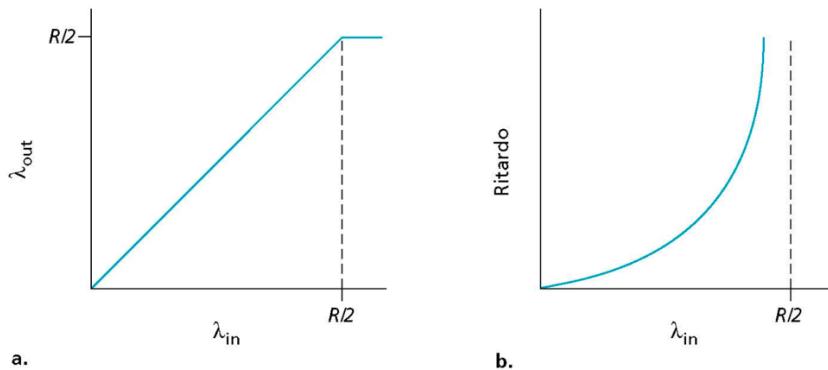
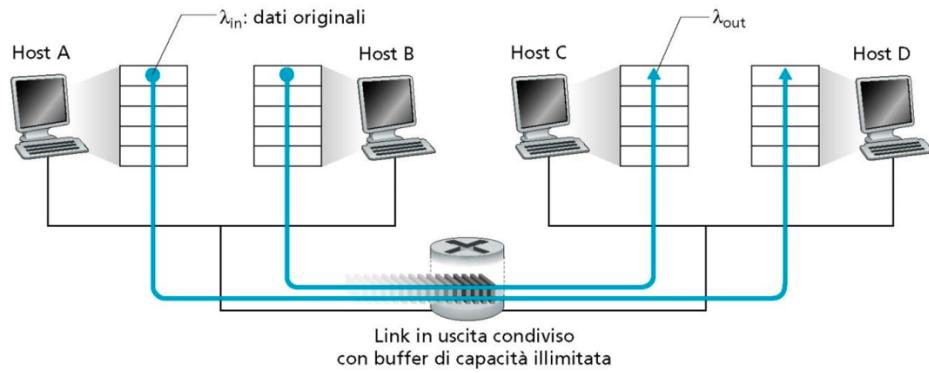
In alternativa, l'**algoritmo di Nagle** è impiegato in applicazioni che inviano dati un byte alla volta (ad esempio, Telnet) e prevede di inviare un primo byte e bufferizzare il resto finché non giunge un ACK; in seguito, invia, in un unico segmento, tutti i caratteri bufferizzati e ricomincia a bufferizzare finché non giunge l'ACK per ognuno di essi. Una pseudo-implementazione è la seguente:

```

if there is new data to send
    if window size >= MSS and available data is >= MSS
        send complete MSS segment now
    else
        if there is unconfirmed data still in the pipe
            enqueue data in the buffer until an ack is received
        else
            send data immediately
        end if
    end if
end if

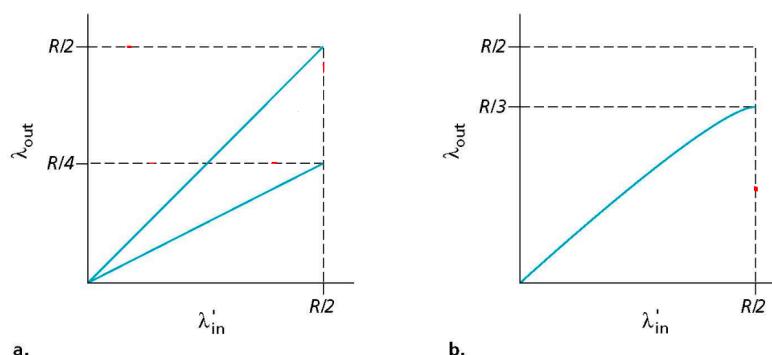
```

La congestione nella rete è tipicamente dovuta a un numero elevato di sorgenti di traffico, che inviano troppi dati o un traffico inviato ad una frequenza troppo elevata. In presenza di questi fenomeni, singoli o concomitanti, la rete è sovraccarica e produce perdita di pacchetti (buffer overflow nei router), ritardi nell'inoltro dei pacchetti (accodamenti nei buffer dei router) o scarso utilizzo delle risorse di rete. Ad esempio, con due mittenti, due riceventi e un router con buffer infinito (non ci sono ritrasmissioni), i ritardi aumentano all'avvicinarsi del limite di capacità del canale e non si può superare il massimo throughput:



Si individuano tre casi:

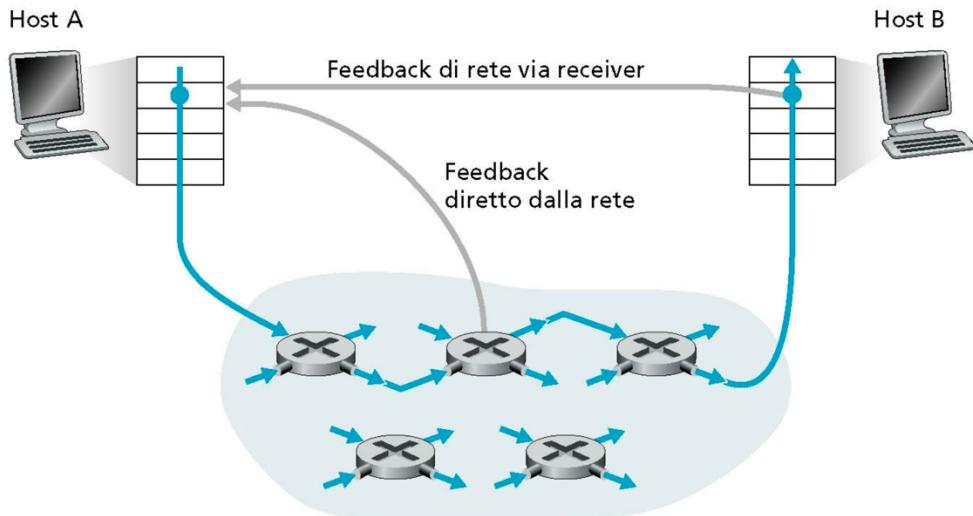
- **Il mittente invia dati solo quando il buffer non è pieno:**
 - Caso ideale, no ritrasmissioni e throughput massimo pari a $R/2$;
- **Scadenza prematura del timer del mittente:**
 - Ogni segmento è spedito, in media, due volte con throughput massimo pari a $R/4$;
- **Il mittente rispedisce un segmento solo quando è sicuro che sia andato perso:**
 - Il throughput effettivo è inferiore al carico offerto (trasmissioni dati originali + ritrasmissioni).



Le principali **tecniche di controllo della congestione** sono due ed hanno approcci ben diversi:

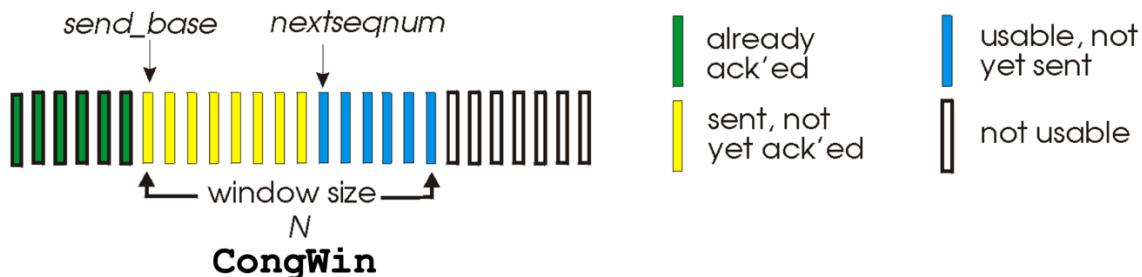
- **Approccio end-to-end** (usato da TCP), nessuna segnalazione esplicita della rete ma, a partire dall'osservazione di ritardi e perdite di pacchetti, gli end system deducono uno stato di congestione della rete;

- **Approccio in base a segnalazione della rete**, i router forniscono informazioni circa lo stato della rete agli end system tramite l'invio di un singolo bit che indica lo stato di congestione (SNA, DECbit, TCP/IP ECN, ATM) o, in alternativa, il sender è informato circa la massima frequenza alla quale può trasmettere.



In TCP, come anticipato, **il controllo della congestione avviene end-to-end**: non si ha **nessun feedback dalla rete e la frequenza di trasmissione è resa variabile, dipendente dalla cosiddetta finestra di congestione**. Considerando **controllo di flusso e di congestione insieme**, si ha:

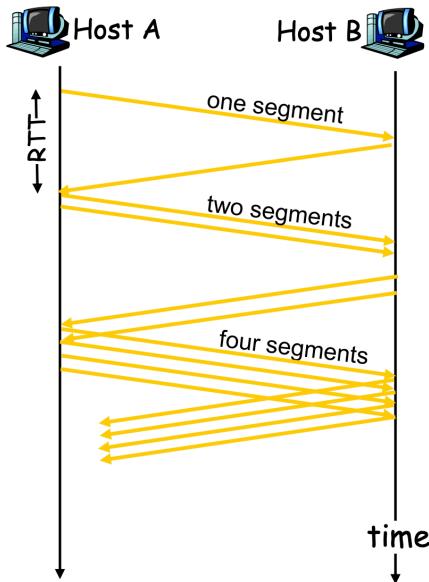
$$LastByteSent - LastByteAcked \leq \min\{RcvWindow, CongWin\}$$



Per il controllo della congestione si può procedere per tentativi per stabilire quanto si può trasmettere, con l'obiettivo di trasmettere alla massima velocità possibile (ovvero, CongWin quanto più grande possibile) senza perdite e utilizzando l'approccio seguente:

- Si aumenta CongWin finché non si verifica la perdita di un segmento, interpretata come il sopraggiungere dello stato di congestione;
- Alla perdita di un segmento:
 - Si decrementa CongWin;
 - Si ricomincia da capo.

Con questo approccio, si possono distinguere due fasi principali: la **slow start**, che corrisponde al momento in cui la **velocità di trasmissione non è ancora massimizzata**, e un **congestion avoidance**, per il quale avviene un **incremento additivo e un decremento moltiplicativo della velocità di trasmissione** (Additive Increase, Multiplicative Decrease, AIMD).



L'algoritmo di slow start appare come segue:

```
//initialization
Congwin = 1 MSS

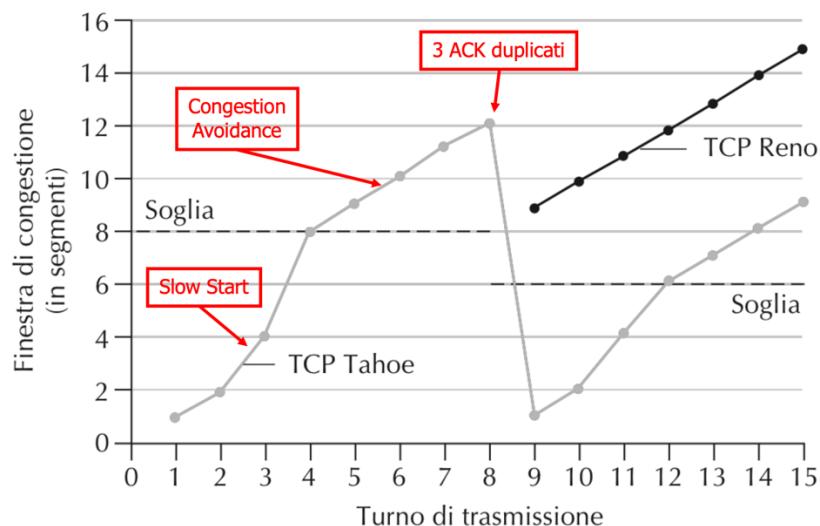
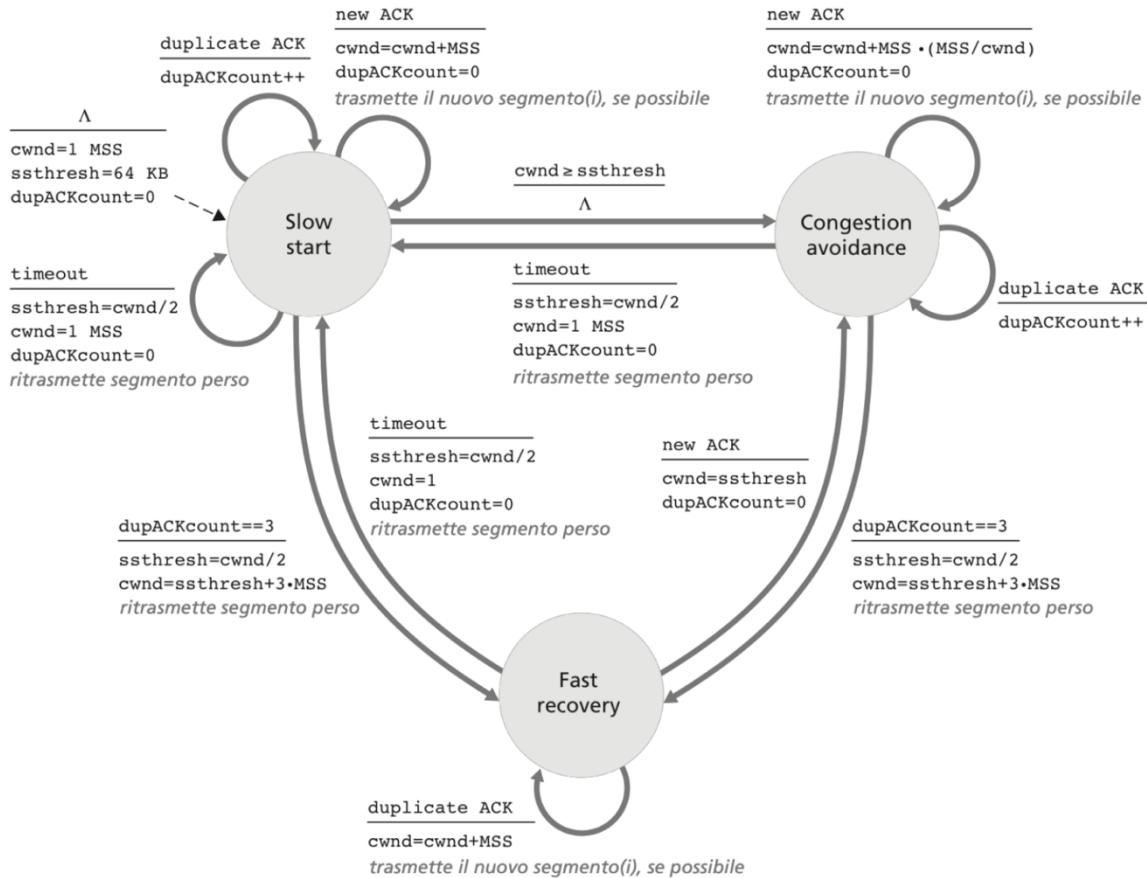
for (each segment ACKed)
    Congwin=Congwin+1MSS
until (loss event OR
    CongWin > threshold)
```

Con una **crescita esponenziale della dimensione della finestra** (ogni RTT), che rende “slow start” un termine improprio. Un evento di perdita può essere un timeout, tre ACK duplicati consecutivi o un ritorno a CongWin pari a 1MSS.

Una volta raggiunta la soglia, si raggiunge la situazione di **Congestion Avoidance**: ci si avvicina con cautela al valore della banda disponibile tra le due estremità della connessione (Additive Increase), incrementando CongWin di $MSS \cdot (MSS/CongWin)$ alla ricezione di un ACK. Al sopraggiungere della congestione si ha un **Multiplicative Decrease**:

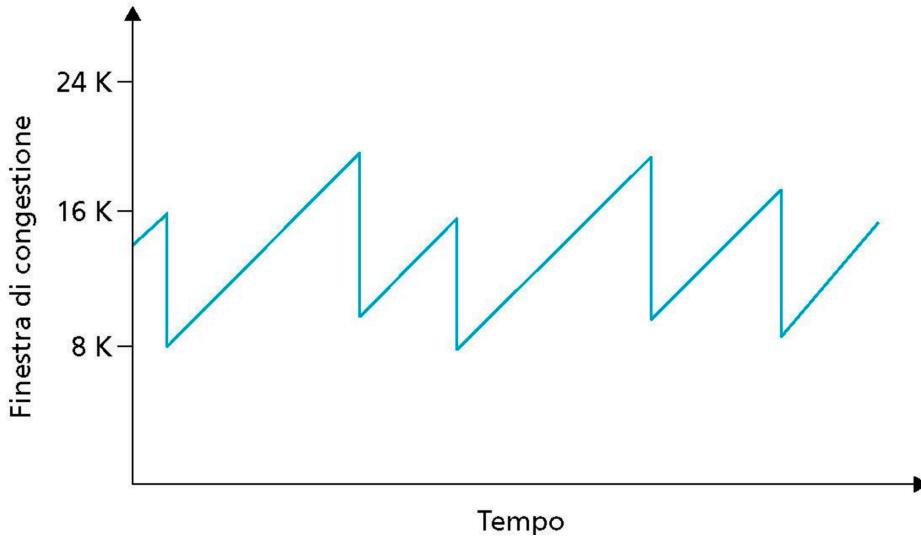
- **Scadenza di un timeout**, si ha uno slow start (Threshold pari a $CongWin/2$ e CongWin pari a 1MSS);
- **Ricezione di tre ACK duplicati consecutivi**:
 - TCP versione Tahoe, slow start;
 - TCP versione Reno, fast recovery (Threshold pari a $CongWin/2$ e CongWin pari a Threshold + 3MSS).

TCP Reno elimina la fase di Slow Start dopo un evento di perdita dovuto dalla ricezione di tre ACK duplicati; tale evento indica che, nonostante si sia perso un pacchetto, almeno tre segmenti successivi sono stati ricevuti dal destinatario. A differenza del caso del timeout, però, la rete mostra di essere in grado di consegnare una certa quantità di dati ed è possibile evitare una nuova slow start, ricominciando direttamente dalla fase di fast recovery.



Volendo riassumere quanto detto finora, se le finestre di congestione è sotto la soglia si raggiunge la condizione di slow start con una crescita esponenziale della finestra, se la finestra supera la soglia si ha una prevenzione della congestione con una crescita lineare, quando si raggiunge l'evento di perdita dedotto da ACK duplicato tre volte, la soglia è posta alla metà del valore attuale della finestra e, in caso di TCP Reno la finestra è posta alla soglia + 3MSS, nel caso di TCP Tahoe, la finestra è posta pari ad un segmento (MSS). Infine, se l'evento di perdita è dedotto da un timeout, la soglia è posta alla metà del valore attuale della finestra e la finestra è pari ad un segmento.

Con TCP Reno, in assenza di timeout, l'andamento della finestra di congestione è un andamento a dente di sega:



Se il massimo valore di CongWin è costante (W) e lo è anche RTT, il throughput medio di una connessione TCP è:

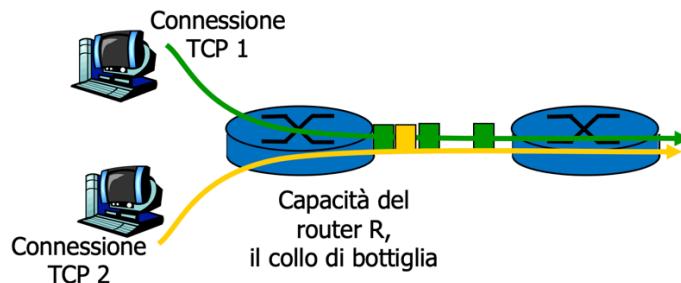
$$T = \frac{0.75 \cdot W}{RTT} = \frac{W^*}{RTT}$$

Con W^* valore medio di CongWin. La stima del throughput di una connessione TCP è stata oggetto di diversi studi, che hanno permesso di semplificare la formula a quella appena illustrata; tuttavia, la formula in questione ignora l'effetto della perdita di pacchetti. Detto L il tasso medio di perdita di un pacchetto, è possibile concludere con più precisione:

$$T = \frac{C \cdot W}{RTT \cdot \sqrt{L}}$$

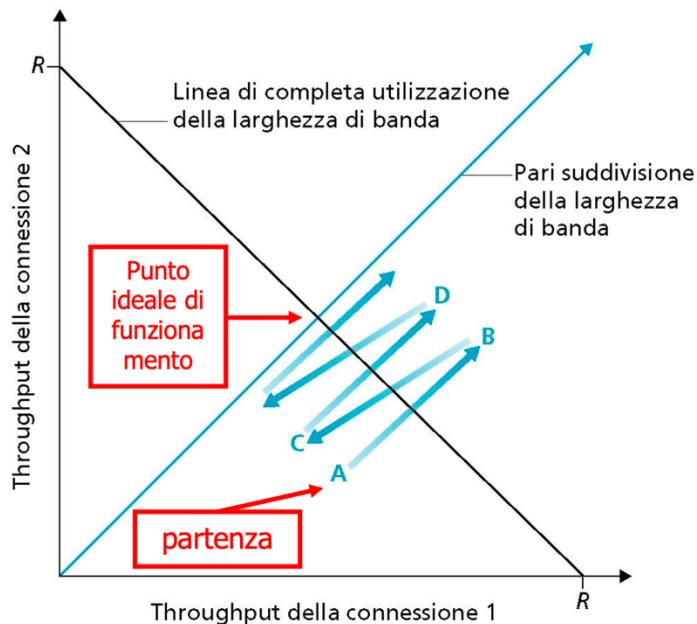
Sapendo che $C = \sqrt{3/2} \approx 1.22$.

Se K connessioni TCP condividono lo stesso collegamento con ampiezza di banda R , che è un collo di bottiglia per il sistema, ogni flusso dovrebbe avere un tasso di trasmissione medio pari a R/K . La proprietà appena enunciata prende il nome di equità (fairness) tra le connessioni TCP.



Per quanto riguarda AIMD, si considerino MSS e RTT uguali per le due connessioni: a parità di dimensioni della finestra, quindi, il throughput è lo stesso ed entrambe le connessioni si trovano

oltre lo slow start; di conseguenza, ci si trova nella fase di prevenzione della congestione, con incremento additivo e decremento moltiplicativo.



Il controllo della congestione in TCP segue il modello AIMD (Additive Increase, Multiplicative Decrease), assicurando un utilizzo equo della banda quando più connessioni competono per le stesse risorse; ad esempio, se due flussi TCP con lo stesso RTT e MSS competono per un router con capacità R , entrambi si adatteranno progressivamente per convergere verso una suddivisione equa della larghezza di banda. Questo comportamento equo si verifica solo se i flussi utilizzano la stessa implementazione TCP.

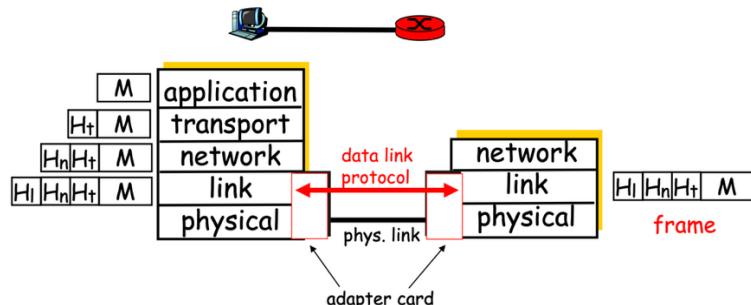
Quando un flusso utilizza un protocollo diverso, come UDP che non implementa il controllo della congestione, la banda non viene distribuita equamente. UDP può saturare la rete senza limitazioni, causando problemi agli altri flussi; similmente, se due connessioni TCP utilizzano implementazioni diverse (es. Reno e Tahoe), il flusso Reno, con un incremento più rapido nella fase di congestion avoidance, avrà un throughput maggiore, compromettendo l'equità.

Gli ISP spesso limitano il traffico UDP per evitare questi problemi, poiché i flussi TCP garantiscono un controllo della congestione più equilibrato.

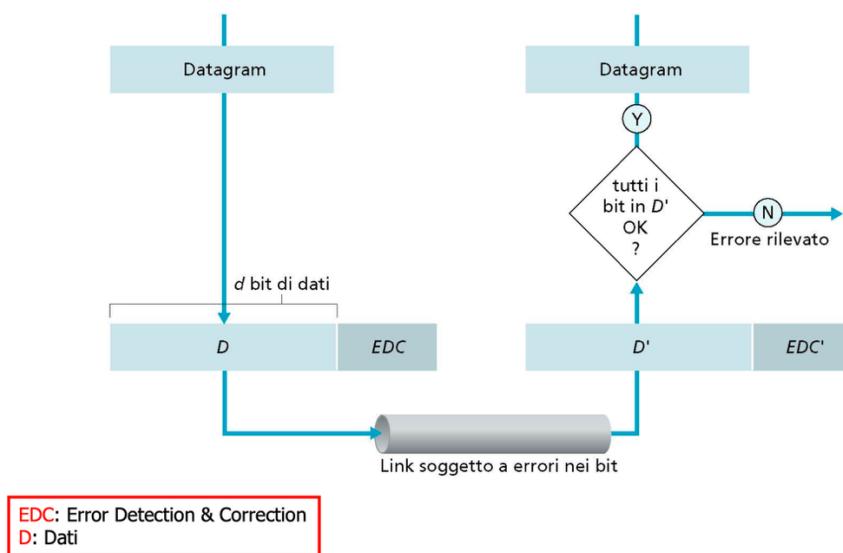
IL LIVELLO DATA LINK

IL LIVELLO DATA LINK E I PROTOCOLLI AD ACCESSO MULITIPLO

Il livello data link si occupa della comunicazione diretta tra due dispositivi fisicamente connessi (host-router, router-router, host-host) e considera, come unità dati, i frame.

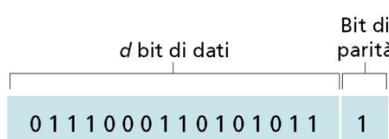


Per Framing ed accesso ai link si intende la pratica di incapsulamento di datagrammi all'interno di frame, con l'aggiunta dei relativi header e trailer, la gestione all'accesso al canale, in caso di mezzo condiviso, e l'utilizzo di indirizzi fisici all'interno dei frame stessi, per poter identificare nodo sorgente e destinazione. Il trasferimento dei dati tra due dispositivi fisicamente connessi è affidabile, utile soprattutto in caso di collegamenti con alta probabilità di errore, come i link wireless; ciò è ottenuto tramite controllo di flusso, regolando la velocità di trasmissione tra mittente destinatario. Il livello data link, pertanto, permette anche la rilevazione degli errori (causati da attenuazione del segnale o da presenza di rumore) da parte del ricevente, il quale segnala l'evento al mittente ed elimina il frame ricevuto o identifica e corregge gli errori su alcuni bit del frame, evitando ritrasmissioni da parte del mittente.



Il controllo e la correzione eventuale di errori possono avvenire per:

- **Controllo di parità ad un bit**, rileva errori su un singolo bit;



- **Controllo di parità a due bit**, rileva ed eventualmente corregge errori su un singolo bit o rileva errori su due bit.

		Nessun errore	Errore correggibile del singolo bit
Parità di riga		1 0 1 0 1 1	1 0 1 0 1 1
$d_{1,1}$...	1 1 1 1 0 0	1 0 1 1 1 0 0
$d_{2,1}$...	0 1 1 1 0 1	0 1 1 1 0 1
...	...	0 0 1 0 1 0	0 0 1 0 1 0
$d_{i,1}$...	$d_{i,j+1}$	$d_{i+1,j+1}$
$d_{i+1,1}$...	$d_{i+1,j}$	$d_{i+1,j+1}$

Parità di colonna

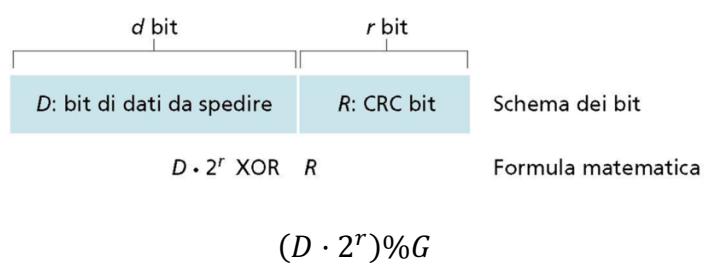
Errore di parità

Errore di parità

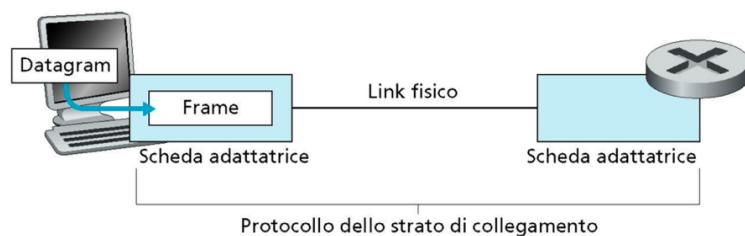
I metodi di checksum, invece, hanno l'obiettivo di rilevare errori su bit all'interno dei segmenti trasmessi (tecnica tipicamente utilizzata a livello trasporto) e dividono il lavoro per:

- **Mittente**, tratta il contenuto del segmento come una sequenza di interi espressi su 16 bit e calcola la checksum come complemento ad uno della somma in complementi ad uno del contenuto del segmento e la inserisce all'interno di un apposito campo dell'header del segmento;
- **Ricevitore**, calcola la somma in complemento ad uno dei dati ricevuti (compresa la checksum) e verifica se il risultato è composto da tutti uno:
 - No, c'è un errore;
 - Si, non è stato rilevato nessun errore ma (come per il livello trasporto) non vuol dire che non ci siano stati errori.

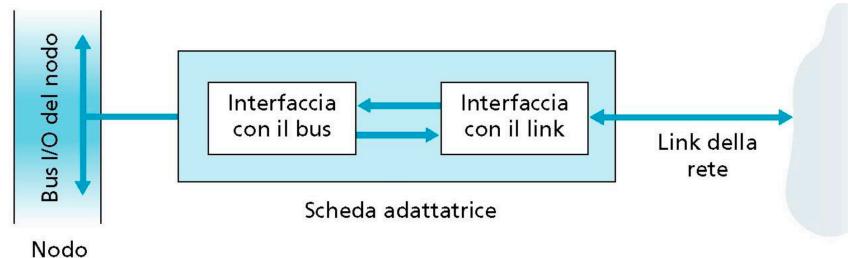
In alternativa, è possibile usare i codici CRC (Cyclic Redundancy Check), tecnica molto usata nella realtà pratica al livello data link. Si considerino i bit di dati, D, come un numero binario, si scelga un opportuno pattern di $r+1$ bit, G e ci si pone l'obiettivo di scegliere r bit di controllo CRC, R, tali che $\langle D, R \rangle$ sia divisibile esattamente per G (modulo 2). Il ricevente, che deve conoscere G, divide i due membri e se il resto della divisione non è nullo, rileva un errore. Con questa tecnica, si possono rilevare tutti gli errori che coinvolgono meno di $r+1$ bit.



Per adattatore di rete si intende un circuito (ad esempio, una scheda PCMCIA) che si occupa di ricevere datagrammi dallo strato di rete, incapsulare i datagrammi ricevuti all'interno di un frame, trasmettere i frame all'interno dei link di comunicazione e, in ricezione, effettuare le operazioni inverse.

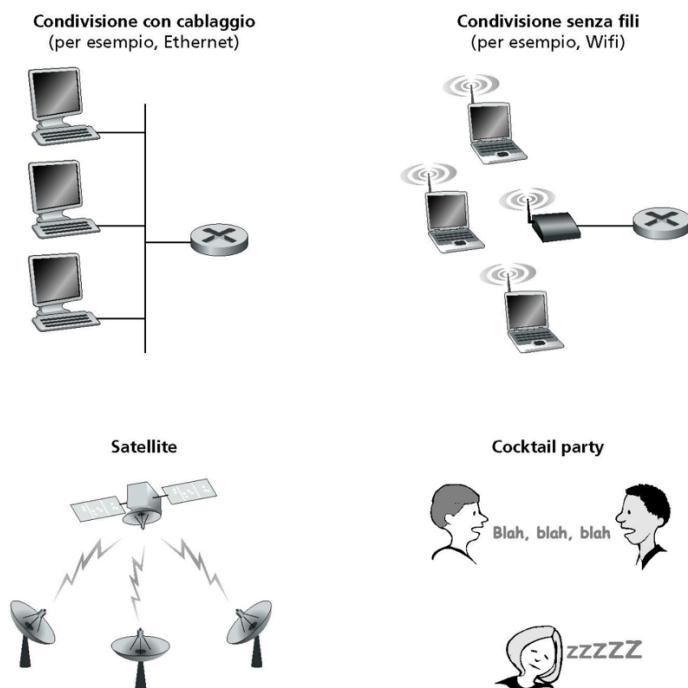


Gli adattatori di rete sono entità semi-autonome: alla ricezione di un frame, **il nodo è coinvolto solo se i dati devono essere passati al livello rete e, un nodo che spedisce un datagramma, delega completamente all'adattatore di rete la trasmissione sul link.** Un adattatore, quindi, è dotato di memoria RAM, di un DSP, di un chip di elaborazione dei frame e di interfacce verso il bus dell'host e verso il link.



Sostanzialmente, esistono due tipi di link:

- **Punto-punto**, utilizzano protocolli Point to Point Protocol (PPP) o Serial Line IP (SLIP);
- **Broadcast**, a mezzo condiviso (Ethernet, Wireless LAN WiFi, Satellite, Reti Cellulari, ...).



Per **protocolli ad accesso multiplo** si intendono protocolli il cui **unico canale di comunicazione è unico**, permettendo **due o più trasmissioni simultanee** da parte dei nodi della rete ma **con il rischio di interferenze** (solo un nodo potrà inviare dati con successo). Un **algoritmo distribuito** determina le modalità di condivisione del canale, ovvero quando una stazione può trasmettere, con le comunicazioni per regolare l'accesso al canale che utilizzano il canale stesso. Le caratteristiche di un protocollo di questo tipo sono le seguenti:

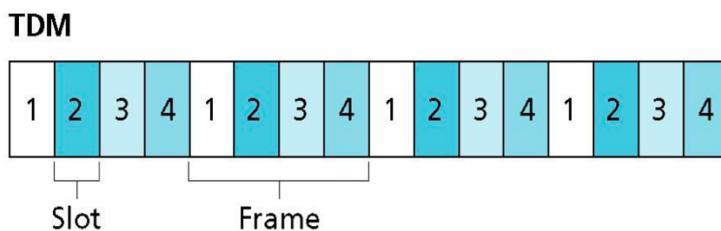
- **Sincronia o asincronia;**
- **Necessità di informazioni riguardanti le altre stazioni;**
- **Robustezza** (ad esempio, in relazione agli errori sul canale);
- **Prestazioni.**

Alcuni **protocolli di accesso multiplo** sono:

- **Channel partitioning**, condivisione efficiente con carico elevato ma poco efficienti con carico leggero (ritardo nell'accesso al canale e banda pari a $1/N$ anche se solo uno degli N nodi è attivo);
- **Random Access**, efficienti con carico leggero (un singolo nodo può utilizzare a pieno il canale) ma poco efficienti con carico elevato (overhead dovuto alla presenza di collisioni);
- **Taking Turns**, cercando di prendere il meglio dagli approcci precedenti:
 - Polling, un nodo master “invita” i nodi “slave” a trasmettere in maniera alternata, con un impiego di messaggi del tipo “Request to Send” o “Clear to Send”
 - Token passing, un token di controllo viene passato da un nodo al successivo in maniera sequenziale e il cui possesso dà diritto alla trasmissione;
 - Entrambi gli approcci presentano un overhead (dovuto al polling e al token), alta latenza e la presenza di un single point of failure (nel master e nel token);

Per un **canale con velocità di R bit/s**, se **un solo nodo ha dati da inviare**, quel nodo ha un throughput di R bit/s, se **M nodi hanno dati da spedire, ognuno di essi ha un throughput medio di R/M bit/s**. Il protocollo per la gestione dell'accesso è **distribuito**, si hanno assenze di single points of failure ed ha un'implementazione economica, essendo semplice.

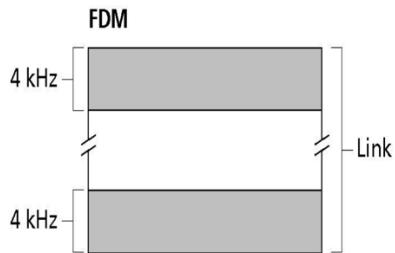
Uno dei protocolli di suddivisione del canale è **TDMA**, Time Division Multiple Access. L'accesso al canale avviene a “cicli”: ogni stazione ottiene uno slot di trasmissione di lunghezza fissa in ogni ciclo e quelli non utilizzati da una stazione vanno deserti. TDMA permette di eliminare le collisioni ed è un protocollo equo, seppur abbia un massimo throughput per un nodo, in una rete di N stazioni, pari a R/N bit/s, nonostante il nodo in esame sia l'unico ad avere frame da spedire, e seppur un nodo debba sempre aspettare il proprio turno nella sequenza di trasmissione.



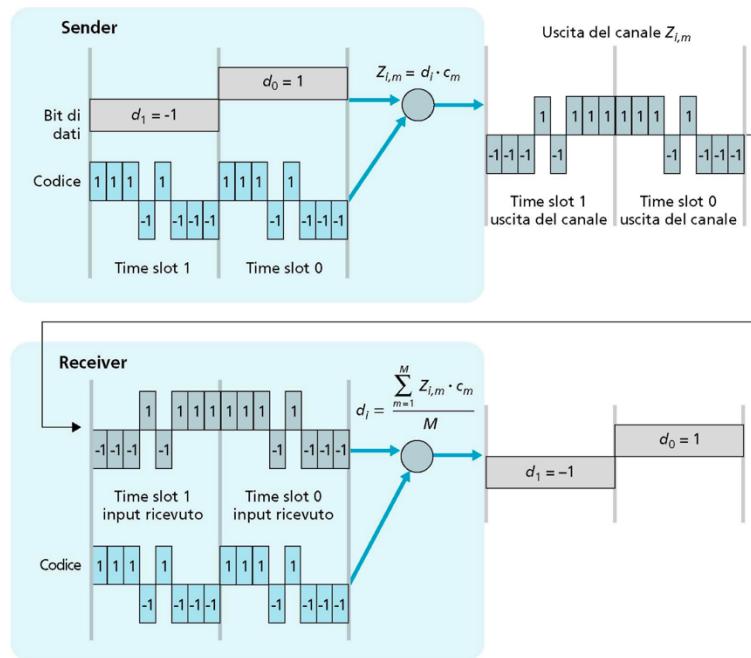
Legenda:

2 Tutti gli slot etichettati “2” sono dedicati
a una specifica coppia sender-receiver

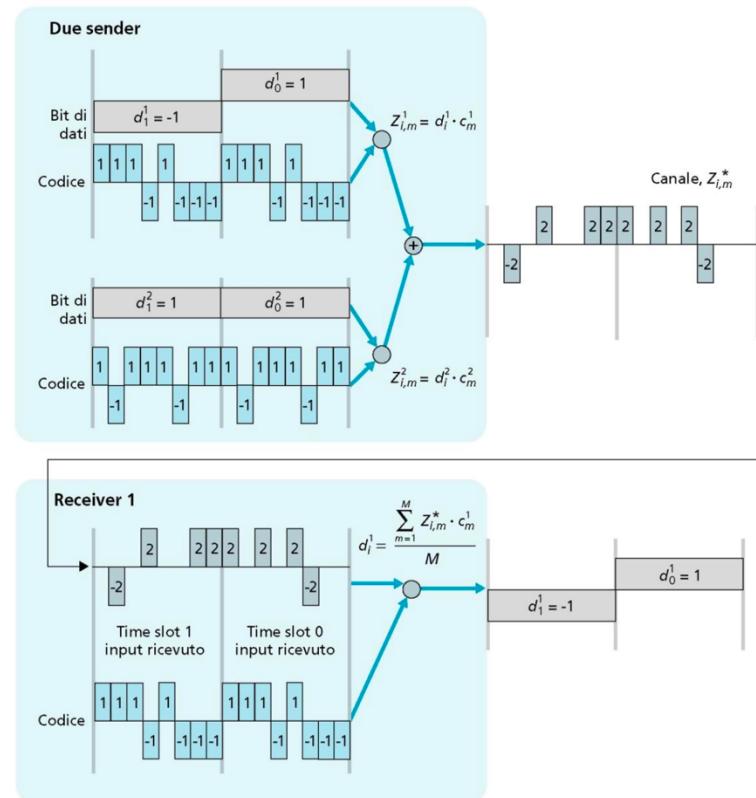
In alternativa, si può usare **FDMA** (Frequency Division Multiple Access), per cui lo spettro di trasmissione è diviso in bande di frequenze, assegnate alle stazioni, e per cui il tempo di trasmissione inutilizzato nelle singole bande di frequenza risulta sprecato. Questo approccio presenta gli stessi vantaggi e gli stessi svantaggi dell'omologo nel dominio del tempo (TDMA).



È possibile usare anche CDMA (Code Division Multiple Access), per cui un codice unico è assegnato ad ogni utente (code set partitioning). Tutti gli utenti condividono la stessa frequenza di trasmissione ma ognuno di essi codifica i propri dati sulla base della chipping sequence, il codice univocamente assegnato, scalarmente moltiplicato con i dati originali o con il segnale codificato per, rispettivamente, codificare e decodificare le informazioni. CDMA consente a diversi nodi di trasmettere simultaneamente, riducendo al minimo l'interferenza nel caso in cui siano stati scelti codici ortogonali, ed è usato principalmente nei canali wireless di tipo broadcast (reti cellulari, satellitari, ...).

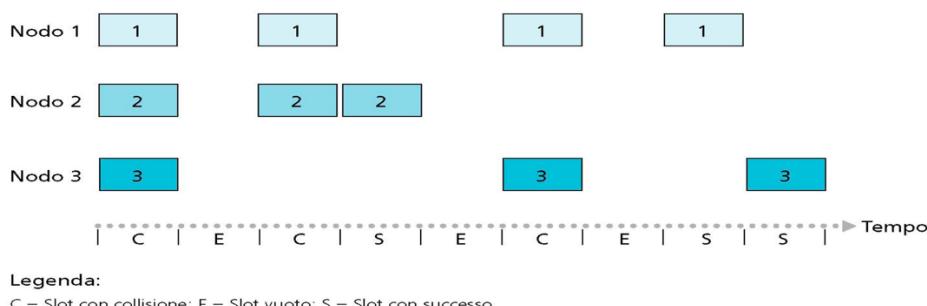


Un evento di interferenza assume la seguente forma (* CDMA lavora nell'ipotesi che i segnali dei bit trasmessi con interferenza siano cumulativi):



Altri protocolli che possono essere impiegati prendono il nome di **protocolli ad accesso casuale** (slotted ALOHA, ALOHA, CSMA e CSMA/CD) e prevedono la trasmissione di un pacchetto di un nodo alla massima velocità consentita dal canale, senza alcuna forma di coordinamento a priori tra i nodi. La trasmissione simultanea di due o più nodi, seguendo questo approccio, porta inevitabilmente alla collisione. Un protocollo ad accesso casuale specifica come rilevare e risolvere queste collisioni.

Slotted ALOHA prevede che tutti i pacchetti abbiano lunghezza di L bit e che il tempo sia diviso in slot di uguale durata (L/R s); se un nodo ha dati disponibili per la trasmissione, trasmette all'inizio del primo slot disponibile (sapendo che tutti i nodi sono sincronizzati, cioè sanno quando iniziano gli slot) e in caso di collisione ritrasmette il pacchetto negli slot successivi, con probabilità ρ , finché l'operazione non va a buon fine.



Qual è la percentuale massima di slot in cui la trasmissione ha successo? Supponendo che N stazioni abbiano frame da trasmettere, ogni stazione trasmette in un determinato slot, con probabilità ρ , e la probabilità S che una trasmissione abbia successo è data, per il singolo nodo, da:

$$s = \rho(1 - \rho)^{N-1}$$

Ma, dato che i nodi sono N:

$$S = Ns = N\rho(1 - \rho)^{N-1}$$

Il valore ottimo di S, per N che tende ad infinito, è:

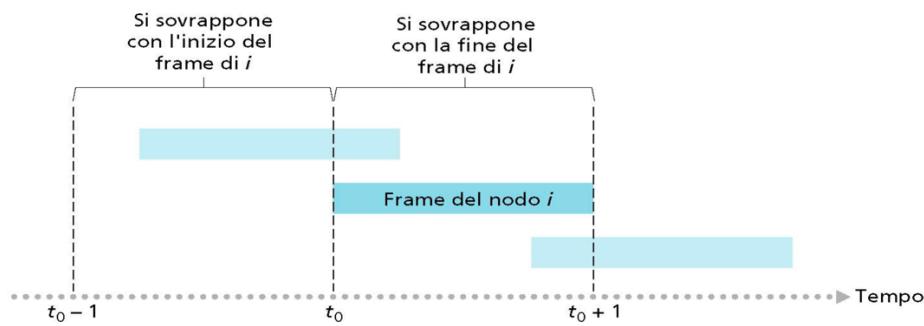
$$\lim_{N \rightarrow +\infty} S = \lim_{N \rightarrow +\infty} N\rho(1 - \rho)^{N-1} = \frac{1}{e} \approx 0.37$$

Cioè, circa il 37%. Considerando $\rho^* = 1/N$ la probabilità di trasmissione che conduce al migliore risultato globale, all'aumentare di N la singola stazione dovrà trasmettere sempre più raramente ed avrà una bassa probabilità di successo, con un throughput globale che tende sempre a $1/e$.

N	$p^*=(1/N)$	$p^* \cdot (1-p^*)^{N-1}$	$N \cdot p^* \cdot (1-p^*)^{N-1}$
2	0,500	0,250	0,5000
3	0,333	0,148	0,4444
4	0,250	0,105	0,4219
5	0,200	0,082	0,4096
6	0,167	0,067	0,4019
7	0,143	0,057	0,3966
8	0,125	0,049	0,3927
9	0,111	0,043	0,3897
10	0,100	0,039	0,3874
11	0,091	0,035	0,3855
12	0,083	0,032	0,3840
13	0,077	0,029	0,3827
14	0,071	0,027	0,3816
15	0,067	0,025	0,3806
16	0,063	0,024	0,3798
100	0,010	0,004	0,3697
1000	0,001	0,0004	0,3681
10000	0,0001	0,00004	0,3679

$$1/e = 0,36787944$$

ALOHA (unslotted) è più semplice e non richiede la sincronizzazione dei nodi, in trasmissione invia il frame non appena i dati sono disponibili ma la probabilità di collisione raddoppia: un frame inviata al tempo t_0 può collidere con altri frame inviati in $[t_0 - 1, t_0 + 1]$.



La probabilità di successo di un nodo è data dal prodotto tra la probabilità che il nodo trasmetta, la probabilità che nessun altro nodo trasmetta nell'istante immediatamente precedente e la probabilità che nessun altro nodo trasmetta nell'istante immediatamente successivo:

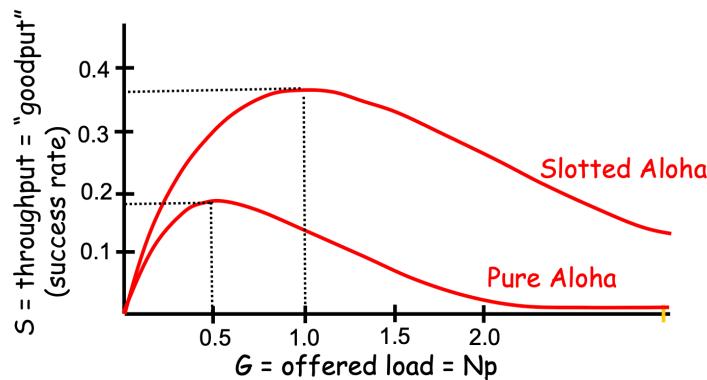
$$S = S(t_0) \cdot \bar{S}(t \in [t_0 - 1, t_0]) \cdot \bar{S}(t \in [t_0, t_0 + 1]) = \rho \cdot (1 - \rho)^{N-1} \cdot (1 - \rho)^{N-1}$$

$$= \rho(1 - \rho)^{2(N-1)}$$

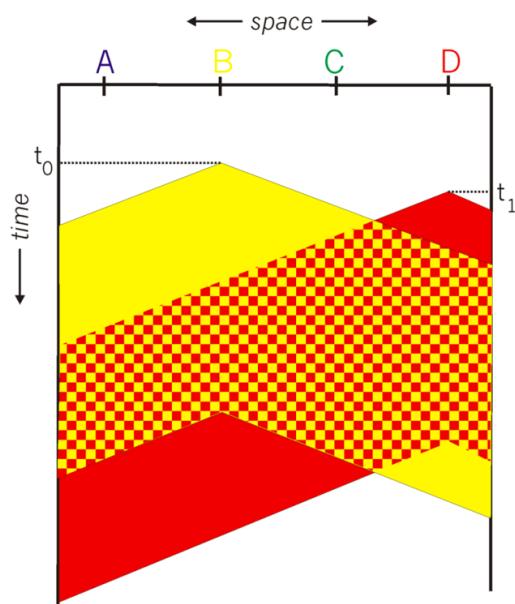
Il valore ottimo, per N che tende ad infinito, è:

$$\lim_{N \rightarrow +\infty} S = \lim_{N \rightarrow +\infty} N\rho(1 - \rho)^{2(N-1)} = \frac{1}{2e} \approx 0.18$$

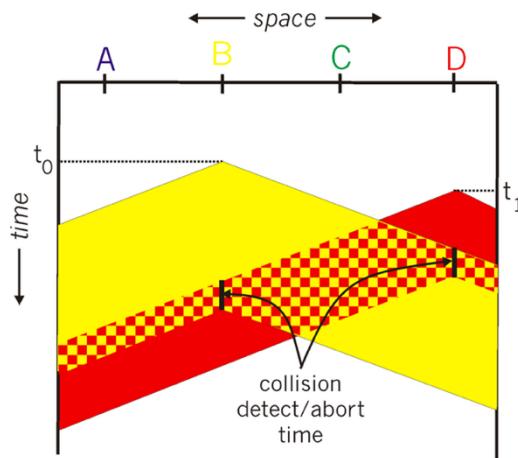
Cioè, circa il 18%. In questo caso, rispetto a slotted ALOHA, il protocollo limita il throughput effettivo del canale di trasmissione:



CSMA, invece, “ascolta prima di parlare”; ovvero: se il canale è libero permette la trasmissione del frame, altrimenti rimanda la trasmissione. CSMA persistente, invece, riprova immediatamente con probabilità ρ quando il canale si libera e CSMA non persistente riprova dopo un intervallo casuale. Con CSMA si possono avere delle collisioni, il ritardo di propagazione fa sì che due nodi possano non ascoltare le reciproche trasmissioni; in caso di collisione, il tempo di trasmissione del frame risulta essere compeltamente sprecato. La distanza ed il ritardo di propagazione concorrono a determinare la probabilità di collisione.



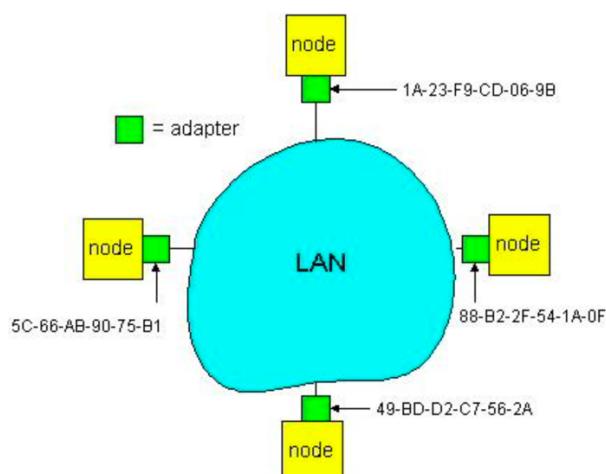
CSMA/CD è **CSMA con Collision Detection**, rileva le collisioni durante la trasmissione e la elimina, riducendo lo spreco di risorse del canale trasmittivo, con eventuali ritrasmissioni persistenti o meno. In concreto, la Collision Detection funziona tramite misurazione della potenza del segnale ricevuto e la si compara con quella del segnale trasmesso:



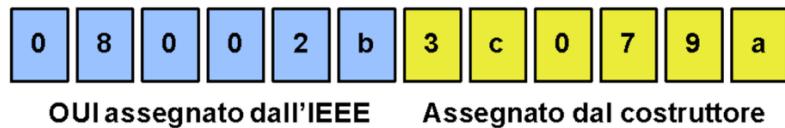
RETI LOCALI ETHERNET

Come già anticipato in precedenza, **due dispositivi fisicamente connessi non comunicano con indirizzi IP**, usati a livello di rete e per permettere la corretta consegna del pacchetto ad un destinatario collegato alla rete, **ma con indirizzi fisici** (indirizzi MAC o indirizzi propriamente fisici), usati per permettere la trasmissione di un frame da una scheda di rete ad un'altra con cui sussiste un collegamento diretto (ovvero, sono situati nella stessa rete fisica). Per la maggior parte delle LAN, gli indirizzi MAC sono di 48 bit e sono cablati nelle ROM delle schede di rete, non permettendone così la modifica e associanone uno univoco per dispositivo.

La distribuzione degli indirizzi MAC è gestita da IEEE, che associa ai produttori di schede di rete una porzione dello spazio di indirizzamento (sempre per garantirne l'univocità ma per mantenere una struttura simil-decentralizzata). Gli indirizzi MAC godono di una certa portabilità: essendo cablati nelle schede di rete, si può spostare una scheda da una LAN ad un'altra e mantenere lo stesso indirizzo MAC, cosa che non è possibile con gli indirizzi IP, dal momento in cui dipendono dalla rete e non dal dispositivo.



Gli indirizzi MAC si compongono di due parti da 3 byte ciascuna: i tre byte più significativi indicano il **lotto di indirizzi acquisito dal costruttore della scheda**, detto anche **vendor code o OUI** (Organization Unique Identifier), mentre i tre meno significativi sono una numerazione **progressiva** decisa dal costruttore:



Esistono **tre tipologie di indirizzi MAC**:

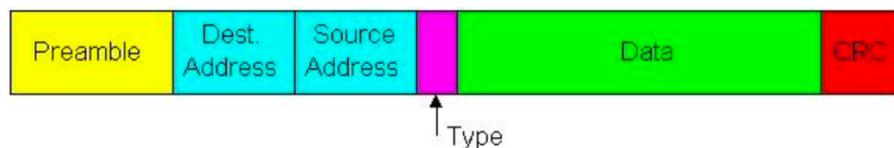
- **Single**, di una singola stazione;
- **Multicast**, di un gruppo di stazioni;
- **Broadcast**, di tutte le stazioni (ff-ff-ff-ff-ff-ff).

Ogni scheda di rete, quando **riceve un pacchetto, lo passa ai livelli superiori** nei seguenti casi:

- **Single**, se il DSAP è uguale a quello hardware della scheda (scritto in una ROM) o a quello caricato da software in un apposito buffer;
- **Multicast**, se ne è stata abilitata la ricezione via software;
- **Broadcast**, sempre.

Ethernet è la tecnologia dominante nel mondo delle LAN, essendo **economica** (20€ per 100Mbps), **la prima ampiamente diffusa, aggiornata** nel corso degli anni e **più semplice ed economica** rispetto alle LAN a token e ad ATM.

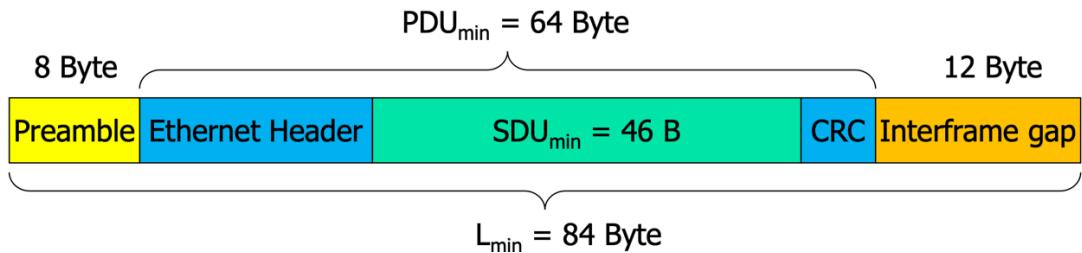
L'interfaccia di rete del mittente **incapsula i datagrammi IP** (o altri pacchetti di livello rete) **in frame Ethernet**:



Il **preamble** è costituito da 7 byte con una sequenza 10101010 seguiti da un byte (SFD) con la sequenza 10101011 ed è utilizzato per sincronizzare i **clock** del mittente e del destinatario. Il frame è ricevuto da tutti gli adattatori di rete presenti sulla LAN e scartata se l'indirizzo di destinazione, specificato insieme a quello del mittente in due appositi campi da 3 byte ciascuno, non coincide con quello della scheda stessa. Il campo **Type**, di 2 byte, indica il protocollo di livello rete sovrastante, principalmente IP, ma sono supportati anche altri protocolli (come Novell IPX e AppleTalk), mentre nel campo **CRC**, di 4 byte, è effettuato un controllo alla destinazione: se l'errore è rilevato, il frame viene scartato.

Non considerando il **preamble** (8 byte), l'**header** (14 byte), il **payload** (PDU_{min} , 46 byte), e il **CRC** (4 byte), la **minima dimensione di un frame Ethernet**, PDU_{min} è 64 byte, mentre la **spaziatura tra i frame** (detta inter-frame gap), corrispondente al tempo di trasmissione, è di 12 byte. Ciò significa che la **dimensione minima di un frame Ethernet**, L_{min} , è:

$$L_{min} = 8 \text{ byte} + 64 \text{ byte} + 12 \text{ byte} = 84 \text{ byte}$$

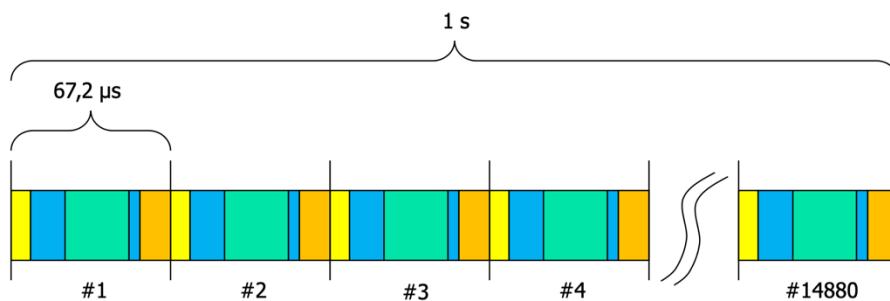


Se la velocità di trasmissione è di 10Mb/s, ovvero di 10^7 b/s, il tempo di trasmissione di un pacchetto di dimensione minima è:

$$\frac{84 \cdot 8}{10^7} s = 67 \mu s$$

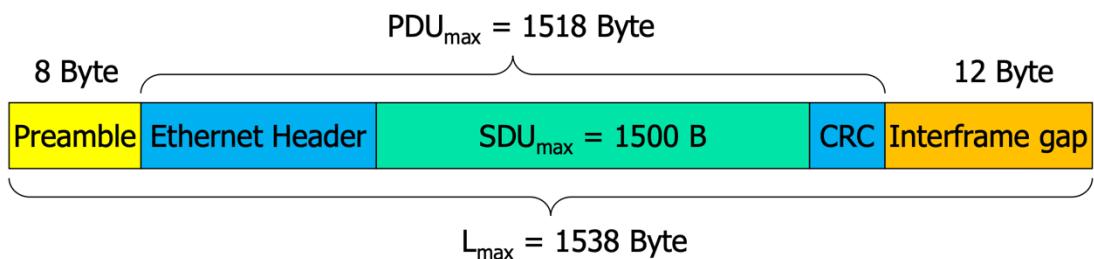
Il reciproco, invece, è il massimo numero di pacchetti di dimensione minima che è possibile trasmettere in un secondo:

$$\frac{1}{67,2 \cdot 10^{-6}} = 14880$$



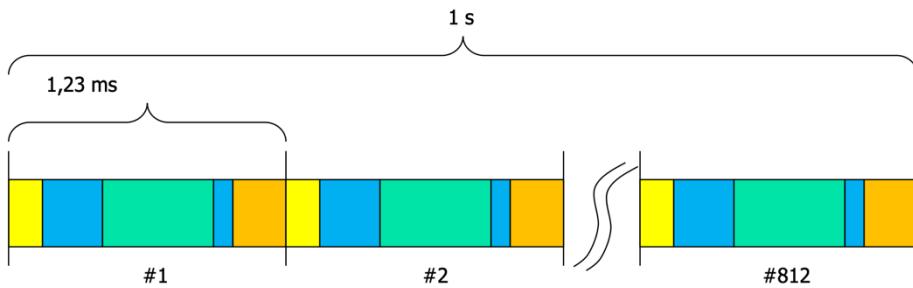
La massima dimensione di una frame Ethernet, PDU_{max} è pari a 1518 byte (con $SDU_{max} = 1500$ byte, MTU di un pacchetto IP su Ethernet) e, pertanto, la massima dimensione di una frame Ethernet trasmessa è:

$$8\text{byte} + 1518\text{ byte} + 12\text{byte} = 1538\text{ byte}$$

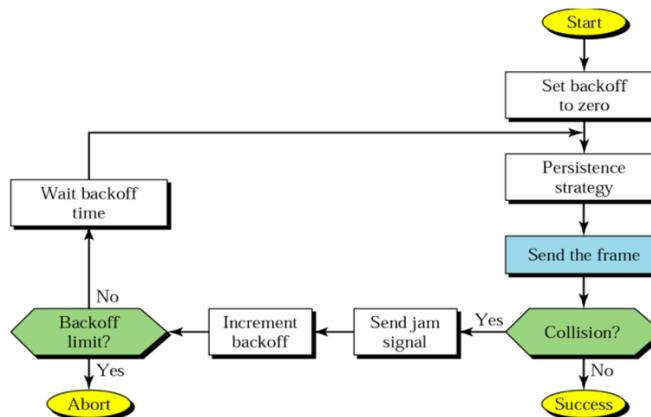


Alla velocità di trasmissione di 10Mb/s, assumendo che tutti i pacchetti non siano di dimensione minima, il massimo numero di pacchetti al secondo diminuisce; paradossalmente, è un bene: meno overhead e meno istruzioni per le stazioni. Il tempo di trasmissione di un pacchetto di dimensione massima è:

$$\frac{1538 \cdot 8}{10^7} s = 1.23ms$$

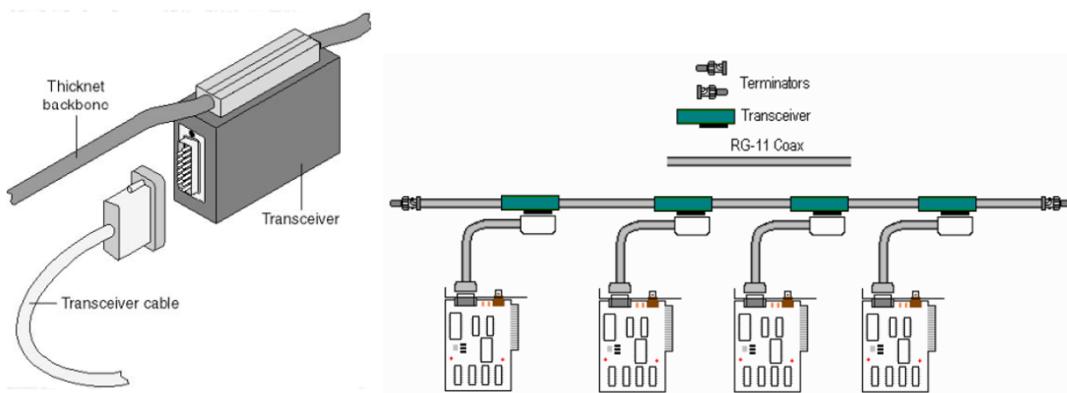


In Ethernet si usa il metodo CSMA non persistente per il controllo delle collisioni. In particolare, in Ethernet, prima di effettuare una ritrasmissione del proprio frame in seguito ad una collisione, ogni stazione effettua una pausa di un tempo casuale secondo un algoritmo detto di Exponential Backoff. Questo evita che si generi un deadlock per continue ritrasmissioni effettuate nello stesso tempo. Si fa inoltre uso di un Jam Signal che consente alle altre stazioni di accorgersi dell'avvenuta collisione, e che è inviato da una stazione che rileva la collisione alle restanti.

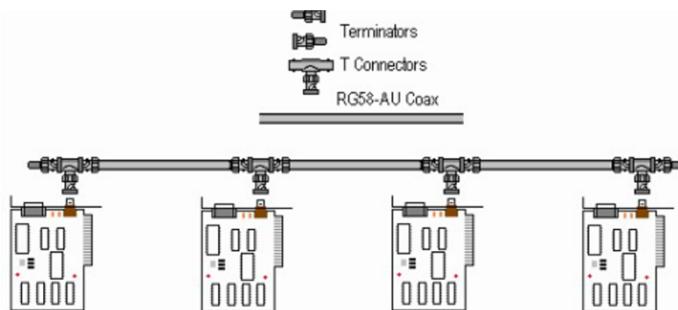


L'algoritmo di Exponential Backoff funziona nel modo seguente. Alla prima collisione sceglie un K tra 0 e 1. Il K scelto funge da multiplo per il ritardo di trasmissione, pari a $K \cdot 512$ bit. Se alla ritrasmissione si verifica una collisione per lo stesso frame, si sceglie un K tra 0, 1, 2, e 3. Dopo un certo $n < 10$ collisioni consecutive la scelta è tra 0 fino a $2^n - 1$. Dopo 10 o più collisioni il massimo valore è 1023. Dopo 16 collisioni tipicamente si termina la trasmissione.

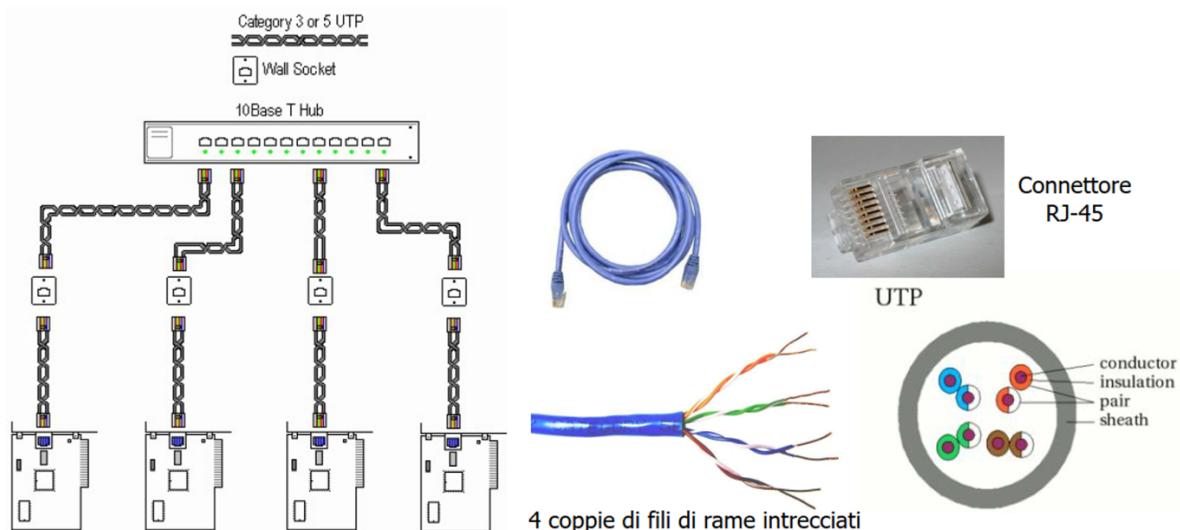
La prima versione di Ethernet è stata 10Base5. Il transceiver si occupa di convertire il segnale magnetico in uno elettrico; questo dispositivo, però, causava varie limitazioni, perché non si poteva spostare ed erano point of failure per tutta la rete. La struttura di una tale rete è mostrata di seguito.



Nella versione seguente, 10Base2, il cavo coassiale è meno spesso, e i transceiver erano presenti all'interno della scheda di rete. Il cavo non era unico, ma era un insieme di più spezzoni, che permetteva di aggiungere o rimuovere spezzoni intermedi o alle estremità. La struttura gestita è mostrata di seguito.



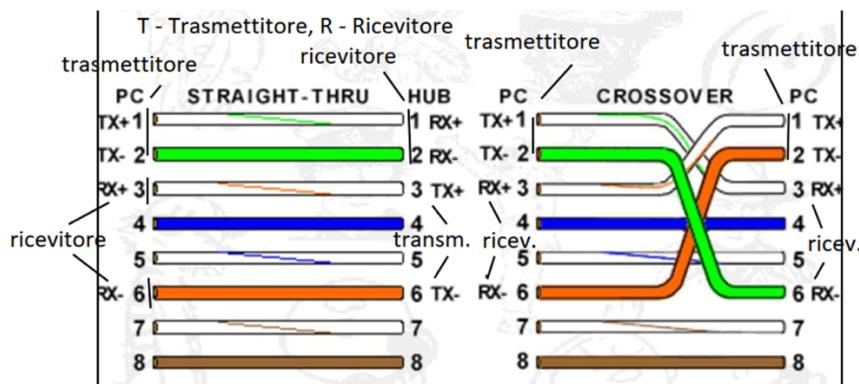
La vera rivoluzione è stata l'abbandono al cavo coassiale, per passare al doppino di rame, quello usato tutt'oggi. Il cablaggio non è più a bus, ma a stella, dove tutti i collegamenti si addensano su un unico dispositivo centrale, un hub. Un hub ripete il segnale ricevuto da ciascuna porta sulle restanti porte. Le versioni di Ethernet che introducono questa tecnologia sono 10BaseT e 100BaseT, con velocità rispettivamente di 10 Mbit/s e 100 Mbit/s.



Il fatto che ogni dispositivo sia agganciato all'hub permette all'hub di disconnettere le schede malfunzionanti, evitando che il rumore generato si propaghi alle altre schede. Si vedrà però successivamente che con il solo uso di hub la topologia virtuale è a bus nonostante quella fisica sia a stella; per creare una effettiva topologia a stella è necessario usare un bridge (o più realisticamente uno switch). Il doppino che si usa nelle reti Ethernet a stella è detto UTP, Unshielded Twisted Pair, dato da quattro coppie di cavi intrecciati. Di fatto, tutti i cavi UTP permettono una comunicazione full-duplex.

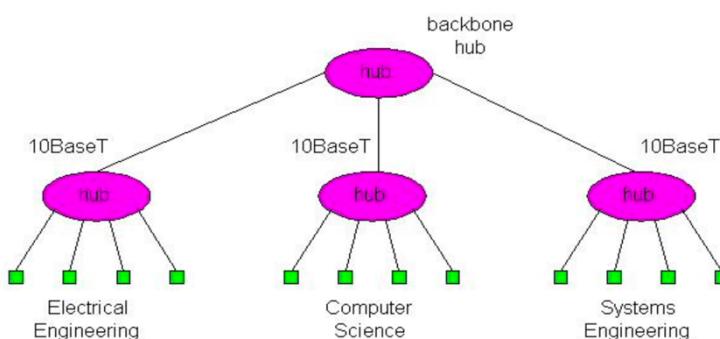
L'intreccio, l'abbinatura, serve per cercare di ridurre gli effetti di diafonia, di accoppiamento elettromagnetico del filo per induzione con altri campi elettromagnetici esterni, o anche tra i campi delle stesse coppie, che creano disturbo. Ciascuna coppia, infatti, è formata da cavi con stessa intensità ma polarità opposta. Tra cavi di categorie maggiori è principalmente migliorata l'abbinatura.

Un cavo UTP straight è tale se è presente la stessa disposizione dei cavi conduttori ad entrambe le estremità. Si usano per collegare end-system con hub/switch. Il cavo UTP cross si usa invece per collegare due end-system; si riconosce perché i colori sono disposti diversamente tra le due estremità. Questa differenza tra i tipi di cavi è dovuta al fatto che tra due end-system (due PC), la posizione di trasmettitore TX e ricevitore RX è la stessa (in figura, TX è sempre sopra, corrispondente ai cavi 1 e 2, e RX è corrispondente ai cavi 3 e 6). Un cavo deve collegare il trasmettitore con il ricevitore, ed il ricevitore con il trasmettitore; affinché ciò sia possibile per due end-system i cavi devono quindi incrociarsi. Si nota poi una doppia polarità sia per TX che per RX, presente per ridurre la diafonia, come già detto in precedenza. Solo 4 degli 8 cavi sono quindi usati per la trasmissione di dati, mentre gli altri 4 restano inutilizzati.



INTERCONNESSIONE DI LAN: HUB AND BRIDGE

L'interconnessione di LAN sorge dall'esigenza di dover collegare i computer di diversi uffici collocati nello stesso edificio (o comprensorio) e dall'intuizione di unirli in un'unica rete LAN. La prima soluzione prevede l'impiego di hub, la seconda di bridges (o switches). Gli hub sono dispositivi di livello fisico che agiscono come ripetitori di bit: riproducono i bit in ingresso ad un'interfaccia su tutte le altre interfacce, vantando di un'elevata semplicità e un basso costo. L'interconnessione di LAN mediante hub avviene attraverso una gerarchia multilivello con un backbone hub al livello più alto; ogni LAN collegata è detta “segmento di LAN”:

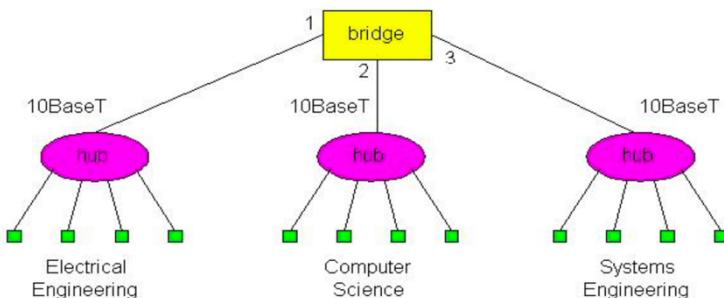


Questo approccio ha:

- **Vantaggi:**
 - L'organizzazione multilivello garantisce una parziale tolleranza ai guasti, porzioni di LAN continuano a funzionare in caso di guasto ad uno o più hub;
 - Si estende la massima distanza tra i nodi, 100 metri per ogni hub;

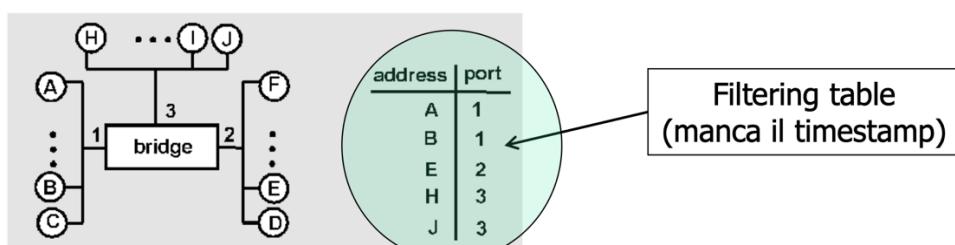
- **Svantaggi:**

- Gli hub non isolano i domini di collisione, le stazioni di un segmento possono subire una collisione per una trasmissione simultanea da parte di una qualunque stazione presente su un qualunque altro segmento;
- La creazione di un singolo dominio di collisione non comporta alcun aumento del throughput massimo, quello complessivo in una rete multilivello è lo stesso di una rete con un unico segmento;
- La realizzazione di un'unica rete LAN impone un limite al numero massimo di stazioni che è possibile collegare, nonché all'estensione geografica che è possibile raggiungere.



I bridge sono dispositivi utilizzati per collegare due o più LAN tra loro in maniera più efficiente dello schema precedente, essendo gli hub dispositivi hardware oggi raramente utilizzati (sono stati sostituiti dagli switch), occupandosi di filtrare i pacchetti in modo che se la destinazione è sullo stesso segmento di LAN del mittente, non è svolta alcuna azione da parte del bridge, altrimenti, se la destinazione è su un segmento di LAN differente da quello del mittente, inoltra il frame sul segmento di LAN corretto.

Ci si chiede, adesso, come faccia il bridge a sapere su quale segmento un frame va inoltrato; i bridge eseguono un algoritmo di apprendimento per scoprire a quali interfacce sono collegati gli host, salvando le informazioni in tabelle dette "filtering tables". Quando un frame è ricevuto, il bridge prende nota del segmento di LAN di provenienza nella filtering table e ne struttura una entry contenente il MAC address del nodo, il port del bridge e il Time Stamp; una entry viene, dopo un certo periodo di tempo, cancellata se non arrivano altre frame dall'host a cui si riferiscono.

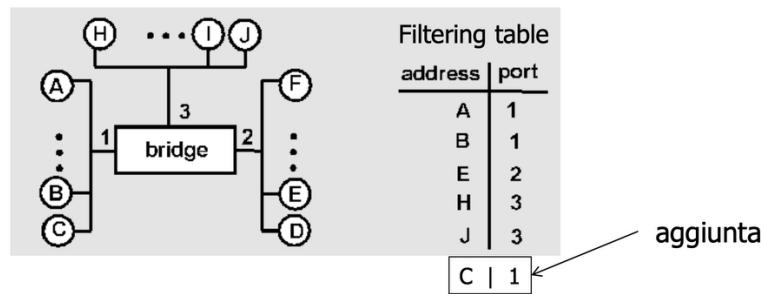


Un esempio in pseudo-codice di filtering procedure è il seguente:

```

if destination is on LAN on which frame was received
  then drop the frame
  else {
    lookup filtering table
    if entry found for destination
      then forward the frame on interface indicated;
      else flood; /* forward on all but the interface on
                     which the frame arrived */
  }
}

```



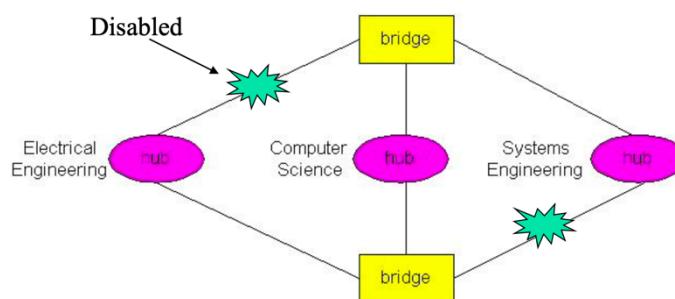
C invia un frame a D, con il bridge che non ha informazioni a riguardo; questo invia in flooding tramite le porte 2 e 3 e scrive la entry (C,1) nella filtering table, mentre il frame è ignorato nella LAN in alto. A questo punto, D risponde con un frame a C e il bridge scrive la entry (D, 2) nella filtering table; poiché, però, il bridge già conosce la posizione di C, inoltra il frame direttamente alla LAN corretta, evitando il flooding.

I bridge, a differenza degli hub, sono dispositivi di livello 2 e, pertanto, sono in grado di leggere le intestazioni di frame Ethernet, esaminare il loro contenuto e selezionare il link di uscita sulla base dell'indirizzo di destinazione. I bridge, inoltre, isolano i domini di collisione grazie alla loro capacità di porre i frame in un buffer (sono dispositivi store and forward); non appena un frame può essere inoltrato su un link di uscita, un bridge usa il protocollo CSMA/CD sul segmento LAN d'uscita prima di trasmettere.

I vantaggi dei bridge sono:

- Isolamento dei domini di collisione, determinando un aumento complessivo del throughput massimo;
- Non introduzione di alcuna limitazione sul numero massimo delle stazioni, né sull'estensione geografica;
- Possibilità di collegare differenti tecnologie, dal momento che sono dispositivi di tipo store and forward;
- Trasparenza, non richiedono alcuna modifica negli adattatori dei computer né configurazione da parte di un amministratore (sono plug and play).

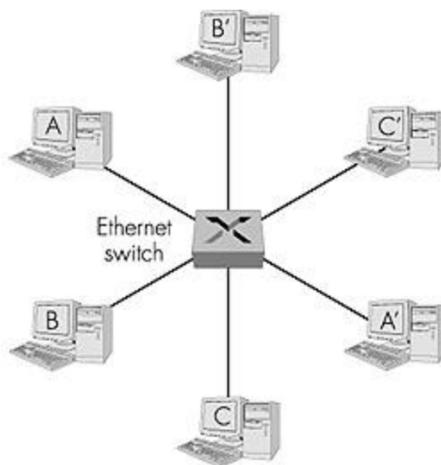
Per incrementare l'affidabilità può essere utile introdurre un certo grado di risonanza a causa di percorsi alternativi simultanei; in loro presenza, vengono create copie molteplici dei frame in loop. La soluzione a questo problema consiste nell'organizzare i bridge mediante uno spanning tree, disabilitando alcune interfacce:



Bridge e router sono entrambi dispositivi di tipo store and forward, i primi del Data Link e i secondi di livello rete, esaminando due diversi tipi di header. I router si basano sulle routing table ed implementano algoritmi di routing, mentre i bridge si basano sulle filtering table ed implementano algoritmi di filtering, learning e spanning tree.

Nei bridge, le operazioni sono più semplici e sono processate meno richieste ma le topologie sono limitate, è necessario uno spanning tree per prevenire i cicli, e non offrono alcuna protezione contro le tempeste broadcast (il broadcast ininterrotto generato da un host è normalmente inoltrato da un bridge). Nei router, possono essere realizzate differenti topologie, i loop sono limitati grazie al contatore TTL e all'impiego di buoni protocolli di routing, e può essere fornita una naturale protezione contro le tempeste broadcast ma richiedono una configurazione a livello IP, rendendoli non plug and play, e richiedono una capacità adeguata a processare una grande quantità di pacchetti. I bridge sono maggiormente utili in caso di reti piccole, con poche centinaia di host, mentre i router sono usati nelle grandi reti, migliaia di host.

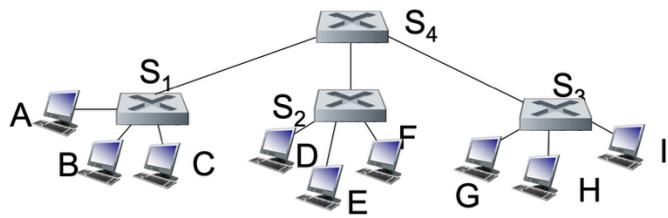
Uno switch Ethernet è un dispositivo il cui comportamento è simile a quello di un bridge, si presentano come box con molte porte di rete e fungono da centro stella del cablaggio in modo simile ad un hub. Tuttavia, a differenza di un hub, uno switch inoltra un frame solo verso la porta del destinatario, grazie ad un'azione di filtraggio simile a quella vista per i bridge. Il vantaggio dello switching consiste nella possibilità di trasmissioni contemporanee tra coppie di porte distinte senza collisioni simultaneamente (ad esempio, $A \rightarrow B$ e $A' \rightarrow B'$)



Mentre un bridge è strettamente sequenziale nella sua elaborazione, uno switch è in grado di effettuare più elaborazioni contemporaneamente. Inoltre, uno switch separa i domini di collisione, sebbene il comportamento di un hub possa tornare utile quando è necessario visualizzare tutto il traffico della rete (come quando si usa un IDS, l'equivalente di un antivirus per rilevare attacchi sulla rete). Gli switch moderni, quindi, permettono anche di configurare una o più porte di mirroring.

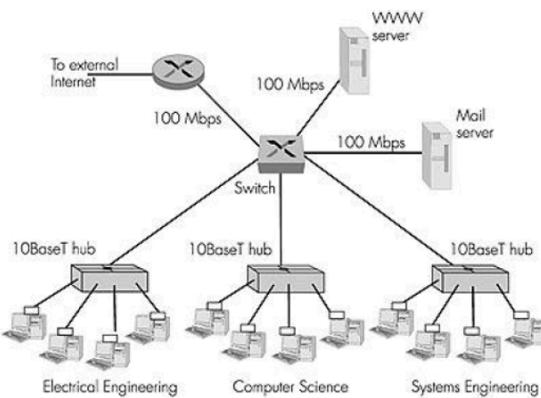
Uno switch è anche in grado di effettuare trasmissione e ricezione contemporaneamente; usando cavi UTP che consentono fisicamente questa duplice trasmissione, i collegamenti tra switch e host possono essere full-duplex (in una rete di questo tipo non sono presenti collisioni).

Per cut-through si intende la possibilità di inviare un frame ancora prima che sia completamente ricevuto, grazie al fatto che è sufficiente ricevere l'header del frame stesso per decidere la porta di uscita. Con l'uso di uno switch per gestire più hub, che collegano a loro volta molteplici host, si hanno più domini di collisioni separati, mentre con l'uso completo di switch non si hanno domini di collisione.

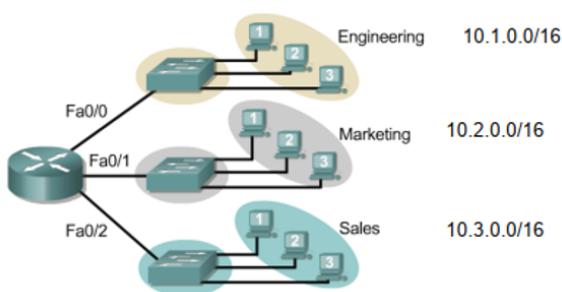


RETI VLAN E WLAN

Tradizionalmente, un insieme di LAN collegate ad Internet può essere strutturato come un insieme di più hub, ciascuno a cui sono collegati molteplici end-system sottostanti ad un unico switch, il quale è collegato con un router che permette la comunicazione con la rete Internet esterna. Ogni insieme di end-system collegati ad uno stesso hub costituisce una LAN differente grazie allo switch che separa i domini di collisione.

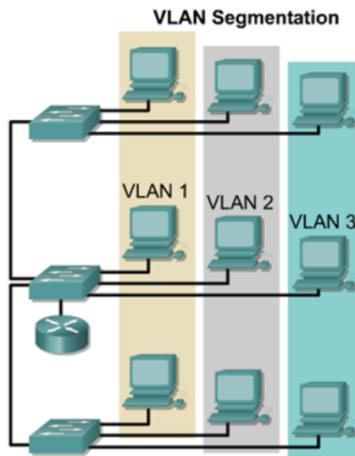


Questa topologia si può ottimizzare sostituendo gli hub con switch ed effettuando un collegamento diretto tra questi e il router, rimuovendo lo switch centrale. Si è visto, infatti, che un router può operare in maniera analoga ad uno switch interessando però anche il livello di rete, seppur sia vero che in molti router moderni vi sono effettivi switch integrati; in entrambi i casi, un partizionamento tradizionale della rete vede l'associazione una sottorete diversa per ogni LAN, quindi per ogni insieme di dispositivi collegati ad uno stesso hub/switch.



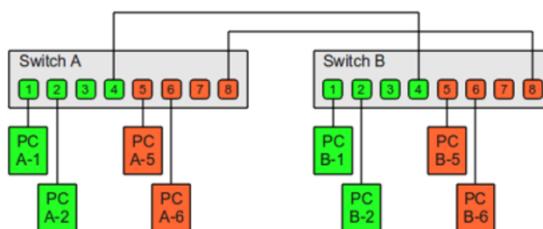
Questo tipo di partizionamento prende il nome di Partizionamento per LAN esistenti; si potrebbe, tuttavia, volere un partizionamento che non riflette la struttura fisica della rete, possibile solo introducendo l'uso delle VLAN. Una VLAN è un gruppo di porte di uno stesso switch, tra cui è possibile effettuare una comunicazione diretta e l'inoltro di traffico broadcast, mentre la comunicazione tra diverse VLAN è possibile solo attraverso il routing (funzioni L3).

Mediante VLAN è possibile utilizzare un unico switch con N VLAN anziché N switch differenti, semplificando la topologia della rete e permettendo di separare il partizionamento della rete dalla sua struttura fisica. In questa topologia, la divisione della rete nelle varie VLAN spetta allo switch, ed avviene al livello Data Link.

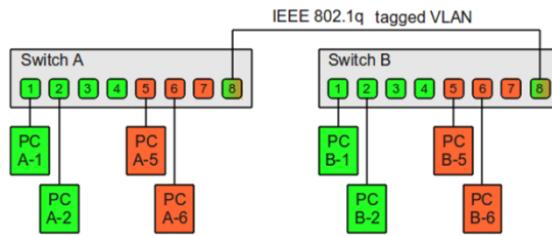


Questa topologia può essere ulteriormente semplificata collegando tutti i dispositivi ad un singolo router, dal momento in cui, come già anticipato, è possibile rimuovere lo switch esterno in modo che sia il router, operando sia a livello data link che a livello di rete, ad occuparsi di dividere la rete nelle varie VLAN.

È possibile che una stessa VLAN sia formata da più gruppi di porte su switch differenti. Si consideri, ad esempio, una coppia di stesse VLAN presenti su due switch. La comunicazione tra le stesse VLAN di switch differenti si può effettuare, nel caso più semplice, collegando una porta i di uno switch alla rispettiva porta i dello stesso switch. Occorrono quindi tanti collegamenti tra gli switch quante sono le VLAN, cosa che non è comoda per numerose VLAN.

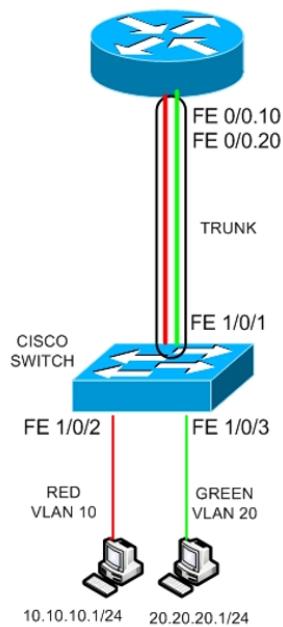


Una alternativa è consentita dal cosiddetto **trunking**, ossia **una singola connessione stabilita tra switch diversi e in cui il traffico si aggrega**. La distinzione tra i pacchetti di ciascuna VLAN non si può effettuare con la versione Ethernet originale, per questo si è necessitato di una modifica detta **VLAN Tagging**, che introduce per ogni frame un'etichetta che indica a che VLAN appartiene. Questa etichetta è rimossa quando il frame è passata ad altri livelli.

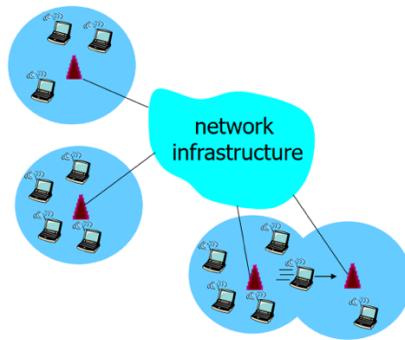


Di fatto, il VLAN Tagging si implementa facendo in modo che ogni switch utilizzi una tabella di filtering per ogni VLAN che supporta, ciascuna delle quali interessa un certo insieme di porte. Se una porta è presente esclusivamente in una tabella di filtering, allora quella porta è associata staticamente ad una VLAN, altrimenti se è presente in molteplici tabelle di filtering, su quella porta può passare il traffico di più VLAN differenti.

Si consideri uno switch collegato ad un router. Se lo switch gestisce VLAN differenti, il traffico di ciascuna VLAN non può essere inviato al router così com'è, perché ciascun traffico dovrebbe essere logicamente distinto. È però possibile fare in modo che una stessa interfaccia di rete di un router possa essere vista come due o più interfacce logiche distinte, permettendo l'utilizzo di VLAN differenti.



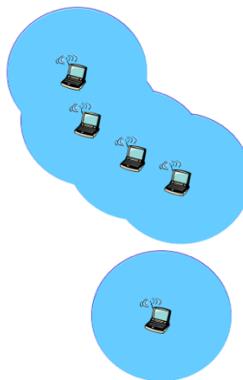
Le reti wireless sono reti che fanno uso di onde elettromagnetiche per trasformare informazioni, o in maniera diretta tra end-system o tramite dispositivi intermedi. Lo spettro elettromagnetico è una risorsa scarsa e quindi il suo uso è regolamentato. Le frequenze delle onde elettromagnetiche adibite all'uso da parte di reti LAN wireless sono dette **bande ISM**, Industrial, Scientific and Medical. L'uso è perlopiù libero, vincolato eventualmente da regolamentazioni nazionali sui limiti massimi di potenza, a differenza dello spettro restante il cui uso richiede specifiche autorizzazioni. Vi sono tre bande ISM: una intorno ai **900MHz**, relativamente stretta, una intorno ai **2.4GHz**, ed una intorno ai **5GHz**, queste ultime utilizzate dallo standard **IEEE 802.11**. In Europa la banda **900 MHz** è usata per il **GSM** e quindi non è usabile senza licenza, a differenza delle restanti due.



Una rete wireless è strutturata come un insieme di terminali con supporto a connettività radio (non essendo vincolati dal cablaggio, possono essere anche mobile), **collegati a elementi infrastrutturali centrali detti genericamente stazioni di base, o access point, che offrono connettività wireless ad un certo numero di terminali che rientrano nel loro raggio di azione**, area detta **Basic Service Set, BSS o “cella”**. **Ogni stazione di base è collegata ad una infrastruttura di rete cablata che funge da backbone**.

In una rete wireless non si verifica collegamento diretto tra i terminali, bensì ogni collegamento è mediato dalla stazione di base a cui sono collegati e lo stesso avviene per collegamenti tra terminali e l'infrastruttura di rete cablata.

Quando un terminale mobile è spostato ad una posizione più prossima ad un'altra stazione di base rispetto a quella a cui è collegato, può verificarsi un **evento di handoff**, se la rete è in grado di accorgersi di questo cambiamento, e **modificare di conseguenza la stazione di base a cui far collegare il terminale**.



In una rete wireless in modalità ad hoc non ci sono stazioni base o infrastrutture bensì la rete è costituita dai soli terminali. È con la collaborazione di più terminali che è possibile effettuare il routing interno alla rete ma questo può portare a casistiche in cui un terminale troppo distante dai restanti non ha modo di scambiare informazioni nella rete.

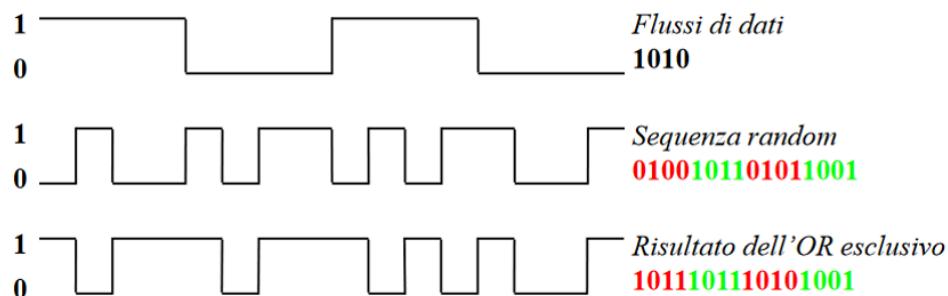
Le reti wireless si possono classificare secondo due criteri distinti: la presenza o meno di infrastruttura e se la consegna di pacchetti deve avvenire con un solo hop o può avvenire con molteplici hop. Reti a singolo hop con infrastruttura sono quelle di uso comune, mentre quelle senza infrastruttura corrispondono alle cosiddette PAN, Personal Area Network, che non permettono collegamento ad Internet. Reti ad hop multipli si basano sulla possibilità delle stazioni di base di comunicare tra loro in maniera wireless, eventualmente rendendo la backbone anch'essa wireless, mentre senza infrastruttura si ottengono reti wireless ad hoc.

Le reti LAN wireless sono tipicamente basate per l'impiego in aree geografiche limitate, e per l'uso di frequenze in bande non licenziate, quindi bande ISM. Le due principali tecniche usate a livello fisico sono 802.11, anche detta WiFi, e Bluetooth, entrambe che usano la tecnica “spread spectrum”, cioè la realizzazione della trasmissione su intervalli di frequenza ampi, in modo da minimizzare l'interferenza da parte di altri dispositivi. Bluetooth fa uso di una tecnica basata su salto di frequenza, dividendo la banda in un numero molto maggiore di canali, facendo in modo che ciascuno sia meno ampio.

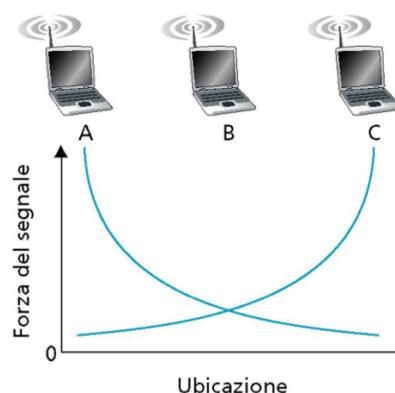
Wi-Fi utilizza la tecnica a sequenza diretta (DSSS) con 11 canali parzialmente sovrapposti; quando c'è molta interferenza, il sistema può cambiare canale. Si basa sul CDMA, dove ogni bit inviato è il risultato dello XOR tra un bit del messaggio e 11 bit di una sequenza pseudo-casuale (chipping). Il ricevente decodifica effettuando un prodotto scalare tra la sequenza ricevuta e quella conosciuta; questa sequenza è condivisa tra mittente e ricevente per garantire la codifica e decodifica.

Per connettersi a Internet, un terminale deve associarsi a un Access Point (AP). Questo avviene tramite:

1. **Scansione passiva:** Gli AP inviano periodicamente beacon per segnalarsi, e i terminali rispondono con una richiesta di connessione.
2. **Scansione attiva:** I terminali inviano una richiesta di beacon in broadcast, a cui gli AP rispondono, consentendo la connessione.

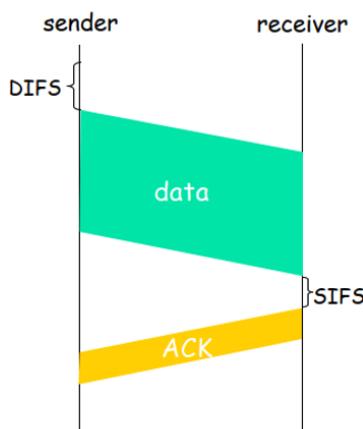


A differenza di connessioni cablate, le connessioni wireless sono più soggette ad attenuazione (fading), interferenza e si possono propagare in cammini molteplici per via della riflessione (multipath propagation), arrivando, quindi, a destinazione in tempi differenti. Si potrebbe pensare di utilizzare il protocollo ad accesso multiplo CSMA/CD; a differenza, però, di connessioni cablate come Ethernet, dove anche si usa il CSMA/CD, in 802.11, e in generale nelle connessioni wireless, risultano problemi ulteriori per la natura della trasmissione wireless stessa.



Nel caso in cui **due terminali sono sufficientemente lontani**, oppure siano separati da un ostacolo fisico, per via della rapida attenuazione, può capitare che i terminali generino segnali che **collidono in una certa zona**, ma ciascun terminale non riesce a rilevare la presenza di un altro segnale oltre al proprio per la bassa intensità con cui gli arriva. Questo fenomeno prende il nome di **problema del terminale nascosto**. Un nodo intermedio a due terminali in questa situazione **non è in grado di distinguere i due segnali in collisione**. Altro problema, detto **del terminale esposto**, è quello per cui **un terminale troppo vicino ad un altro che sta trasmettendo dati in una certa direzione non può effettuare trasmissioni neanche se il nodo a cui vuole trasmettere è in direzione opposta**.

Per evitare i problemi che derivano dall'uso del CSMA/CD con le connessioni wireless, si sceglie in 802.11 di non effettuare proprio la rilevazione delle collisioni; come alternativa, si cerca di evitare attivamente le collisioni, cioè sia adopera una Collision Avoidance, per cui il protocollo prende il nome di CSMA/CA. Un primo accorgimento che il protocollo implementa è l'**uso di un frame di riscontro**, nel seguente modo:



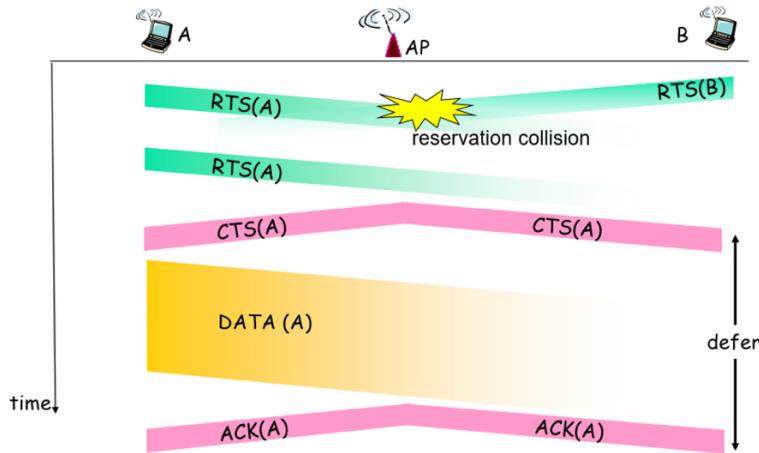
Il mittente verifica se il canale è inattivo e, in caso affermativo, attende un intervallo di tempo detto **DIFS**, dopo il quale trasmette un intero frame, altrimenti attende un tempo di backoff casuale che è decrementato solo mentre il canale è inattivo e dopo il quale invia un frame; tuttavia, se dopo ciò non è ricevuto un ACK, viene incrementato l'intervallo di backoff e si ripetono i passaggi. Il destinatario, invece, alla ricezione di un frame attende un intervallo di tempo detto **SIFS**, dopo il quale risponde con un ACK. I valori dei due intervalli di tempo sono scelti opportunamente, con il vincolo che **SIFS < DIFS**, in modo da **assegnare più priorità di accesso agli ACK rispetto ai frame con dati**.

Si ricorda che in reti wireless con infrastruttura i ruoli di mittente e ricevitore sono svolti da host e da AP in maniera alternata, e non da due host come nella modalità ad hoc.

Di base un nodo può sapere se il canale è inattivo o meno ascoltando sul canale ma, ulteriormente, ogni frame inviato da un mittente contiene nell'header anche la durata della trasmissione dei dati, in modo che i restanti nodi possano sapere con certezza che il canale sarà occupato per la durata specificata.

La tecnica vista, però, non è completamente sufficiente, si possono verificare collisioni su frame dati. Una soluzione aggiuntiva alla precedente, che risolve del tutto le collisioni, si basa sulla **possibilità da parte del mittente di prenotare il canale utilizzando una piccola frame RTS, request to send, inviata in CSMA, a cui il ricevente risponde con un frame CTS, clear to send, inviata in broadcast in modo che tutti i nodi possano sapere dell'avvenuta prenotazione e che il**

nodo mittente possa iniziare a inviare i dati in maniera esclusiva. Si nota che i frame RTS, inviati in CSMA, possono collidere ma l'effetto di una collisione su frame così piccoli è trascurabile.



Un frame 802.11 (usato nelle reti Wi-Fi) ha quattro indirizzi MAC, a differenza di un frame Ethernet che ne ha solo due. Questi indirizzi servono per gestire la comunicazione tra dispositivi wireless e cablati in diverse configurazioni:

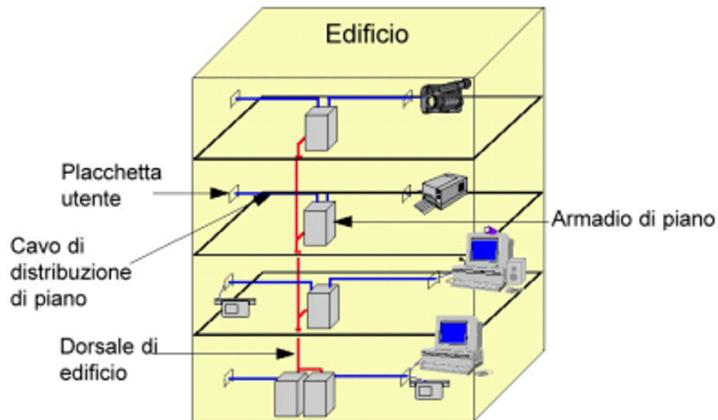
1. **Infrastruttura con un solo AP:**
 - **Primo indirizzo**, dispositivo destinatario;
 - **Secondo indirizzo**, dispositivo sorgente;
 - **Terzo indirizzo**, interfaccia del router a cui l'AP è collegato;
2. **Modalità ad hoc (senza AP):**
 - Il **quarto indirizzo** è l'AP virtuale creato per gestire la rete;
3. **Infrastruttura con più AP (reti aziendali):**
 - **Primo e secondo indirizzo**, host sorgente e destinazione;
 - **Terzo e quarto indirizzo**, gli AP collegati agli host sorgente e destinazione.

frame control	duration	address 1	address 2	address 3	seq control	address 4	payload	CRC
2	2	6	6	6	2	6	0 - 2312	4

In una rete domestica, la configurazione più comune è un AP connesso a un router; ad esempio:

- Quando un host invia dati all'AP, il frame Wi-Fi include gli indirizzi MAC di:
 1. AP;
 2. Host sorgente;
 3. Interfaccia del router;
- Quando l'AP inoltra i dati al router, il frame Ethernet ha solo due indirizzi MAC:
 1. Router;
 2. Host sorgente.

La gestione di questi indirizzi avviene tramite i flag to AP e from AP nel frame Wi-Fi, specificando la direzione dei dati.



Il cablaggio delle reti negli edifici segue standard che regolano una struttura gerarchica a tre livelli:

1. **Livello piano:** Ogni piano ha un centro stella (armadio di piano) dove convergono i cavi dei dispositivi tramite un cablaggio orizzontale.
 - I dispositivi si collegano a prese utente e da lì i cavi arrivano all'armadio, passando per un patch panel, dove terminano i cavi orizzontali;
 - I patch cord collegano il patch panel agli switch;
2. **Livello edificio:** Gli armadi di piano di un edificio sono collegati al centro stella dell'edificio tramite un cablaggio verticale (intra-building backbone);
3. **Livello campus:** I centri stella degli edifici sono collegati tra loro tramite un cablaggio tra edifici (inter-building backbone).

I tipi di cavi, invece, sono:

- **Cablaggio orizzontale:** Generalmente con cavi UTP, e non supera i 90 metri;
- **Cablaggi verticali e tra edifici,** solitamente in fibra ottica, che può essere:
 - Multimodale (MMF), economica ma con un raggio d'azione limitato;
 - Monomodale (SMF), più costosa ma ideale per lunghe distanze (chilometri).

In contesti locali si preferiscono le fibre multimodali, mentre le monomodali sono usate per collegamenti più estesi.

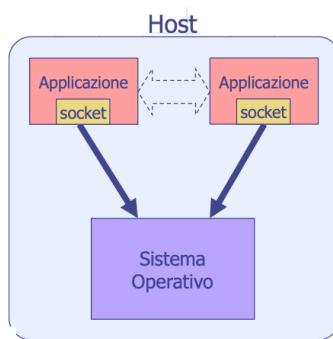
APPLICAZIONI CLIENT-SERVER E LA SICUREZZA IN RETE

INTRODUZIONE ALLA COMUNICAZIONE CLIENT-SERVER CON API

Per socket si intende un'astrazione di canale di comunicazione tra processi, locali o remoti, e attraverso cui un'applicazione può ricevere o trasmettere dati; i meccanismi restano (quasi) indipendenti dal supporto fisico su cui le informazioni viaggiano, permettendo l'esistenza dell'astrazione stessa. Inizialmente i socket nascono in ambiente UNIX, negli anni 80 la Advanced Research Project Agency finanziò l'Università di Berkley per implementare la suite TCP/IP nel sistema operativo in questione (nella versione 4.1cBSD) ma finirono per sviluppare il set originario di funzioni che fu chiamato interfaccia socket.

I socket si presentano sotto forma di un'API, cioè di un insieme di funzioni che le applicazioni possono invocare per ricevere il servizio desiderato e rappresentano un'estensione delle API di UNIX per la gestione dell'I/O su periferica standard (files su disco, stampanti, ecc...); l'API per i socket è poi divenuta uno standard de facto ed è oggi diffusa nell'ambito di tutti i maggiori Sistemi Operativi (Linux, FreeBSD, Solaris, Windows, ecc...).

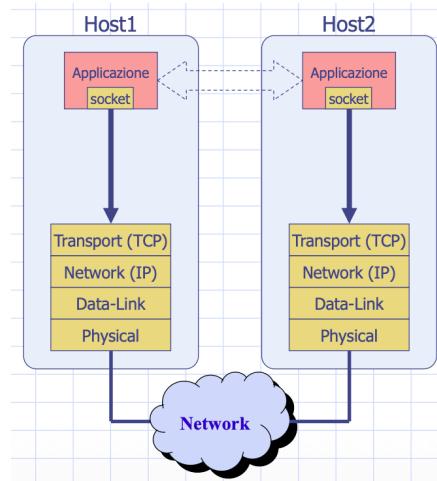
L'interazione tra l'Applicazione e il Sistema Operativo avviene con la prima che chiede al secondo di utilizzare i servizi di rete, il Sistema Operativo crea una socket e la restituisce all'Applicazione (in particolare, restituisce il suo descrittore), la quale prima la utilizza (tramite operazioni di Open, Read, Write e Close) e poi la chiude restituendola al Sistema Operativo. Due applicazioni in esecuzione sulla stessa macchina scambiano dati (comunicazione locale) utilizzando l'interfaccia delle socket ed un meccanismo di Interprocess Communication (IPC):



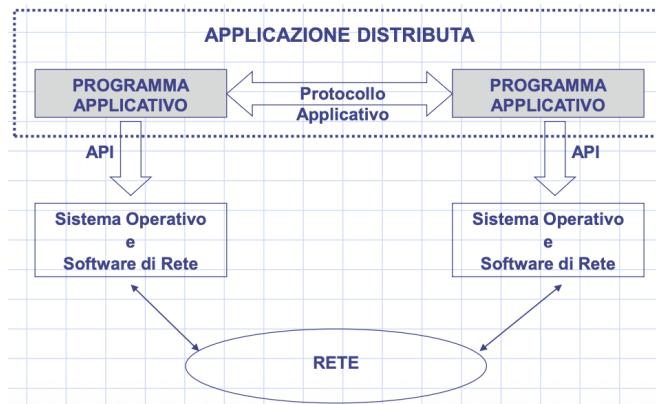
Esistono due tipologie di socket per una comunicazione locale:

- **Socket Unix-domain (PF_UNIX);**
- **Socket TCP/IP (PF_INET).**

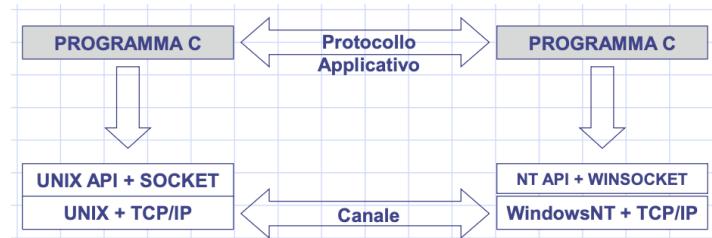
Per le comunicazioni remote si utilizza solo quest'ultimo tipo di socket, end-to-end realizzata tramite un protocollo di trasporto (TCP o UDP).



Un esempio di applicazione distribuita, invece:



La **configurazione di riferimento di un'applicazione distribuita basata su TCP/IP e socket** è la seguente:



Il paradigma di riferimento quando si parla di comunicazione tramite socket è il paradigma del client-server: l'entità che prende l'iniziativa di comunicare è detta **client**, conosce l'indirizzo del server e gli chiede un determinato servizio tramite una richiesta, mentre l'entità che **risponde a tali richieste è il server**, il quale deve aver divulgato il proprio indirizzo e che rimane in attesa di richieste di servizio.

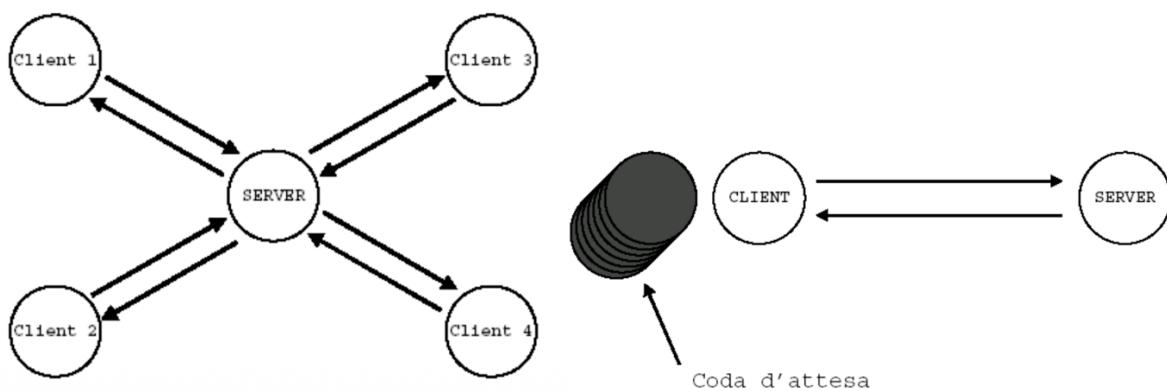
Un terminale di comunicazione (end-point) è identificato univocamente dalla coppia (indirizzo, processo), in modo che una comunicazione end-to-end sia identificata attraverso la quintupla:

{protocol, local-address, local-process, foreign-address, foreign-process}

Per una **comunicazione basata sui protocolli TCP/IP**:

- Protocol è TCP o UDP;
- Local-address e foreign-address sono indirizzi IP;
- Local-process e foreign-process sono numeri di port TCP o UDP.

Un server può ricevere chiamate anche da più client diversi e ognuna di queste comunicazioni richiederà un certo tempo prima di potersi considerare conclusa. **Un server che accetta più comunicazioni simultanee** (ovvero che quando arriva una chiamata, mentre è già impegnato in una comunicazione, non la rifiuta ma la gestisce) è **definito concorrente**, mentre **se accetta una sola comunicazione alla volta è detto iterativo**; in quest'ultimo caso, **una richiesta può essere servita solo quando la precedente si è già conclusa** (paradigma applicato nel modello di comunicazione telefonica di base).



Un'ulteriore tipo di comunicazione prende il nome di “**Connection Oriented**” e presuppone che i due end-point dispongano di un canale di comunicazione che trasporti i flussi, sia affidabile, sia dedicato e che preservi l’ordine delle informazioni. Il canale si comporta, cioè, come una sorta di “tubo”: tutto ciò che viene inserito al suo interno arriverà inalterato dall’altro lato e nello stesso ordine con cui è stato immesso (non è detto, però, che vengano mantenuti i limiti dei messaggi). La comunicazione telefonica è molto simile ad un tipo di comunicazione Connection Oriented.

Il paradigma di comunicazione **Datagram** (anche detta **connectionless**) presuppone che il canale trasporti i messaggi non sia affidabile, sia condiviso e non preservi l’ordine delle informazioni. Se si inviano dieci messaggi dall’altro lato, essi possono anche arrivare mescolati tra di loro e tra i messaggi appartenenti ad altre comunicazioni (i limiti dei messaggi, però, vengono comunque preservati). La posta ordinaria è molto simile ad un tipo di comunicazione Datagram.

Nel momento in cui si crea una socket, è necessario specificare la modalità di comunicazione che si desidera utilizzare: **byte-stream**, i dati vengono trasferiti come una sequenza ordinata di byte e consegnati in modo affidabile (**SOCK_STREAM**), o **datagram**, i dati vengono trasferiti come messaggi indipendenti senza garanzie di consegna (**SOCK_DGRAM**). Quando la socket è associata allo stack TCP/IP, la modalità byte-stream implica l’impiego del protocollo TCP, mentre la modalità datagram implica il protocollo UDP.

Per **naming/binding** si intende l’operazione con cui, ad una socket già creata, è associato un endpoint di trasporto, specificando local-address e local-process.

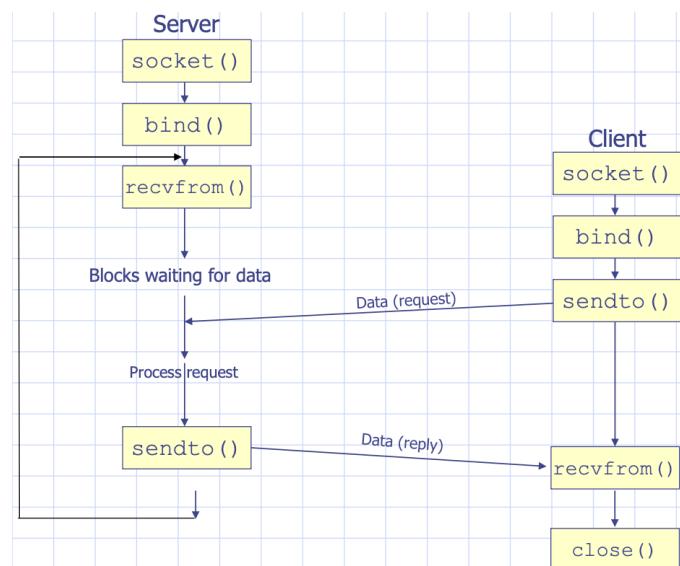
Per la progettazione di un server TCP, bisogna creare un endpoint (tramite richiesta al Sistema Operativo), instaurare un collegamento dell’endpoint ad un port (un processo sospeso in attesa svolge il ruolo di ascolto sul port) e permettere l’accettazione della richiesta di un client, con conseguenti letture e scritture sulla connessione, per poi chiuderla.

Per la progettazione di un client TCP, bisogna creare un endpoint (tramite richiesta al Sistema Operativo), creare una connessione (implementando open di TCP, 3-way handshake) e implementare operazioni di lettura e scrittura sulla connessione (analogo ad operazioni su file in Unix), per poi chiuderla (implementando close di TCP, 4-way handshake).

Per la progettazione di un server UDP, bisogna creare un endpoint (tramite richiesta al Sistema Operativo), collegare l'endpoint ad un port (open passiva in attesa di ricevere datagram) e implementare la ricezione e l'invio di datagram e la chiusura dell'endpoint.

Per la **progettazione di un client UDP**, bisogna creare un **endpoint** (tramite richiesta al Sistema Operativo) e **implementare la ricezione e l'invio di datagram e la chiusura dell'endpoint**.

Le primitive per la comunicazione Datagram sono le seguenti:

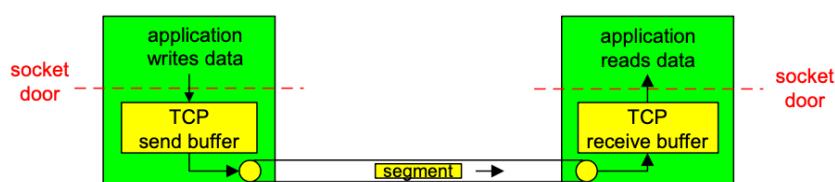


COMUNICAZIONE CLIENT-SERVER CON TCP E UDP IN PYTHON

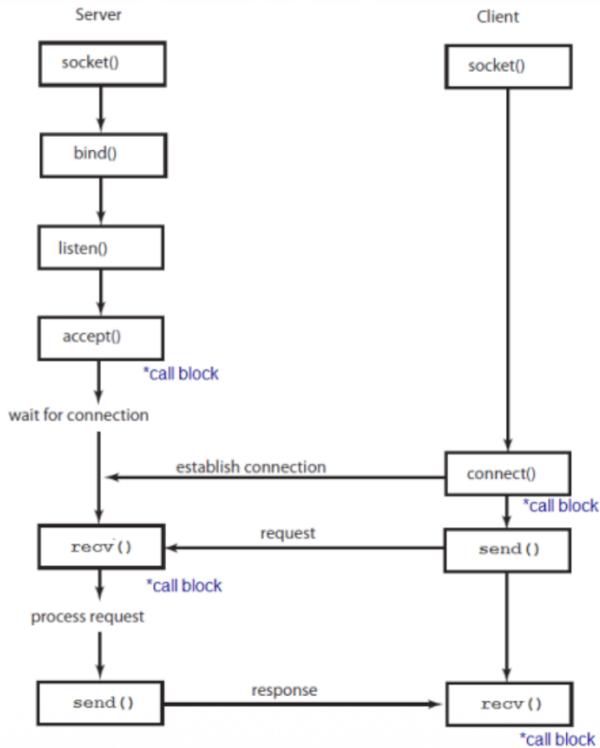
La comunicazione con TCP è connection-oriented, una volta stabilita la connessione tra due endpoint TCP realizza il trasferimento affidabile di stream di byte in entrambi i versi; il processo che prende l'iniziativa di stabilire la connessione è il client, mentre l'altro è il server. Ciascun datagramma IP è associato alla connessione TCP da:

- Indirizzo IP dell'host mittente;
 - Indirizzo IP dell'host destinazione;
 - Port number sorgente scelto dal mittente;
 - Port number destinazione sul quale il processo destinatario è in ricezione.

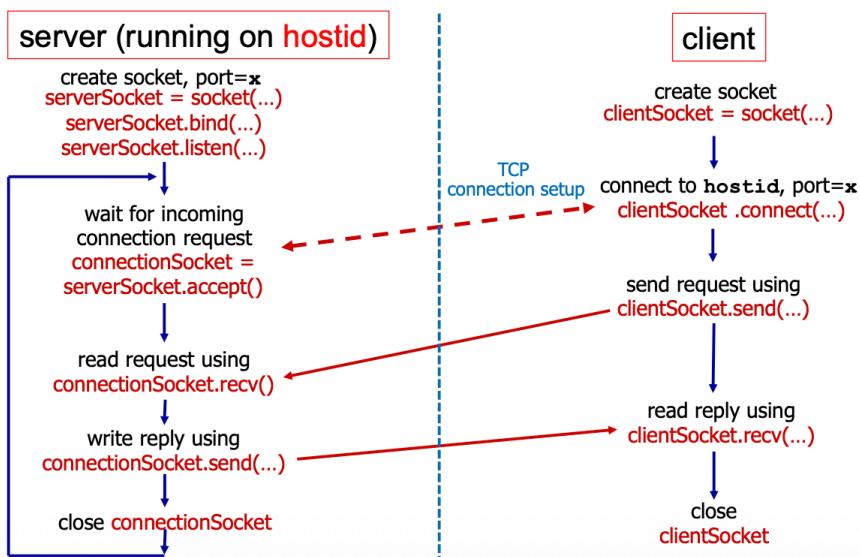
In momenti diversi, sia il client che il server possono inviare dati all'altra parte, il pattern della comunicazione dipende dal protocollo di livelli applicativo.



Client e server creano un oggetto istanza della classe Socket, passando come parametri le costanti predefinite `socket.AF_INET` e `socket.SOCK_STREAM` (per IPv6 si usa come primo parametro il valore `socket.AF_INET6`). Successivamente, **sugli oggetti socket si invocano dei metodi secondo uno schema differente per le due entità**; le funzioni `connect()`, `accept()` e `recv()` sono bloccanti e all'uscita delle prime due la connessione è stata stabilita (3-way handshake).



Lo schema di un sistema client-server con TCP è il seguente:



In Python, il metodo `send(bytes_to_send)` accetta come parametro un oggetto di tipo `bytes` e restituisce come valore di ritorno il numero di byte effettivamente trasferiti: se il numero in questione è inferiore alla dimensione dell'argomento, l'applicazione deve effettuare un'ulteriore trasmissione dei bytes residui; per inviare una stringa di testo occorre dapprima trasformare la

stringa in un oggetto di tipo bytes (con `encode('ascii')`) e poi **effettuare la decodifica in ricezione** (con `decode('ascii')`).

Il metodo **`sendall(bytes_to_send)`** accetta come parametro un oggetto di tipo bytes ma è bloccante: si esce dalla funzione solo quando tutti i byte sono stati effettivamente trasferiti alla controparte; per inviare una stringa di testo, occorre effettuare gli stessi passaggi specificati per `send()`.

Un TCP client in Python 3 può essere implementato come segue:

```
import sys, getopt, socket
# ... def usage():
# ... def read_args():
SERVER_ADDRESS = "127.0.0.1"
SERVER_PORT = 12000
# read command line arguments
read_args()
# get user keyboard input
sentence = input("Input lowercase sentence: ")
# create TCP socket
clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# connect socket to remote server
clientSocket.connect((SERVER_ADDRESS, SERVER_PORT))
print("Connected to server at (%s, %d)" % (SERVER_ADDRESS, SERVER_PORT))
# Send sentence into socket, no need to specify server IP and port
request = sentence.encode()
clientSocket.send(request)
# read reply message from socket into response
response = clientSocket.recv(1024)
# print out received response
print("Received reply <%s>" % response.decode())
# close socket
clientSocket.close()
print("Connection closed with server at (%s, %d)" % (SERVER_ADDRESS, SERVER_PORT))

# ...

def usage():
    print("Usage: %s [-a X.X.X.X] [-p N]" % sys.argv[0])
    print("Arguments:")
    print("-a X.X.X.X --> server listening address (default 127.0.0.1)")
    print("-p N --> server listening port number in 1024..65535 (default 12000)")
    print("Flags:")
    print("-h --> help")

# ...
```

```
# ...
def read_args():
    global SERVER_ADDRESS, SERVER_PORT
    try:
        opts, args = getopt.getopt(sys.argv[1:], 'ha:p:', ['help', 'address=', 'port='])
    except getopt.GetoptError as err:
        print("ERROR: ", err)
        usage(); sys.exit()

    for opt, arg in opts:
        if opt in ('-h', '--help'):
            usage(); sys.exit()
        elif opt in ('-a', '--address'):
            SERVER_ADDRESS = arg
        elif opt in ('-p', '--port'):
            if (arg.isnumeric()) and (int(arg) >= 1024) and (int(arg) <= 65535):
                SERVER_PORT = int(arg)
            else:
                print("ERROR: -p option must be an integer in 1024..65535")
                usage(); sys.exit()
        else:
            print("Unrecognized argument !")
            usage(); sys.exit()

# ...
```

Mentre un server:

```
import sys, getopt, socket
# ... def usage():
# ... def read_args():
SERVER_ADDRESS = "0.0.0.0"
SERVER_PORT = 12000
# read command line arguments
read_args()
# create TCP socket
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serverSocket.bind((SERVER_ADDRESS, SERVER_PORT)) # bind socket to local port
serverSocket.listen(1) # put socket in passive mode
while True:
    # wait for incoming connections on accept(), new socket created on return
    print("Server waiting on (%s, %d)" % (SERVER_ADDRESS, SERVER_PORT))
    connectionSocket, clientAddress = serverSocket.accept()
    # receive request on newly established connectionSocket
    request = connectionSocket.recv(1024)
    print("Received request from (%s, %d)" % (clientAddress[0], clientAddress[1]))
    # convert message to upper case
    response = request.upper()
    # send back modified string to client
    connectionSocket.send(response)
    print("Sent reply to (%s, %d)" % (clientAddress[0], clientAddress[1]))
    connectionSocket.close()
```

Un HTTP client in Python 3 può essere implementato come segue:

```
import socket
from urllib.parse import urlparse

while True:
    url = input("Type the object's URL: ")
    if (url == ""):
        break
    elif (url == "*"):
        url = "http://127.0.0.1:8080/"

    o = urlparse(url)
    if (o.hostname != None):
        SERVER_ADDRESS = o.hostname
    else:
        print("Error in parsing URL <%s>. Retry." % url); continue

    if (o.port != None):
        SERVER_PORT = o.port
    else:
        SERVER_PORT = 80

    if (o.path != None):
        PATH = o.path
    else:
        PATH = "/"

#...

#...
clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print("Connecting to server at (%s, %d) " % (SERVER_ADDRESS, SERVER_PORT))
try:
    clientSocket.connect((SERVER_ADDRESS, SERVER_PORT))
except OSError as err:
    print("Error. Retry."); continue

request = "GET %s HTTP/1.0\r\n\r\n" % PATH
print(request)
clientSocket.send(request.encode('ascii'))

print("Waiting for response ", end="... ")
reply = clientSocket.recv(1500)
print("received")
clientSocket.close()

print("Response:")
print(reply.decode('ascii'))
```

Mentre un server:

```
import http.server
import socketserver

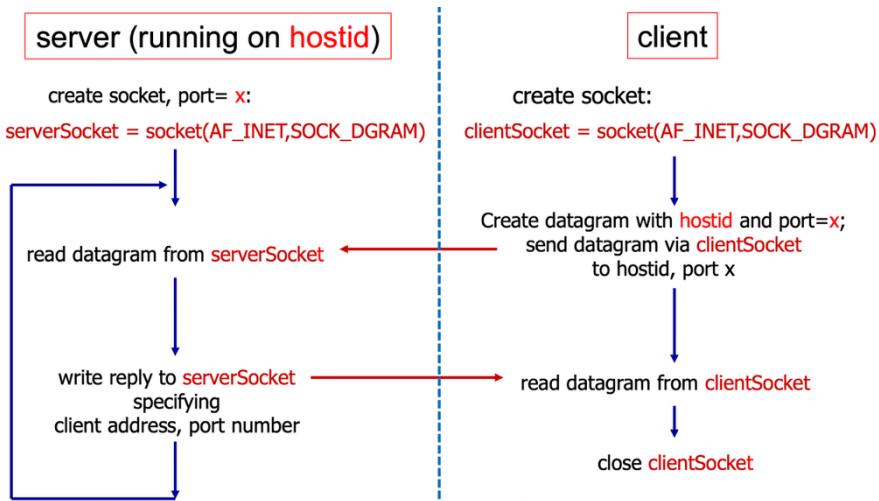
Handler = http.server.SimpleHTTPRequestHandler
DEFAULT_PORT = 8080

try:
    port = int(sys.argv[1])
except:
    port = DEFAULT_PORT

print ("Serving at port %s/tcp ..." % port)

httpd = socketserver.TCPServer(("0.0.0.0", port), Handler)
httpd.serve_forever()...
```

Lo schema di un sistema client-server con UDP è il seguente:



Un UDP client in Python 3 può essere implementato come segue:

```
import sys, getopt, socket
# ... def usage():
# ... def read_args():
SERVER_ADDRESS = "127.0.0.1"
SERVER_PORT = 12000
# read command line arguments
read_args()
# get user keyboard input
request = input("Input lowercase sentence: ")
# create UDP socket
clientSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# send request to server
clientSocket.sendto(request.encode(), (SERVER_ADDRESS, SERVER_PORT))

# wait for server response
print("Waiting for server response...")
response, serverAddress = clientSocket.recvfrom(2048)

# Print out received response
print(response.decode())

# Close socket
clientSocket.close()
```

Mentre un server:

```
import sys, getopt, socket
# ... def usage():
# ... def read_args():
SERVER_ADDRESS = "0.0.0.0"
SERVER_PORT = 12000
# read command line arguments
read_args()
# create UDP socket
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
serverSocket.bind((SERVER_ADDRESS, SERVER_PORT)) # bind socket to local port

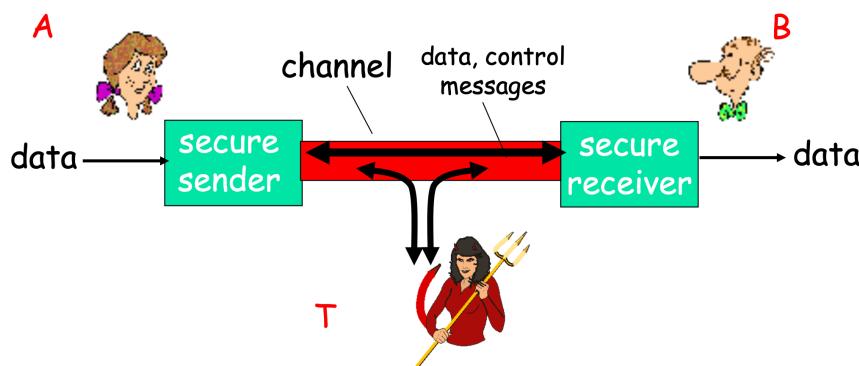
while True:
    print("Server waiting on (%s, %d)" % (SERVER_ADDRESS, SERVER_PORT))
    # receive request
    request, clientAddress = serverSocket.recvfrom(2048)
    print("Received request from (%s, %d)" % (clientAddress[0], clientAddress[1]))
    # convert message to upper case
    response = request.upper()
    # send back modified string to client
    serverSocket.sendto(response, clientAddress)
    print("Sent reply to (%s, %d)" % (clientAddress[0], clientAddress[1]))
```

LE TECNICHE CRITTOGRAFICHE

La sicurezza della comunicazione in rete coinvolge diversi aspetti:

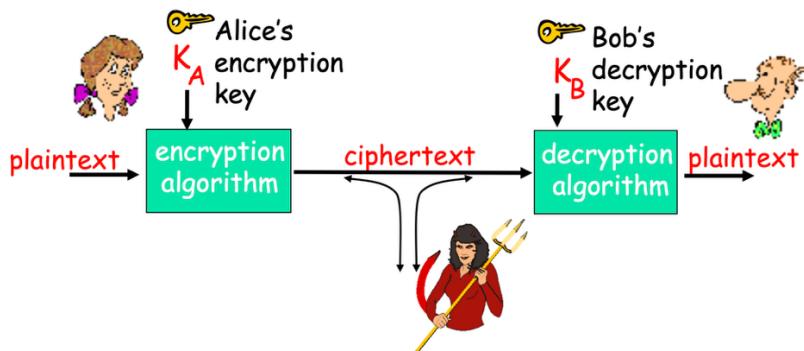
- **Riservatezza**, riferita alla necessità per cui solo il mittente ed il destinatario “legittimo” debbano essere in grado di comprendere il contenuto di un messaggio (qui intervengono le tecniche crittografiche, utili in primo luogo a proteggere la confidenzialità della comunicazione tramite cifratura (encryption) da parte del mittente e decifratura (decryption) da parte del destinatario);
- **Integrità dei messaggi**, riferita alla necessità per cui mittente e destinatario di un messaggio desiderino essere certi che i messaggi scambiati non siano alterati da una terza parte senza che se ne possano accorgere;
- **Autenticazione**, riferita alla necessità per cui mittente e destinatario di un messaggio desiderino essere reciprocamente sicuri dell’identità della controparte;
- **Accessibilità e disponibilità dei servizi**, riferita alla necessità per cui i servizi in rete debbano essere protetti da eventuali attacchi (ad esempio, attacchi DoS, Denial of Service).

Si considerino **due utenti**, A e B, che intendono comunicare in sicurezza attraverso la rete; tuttavia, l’utente T può ascoltare i messaggi scambiati da A e B, eventualmente alterandoli, cancellandoli o creandone dei falsi.



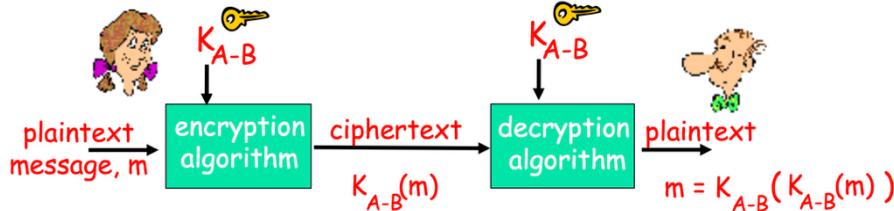
La terza parte può intercettare i messaggi inviati da A e B (eavesdropping), inviare pacchetti con il campo source address falso (spoofing) in modo da fingere di essere A, dirottare la comunicazione tra A e B piazzandosi “in mezzo”, potenzialmente fingendo all’uno di essere l’altro (hijacking) o impedire al servizio offerto da B di essere utilizzabile da altri utenti, tra cui A, sovraccaricando le risorse di B (denial of service).

Un **sistema crittografico** è un sistema in grado di cifrare e decifrare un messaggio attraverso l’uso di un **algoritmo** e di una **chiave**, in formato di stringa alfanumerica. Il messaggio da cifrare è detto “testo in chiaro” (plaintext), mentre il **risultato dell’algoritmo crittografico** è detto “testo cifrato” (ciphertext).

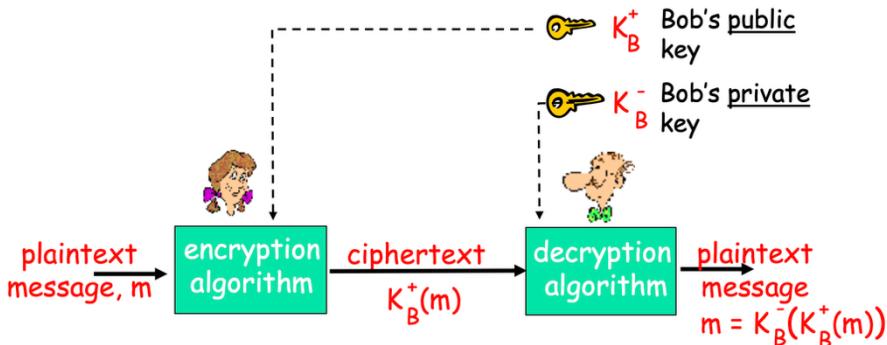


Un **sistema crittografico** può implementare **due tipologie di crittografia**:

- **Crittografia a chiave simmetrica**, per cui mittente e destinatario usano la stessa chiave (un segreto condiviso) per encryption e decryption;



- **Crittografia a chiave pubblica**, per cui la chiave di encryption è pubblica (nota a tutti) ma quella di decryption privata e segreta.



Secondo il **modello a chiave simmetrica**, A e B conoscono entrambi la chiave crittografica K_{A-B} e rimane solo da determinare il metodo con cui i due terminali della comunicazione si mettano d'accordo sul valore della chiave. Esistono diversi tipi di crittografia a chiave simmetrica, un esempio è il **cifrario per sostituzione**, per cui un'unità di testo del plaintext è sostituita con corrispondenti sequenze di simboli nel testo cifrato secondo uno schema regolare; in particolare, il **cifrario monoalfabetico** prevede una corrispondenza fissa tra ciascuna lettera dell'alfabeto in chiaro ed una lettera dell'alfabeto cifrato:

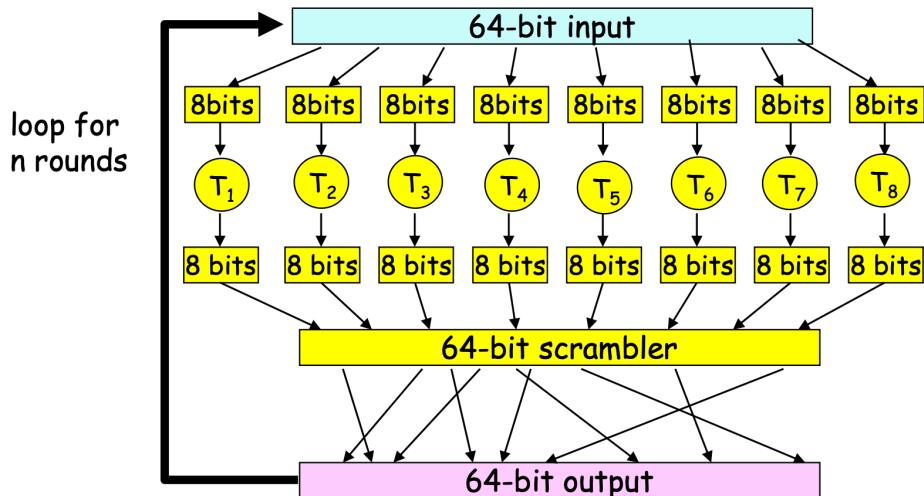
plaintext:	abcdefghijklmnopqrstuvwxyz
↓	↓
ciphertext:	mnbvcxzasdfghjklpoiuytrewq

Per cui la parola **bob** sarà sostituita con **nkn**. La forza di questo tipo di crittografia sta nella sua **robustezza quando lo si attacca con forza bruta** (tentando casualmente ogni possibile combinazione di lettere per ottenere il rapporto tra plaintext e ciphertext), richiedendo $26! = 2^{88}$ tentativi; tuttavia, **questo vantaggio va a cadere quando si entra nella critto-analisi**, ovvero studiando pattern ricorrenti e facendo analisi statistiche delle frequenze di occorrenza.

Un altro tipo di crittografia a chiave simmetrica è quello del **cifrario a blocchi**: un blocco di k bit del testo in chiaro è codificato con altri k bit nel testo in codice secondo uno schema fisso; ad esempio, con k pari a 3:

000 → 110	001 → 111	010 → 101	011 → 100
100 → 011	101 → 010	110 → 000	111 → 001

Il messaggio 010 110 001 111 è, così, codificato in 101 000 111 001. Il numero di permutazioni di 2^k (8 in questo caso) oggetti è $2^k!$ (40320 in questo caso), rendendo necessario aumentare k (almeno a 64); tuttavia, per valori di k elevati, l'implementazione inizia ad essere difficoltosa: cifratura e decifratura richiedono una tabella di 2^k elementi in memoria. Una soluzione potrebbe essere quella di ottenere valori elevati di k per composizione di valori più piccoli:



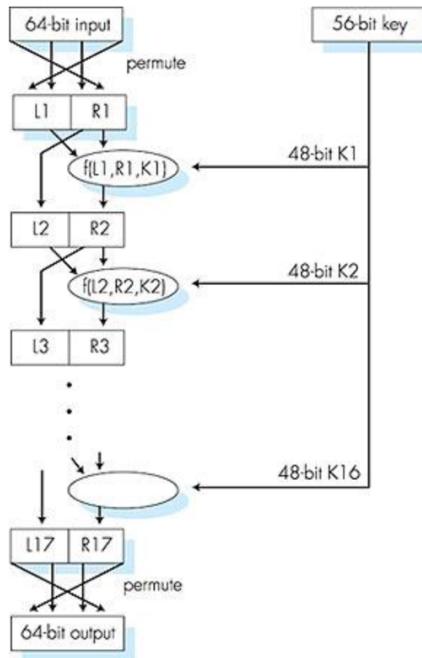
Dopo un'iterazione, ogni singolo bit del testo in chiaro influenza otto bit del testo cifrato. Se la funzione di rimescolamento è fissa, la chiave è costituita dalle 8 tabelle di permutazione. Esempi di questo tipo di cifratura a blocchi sono DES, 3DES e AES.

DES, Data Encryption Standard, è uno standard statunitense (NIST 1993) che usa una chiave simmetrica di 56 bit in un plaintext di input di 64 bit; la sua sicurezza è descritta dalla stessa chiave, una stringa di 56 bit che recita “Strong cryptography makes the world a safer place” e che è stata decrittata con bruteforce in 4 mesi; inoltre, non è ancora conosciuto per questo standard alcuna backdoor di decrypting. Ciò che rende DES ancora più sicuro è l’uso di tre chiavi sequenziali (3DES) per ogni dato e l’uso di cipher-block chaining.



DES codifica blocchi di 64 bit con una chiave di 56 bit e 8 bit di parità; in particolare, la chiave è memorizzata su 64 bit, con ogni ottavo bit calcolato come bit di parità per i precedenti 7. Il funzionamento del DES prevede i seguenti passaggi:

- Permutazione iniziale;
- 16 iterazioni in cui si applica una funzione f, usando in ciascuna iterazione 48 bit della chiave;
- Permutazione finale.



nota a tutti, e una privata per la decifratura, nota solo al destinatario. **Un algoritmo di cifratura a chiave pubblica deve:**

- Trovare una coppia di chiavi $K^+(\cdot)$ e $K^-(\cdot)$ tali che $K^-(K^+(m)) = m$;
- Rendere impossibile, nota la chiave pubblica $K^+(\cdot)$, calcolare la chiave privata $K^-(\cdot)$.

Un esempio di **algoritmo di cifratura a chiave pubblica** è **RSA** (Rivest, Shamir e Adleman, i suoi ideatori) e prevede che:

1. **Siano scelti due numeri primi grandi, p e q ;**
2. **Si calcolino $n = pq$ e $z = (p - 1)(q - 1)$;**
3. **Si sceglie un numero $e < n$ che non abbia fattori comuni con z (e e z devono essere relativamente primi);**
4. **Si sceglie un numero d tale che $ed - 1$ sia esattamente divisibile per z (ovvero, che $ed \% z = 1$);**
5. **La chiave pubblica è (n, e) e la chiave privata (n, d) .**

Dati (n, e) e (n, d) :

- La cifratura del testo in chiaro m (stringa di bit) si effettua calcolando $c = m^e \% n$;
- La decifratura del testo cifrato c (stringa di bit) si effettua calcolando $m = c^d \% n$.

Pertanto:

$$m = (m^e \% n)^d \% n$$

Con % operatore di resto della divisione. La scelta di questa formula risiede in un **utile risultato di teoria dei numeri**; se p e q sono primi e $n = pq$, allora:

$$x^y \% n = x^{y\%(p-1)(q-1)} \% n$$

Per cui:

$$(m^e \% n)^d \% n = m^{ed} \% n = m^{ed\%(p-1)(q-1)} \% n = m \% n = m$$

Dovuta proprio al fatto che $ed\%(p - 1)(q - 1) = 1$. Un'altra proprietà notevole dell'algoritmo RSA è la seguente:

$$K^-(K^+(m)) = m = K^+(K^-(m))$$

Ciò significa che **le chiavi possono invertirsi**, a chiave pubblica può funzionare da chiave privata e viceversa.

La robustezza di RSA dipende dal fatto che non sono noti algoritmi veloci per la fattorizzazione di numeri interi grandi, anche se $n = pq$ è noto, **i due fattori p e q non sono velocemente determinabili** e, di conseguenza, **non è velocemente determinabile z** , da cui si potrebbe facilmente risalire alla componente d della chiave privata (essendo già nota la componente e della chiave pubblica). **L'elevamento a potenza richiesto da RSA, però, è computazionalmente oneroso**, algoritmi a chiave simmetrica come DES sono dalle 100 alle 1000 volte più veloci di RSA; proprio per questo motivo, nelle comunicazioni su rete, RSA è spesso usato in combinazione con DES o 3DES.

INTEGRITÀ DEI MESSAGGI E PROTOCOLLI DI AUTENTICAZIONE

Una funzione di hash H è una funzione non invertibile che mappa una stringa m di lunghezza arbitraria in una stringa h di lunghezza minore prefissata (digest):

$$h = H(m)$$

Le funzioni hash crittografiche sono particolari funzioni di hash che godono di alcune proprietà che le rendono adatte all'uso nella crittografia:

- **Resistenza alla pre-immagine**, per cui, dato un valore di hash h , sia computazionalmente intrattabile la ricerca di una stringa m tale che:

$$h = H(m)$$

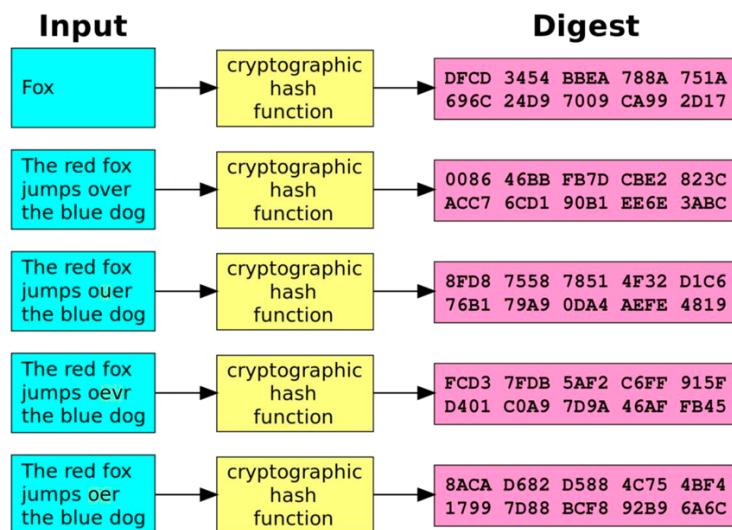
- **Resistenza alla seconda pre-immagine**, per cui, data una stringa m_1 , sia computazionalmente intrattabile la ricerca di una stringa m_2 tale che:

$$H(m_1) = H(m_2)$$

- **Resistenza alle collisioni**, per cui sia computazionalmente intrattabile la ricerca di una coppia di stringhe m_1 e m_2 tali che:

$$H(m_1) = H(m_2)$$

Di seguito è mostrato un **esempio concreto di hashing**, prendendo in considerazione 20 byte restituiti dalla funzione di hash SHA-1 per cinque diversi messaggi, evidenziando come piccole modifiche restituiscano in output digest completamente diversi:



Data una funzione di hash $H(\cdot)$, si dice che si produce una collisione quando due documenti diversi hanno lo stesso valore di hash. La internet checksum, usata da TCP e UDP per verificare l'integrità di una PDU arrivata a destinazione, produce un digest di lunghezza fissa (16 bit) da un messaggio di lunghezza arbitraria ma, sebbene matematicamente lo possa essere considerata, crittograficamente non è una funzione di hash; infatti, dato un messaggio con un dato valore di checksum, è facile trovare un altro messaggio con lo stesso valore di checksum:

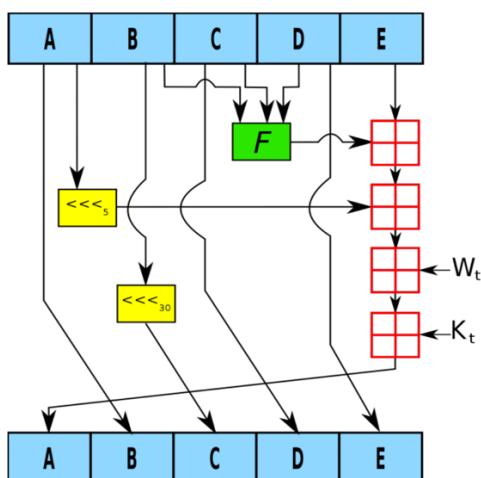
<u>messaggio</u>	<u>ASCII format</u>
I O U 1	49 4F 55 31
0 0 . 9	30 30 2E 39
9 B O B	39 42 4F 42
	<hr/>
	B2 C1 D2 AC

<u>messaggio</u>	<u>ASCII format</u>
I O U 1	49 4F 55 31
0 0 . 9	30 30 2E 39
9 B O B	39 42 4F 42
	<hr/>
	B2 C1 D2 AC

In pratica, sono utilizzate perlopiù due funzioni di hash:

- **MD5** (definita in RFC 1321 e ideata da Ron Rivest), calcola un valore di hash da 128 bit in 4 passi ma, recentemente nel 2005, sono state trovate delle tecniche di attacco che sono in grado di trovare collisioni, rendendo, almeno sul piano teoretico, non più inutilizzabile MD5;
- **SHA-1**, calcola un valore hash da 160 bit con un algoritmo basato su 80 iterazioni ed è utilizzato da diversi protocolli ed applicazioni (come TSL, SSL, PGP, SSH, S/MIME e IPsec), anche se nel 2005 un gruppo di ricercatori cinesi ha annunciato di aver trovato una tecnica in grado di rilevare collisioni con un numero di operazioni nell'ordine di 2^{69} e se nel 2017 un gruppo di ricercatori di Google ha annunciato di aver realizzato con successo due diversi file PDF aventi lo stesso hash in un numero di tentativi di circa 2^{63} , rendendo, almeno sul piano teoretico, non più inutilizzabile SHA-1.

SHA-1 è chiamato **compression function** ed è formato da **4 cicli da 20 iterazioni ciascuno**, per un totale di **80 iterazioni**. Di seguito è mostrata una sola iterazione, considerando A, B, C, D ed E come parole di 32 bit; F è una funzione non lineare che varia ad ogni ciclo, \ll_n denota una rotazione dei bit di sinistra di n posti (con n che cambia ad ogni ciclo), K_t è una costante e il blocco + denota l'addizione modulo 2^{32} :



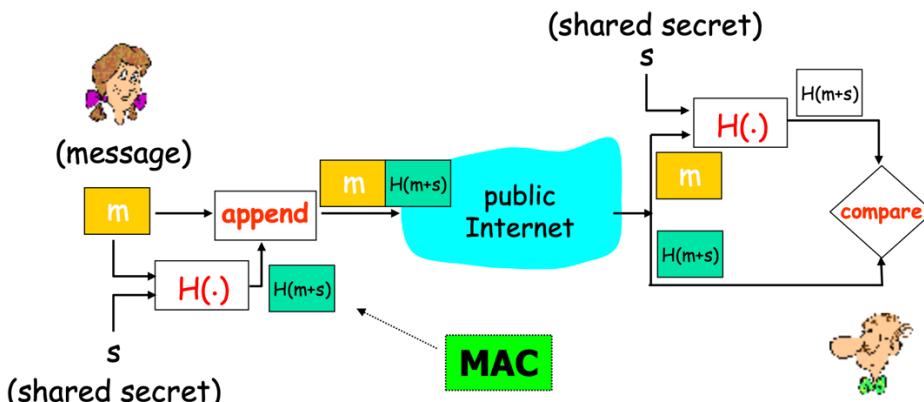
- **SHA-2**, indicando in realtà quattro diversi algoritmi (SHA-224, SHA-256, SHA-384 e SHA-512) che producono digest di lunghezza in bit pari al numero indicato nella propria sigla con un algoritmo simile a quello di SHA-1 (con il secondo e il quarto che lavorano con word di 32 e 64 bit, pur essendo sostanzialmente strutturati ugualmente, mentre il primo e il terzo sono solo versioni troncate dei corrispettivi precedenti due, con hash calcolati con differenti valori

iniziali) ma, almeno per ora, sicuro (infatti, ha sostituito SHA-1 in molte delle applicazioni precedentemente citate);

- **SHA-3**, è un sottoinsieme della famiglia di primitive crittografiche Keccak e l'ultimo membro della famiglia Secure Hashing Algorithm (SHA), è stato rilasciato dal NIST il 5 agosto 2015 e il suo codice sorgente dell'implementazione di riferimento è di pubblico dominio;

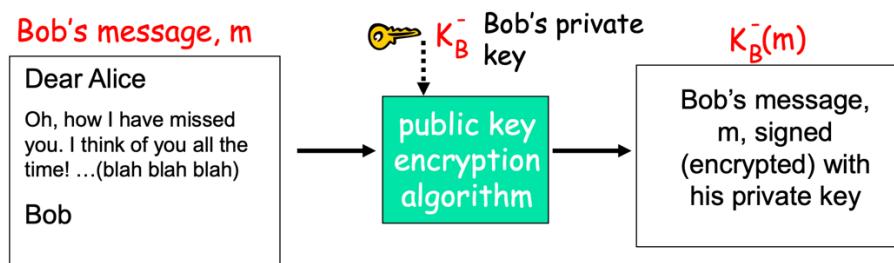
Si consideri la **comunicazione tra due individui, A e B; B riceve un messaggio da A e desidera assicurarsi che il messaggio non sia stato alterato da alcuno dopo che A lo ha inviato**. La prima soluzione è l'**uso di funzioni hash crittografiche**: A trasmette il messaggio insieme al valore di hash calcolato sul messaggio originario e, se il messaggio arriva a B alterato da un terzo T, il valore di hash calcolato da B sul messaggio stesso sarà completamente differente da quello generato e inviato da A a partire dal testo originale; tuttavia, questa verifica non è sufficiente: se T facesse pervenire a B l'hash ricalcolato sul messaggio alterato, B non avrebbe modo di scoprire la manomissione, occorre un segreto condiviso esclusivamente da A e B.

Una soluzione al problema può essere l'uso di un **Message Authentication Code (MAC)**, che **consente di controllare l'integrità di un messaggio attraverso l'uso di una funzione hash ed una chiave segreta s (chiave di autenticazione) nota ad entrambi (shared secret)**.



La firma digitale è una **tecnica crittografica** che ha gli stessi scopi della **firma cartacea**: il mittente di un messaggio (A) appone la sua **firma digitale** al fine di stabilire che egli è il creatore/autore del testo in esso contenuto. Come la firma tradizionale, anche quella digitale deve essere **verificabile e non falsificabile**: il destinatario del messaggio (B) deve poter provare ad un terzo che è stato B, e nessun altro (inclusa la stessa A) ad approvare la firma al documento.

Una semplice tecnica di firma digitale è la seguente: **B “firma” il messaggio crittografando con la sua chiave privata K_B^- creando un messaggio firmato $K_B^-(m)$** .



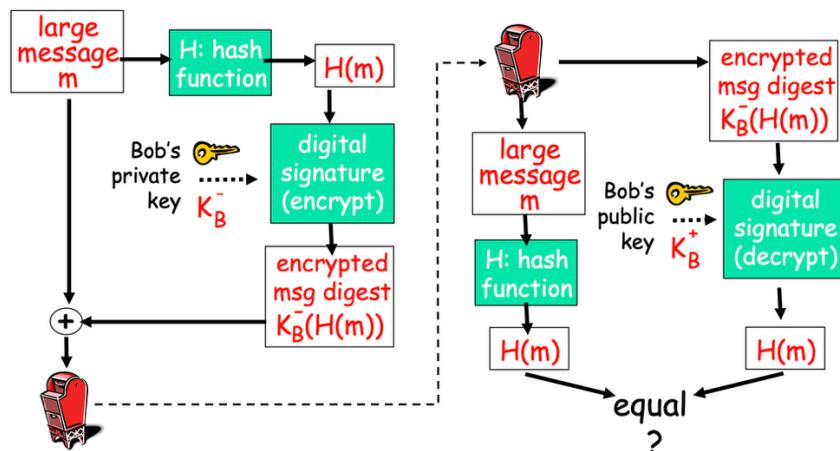
A riceve il messaggio m , con la firma digitale $K_B^-(m)$ e verifica che sia stato effettivamente firmato da B decifrando, con la chiave pubblica di B K_B^+ , il testo cifrato ricevuto $K_B^-(m)$ e verifica che:

$$K_B^+(K_B^-(m)) = m$$

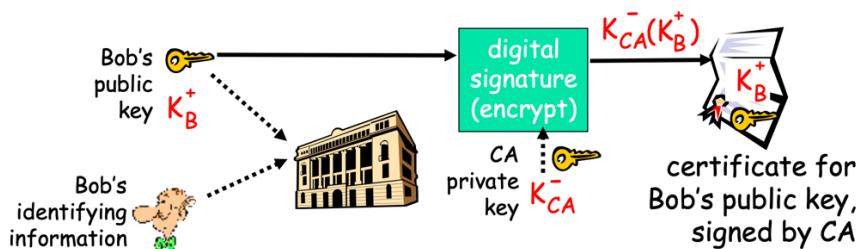
Se tale uguaglianza è rispettata, chiunque abbia firmato m deve possedere la chiave privata di B; in questo modo, A verifica che:

- B ha firmato m ;
- Nessuno ha firmato m ;
- B ha firmato m ma non m' .

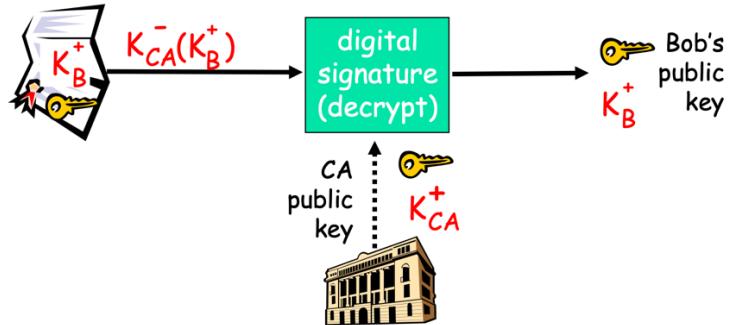
La firma digitale è uno strumento giuridicamente vincolante (in Italia, i presupposti giuridici si fondano soprattutto sull'articolo 15 comma 2 della legge 15 marzo 1997 n.59, la Bassanini 1) e A può portare il documento m , e la relativa firma $K_B^-(m)$, da un giudice per provare che B ha firmato m .



Quando A ottiene la chiave pubblica di B (da un sito web, una mail o altrove), come fa ad essere sicura che sia realmente la chiave pubblica di B e non di un terzo T? Il problema in questione prende il nome di problema di distribuzione affidabile delle chiavi pubbliche e viene risolto con un sistema di **Trusted Certification Authority** (o CA), un ente (pubblico o privato) abilitato a rilasciare un certificato digitale tramite procedura di certificazione che segue standard internazionali e conformi alla normativa europea e nazionale in materia. Un CA associa, in modo sicuro, una chiave pubblica ad una particolare entità E, la quale registra la sua chiave pubblica verso la CA; a tal fine, E fornisce la sua “prova di identità” alla CA, mentre quest’ultima crea un “certificato” che collega E alla sua chiave pubblica. Un certificato contiene la chiave pubblica di E firmata digitalmente dalla CA, che afferma “Questa è la chiave pubblica di E”.



Quando A desidera la chiave pubblica di B, ottiene il certificato di B (da B o da altri) e applica la chiave pubblica della CA al certificato di B, ottenendo la chiave pubblica di B.



Un certificato contiene un numero di serie, unico tra quelli emessi dalla CA, informazioni riguardo il possessore del certificato, informazioni su chi ha emesso il certificato, date di validità e la firma digitale di chi ha emesso il certificato, rendendo l'emissione della firma a tutti gli effetti un atto giuridicamente vincolante.

Per la costruzione di un protocollo di autenticazione (AP) si considerino le seguenti iterazioni, considerando come obiettivo la prova di identità di A a B:

- **Versione 1.0**

A comunica a B di essere A

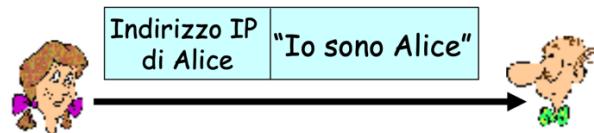


Il protocollo è fallimentare: B non può vedere A, un terzo T può fingersi A senza che B se ne accorga:

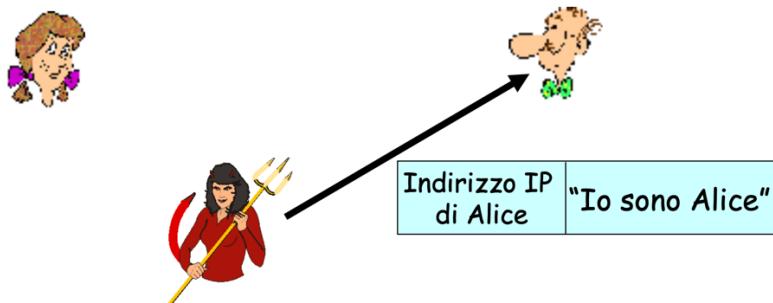


- **Versione 2.0**

A comunica a B di essere A in un pacchetto IP contenente il proprio indirizzo IP come indirizzo IP sorgente.

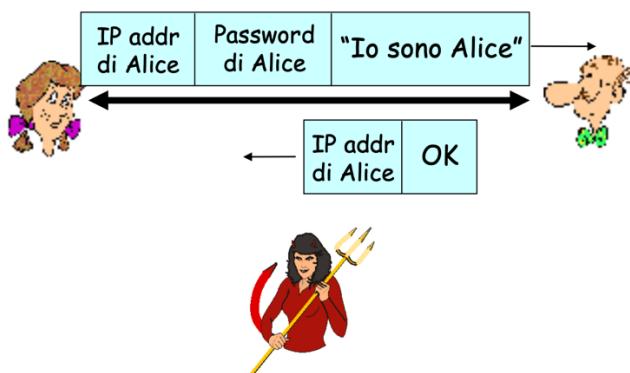


Il protocollo è fallimentare: un terzo T può creare un pacchetto IP facendo lo spoofing dell'indirizzo IP di A.

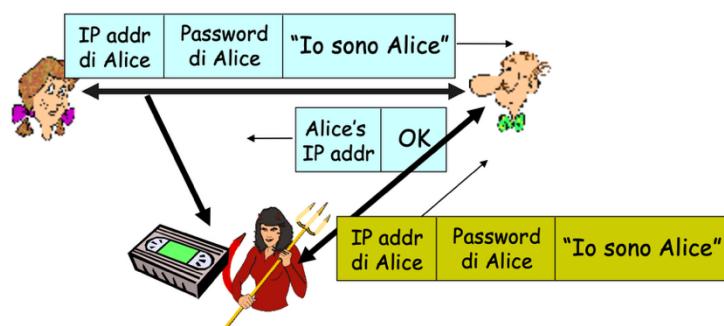


- **Versione 3.0**

A comunica a di essere A e manda la sua password segreta per provare la sua affermazione.

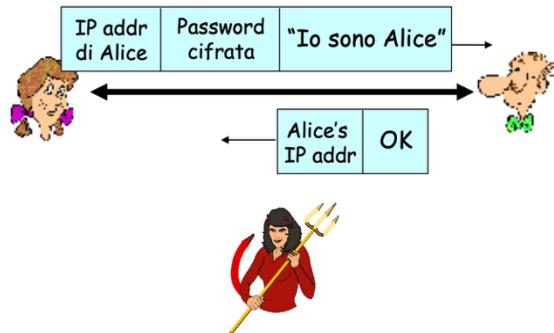


Il protocollo è soggetto ad **attacco playback**: un terzo T regista il pacchetto di autenticazione di A e, successivamente, lo rimanda a B.



- **Versione 3.1**

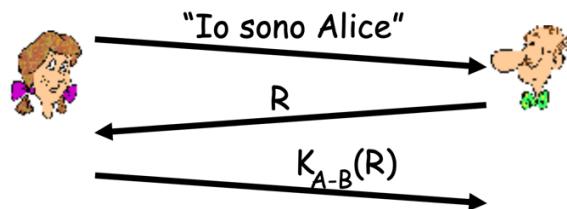
A comunica di essere A e manda la sua password segreta cifrata per provare la sua affermazione.



Il protocollo è soggetto ad **attacco playback**, esattamente allo stesso modo del caso precedente.

- **Versione 4.0**

L'obiettivo cambia, si passa dalla necessità di autenticare A a B alla **necessità di impedire l'attacco playback**. La soluzione è Nonce: B invia ad A un nonce, un numero R usato una sola volta, la quale deve restituire lo stesso numero ma cifrato con la propria chiave segreta.

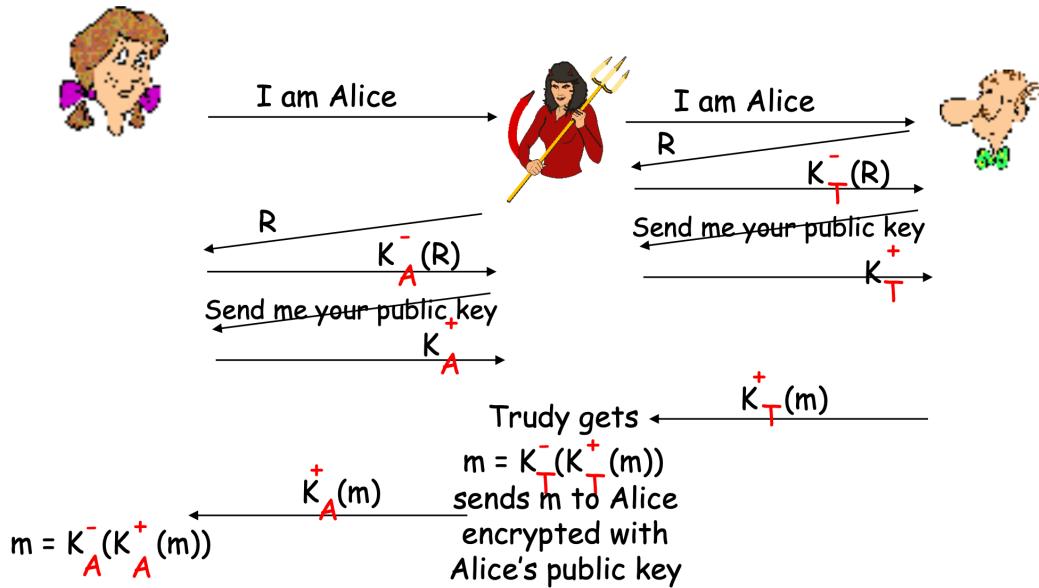


- **Versione 5.0**

Lo schema di base è lo stesso, se non che B richiede ad A la sua chiave privata. B calcola $K_A^+(K_A^-(R))$ e sa che A possiede la chiave privata tale che $K_A^+(K_A^-(R)) = R$; pertanto, se la valutazione di B è errata, l'autenticazione fallisce.

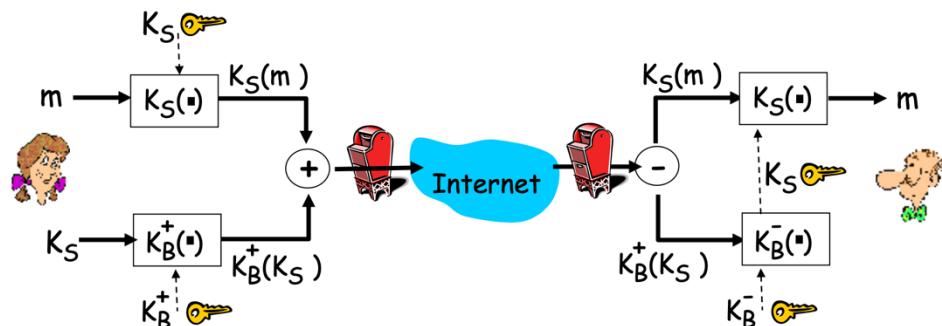


Il protocollo è soggetto ad **attacco “man in the middle”**: un terzo T finge di essere A con B e di essere B con A:

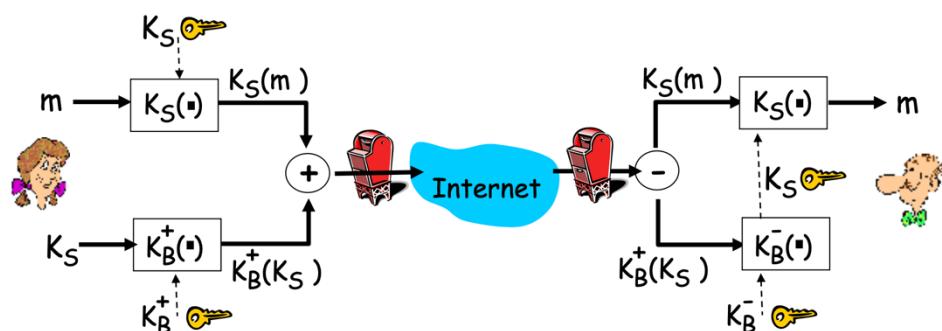


Questo tipo di attacco è difficile da rilevare: B riceve tutto quello che A gli manda, e viceversa, però anche T riceve tutti i messaggi. Il difetto di sicurezza di questa versione è legato alla distribuzione delle chiavi pubbliche.

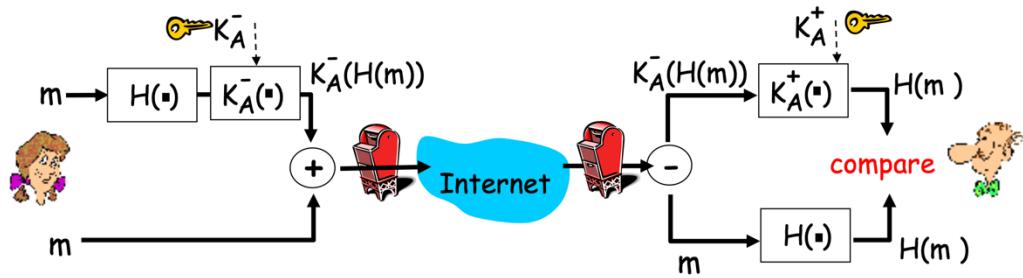
Si consideri ora il caso dell'invio di mail sicure: A vuole inviare un messaggio confidenziale a B, genera una chiave simmetrica privata K_S , cifra il messaggio m con K_S , cifra K_S con la chiave pubblica di B e invia sia $K_S(m)$ che $K_B(K_S)$ a B.



B, invece, usa la sua chiave privata per decifrare e recuperare K_S e usa K_S per decifrare $K_S(m)$ per recuperare m .



Un altro caso può essere quello in cui A vuole che B possa essere sicuro dell'identità del mittente e dell'integrità del messaggio ricevuto; A, quindi, appone la propria firma digitale al messaggio, che viene inviato in chiaro con la firma digitale.



Nel caso in cui A volesse inviare un messaggio confidenziale a B e che B possa essere sicuro dell'identità del mittente e dell'integrità del messaggio ricevuto, invece, A usa 3 chiavi: la propria chiave privata, la chiave pubblica di B e la chiave simmetrica appena generata K_S .

