Dispensa di

Reti Informatiche

Autore: Gabriele Frassi

Università di Pisa
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea triennale in Ingegneria Informatica

Prof. Giuseppe Anastasi, ing. Francesco Pistolesi

SOMMARIO DISPENSA

1 PREMESSE DELL'AUTORE

La presente dispensa è stata realizzata da una persona che ha seguito il corso di Reti Informatiche durante l'A.A.2021-2022, ma ha riseguito parte del corso durante l'A.A.2022-2023. Troverete all'interno capitoli dedicati alla teoria del prof. Anastasi e capitoli dedicati ai laboratori dell'ing. Pistolesi.

La quasi totalità degli argomenti sono coperti, ma mi preme segnalarvi le seguenti assenze:

- **Capitolo sulle applicazioni**
  Protocollo FTP (per chi ha seguito nell'A.A.21-22 e precedenti) e Video streaming (per chi ha seguito nell'A.A.22-23).
- **Capitolo sull'Internetworking**
  IPv6 e forwarding.
- **Capitolo sulla sicurezza**
  SSL (per chi ha seguito nell'A.A.21-22 e precedenti) e TLS (per chi ha seguito nell'A.A.22-23).
- **Capitolo Wireless e Mobile.**
  Spiegazione su 4G, 5G, LTE (introdotte nell'A.A.22-23).
- **Lezioni di laboratorio dell'ing. Pistolesi**.
  Spiegazione su NAT e PAT/NAPT.

Il mio consiglio è di prendere a riferimento (come indice degli argomenti dell'esame) le diapositive dei due docenti, in modo tale da non farsi sfuggire niente.

- **Orale di Anastasi**.
  Anastasi ha passione per gli schemi, memorizzate qualunque cosa che assomigli a uno schema. Conveniente avere a mente il formato dei pacchetti (sapere nel dettaglio semantica e sintassi, eventualmente sapere rappresentare il formato dei pacchetti come nelle figure delle diapositive).
- **Orale di Pistolesi**.
  Pistolesi inizia l'orale ponendo domande sul progetto (che possono vertere su cose che lui non condivide o su cose che non è stato in grado di comprendere a pieno leggendo la documentazione). A seguito di questo la prima domanda verte sempre sull'area di programmazione (parlare di particolari primitive, ad esempio). Successivamente si muove sulla parte rimanente delle sue lezioni. Molto frequentemente fa usare il suo computer presentando una macchina virtuale che non si connette ad Internet: compito del candidato è capire perché non funzioni usando i comandi e i file di configurazione presentati a lezione.

Su Github trovate un elenco di domande fatte negli appelli passati: consultatelo!

Fonti della dispensa.

- Lezioni e diapositive del prof. Anastasi (le diapositive sono una versione modificata di quelle degli autori del libro del corso "Reti di calcolatori e Internet")
- Lezioni e diapositive dell'ing. Pistolesi

2 PREMESSE SUL CORSO

2.1 Ore di lezione

Il corso di Reti informatiche è da 9 CFU, quindi 90 ore suddivise in:

- lezioni di teoria del prof. Anastasi;
- laboratori dell'ing. Pistolesi.

2.2 Argomenti propedeutici

Per il superamento dell'esame è necessario aver verbalizzato l'esame di Calcolatori elettronici, dove sono affrontati i principali argomenti propedeutici al corso:

- architettura di un calcolatore;
- gestione delle interruzioni;
- operazioni di I/O;
- organizzazione della memoria;
- concetto di processo;
- gestione dei processi.

Si richiede la conoscenza di almeno un linguaggio di programmazione ad alto livello: nel corso, così come nel progetto da presentare all'esame, è richiesto il C (i laboratori introdurranno gli argomenti dal punto di vista di chi ha studiato il C++, evidenziando le differenze tra C e C++). Il docente suggerisce di seguire di pari passo il corso di Sistemi Operativi.

2.3 Argomenti

2.3.1 Introduzione alle reti informatiche

Per prima cosa è necessario dare una definizione a Internet, infrastruttura estremamente complessa: si distingue il punto di vista dell'utente (che usufruisce del servizio) da quello dell'Ingegnere (visione dadi e bulloni). Partendo dalla definizione ingegneristica introdurremo:

- gli "ingredienti" principali di una rete (host, router, switch, reti di accesso, link di comunicazione);
- la distinzione tra frontiera (*edge*) e nucleo (*core*) della rete, la struttura del nucleo e il ruolo degli *Internet Service Provider*;
- le caratteristiche e lo scopo dei protocolli di rete;
- gli approcci con cui è possibile gestire il core della rete, cioè *packet-switching* e *circuit-switching*;
- concetti di ritardo di pacchetti, perdita di pacchetti e throughput (analisi della performance di rete);
- il modello stratificato su cui si basa Internet.

2.3.2 Applicazioni di Rete

L'argomento fondamentale è l'introduzione dei due paradigmi fondamentali, adottati nella realizzazione di un'applicazione di rete:

- client-server (il server offre il servizio, mentre il client lo richiede) e
- peer-to-peer (comportamento variabile degli "attori" coinvolti, in alcuni casi possono comportarsi come server e in altri come client).

Per quanto riguarda il peer-to-peer approfondiremo due aspetti: la ricerca dei file e il file downloading. Vedremo i seguenti esempi di applicazioni di rete, basate sui due paradigmi:

- Web, posta elettronica e DNS per quanto riguarda il paradigma client-server;
- bitTorrent per quanto riguarda il paradigma peer-to-peer.

Si deve capire, nella progettazione di un'applicazione, quale sia il modello più adatto da utilizzare. Introdurremo anche il concetto di *socket* (non standardizzato, ma di ampio utilizzo) per mezzo di API. Nelle lezioni di laboratorio saranno introdotte le primitive socket da utilizzare per la realizzazione del progetto.

2.3.3 Reti a collegamento diretto (Direct Connection Networks)

In questo capitolo si affronta il livello Link della pila protocollare, dove si ragiona in termini di frame. Il punto di partenza è la descrizione di un collegamento punto-punto (tra due dispositivi): prima quello ideale e infine quello reale, dove si tiene conto dell'introduzione dell'errore. Emerge la necessità di elaborare soluzioni atte a minimizzare l'errore (si vuole rendere il collegamento reale il più simile possibile a quello ideale).

- Per prima cosa introdurremo i servizi per la mitigazione dell'inaffidabilità: *error detection*, *retransmission* ed *error correction*.
- Successivamente descriveremo dettagliatamente il *Reliable Data Transfer Protocol* (RDT), un protocollo per la realizzazione di un servizio affidabile nel contesto di un canale inaffidabile. Ulteriori protocolli che introdurremo sono go-back-N e Selective Repeat, basati su un approccio differente rispetto a RDT.
- Protocolli point-to-point: caratteristiche e formato del frame.
- Passaggio da collegamenti punto punto a reti ad accesso multiplo:
    - protocolli Multiple Access (con gestione delle collisioni)
    - reti LAN (caratteristiche, topologia, comunicazione broadcast, indirizzi fisici MAC e rete Ethernet come principale esempio di rete LAN)

Le reti LAN con la loro topologia a stella rappresentano un'anticipazione del capitolo successivo.

2.3.4 Reti a commutazione di pacchetto (Packet Switched Networks)

Le reti LAN sono caratterizzate dalla presenza di dispositivi detti switch, che permettono un instradamento intelligente dei pacchetti all'interno di una rete.

- Tabelle di forwarding, consultate ad ogni passaggio di pacchetto per determinare verso quale uscita dello switch instradare il pacchetto. Riempimento della tabella secondo approccio plug and play.
- Reti Ethernet con switch
- Reti LAN virtuali (VLAN).

Le novità introdotte permettono la realizzazione di una rete locale avente estensione geografica maggiore, così come una sua gestione più efficiente. Tutto ciò non è ancora sufficiente.

2.3.5 Interconnessione di reti (Internetworking)

Il passaggio successivo è la cosiddetta astrazione dell'Internetworking, implementata a livello Network: si offre l'impressione che il pacchetto sia trasmesso su un'unica rete, nei fatti il pacchetto attraversa una molteplicità di reti con caratteristiche diverse (ad esempio i protocolli, peculiarità che emergono nel livello Datalink visto nei capitoli precedenti).

- Per prima cosa evidenzieremo la centralità del router, dispositivo implementato a livello Network. Distingueremo routing da forwarding e introdurremo le caratteristiche base dell'architettura del router (da un punto di vista funzionale).
- Concluderemo spiegando il protocollo IP, che permette l'attuazione di questa astrazione: formato dal datagram IP, frammentazione del datagram, ruolo degli indirizzi IP e assegnazione degli stessi.
- Network Address Translation per alleviare il problema della scarsità di indirizzi IP in IPv4: distinzione tra indirizzi pubblici e privati, NAT Translation Tables.

2.3.6 Trasporto

Concludiamo introducendo i protocolli di trasporto:

- protocollo TCP (*Transfer Control Protocol*);
- protocollo UDP (*User Datagram Protocol*).

Introdurremo al volo UDP (con tanto di formato del segmento) e approfondiremo TCP: creazione e scioglimento della connessioni TCP, Reliable Data Transfer (come lo otteniamo, come calcoliamo il timeout), formato del segmento TCP, controlli di flusso e controlli di congestione.

2.3.7 Sicurezza

La questione della sicurezza emerge con la nascita dell'Internet di massa: numero elevato di utenti che non possono essere considerati benevoli aprioristicamente (prima Internet era usato solo in ambito accademico e militare). Due sono gli aspetti:

- Comprendere cosa si intenda con sicurezza per progettare applicazioni sicure;
- Favorire una maggiore consapevolezza dell'utente (awareness), che molto spesso introduce lui stesso software malevoli (malware) nel dispositivo (cavallo di Troia).

Introdurremo per prima cosa esempi di azioni malevoli da parte dell'utente e gli aspetti che concorrono alla definizione di sicurezza in una particolare applicazione. Si introduce la crittografia (cifratura e decifrazione, principali cifrari, crittografia a chiave simmetrica e crittografia a chiave pubblica, scambio di chiavi per mezzo del KDC o della crittografia a chiave pubblica). Vedremo Message Authentication Code (MAC), firma digitale e realizzazione di servizi di autenticazione. Vedremo anche degli esempi di implementazione:

- posta elettronica,
- protocollo IPsec,
- firewall e Intrusion detection systems.

2.3.8 Reti Wireless e Mobili

Il capitolo è un'introduzione agli argomenti che saranno affrontati nelle magistrali UniPI di Ingegneria Informatica. Fino a questo istante si è parlato di reti con link di comunicazione wired e guidati.

- Si introducono le caratteristiche della connessione Wireless, con le problematiche ad essa connesse: la propagazione multidirezionale, il problema della path loss, il problema del nodo nascosto.
- Si classificano le reti Wireless tenendo conto della presenza o meno di un'infrastruttura di rete fissa (intrastructure-based vs infrastructureless) e dell'area di trasmissione della rete. Si distinguono anche tecnologie single hop da tecnologie multi-hop.
- Segue l'introduzione di alcune tecnologie wireless:
    - Reti Wifi (WLAN), dove si affronta il problema delle collisioni e il problema del nodo nascosto introducendo una versione modificata del protocollo CSMA;
    - Reti cellulare (un semplice assaggio)
    - Reti Bluetooth, con esempi di applicazione e protocollo di accesso.
- Si è affrontata anche la questione della mobilità rispetto al protocollo IP: come posso conciliare la necessità di mantenere indirizzi IP permanenti con cambi continui di indirizzi in passaggio da sottoreti ad altre sottoreti?

2.4 Laboratorio

I laboratori svolti dall'ing. Pistolesi affronteranno i seguenti argomenti:

- Introduzione ai sistemi Linux/Unix (lezione congiunta di Reti Informatiche e Sistemi Operativi)
- configurazione della rete (elementi alla base della configurazione, configurazione manuale, risoluzione dei nomi, configurazione automatica per mezzo del DHCP server);
- comandi per testare la connessione (comando ping per testare la connettività e comando traceroute per l'individuazione dello shortest path percorso);
- nozioni per la programmazione di applicazioni distribuite (differenze tra C e C++, concetto dietro l'astrazione del socket e tutte le primitive socket utili a gestire connessioni TCP, uso delle primitive socket per gestire connessioni UDP e distinzione tra protocolli text e protocolli binary);
- firewall (firewall a filtraggio di pacchetto, impostazione delle regole, comando netfilter e comando iptables per gestire il contenuto delle tabelle);
- Server Web Apache (scopo dell'applicazione, configurazione e Multi-Processing Module);
- Algoritmi di instradamento (algoritmi globali e algoritmi decentralizzati, routing gerarchico, divisione della rete in autonomous system, algoritmi intra-AS e inter-AS, introduzione ai principali algoritmi e loro caratteristiche principali)

2.5 Risposta ad ogni domanda: dipende

Una cosa che riecheggia molto spesso in questo corso è la risposta preferita degli ingegneri ad ogni domanda:

Dipende

Il professore affronta la cosa con un certo divertimento (soprattutto quando la gente inizia a rispondergli sempre dipende durante le lezioni), ma dietro questo divertimento si cela un insegnamento fondamentale: non esistono soluzioni universali a un problema. È necessario valutare il contesto ogni volta che si deve assumere una decisione (Esempio: scelta di un protocollo in un particolare contesto).

- Lo si tenga a mente in un colloquio di lavoro.
- Lo si tenga a mente durante l'esame (dipende non è una risposta sbagliata, ma bisogna dire anche *da cosa dipende*).

3 INTRODUZIONE ALLE RETI INFORMATICHE

3.1 Definizione di Internet

3.1.1 Introduzione alle diverse visioni

Come possiamo definire Internet? La risposta non è semplice! Nei fatti possiamo rispondere da uno dei seguenti punti di vista:

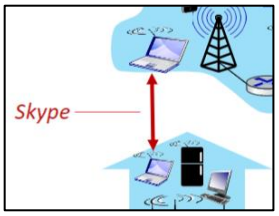
- Il punto di vista dell'Ingegnere (detta anche visione dadi e bulloni), che conosce Internet dall'interno e sa quali meccanismi sono alla base del suo funzionamento;
- Il punto di vista dell'utente medio, che conosce Internet dall'esterno e non ha una minima idea di quali meccanismi siano alla base del suo funzionamento (in un certo senso è come se vedesse una scatola chiusa, senza conoscerne il contenuto).

Per avere un'idea più chiara pensiamo al bosco: se lo vediamo dall'esterno ci sembra una macchia verde, se lo vediamo dall'interno vediamo piante e animali che lo caratterizzano.

3.1.2 Internet dal punto di vista dall'utente medio

Internet è un'infrastruttura che offre servizi: web, streaming, e-mail, giochi, e-commerce, social media, news, downloading…

I servizi in questione sono offerti alle applicazioni, che possono usufruirne per mezzo delle API (acronimo di *Application Programming Interface*). Parleremo soprattutto di applicazioni distribuite, cioè applicazioni dove avviene uno scambio di dati tra *hosts*.

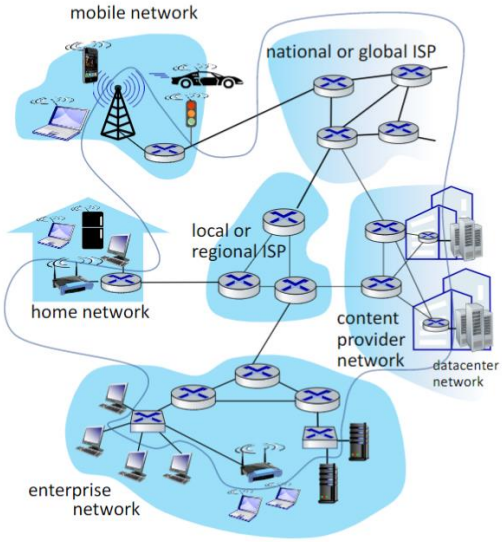


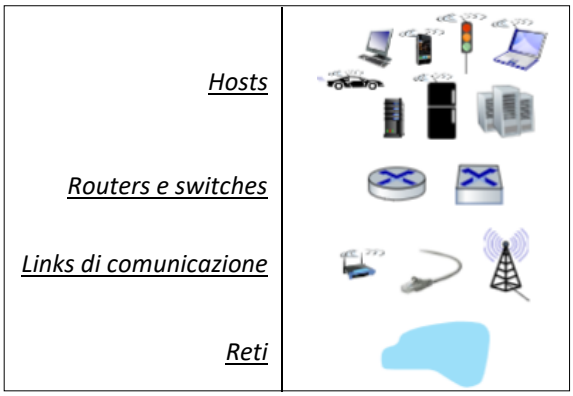
Esempio: due hosts che comunicano attraverso Skype.

3.1.3 Internet dal punto di vista dell'Ingegnere (visione *dadi e bulloni*)

3.1.3.1 Introduzione

Internet è un sistema che interconnette milioni di dispositivi[1] detti host, sui quali girano processi applicativi.





- Possiamo definire Internet letteralmente ***una rete di reti***.
- Una rete è costituita da dispositivi (gli *host*), router e link di comunicazione.
  - Gli ***host*** sono dispositivi collegati a una rete per mezzo di *reti di accesso*.
  - I ***router*** permettono la comunicazione tra reti di tipo diverso (sono come degli interpreti).
  - I ***link di comunicazione*** sono ciò che permette il collegamento tra routers di reti distinte. Questi possono essere *wired* (fibra, cavo …) o *wireless* (radio, satellite …)
- Una rete è gestita da una particolare organizzazione.

[1] Occhio a parlare solo di computer. Si legga più avanti la brevissima sezione introduttiva all'Internet of Things.

- Non ragioniamo in termini di trasmissione di messaggi, ma di pacchetti. I pacchetti sono sequenze di bit trasmesse da un processo dell'host mittente a un processo dell'host destinatario.
    - Se il messaggio è di brevi dimensioni coincide con un unico pacchetto.
    - Se il messaggio è di dimensione rilevante allora sarà diviso tra più pacchetti.

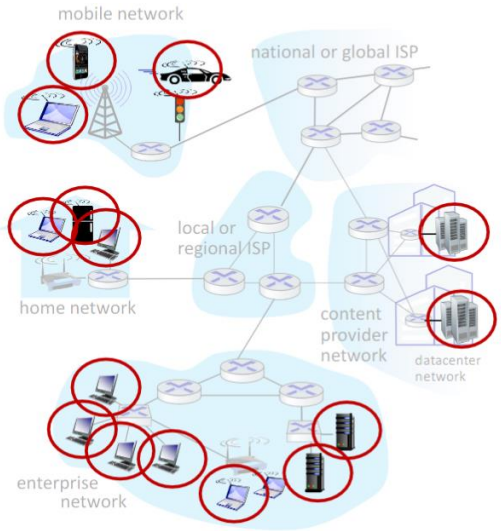
  I pacchetti, inoltre, contengono ulteriori informazioni utili alla trasmissione dello stesso e alla verifica della sua integrità.

L'Ingegnere informatico ragiona dal punto di vista funzionale, ergo ignorerà aspetti che interessano soprattutto ai telecomunicazionisti (che stanno molto simpatici al prof. Anastasi - mhm).

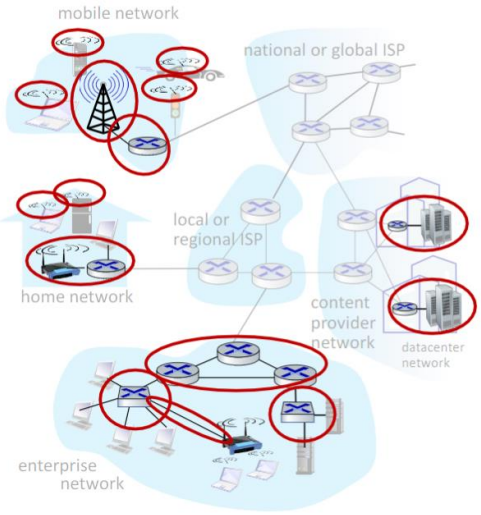
3.1.3.2 Struttura di Internet: confini (edge) e nucleo (core) di Internet

Approfondiamo ulteriormente quanto già detto sulla struttura della rete Internet, distinguendo:

- i confini della rete (*network edge*) costituiti dagli host, detti anche sistemi periferici (*end systems*) proprio per la loro posizione all'interno dell'infrastruttura;
- le reti di accesso;
- mezzi fisici (in primis link di comunicazione, che possono essere *wired* o *wireless*);
- il nucleo della rete (*network core*).



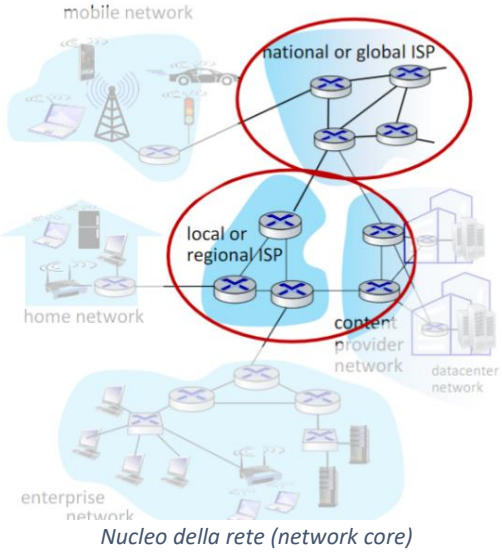
Confini della rete



Reti di accesso e mezzi fisici

Il concetto di *network core* è la novità principale che introduciamo rispetto agli elementi della pagina precedente. Abbiamo detto che reti diverse si comunicano per mezzo di routers.

- **Collegamenti diretti?**
  La prima cosa che ci potrebbe venire in mente è la presenza di un collegamento diretto tra le due reti comunicanti, per mezzo di un unico link dedicato. Questa cosa non è possibile per i costi e per il numero di reti esistenti (soluzione non scalabile).
- ***Network core***.
  Segue che due reti comunichino fra di loro per mezzo di un percorso costituito da più router. Insiemi di routers possono costituire delle reti: l'unione di queste reti è detta network core.
- **Chi gestisce il *network core*?**
  Una rete del network core ha come gestore il cosiddetto Internet Service Provider (ISP), che offre ai suoi clienti una rete di routers attraverso cui permettere la comunicazione. Gli ISP possono essere locali (Esempio: UniPI), regionali o nazionali (Esempio: TIM): la distinzione è influenzata principalmente dalla dimensione della rete.



Nucleo della rete (network core)

3.1.3.3 Classificazione degli host: host client e host server

Gli host sono classificabili nelle seguenti categorie:

- *client*, che richiedono servizi;
- *server*, che offrono servizi.

Un processo dell'host client richiede il servizio, e questo servizio è eseguito da un processo dell'host server: quanto detto costituisce il modello client-server alla base di numerose applicazioni distribuite.

Generalizzazione: modello P2P. Il modello peer to peer supera la distinzione netta tra host client e host server: abbiamo host alla pari, ciascun host può assumere in alcuni casi il comportamento dell'host client e in altri casi il comportamento dell'host server.

3.1.3.4 Il passaggio dall'Internet of computers all'Internet of Things

In principio la rete collegava esclusivamente calcolatori (uso soprattutto accademico e militare), successivamente ha permesso la comunicazione tra le persone (da *Internet of computers* si è passati a parlare di *Internet of people*). L'evoluzione finale della rete ci porta a parlare di *Internet of Things*: abbiamo centinaia di dispositivi connessi alla rete, anche i più impensabili. Si veda la seguente figura.

Si parla di *Internet of Things* perché le nuove tecnologie porteranno *le cose connesse* a superare in numero *le persone connesse*.

- Innovazioni rilevanti per il consumatore.
- Necessità per il giurista di riflettere sul trattamento dei nostri dati (si tenga a mente che *le cose*, per poter funzionare, devono raccogliere informazioni – siamo sicuri che le informazioni siano utilizzate solo per lo stretto indispensabile e che non vengano trasmesse a terzi per ulteriori fini?).

Il numero di dispositivi non tradizionali connessi alla rete è ciò che ha portato questo corso, inizialmente noto come *Reti di calcolatori*, ad assumere la dicitura attuale (*Reti informatiche*).

3.2 Definizione di protocollo

La rete è un insieme di host che hanno necessità di comunicare tra loro. In ogni contesto dove è necessario comunicare serve definire dei protocolli.

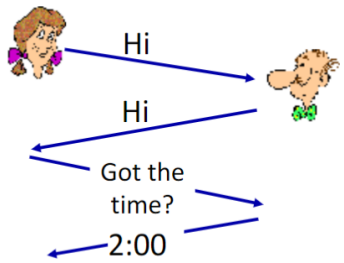
3.2.1 Esempi: i protocolli umani

Prima di introdurre la definizione formale cerchiamo di costruire un'idea partendo da concetti comuni.

- **Lezione di Reti Informatiche: spiegazioni e domande**.
  Siamo in un'aula e stiamo seguendo la lezione di Reti Informatiche. La lezione è caratterizzata dal docente che parla, ma anche da un'interlocuzione con i suoi studenti. Come avviene questa interlocuzione? Imponendo delle regole!
  - Il prof. Anastasi modera il dibattito.
  - Durante la spiegazione parla solo il professore.
  - Quando il professore ha finito di spiegare una certa sezione chiede se ci sono domande.
  - Lo studente alza la mano e attende che il professore dia il via libera. Il professore gestisce una domanda alla volta.
  - Lo studente abbassa la mano e pone la domanda quando ha ricevuto il via libera dal prof.
  - Il professore risponde alla domanda.

  Possiamo immaginarci il protocollo come un insieme di regole.

- **Protocollo per chiedere l'ora**.
  Il protocollo per chiedere l'ora regola l'interlocuzione tra due soggetti: un soggetto è colui che vuole sapere l'ora, l'altro è quello che la fornisce.
  - L'utente che vuole sapere l'ora saluta: Hi!
  - L'utente a cui verrà chiesta l'ora ricambia il saluto: Hi!
  - Il primo utente chiede l'ora
  - Il secondo utente legge l'ora al primo.
  - Il primo utente ringrazia.
  - Il secondo ricambia il ringraziamento.

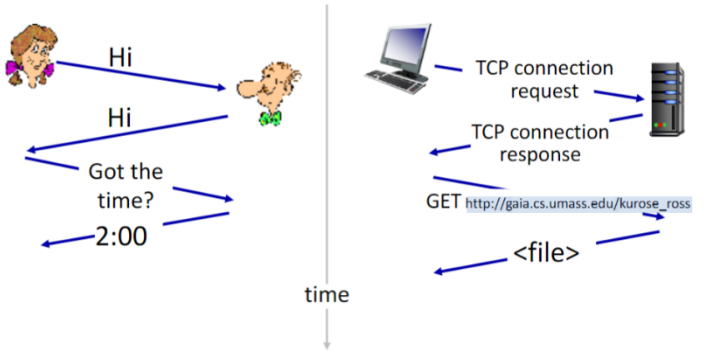


  Quello che osserviamo è che abbiamo una sequenza di messaggi ben precisa, dove ciascun messaggio presenta un formato ben preciso. Il protocollo definisce l'ordine dei messaggi (l'utente che chiede l'ora non inizia la discussione ringraziando, ma salutando – mentre l'utente che restituisce l'ora ricambierà il ringraziamento solo dopo essere stato ringraziato) e il loro formato: se si rispetta il protocollo si riesce a ottenere l'obiettivo desiderato.

Morale della favola: la nostra vita è caratterizzata da protocolli.

3.2.2 Il protocollo nelle Reti informatiche: definizione e creazione dei protocolli

Un protocollo definisce il formato e l'ordine dei messaggi scambiati tra due o più ***entità di rete*** (gli host sostituiscono le persone, se si fa confronto coi protocolli umani), così come le azioni intraprese in fase di trasmissione e/o di ricezione di un messaggio o di un altro evento. Abbiamo fatto un primo esempio formale parlando di protocollo TCP, che vedremo approfonditamente più avanti.



Protocollo umano visto prima a confronto col protocollo TCP.

Si distinguano due host: *host client* e *host server*.

- L'host client richiede all'host server di stabilire una connessione TCP.
- L'host server risponde affermativamente.
- L'host client, adesso che la connessione è stata stabilita, richiede una particolare pagina[2].
- L'host server risponde inviando il contenuto della pagina.
- [Extra] Il client segnala la volontà di chiudere la connessione TCP.

Qualcuno deve definire questi protocolli informatici, e deve esserci un protocollo per poter definire i protocolli stessi!

- Chi approva gli standard?
  I protocolli sono definiti dalla IETF (*Internet Engineering Task Force*).
- Chi propone uno standard? Come si arriva all'approvazione?
  Un soggetto propone un nuovo protocollo per mezzo di una RFC (*Request for Comments*): esso è un documento scritto da chi sviluppa un'applicazione, e sul quale la community discute suggerendo modifiche. Dopo una serie di discussioni il documento, con eventuali emendamenti proposti dalla comunità, viene approvato definitivamente e diventa standard.
- Perché certi soggetti hanno necessità di proporre uno standard?
  Lo standard permette a soggetti diversi di sviluppare parti diverse di una stessa applicazione: si pensi al browser, questo deve potersi interfacciare con tutti i server esistenti al mondo. Questo è possibile solo rispettando i protocolli definiti negli anni.

[2] La pagina posta ad esempio è il sito degli autori del libro, dove troverete le diapositive originali dei libri su cui Anastasi si è basato.

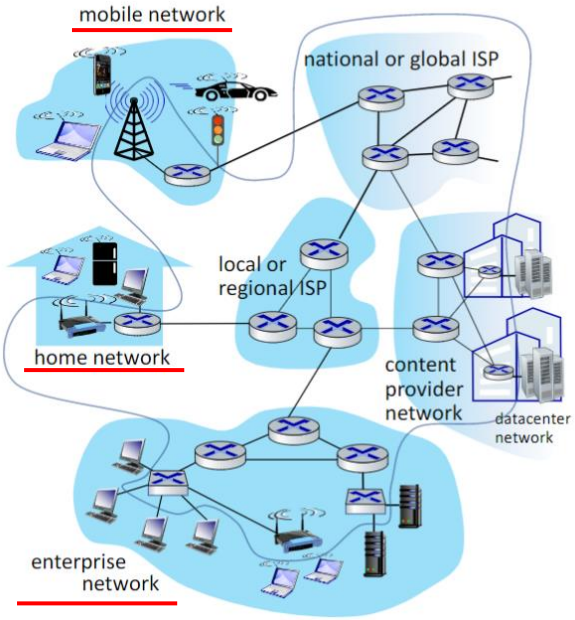
3.3 Reti di accesso (*access networks*)

3.3.1 Introduzione

Gli host sono collegati alla rete Internet per mezzo di reti di accesso. Formalmente definiamo la rete di accesso come *quella rete che connette fisicamente un sistema al suo edge router*, dove con edge router intendiamo il primo router sul percorso che un pacchetto dovrà attraversare per passare da host mittente ad host destinatario (ricordarsi la spiegazione precedente, il pacchetto viaggia passando da router).

Le reti di accesso sono classificabili in:

- **Reti residenziali** (la rete domestica)
  Utilizzata da utenti che vivono nella stessa abitazione.
- **Reti istituzionali**
  Fornita da enti/organizzazioni (UniPI, ma anche una società privata).
- **Reti mobili**.
  4G, 5G, ma anche la rete Wifi di un bar (le reti pubbliche pensate per hosts in mobilità).

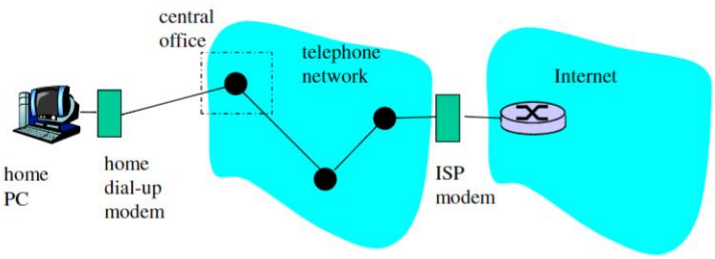


Di queste reti analizzeremo il *rate di trasmissione* (cioè quanti bit per secondo possono essere trasmessi) e se la rete è *dedicata* o *condivisa* (in questo caso la banda totale viene divisa per il numero di utenti presenti, e questo significa minore velocità).

3.3.2 Reti di accesso residenziali

3.3.2.1 Dial-Up Modem (deprecata)

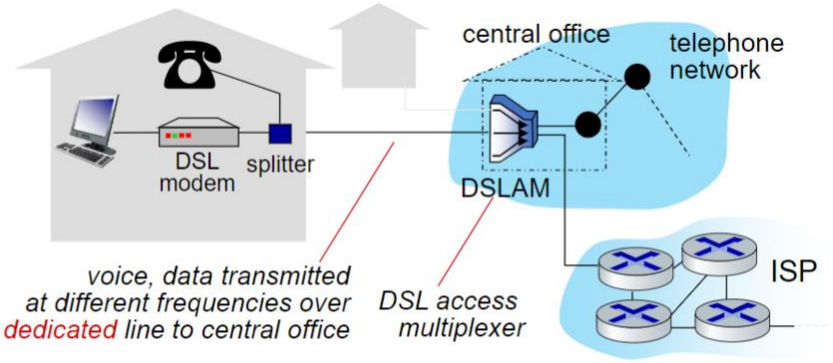
Prima forma di collegamento a Internet. Tecnologia ormai superata, che si basava sulle infrastrutture telefoniche (pervasive in Europa). Tre questioni…



- In questa tecnologia modem e Router erano dispositivi distinti: il modem si connette al router per mezzo della rete telefonica.
- Rete tradizionale pensata per trasmettere segnale vocale e non sequenze di bit: necessario svolgere un'operazione di modulazione prima di mettere il segnale sulla rete telefonica (per ragioni di efficienza), successivamente sarà svolta un'operazione di demodulazione del segnale tra la rete telefonica e il router del provider.
- Velocità molto bassa: al più 56Kbps.
- Assenza di una linea fisica dedicata per Internet: questo significa che non era possibile navigare in Internet e telefonare allo stesso tempo.

3.3.2.2 ADSL

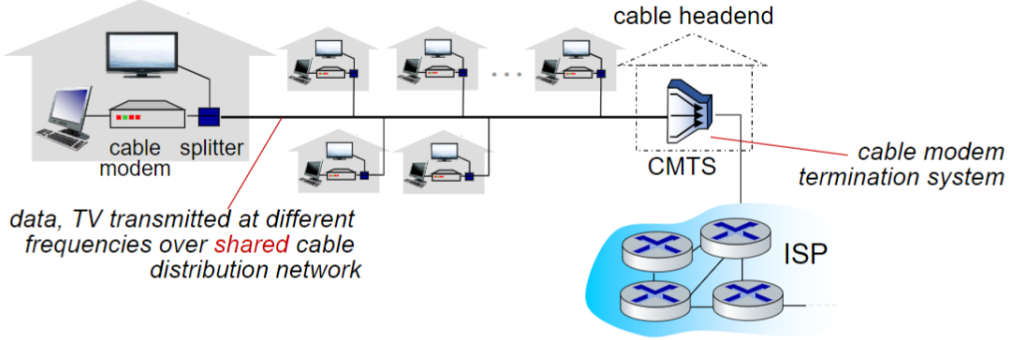
Tecnologia più diffusa in Europa, anche se sarà progressivamente soppiantata dalla fibra ottica.



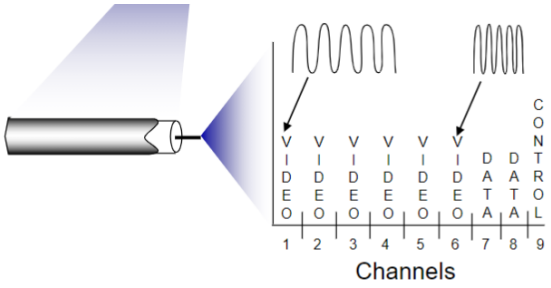
- Banda più larga, una parte dedicata alle comunicazioni telefoniche e una parte dedicata solo a Internet. Diventa possibile telefonare e navigare in Internet contemporaneamente.
- Modem e router sono stati integrati, diventando un unico oggetto.
- La lettera A sta per *asymmetric*: il bitrate disponibile in downstream è maggiore del bitrate disponibile in upstream (tecnologia pensata per modelli di tipo client-server, dove il client usufruisce di servizi e quindi il flusso è maggiore in downstream – cioè dall'esterno verso il client).

3.3.2.3 Rete via cavo

Tecnologia diffusa negli Stati Uniti, dove è pervasiva la televisione via cavo.



- Modem e router sono distinti: è il cavo che collega il modem, al router.
- Tecnologia asimmetrica come l'ADSL, ma il cavo è condiviso: un certo numero di utenti usufruiscono della stessa rete.
- Si hanno diversi canali (divisione delle frequenze, roba che vedremo a Comunicazioni numeriche): alcuni per il video, alcuni per l'audio, alcuni per i dati, e altri canali di controllo (verifica dell'integrità di quanto trasmesso).



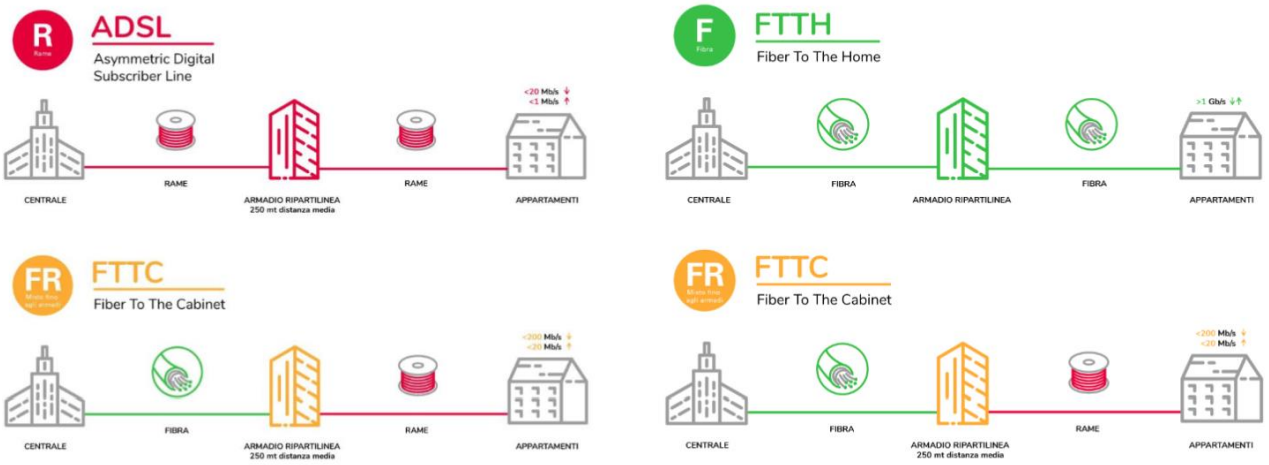
3.3.2.4 Fibra ottica

La fibra ottica è la tecnologia più recente, che nel tempo soppianterà la rete ADSL in Europa.

Non abbiamo più la trasmissione di un segnale elettrico per mezzo di fili di rame, ma di un segnale luminoso: la cosa ha i suoi vantaggi in quanto riduce disturbi dovuti a fenomeni naturali (non si hanno i disturbi tipici dei campi elettromagnetici) e aumenta drasticamente la velocità.

Distinguiamo due tipologie di infrastruttura:

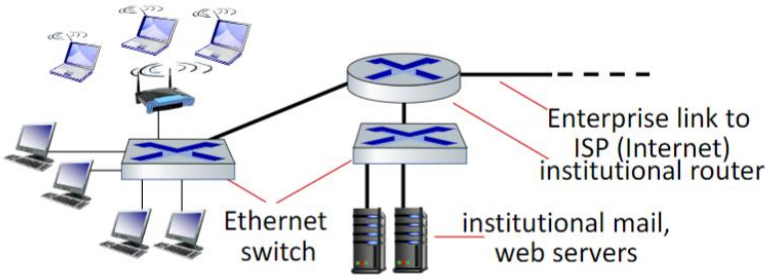
- ***fiber to the cabinet***, dove la fibra va dalla centrale alla centralina (dalla centralina all'abitazione rimangono i classici collegamenti in filo di rame);
- ***fiber to the home***, l'obiettivo finale dove la fibra collegherà direttamente la centrale con l'abitazione.



La differenza fondamentale sta nella velocità, minore nella prima tipologia (200 Mb/s in downstream e 20 Mb/s in upstream) a causa dei collegamenti finali in rame. Il futuro passaggio a *fiber to the home* (al momento utopistico per la dimensione degli interventi e i costi molto elevati) permetterà di innalzare la velocità oltre il Gb/s (si parla a tal proposito di *società del Gb*).

3.3.3 Reti di accesso aziendali (e residenziali): gestire un numero elevato di utenti

Le reti di accesso aziendale/istituzionale appartengono ad organizzazioni e sono usate per connettere alla rete i dispositivi utilizzati nel contesto dell'organizzazione. Il punto di partenza per la loro realizzazione sono le tecnologie viste prima, con la differenza che ci si trova a gestire un numero maggiore di dispositivi. Quanto diciamo vale anche nel contesto di una rete residenziale.



Si parla di una *Local Area Network* (LAN) all'interno della quale operano i dispositivi. Di queste LAN ci interessano soprattutto quelle in tecnologia Ethernet (*wired*) e quelle Wireless.

Tipologia di rete	**Velocità**
Reti di accesso Ethernet	100Mbps, 1Gbps, 10Gbps
Reti di accesso locale Wireless	11, 54, 450Mbps
Reti di accesso mobili	10Mbps

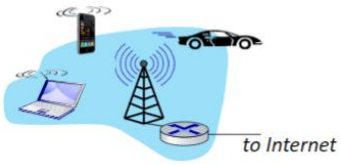
- **Reti di accesso wired: reti LAN, con particolare riferimento a Ethernet**
  Nelle LAN basate su tecnologia Ethernet si connettono numerosi dispositivi per mezzo di doppini di rame. Tutti i doppini di rame sono connessi a un dispositivo noto come switch: ogni dispositivo ha un canale dedicato, ergo non influisce sulle prestazioni percepite dal singolo host.

  Lo switch, o anche una rete di switch, è connessa a sua volta al router e quindi alla rete Internet.

- **Reti di accesso locale Wireless (*Wireless local area network, WLAN*)**
  Abbiamo uno o più access points presenti all'interno dell'edificio, a cui possono essere connessi i dispositivi. Gli access point sono, quasi sicuramente, connessi a una rete aziendale basata sulla tecnologia Ethernet precedentemente introdotta.

3.3.4 Reti di accesso mobile (*Wide-area cellular access networks*)

Discorso a parte sono le reti di accesso mobile, che rientrano nell'insieme delle reti Wireless (insieme alle WLAN), ma presentano differenze importanti. Sono reti messe a disposizione da un operatore telefonico, che pone la sua infrastruttura di rete in tutto il territorio dove opera.



La cosa supera sotto certi aspetti la rete istituzionale, dato che si amplia ulteriormente l'estensione geografica della rete, così come cambia il modo in cui si accede ad essa (Esempio: persone che accedono mentre sono in movimento in automobile o sul treno).

Tipicamente la velocità di queste reti è minore di quella delle precedenti (soprattutto rispetto a quelle wired), anche se reti mobili di nuova generazione (4G, 5G, …) colmeranno questo *gap* nel tempo. Il numero di host connessi può influire pesantemente sulla qualità della rete.

3.4 Link di collegamento (o mezzi trasmissivi)

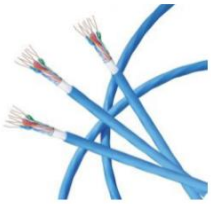
3.4.1 Introduzione

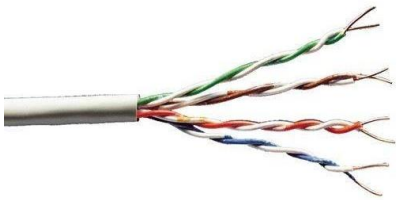
Con *link di collegamento* alludiamo a ciò che si ha tra trasmettitore e ricevitore. Il pacchetto (la sequenza di bit) viene posto nella forma di un segnale fisico (in generale un segnale elettrico, ma può essere anche segnale luminoso se si pensa alla fibra), segnale che viene fatto propagare in un mezzo trasmissivo (link). Distinguiamo link guidato da link non guidato:

- nel **link guidato** il segnale si propaga attraverso un mezzo fisico;
- nel **link non guidato** il segnale si propaga nell'etere, ergo la propagazione avviene in diverse direzioni.

3.4.2 Tecnologia: doppino telefonico (*Twisted Pair*)

La tecnologia più diffusa è quella del doppino telefonico. Abbiamo due fili di rame isolati e disposti a spira tra loro, e posti all'interno di un cavo. Il passaggio di corrente genera un campo elettromagnetico: da un punto di vista ideale costruire una spira significa minimizzare le interferenze dovute ad altri doppini presenti nelle vicinanze.





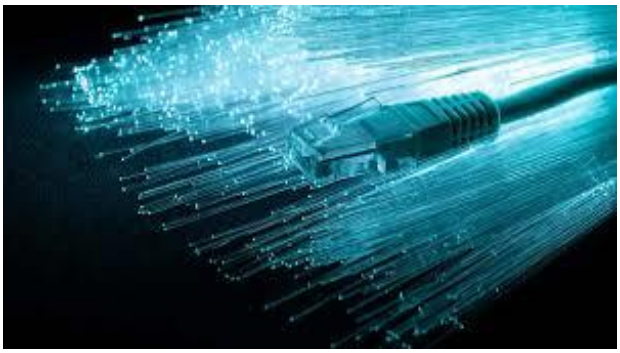
Si suppone che nel tempo questa tecnologia venga soppiantata dalla fibra ottica, ma ancora è molto utilizzata in quanto economica e in grado di fornire velocità trasmissive di 10Gbps. Non copre grandi distanze perché il segnala si degrade e perde in potenza (applicazione soprattutto nelle LAN).

3.4.3 Tecnologia: cavo coassiale (*coaxial cable*)

Nel cavo coassiale abbiamo sempre due conduttori di rame, ma questi sono concentrici e paralleli (e non più intrecciati). All'esterno abbiamo una guaina esterna che funge da isolante. Possibili canali diversi per mezzo della divisione per frequenza, con una banda di 100 Mbps per canale.

3.4.4 Tecnologia: fibra ottica (*fiber optic cable*)

Il mezzo consiste in fibre di vetro attraverso cui si propaga un segnale luminoso (e non più un segnale elettrico come nelle tecnologie precedenti). Il bitrate del mezzo fisico è molto più elevato: si parla di 10/100 Gbps, con un tasso di errore molto più basso rispetto ai mezzi tradizionali, visto l'assenza di disturbi elettromagnetici.



Maggiore affidabilità implica compiere distanze più elevate e quindi poter collocare a distanze maggiori i ripetitori.

3.4.5 Tecnologia: segnale wireless

In questo caso il mezzo fisico non è più wired: il segnale viene propagato nello spettro elettromagnetico. Emergono questioni come:

- la riflessione del segnale;
- l'ostruzione dovuta agli oggetti;
- l'ostruzione dovuta all'ambiente esterno.

Il bitrate è minore (si rimane nei Mbps). Il ritardo in alcune circostanze potrebbe essere significativo, influendo negativamente sule comunicazioni interattive.

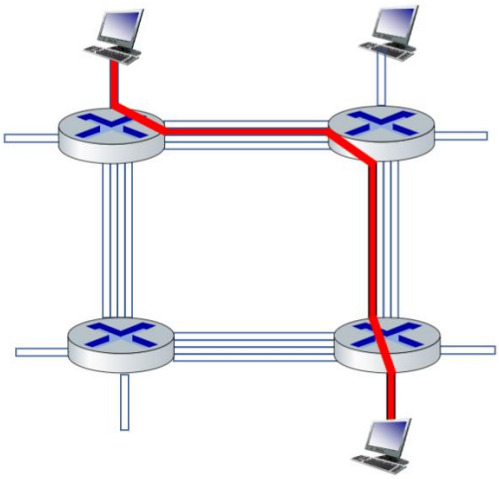
3.5 Nucleo della rete (*network core*)

3.5.1 Approcci nella gestione dei pacchetti nel core

Vogliamo trasmettere i dati da un sorgente a un destinatario, passando dal nucleo della rete. Come avviene questo trasferimento? Abbiamo due approcci possibili:

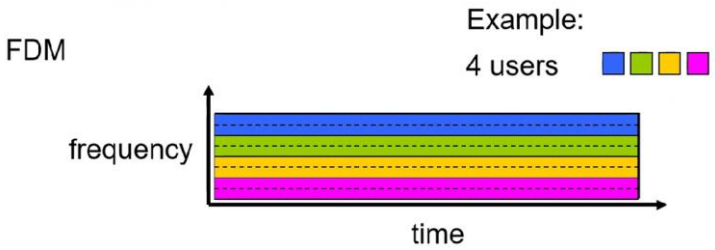
- *circuit-switching*, e
- *packet-switching*.

3.5.1.1 Circuit switching

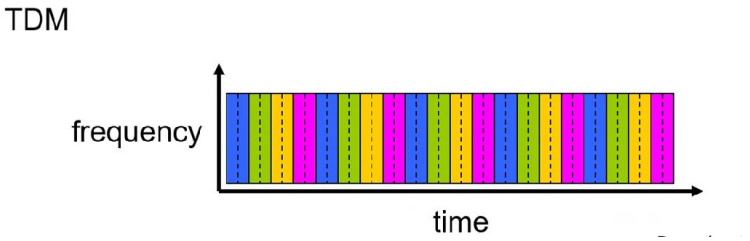


Approccio adottato nella telefonia tradizionale: si stabilisce un circuito tra il chiamante e il chiamato, che permane per tutta la durata della comunicazione. Nella telefonia tradizionale il circuito era fisico, stabilito da un operatore umano per mezzo del movimento di fili.

- Il circuito deve essere attivato al momento del lancio della chiamata e disattivato alla chiusura della stessa: si ha una fase di *call setup*.
- Le prestazioni sono garantite in quanto il circuito è dedicato per chiamante e chiamato.
- Nel contesto della rete il circuito non è fisico in senso letterale.
    - Lo si ottiene per mezzo della tecnica della divisione delle frequenze (per avere un'idea più chiara si pensi a quando ci sintonizziamo sulle radio, dobbiamo indicare una particolare frequenza per ascoltare una particolare emittente).



- Una via alternativa può essere una suddivisione temporale: si divide il tempo in slot, ciascun slot è dedicato a una particolare coppia di utenti.



Si consideri che il costo è correlato al tempo di utilizzo: segue che l'approccio del circuit-switching è valido esclusivamente in un contesto di comunicazione continua.

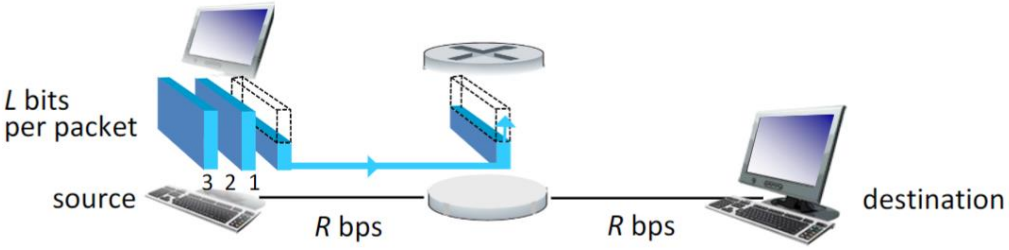
- Nel contesto della telefonia classica il metodo è valido perché la telefonata è lanciata esclusivamente quando si deve parlare, e si aggancia quando non si ha più nulla da dire.
- Nel contesto di Internet ciò non va più bene: l'attività dell'utente è caratterizzata da un'alternanza tra attività intensa e momenti di pausa. Correlare il costo al tempo di utilizzo significa pagare anche per i momenti in cui non si naviga in Internet.

Si consideri anche che stabilire un circuito significa precludere l'accesso a soggetti diversi dal chiamante e dal chiamato: le prestazioni precipitano drasticamente se i due utenti non comunicano costantemente (si ha uno spazio non utilizzato, che potrebbe essere utilizzato da altri soggetti).

3.5.1.2 Packet switching

Possiamo migliorare le prestazioni ponendo l'approccio del *packet-switching*. I dati sono trasmessi dalle sorgenti di traffico nella forma di pacchetti, che abbiamo già definito precedentemente.

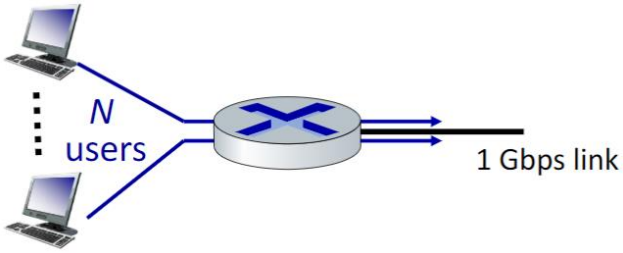
- **Link condiviso**.
  Si ha un link messo a disposizione di un certo numero di utenti (risorse condivise tra più utenti, e non più riservate a una coppia): si parla di *multiplexing statistico*[3]. La banda utilizzata è quella a disposizione del link.
- **Traffico adatto all'approccio**.
  Adeguato per il traffico intermittente tra calcolatori, meno adeguato quando si richiede un traffico continuo (si pensi allo streaming): nonostante questo ci si è mossi verso l'approccio packet-switching anche per traffico voce e video (si sono introdotte correzioni per minimizzare il ritardo, inoltre la popolarità di Internet ha fatto il resto).
- ***store and forward***.
  Il packet-switching si basa sull'approccio *store and forward*: in una comunicazione basata su pacchetti il nodo riceve e memorizza l'intero pacchetto prima di inoltrarlo verso altri nodi.



[3] Da capterra.it: *Il multiplexing statistico è una tecnica usata per l'assegnazione di slot di canali nella trasmissione di rete. I canali di trasmissione sono assegnati in modo automatico in base alle necessità*. Si dice statistico perché ci si basa sul fatto statistico che quando una sorgente genera traffico non faranno la stessa cosa altre sorgenti.

3.5.1.3 Confronto con circuit switching e costo della maggiore efficienza

Supponiamo di avere N utenti che vogliono trasmettere traffico.



Gli utenti sono intermittenti, cioè generano traffico ma non continuità (sono attivi solo per il 10% del tempo). Quando un utente è attivo genera 100 kb/s. Il traffico di tutti questi utenti è convogliato da un router verso un unico link di uscita, che ha capacità di 1Mbps[4].

Ci chiediamo: quanti utenti possiamo connettere a questo dispositivo, assumendo come validi i comportamenti descritti?

- Nell'approccio *circuit-switching* abbiamo al massimo 10 utenti (si divide 1Mbps per 100 kbps)
- Nell'approccio *packet-switching* si sfrutta il multiplexing statistico: bisogna vedere come si comportano gli utenti, ma logicamente pensiamo che si possa mettere un numero maggiore di utenti. È dimostrabile che fino a 35 utenti la probabilità che si abbiano più di 10 utenti attivi contemporaneamente (che significherebbe perdere pacchetti) è inferiore allo 0.0004 (trascurabile).

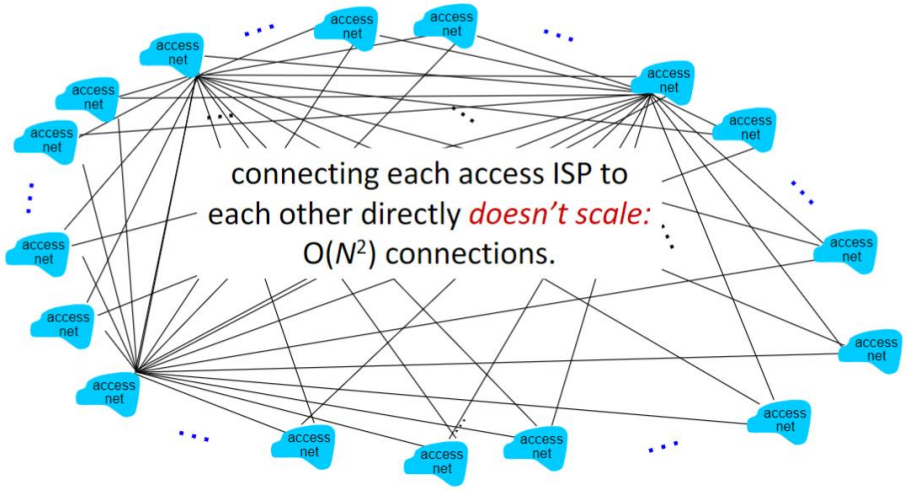
Morale della favola: packet-switching è vincente! Permette un aumento del traffico, inoltre non è presente la call-setup dato che non è un circuito da allocare. Si consideri, tuttavia, che l'efficienza ha un costo:

- **Congestione**.
  Una differenza elevata di velocità di ingresso e velocità di uscita genera congestionamento. Gestiamo ciò introducendo una coda: fare ciò significa creare un buffer dove salvare i pacchetti in attesa di essere trasmessi sul link. **Se un pacchetto rimane in coda si genera ritardo**!
- **Memoria del buffer**.
  Lo spazio riservato al buffer non è infinito: se il congestionamento persiste la memoria verrà esaurita, e a quel punto non sarà più possibile gestire altri pacchetti. Due strade possibili (in ogni caso avremo perdita di pacchetti):
  - sovrascrivere un pacchetto già presente in coda;
  - rinunciare a porre in coda l'ultimo pacchetto considerato.

3.5.2 Struttura di Internet

3.5.2.1 Idea di partenza (sbagliata)

Supponiamo di avere milioni di reti di accesso: ogni rete ha un router verso l'esterno, e si ha un collegamento dedicato verso qualunque rete di accesso nel mondo.



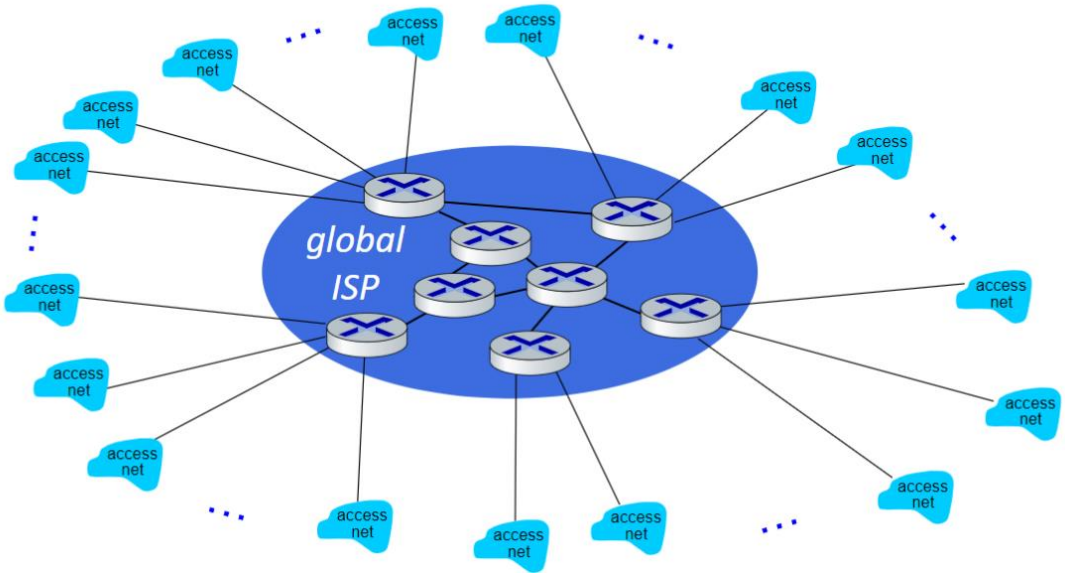
[4] Mbps e kb/s dei telecomunicazionisti: kb, ad esempio, vale 1000 e non 1024.

La soluzione non è adatta: non scala ed è molto costosa (il numero di link dedicati da realizzare è spropositato. Si consideri che dati N nodi questo dovrà essere collegato ad $N-1$ nodi: segue il seguente numero di connessioni

$$N(N-1) = N^2 - N \Longrightarrow O(N^2 - N) \longrightarrow O(N^2)$$

3.5.2.2 Proposta successiva: rete di routers globale

L'idea successiva è quella di introdurre una rete di routers attraverso cui viaggiano i nostri pacchetti.

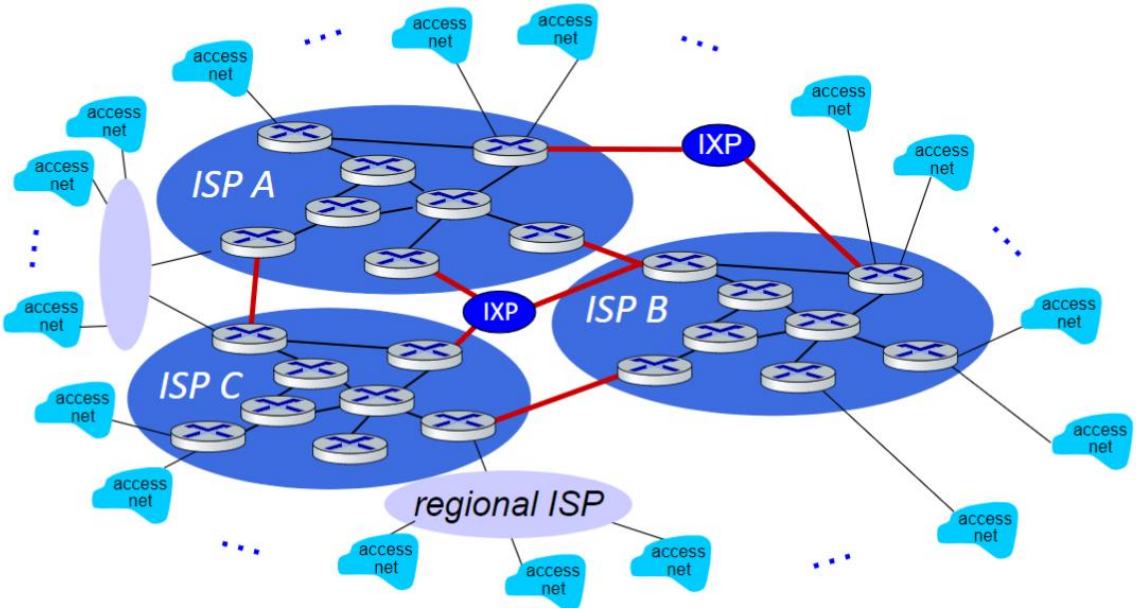


La rete è gestita da un Internet Service Provider, che fornisce i servizi della rete a un cliente dietro compenso, e alle condizioni da lui stabilite. La soluzione è migliore di quella iniziale, ma presenta problemi:

- politici, in quanto affidare l'intera infrastruttura a un unico provider significa cedere il controllo dell'intero Internet a una nazione (e da questo possono emergere problemi come quello della censura);
- economici, una rete globale unica è molto più costosa
- tecnici, si ha un *single point of failure* (in caso di problemi viene meno l'intera infrastruttura).

3.5.2.3 Idea finale: reti di routers con più ISP

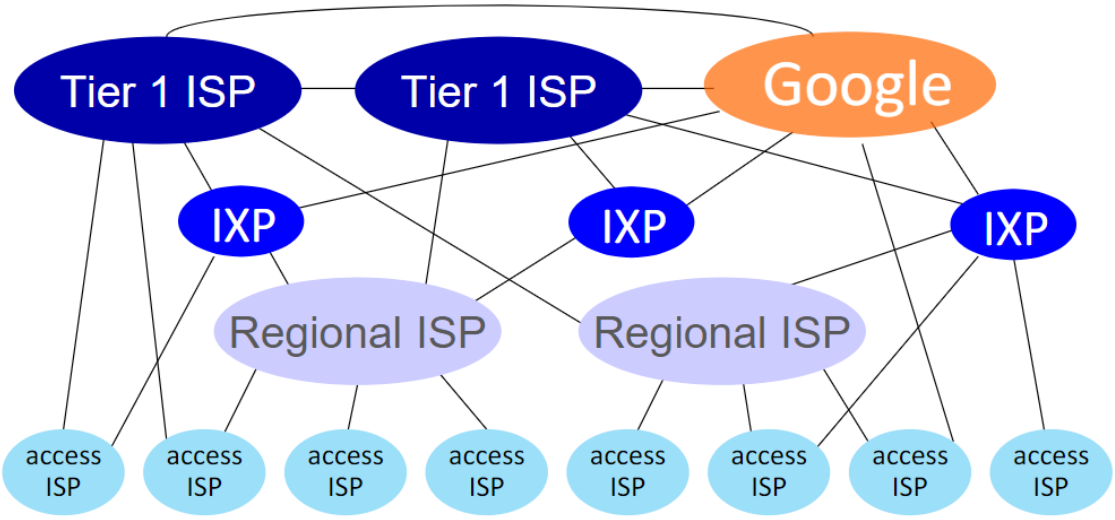
La proposta finale è quella di una rete di reti gestite da più Internet Service Provider.



In questo contesto abbiamo:

- ISP di livello 1 (*tier-1*), con copertura nazionale e/o internazionale;
- ISP di livello 2 (*tier-2*, o *Regional ISP*), i cui servizi si basano sugli ISP di livello 1;

- *Content Provider Networks*, fornitori di servizi che collegano i loro data center ad Internet, bypassando ISP di livello 1 e Regional ISP;
- *Internet Exchange Points* (IXP) che collegano le varie reti.



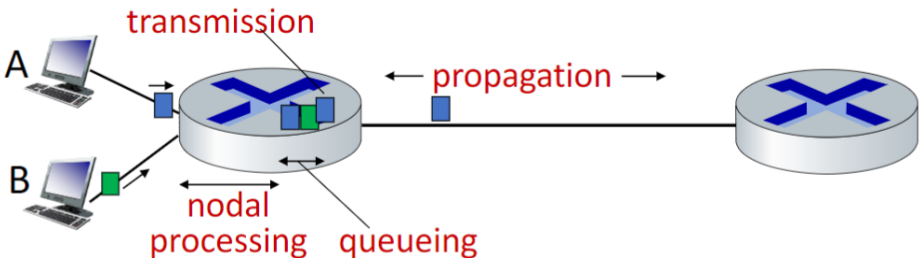
3.6 Performance della rete

3.6.1 Ritardo (*delay*)

3.6.1.1 Trasmissione da un router sorgente a un router destinatario

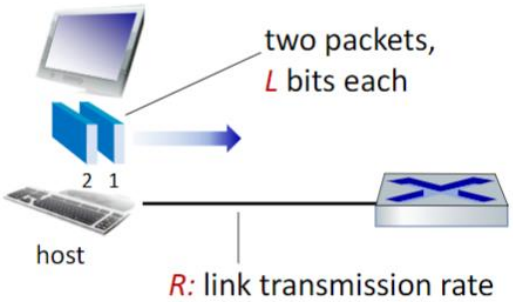
Quali sono le cause principali di ritardo nella trasmissione di un pacchetto?

1. La memorizzazione del pacchetto nel buffer (coda in entrata).
2. La verifica dell'integrità del pacchetto.
3. La consultazione della tabella di forwarding per decidere il link verso cui indirizzare il pacchetto.
4. Posizionamento del pacchetto su buffer in uscita (coda in uscita).



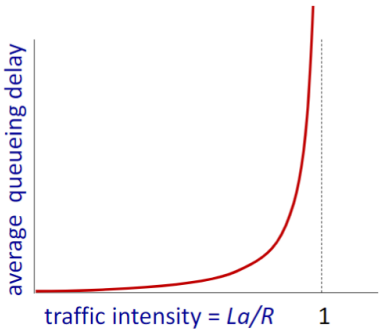
Possiamo definire il ritardo come la somma dei seguenti contributi:

$$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$$

Contributi al ritardo	
d_{proc}	Ritardo di *processing*, dovuto all'elaborazione del pacchetto nel nodo. L'applicazione prepara i pacchetti conoscendone la lunghezza e ponendo al suo interno: - Il contenuto del messaggio, o parte del contenuto nel caso in cui il messaggio venga diviso in più pacchetti; - bits informativi sul pacchetto (poniamo, ad esempio, la destinazione del pacchetto); - bits per la verifica dell'integrità del messaggio. Si determina anche il link di uscita consultando la tabella di forwarding. Il ritardo è costante e in generale trascurabile, quantificabile in valori $< msec$.
d_{queue}	Ritardo di *queueing* dovuto al posizionamento del pacchetto in coda. Non è determinabile a priori, dipende dal posizionamento del pacchetto in coda e dal congestionamento della rete.
d_{trans}	Ritardo di trasmissione del pacchetto, dovuto alla generazione del segnale fisico posto sul link di comunicazione (il passaggio dal buffer della coda di uscita al link di comunicazione). Come si calcola? Consideriamo un certo numero di pacchetti: - ciascun pacchetto ha dimensione L bits; - il *rate di trasmissione* del link di comunicazione è R bits/sec.  Il tempo necessario per trasmettere un pacchetto di L bit nei link è il seguente rapporto $d_{trans} = \text{Packet transmission delay} = \frac{\text{L (bits)}}{\text{R (bits/sec)}}$ Fissato R, il ritardo varia esclusivamente in funzione della dimensione del pacchetto (L).
d_{prop}	Ritardo dovuto alla propagazione del segnale fisico sul link di comunicazione. Contributo molto piccolo se ci troviamo in reti locali, diverso da d_{trans} $d_{prop} = \text{Propagation delay} = \frac{d}{s}$ Dove d è la lunghezza del link di comunicazione, mentre s è la relativa velocità di propagazione.

3.6.1.2 *Studio del ritardo dei pacchetti*

Proviamo a studiare il ritardo dei pacchetti.



- Abbiamo sempre L ed R, rispettivamente la dimensione di ogni singolo pacchetto e la banda del link di comunicazione.
- Abbiamo anche a, che consiste nel numero di pacchetti al secondo.

Otteniamo che l'intensità del traffico, cioè il numero di bit al secondo, consiste nel seguente calcolo

$$\text{Traffic intensity} = \frac{L \cdot a}{R}$$

Aumentiamo l'intensità del traffico aumentando a (numero di pacchetti al secondo): il ritardo di accodamento non è più nullo, ma inizialmente non ha valori significativi. A un certo punto aumenta in maniera vertiginosa e tende a infinito (si veda il grafico a lato).

- Se l'intensità del traffico è << 1 (prossima allo zero) allora non c'è attesa: la velocità in entrata è molto minore rispetto a quella di uscita.
- Se l'intensità è prossima ad 1 il ritardo diventa significativo.
- Se l'intensità del traffico è > 1 allora il lavoro da gestire è vertiginosamente grande rispetto alla banda del link di comunicazione.

3.6.1.3 *Trasmissione end-to-end (percorso di N router)*

Supponiamo che un nodo sorgente voglia trasmettere un pacchetto a un nodo destinatario, e che tra questi vi siano N router. Il pacchetto dovrà essere trasmesso $N + 1$ volte! Otteniamo:

$$d_{e2e} = (N + 1) \cdot \left(d_{proc} + d_{queue} + d_{trans} + d_{prop}\right)$$

3.6.2 Perdita di pacchetti (*loss*)

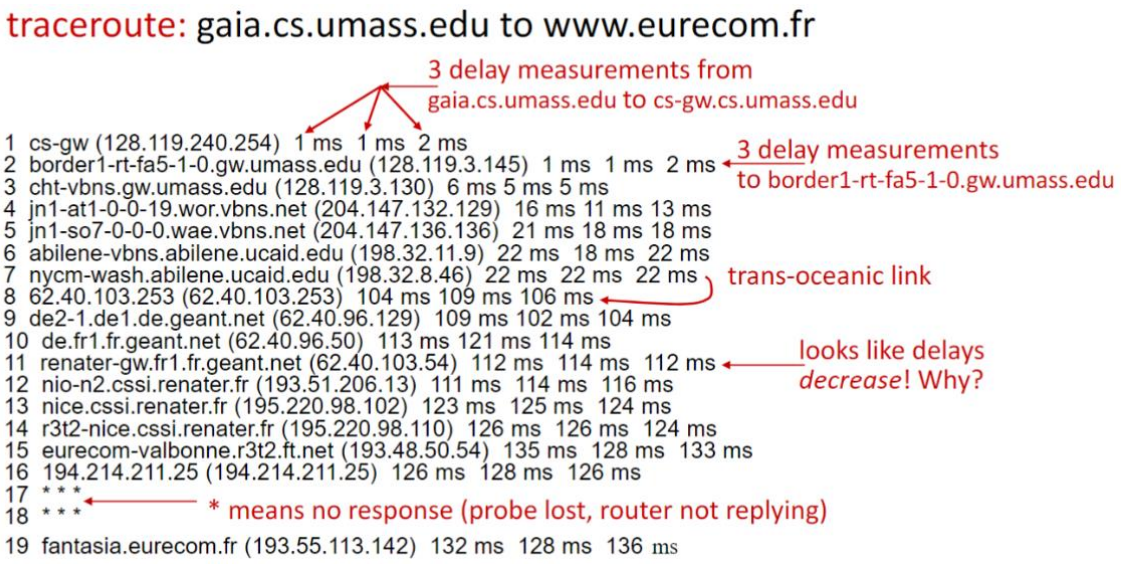
Ribadiamo gli approcci possibili per gestire il buffer di una coda pieno:

- buttare via il pacchetto arrivato;
- buttare via un pacchetto già presente in memoria ed ospitare quello nuovo.

È così tragico perdere un pacchetto? No, è più una perdita di tempo perché l'host dovrà inviare nuovamente il pacchetto (si ricomincia da zero, aumenta il consumo di energia ma soprattutto il traffico e il ritardo). È comunque un fenomeno da evitare.

3.6.3 Comando per la misurazione dei ritardi e delle perdite: *traceroute*

Il comando *traceroute* misura il ritardo nel passaggio dei pacchetti da sorgente a destinazione.



Per ogni router:

- si inviano tre pacchetti di prova dall'host;
- i pacchetti ritorneranno al sender;
- si fanno stime sul ritardo.

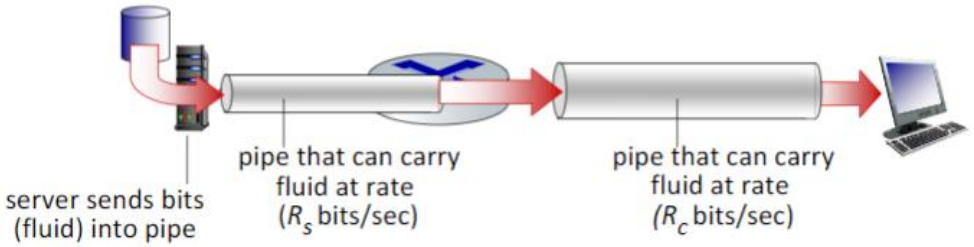
Osservazione: il percorso intermedio potrebbe cambiare durante i tre invii.

3.6.4 Numero di bit trasmessi al secondo (*throughput*)

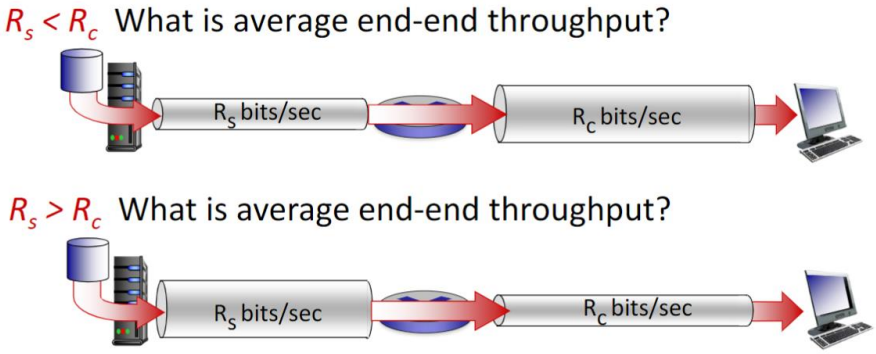
Il *throughput* consiste nel numero di bit trasmessi dal sender al ricevitore per unità di tempo. La cosa si distingue dalla rate/capacità del link di comunicazione:

- Il percorso è caratterizzato da tratti aventi capacità diverse.
- La capacità di un particolare link può portare all'effetto collo di bottiglia (bottleneck link).

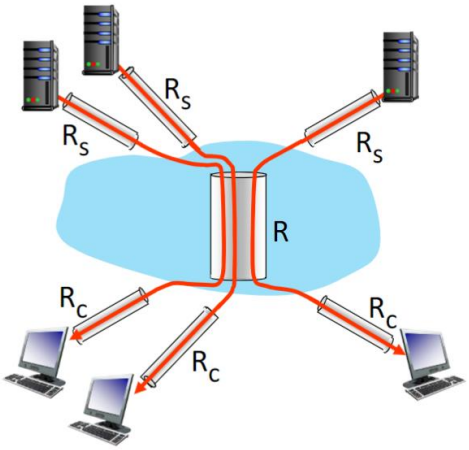
Il throughput è di due tipi: istantaneo (in un particolare momento temporale) e medio (si studia il rate su un periodo più lungo). Supponiamo di avere un server che trasmette dei bit a un client



Quanti bit riceverà il client? Il minimo tra i due rate, si ha l'effetto collo di bottiglia citato prima.



Nel caso in cui si consideri Internet si deve suppore che questo possa provocare l'effetto collo di bottiglia.



La banda di Internet è divisa tra gli utenti: supponiamo in modo del tutto teorico e lontano dalla realtà che vi siano dieci utenti, segue che il rate di Internet è $R/10$. Il collo di bottiglia è $min\{R_C, R_S, R/10\}$

3.7 Modello stratificato (*protocol layers*)

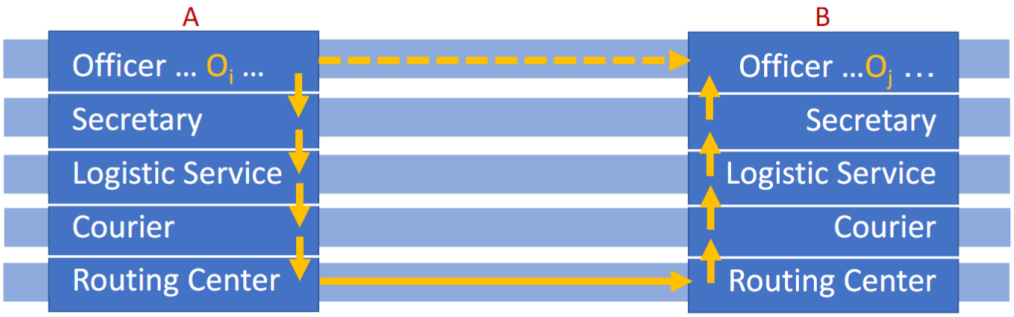
3.7.1 Regole base del modello stratificato

Il sistema di rete è organizzato per mezzo di componenti: hosts, routers, links, applicazioni, protocolli, hardware e software. La complessità del sistema richiede un'organizzazione dello stesso secondo un *modello stratificato*. Le regole base sono le seguenti:

- ogni livello fa qualcosa;
- ogni livello fa qualcosa basandosi su quanto già fatto in altri livelli.

3.7.2 Esempio introduttivo: organizzazione di un sistema postale

Facciamo un esempio immaginando l'organizzazione di un sistema postale di società di trasporti private. Abbiamo una società A e una società B organizzate per mezzo di un modello a strati.



- Il livello Officer raccoglie gli alti dirigenti di una società.
  Supponiamo che l'officer O_i della società A voglia trasmettere un messaggio all'officer O_j della società B.
- Il dirigente si limita a mettere la busta nella cassetta della sua scrivania. Il lavoro del dirigente O_i è finito, non deve fare altro. L'idea che ha il dirigente è che il messaggio arrivi direttamente al destinatario, *un po' come per magia*.
- In realtà succedono altre cose:
  - Il segretario di un particolare dipartimento raccoglie i contenuti delle cassette dei relativi dirigenti e mettono tutto in un unico contenitore.
  - I contenitori di tutti i dipartimenti saranno portati in portineria, e costituiranno un unico oggetto in movimento.
  - A un certo punto passerà un corriere che porterà il tutto in un centro di smistamento

  A questo punto c'è il trasporto, che ci porta dal centro di smistamento (Routing Center) della società A al centro di smistamento della società B.
  - Il corriere della società recupera il materiale dal centro di smistamento e lo porterà nella sede della società B.
  - Il materiale viene diviso per dipartimenti: il segretario di ogni dipartimento riceverà il materiale relativo al suo dipartimento.
  - Il segretario del dipartimento conclude l'opera mettendo le lettere nelle cassette dei dirigenti.

  In questa opera di smistamento è stato portato a destinazione il messaggio citato all'inizio dell'esempio, dal dirigente O_i della società A al dirigente O_j della società B.

3.7.3 Vantaggi dell'organizzazione a strati

I vantaggi sono diversi:

- Possibilità di identificare responsabilità all'interno di ogni strato, e le relazioni tra le varie componenti dell'intero sistema.
- Organizzazione modulare, rende agevole la manutenzione e l'aggiornamento dell'intero sistema.
  - Supponiamo di voler passare da un livello del modello a strati a uno inferiore.
  - Il livello inferiore presenta un'interfaccia, ossia come è visto dal livello superiore.
  - Il passaggio avviene senza tenere a mente le procedure adottate in ogni livello:
    - Il livello superiore passa le informazioni a livello inferiore e non sa cosa succede nel livello inferiore;

- Il livello inferiore riceve le informazioni dal livello superiore e non sa cosa è successo nel livello superiore.

Questo rende possibile intervenire in un singolo stratto, in quanto modifiche in esso non renderanno necessarie modifiche negli altri strati (a meno che non si vada ad alterare le relazioni tra gli strati, cioè le interfacce).

3.7.4 Modello a strati in Internet

3.7.4.1 Stratificazione dei protocolli

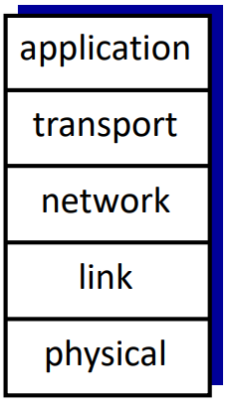
Quello che noi facciamo è attuare la stratificazione dei protocolli. Ciascun livello può essere implementato in uno dei seguenti modi:

- via hardware;
- via software;
- combinazione di hardware e software.

Complessivamente si parla di pila di protocolli (detta in inglese *protocol stack*).

3.7.4.2 Modello attuale con spiegazione dei livelli

Il modello di pila protocollare attualmente in uso prevede cinque livelli:



- **Livello applicazione**
  Nel livello applicativo hanno sede le applicazioni di rete: web, posta elettronica, file transfer, videogiochi. In questo livello risultano implementati protocolli alla base del funzionamento delle applicazioni: HTTP (per il web), SMTP (per la posta elettronica), FTP (per il file transfer), …. In questo livello si genera il messaggio da inviare ad altri host.
- **Livello trasporto**
  Il livello di trasporto consiste in quell'insieme di servizi che vengono offerti direttamente alle applicazioni per permettere la comunicazione tra host, precisamente la comunicazione tra due processi (di host diversi, il passaggio ulteriore rispetto alla comunicazione tra processi vista a Calcolatori elettronici). I protocolli principali sono TCP e UDP, che hanno lo stesso scopo ma presentano differenze rilevanti.
- **Livello network**
  Il livello di rete gestisce la logica con cui i pacchetti vengono trasmessi all'interno della rete globale. Si parla, non a caso, di algoritmi di instradamento.
- **Livello link (o datalink)**
  Si gestisce l'instradamento dei pacchetti, dall'host al mezzo fisico dove avverrà effettivamente il trasporto.
- **Livello fisico**
  Il livello fisico, il più basso, consiste nel mezzo attraverso cui si ha la trasmissione vera e propria (solo qua possiamo dire di non essere nell'host). In questo livello il messaggio confezionato a livello applicativo è visto come un mero segnale fisico trasmesso per mezzo di un'adeguata infrastruttura.

3.7.4.3 Passaggio dei pacchetti tra i vari livelli: incapsulamento

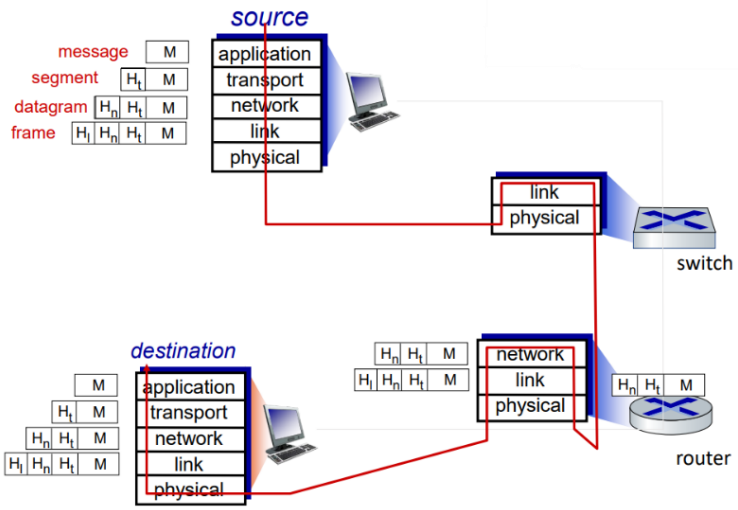
Pacchetto è termine generico con cui alludiamo a un blocco di bit, termine valido in qualunque livello della pila protocollare. A seconda del livello a cui ci troviamo il pacchetto può assumere un nome più specifico:

Livello applicazione	*message*
Livello trasporto	*segment*
Livello network	*datagram*
Livello datalink	*frame*

Se non si vuole sbagliare pacchetto è il termine più sicuro, salvo quei casi in cui potrebbe essere chiesto il nome specifico (cosa a cui potete rispondere solo se sapete a che livello della pila protocollare ci si trova, se non sapete come si chiama il pacchetto allora si presume che non si sappia il livello a cui ci si trova).

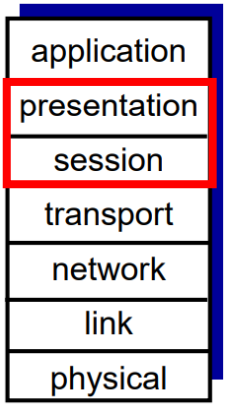
Si consideri che ad ogni livello il pacchetto viene manipolato, includendo nuove informazioni utili per quel livello. Si parla di incapsulamento: nella pila dell'host source il livello inferiore incapsula il contenuto fornito dal livello superiore.

Nella pila del destinatario il pacchetto viene "scomposto": il livello applicazione riceverà il messaggio nel modo in cui è stato inviato dall'host source (senza le informazioni aggiunte dai livelli inferiori).



3.7.4.4 Riferimento al passato: modello ISO/OSI

Il primo modello di pila protocollare risale agli anni 70 ed è noto come modello ISO/OSI. Questo modello è nato prima dell'avvento di Internet e prevedeva due ulteriori livelli: il livello presentation (interpretazione dei dati scambiati – per esempio criptaggio e decriptaggio) e il livello session (??).



Il modello ha avuto successo accademico ed è stato insegnato per molti anni, ma è venuto meno soprattutto dopo l'avvento di Internet.

4 APPLICAZIONI DI RETE (NETWORK APPLICATIONS)

4.1 Introduzione

4.1.1 Livello più alto della pila protocollare

Pila protocollare
Appplication ***messages***
Transport *segments*
Network *datagrams*
Link / Datalink *frames*
Physical

Le applicazioni girano sugli hosts, ricordiamo. Abbiamo introdotto nel capitolo precedente la divisione a strati nel contesto della rete. Integriamo osservando che:

- sui dispositivi nella periferia della rete (hosts in primis) è necessario sviluppare l'intera pila;
- nei dispositivi intermedi ci si limita a sviluppare i due livelli inferiori: *physical* e *link*.

Le applicazioni di rete di cui parleremo in questo capitolo sono presenti solo dove è stata sviluppata l'intera pila (quindi dove è presente il livello applicazione).

4.1.2 Processi studiati a Calcolatori elettronici: differenze

Ricordare il concetto di processo da calcolatori. Normalmente due processi su uno stesso host possono comunicare in uno dei seguenti modi:

- scambio di messaggi;
- *inter-process communication*.

La novità è che abbiamo un processo client e un processo server che si trovano in host diversi, e comunicano tra di loro:

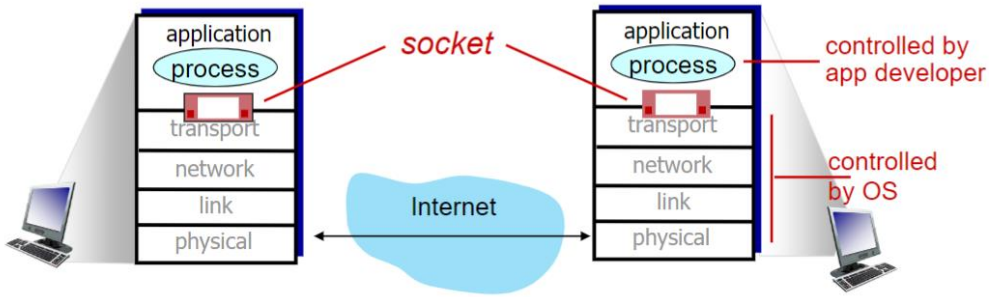
- Il processo client si trova sull'host client;
- Il processo server si trova sull'host server.

Il meccanismo è più complicato perché non abbiamo più una memoria condivisa: i contenuti scambiati devono viaggiare attraverso la rete.

4.1.3 Sockets

4.1.3.1 Idea di base: collocazione nella pila e utilità

Il socket (letteralmente presa, si pensi alla presa del telefono) può essere immaginato come "una stazione, una cassetta della posta". È un'astrazione, *un qualcosa che nella realtà non esiste ma che si fa credere al programmatore che esista*[5].



Dalla figura osserviamo come il socket sia un'interfaccia a cavallo tra lo strato *application* e lo strato *transport*: possiamo parlare a tutti gli effetti di API.

Per avere un'idea più chiara di cosa sia il socket consideriamo un edificio con diverse abitazioni, e quindi diverse porte. Supponiamo che l'edificio sia *Via Diotisalvi, 2, Pisa*.

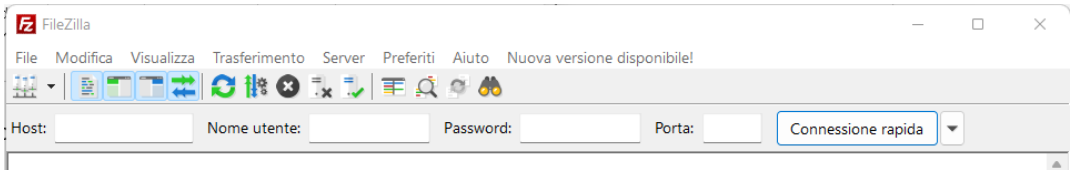
- Voglio inviare una lettera (posta tradizionale) a uno degli abitanti dell'edificio.

[5] Giusto per fare un confronto: il file è un concetto astratto che fa comodo solo all'utente. Negli strati inferiori non si parla più di file, ma di sequenze di bit.

- Dire solo l'indirizzo *Via Diotisalvi, 2, Pisa* è insufficiente: questo perché noi non ci basta inviare la lettera all'edificio, ma ad un soggetto all'interno dell'edificio.
- Segue che dobbiamo indicare il numero della porta e non solo l'indirizzo dell'edificio.

Vale la stessa cosa nel contesto delle reti! Abbiamo un host dove sono eseguiti centinaia di processi.

- Un processo in un host differente da quello citato vuole trasmettere un messaggio a uno dei processi, tra i centinaia detti.
- Abbiamo già detto che l'host è identificato da un numero di indirizzi IP pari alle schede di rete.
- Dire il solo indirizzo IP è insufficiente, perché nel trasmettere il messaggio dobbiamo indicare anche quale processo dovrà riceverlo!



Filezilla richiede di indicare la porta per poter stabilire una connessione col server. Se non lo si indica FileZilla adotta quella che per convenzione è la porta utilizzata dai server per stabilire queste tipologie di comunicazioni (la porta 21)

4.1.3.2 Operazioni possibili

I sockets sono gestiti nel codice delle applicazioni per mezzo di chiamate di funzione, che permettono di svolgere le seguenti operazioni:

- *send*, trasmissione di pacchetti verso destinatari;
- *receive*, verifica della presenza di nuovi pacchetti (inviati da mittenti per mezzo del comando precedente).

Ovviamente la possibilità di svolgere queste operazioni presume che, precedentemente, sia stata stabilità un'associazione tra il programma e il *socket*.

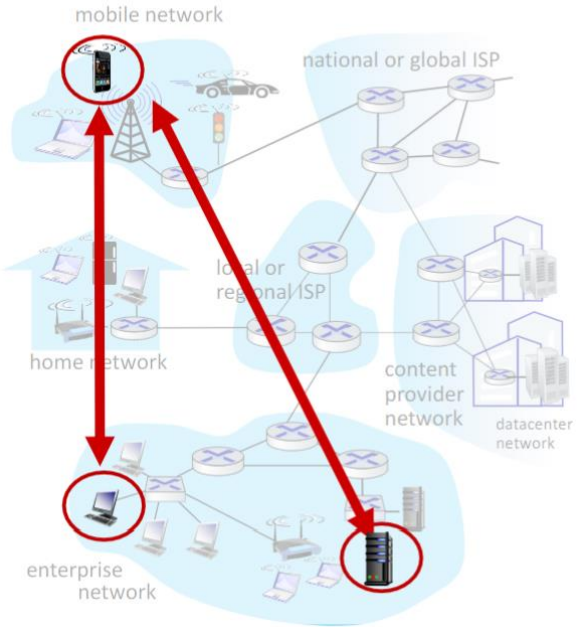
Le primitive richiamano moltissimo il paradigma client-server, poiché i socket risalgono agli anni 80 (il paradigma P2P nasce durante gli anni 90).

4.1.4 Modelli su cui può basarsi un'applicazione

4.1.4.1 Modello client-server

Consideriamo il modello client-server, dove distinguiamo due tipologie di processi:

- il processo server che offre un servizio;
- i processi client che possono collegarsi al processo server e usufruire dei servizi.



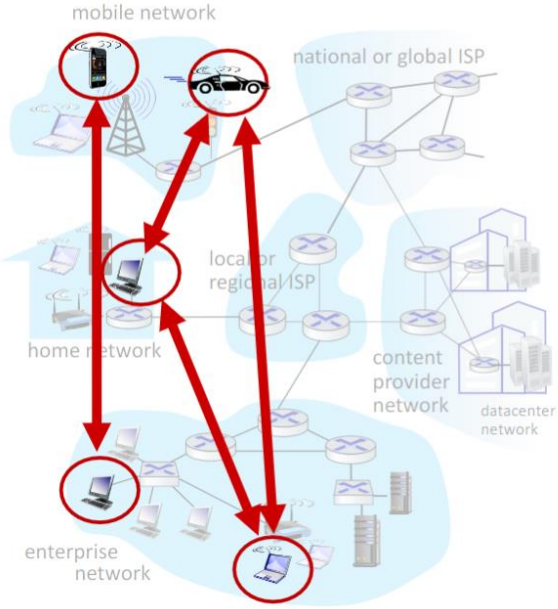
Vi è una chiara distinzione dei ruoli: chi è client (server) si comporta sempre come client (server).

- **Host server**.
  - Il server deve rimanere sempre accesso.
  - Il server deve avere un indirizzo IP permanente (come fa l'utente a stabilire la comunicazione frequentemente se l'indirizzo IP cambia di continuo?).
  - Necessarie risorse per poter interloquire con tutti i client che richiedono l'accesso ai servizi. In particolare, quello che viene fatto molto spesso è la creazione di data centers, cioè l'utilizzo di più hosts per la creazione di un server virtuale più potente e in grado di gestire tutte le richieste (scalabilità).

- **Host client**.
    - Il client è la macchina con cui l'utente si connette al server per usufruire del servizio.
    - Non è necessario tenere il client acceso h24.
    - L'indirizzo IP può essere dinamico senza problemi.
    - Non è necessario avere un client estremamente potente.
    - I client non comunicano tra di loro, dialogano solo col server.

4.1.4.2 Modello peer-to-peer (P2P)

Nel modello peer-to-peer viene meno la divisione netta dei ruoli, tipica del modello precedente.



- **Comportamento fluido degli hosts**.
  Abbiamo dei *peer*, che possono avere comportamenti diversi in momenti diversi: si comportano da client se richiedono servizi, si comportano da server se offrono servizi.
- **Assenza di un'infrastruttura di rete**.
  Non si ha un data center come nel caso del modello client-server: questo perché le risorse non sono più concentrate in un server virtuale, ma distribuite sui peer, e quindi sugli host che costituiscono la rete.
- Non è necessario che i peer siano accesi h24.
- Non è necessario che i peer abbiano indirizzi IP permanenti.

4.1.5 Caratteristiche dei protocolli a livello applicazione

4.1.5.1 Elementi definiti dal protocollo

Le applicazioni sono regolate da protocolli. Questi includono le seguenti informazioni:

- la tipologia del messaggio scambiato
  (se è una richiesta di un client o la risposta di un server, ad esempio)
- la sintassi del messaggio, cioè la sua struttura
  (tra le cose includiamo pure la lunghezza del messaggio e dei vari campi che lo caratterizzano)
- la semantica del messaggio
  (il significato delle informazioni contenute in ciascun campo)
- regole su quando e come i processi si scambiano i messaggi
  (l'ordine dei messaggi, ad esempio)

4.1.5.2 Protocolli aperti e protocolli proprietari

Si distinguono:

- **Protocolli aperti**.
  Sono protocolli definiti da un ente secondo i modi già descritti, ma anche protocolli non validati e diventati rilevanti per la loro popolarità. Ogni persona ha accesso alla definizione del protocollo, favorendo così l'interoperabilità.
- **Protocolli proprietari**.
  Sono protocolli definiti da un soggetto privato. Il funzionamento del protocollo è ignoto: l'utente vede una scatola chiusa di cui non conosce il contenuto, ma sa cosa entra e cosa esce.

4.2 Protocolli a livello di trasporto: anticipazioni

4.2.1 Premessa: caratteristiche di applicazioni con trasporto dei dati

Quando parliamo di applicazioni dobbiamo tenere a mente le seguenti caratteristiche, che ci porteranno a scegliere un particolare protocollo di trasporto.

Pila protocollare
Appplication *messages*
Transport *segments*
Network *datagrams*
Link / Datalink *frames*
Physical

- ***Reliability* (integrità dei dati)**
  Un'applicazione con questa proprietà richiede un trasferimento affidabile di ogni singolo bit: questo significa dire che i bit ricevuti a destinazione devono essere quelli inviati dal mittente, senza alterazione. Non si parla solo di variazione del contenuto dei pacchetti, ma anche di variazione dell'ordine dei pacchetti!
  - Applicazioni con questa proprietà sono dette *loss-intolerant* (Esempio: file transfer),
  - Applicazioni che tollerano perdite di dati sono dette *loss-tolerant* (Esempio: streaming).
- ***Timing***
  Alcune applicazioni sono effettive solo se il ritardo nella trasmissione di contenuti è molto basso (Esempio: streaming, se una persona fa una domanda a un altro soggetto questo non può ricevere la domanda dieci minuti dopo).
- ***Throughput* minimo**
  Alcune applicazioni bisogno di un throughput minimo per essere effettive, cioè hanno bisogno di ricevere un minimo di bit al secondo per poter essere funzionanti. Si pensi allo streaming: se il throughput è basso, e quindi il numero di bit al secondo ricevuti è basso, allora si incorre in blocco della chiamata e/o degradazione della qualità dell'immagine.
- ***Security***
  Alcune applicazioni richiedono sicurezza: si allude principalmente alla riservatezza del comunicazione (quando si lavora con dati sensibili), ma anche all'integrità dei dati (se gestisco un bonifico bancario soggetti terzi non devono alterare la cifra del bonifico) o all'autenticazione. Introdurremo a tal proposito un protocollo che è estensione dei precedenti (aggiunta di uno strato alla pila).

Si considerino le seguenti applicazioni di uso comune, dove si confrontano realibility (data loss), throughput e timing (time sensitive?)

application	data loss	throughput	time sensitive?
file transfer/download	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kbps-1Mbps video:10Kbps-5Mbps	yes, 10's msec
streaming audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	Kbps+	yes, 10's msec
text messaging	no loss	elastic	yes and no

- Le prime tre applicazioni, diverse, presentano le stesse caratteristiche: non devono esserci perdite, ma allo stesso tempo non si richiede un throughput minimo e non sono sensibili al ritardo.
- Le applicazioni di streaming audio/video tollerano il ritardo, ma richiedono un throughput minimo e sono sensibili al ritardo (sensibilità rilevante nel caso di streaming dal vivo, mentre nel caso di trasmissione di un video già realizzato in passato la sensibilità è minore – si consideri che in questo ultimo caso è abitudine generare un ritardo per permettere una bufferizzazione senza ritardi)
- Giochi interattivi tollerano il ritardo, ma è richiesto un throughput minimo e si ha fotte sensibilità verso il ritardo ("interattivo", è logico)
- Text messaging non tollera perdite, non si richiede un particolare throughput minimo (messaggi inviati la sera possono arrivare il giorno dopo). Sulla sensibilità al ritardo si potrebbero fare

discussioni: non è di per se un problema, ma potrebbe diventarlo se la persona riceve in ritardo un messaggio rilevante.

4.2.2 Protocollo TCP (Transfer Control Protocol) per Stream service

Con *Stream service* si intende un servizio di tipo affidabile. Con trasporto affidabile si intende che tutti i pacchetti spediti arrivano a destinazione, arrivano integri e arrivano nell'ordine con cui sono stati spediti.

- ***Connection-oriented***.
  Necessario un setup per stabilire una connessione tra processo client e processo server. Il circuito non è fisico: semplicemente le due parti allocano risorse per gestire la connessione, e il percorso svolto dai pacchetti sarà sempre lo stesso.
- ***flow control* e *congestion control***.
  L'invio è *controllato*, su due aspetti.
  - **Controllo sul flusso**. Il protocollo garantisce che il sender non invierà al receiver un "flusso troppo veloce di dati" (si evita di riempire il buffer del receiver al punto da andare in overflow). Il receiver indicherà al sender lo spazio rimasto nel suo buffer, e il sender si adegua alterando il rate di trasmissione.
  - **Controllo di congestione**. Immaginiamo i router come gli incroci di una strada, questi incroci possono essere congestionati. Stessa cosa il router: in presenza di traffico elevato il buffer del router viene riempito in modo rilevante, contribuendo all'aumento di ritardo e portando in alcuni casi a perdite di pacchetti. La congestione non dipende di per sé dal router (individuo) ma dal comportamento del sistema nel suo complesso (collettività): ogni nodo fa la sua parte riducendo il rate di trasmissione del pacchetto.
- **Cosa non viene offerto**.
  Non si hanno garanzie sul ritardo massimo sperimentato dai pacchetti, su un throughput minimo e sulla sicurezza.

4.2.3 Protocollo UDP (User Datagram Protocol) per Datagram service

Con *Datagram service* si intende un servizio non affidabile: la sorgente invia pacchetti, ma non è detto che questi arrivino a destinazione, arrivino integri o arrivino nell'ordine con cui sono stati inviati. Si parla di datagram perché ogni pacchetto è gestito individualmente, in contrasto col flusso dello Stream service.

- Non si ha affidabilità nella connessione.
- Non si ha setup.
- Non si ha controllo di congestione o controllo del flusso.
- Non si garantisce niente sulle cose che il TCP non offre.

Per quale motivo ci interessa UDP? Perché offre un servizio snello, non introduce ritardi contrariamente a TCP (che svolge controlli intervenendo sul rate di trasmissione).

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote login	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (eg Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

- Datagram è valido per applicazioni interattive loss tolerant che sono sensibili al ritardo.

- Si hanno applicazioni intermedie, come quelle di streaming: ritardo da minimizzare, ma si tollera anche ritardo di qualche secondo. Inizialmente si è privilegiato UDP, successivamente sono state sviluppate anche applicazioni basate su TCP.
- Non si affronta minimamente il problema della sicurezza, poiché TCP e UDP sono stati proposti agli albori, quando la sicurezza non era ancora un argomento rilevante.

4.2.4 Transport Layer Security (TLS)

Quando la sicurezza è diventata argomento rilevante si è intervenuti introducendo il TLS (Transport Layer Security): potremo definirlo un protocollo, ma in realtà consiste in un ulteriore strato posto sopra il TCP. Questo significa che:

- le applicazioni richiamo le funzioni del TLS;
- le funzioni del TLS invocano quelle relative al TCP.

Il TLS permette la trasmissione di messaggi cifrati garantendo l'integrità dei dati (si pensi che inviare una password in Internet con protocollo TCP o UDP significherebbe trasmetterla in chiaro). La comunicazione è preceduta da un'autenticazione end-point.

4.3 Esempi di applicazioni client-server

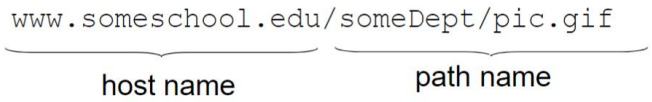
4.3.1 Applicazioni Web

Pila protocollare
Appplication ***messages***
Transport *segments*
Network *datagrams*
Link / Datalink *frames*
Physical

4.3.1.1 Pillole di Progettazione Web

Una pagina web è costituita da un numero di oggetti, ciascuno dei quali può essere memorizzato in server Web diversi.

- Gli oggetti possono essere file HTML, ma anche immagini JPEG, applicazioni JPEG, file audio…
- Una pagina web è costituita un file HTML base, che include un certo numero di oggetti embedded (inclusi nella stessa pagina, ma provenienti da fonti diverse e riferiti per mezzo di URL, *Uniform Resource Locator*). L'URL è strutturato in:



  - protocollo da utilizzare (solitamente http);
  - *host name*, nome simbolico dell'host che indica dove la risorsa è disponibile;
  - *path name*, percorso all'interno dell'host necessario per raggiungere la risorsa.

  All'interno dell'host name abbiamo il dominio.

4.3.1.2 Protocollo Hypertext Transfer Protocol (HTTP)

4.3.1.2.1 Spiegazione

Il protocollo HTTP è alla base del funzionamento dei browsers. Si basa tra un client e un server:

- il client è il dispositivo che esegue il browser;
- il server è quello che noi chiamiamo web server, che gestisce l'interazione coi client per mezzo di apposito applicativo (il più noto è Apache)

Precisamente:

- Il client richiede l'accesso a un oggetto web;
- Il server riceve la richiesta di accesso e invia, in risposta, l'oggetto richiesto.

Si faccia attenzione a non confondere l'applicazione (il browser – Chrome, Firefox…) col protocollo su cui si basa l'applicazione (il protocollo HTTP, che tutti i browser seguono per richiedere accesso agli oggetti).

L'affidabilità richiesta nella trasmissione degli oggetti web richiedere l'uso del protocollo di trasporto TCP. Precisamente:

- il client apre una connessione TCP col server (porta 80);
- il server accetta la connessione TCP richiesta dal client;
- il client e il server si scambiano richieste HTTP per mezzo di un socket, secondo i formati spiegati più avanti (*HTTP Request message* ed *HTTP Response message*);
- la connessione TCP viene chiusa.

4.3.1.2.2 Protocollo *stateless*

Il protocollo HTTP è definito stateless: il server non mantiene alcuna informazione sulle richieste passate.

- Prendiamo l'esempio più banale di tutti: il browser lento.
- L'utente è impaziente e inizia a inviare la richiesta nuovamente per diverse volte.
- Ogni richiesta dovrà essere gestita, e sarà gestita da zero perché il protocollo è stateless.

La decisione potrebbe sembrare controproducente, ma è la più conveniente per la complessità di un possibile protocollo *statefull*. Sarebbe necessario :

- mantenere lo storico delle richieste (occupazione maggiore di memoria);
- assicurarsi che il client e il server abbiano le stesse informazioni (intervenendo in caso di inconsistenze).

Vedremo più avanti che con i cookies e la cache è possibile rendere questo protocollo *statefull*, introducendo la memorizzazione di informazioni sul client (come al solito si tende a portare la complessità alla periferia della rete).

4.3.1.2.3 Protocollo HTTP non persistente VS HTTP persistente

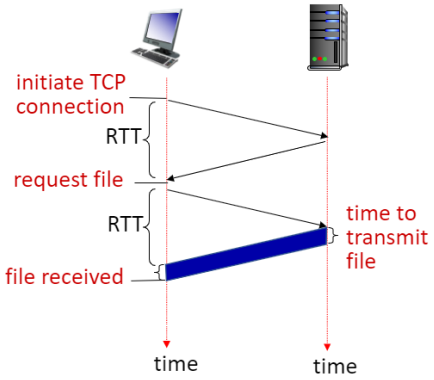
Il protocollo HTTP può essere persistente o non persistente.

- **Protocollo HTTP non persistente**.
  La connessione TCP viene aperta per trasmettere al più un oggetto. Supponiamo che l'utente voglia accedere alla pagina www.someSchool.edu/someDepartment/home.index che contiene testo e riferimenti a dieci immagini JPEG.
  - Il client richiede l'apertura di una connessione TCP col server www.someSchool.edu alla porta 80 (in generale la porta utilizzata è questa)
  - Il server www.someSchool.edu, che stava attendendo richieste su quella porta, accetta la connessione notificando il client. Il client e il server adesso comunicano col socket.
  - Il client invia un ***messaggio di richiesta*** nel socket, indicando che vuole l'oggetto someDepartment/home.index.
  - Il server risponde ponendo nel socket il ***messaggio di risposta*** e chiude la connessione TCP.

  Gli step precedenti sono ripetuti dal primo all'ultimo per ogni immagine JPEG.
- **Protocollo HTTP persistente**.
  Una singola connessione TCP può essere usata per trasmettere più oggetti tra un client e un server. La cosa è introdotta in HTTP/1.1.

Facciamo un confronto dal punto di vista dei tempi di risposta. Per prima cosa definiamo RTT (*Round Trip Time*) come il tempo di cui ha bisogno un piccolo pacchetto per viaggiare dal client al server e tornare indietro. Riflettiamo sul protocollo non persistente:

- Due RTT sono necessari per l'handshake, dove si ha apertura di connessione, accettazione della connessione e messaggio di richiesta (*three-way handshake*).
- Oltre ai due RTT detti si deve considerare il tempo necessario a trasmettere l'oggetto Web richiesto.



Otteniamo:

$$\text{Non persistent HTTP response time} = 2\ \text{RTT} + \text{file transmission time}$$

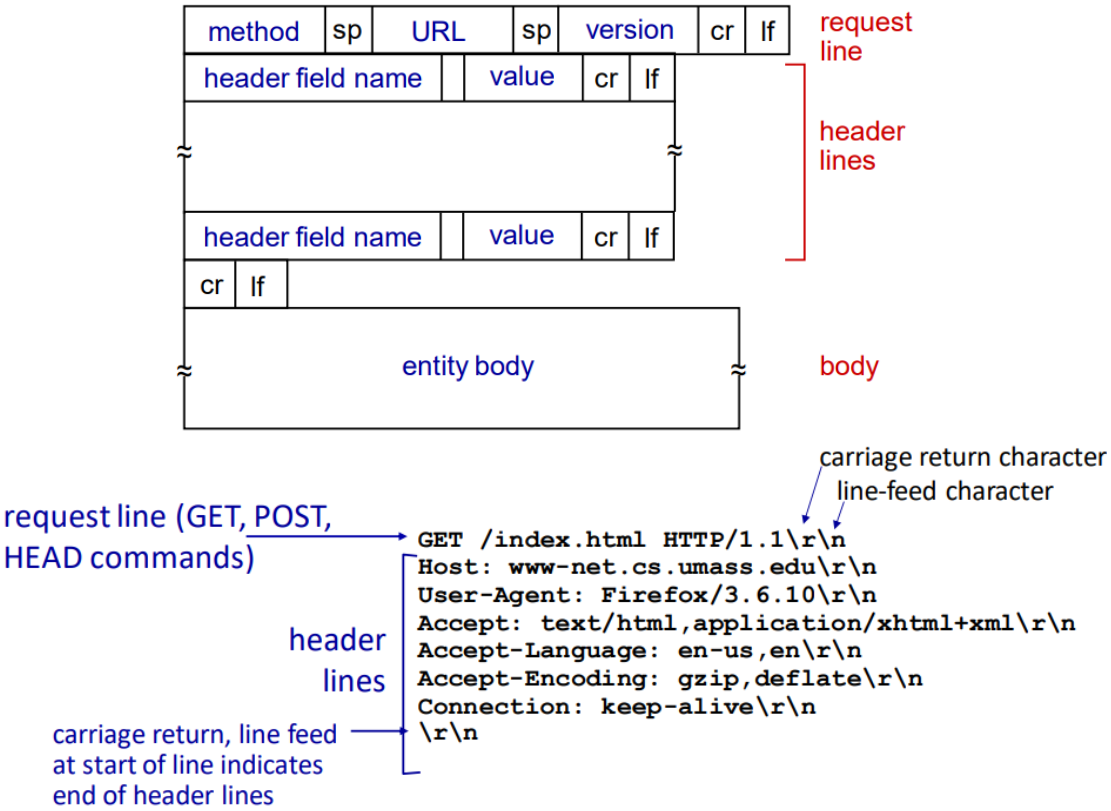
Nel caso di trasmissione di più messaggi 2RTT è moltiplicato tante volte quanti gli oggetti richiesti, mentre il file transmission time consiste nella somma dei tempi di trasmissione di ogni singolo file richiesto.

- Chiaramente in caso di connessione persistente i tempi si riducono: non abbiamo più una continua apertura e chiusura di connessione TCP (una sola attivazione di connessione TCP, si ottiene una pagina HTML ed eventuali oggetti presenti nella pagina possono essere richiesti immediatamente senza aprire ulteriori connessioni TCP).
- Si consideri anche l'onere maggiore per il web server dal punto di vista delle risorse in una connessione non persistente (allocazione in memoria e gestione in contemporanea di centinaia di richieste dai client).

Per quanto riguarda la chiusura della connessione l'approccio adottato è la chiusura automatica della connessione da parte del server a seguito di un periodo di inattività da parte del client (un *timeout* definito a priori).

4.3.1.2.4 HTTP Request message format

Abbiamo già anticipato che dopo aver stabilito una connessione il client invia un messaggio di richiesta e il server trasmette un messaggio di risposta.

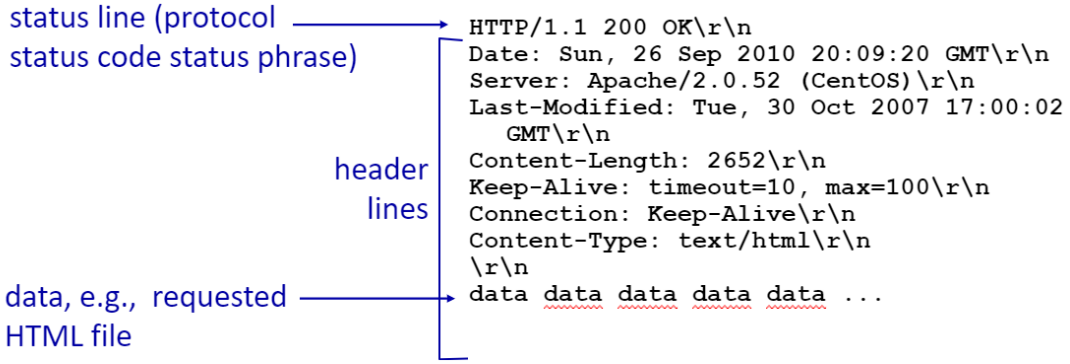


- Ogni messaggio trasmesso nel protocollo HTTP è in codifica ASCII (leggibile anche all'utente).
- Ogni riga si conclude con carattere *ritorno carrello* e *nuova riga* (per avere chiaro il concetto si pensi alla macchina da scrivere, e alle azioni che si facevano per andare a capo).
- La prima riga è detta ***request line***, e contiene:
    - il method (la tipologia di richiesta, il valore solitamente utilizzato è GET, ma abbiamo anche POST, HEAD E e PUT)
        - In GET il passaggio di parametri avviene direttamente dall'URL (body vuoto)
          Esempio: prova.php?parametro1=valore1&parametro2=valore2
        - In POST il passaggio di parametri avviene attraverso il body del messaggio
        - Con HEAD viene richiesto al server l'invio del messaggio di risposta senza l'oggetto

        - Con PUT si caricano nuovi files nel server (il contenuto è posto nel body)
    - l'oggetto richiesto al server;
    - la versione del protocollo http utilizzata dal browser.
- Le righe successive sono le ***header lines*** (righe di intestazione) e consistono in un insieme di campi, ciascuno accompagnato da un valore. Le informazioni poste nei campi dell'esempio in figura sono le seguenti:
    - l'host dove si trova l'oggetto a cui stiamo richiedendo accesso;
    - l'User-Agent, cioè il browser utilizzato per lanciare la richiesta;
    - l'Accept, le tipologie di media accettate nella risposta (se si richiede una mera pagina HTML si pone il text/html posto in figura, se il media richiesto è di altro tipo si pone altro (ad esempio se stiamo richiedendo un'immagine png porremo image/png);
    - il linguaggio preferito dal browser (non è detto che il server soddisfi la richiesta, se l'oggetto richiesto non è disponibile nella lingua richiesta allora l'oggetto sarà inviato nella lingua di default);
    - con Connection avente per valore keep-alive si richiede che la connessione TCP venga mantenuta aperta dopo aver soddisfatto la richiesta.
- Si pone infine una riga vuota (anch'essa conclusa con ritorno carrello e nuova riga).

4.3.1.2.5 HTTP Response message format

Il messaggio di risposta è codificato anch'esso in formato ASCII.



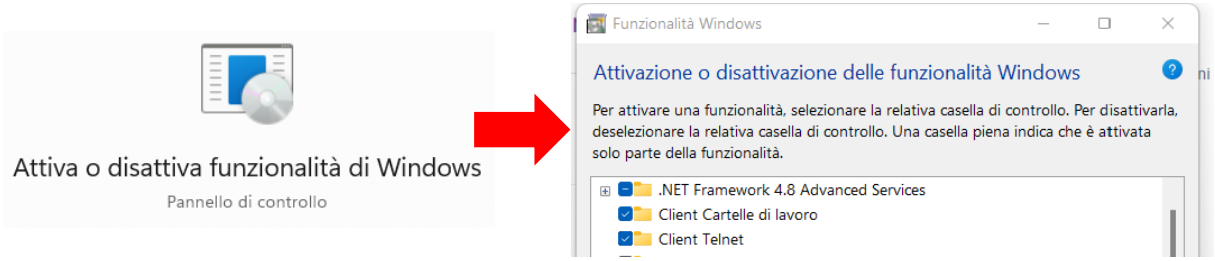
- Come prima abbiamo alla fine di ogni riga i caratteri ritorno carrello e nuova linea.
- La prima linea è la riga di stato (status line) che contiene la versione del protocollo utilizzata dal server e il codice di stato (response status code). Tra i possibili valori del codice di stato abbiamo:
    - ***200 OK***
      La richiesta ha avuto successo e il server pone nel messaggio di risposta l'oggetto richiesto dal client.
    - ***301 Moved Permanently***
      L'oggetto è stato rimosso permanentemente dalla directory indicata, non si pone l'oggetto nel body ma l'indirizzo della nuova collocazione nel campo Location (che nell'esempio in figura non è compilato)
    - ***400 Bad Request***
      Il server non ha compreso il messaggio di richiesta
    - ***404 Not Found***
      L'oggetto richiesto non esiste sul server
    - ***505 HTTP Version Not Supported***
      La versione del protocollo HTTP richiesta non è supportata.
- Successivamente abbiamo le linee di intestazione caratterizzate da campi (come nel messaggio di richiesta). Tra:
    - la data di creazione e invio del messaggio di risposta da parte del server nel campo Date;
    - la data dell'ultima modifica apportata all'oggetto restituito nel campo Last-Modified;
    - la dimensione in byte dell'oggetto inviato nel campo Content-Length;
    - nel campo Server si indica che la generazione è avvenuta da parte di un Web Server Apache;

- il campo Connection segnala che il server manterrà aperta la connessione TCP, secondo le condizioni poste nel campo Keep-Alive (si indica un timeout e un numero massimo di richieste lanciabili nella stessa connessione TCP);
- il campo Content-Type la tipologia dell'oggetto restituito (in figura text/html, si risponde alla richiesta di una mera pagina HTML)

- Segue una riga vuota (presenti solo caratteri ritorno carrello e nuova riga) e il body, contenente l'oggetto richiesto dal client.

4.3.1.2.6 Esempio di applicazione del protocollo HTTP: comando telnet su cmd

È possibile testare il protocollo HTTP attivando il comando telnet sul prompt dei comandi



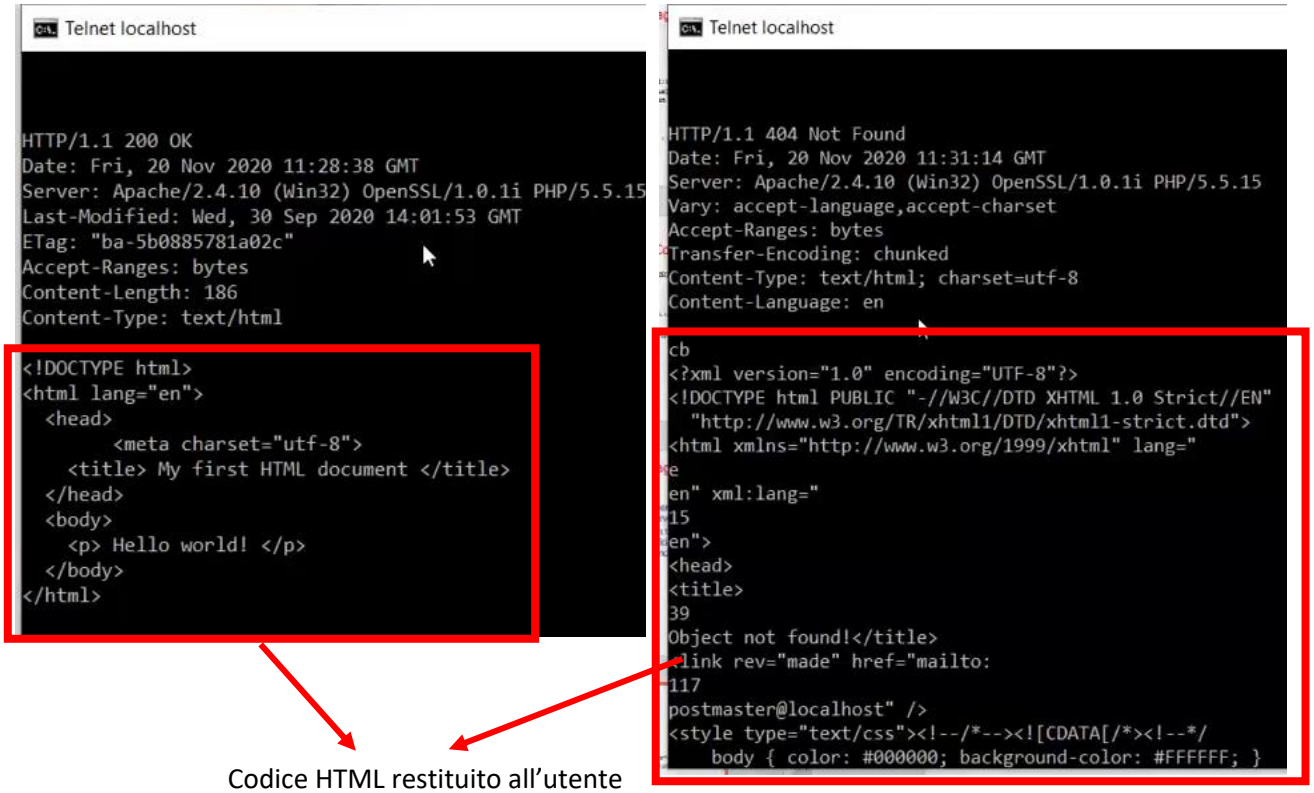
Fatto questo è possibile utilizzare il comando da prompt dei comandi

```
telnet <sito web> 80
```

Col comando precedente si apre una connessione TCP sulla porta 80 (la porta usata di default nel protocollo) col server indicato. Tutto ciò che adesso verrà scritto sul CMD verrà trasmesso sulla porta 80. È possibile sbizzarrirsi inviando un messaggio di richiesta HTTP nei modi indicati precedentemente.

```
GET /html.html HTTP/1.1
Host: localhost
```

```
GET /paginainesistente.html HTTP/1.1
Host: localhost
```



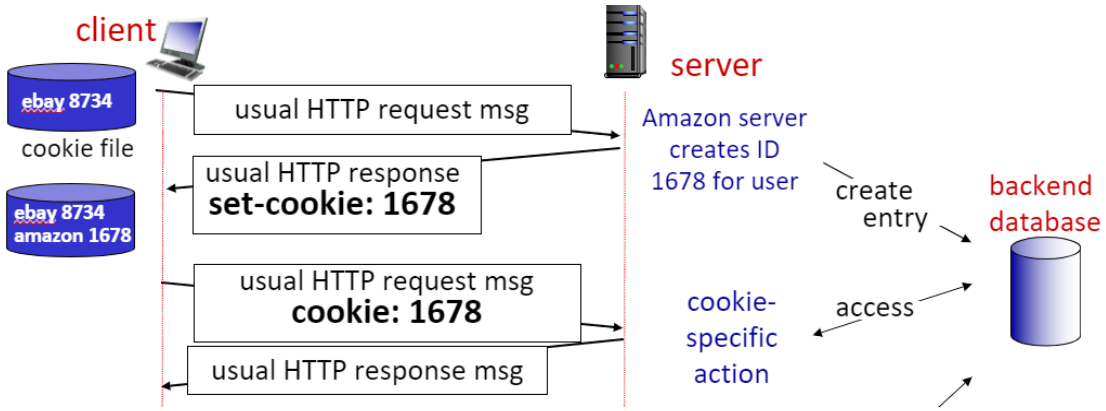
Codice HTML restituito all'utente

4.3.1.3 Cookies

I cookies permettono di adottare una soluzione ibrida tra approccio *stateless* e approccio *statefull*

- Si introduce il campo *Set-Cookie* nell'intestazione del *Response message* del server.
- Si introduce il campo *Cookie* per l'intestazione del *Request message* del client.
- I cookie sono memorizzati in un apposito file del client e sono gestiti dal browser.
- Da parte del server si richiede la memorizzazione di informazioni in un database.

Per spiegare i cookies prendiamo ad esempio Susan, che fa acquisti online su Amazon.com



- Il server di Amazon, quando riceve la prima richiesta HTTP dall'host di Susan, crea un identificativo numerico per Susan e lo trasmette al client ponendo nelle righe di intestazione del messaggio di risposta `Set-Cookie: IDUTENTE`.
- La creazione effettiva del Cookie è ad opera del browser di Susan: questo, ricevuto il messaggio di risposta dal server, vede l'identificativo creato per Susan nel campo Set-Cookie e di conseguenza aggiorna il file dei cookie ponendo l'identificativo appena ricevuto in una nuova riga.
- Al momento della creazione dell'identificativo numerico il server ha iniziato a raccogliere le informazioni di Susan che desidera memorizzare creando nuove righe del database: in ciascuna riga è presente l'identificativo utente trasmesso a Susan precedentemente.
- È passata una settimana: supponiamo che Susan lanci una nuova richiesta. Pone nelle righe di intestazione del messaggio di richiesta il campo Cookie con l'identificativo utente, recuperato dal file dei Cookie.
- Il server, ricevuto il messaggio di richiesta, trova tra i campi il Cookie con l'identificativo utente: può lanciare un'interrogazione nel database e recuperare le informazioni relative a Susan.
  L'associazione è possibile perché l'identificativo utente è presente sia nel Cookie sia nel database.

I cookies rivestono un'utilità fondamentale per servizi di e-commerce, e in generale per tutti quei servizi che richiedono l'identificazione di un'utente (come è possibile che il browser si ricordi chi siamo dopo aver fatto login con username e password in un qualunque servizio?).

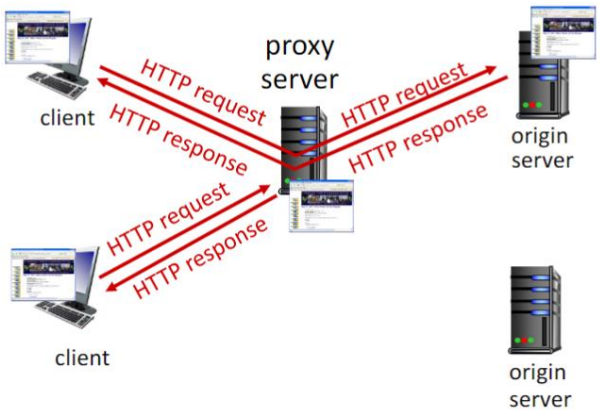
Rivestono una certa utilità anche nel tracciare le abitudini degli utenti: il proprietario di un servizio (per esempio Amazon) può tracciare le abitudini dell'utente, memorizzarle e utilizzarle a proprio vantaggio (Esempio: memorizzare i prodotti visitati da un utente per caricare pubblicità mirate).

La cosa è vantaggiosa per chi offre il servizio, ma sicuramente pone questioni etiche e di violazione della privacy dal punto di vista dell'utente.

4.3.1.4 Web-Caching

4.3.1.4.1 Idea di base

L'interazione tra client e server è di tipo request-response: il client manda un messaggio di richiesta e il server reagisce inviando un messaggio di risposta col contenuto richiesto. Client e server possono trovarsi a distanze elevate, e questo può richiedere tempi rilevanti.



Vogliamo minimizzare il Round Trip Time (RTT), cioè il tempo che passa tra l'invio del messaggio di richiesta e la ricezione del messaggio di risposta. Si consideri che normalmente un pacchetto passa attraverso più nodi: andiamo a introdurre un proxy server, un nodo dotato di cache/buffer dove viene memorizzato il contenuto di precedenti richieste.

- Supponiamo di richiedere l'accesso per la prima volta a una certa pagina Web.
  - L'HTTP request rivolto a un particolare server passa dal proxy server.
  - Il proxy server verifica se ha nella sua memoria il contenuto richiesto.
  - Essendo la prima richiesta il proxy server non trova nulla, quindi inoltra l'HTTP request al server originario.
- Supponiamo di richiedere nuovamente accesso alla pagina Web: il proxy server individua il contenuto nella sua memoria. Non si inoltra l'HTTP Request al server, ma si risponde direttamente al client con l'HTTP response.

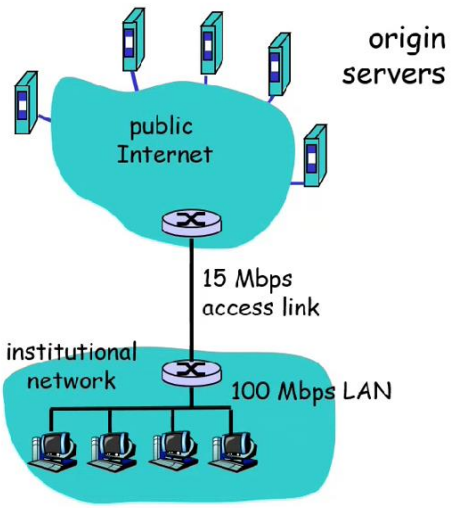
Il tempo di risposta *statistico* è ridotto in modo rilevante, si riduce il carico di lavoro del server finale (decongestionamento), si riduce complessivamente il traffico in Internet (dato che si gestiscono localmente le richieste). Due strade possibili su chi gestisce il proxy server:

- l'amministratore della rete, per esempio in una rete istituzionale;
- l'ISP locale.

Il meccanismo è completamente trasparente: l'utente non ne è consapevole e può ignorarne l'esistenza.

4.3.1.4.2 Esempio di web-caching

Supponiamo di trovarci in una rete istituzionale con una velocità di 100Mbps. La rete è connessa alla rete pubblica per mezzo di un link con velocità 15Mbps. Gli utenti di questa rete istituzionale hanno necessità di visitare siti web. Complessivamente:



- si scaricano oggetti di dimensione media 1Mbit;
- in media complessivamente sono lanciate 15 req/sec.

Il ritardo complessivo, che vogliamo minimizzare, consiste nella seguente somma

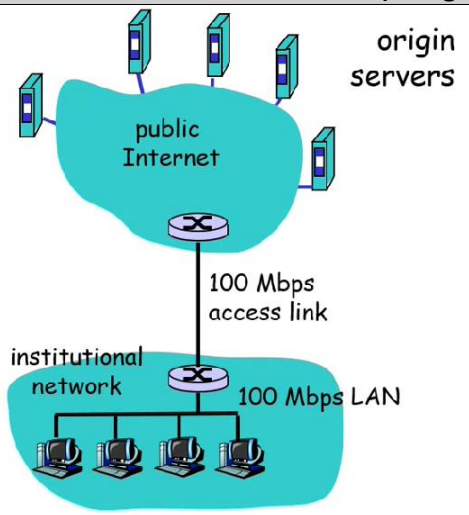
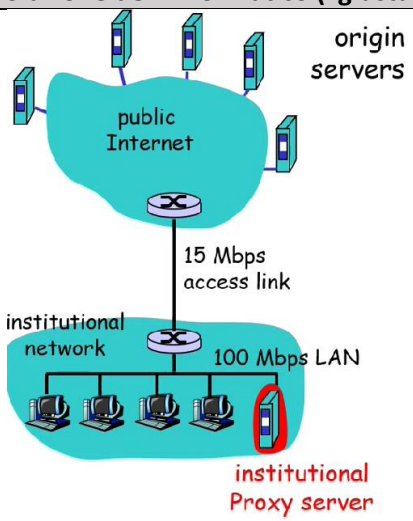
$$\text{Total delay} = \text{Internet delay} + \text{access delay} + \text{LAN delay}$$

Cosa possiamo dire?

- Supponiamo idealmente che Internet pubblico non faccia effetto collo di bottiglia: per comodità affermiamo che il ritardo è di 2 sec (valore non elevato).
- La rete istituzionale è utilizzata al 15% (15Mbps VS 100Mbps): il contributo al ritardo è piccolo, misurabile in millisecondi.
- Il link di accesso è utilizzato al 100%: problema, il ritardo access delay è elevato!

Dunque: $\text{Total delay} = \text{Internet delay} + \text{access delay} + \text{LAN delay} = 2\ sec\ +\ minutes\ +\ millisecond$

Cosa proponiamo?

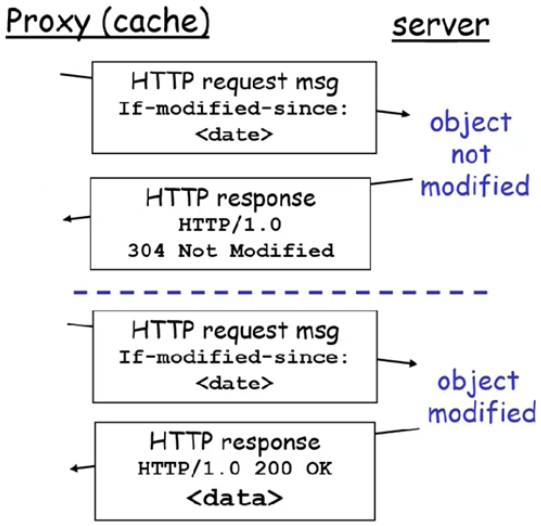
Soluzione del telecomunicazionista (“sbagliata”)	Soluzione dell’informatico (“giusta”)
	
Il telecomunicazionista propone di porre un link di accesso con velocità di 100Mbps. La soluzione funziona dato che: - L’utilizzo della LAN è lo stesso - L’utilizzo del link di accesso è ridotto dal 100% al 15% In sostanza $\text{Delay} = 2\ sec\ +\ msecs\ +\ msecs$ C’è solo un problema: questo upgrade può essere estremamente costoso!	L’informatico non interviene sul link di accesso, ma introduce un proxy server nel contesto della rete istituzionale. - Supponiamo che il 40% delle richieste sia soddisfatto immediatamente dal proxy, e il rimanente 60% deve “uscire fuori” (la richiesta deve passare dal link di uscita per raggiungere il server) - L’utilizzo del link di accesso è ridotto dal 100% al 60%: segue piccolo ritardo trascurabile nella somma (10 ms) In sostanza (0.01s tempo per visitare il proxy server) $\text{Delay} = 0.6\ (2\ s\ + 0.01\ s) +\ 0.4 \cdot 0.01\ s\ =\ 1.21 s$

Dopo l’intervento dell’informatico il telecomunicazionista verrà licenziato (cit.).

4.3.1.4.3 Conditional GET

Il GET condizionale è una variazione del Web-caching appena spiegato. Si include nell’HTTP Request il campo `If-modified-since`, dove indichiamo una particolare data. Succederà quanto segue:

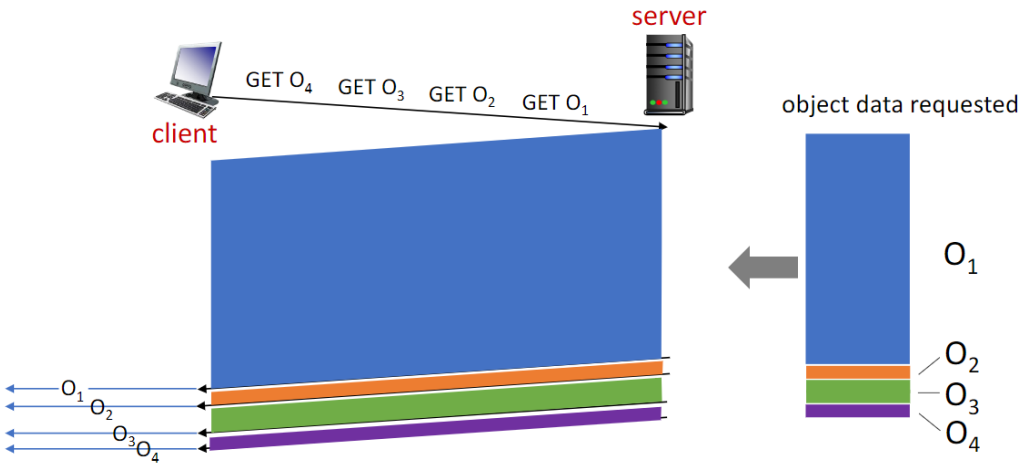
- l’*HTTP Response message* contiene errore 304 nel caso in cui l’oggetto richiesto non sia stato modificato a seguito di quella data, il messaggio contiene solo l’errore e non l’oggetto;
- l’*HTTP Response message* contiene l’oggetto nel caso in cui questo sia stato modificato a seguito della data indicata nel campo (viene restituito normalmente, come abbiamo già visto nelle pagine precedenti).



4.3.1.5 HTTP/1.1, HTTP/2 e HTTP/3

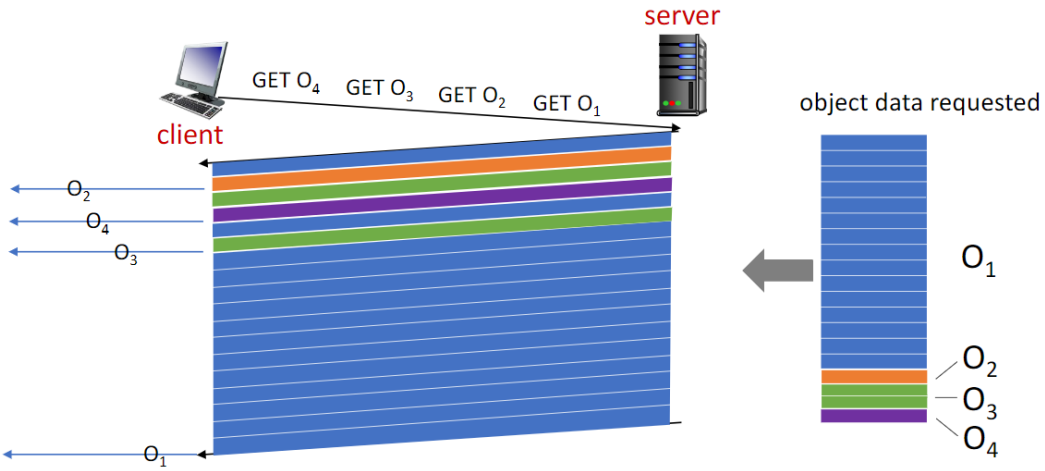
Le versioni successive del protocollo HTTP hanno introdotto le seguenti novità:

- **HTTP/1.1**
  La versione 1.1 di HTTP introduce una cosa già definita possibile nelle pagine precedenti: l'invio di più richieste di oggetti per mezzo di un'unica connessione TCP. Il server gestisce le richieste con logica FCFS (First-Come-First-Served). Si consideri il seguente esempio con quattro oggetti, di cui il primo di dimensioni notevoli:



  È evidente il tempo di attesa elevato per gli oggetti successivi al primo, nonostante questi siano di dimensione decisamente minore.
- **HTTP/2**
  La versione 2 introduce maggiore flessibilità nell'invio degli oggetti.
  - Nella versione 1.1 la logica FCFS potrebbe portare a tempi di attesa non banali per piccoli oggetti, se preceduti nell'ordine da oggetti di dimensione molto grande.
    In caso di perdite la *loss recovery* contribuisce ad aumentare ulteriormente i tempi.
  - Nella versione 2 si determina l'ordine di trasmissione degli oggetti basandosi sulle priorità del client, ergo FCFS non è l'unica logica possibile. Si mitigano i problemi della versione 1 dividendo in frame gli oggetti.

  A proposito della divisione in frame riprendiamo l'esempio precedente e gestiamolo con la divisione in frame



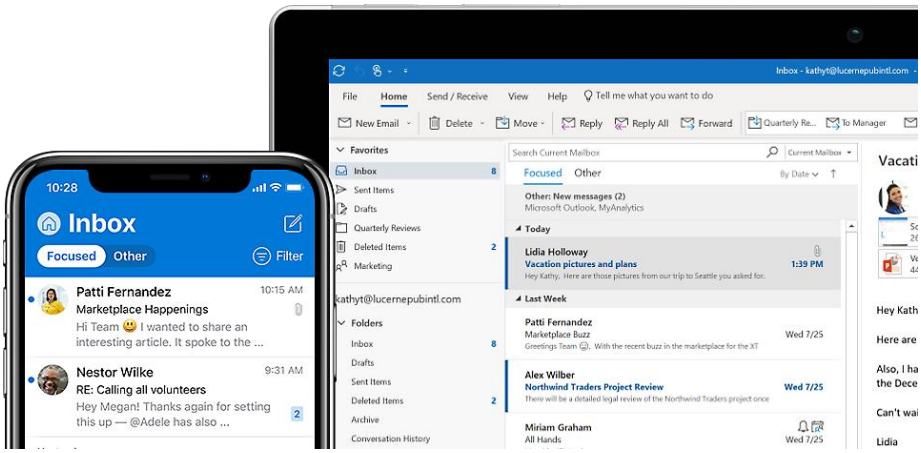
  L'introduzione dei frame nella seconda versione permette di ottenere gli oggetti in un ordine diverso, dando precedenza agli oggetti di dimensione più piccola (a discapito del primo oggetto, che sarà recapitato leggermente in ritardo rispetto a prima). La soluzione è sicuramente un miglioramento ma permane rallentamento nella trasmissione degli oggetti in caso di *recovery loss*.
- **HTTP/3**
  La versione 3 introduce controlli a tutela della sicurezza, e controlli di congestione su UDP (che ricordiamo non essere il protocollo di trasporto raccomandato).

4.3.2 Posta elettronica

4.3.2.1 Componenti fondamentali dell'applicazione

La posta elettronica si basa su tre componenti:

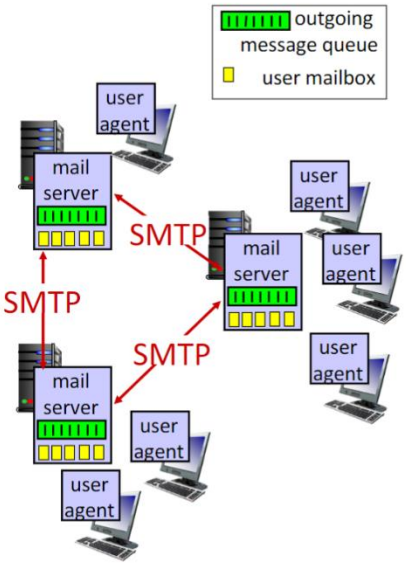
- gli ***user agent*** (il programma usato dall'utente per ricevere, leggere e inviare messaggi);



- i ***mail server*** (il server di posta elettronica, quello che effettivamente gestisce la posta elettronica);
- il protocollo ***Simple Mail Transfer Protocol*** (SMTP, interazione tra mail server);
- il protocollo ***Internet Message Access Protocol*** (IMAP, interazione tra mail server e user agent).

Il mail server è gestito dal provider del servizio di posta elettronica.

All'interno del mail server è presente una mailbox dedicata ai messaggi di posta elettronica di un particolare utente: il server, ricevuto un nuovo messaggio dall'esterno, controlla il destinatario e colloca il messaggio ricevuto nella relativa mailbox.



Si osservi che il nome "mail server" può essere oggetto di ambiguità, in quanto lo stesso si comporta sia da client che da server:

- funge da client quando invia una mail a un altro mail server;
- funge da server quando riceve una mail da un altro mail server.

A tal proposito notare nella figura la presenza di code in entrata e code in uscita per ogni mail server.

4.3.2.2 Protocollo Simple Mail Transfer Protocol (SMTP)

Il protocollo SMTP è alla base della comunicazione tra i vari mail server, che come già anticipato sono i reali interlocutori nella posta elettronica (ciò che fa l'utente è scaricare sull'host i messaggi sul mail server, e in caso di invio del messaggio colui che effettivamente svolge l'azione è il mail server e non l'user agent).

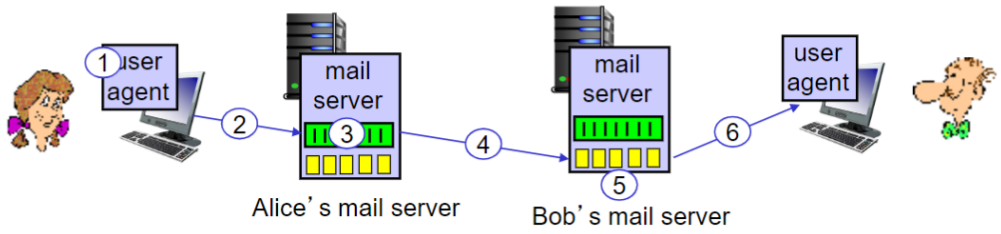
Si ricorre al protocollo TCP per garantire una comunicazione affidabile (cosa che non ci dovrebbe stupire per quello che abbiamo già spiegato). L'esecuzione del protocollo prevede:

- apertura di una connessione TCP;
- esecuzione del protocollo vero e proprio, come diremo subito dopo;
- chiusura della connessione TCP.

Il protocollo SMTP vero e proprio consiste in:

- *handshaking*;
- trasferimento del messaggio (messaggi per lo scambio di informazioni utili alla trasmissione del messaggio, e infine il messaggio stesso);
- chiusura.

Il messaggio presenta una struttura simile a quella vista nel protocollo HTTP (con commands e response), inoltre è richiesta una codifica in 7-bit ASCII (all'epoca è stato pensato con questo tipo di codifica, prima dell'estensione alla forma attuale, chiaramente adesso si è andati oltre). Per avere le idee più chiare immaginiamo un'interlocuzione via posta elettronica tra due soggetti: Alice e Bob!



1. Alice usa il suo programma di posta elettronica per comporre il messaggio, quando ha finito invia.
2. Agli occhi di Alice il messaggio viene inviato direttamente a Bob, in realtà viene trasmesso per prima cosa al mail server di Alice. Viene collocato nella coda in ingresso.
3. Il mail server di Alice, ricevuto il messaggio, apre una connessione TCP col mail server di Bob.
4. Una volta stabilita la connessione il messaggio viene trasmesso dal mail server di Alice al mail server di Bob.
5. Il mail server riceve il messaggio, verifica chi è il destinatario (Bob) e colloca il messaggio nella sua mailbox.
6. Il messaggio sarà disponibile quando Bob scaricherà la mailbox col suo user agent.

Perché il mail server di Alice non gira sul computer di Alice? Perché il mail server di Bob non gira sul computer di Bob? Per comodità, una volta stavano insieme, adesso con più dispositivi è scomodo (io voglio avere la possibilità di leggere la posta elettronica su più dispositivi).

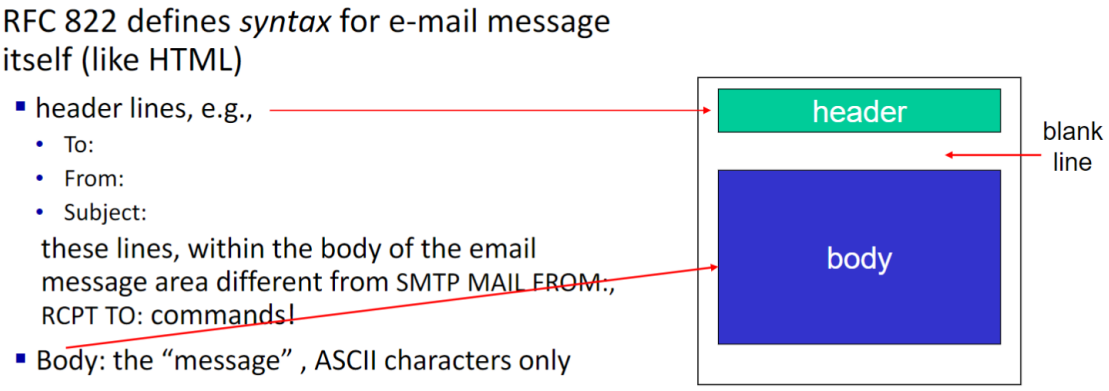
Possiamo immaginare un'interazione tra server nella seguente forma (chiaramente è una semplificazione, ma i comandi HELO, MAIL FROM, RCPT TO, DATA e QUIT sono comandi reali). Ricordarsi che il client è il mail server che invia il messaggio, mentre il server è il mail server che riceve il messaggio.

```
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
```

- Le prime tre righe sono convenevoli dove server e client "si presentano" col loro identificativo.
- Le quattro righe successive sono uno scambio di informazioni utili alla trasmissione del messaggio: il client indica prima il mittente e dopo il destinatario del messaggio, attendendo ogni volta che il server risponda affermativamente.
- Il client segnala con DATA la volontà di iniziare a trasmettere il contenuto della mail, il server risponde affermativamente con la regola che prevede conclusione del contenuto nella forma CRLF.CRLF
- A seguito della riga col solo punto il server accetta il messaggio e il client chiude l'interlocuzione avviata coi convenevoli (NON LA CONNESSIONE TCP, CONNESSIONE PERSISTENTE).

4.3.2.3 *Formato del messaggio nel protocollo SMTP*

Il protocollo SMTP definisce il formato del messaggio di posta elettronica scambiato tra i due mail server. L'header prevede la presenza di mittente, destinatario e titolo del messaggio, il body è il messaggio vero e proprio, posto in codifica ASCII.

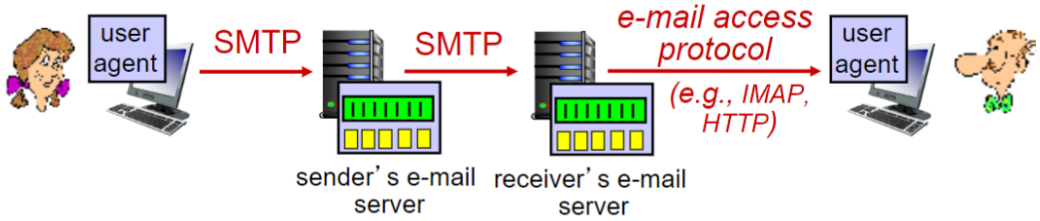


4.3.2.4 *Confronto tra protocollo SMTP e protocollo HTTP*

- **Analogie**
  - Entrambi hanno comandi e risposte espressi in codifica ASCII (con la differenza che SMTP richiede codifica in 7-bit ASCII).
  - SMTP e HTTP si basano su connessioni persistenti (con la differenza che HTTP offre possibilità di scelta).
- **Differenze**
  - Nel protocollo HTTP il client si collega al server per scaricare dati (pull). Nel protocollo SMTP il client si collega al server per inviare dati (push).
  - In HTTP ogni oggetto è incapsulato in un suo messaggio di risposta.
    In SMTP più oggetti potrebbero essere inviati in un unico *multipart messagge*.
  - SMTP determina la fine del messaggio con CRLF.CRLF.
    In HTTP si può individuare l'ultimo byte per mezzo del campo `Content-Length`.

4.3.2.5 *Protocollo IMAP per l'accesso alla posta elettronica*

Con le spiegazioni precedenti abbiamo chiarito come avviene l'interazione tra mail server, ma non abbiamo ancora detto come avviene l'interazione tra user agent e mail server.



Introduciamo a tal proposito un ulteriore protocollo: il protocollo IMAP.

- Gestisce il download della mailbox dal server all'host.
- Permette l'organizzazione della mailbox in cartelle.
- Ha soppiantato il protocollo POP, che contrariamente all'IMAP gestisce in modo diverso il download dei messaggi:
  - nel POP si scaricano tutti i messaggi eliminandoli dal mail server (questo significa che uno non potrà accedervi da ulteriori dispositivi)
  - nell'IMAP si mantengono i messaggi sul server.

  Inoltre, solo il protocollo IMAP permette l'organizzazione della mailbox in cartelle (in realtà gli user agents hanno sempre previsto l'organizzazione in cartelle, ma modifiche alla struttura non venivano memorizzate sul server e quindi perdute in assenza di backup).

Osservazione. Possibile l'accesso alla mail con protocollo HTTP: server acceduto da una pagina web, come se fosse un server web.

4.3.3 Sistema di nomi di dominio a livello applicazione (*Domain Name System*, DNS)

4.3.3.1 Introduzione

Ogni host in rete è identificato per almeno un indirizzo IP, che sappiamo essere una sequenza di 32bit molto lunga. Per poter raggiungere un sito web è necessario ricordare l'indirizzo IP del server che ospita il sito web, cosa impossibile! Per questo motivo si è introdotto il *Domain Name System* (DNS)!

- Il DNS è un'applicazione che permette di stabilire corrispondenze tra indirizzi IP e nomi simbolici. La sua presenza è fondamentale per il funzionamento di Internet (nelle diapositive si parla del DNS come *core Internet function*). Più avanti sono introdotti ulteriori scopi del DNS.
- Il client è il soggetto che chiede la traduzione, il server invece è colui che offre la traduzione.

- **Protocollo UDP**
  Si basa sul protocollo di trasporto UDP, in quanto l'applicazione viene tirata in ballo moltissime volte in periodi di tempo molto piccoli (pensiamo al numero di volte in cui noi lanciamo una richiesta in rete digitando il nome di un sito web, o accedendo a un qualunque sito web per mezzo di un link su altre pagine – la lentezza va evitata perché potrebbe essere fonte di ulteriori rallentamenti se si pensa al comportamento dell'utente[6])

4.3.3.2 Database distribuito

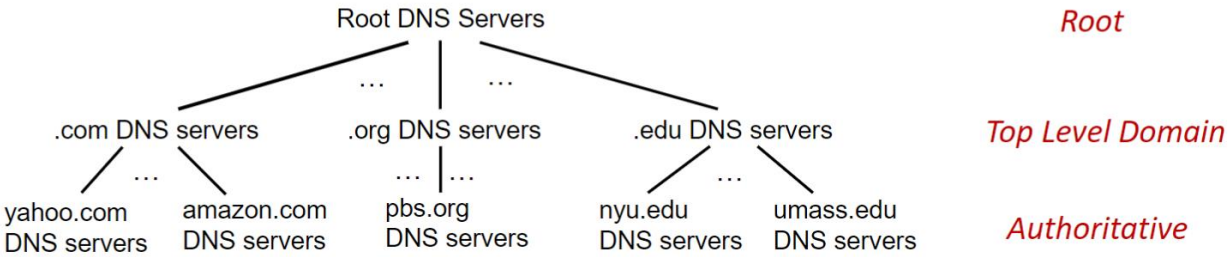
Un server centralizzato con un unico database è inconcepibile per i seguenti motivi:

- politico, si corre il rischio di dare in mano a un'autorità politica uno strumento della collettività mondiale (possibili ricatti sul venir meno dell'infrastruttura[7], oppure censura con rimozione dei nomi simbolici di siti scomodi alle autorità);
- *single point of failure*, se viene meno il punto allora viene meno l'intera infrastruttura di rete;
- non scala, diventa difficile gestire un numero sempre più elevato di richieste (un solo server potrebbe trovarsi a gestire in un giorno 600 miliardi di accessi).

La strada adottata è quella di un database distribuito e gerarchico. Distribuzione rispetto ai vari livelli di dominio, ma anche distribuzione rispetto al singolo livello di dominio.

- Viene meno il *single point of failure*
- Si migliora l'efficienza, perché il traffico viene distribuito (quindi aumenta la velocità).

La seguente figura fornisce un'idea di base



Tre tipologie di server:

- In cima abbiamo i ***root DNS Server***, da cui possiamo ricollegarci a tutti i top level domain esistenti. Questa root è gestita da un ente non governativo noto come ICANN (*Internet Corporation for Assigned Names and Numbers*).

- A livello inferiore troviamo i ***DNS relativi ai Top Level Domain***, curati dalle autorità di registrazione (nel caso dell'Italia abbiamo Registro.it). Il Top Level Domain è relativo in generale a un paese, ma il numero di top-level domain non relativi a particolari paesi è cresciuto a dismisura negli ultimi anni.

[6] L'utente manda nuove richieste se non riceve risposta sulla precedente.
[7] Tipo la Russia col gas.

- Nel livello più basso individuiamo gli ***authoritative DNS server***: l'associazione tra nome simbolico e indirizzo IP è stata già ottenuta, ma si introducono ulteriori DNS per permettere l'indicazione di sottodomini. Sono mantenuti dall'organizzazione che usufruisce del dominio, oppure dal fornitore di servizi.

Supponiamo che il client voglia ottenere l'indirizzo IP associato ad amazon.com (si guardi sempre la figura nella pagina precedente):

- Il client si rivolge al Root DNS server per ottenere l'indirizzo IP del .com DNS server
- Il client si rivolge al .com DNS server per ottenere l'indirizzo IP del amazon.com DNS server
- Il client si rivolge al amazon.com DNS server per ottenere l'indirizzo IP di amazon.com

4.3.3.3 Local DNS

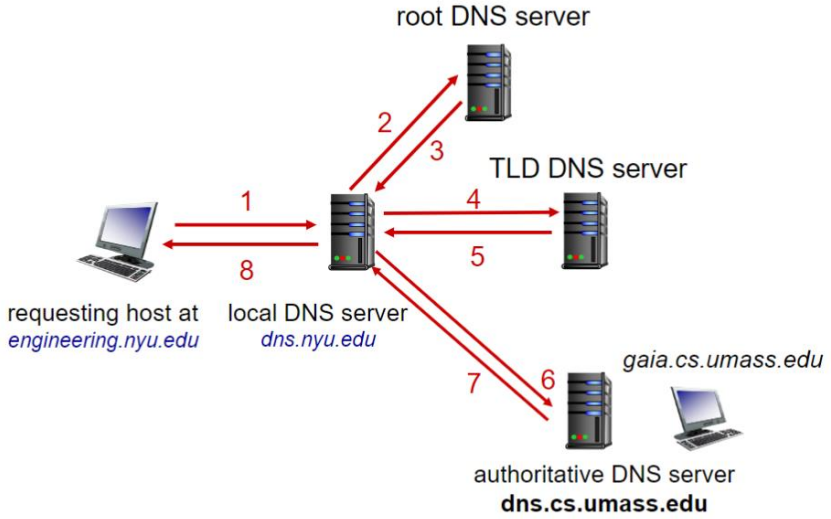
Il DNS locale non appartiene alla gerarchia precedentemente descritta.

- Possiamo immaginarci l'introduzione, all'interno della rete locale in cui opera l'host, di un ulteriore DNS server.
- Questo DNS server è il primo a cui l'host si rivolge per poter tradurre il nome.
- Funge da proxy: nel caso in cui la corrispondenza non sia presente assume le veci dell'host e inoltra al root DNS server la richiesta di traduzione lanciata dall'host.

L'importanza del DNS locale è tale da distinguere nelle impostazioni, in generale, un *local DNS primario* da un *local DNS secondario*. È un'ottima memoria cache che permette di ridurre l'operatività sulla rete, anche se bisogna stare attenti all'obsolescenza dei contenuti.

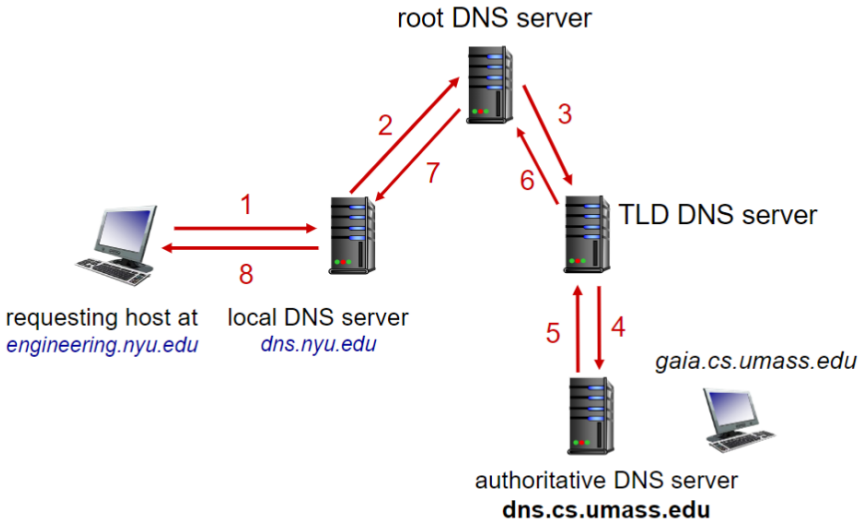
4.3.3.4 Interazione tra DNS server: approccio iterativo VS approccio ricorsivo

L'interazione tra servers può essere gestita con uno tra i seguenti approcci: approccio iterativo o approccio ricorsivo. Partiamo dal primo:



- l'host interagisce col local DNS;
- il local DNS server non ha trovato la corrispondenza e quindi interagisce col Root DNS server;
- il root DNS server risponde affermando di non conoscere l'indirizzo IP, ma rimandando a un ulteriore server (quello di livello di dominio inferiore)
- il local DNS server continua a interagire coi vari DNS server fino a quando non troverà l'indirizzo IP, a quel punto lo restituisce all'host.

L'approccio ricorsivo sposta la palla dal local DNS ai DNS server.



- Abbiamo sempre l'host che dialoga col local DNS server.
- Il local DNS server, se non ha la corrispondenza nome simbolico – indirizzo IP richiesta, si rivolge al root DNS server.
- La novità sta nel fatto che il local DNS server non debba fare più nulla, se non restituire l'indirizzo IP all'host: questo perché il root DNS server e i successivi vanno a interloquire in prima persona con i servers di livello inferiore, ricevendo da essi una risposta.

Quale approccio è migliore? Dipende dal punto di vista:

- per il local DNS server è sicuramente migliore l'approccio ricorsivo;
- per il DNS server nella gerarchia è sicuramente migliore l'approccio iterativo.

In generale si preferisce l'approccio iterativo perché è quello che permette di scaricare in periferia (come sempre) la complessità (immaginarsi l'approccio ricorsivo nel gestire migliaia di richieste).

Si minimizza il traffico introducendo ulteriori meccanismi di caching:

- le informazioni sono memorizzate nel local DNS server (in modo tale da evitare accessi anche nel solo root DNS server);
- si introducono meccanismi di timeout per prevenire la conservazione di informazioni non aggiornate.

4.3.3.5 Servizi offerti dall'applicazione

Il servizio principale offerto dal DNS è quello che abbiamo descritto fino ad ora (traduzione hostname to IP address), ma non è l'unico. Si segnalano:

- ***host aliasing***
  Possibilità di associare più nomi di dominio all'indirizzo IP (nome canonico è il primo, quello ulteriore è l'alias).
- ***mail server aliasing***
  Individuazione delle corrispondenze tra indirizzi di posta elettronica e corrispondenti Web Server.
- ***load distribution***
  Possibilità di associare più indirizzi IP allo stesso nome di dominio. Si fornisce come risposta una lista di indirizzi e l'host ne sceglie uno, passando ai successivi nel caso in cui i precedenti non funzionino. Ogni volta che un utente accede all'elenco di indirizzi la lista viene traslata di una posizione, in modo tale che l'utente successivo utilizzi un indirizzo IP diverso.

4.3.3.6 Tipologie di record nel database distribuito

Fino ad ora abbiamo parlato del DNS dal punto di vista della richiesta e della risposta, ma non siamo andati nel dettaglio sul contenuto del database (ci siamo limitati a parlare del carattere distribuito e gerarchico del database). Abbiamo un numero elevatissimo di record, basati sul seguente formato:

RR format (*Resource Record*): (name, value, type, ttl)

Il campo *type* permette di gestire i diversi servizi che il DNS mette a disposizione: a type diverso segue interpretazione diversa degli altri campi del RR format. Il campo ttl (*Time To Leave*) segnala il tempo di vita del record. Seguono i principali tipi (non li vediamo tutti):

- **type = A (*Address*)**
  Servizio principale: traduzione del nome simbolico in un corrispondente indirizzo IP.
  - **name**: nome dell'host (Es: www.unipi.it)
  - **value**: corrispondente indirizzo IP.

- **type = CNAME (Canonical Name)**
  Servizio di traduzione degli alias.
  - **name**: nome dell'alias
  - **value**: nome canonico dell'host

- **type = NS (Name Service, o Name System)**
  Utilizzati per capire qual è il DNS server per un particolare dominio. Qual è il server authoritative per un particolare dominio?
  - **name**: dominio
  - **value**: nome simbolico del DNS server authoritative

- **type = MX (Mail eXchange)**
  Servizio a supporto della posta elettronica. Utilizzato ogni volta che si spedisce un messaggio di posta elettronica.
  - **name**: seconda parte dell'indirizzo di posta elettronica (dopo la chiocciola)
  - **value**: nome canonico del mail server associato al dominio di posta elettronica.

4.3.3.7 Esempio: aggiornamento del database a seguito di creazione di sito web

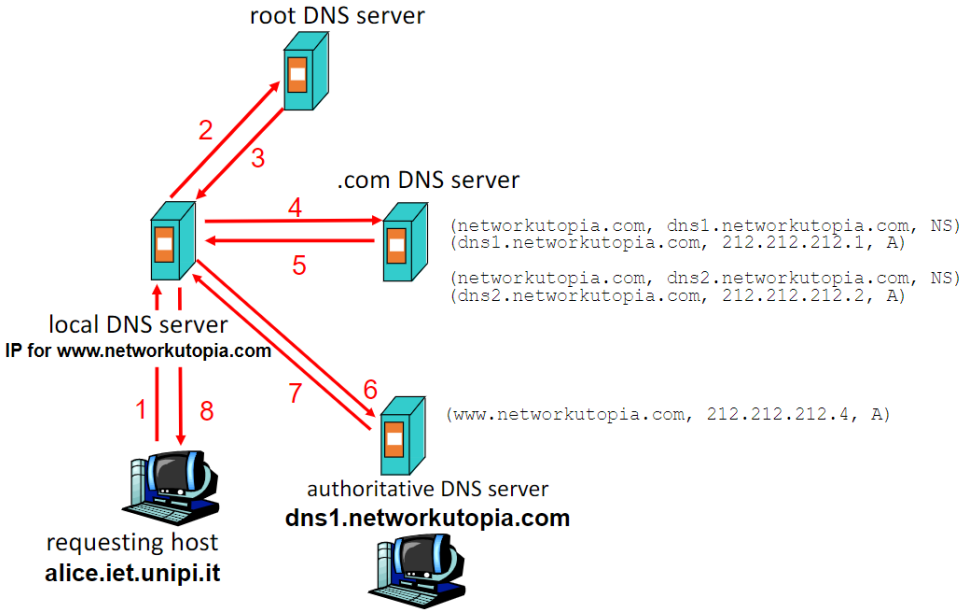
Supponiamo di aver fondato una startup di nome Network Utopia (prendiamo l'esempio di Anastasi) e di voler creare un sito web.

- Ci serve il dominio, lo registriamo presso il relativo DNS registrar. Supponiamo che ci interessi registrare networkutopia.com: lo faremo presso il registrar del dominio .com
- Quali sono i record da aggiungere presso il .com DNS server?
  ```
  (networkutopia.com, dns1.networkutopia.com, NS)
  (dns1.networkutopia.com, 212.212.212.1, A)
  ```
  Per prima cosa segnaliamo la corrispondenza tra il dominio networkutopia.com e il nome simbolico del DNS server authoritative, successivamente segnaliamo la corrispondenza tra nome simbolico del DNS server authoritative e il suo indirizzo IP.
- Quali sono i record da aggiungere presso il DNS server authoritative?
  ```
  (networkutopia.com, 212.212.212.4, A)
  ```
  Si segnala la corrispondenza tra dominio e relativo indirizzo IP. Nel caso in cui esistano indirizzi di posta elettronica associati a quel dominio si aggiunge anche un record di tipo MX, in modo tale da stabilire una corrispondenza col relativo mail server.

Segue nella figura il comportamento dei DNS server, supponendo di adottare l'approccio iterativo (default):

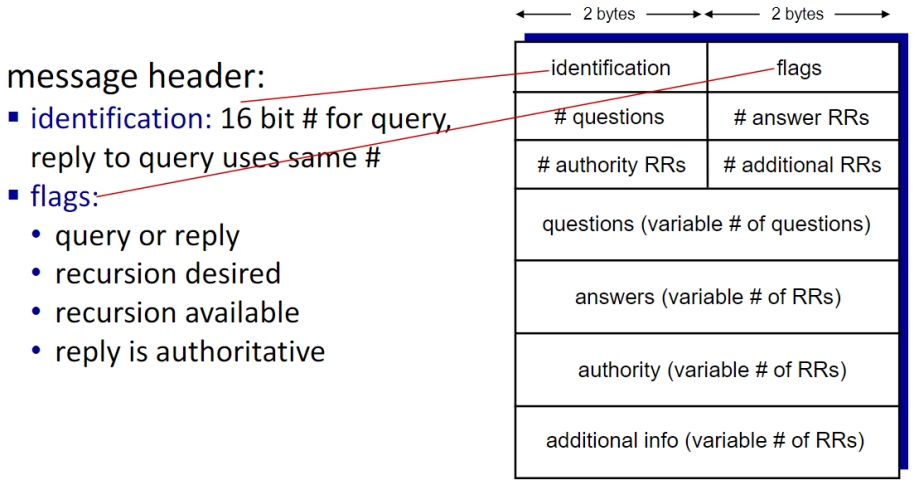
- Il primo che riceve la richiesta è il local DNS server. Individua che non ha al suo interno il record con la corrispondenza desiderata.

- Il local DNS server si rivolge al root DNS server e ottiene l'indirizzo IP del .com DNS server.
- Il local DNS server si rivolge al .com DNS server e ottiene l'indirizzo IP del DNS server authoritative. Si osservi che nella figura sono previsti due possibili indirizzi IP (la load distribution citata precedentemente).
- Il local DNS server si rivolge al DNS server authoritative, che fornirà col record di tipo A l'indirizzo IP del web server.



4.3.3.8 DNS protocol message (both query and reply)

Si hanno due tipologie di messaggi trasmessi nel contesto del DNS: query e reply. Il loro formato è lo stesso!



- L'*identification* è l'identificatore della query, avente dimensione di 16 bit. La risposta (reply) a una particolare query presenta lo stesso identificatore della query, cosa necessaria visto l'assenza di una connessione TCP (logicamente sappiamo che saranno eseguite molte query in poco tempo, ergo è necessario un identificatore per associare reply a query)
- Il campo dei flag contiene, appunto, dei flag. Tra questi abbiamo i seguenti:
  - flag che segnala se il messaggio consiste in una query o in una reply;
  - due flag per introdurre un approccio ricorsivo (un bit usato dalla query per segnalare il desiderio di ricorso all'approccio ricorsivo, un altro bit usato dalla reply per segnalare se è possibile adottare l'approccio ricorsivo);
  - flag che segnala se il server è authoritative per il nome su cui c'è stata la query (bit usato nella reply alla query)
- Segue il corpo del messaggio, dove abbiamo campi usati dalle query e campi usati dalle reply.

4.4 Esempi di applicazioni P2P

4.4.1 Ricerca di contenuti in un'applicazione P2P

4.4.1.1 Esempi e idea alla base dell'indice

La prima domanda che ci poniamo, in un contesto decentralizzato come quello P2P, è come rendere accessibili i contenuti: prima del download di un certo contenuto è necessario indicare al peer l'indirizzo dell'interlocutore che fornirà tale contenuto. Pensiamo ai seguenti esempi

- ***File sharing***
  I files sono resi disponibili dai peer. Il peer vuole sapere l'indirizzo IP di chi possiede il file desiderato.
- ***Instant Messaging***
  Gli username degli utenti sono associati a indirizzi IP.
  - Se l'utente A si connette allora l'indirizzo IP dell'utente A deve essere associato al suo username.
  - Se l'utente B avvia l'applicazione verifica quali amici sono connessi alla rete. L'utente A risulterà connesso alla rete e verrà segnalato, col suo indirizzo IP.

Possiamo fare le cose dette ricorrendo a un indice: l'indice è un database di coppie $(\text{key},\text{value})$ collocato in un particolare server a cui gli utenti si rivolgono.

- Key rappresenta il contenuto richiesto.
- $Value$ contiene l'indirizzo IP del peer che mi può offrire il contenuto (potrei avere associati più valori alla chiave, ad esempio più indirizzi IP).

$$(\text{Led Zeppelin IV}, 203.17.123.38)$$

L'indice deve essere aggiornato nel tempo: aggiunta di nuovi elementi, ma anche rimozione di dati non più aggiornati.

4.4.1.2 Primo approccio: centralizzato

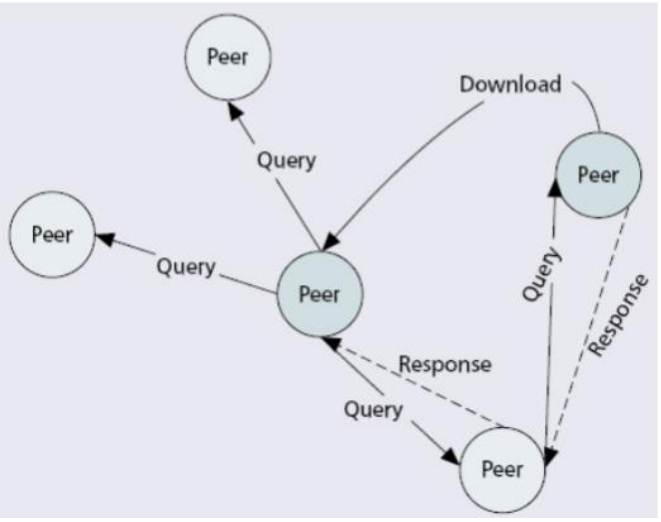
Il primo approccio adottabile è quello centralizzato: abbiamo un indice ospitato per intero in un server o in una server farm.

- Il peer si rivolge al server per richiedere particolari contenuti
- Il server risponde restituendo i dati necessari per usufruire di quel contenuto (in primis l'indirizzo IP del peer che offre il contenuto).

4.4.1.3 Secondo approccio: query flooding (decentralizzazione)

L'approccio query flooding è l'esatto opposto del primo approccio: l'indice è completamente decentralizzato, distribuito su tutti i peer che fanno parte della comunità.

- Ogni peer implementa parte dell'indice, precisamente i contenuti messi a disposizione dall'utente.
- L'insieme di tutti i peer implementa l'indice nella sua interezza.



Domanda: se l'indice è completamente decentralizzato a quale interlocutore deve rivolgersi l'utente per un particolare contenuto? A tutti!

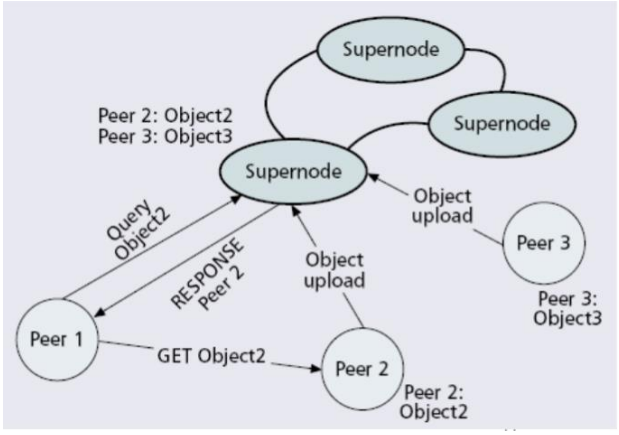
- Il peer ha un elenco (non ci interessa come lo ha generato) di indirizzi IP di altri peer, che si potrebbero definire i suoi "vicini di casa".
- Il peer stabilisce connessioni TCP con tutti o parte dei peer presenti in questo elenco.

- I peer a cui si è rivolto hanno a loro volta altri vicini, possono connettersi con ulteriori peer espandendo la ricerca: si parla di *query flooding* perché la domanda si propaga tra i peer, si potrebbe dire per mezzo di passaparola.
    - Si prevede un contatore in ogni query, con lo scopo di limitare il propagarsi di richieste: lo si decrementa, e raggiunto lo zero non si inviano più richieste. Si riducono i tempi, ma allo stesso tempo si potrebbe avere un orizzonte limitato: non si trova il contenuto perché non si è dato la possibilità di svolgere un adeguato numero di richieste.
- Non appena viene trovato un peer col contenuto richiesto la ricerca non viene più propagata. **Il peer col contenuto lo trasmette al peer da cui ha avuto inizio la ricerca**.

Nell'approccio centralizzato l'overhead è minimo, il tempo di risposta è abbastanza limitato: non possiamo dire lo stesso nel query flooding, dove i tempi sono rilevanti.

4.4.1.4 Terzo approccio: Hierarchical Overlay (ibrido degli approcci precedenti)

L'approccio *Hierarchical Overlay* è un ibrido dei precedenti. Fino ad ora abbiamo supposto che i peer siano tutti allo stesso livello, adesso supponiamo che vi siano dei peer gerarchicamente superiori (che definiamo supernodi).



- Non sono entità particolari e strutturate come nel caso di Napster, ma hanno una disponibilità elevata di risorse e rimangono connessi alla rete per una percentuale di tempo elevata.
- Implementano l'indice, che rimane distribuito tra i supernodi. Il nodo ordinario che vuole fornire un contenuto si associa a un supernodo e segnala a questo tale contenuto.

Nella figura a lato abbiamo:

- Peer 2 che mette a disposizione Object 2 e lo segnala a un supernodo.
- Peer 3 che mette a disposizione Object 3 e lo segnala al solito supernodo.
- Peer 1 che richiede l'accesso a Object 2: il supernodo segnala che l'oggetto è fornito da Peer 2, conseguentemente Peer 1 stabilisce una connessione con Peer 2 (`GET Object 2`).

4.4.1.5 Quarto approccio: Distributed Hash Table (DHT)

La tabella Hash distribuita è un ulteriore approccio con cui gestire l'indice.

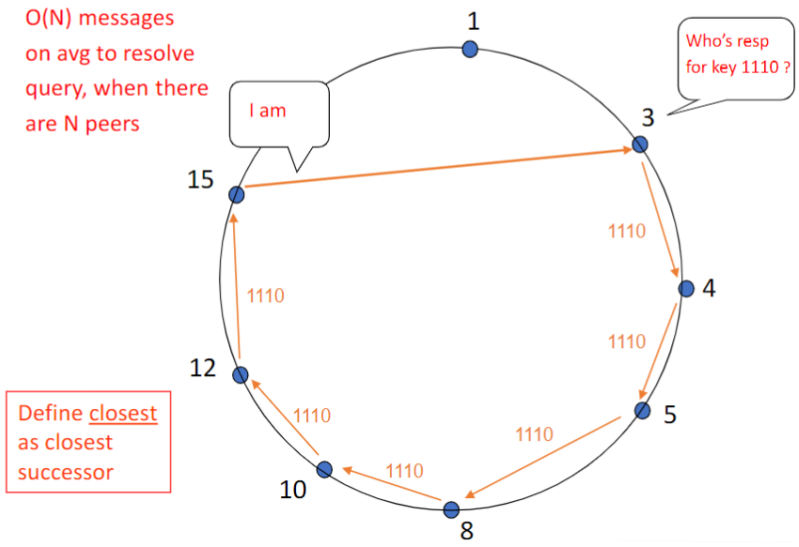
- Ogni peer è identificato da una stringa di n bit (nel range di interi $[0, 2^n - 1]$, tutti gli identificatori dei peer appartengono a suddetto range).
- È necessario identificare anche i contenuti per mezzo di chiavi. Generiamo queste chiavi per mezzo di una funzione hash
  `key = h("Led Zeppelin IV")`

Ragioniamo sull'inserimento di un nuovo contenuto.

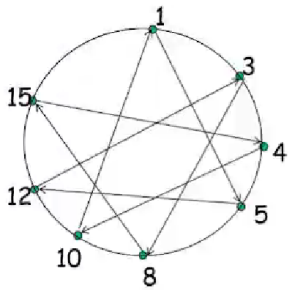
- Il contenuto è memorizzato presso un particolare peer, ma ci chiediamo: come scegliamo il peer?
    - Si calcola la chiave del contenuto citata prima, e
    - si assegna il contenuto *al successore più vicino*!

  Facciamo un esempio.
    - $n = 4$, gli identificatori appartengono al range $[0; 15]$
    - I peer sono identificati dai seguenti valori: 1,3,4,5,8,10,12,14
    - Supponiamo di avere chiave 13: il valore più vicino è 15, quindi il contenuto identificato dalla chiave 13 sarà assegnato al peer con identificatore 15.
    - Supponiamo di avere chiave 15: il valore più vicino è 1 (immaginarsi un array circolare), quindi il contenuto identificato dalla chiave 15 sarà assegnato al peer con identificatore 1.

Come può un qualunque nodo, data la chiave relativa a un contenuto, individuare il peer responsabile? Prendiamo il seguente esempio, dove ogni peer memorizza gli indirizzi IP di successore e predecessore.



- Il peer 3 è quello che chiede chi è il peer responsabile del contenuto con chiave 15 (1110).
- Il peer 4 riceve la query, e consta che non è responsabile per quel contenuto. Inoltra la richiesta al peer 5.
- Il peer 5 riceve la query, e consta che non è responsabile per quel contenuto. Inoltra la richiesta al peer 8.
- Si prosegue così fino al peer 15. Il peer 15 riceve la richiesta e consta di essere il responsabile per quel particolare contenuto. Conclude stabilendo una connessione col peer 3, quello che ha lanciato la query.



shortcuts

Possibile l'introduzione di *shortcuts*: non ci limitiamo a collegare un peer con predecessore e successore, ma anche con un ulteriore nodo posto a metà tra i precedenti. La cosa permette la riduzione del numero di passi (con la stessa richiesta di prima ho solo due passaggi: 3 -> 8 -> 15).

4.4.1.6 Esempio sulla ricerca: Napster

Napster è stata la prima applicazione P2P di successo. Risalente a fine anni 90, il suo scopo era quello di permettere la condivisione di contenuti musicali tra gli iscritti al servizio.

- **Indice centralizzato**. L'unica cosa che Napster fornisce è l'indice con gli indirizzi IP degli utenti. Ogni volta che l'utente diventava attivo l'applicazione notificava all'indice il suo indirizzo IP.
- **Ricerca contenuti**. L'utente che offre contenuti li registra presso l'indice. L'utente alla ricerca di un particolare brano musicale lancia una ricerca presso il server di Napster, e riceverà come risposta l'indirizzo IP dell'utente (memorizzato nell'indice). Lo stesso utente, ottenuto l'indirizzo IP, stabilisce una connessione TCP col peer che ha il contenuto.
- **Difetti**.
    - **Server centralizzato**.
      Single point of failure (se fallisce il server l'intera comunità non può accedere al servizio)

- **Effetto bottleneck**.
  Un solo server significa avere congestione elevata ed effetto collo di bottiglia.
- **Copyright Infringement**.
  I contenuti condivisi sono, nella maggior parte dei casi, illegali (violazione delle leggi sul copyright). Napster è stata condannata (concorso esterno) a risarcimenti milionari a seguito di denunce di autori musicali.

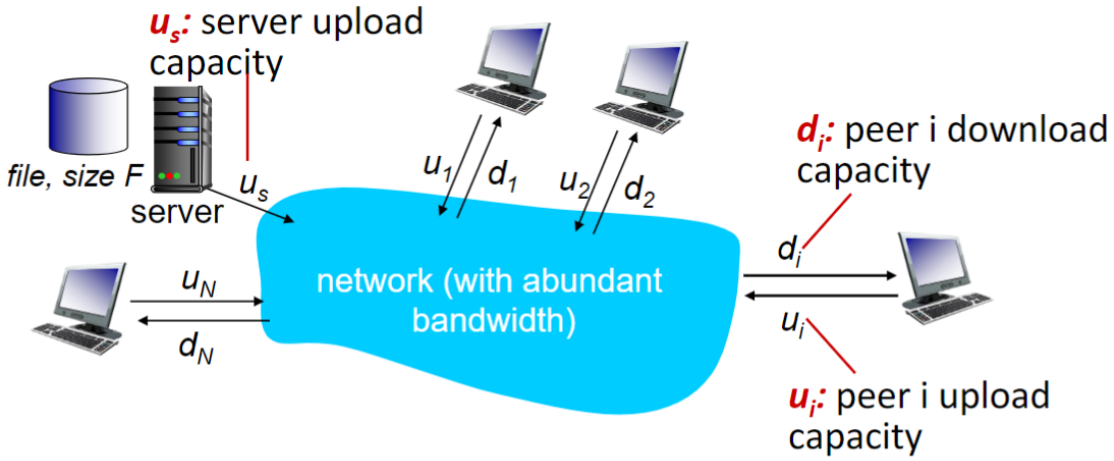
4.4.2 File Download

4.4.2.1 Approccio più conveniente: client-server o P2P?

Precedentemente abbiamo trattato come avviene la ricerca dei peer che offrono specifici contenuti. Cerchiamo di capire adesso come avviene il download. Le vie possibili sono due:

- download da un singolo peer;
- download da una molteplicità di peer.

Poniamoci una domanda: dato un particolare contenuto risulta più conveniente adottare l'approccio P2P (peer che trasmettono ad altri peer) o l'approccio client-server (introduzione di un server)?



- Abbiamo un file di una certa dimensione (size F, abbastanza grande). Vogliamo distribuire ogni nuova versione di questo file ad N utenti (dove N è un numero molto elevato).
- Definiamo la capacità di upload u_s per il server, per ogni peer definiamo:
  - Capacità di upload u_i
  - Capacità di download d_i

  I link relativi alle capacità costituiscono gli unici colli di bottiglia di nostro interesse (supporremo idealmente che la rete non costituisca collo di bottiglia – *network with abundant bandwidth*).

Utilizziamo come dato di confronto tra i due approcci il *tempo di distribuzione del file*. Esso consiste nel tempo definito dal momento in cui il file è reso disponibile agli utenti per il download all'istante in cui tutti gli utenti hanno scaricato una copia del file. Stimiamo questo valore in entrambi i contesti

- ***Client-server***
  - Si inviano $N \cdot F$ bit complessivi.
  - Sicuramente il tempo non sarà superiore al tempo necessario per fare l'upload in rete
    $$\frac{\mathrm{N} \cdot \mathrm{F}}{u_s}$$
  - Basta dire questo? Occhio al download rate di ogni singolo client (è anch'esso collo di bottiglia). Si vuole il valore più grande tra questi: dato che cambia solo il denominatore per trovare il rapporto più grande mi basta scegliere il client i-esimo tale per cui il denominatore è più piccolo
    $$\max_i \frac{F}{d_i} \Longrightarrow \frac{\mathrm{F}}{\min_i(d_i)}$$

- Otteniamo che il tempo per distribuire un file di dimensione F ad N nodi con approccio client-server è il seguente

$$d_{cs} \geq max\left\{\frac{\mathrm{N}\cdot\mathrm{F}}{u_s}, \frac{\mathrm{F}}{\min_i(d_i)}\right\}$$

- ***Peer-to-peer***
  - Si mantiene il server, che carica una copia del file in rete. Questa copia sarà ottenuta da un peer, che successivamente contribuirà inviandola totalmente o in parte ad altri peer che la richiedono.

$$\frac{\mathrm{F}}{u_s}$$

  - Vale il discorso detto prima su Internet che non costituisce collo di bottiglia
  - Ogni peer deve scaricare il file, quindi anche qua si considera

$$\frac{\mathrm{F}}{\min_i(d_i)}$$

  - Rimane da considerare l'upload in rete del file da parte dei dispositivi. Dato che un peer può contribuire totalmente o anche solo parzialmente ci si immagina il tutto come un singolo nodo avente capacità di upload $u_s + \sum u_i$. Tenendo a mente che si scambiano N copie del file otteniamo

$$\frac{\mathrm{N}\cdot\mathrm{F}}{u_s + \sum u_i}$$

  - Otteniamo che il tempo per distribuire un file di dimensione F ad N nodi con approccio P2P è il seguente

$$d_{P2P} \geq max\left\{\frac{\mathrm{F}}{u_s}, \frac{\mathrm{F}}{\min_i(d_i)}, \frac{\mathrm{N}\cdot\mathrm{F}}{u_s + \sum u_i}\right\}$$

Concludiamo le nostre riflessioni mettendo insieme le formule trovate, poniamo $N \to +\infty$

$$d_{cs} \geq max\left\{\frac{\mathrm{N}\cdot\mathrm{F}}{u_s}, \frac{\mathrm{F}}{\min_i(d_i)}\right\} \qquad d_{P2P} \geq max\left\{\frac{\mathrm{F}}{u_s}, \frac{\mathrm{F}}{\min_i(d_i)}, \frac{\mathrm{N}\cdot\mathrm{F}}{u_s + \sum u_i}\right\}$$

- **Client-server**
  Nel caso del modello client-server prevale il primo termine, l'unico tra i due dipendenti da N. Abbiamo supposto che il rapporto $\frac{\mathrm{F}}{u_s}$ sia costante.
- **Peer-to-peer**
  Nel caso del modello peer-to-peer l'unico termine dipendente da N è il terzo. Supponiamo idealmente che gli u_i siano tutti uguali. Otteniamo (tenendo a mente che $u_s \ll \mathrm{N}\cdot u_i$)

$$\lim_{N\to+\infty} \frac{\mathrm{N}\cdot\mathrm{F}}{u_s + \mathrm{N}\cdot u_i} \longrightarrow \frac{\mathrm{N}\cdot\mathrm{F}}{\mathrm{N}\cdot u_i} = \frac{\mathrm{F}}{u_i}$$

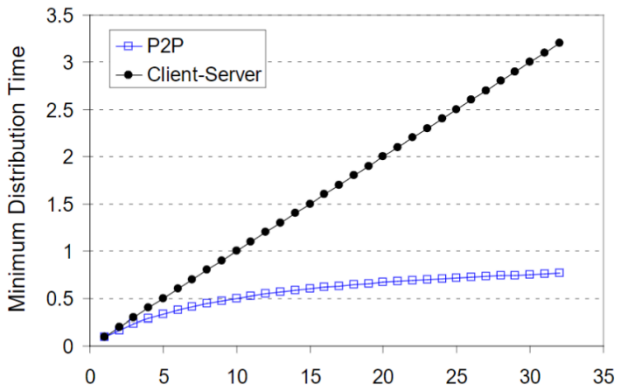
Possiamo rappresentare l'andamento col grafico a lato

- col modello client-server abbiamo una retta

$$d_{cs} = \mathrm{N}\cdot\frac{\mathrm{F}}{u_s}$$

  abbiamo detto che $\frac{\mathrm{F}}{u_s}$ è costante.
- Col modello P2P abbiamo una funzione che tende a

$$\lim_{N\to+\infty} \frac{\mathrm{N}\cdot\mathrm{F}}{u_s + \mathrm{N}\cdot u_i} = \frac{\mathrm{F}}{u_i}$$

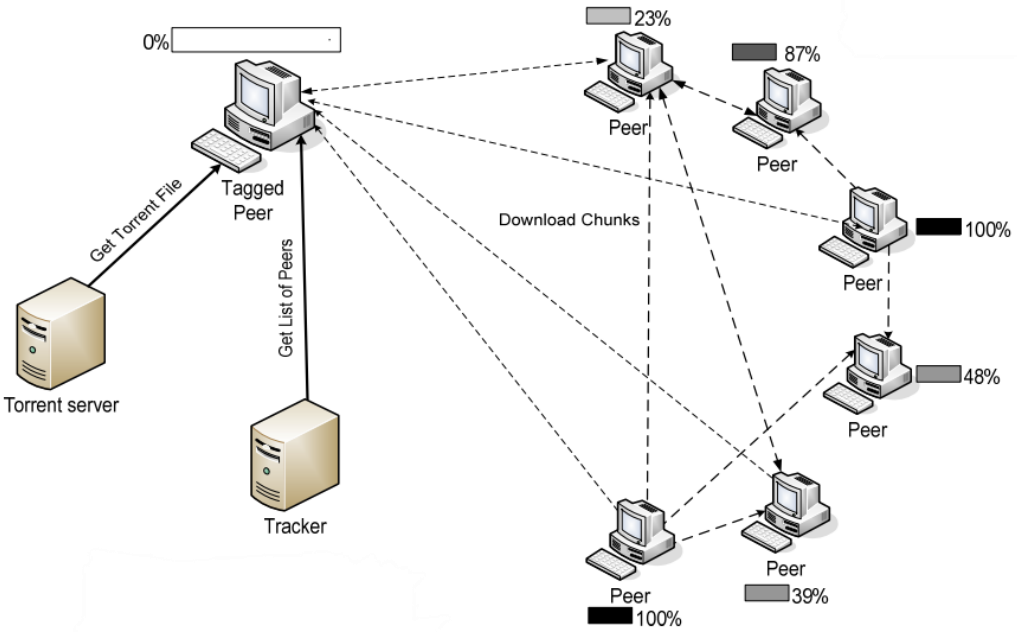


4.4.2.2 Protocollo bitTorrent

Con la spiegazione della pagina precedente abbiamo anticipato un aspetto interessante dei protocolli P2P moderni: il fatto che un peer scarichi un particolare file rivolgendosi a una molteplicità di peer. Ciò è in contrasto con l'approccio dei primi protocolli P2P:

- un peer che scarica, che potremo definire un client;
- un peer che fornisce il contenuto, che potremo definire un server.

Il protocollo bitTorrent per la distribuzione di file rappresenta una rottura rispetto a questo schema.



L'aspetto innovativo di bitTorrent è che il download di un particolare contenuto da parte di un peer (in figura il ***Tagged Peer***, che non ha ancora scaricato niente) avviene per mezzo dell'interazione con una pluralità di peer: ciascuno di questi contribuisce al download del contenuto fornendo parte dello stesso (lo scaricano e successivamente rimangono connessi a supporto degli altri peer che richiedono il contenuto).

Per prima cosa introduciamo una serie di definizioni.

- ***torrent***
  L'insieme di peer coinvolti nella distribuzione di un particolare file è detto *torrent*. Il *Tagged Peer* aderisce al torrent per scaricare un particolare contenuto. Tutti i peer hanno aderito al torrent, in momenti diversi, per poter scaricare il contenuto.
- ***licker***
  I peer appartenenti al torrent sono detti *licker* se all'istante considerato non hanno ancora scaricato l'intero contenuto. Tradotto significa "sanguisuga, leccapiedi". Scaricano il contenuto grazie al contributo degli altri peer presenti nel torrent.
- ***seeder***
  I peer appartenenti al torrent sono detti *seeder* se all'istante considerato hanno scaricato il 100% del contenuto. Sono peer generosi: pur avendo scaricato l'intero contenuto rimangono all'interno del torrent supportando gli altri peer nel download del contenuto.
- ***free rider***.
  I peer che scaricano contenuti e non contribuiscono verso altri (egoisti, penalizzati dal protocollo) peer sono detti *free rider*.
- ***tracker***
  Il *tracker* è un'entità che tiene traccia dei peer che hanno aderito a un particolare torrent. I peer si registrano presso il tracker per segnalare la loro adesione al torrent, e "rinnovano" periodicamente questa adesione. Per comodità tratteremo il tracker come un'entità centralizzata.
- ***torrent server***
  Il *torrent server* è un'entità a cui il *Tagged Peer* si rivolge per ottenere un file .torrent, avente al suo interno le informazioni utili per stabilire una connessione col Tracker.

Il file è diviso in blocchi di 256KB (dimensione tipica) detti ***chunk***. Ogni peer nel torrent fornisce a chi vuole scaricare il contenuto un certo numero di chunk.

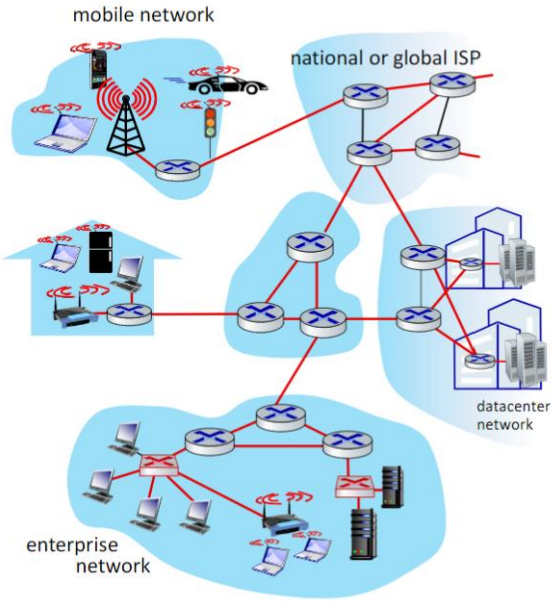
- Il tracker fornisce al Tagged peer una lista di peer attivi nel torrent.
- Il Tagged peer apre connessioni TCP con parte di questi peer (maggior numero di connessioni possibili).
- Il Tagged peer richiede e ottiene dai peer con cui ha stabilito connessione l'elenco dei chunk che sono in grado di offrire. Questa richiesta verrà lanciata periodicamente.
- Il Tagged peer analizza gli elenchi e richiede specifici chunk ai vari peer. Inizia adottando un approccio *rarest-first*: da priorità ai più rari, cioè quelli messi a disposizione da un numero basso di peer. Perché?
    - Supponiamo che il chunk sia fornito da un solo peer: se questo abbandona il torrent il chunk non è più disponibile.
    - È sensato partire dai più rari in modo tale che ogni chunk sia sempre disponibile all'interno del torrent.
- Il Tagged peer scarica chunk: ottiene progressivamente il contenuto, ma allo stesso tempo inizia ad aiutare gli altri peer fornendo i chunk già presenti nella sua memoria. In presenza di un numero elevato di richieste deve decidere a chi dare precedenza. Lo si fa secondo criterio *tit-for-tat* (pan per focaccia):
    - si selezionano i primi quattro peer che trasmettono dati al Tagged peer con velocità più alta;
    - ogni trenta secondi si seleziona in modo randomico un peer che non rientra tra quelli con cui abbiamo interagito, se questo interagisce col Tagged Peer con una velocità elevata allora questo andrà a sostituire il più lento dei quattro peer selezionati precedenti (strategia nota come *optimistically unchoke*, pensata per evitare che si selezionino sempre i soliti quattro peer);
    - se si considera la velocità è chiaro che i free riders sono penalizzati nella graduatoria (se uno non trasmette la velocità è nulla).

5 RETI A COLLEGAMENTO DIRETTO (DIRECT CONNECTION NETWORKS)

5.1 Introduzione

5.1.1 Oggetto del capitolo

Nei capitoli precedenti abbiamo distinto periferia e nucleo della rete. Abbiamo visto una rete è costituita da nodi collegati per mezzo di link wired o wireless.



Pila protocollare
Appplication *messages*
Transport *segments*
Network *datagrams*
Link / Datalink *frames*
Physical

Il capitolo pone enfasi su come interagiscono due nodi adiacenti, nel trasferimento di dati. Si parla anche di interazione tra più di due nodi nel contesto di reti aventi piccola estensione territoriale (i nodi non si trovano a distanze elevate tra loro).

5.1.2 Collegamento tra due dispositivi

5.1.2.1 Collegamento ideale (affidabile)

Immaginiamoci due computer connessi da un link fisico.

È nostro interesse trasmettere una sequenza di bit (zeri e uno) attraverso il link fisico. Lo facciamo rappresentando zeri e uno nella forma di segnali fisici: nel caso di cavi in rame il segnale fisico è un segnale elettrico (per esempio valori di tensione, uno sopra una certa soglia e zero sotto un'altra soglia), nel caso della fibra è un segnale luminoso. Supponiamo di lavorare con segnali elettrici, poniamo:

- un trasmettitore a valle del computer mittente che codifica il messaggio (trasformazione delle sequenze binarie in segnali elettrici);
- un ricevitore a capo del computer destinatario che decodifica il messaggio (trasformazione dei segnali elettrici in sequenze binarie).

La cosa non è semplice perché vogliamo utilizzare il link fisico per trasmettere sequenze molto lunghe. Si consideri che in generale i dispositivi sono dotati di *transceiver* (non hanno solo il trasmettitore o solo il ricevitore) in quanto la comunicazione deve essere bidirezionale: entrambi i computer devono avere la possibilità di comunicare l'uno con l'altro. Le comunicazioni bidirezionali sono definite:

- *half-duplex* nel caso in cui i due soggetti non possano comunicare l'uno con l'altro in simultanea;
- *full-duplex* nel caso in cui i due dispositivi possano comunicare l'uno con altro in simultanea.

Per motivi di semplicità tratteremo comunicazione half-duplex.

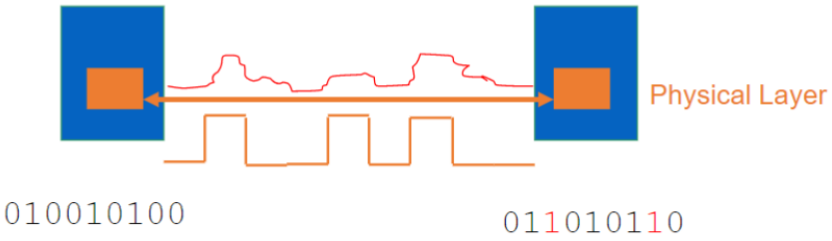
5.1.2.2 Collegamento reale (inaffidabile)

La visione appena introdotta è solo ideale. Nell'introdurre i concetti fondamentali non abbiamo considerato una serie di problematiche:

- il segnale si attenua all'aumentare delle distanze (si pensi alla lezione in aula, più si è lontani dalla cattedra maggiori sono le difficoltà ad ascoltare il docente con chiarezza);
- il segnale si sovrappone ai cosiddetti segnali di disturbo, dovuti a sorgenti nell'ambiente esterno (sempre parlando di lezione in aula, il docente che parla con gli studenti che nel mezzo fanno confusione).

Il risultato è il seguente:

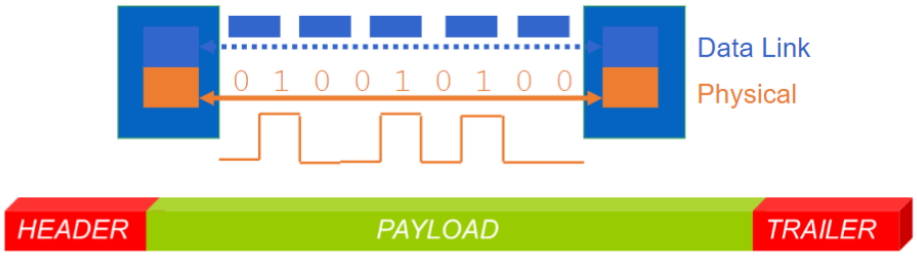
- il dispositivo mittente codifica la sequenza di bit;
- il **link fisico è inaffidabile** per le cose dette e quindi il segnale elettrico viene alterato;
- il dispositivo destinatario riceve segnali elettrici diversi da quelli trasmessi, a seguito di decodifica dei segnali elettrici viene ottenuta una sequenza di bit diversa da quella trasmessa dal mittente.



È inevitabile lavorare con un link non ideale: compito nostro è individuare le soluzioni in grado di ridurre il cosiddetto *biterror rate* del link fisico, cioè la percentuale di bit ricevuti non correttamente rispetto alla quantità di bit inviati. L'idea non è eliminare l'errore, ma fare in modo che questo diventi trascurabile.

5.1.3 Divisione delle sequenze di bit in trame (framing)

Maggiore è la lunghezza della sequenza di bit maggiore è il rischio che questa risulti alterata nel passaggio dal link fisico. La prima cosa che si fa è dividere la sequenza di bit in trame (sequenze più piccole), in modo che la probabilità di avere bit non alterati sia non trascurabile.



- Mittente e destinatario concordano una lunghezza delle trame.
- Ogni trama è caratterizzata da:
  - *Payload* (contenuto vero e proprio della trama);
  - *header* e *trailer* (sequenza iniziale e sequenza finale che contengono bit utili alla trasmissione del messaggio[8]).

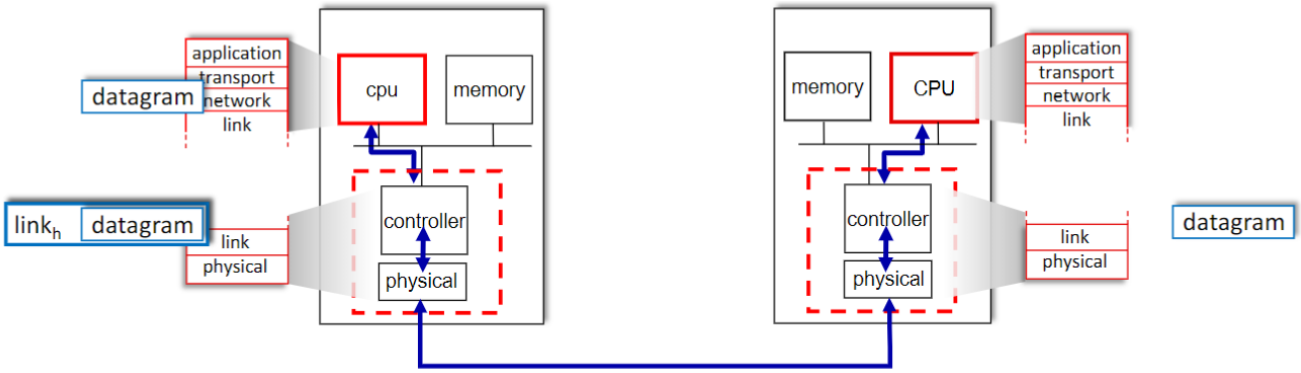
Il mittente vuole trasmettere una sequenza di bit: la prende e la scompone in trame, inserendo header e trailer, infine trasmette ogni trama sul link fisico. Il destinatario riceve le varie trame: elimina header e trailer e concatena i payload ricostituendo la sequenza di bit.

[8] Si prenda ad esempio i bit di ridondanza, di cui parleremo a breve. Stanno nel trailer

5.1.4 Implementazione del livello data link all'interno degli host

Il livello è implementato in ogni singolo host, che presenta almeno una *Network Interface Card*, cioè la scheda di rete! L'implementazione non è solo hardware, ma anche software (si pensi al driver della scheda di rete).

Le interfacce di due dispositivi comunicano tra di loro per mezzo del link fisico:



- La CPU comunica all'interfaccia, per mezzo del bus, il datagram da trasmettere. La sequenza di bit (datagram) viene organizzata in frame. Ogni frame è trasmesso sul link fisico.
- L'interfaccia dell'host destinatario riceve il frame, che viene decodificato (si riottiene il datagram) e trasmesso alla CPU per mezzo del bus.

Sia mittente che destinatario svolgeranno operazioni (di cui parleremo nella sezione successiva) atte a mitigare l'inaffidabilità del canale di comunicazione.

5.2 Mitigazione dell'inaffidabilità del link

5.2.1 Servizi offerti dal livello data link

Il livello data link offre i seguenti servizi:

- ***Error detection***
  Si introducono tecniche in grado di segnalare la presenza di errori (implementazione da lato di chi riceve trame)
- ***Retransmission***
  Dopo aver individuato errori (*error detection*) in un frame si chiede un nuovo invio dello stesso. Cosa buona se il link non è rumoroso e ha un minimo di affidabilità (in caso contrario sarebbe altamente probabile ricevere un'altra sequenza corrotta).
- ***Error correction***
  Non ci si limita ad attestare l'integrità o meno del pacchetto, ma si cerca di intervenire sulla sequenza ricevuta correggendo gli errori. Cosa utile perché permette di evitare la retransmission.

5.2.2 *Error detection*

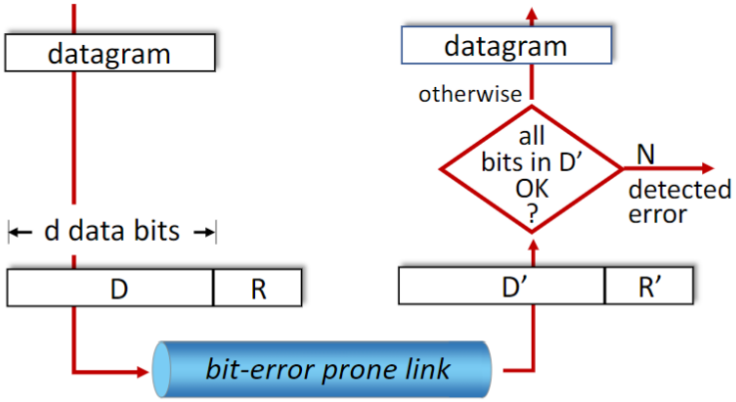
5.2.2.1 Strategia di base

Abbiamo un datagram (blocco di bit) che un mittente vuole trasmettere a un destinatario per mezzo di un link inaffidabile (*bit-error prone link*).

- I bit D consistono nei bit del datagram che l'host mittente vuole trasmettere.
- Si aggiungono i *bit di ridondanza* R, che sono calcolati per mezzo di un algoritmo concordato dai due interlocutori. L'input dell'algoritmo è la sequenza D.

Precisamente

- Il mittente esegue l'algoritmo concordato ponendo in ingresso D: ottiene R. La concatenazione di D ed R viene trasmessa sul link di comunicazione.



- Il destinatario riceve la sequenza. Sa (lo ha concordato col mittente) che certi bit sono i bit di ridondanza, mentre gli altri sono il contenuto effettivo. Si parla di D' ed R' (e non di D ed R) in quanto i due blocchi potrebbero essere stati alterati. Viene eseguito lo stesso algoritmo ponendo in ingresso D':
  - se l'algoritmo restituisce R' allora **si potrebbe pensare** che non ci siano state manipolazioni;
  - se l'algoritmo restituisce una sequenza diversa da R' allora sicuramente ci sono errori.

  Casi possibili:
  - viene alterato solo D'
  - viene alterato solo R'
  - vengono alterate entrambe le sequenze.

  Si pone al condizionale ("si potrebbe pensare") in quanto nell'ultimo caso potrebbe succedere che `algoritmo(D')=R'`
  anche in presenza di alterazioni (R' alterato in modo tale da risultare `algoritmo(D')`, cosa rara).

5.2.2.2 Criteri per la scelta dell'algoritmo di error detection

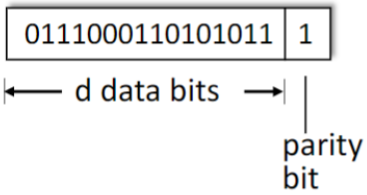
La scelta dell'algoritmo di error detection è influenzata da:

- il *biterror rate* del link (che è caratteristica del link fisico adottato e ne indica l'affidabilità)
- costo dell'algoritmo
  - complessità computazionale (come sempre)
  - numero di bit di ridondanza (maggiore è il numero di bit, maggiore è l'affidabilità dell'algoritmo, ma maggiore è anche l'occupazione di memoria).

5.2.2.3 Single bit parity (Controllo di parità)

Il controllo di parità consiste nell'introdurre un singolo bit di ridondanza, che avrà un certo valore a seconda della parità concordata:

- parità pari, si imposta il bit di ridondanza in modo tale da avere un numero pari di bit settati;
- parità dispari, si imposta il bit di ridondanza in modo tale da avere un numero dispari di bit settati (o un numero pari di bit nulli).



La probabilità di errore è alta:

- con parità pari si registrano variazioni solo se il numero di bit settati è dispari;
- con parità dispari si registrano variazioni solo se il numero di bit settati è pari.

5.2.2.4 checksum (somma di controllo)

Il *checksum* richiede un numero maggiore di bit di ridondanza (solitamente 16 bit). Si consideri l'esempio a lato, dove per comodità ridurremo il numero di bit da 16 a 4.

B= 1010 0011 1010

1010 +
0011 =

1101 +
1010 =

~~1~~0111 +
↳1 =

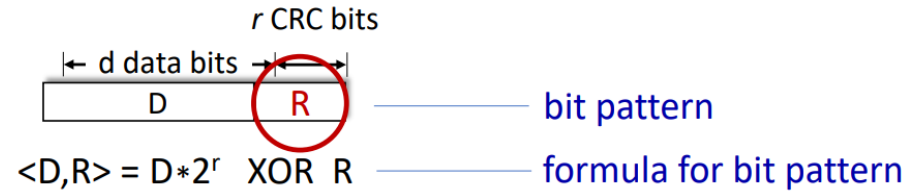
1000 ~
0111

- Si divide il contenuto del pacchetto in sequenze da 4 bit l'uno. Ogni sequenza ottenuta è trattata come un numero intero
- Si sommano i blocchi, e ci si sbarazza dell'eventuale cifra significativa in più andando a svolgere un'ulteriore somma.
- Si conclude complementando ogni singolo bit: il risultato è la sequenza R da trasmettere al destinatario assieme a D.
- Il destinatario, ricevuto il pacchetto, svolge le stesse operazioni partendo da D': se il blocco ottenuto è uguale al blocco R' ricevuto allora non si hanno errori.

La probabilità di errore è minore rispetto a prima, ma è ancora possibile assumere come corretto un pacchetto che non lo è (per le stesse ragioni).

5.2.2.5 Cycle Redundancy Check (CRC)

Il più complesso e potente tra gli algoritmi che proponiamo.

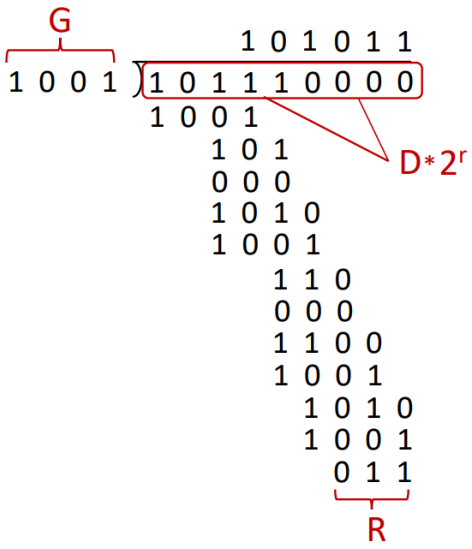


Gli elementi di partenza sono i seguenti:

- la sequenza D, che ha un numero di bit non noto a priori e che trattiamo come un numero binario;
- la sequenza generatrice G, che i due interlocutori concordano (*sender* e *receiver*) e che avrà r+1 bit (dove r è il numero di bit della sequenza R).

Il sender deve scegliere la sequenza R (ribadiamo, di r bit) in modo tale che la concatenazione di bit $< D, R >$ risulti esattamente divisibile per il generatore G. Il receiver, ricevuta la sequenza dal livello fisico, la divide per G: se l'operazione restituisce un resto diverso da zero allora il pacchetto presenta errori.

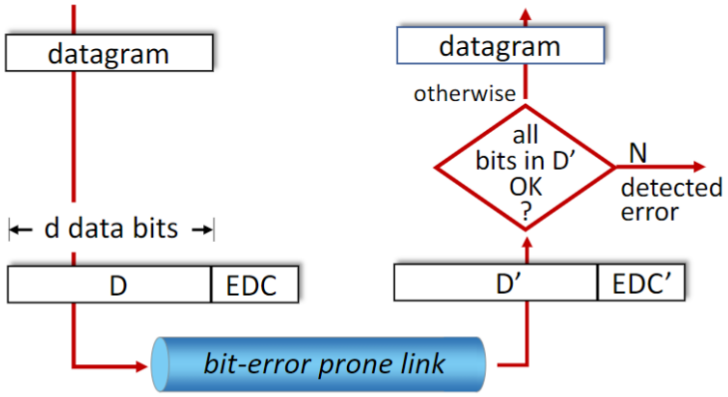
- **Operazioni nell'aritmetica modulo 2** [Ricordare da Reti Logiche].
  Le operazioni descritte sono svolte in aritmetica modulo 2.
  - Addizione e sottrazione sono svolte con l'operatore XOR, poichè in CRC si ignorano prestiti e riporti. Esempi:
    ```
    1011 XOR 0101 = 1110 -> 1011 - 0101 = 1110
    1001 XOR 1101 = 0100 -> 1001 - 1101 = 0100
    ```
  - La moltiplicazione tra un numero binario e 2^r consiste in una traslazione a sinistra del numero binario (aggiunta di r zeri sulla destra)
  - Otteniamo quindi la sequenza $< D, R >$ nel seguente modo (concatenazione)
    $$D \times 2^r \text{ XOR } R = < D, R >$$
- **Riformulazione del problema**.
  Data la seguente uguaglianza
  $$D \times 2^r \text{ XOR } R = < D, R >$$
  Vogliamo trovare R tale che
  $$D \times 2^r \text{ XOR } R = nG$$
  che possiamo porre anche nel seguente modo
  $$D \times 2^r = nG \text{ XOR } R$$
  Per trovare R calcoliamo quanto segue
  $$R = \text{resto}\left(\frac{D \times 2^r}{G}\right)$$



Esempio di calcolo di R a partire da $D2^r$ e G

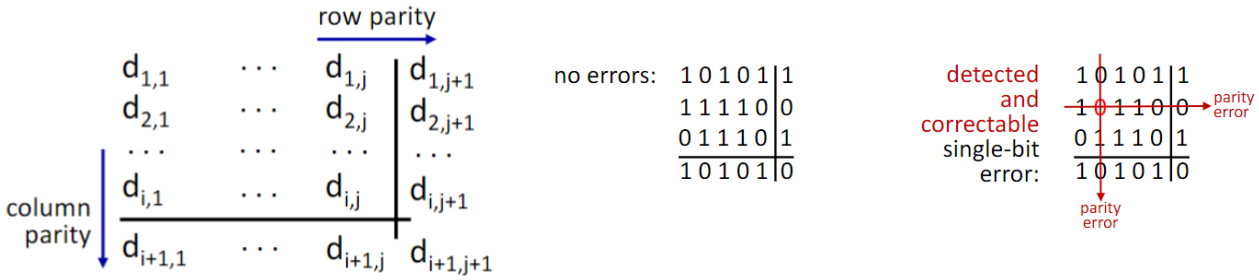
5.2.3 Error correction: strategia e dimension parity checking

Nella error correction si mantiene lo stesso approccio della error detection, ma i bit di ridondanza sono utilizzati non solo per affermare la presenza o meno di errori, ma anche per localizzare gli errori. Fare questa cosa risulta estremamente vantaggioso, in quanto evitiamo la retransmission e soprattutto è semplice (un bit sbagliato uguale a zero viene settato, un bit sbagliato uguale ad uno viene resettato).



Chiaramente la cosa ha costo maggiore rispetto al fare solo error detection. La tecnica più conosciuta di error correction (e la migliore dal punto di vista della error detection) è la *two-dimensional bit parity* (controllo di parità a due dimensioni).

- Consideriamo una sequenza di bit e la disponiamo nella forma di una matrice di dimensioni i e j.
- Si aggiungono i bit di ridondanza nel modo mostrato in figura, l'esito finale è una matrice di dimensioni i+1 e j+1. Si parla di metodo proattivo in quanto i bit si aggiungono subito, a priori e in ogni caso (quindi abbiamo un costo indipendentemente dalla presenza o meno di errori).



Si usa dove la *biterror rate* è molto elevata, non è il più utilizzato

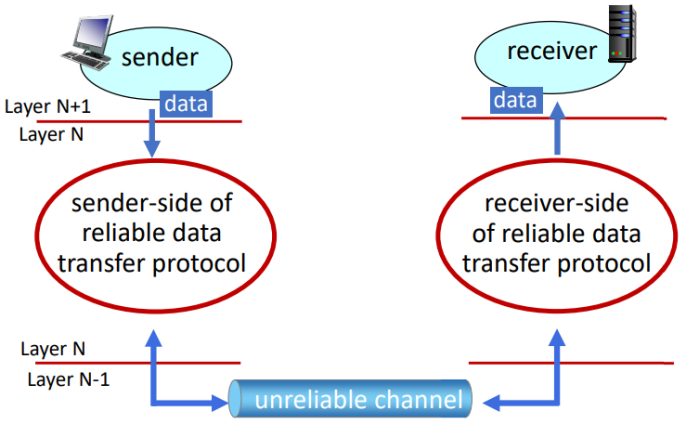
5.3 Reliable Data Transfer Protocol

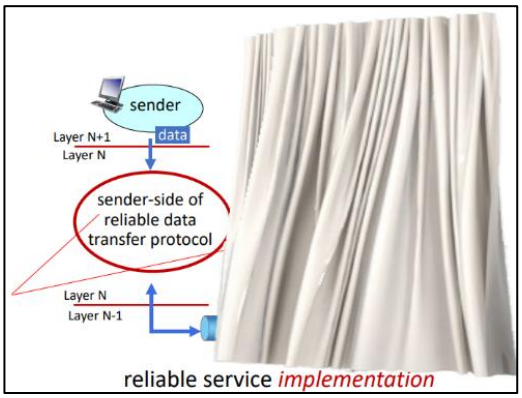
5.3.1 Introduzione

Il Reliable Data Transfer Protocol permette la realizzazione di un servizio affidabile all'interno di un canale inaffidabile.

Il protocollo si articola in lato sender e lato receiver e mira a creare l'astrazione di un servizio affidabile per mezzo di alcune operazioni, ad esempio la *retransmission* (citata nella sezione precedente). Le azioni svolte dipendono soprattutto dall'inaffidabilità del canale di comunicazione:

- errori introdotti nel livello sottostante;
- pacchetti non ricevuti dal destinatario (non è tanto l'errore, ma non ricevere proprio il pacchetto);
- pacchetti ricevuti dal destinatario in un ordine diverso da quello di invio.





Si consideri un ulteriore aspetto: il *sender* non sa niente di ciò che succede nel *receiver*, e viceversa. L'implementazione tiene conto esclusivamente delle informazioni conosciute e di quelle eventualmente scambiate per mezzo di pacchetti.

Premessa fondamentale. Nelle spiegazioni che seguono si parla di pacchetti in generale e non di frame in quanto questo protocollo può essere implementato in qualunque livello della pila protocollare. La descrizione è posta in generale, ma l'implementazione che ci interessa è quella a livello *data link*.

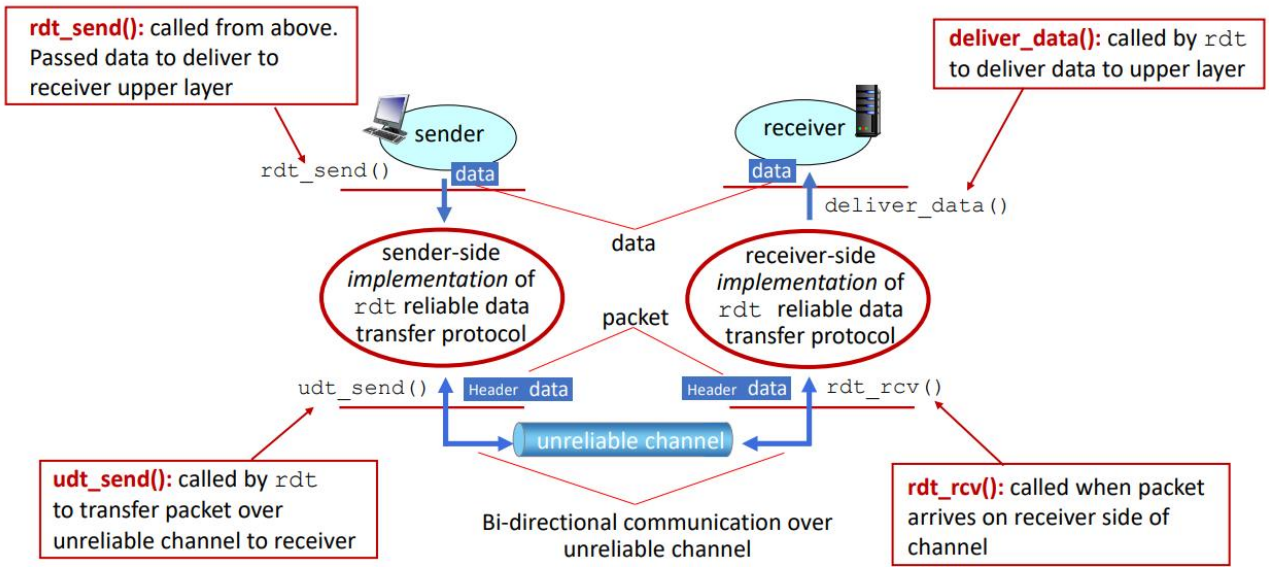
5.3.2 Premessa ripasso: formalismo dell'automa a stati finiti

Spiegheremo il protocollo ricorrendo al formalismo dell'automa a stati finiti, che abbiamo visto per la prima volta a Reti logiche e formalizzato a Ingegneria del software. Poniamo per comodità la seguente immagine:

- Si disegna un grafo con nodi e archi.
- Un nodo rappresenta uno dei possibili stati in cui può trovarsi l'automa.
- Gli archi rappresentano i possibili eventi che determinano il passaggio dell'automa da uno stato a un altro. È possibile indicare (nel modo indicato in figura) eventuali azioni svolte dall'automa durante la transizione di stato.

5.3.3 Interfacce introdotte

Si introducono le seguenti interfacce (*con nomi tutt'altro che intuitivi*, cit.) secondo il modello a strati (ricordarsi che un livello comunica con quello superiore o quello inferiore per mezzo dell'invocazione di interfacce): si parla di astrazione in quanto il sender pensa di comunicare direttamente col receiver, ignorando quanto avviene nei livelli inferiori.



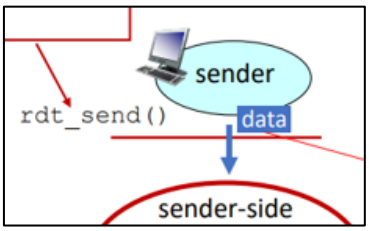
5.3.4 Prima versione (rdt1.0, sbagliata)

Costruiamo una prima versione del protocollo supponendo che il canale sia affidabile:

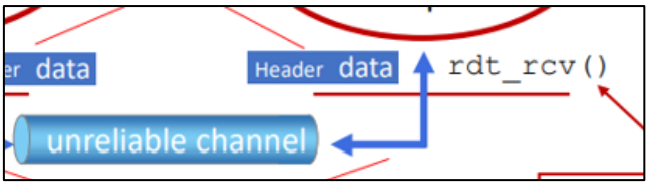
- niente errori nei bit;
- niente perdita di pacchetti.

Adotteremo nelle prime versioni del protocollo l'approccio ***stop and wait***: una volta inviato il pacchetto il sender attende la risposta del receiver prima di inviare altro (approccio che vedremo essere fallimentare).

- ***Sender***.
  Il sender attende che il livello sovrastante trasmetta dei dati: se succede (invocazione della `rdt_send` nel livello superiore) si creano i pacchetti e li si inviano sul link inaffidabile invocando la `udt_send`.



- ***Receiver***.
  Il receiver attende di ricevere pacchetti dal livello fisico (il link, invocazione della `rdt_rcv`): quando succede provvede ad estrarre i dati dal pacchetto e a consegnare i dati al livello superiore (il destinatario) invocando la `deliver_data`.



5.3.5 Seconda versione (rdt2.0, rtd2.1, rtd2.2, sbagliata)

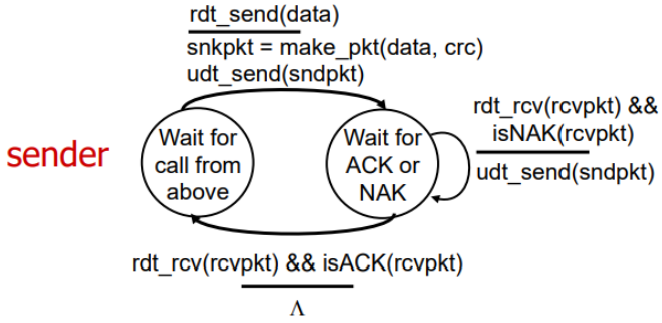
5.3.5.1 rdt2.0 (Individuazione degli errori e retransmission)

La versione precedente è sbagliata in quanto non tiene conto dell'inaffidabilità del link di comunicazione.

Nella nuova versione si introduce la *error detection* e la *retransmission*. A questo segue un problema: come può il sender sapere l'esito della trasmissione se questo non sa niente di ciò che succede nel *receiver*?

- sender e receiver si trasmettono ulteriori pacchetti.
- Il receiver svolge error detection per verificare la presenza di errori nel pacchetto.
- In ogni caso il receiver dovrà comunicare al sender, per mezzo di un ulteriore pacchetto, se ha trovato errori. Due possibili risposte:
  - *acknowledge* (ACK) per segnalare al sender che il pacchetto ricevuto è ok;
  - *negative acknowledge* (NAK) per segnalare al sender che il pacchetto ricevuto ha errori.
- Il sender ritrasmette il pacchetto incriminato se riceve il NAK.

Descriviamo il sender col solito formalismo!

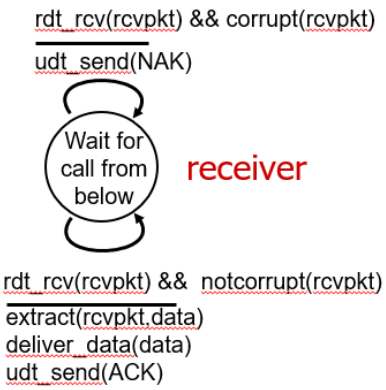


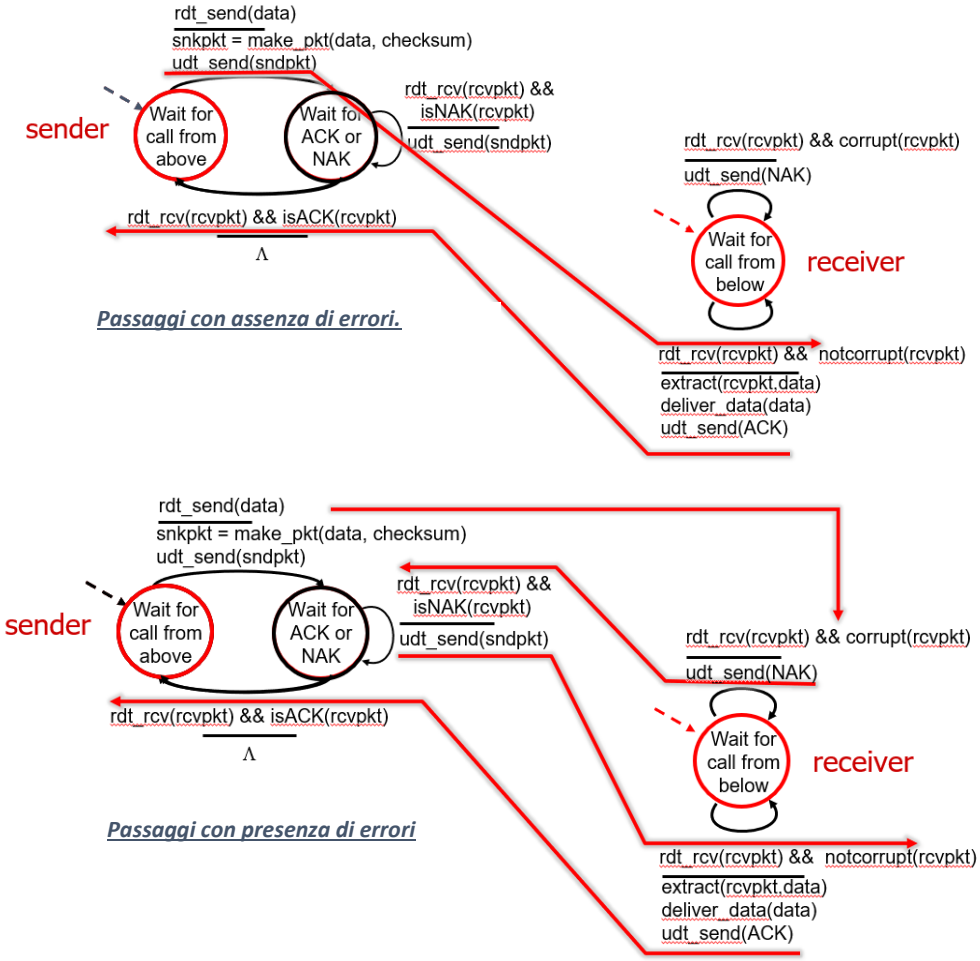
- Lo stato iniziale è il solito visto nel rdt1.0: come prima attende l'invocazione della `rdt_send` e nel caso invia il pacchetto invocando la `udt_send`.
- La novità sta nel nuovo stato "Wait for ACK or NAK", dove il sender si pone dopo aver inviato il pacchetto.
- Il sender dovrà ricevere ACK o NAK dal receiver, quindi attende invocando la `rdt_rcv`. Ricevuto il pacchetto verifica se si tratta di ACK o NAK e agisce di conseguenza (occhio alle funzioni `isNAK` e `isACK`).
- Si mantiene l'approccio stop and wait: dopo aver inviato il pacchetto non se ne inviano altri fino a quando non riceverà un ACK o un NAK.
  - Se riceve l'ACK il sender ritorna nello stato iniziale e aspetta un nuovo pacchetto dal livello superiore (il simbolo Λ segnala che non ci sono azioni da fare durante la transizione di stato).
  - Se riceve il NAK il sender ritrasmette il pacchetto precedentemente inviato (*retransmission*) e rimane in attesa di un nuovo ACK/NAK (quindi non cambia il suo stato)

Consideriamo adesso il receiver!

Nel receiver rimane solo lo stato "Wait for call from below", ma gli eventi possibili sono due:

- Invocazione della `rdt_rcv` e ricezione di un pacchetto corrotto (funzione `corrupt`): lo stato non cambia e si invia un NAK (`udt_send`) al sender.
- Invocazione della `rdt_rcv` e ricezione di un pacchetto non corrotto (funzione `notcorrupt`): il pacchetto viene trattato come già visto nella rdt1.0 (funzioni `extract` e `deliver_data`), ma in aggiunta si invia l'ACK (`udt_send`).





La versione è già un miglioramento, ma non tiene minimamente conto dell'idea che gli stessi ACK e NAK possano essere alterati durante il passaggio sul link. In particolare:

- se il receiver trasmette il NAK e il bit viene alterato allora il sender riceverà un ACK, quindi il sender non si accorge che il receiver ha segnalato errori e non ritrasmette il pacchetto;
- se il receiver trasmette l'ACK e il bit viene alterato allora il sender riceverà un NAK, quindi il sender invierà nuovamente il pacchetto nonostante non siano stati individuati errori.

5.3.5.2 rdt2.1 (Manipolazione degli ACK/NAK e pacchetti duplicati)

La nuova versione mira a risolvere il problema precedentemente spiegato. Quale soluzione possiamo adottare?

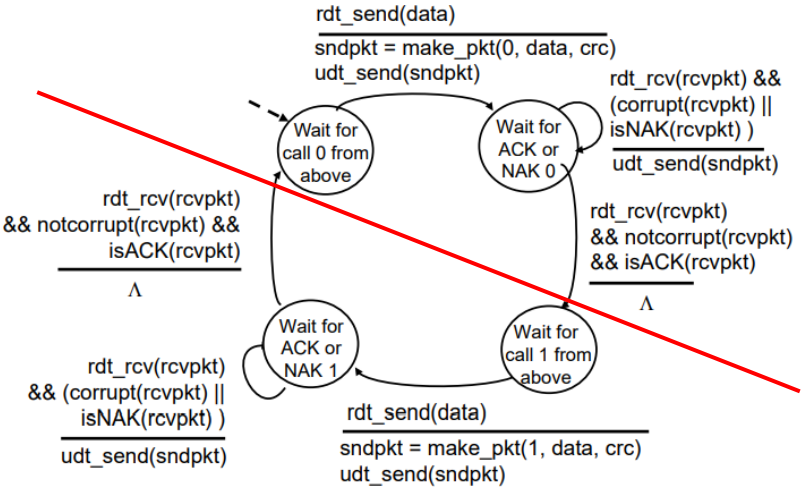
- **Error correction sugli ACK e sui NAK: in caso di errore si corregge e otteniamo il valore corretto**. Non ci piace perché si introducono bit di ridondanza e non è detto che funzioni sempre.
- **Error detection su ACK e NAK e retransmission degli ACK e NAK in caso di errore**. Problema: si entra in un loop da cui non si esce più (chi mi valida gli ACK e i NAK inviati successivamente? E quelli ancora dopo? *Fan dei fan delle strutture ricorsive*)
- **Approccio definitivo**. L'approccio adottato è il seguente: si fa error detection, ma ogni individuazione di errore viene trattata come un NAK. Questo significa che il sender invierà nuovamente il pacchetto (e non si limiterà ai soli ACK e NAK come prima)

L'ultima soluzione ha un effetto collaterale: introduce duplicati di pacchetti (si rompe ancora di più l'affidabilità del link di comunicazione, in presenza di duplicati ciò che viene trasmesso è diverso da ciò che viene ricevuto). Dobbiamo essere in grado di gestire anche i duplicati.

- Si introduce un numero di sequenza nei vari pacchetti spediti, in modo tale che sia sender che ricever possano individuare duplicati (si ricordi che il sender non può sapere a priori cosa succede nel receiver, e viceversa).

- Sia il sender che il receiver pongono nella loro memoria il contatore di numero di sequenza, che incrementano non appena inviano un pacchetto. Il contatore indica al sender/receiver quale numero di sequenza dovrà avere il prossimo pacchetto che riceveranno.
    - Se il numero di sequenza è quello che ci si aspetta non abbiamo un duplicato.
    - Se il numero di sequenza è inaspettato, cioè il numero di sequenza è già stato inviato precedentemente, allora abbiamo un duplicato e si agisce nei modi detti precedentemente.
- Chiaramente introdurre il numero di sequenza (nuovo campo) ha un costo: la sua dimensione. Quanto dovrà essere lungo questo campo? Si consideri la politica stop and wait, inoltre siamo certi che il sender non invierà un ulteriore pacchetto fino a quando non avrà gestito il precedente (prima di trasmetterne un altro attende l'ACK o il NAK). Morale della favola: è sufficiente un solo bit, si ha incremento modulo 2.

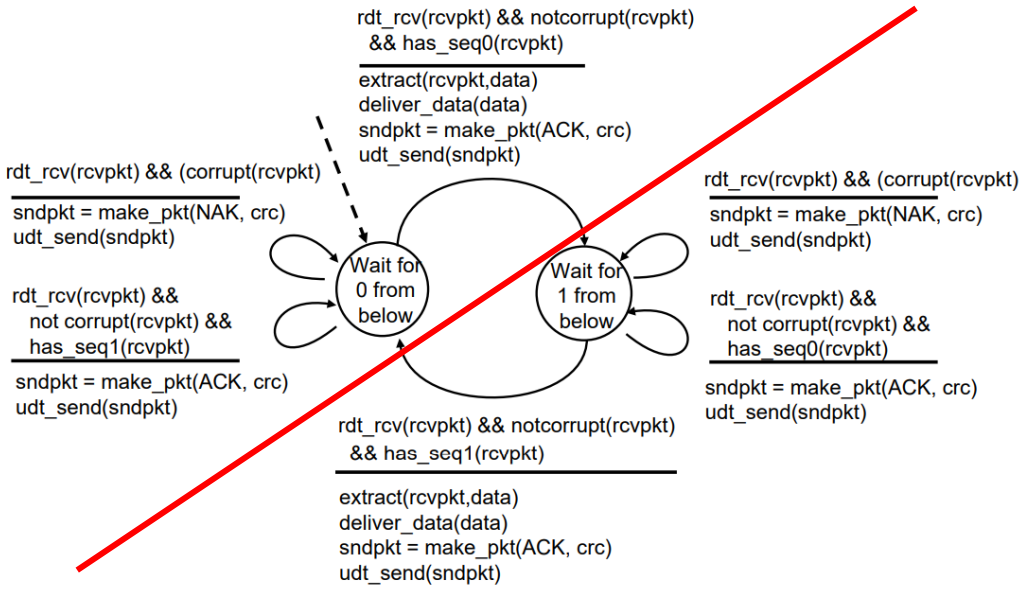
Consideriamo il sender:



- La prima cosa che osserviamo è che il numero di stati che può assumere il sender raddoppia, e che lo schema realizzato col formalismo è simmetrico rispetto alla linea tracciata. Questo perché il contatore del numero di sequenza (che abbiamo detto essere di un solo bit) è posto in modo intrinseco negli stati:
    - quando il sender assume lo stato "Wait for call 0 from above" si aspetta un pacchetto avente numero di sequenza zero;
    - quando il sender assume lo stato "Wait for call 1 from above" si aspetta un pacchetto avente numero di sequenza uno;
    - quando lo stato si trova negli altri due stati si aspetta un ACK o un NAK (a cui non associamo un numero di sequenza)
- Prendiamo per comodità gli stati solo gli stati sopra la riga (sotto è uguale per la simmetria).
    - Lo stato di partenza è "Wait for call 0 from above": il pacchetto ricevuto dal livello superiore sarà inviato con numero di sequenza zero. Per ora nulla di diverso, a parte la presenza del numero di sequenza.
    - Nello stato successivo il sender attende di ricevere un ACK o un NAK. Attenzione agli eventi possibili:
        - ricezione dell'ACK (`rdt_rcv`) non corrotto (`notcorrupt`);
        - ricezione del NAK (`rdt_rcv`) non corrotto (`isNAK`);
        - ricezione di un ACK o un NAK (`rdt_rcv`) corrotto (`corrupt`), nello schema gestito assieme all'evento precedente.

      Col primo evento il sender incrementa il contatore, quindi passa allo stato "Wait for call 1 from above", con gli altri due eventi il contatore non viene incrementato e si rinvia nuovamente il pacchetto coi dati.
    - I duplicati sono gestiti correttamente: finchè rimaniamo in questi stati il sender ignorerà qualunque pacchetto con numero di sequenza 1.

Consideriamo adesso il receiver



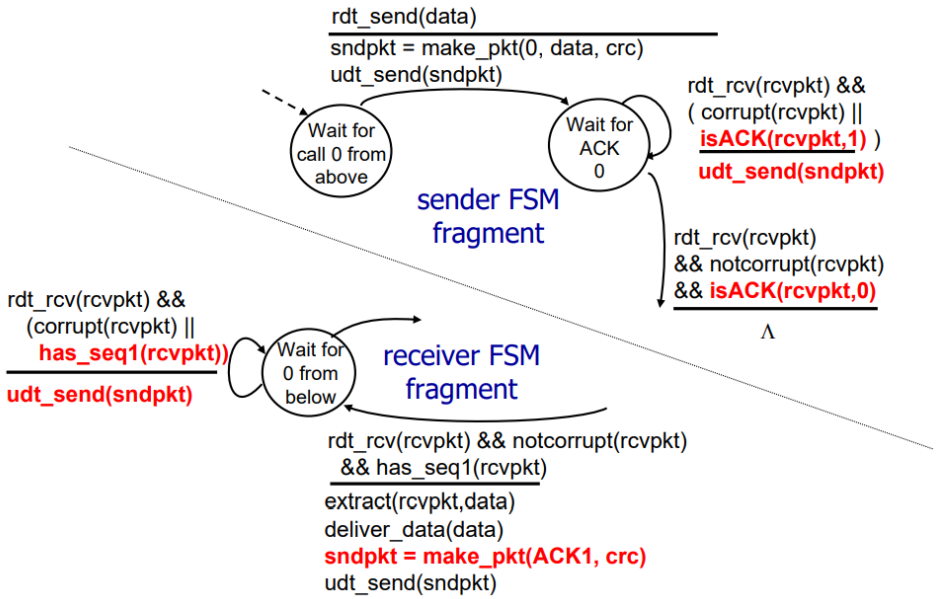
- Anche qua simmetria, si raddoppiano gli stati per introdurre il contatore del numero di sequenza.
- Gli eventi possibili sono i seguenti:
  - Il receiver ha ricevuto un pacchetto (`rdt_rcv`) non corrotto (`notcorrupt`) e con numero di sequenza zero (`has_seq0`), si fanno le stesse cose già viste nelle versioni precedenti e si cambia stato;
  - Il receiver ha ricevuto un pacchetto (`rdt_rcv`) corrotto (`corrupt`) , si scarta il pacchetto e si invia un NAK al sender (`udt_send`);
  - Il receiver ha ricevuto un pacchetto (`rdt_rcv`) non corrotto (`not_corrupt`) e con numero di sequenza uno (`has_seq1`), si scarta il pacchetto e si invia un ACK di avvertimento al sender (`udt_send`, lo si invia perché il duplicato è dovuto a una ricezione non corretta dell'ACK, o addirittura a una non ricezione).

5.3.5.3 rdt2.2 (Semplificazione con rimozione del NAK)

Vogliamo realizzare una versione più semplice ed elegante dell'algoritmo precedente dove utilizziamo esclusivamente gli ACK.

- Si mantiene il numero di sequenza, ma lo si pone anche negli ACK e nei NAK, contrariamente alla versione precedente.
- **Se ci troviamo negli stati relativi a numero di sequenza zero allora il NAK è l'ACK di tipo 1.**
- **Se ci troviamo negli stati relativi a numero di sequenza uno allora il NAK è l'ACK di tipo 0.**

Qui di seguito poniamo parte degli schemi visti in rdt2.1, con le modifiche evidenziate in rosso grassetto.



5.3.6 Terza versione (rtd3.0, definitiva)

5.3.6.1 Spiegazione

Nella terza versione, che è quella definitiva, si considera un'ulteriore ipotesi: che i pacchetti non arrivino proprio a destinazione. Due i possibili eventi:

- il sender invia un pacchetto e il ricevitore non lo riceve;
- il ricevitore invia un ACK o un NAK e il sender non lo riceve.

Se operiamo a livello trasporto il fenomeno della perdita di pacchetto è abbastanza comune: si consideri che i pacchetti devono essere mantenuti per un certo periodo in memoria (i pacchetti trasmessi rimangono in memoria finchè non ci sarà ACK), cosa succede se il buffer è pieno? Perdita! Dobbiamo introdurre delle modifiche che ci permettano di individuare perdite di pacchetti.

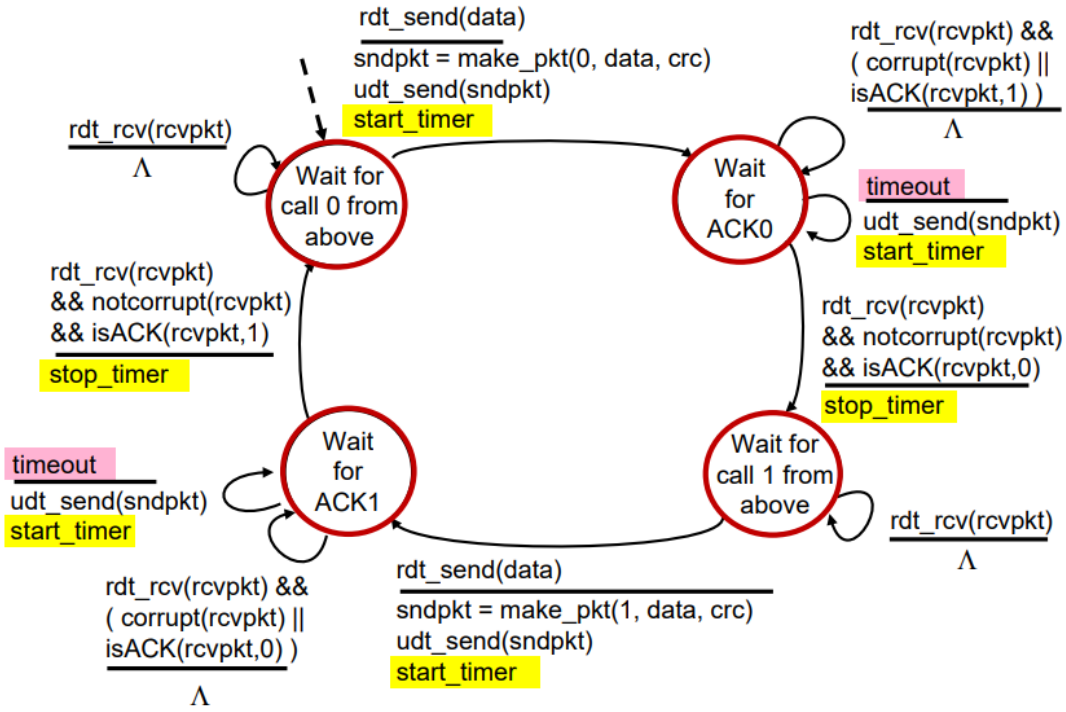
$$\text{Timeout} > \text{RTT}$$

- Si introduce un timeout, che ha come valore minimo il Round Trip Time. Nel calcolo si considera:
  - il tempo necessario al pacchetto per andare dal trasmettitore al ricevitore;
  - il tempo necessario all'ACK per andare dal ricevitore al trasmettitore.

  Si considera ritardo di processing, ritardo di trasmissione e ritardo di propagazione per entrambi i contributi. Si deve considerare anche il tempo necessario per fare *error detection*. A livello datalink possiamo determinare RTT con un'approssimazione ragionevole, mentre a livello trasporto la cosa non è più possibile (a quel punto si fa una stima, si guarda al passato per predire il futuro).
- Se scatta il timeout assumo che il pacchetto non sia arrivato a destinazione, quindi lo invio nuovamente. Si osservi che:
  - un timeout troppo lungo rende il trasferimento troppo lento;
  - un timeout troppo corto provoca formazione inutile di duplicati.
- Necessario mantenere in memoria un timer.

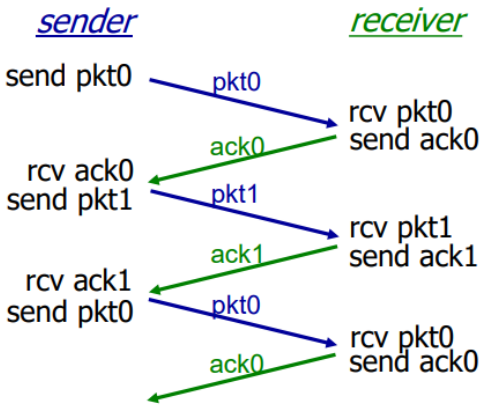
Consideriamo il sender, dove lo schema è quello della versione rdt2.2., ma con le aggiunte evidenziate:

- Aggiunta dell'evento `timeout` negli stati "Wait for ACK"
- Introduzione delle invocazioni di `start_timer` quando si invia un nuovo pacchetto
- Introduzione delle invocazioni di `stop_timer` quando abbiamo ricevuto il relativo ACK.

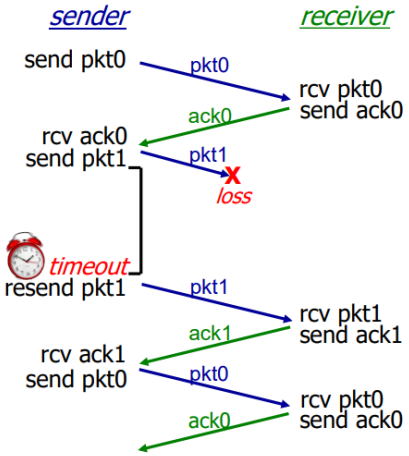


5.3.6.2 *Esempi di applicazione del protocollo*

Consideriamo i seguenti esempi



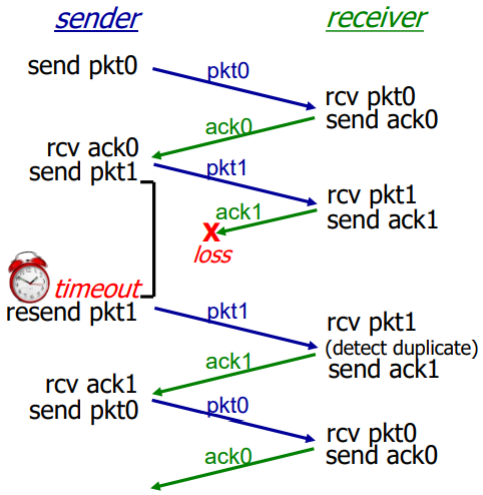
In questo esempio tutto è ok, sia nella trasmissione dei pacchetti sia nell'invio degli ACK.



In questo esempio si ha perdita del pacchetto `pkt1`.

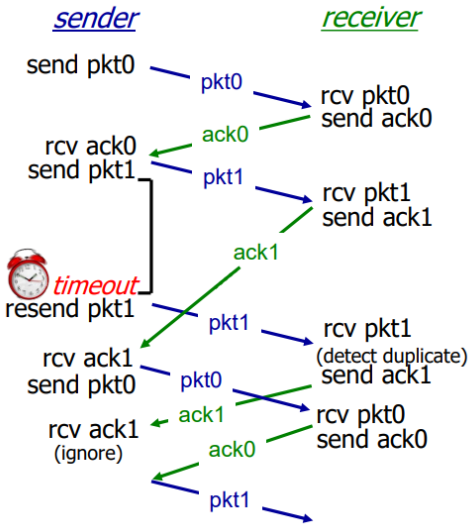
- Il receiver non lo riceve e quindi non svolge particolari azioni.
- Il sender invia nuovamente `pkt1`, superato il timeout

Quanto segue è intuitivo, dato che il receiver non aveva ricevuto niente (si riprende da dove si era rimasti a causa della perdita di `pkt1`).



In questo esempio `pkt1` viene trasmesso con successo, ma il relativo ACK no.

- Il sender non riceve l'ACK, quindi superato il timeout invia nuovamente il pacchetto.
- Il receiver riceve un pacchetto duplicato: se ne accorge dal numero di sequenza e lo scarta. Allo stesso tempo invia un ACK al sender (quello che il sender non aveva ricevuto).



Il timeout impostato è troppo breve: quello che succede è che l'ACK relativo a `pkt1` arriva a destinazione troppo tardi.

- Prima dell'arrivo di `pkt1` è scattato il timeout del sender. Questo invia nuovamente `pkt1`.
- Il receiver si accorge del duplicato dal numero di sequenza e lo scarta. Invia nuovamente l'ACK relativo al pacchetto.
- Il sender ha ricevuto l'ACK arrivato in ritardo. Dopo un po' riceve nuovamente lo stesso ACK, che viene scartato (il sender si aspettava di ricevere un ACK di tipo 0, invece ha ricevuto un ACK di tipo 1).

5.3.7 Prestazioni del protocollo RDT: passaggio da *stop and wait* a *pipelining*

Il protocollo rdt3.0 risulta sicuramente valido nel risolvere i problemi affrontati, ma ha prestazioni molto basse. Cerchiamo di capire perchè.

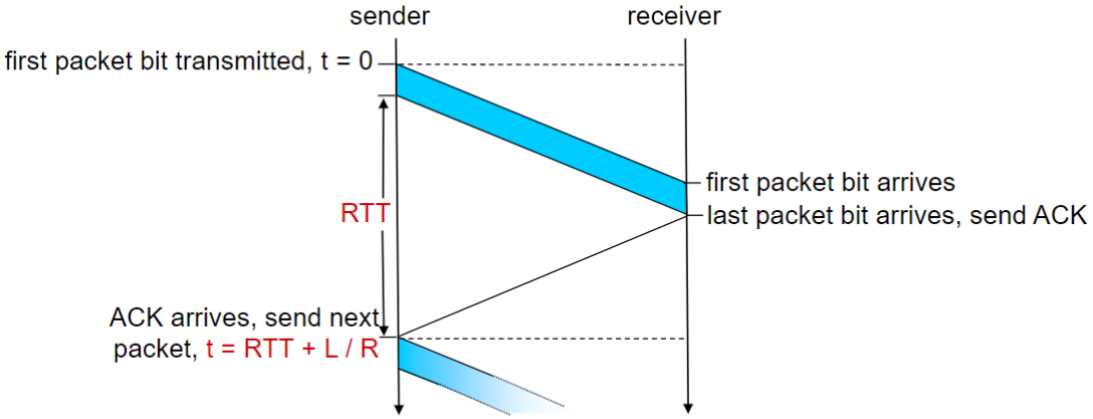
- La prima osservazione che possiamo fare è che contribuisce a questa inefficienza l'approccio *stop and wait*: il mittente non può trasmettere pacchetti fino a quando non avrà ricevuto l'ACK per il pacchetto precedente.
- Supponiamo di avere un link con capacità di 1 Gbps, il ritardo di propagazione è di 15 ms e vogliamo trasmettere pacchetti di dimensione 8000 bit. Otteniamo il ritardo di trasmissione

$$d_{trans} = \frac{L}{R} = \frac{8000\ bits}{10^9\ bps} = 8\,\text{microseconds}$$

Da cui calcoliamo l'utilizzazione

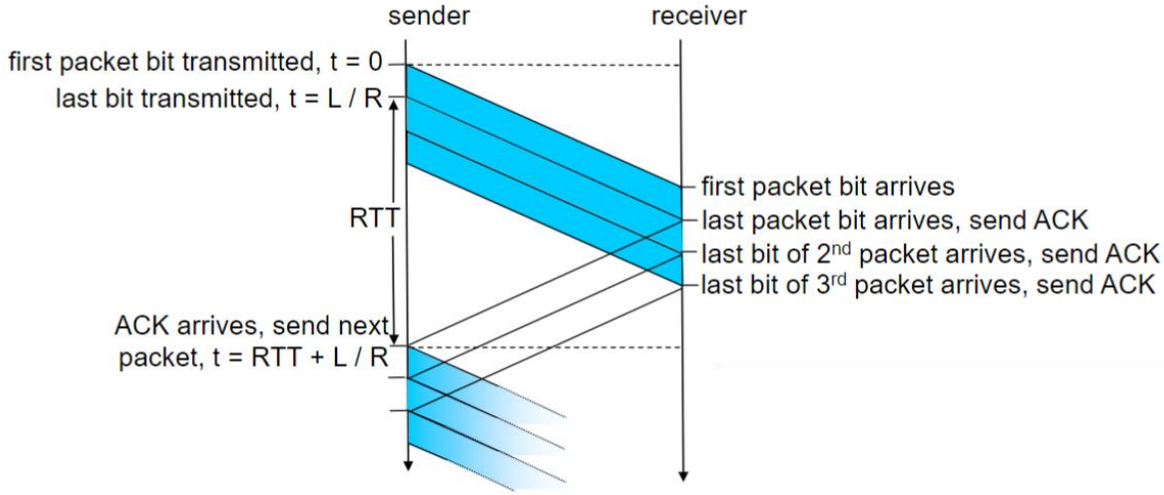
$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{0.008}{30.008} = 0.00027 = 0.027\%$$

Un valore bassissimo: è come se io trasmettessi pochissima acqua con un tubo molto grande.



Chiaramente non possiamo agire sulla distanza (e quindi ridurre RTT): è come affermare che il mio algoritmo funziona bene solo su piccole distanze (l'algoritmo deve funzionare bene con qualunque distanza!).

La soluzione consiste nel superare l'approccio stop and wait in favore di un approccio pipelining: supponiamo di voler trasmettere tre pacchetti, e di voler calcolare l'utilizzazione col nuovo approccio.



Otteniamo

$$U_{sender} = \frac{3 * L/R}{RTT + L/R} = \frac{0.0024}{30.008} = 0.00081 = 0.081\%$$

Abbiamo un miglioramento con fattore 3!

5.3.8 Protocollo Go-back-N (GBN)

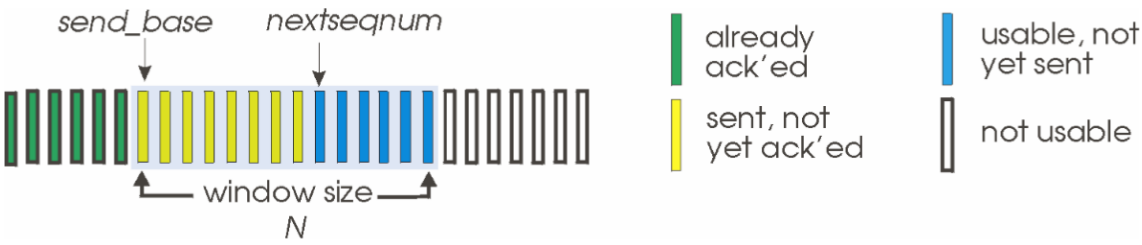
5.3.8.1 Introduzione

Il protocollo Go-back-N è uno dei due protocolli basati sull'approccio pipelining.

Definiamo numero di sequenza l'identificativo di un particolare pacchetto.

Si gestiscono più ACK in contemporanea, ma non se ne possono gestire più di N: data una sequenza di numeri di sequenza (gioco di parole) l'insieme dei numeri relativi ai pacchetti su cui stiamo lavorando è detta "finestra" – *window* – ed N è definita *ampiezza della finestra*.

Rappresentiamo la cosa graficamente e definiamo alcuni numeri di sequenza rilevanti:



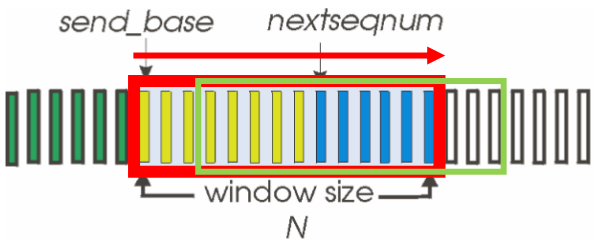
- `send_base` (più avanti definito anche come `base`) è il numero di sequenza del pacchetto più vecchio che non ha ricevuto *acknowledge*;
- `nextseqnum` consiste nel più piccolo numero di sequenza inutilizzato all'interno della finestra, cioè il numero di sequenza che assoceremo al prossimo pacchetto inviato.

Definiamo i seguenti intervalli di numeri di sequenza…

- $[0;\ send\_base\ -\ 1]$
  Numeri di sequenza relativi a pacchetti già inviati di cui il sender ha già ricevuto l'ACK. Siamo fuori dalla finestra e parliamo di pacchetti già sistemati: non ci interessano più e non abbiamo bisogno di tenerli memorizzati nel buffer.
- $[send\_base;\ nextseqnum\ -\ 1]$
  Numeri di sequenza relativi a pacchetti già inviati, ma di cui il sender non ha ancora ricevuto l'ACK. Siamo dentro la finestra.
- $[nextseqnum;\ send\_base + N - 1]$
  Numeri di sequenza non ancora utilizzati. Siamo dentro la finestra, questi numeri verranno utilizzati se ci saranno pacchetti da inviare immediatamente.
- $[send\_base + N;\ \ldots]$
  Numeri di sequenza non ancora utilizzati. Contrariamente ai precedenti siamo fuori dalla finestra, questi numeri non possono essere utilizzati finchè il sender non riceve ACK relativamente ai pacchetti dentro la finestra.

Quello che si deve immaginare è una finestra scorrevole (si parla non a caso di *sliding-window protocol*):

- ogni volta che si riceve un ACK avviene la traslazione della finestra,;
- i pacchetti che hanno ricevuto l'ACK diventano verdi ed escono dalla finestra;
- entrano nella finestra un numero di pacchetti pari al numero di pacchetti usciti (pacchetti bianchi diventano blu – passano dall'ultimo intervallo definito al penultimo).



Finestra traslata a seguito dell'acknowledge per i primi tre pacchetti nella finestra

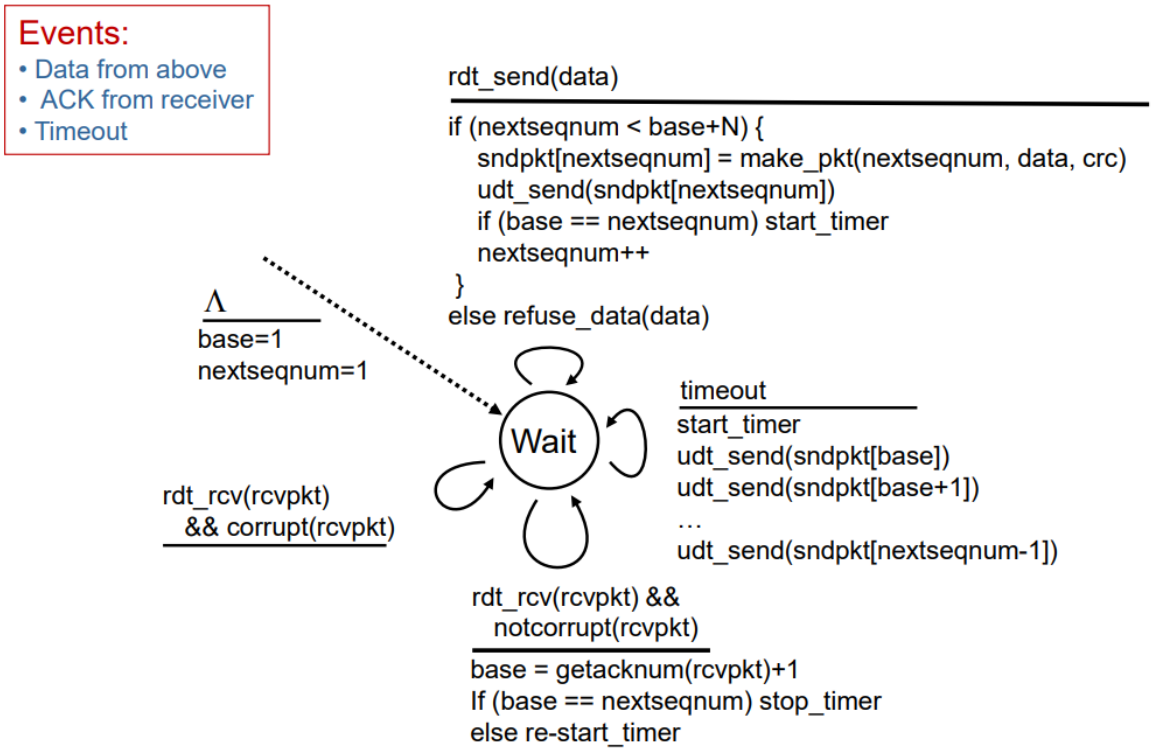
Ulteriori riflessioni…

- **Si parla di ACK cumulativo**.
  - Rappresentiamo con la notazione $ACK(n)$ l'acknowledge di tutti i pacchetti fino a quello avente numero di sequenza n.
  - Ricevere questo $ACK(n)$ significa muovere/traslare la finestra, in modo tale che send_base (cioè il primo pacchetto della finestra) sia n+1.
- **Timer**. Si mantiene un timer per il pacchetto più vecchio, cioè quello identificato dal numero di sequenza `send_base`.
  - Nel caso in cui vi sia timeout il sender ritrasmette tutti i pacchetti della finestra (tutti i pacchetti che sono già stati trasmessi e che non hanno ricevuto l'ACK – quelli gialli).
- **Perché ragioniamo nell'ottica della finestra?** Minimizzazione dell'occupazione del buffer.
  - Il sender tiene in memoria soltanto i pacchetti posti nella finestra, può cancellare i pacchetti che sono stati inviati e che hanno ricevuto ACK – quelli verdi, abbastanza ovvio)
  - Il ricevitore ha poche risorse (addirittura si potrebbe dire che non ha un buffer), si scarica la complessità a monte (al sender).

5.3.8.2 Comportamenti di sender e receiver

Avviso: nel programma dell'A.A.22-23 Anastasi ha spiegato il protocollo ignorando il formalismo.

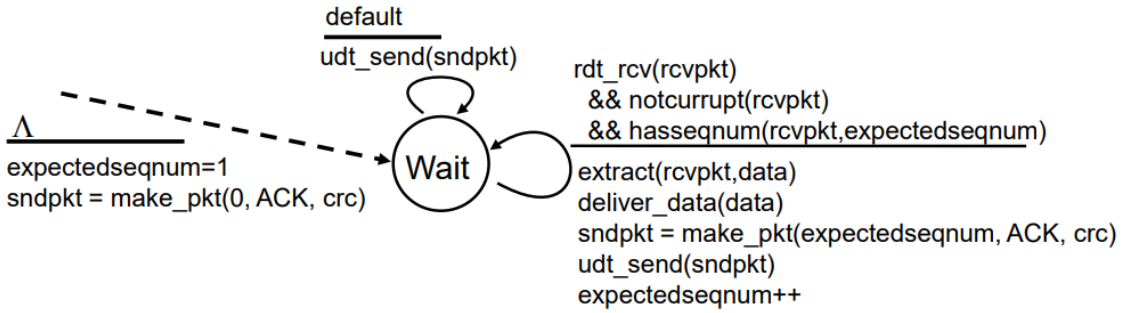
Approfondiamo recuperando il formalismo degli automi a stati finiti. Per prima cosa riflettiamo sugli eventi che si possono manifestare nel sender:



- **Inizializzazione**
  Si pone `base = 1` e `nextseqnum = 1`. Successivamente il sender si pone in attesa di eventi:
  - ricezione di dati da trasmettere dal livello superiore;
  - ricezione di ACK dal receiver;
  - timeout.
- `rdt_send(data)`
  Si verifica se il prossimo numero di sequenza è interno alla finestra, cioè appartiene all'intervallo $[base;\ base\ +\ N\ -1]$: se ciò non è vero il pacchetto non viene inviato e lo si segnala al livello superiore con `refuse_data`. Se siamo sempre dentro la finestra procediamo come segue:
  - costituiamo il pacchetto per il livello inferiore e lo inviamo con la `udt_send`;
  - verifichiamo se il pacchetto appena inviato consiste nel primo pacchetto della finestra (`base == nextseqnum`), in quel caso avviamo il relativo timer

    - incrementiamo `nextseqnum`, dato che abbiamo trasmesso un nuovo pacchetto
- `rdt_rcv(rcvpkt) && corrupt(rcvpkt)`
  Se il pacchetto ricevuto è corrotto non si fa niente.
- `rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)`
  Il pacchetto ricevuto è un ACK.
    - Trasliamo la finestra: si incrementa il numero n associato all'ACK, questo ultimo recuperato con la funzione `getacknum`. Otteniamo così il nuovo valore di `base`.
    - Se non c'è più nessun pacchetto giallo in attesa (`base == nextseqnum`) il timer viene bloccato. Sarà riavviato da zero col manifestarsi dell'evento `rdt_send(data)`.
    - Se `base == nextseqnum` è falso allora significa che ci sono ancora pacchetti gialli da considerare. Riavviamo da zero immediatamente il timer, che adesso sarà associato al pacchetto identificato dal nuovo `base`.
- `timeout`
  In caso di timeout facciamo ripartire il timer da zero (start_timer) e inviamo nuovamente tutti i pacchetti della finestra che avevamo già inviato (quelli gialli).

Per quanto riguarda il receiver:



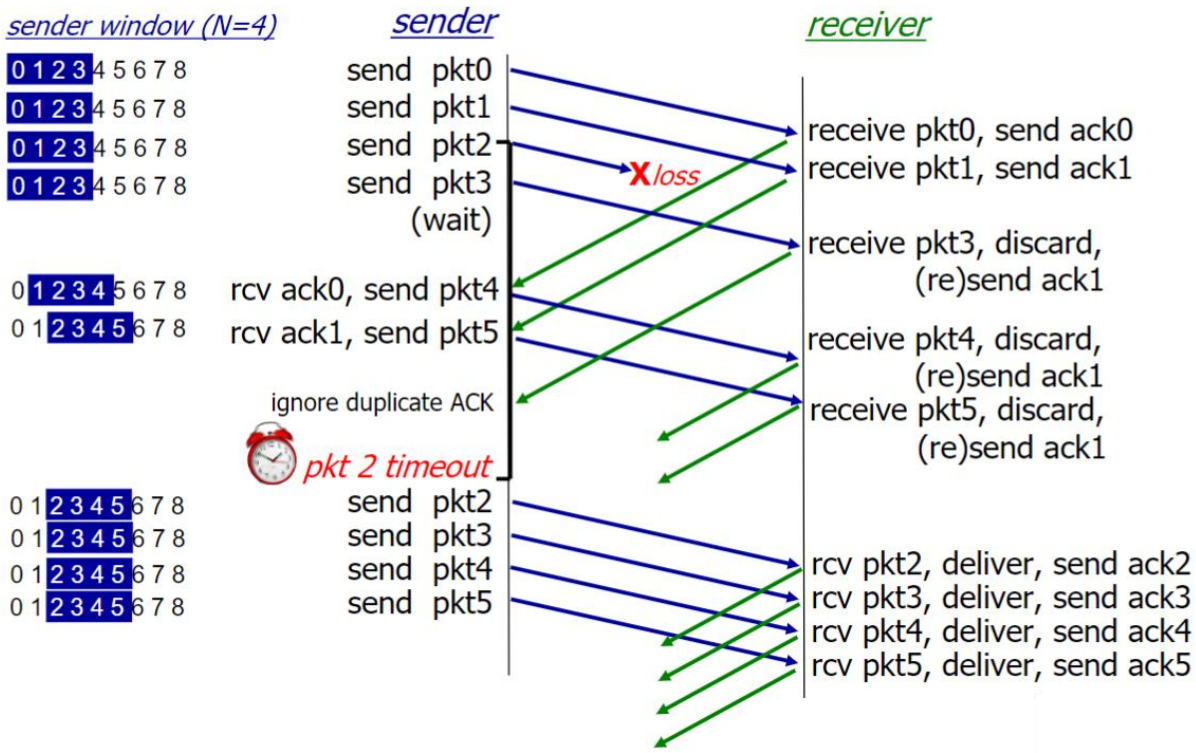
- **Inizializzazione**
  Si pone `expectedseqnum = 1`, in modo tale che sia sincronizzato col sender. Il sender invia un pacchetto al receiver per sincronizzare il valore, il receiver risponde con un ACK.
- `rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && hasseqnum(rcvpkt, expectedseqnum)`
  Il pacchetto ricevuto non è corrotto e il numero di sequenza è quello che ci aspettiamo. Si osservi che l'evento esclude i pacchetti fuori sequenza.
    - Estraggo i dati dal pacchetto con `extract` e li invio al livello superiore con `deliver_data`
    - Invio un pacchetto di ACK con `udt_send` al sender.
    - Incremento `expectedseqnum`.
- `default`.
  In tutti gli altri casi il receiver invia al sender l'ultimo acknowledge inviato (per esempio pacchetti corrotti, oppure pacchetti non corrotti fuori sequenza).

5.3.8.3 Vantaggi e svantaggi

- **Vantaggi**.
  Sicuramente si semplifica il lavoro al receiver, scaricando la complessità a monte del server. La cosa è utile quando sappiamo che il receiver non ha molte risorse. Addirittura non c'è bisogno di gestire un buffer. Permette di limitare il numero di pacchetti memorizzati nel buffer del sender.
- **Svantaggi**.
  Il protocollo è poco intuitivo, molto spesso richiede trasmissioni non necessarie di pacchetti (si pensi ai pacchetti gialli inviati nuovamente). Consumo di banda e di energia.

5.3.8.4 Esempio di esecuzione

Consideriamo il seguente esempio di interlocuzione tra sender e receiver, con a lato la finestra che trasla

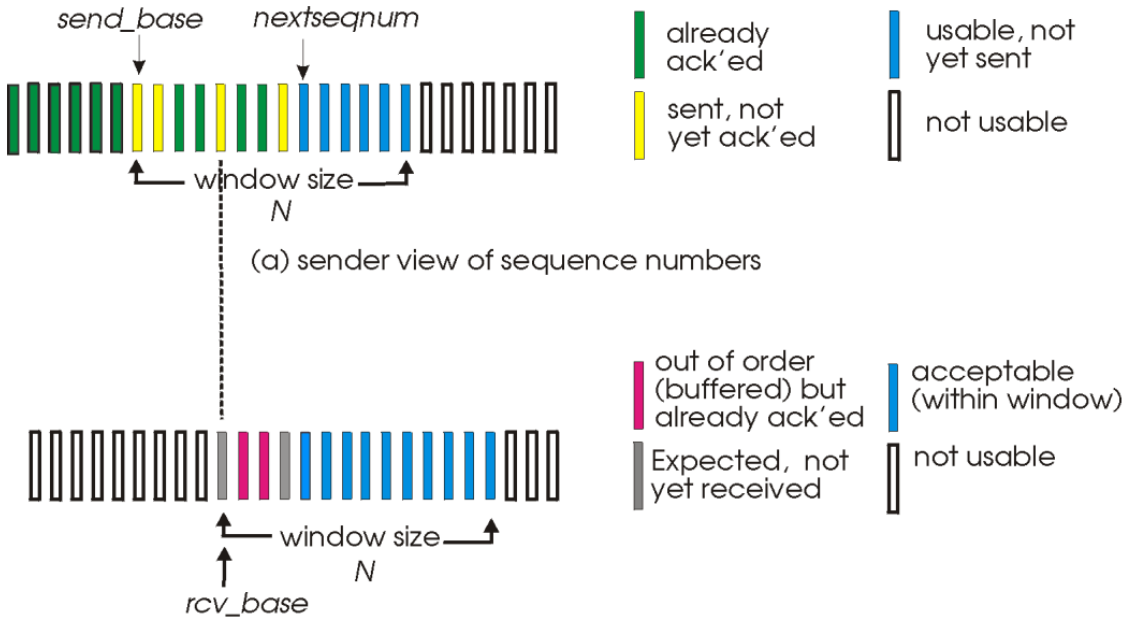


5.3.9 Protocollo Selective Repeat

5.3.9.1 Introduzione

Il protocollo selective repeat è un altro protocollo basato su approccio pipelining. Come prima si gestiscono al più N pacchetti, ma in questo caso l'ACK è individuale e non cumulativo. In caso di problemi il sender trasmette nuovamente solo i pacchetti che individualmente non hanno ricevuto l'acknowledge.

- Non si buttano più via i pacchetti fuori sequenza: vengono messi in un buffer (ecco che torna il buffer nel receiver, contrariamente a go-back-N) e introdotti al momento opportuno.
- Non si ha più un solo timer, ma tanti timer quanti i pacchetti inviati.
- Si mantiene la finestra nel receiver (approccio simile a quello visto in go-back-N), mentre si assumono comportamenti diversi nel receiver.



(a) sender view of sequence numbers

(b) receiver view of sequence numbers

5.3.9.2 *Comportamenti di sender e receiver*

Per approfondire il comportamento del selective repeat si ricorre allo pseudocodice:

sender

data from above:
- if next available seq # in window, send packet

timeout(*n*):
- resend packet *n*, restart timer

ACK(*n*) in [sendbase,sendbase+N]:
- mark packet *n* as received
- if n smallest unACKed packet, advance window base to next unACKed seq #

receiver

packet *n* in [rcvbase, rcvbase+N-1]
- send ACK(*n*)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order packets), advance window to next not-yet-received packet

packet *n* in [rcvbase-N,rcvbase-1]
- ACK(*n*)

otherwise:
- ignore

Sender

Gli eventi sono gli stessi analizzati in go-back-N, tuttavia:

- Il timeout è relativo a un singolo pacchetto, non si ACK cumulativo;
- Nel caso di ACK(n) relativo a un pacchetto appartenente alla finestra si segna il pacchetto come ricevuto, e si trasla la finestra se il pacchetto con numero di sequenza più basso riceve l'acknowledge.

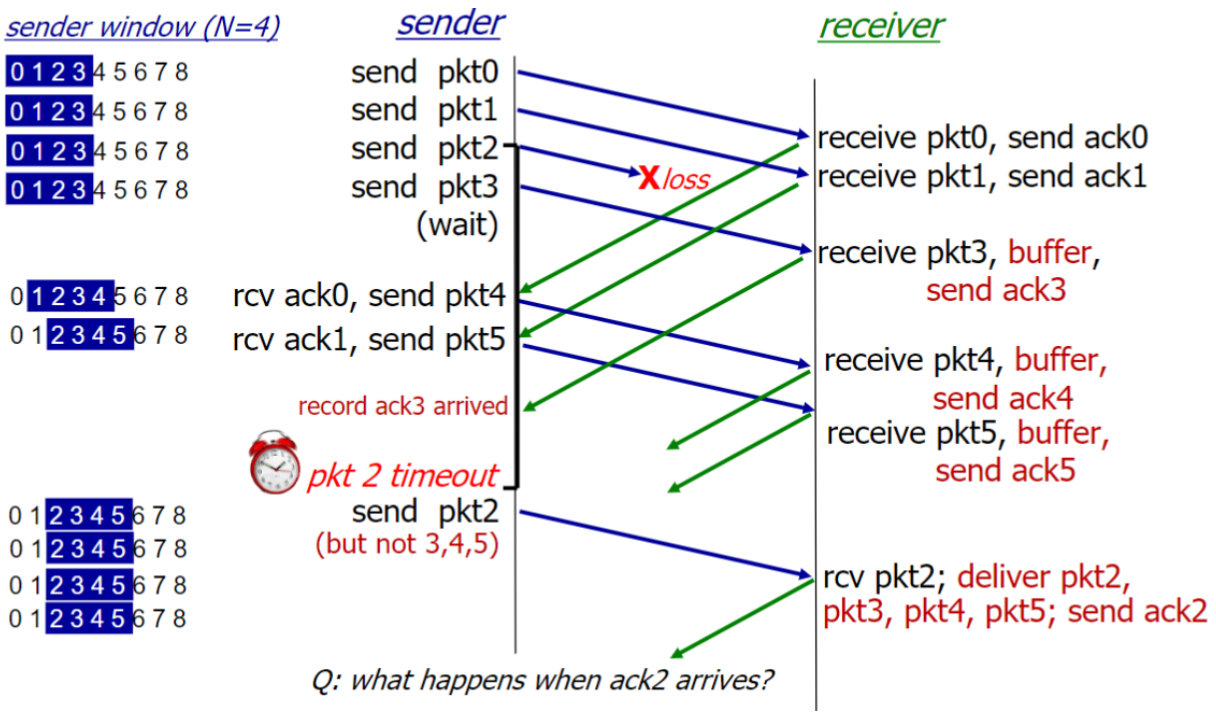
Receiver

Quando riceve il pacchetto invia l'ACK per il singolo pacchetto. Osserva il numero di sequenza:

- Se riceve un pacchetto fuori sequenza lo memorizza in un buffer, reintroducendolo al momento opportuno (lo riporta in sequenza)
- Se riceve un pacchetto in sequenza lo consegna al livello superiore.

5.3.9.3 *Esempio di esecuzione*

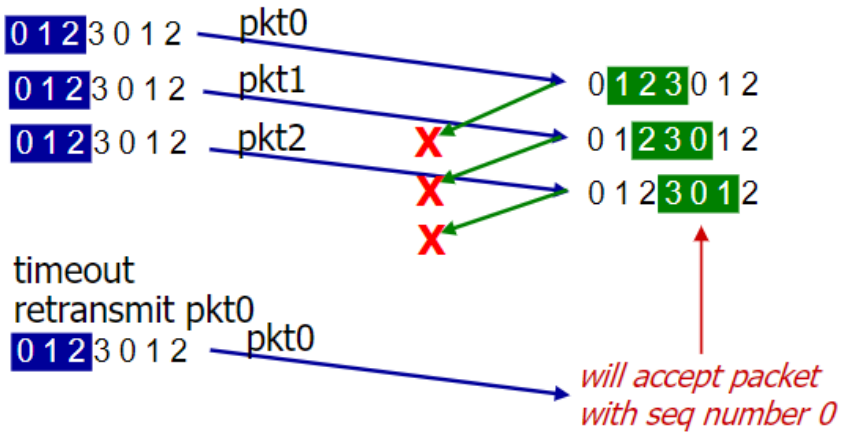
Consideriamo il seguente esempio di interlocuzione tra sender e receiver



5.3.9.4 Dilemma di Di Nucci

Il titolo fa riferimento a uno studente che è stato tirato in ballo dal docente durante la lezione, ovviamente il dilemma non va chiamato così all'esame

Il protocollo risulta avere un problemino in un particolare caso. Si consideri la seguente situazione, dove la finestra del receiver viene traslata a seguito dell'invio di ACK (ricordarsi la circolarità dei numeri di sequenza). Il problema fondamentale è che gli ACK non vengono ricevuti dal sender, ergo la finestra del sender non trasla!



Quello che succede è che i pacchetti precedentemente trasmessi saranno ritrasmessi come duplicati. La cosa si risolve facendo in modo che la grandezza della finestra sia inferiore rispetto alla grandezza della sequenza.

5.4 Point to Point Protocols (PPP)

5.4.1 Caratteristiche dei protocolli PPP

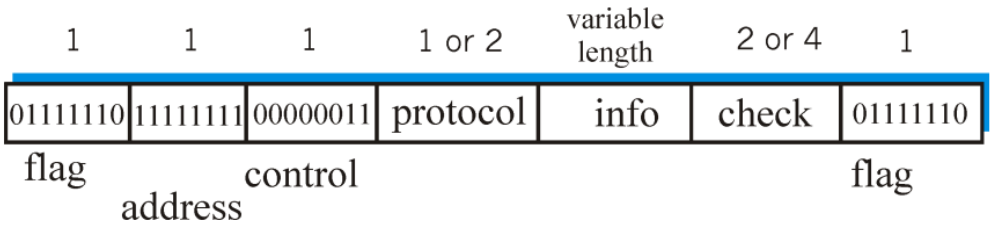
Introduciamo i protocolli Point to Point (PPP). Essi prevedono le seguenti caratteristiche…

- ***Packet framing***
  Divisione del pacchetto in frame, di cui si definisce il formato. Ciascun frame verrà poi gestito singolarmente (nel senso che error detection si fa sui singoli frame).
- ***Bit transparency***
  Deve poter essere inviabile qualunque sequenza di bit (si veda più avanti il problema risolto coi caratteri di escape).
- ***Error detection***
  Prevista le error detection, ma non si compie error correction (non si parla neanche di Reliable Data Transfer Protocol, con gli ACK). Non si prevede error recovery: i pacchetti con errori si buttano via. Possibile un invio fuori ordine dei pacchetti (dovuto ad esempio al buttare via pacchetti con errori).
- ***Network layer address negotiation***
  I nodi possono negoziare alcuni parametri (sia di livello data link che di livello network), ad esempio l'indirizzo di livello network.

Non è supportata la *comunicazione point-to-multi-point*.

5.4.2 Formato del frame

Il frame prevede la seguente struttura



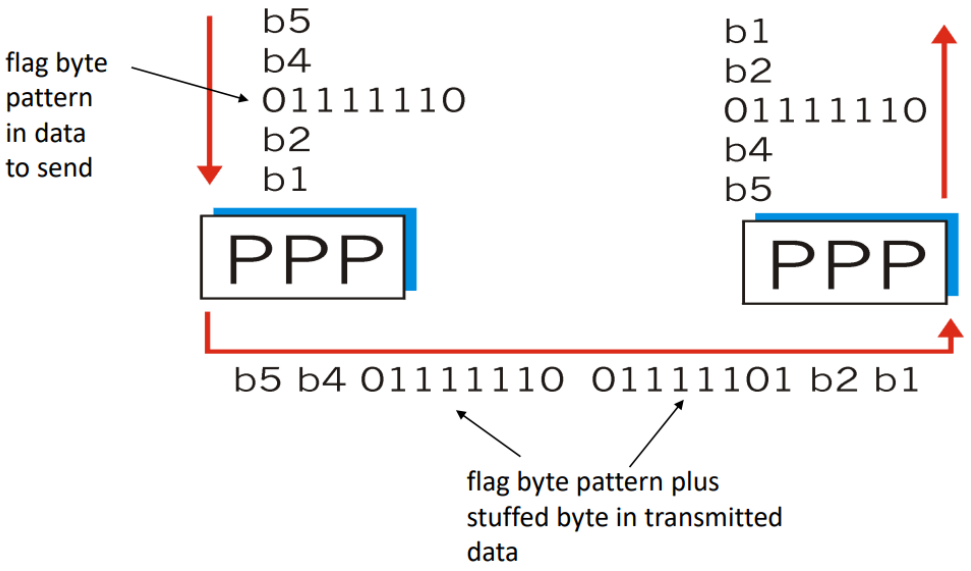
- Flag iniziale con cui il trasmettitore avverte il ricevitore che sta trasmettendo un frame. Ha una configurazione particolare: 01111110
- Campo address costituito da soli 1. Ma a cosa serve questo campo se parliamo di un protocollo point-to-point (il destinatario è lo stesso)? In passato alcuni protocolli supportavano la comunicazione point-to-multipoint (mainframe).
- Campo control, anch'esso con configurazione particolare. È un campo particolare lasciato per l'introduzione di future funzionalità.
- Campo protocol, al più due byte. Indica il protocollo adottato (all'epoca esistevano più protocolli)
- Campo info / payload: lunghezza variabile (di default vale 1500 byte, le due parti possono negoziare e ridurre la dimensione del campo), contiene il contenuto del frame.
- Campo check per controllo CRC (Cycle Redundancy Check). Posti in fondo (nel trailer) visto che vengono calcolati durante la trasmissione (*on the fly*).
- Flag finale, uguale al flag iniziale.

5.4.3 byte stuffing per l'attuazione della bit transparency

Come posso trasmettere byte di flag come contenuto? Si svolge l'operazione di byte stuffing:

- il trasmettitore PPP si accorge della sequenza e introduce un byte di escape prima del byte di flag;
- il ricevitore PPP rimuove il byte di escape (*byte unstuffing*).

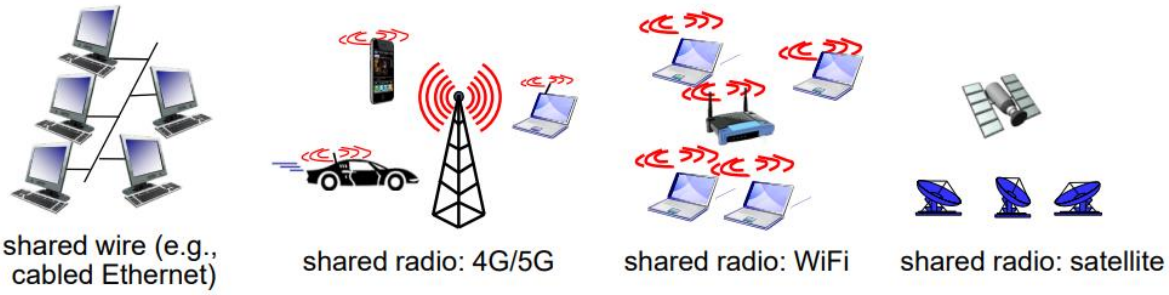
- Sender (byte stuffing)
  - 01111110 → 01111101 01111110
  - 01111101 → 01111101 01111101
- Receiver (byte unstuffing)
  - 01111101 01111110 → 01111110
  - 01111101 01111101 → 01111101



E se volessi trasmettere una sequenza uguale al byte di escape? Introduco un altro byte di escape (per dirla con un esempio personale e non citato dal prof, il doppio backslash per andare a capo in LaTeX).

5.5 Passaggio da reti punto punto a reti ad accesso multiplo

Fino ad ora abbiamo parlato di un link fisico punto – punto. Cosa succede se vogliamo collegare più dispositivi insieme? Mantenere l'approccio punto-punto introdotto nel capitolo precedente significherebbe immaginare per ogni host, dati N host, $N-1$ collegamenti verso gli altri host della rete: l'approccio è molto costoso economicamente parlando (non si può pensare a una cosa del genere con N molto elevato) e non si ha scalabilità. Dobbiamo andare oltre.



L'idea è di introdurre dei link di comunicazione *broadcast*, cioè dei link condivisi a cui viene collegato un numero limitato di computer.

- Se un host vuole rivolgere un pacchetto a più dispositivi lo pone sul link broadcast.
- Tutti i dispositivi collegati dal link broadcast riceveranno il pacchetto.
- Approccio tipico delle versioni più vecchie di Ethernet, anche delle reti satellitari e Wifi.

Chiaramente in un link condiviso è necessario stabilire delle regole, visto la presenza di più interlocutori: a tal proposito introduciamo i Multiple access protocols (protocolli per reti ad accesso multiplo).

5.6 Multiple Access Protocols

5.6.1 Cosa vogliamo fare

In un protocollo per reti ad accesso multiplo si deve:

- determinare quando un nodo può trasmettere sul canale;
- ipotizzare la presenza di un unico canale di comunicazione (quindi non vi è un ulteriore canale per coordinare la comunicazione).

Spieghiamo meglio l'ultimo punto, prendendo a paragone l'interlocuzione col docente. Nell'aula dove avviene la lezione del prof. Anastasi si hanno più interlocutori.

- Due canali: il canale uditivo e il canale visivo.
- Il canale uditivo è quello attraverso cui ascoltiamo la spiegazione del docente e ciò che ci dicono i colleghi.
- Il canale visivo è quello attraverso cui chiediamo di poter parlare, con alzata di mano. Il docente vede la mano alzata, smette di parlare e concede la parola allo studente.

I protocolli che andremo a descrivere, se pensiamo al contesto dell'aula, pongono come ipotesi l'assenza del canale visivo: si ha solo il canale uditivo, e attraverso questo dobbiamo sia trasmettere i pacchetti, sia gestire le collisioni.

Con collisione si intende la ricezione da parte di un nodo di due o più segnali allo stesso tempo.

5.6.2 Caratteristiche di un protocollo ideale

Prendiamo un canale dove la capacità della banda (rate) è R bps. Le caratteristiche ci piacerebbe avere nel protocollo sono le seguenti.

- **Piena utilizzazione**.
  Quando un solo nodo trasmette lo deve poter fare occupando l'intera banda (rate R).
- **Fairness**.
  Quando più nodi vogliono trasmettere lo devono poter fare con un rate R/M (si divide la banda in parti uguali tra gli M nodi che vogliono trasmettere – stesse opportunità per ogni nodo).
- **Protocollo completamente decentralizzato**.
  Il protocollo deve essere completamente decentralizzato: non si pone una gerarchia tra i nodi (dando a un particolare nodo il compito di coordinare le trasmissioni di tutti i nodi presenti nella rete), non si ha la sincronizzazione del clock (cioè i nodi non devono essere sincronizzati dal punto di vista del tempo, si pensi agli slot che vedremo più avanti).
- **Semplicità**.
  Il protocollo deve essere semplice (regola americana del KIS, *Keep It Simple*, o *Keep It Stupid*). Su questo supporremo che i protocolli affrontati siano semplici: la semplicità è un concetto soggettivo, è un po' come cercare di definire la bellezza (dipende dal soggetto).

Per ogni protocollo da noi affrontato verificheremo se le caratteristiche dette sono presenti.

5.6.3 Classificazione dei protocolli

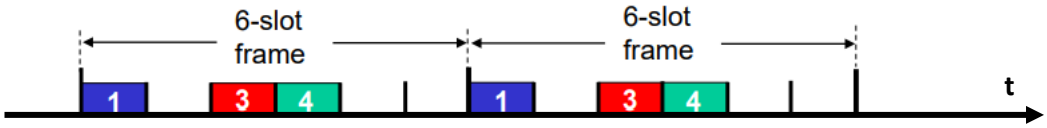
Si individuano i seguenti tipi di protocolli.

- **Protocolli a partizionamento di canale**.
  Il mezzo condiviso "viene usato in condivisione". Si divide il canale nel dominio del tempo o nel dominio della frequenza, ad ogni nodo si alloca un frammento del canale. Ogni frammento è per uso esclusivo.
- **Protocolli ad accesso casuale**.
  Il canale non è diviso come prima: si permettono le collisioni e si introducono delle procedure di recupero da applicare in caso di collisioni.
- **Soluzioni ibride** (*taking turns*).

5.6.4 Protocolli a partizionamento di canale

5.6.4.1 TDMA: time division multiple access

Il TDMA prevede la divisione del canale nel dominio del tempo.



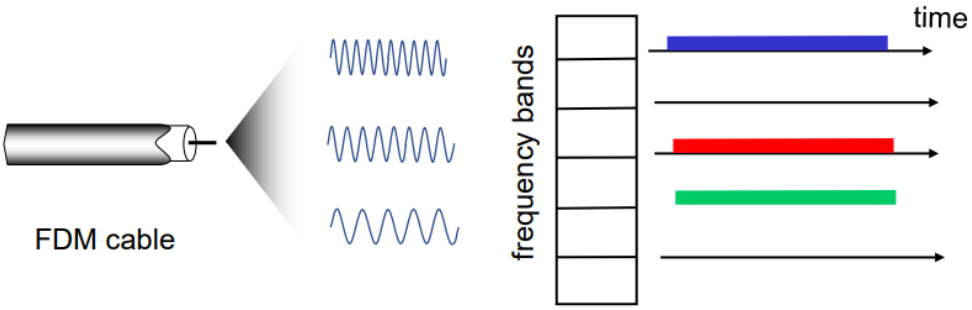
Dividiamo l'asse del tempo in frame, il cui contenuto si ripete periodicamente. Il frame è diviso a sua volta in slots, ciascuno dei quali è assegnato a un particolare nodo. È possibile che uno slot non venga assegnato (come in figura).

Riflettiamo sulle caratteristiche:

- **Piena utilizzazione?**
  Un solo nodo non occupa tutta la banda, ma solo lo slot che gli viene assegnato.
- **Fairness?**
  In presenza di più nodi quanto posto nelle caratteristiche ideali è valido solo se abbiamo un numero di slot pari al numero di nodi M nella rete.
- **Protocollo completamente decentralizzato?**
  Tolto un nodo coordinatore è necessario che vi sia una sincronizzazione del clock (ogni nodo deve sapere quando poter iniziare a trasmettere, altrimenti collide con altri nodi).

5.6.4.2 FDMA: frequency division multiple access

Lo spettro del link è diviso nel dominio della frequenza, in bande di frequenza. Si prenda l'esempio delle stazioni radiofoniche: si assegna una banda di frequenza e l'ascoltatore si sintonizza su quella frequenza per ascoltare l'emittente.



5.6.5 Protocolli ad accesso casuale

Il canale non è diviso come prima: un nodo che vuole trasmettere lo fa con rate R, senza coordinarsi con gli altri nodi. Protocolli buoni con poco traffico.

Conseguenza: sono possibili collisioni! Il protocollo deve prevedere:

- individuazione delle collisioni;
- recupero dalle collisioni.

5.6.5.1 Slotted ALOHA

Slotted ALOHA è la versione aggiornata di Pure ALOHA: la trattiamo prima per comodità. Assumiamo che:

- i pacchetti sono di dimensione fissa e il tempo è diviso in slot di durata opportuna (trasmissione del pacchetto e ricezione dell'ACK);
- il nodo non può trasmettere quando gli pare, deve attendere l'inizio dello slot successivo;
- i nodi sono sincronizzati, cioè hanno un clock comune e sanno insieme quando inizia lo slot successivo;
- se due o più nodi trasmettono insieme si ha una collisione, **e tutti i nodi coinvolti se ne accorgono**.

Come si comportano i nodi in caso di collisione, o in generale quando il pacchetto viene trasmesso male?

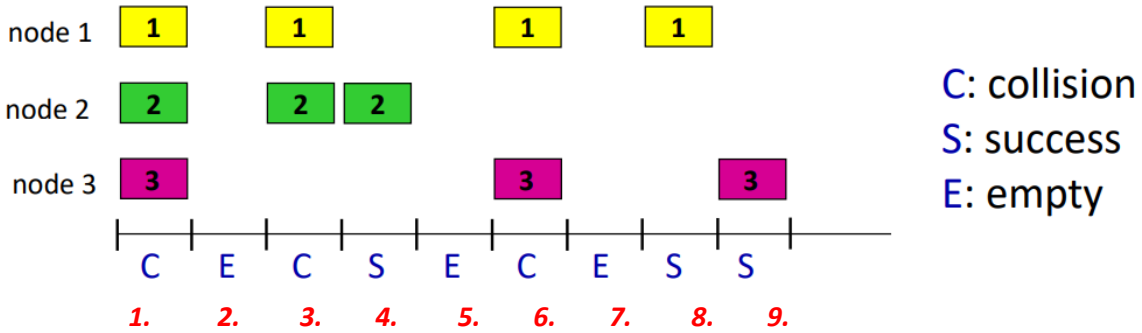
- Il receiver invia l'ACK se tutto va bene

- Non invia niente se qualcosa è andato storto, il sender non riceve nulla e superato lo slot temporale comprende che qualcosa è andato storto. In caso di errore di trasmissione diverso dalla collisione potrebbe convenire inviare subito il pacchetto, ma noi non abbiamo modo di distinguere gli errori.

Quindi:

- I nodi si accorgono che c'è stata collisione.
- Ogni nodo "lancia una monetina" per decidere se trasmettere nuovamente oppure no.
- Si continua così fino a quando non saranno trasmessi tutti i pacchetti.ù

Si prenda come esempio quanto posto in figura, dove abbiamo tre nodi che trasmettono (node 1, 2 e 3)



1. I nodi non si coordinano e nel primo slot trasmettono tutti e tre un pacchetto. Viene segnalata la collisione e nessuno ha trasmesso con successo il pacchetto.
2. Tutti i nodi "lanciano la monetina" e nessuno invia un pacchetto.
3. Tutti i nodi "lanciano la monetina": node 1 e node 2 trasmettono, node 3 no. Si ha collisione, dunque nessuno ha trasmesso con successo il pacchetto.
4. Tutti i nodi "lanciano la monetina": trasmesse solo node 2, non essendoci conflitti la trasmissione del pacchetto ha successo.
5. Node 1 e node 3 "lanciano la monetina": nessuno trasmesse pacchetti.
6. Node 1 e node 3 "lanciano la monetina": entrambi inviano un pacchetto. Si ha collisione, dunque nessuno ha trasmesso con successo il pacchetto.
7. Node 1 e node 3 "lanciano la monetina": nessuno trasmesse pacchetti.
8. Node 1 e node 3 "lanciano la monetina": trasmesse solo node 1, non essendoci conflitti la trasmissione del pacchetto ha successo.
9. Node 3 "lancia la monetina": trasmette solo lui, non essendoci conflitti la trasmissione del pacchetto ha successo.

Si sprecano slot, per trasmettere tre pacchetti si impiegano nove slot (utilizzazione di 1/3). Il protocollo è fair sul lungo tempo, e non è decentralizzato (sincronizzazione del clock, visto che dividiamo la linea temporale in slot). Determiniamo analiticamente l'utilizzazione, da un punto di vista probabilistico. Supponiamo di avere N nodi, ciascuno ha un pacchetto pronto per la trasmissione (idealmente può trasmettere in ogni slot).

- Probabilità che un particolare nodo trasmetta in uno slot con successo: p
  Probabilità che un particolare nodo non trasmetta: $1-p$
- Probabilità che un particolare nodo trasmetta e gli altri no: $\mathrm{p}(1-\mathrm{p})^{\mathrm{N}-1}$
  (eventi indipendenti tra loro, si ottiene la probabilità detta moltiplicando per tutte le probabilità)
- Probabilità che un nodo qualsiasi abbia successo: $\mathrm{Np}(1-\mathrm{p})^{\mathrm{N}-1}$
  (si somma N volte la probabilità che un particolare nodo trasmetta con successo e gli altri no).
- Quello che noi vogliamo fare è trovare p^* tale che $\mathrm{Np}(1-\mathrm{p})^{\mathrm{N}-1}$ sia massimizzato.
  Lo si fa uguagliando a zero la derivata prima della funzione.
- Si calcola il limite di $Np^*(1-p^*)^{N-1}$ con $\mathrm{N}\to\infty$: otteniamo

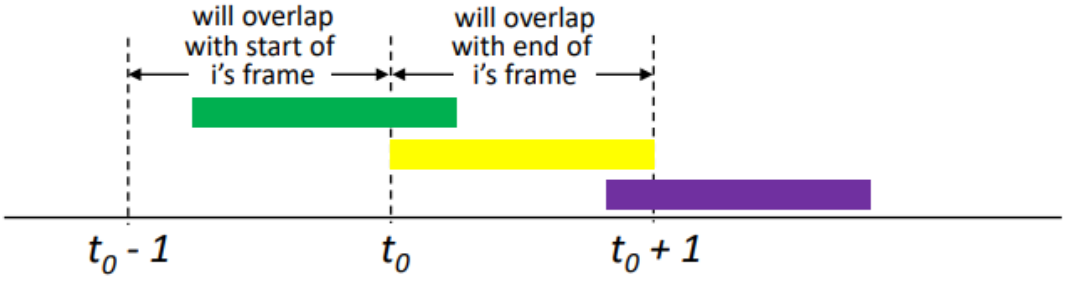
$$\lim_{N\to\infty} Np^*(1-p^*)^{N-1} = \frac{1}{\mathrm{e}} = 0.37$$

  cioè il 37% di utilizzo.

Non siamo contenti di quanto ottenuto! Si consideri che in pure ALOHA abbiamo un'utilizzazione peggiore.

5.6.5.2 Pure ALOHA

Pure ALOHA è la versione originale di ALOHA, che contrariamente alla versione slotted non prevede una sincronizzazione del clock: non si ha una divisione temporale, la linea del tempo è continua e l'utente inizia a trasmettere liberamente.



Il pacchetto giallo viene trasmesso all'istante temporale t_0: si ha sovrapposizione se altri pacchetti vengono trasmessi nell'intervallo di tempo $[t_0 - 1; t_0 + 1]$. Possibile sovrapposizione parziale e totale dei pacchetti. Calcoliamo la probabilità che un nodo trasmetta con successo un pacchetto, moltiplicando le seguenti probabilità:

- la probabilità che un particolare nodo trasmetta un pacchetto
- la probabilità che nessun altro nodo trasmetta pacchetti nell'intervallo temporale $[t_0 - 1; t_0]$
- la probabilità che nessun altro nodo trasmetta pacchetti nell'intervallo temporale $[t_0; t_0 + 1]$

Otteniamo

$$P = p \cdot (1-p)^{N-1} \cdot (1-p)^{N-1} = p \cdot (1-p)^{2(N-1)}$$

Come prima troviamo p^* tale da massimizzare la probabilità e infine calcoliamo il limite con $N \to \infty$

$$\lim_{N\to\infty} p \cdot (1-p)^{2(N-1)} = \frac{1}{2e} = 0.18$$

Cioè il 18% di utilizzo (dato peggiore)!

Conclusione: si è deciso di introdurre la sincronizzazione in slotted ALOHA per migliorare l'efficienza. Ovviamente in slotted ALOHA si ha un overhead maggiore, necessario per la sincronizzazione del clock.

5.6.5.3 CSMA (Carrier sense multiple access) e CSMA/CD (Collision Detection)

Il protocollo CSMA (*Carrier Sense Multiple Access*) presenta un approccio diverso rispetto ad ALOHA:

- ALOHA prevede una retransmission a seguito di collisioni (i nodi trasmettono pacchetti senza ascoltare);
- CSMA prevede un iniziale ascolto del canale e trasmissione solo se il canale viene percepito come libero (se viene percepito come occupato allora il nodo rimanda la trasmissione del pacchetto).

Il protocollo è stato inizialmente progettato per la Ethernet (si consideri che è cambiato profondamente nel tempo), ma successivamente pensato anche per le reti Wireless (dove però non si ha la *collision detection*, ma una *collision avoidance*). Entrambe le varianti di CSMA sono affrontate nei relativi capitoli.

A questo punto ci chiediamo: abbiamo eliminato il problema delle collisioni? Purtroppo no, il ritardo di propagazione dei pacchetti potrebbe provocare una collisione perché i nodi non si sentono a causa del ritardo: tutto questo nonostante la corretta applicazione del protocollo CSMA.

Si cerca di risolvere introducendo la variante CSMA/CD, che prevede *collision detection*.

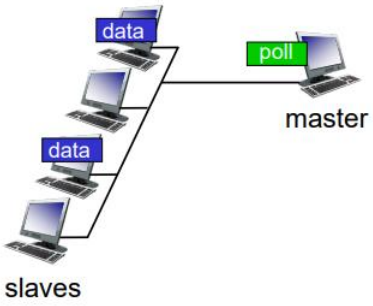
- **Come è possibile individuare una collisione?**
  Chi trasmette analizza la potenza ricevuta sul canale. Conosce la potenza trasmessa e quindi la può confrontare: se la potenza ricevuta è *significativamente* maggiore di quella trasmessa allora siamo in presenza di una collisione.
- **Cosa si fa in caso di collisione?**
  Idealmente si abortisce. Nei fatti entrambi continuano a trasmettere: trasmettono, precisamente, un segnale JAM di potenza maggiore. Si rafforza la collisione in modo tale che venga sentita da tutti i nodi.

5.6.6 Soluzioni ibride

Come al solito la soluzione migliore è un ibrido tra le tipologie di soluzioni precedenti.

5.6.6.1 polling

Dati M nodi il polling prevede un master ed $M - 1$ slave (detto in modo non politicamente corretto).



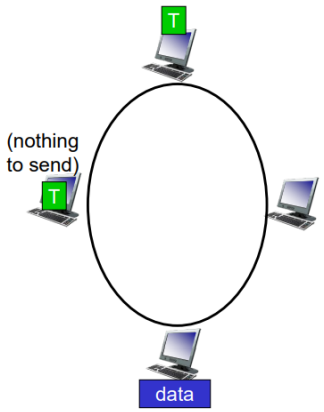
- Il master interroga tutti gli slave, a turno.
- Ad ogni slave chiede se ha pacchetti da trasmettere.
- Lo slave trasmette pacchetti se ne ha.

Cosa possiamo dire?

- Non si utilizza la rete al 100% perché ci sono i pacchetti di polling, non si ha una divisione uguale in presenza di M nodi: questo perché si ha overhead con l'introduzione dei pacchetti di interrogazione.
- È un protocollo fair? Decide il master, dipende da lui.
- Non è decentralizzato: c'è il master. Si ha single point of failure: se non c'è il master il link non funziona. *Morto un re se ne fa un altro* (cit.)

5.6.6.2 token passing

Dati M nodi il token passing prevede il passaggio di un token, un pacchetto speciale, tra un nodo e un altro in modo sequenziale. Un nodo che vuole trasmettere un pacchetto attende il passaggio del token. Non può trasmettere se non quando riceverà quel pacchetto: il fatto che il pacchetto sia "in mano sua" e non di altri nodi garantisce che il mezzo sia libero.



- Anche qua non si ha utilizzo al 100% per i pacchetti token, e non si ha divisione uguale in presenza di M nodi: overhead dovuto alla trasmissione del pacchetto.
- È un protocollo fair? Se tutti trasmettono un solo pacchetto sì.
- Decentralizzato? No, se il token non c'è la rete non funziona: anche qua problema del single point of failure.

5.7 Local Area Network (LAN)

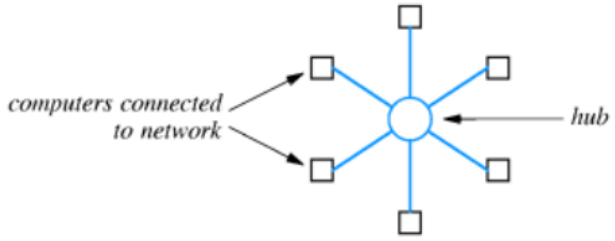
5.7.1 Definizione di LAN

Una rete locale è una rete che connette un numero limitato di dispositivi per mezzo di un link condiviso di tipo broadcast.

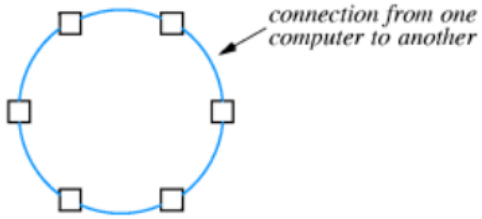
- Hanno limitata estensione geografica (un edificio, ma anche un insieme di edifici)
- La velocità è alta: si va dai 100 Mbps ai 10 Gbps.

5.7.2 Topologia delle LAN

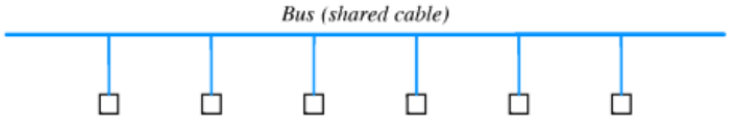
I nodi sono collegati al mezzo fisico comune secondo particolari configurazioni. Vediamo i seguenti esempi



A stella, presente un dispositivo centrale (hub)



Ad anello



A bus, attenzione al segnale che si riflette e torna indietro quando si arriva agli estremi del cavo.
Si ha collisione tra segnali. Si introducono delle resistenze che impediscono la riflessione del segnale.

5.7.3 Comunicazione unicast (sender - receiver) con mezzo di tipo broadcast

Abbiamo introdotto l'idea del mezzo fisico condiviso, quindi tutto ciò che viene posto sul mezzo viene ricevuto da tutti. Nella maggior parte dei casi la comunicazione non è 1 a tutti, ma 1 a 1: è possibile fare una comunicazione sender - receiver con un mezzo di tipo broadcast? Detto in altro modo: come posso scambiare messaggi con un solo interlocutore, senza essere sentito da altri, nonostante il mezzo sia condiviso?

- Necessario uno schema di indirizzamento che permetta identificazione univoca nella LAN.
- Ogni pacchetto posto sul link broadcast presenterà un indirizzo destinatario. La scheda di rete, ricevuto il pacchetto, verifica se il pacchetto è rivolto a lui. Se non lo è ignora.
- **Necessario "educare" la scheda di rete nella componente hardware**.

Attenzione: la sicurezza non è garantita. La scheda di rete può essere posta in modalità "promiscua": in questo caso i pacchetti sul link condiviso vengono tutti memorizzati (può essere utile all'amministratore di rete per monitorare il traffico, ma è soprattutto utile a chi vuole sniffare[9]).

E nel caso volessi comunicare con tutti i dispositivi collegati dal link fisico? Di certo non faremo, dati N nodi, N comunicazioni di tipo unicast (quelle descritte prima). Lo schema di indirizzamento prevede un *indirizzo di broadcast* che può essere utilizzato per trasmettere pacchetti a tutti i dispositivi.

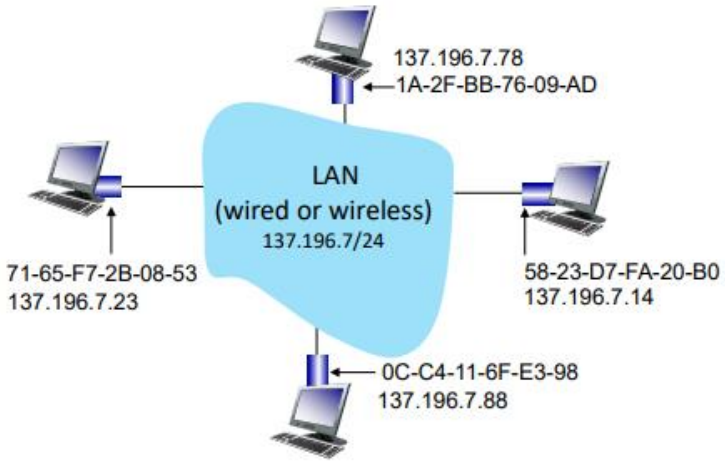
5.7.4 Indirizzi fisici MAC

Lo schema di indirizzamento garantisce univocità. È nostro interesse avere univocità non solo rispetto alla LAN, ma addirittura univocità globale (supponiamo che il dispositivo venga spostato da una rete a un'altra, vogliamo che non si perda l'univocità senza alterare gli indirizzi).

[9] *Inutile dire che questa cosa è illegale, se ci provate vi scoprono. Bisogna essere esperti nel fare i ladri. La raccomandazione è di non costruire uno sniffer, io vi ho solo detto che si può fare (cit.)*

- Un ente globale assegna questi indirizzi (IEEE, Institute of Electrical and Electronics Engineers).
- Gli indirizzi sono assegnati alle schede di rete e non ai singoli dispositivi (ricordiamo che un dispositivo può presentare più di una scheda di rete).
- Gli indirizzi sono a 48 bit, di cui i 24 più significativi identificano il costruttore (l'IEEE assegna solo i bit più significativi, il costruttore gestisce i 24 bit meno significativi garantendo l'univocità sui dispositivi che escono dalla sua fabbrica).

Si parla di indirizzo fisico MAC. L'indirizzo è diverso rispetto al più noto indirizzo IP (detto indirizzo software): l'indirizzo MAC identifica la rete a livello di rete locale, l'indirizzo IP identifica il nodo in un contesto più ampio (quello di Internet).

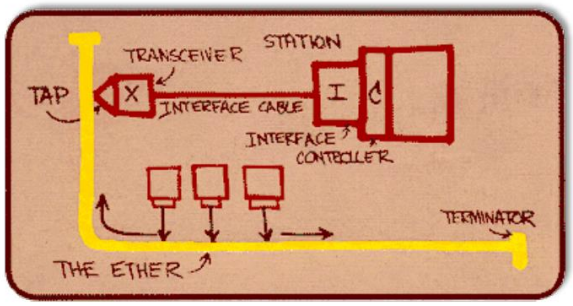


Ogni scheda di rete si vedrà assegnata un indirizzo fisico MAC e un indirizzo software IP. Gli indirizzi fisici sono scritti in esadecimale e divisi in coppie di cifre. Ritorneremo più avanti sulla differenza tra indirizzi MAC e indirizzi IP.

5.7.5 Esempio di LAN: Ethernet

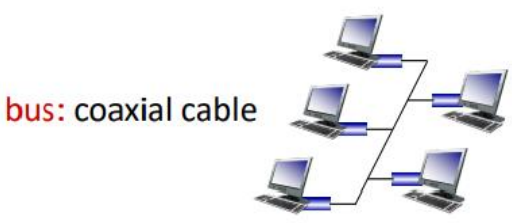
5.7.5.1 Proposta originaria

L'Ethernet è la tecnologia LAN dominante. Vediamo la versione iniziale.



Metcalfe's Ethernet sketch

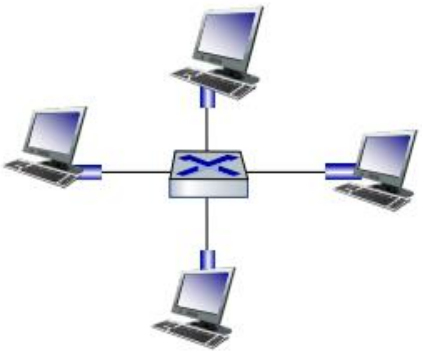
- Abbiamo un bus (quello in giallo) detto *etere*, che presenta terminatori agli estremi (come già anticipato). Da etere deriva Ethernet.
- Computer ha scheda di rete, costituita da controllore e interfaccia (che codifica/decodifica i segnali fisici). Contrariamente ad oggi l'interfaccia non era connessa direttamente alla rete, ma si aveva separato un dispositivo detto tranceiver (che codificava/decodificava il segnale).
- Topologia a bus, con collisioni nel caso in cui più di un nodo trasmetta un segnale sull'etere (unico dominio di collisione). Basata su cavo coassiale.



Ha avuto grande successo per la sua semplicità ed economicità.

5.7.5.2 Passaggio da topologia a bus a topologia a stella: anticipazione del capitolo successivo

La topologia a bus è stata soppiantata dagli anni 90 da una topologia a stella con dispositivo centrale, basata su doppino telefonico. Inizialmente si è posto come dispositivo centrale un hub, successivamente l'hub è stato sostituito dallo switch.



Topologia a stella con switch

- **Cos'è un hub?**
  Un dispositivo che opera a livello fisico e dato un segnale in ingresso lo diffonde su tutte le uscite. È un semplice amplificatore (che vedrete ad Elettronica digitale). Non si adottano particolari algoritmi, e soprattutto il segnale viene reindirizzato verso OGNI uscita (senza possibilità di scelta).
- **Cos'è uno switch?**
  Lo switch è un dispositivo più elaborato, che opera a livello datalink (quindi due livelli della pila implementati, a differenza dell'hub dove si implementa un livello). Contrariamente a prima non vede segnali fisici, ma frame: li riceve, li memorizza nel buffer e successivamente li diffonde sui canali di uscita. Contrariamente a prima è possibile reindirizzare i frame verso particolari uscite, sulla base di algoritmi che vedremo (si guarda l'indirizzo del destinatario e si prende una decisione).

La differenza sostanziale è che con l'hub si ha un unico dominio di collisione, mentre nel caso dello switch le collisioni possono essere evitate, grazie alla bufferizzazione (pacchetti trasmessi in contemporanea vengono disposti in sequenza, si pensi a mittenti diversi che trasmettono pacchetti a uno stesso destinatario in contemporanea). Con lo switch si adotta l'approccio store and forward.

Approfondiremo nel capitolo successivo (Reti a commutazione di pacchetto) cosa succede nel contesto delle reti con switch.

5.7.5.3 Struttura del frame

Il frame trasmesso in Ethernet presenta la seguente struttura:

- Il preambolo consiste in 8 byte fissi:
  - 7 byte con pattern 10101010, e
  - un byte con pattern 10101011

  Si utilizza questa sequenza di byte per la sincronizzazione del clock tra sender e receiver. Il ricevitore fa un campionamento al centro dell'impulso, genera un livello di tensione e lo confronta con la soglia: se siamo sopra la soglia avevamo un 1 logico, altrimenti uno 0 logico. Questa cosa è necessaria perché vi è sempre un minimo di scostamento tra clock del sender e clock del receiver.
- L'indirizzo di destinazione (48 bit), il nodo riceve il frame e lo considera se l'indirizzo di destinazione coincide col proprio indirizzo o quello di broadcast.
- L'indirizzo sorgente (48 bit, mittente)
- Il tipo (1 byte), cioè il protocollo di livello superiore a cui si deve passare i dati contenuti nel campo payload. Solitamente protocollo IP, ma sono possibili anche altri protocolli.
- Il payload consiste nei dati effettivamente trasmessi dal frame.
- CRC sono bit per il Cycle Redundancy Check (4 byte).

La NIC (Network Interface Card):

- riceve un pacchetto dal livello superiore;
- crea un frame, ponendo tutti i dati precedentemente detti.

5.7.5.4 Caratteristiche della rete Ethernet

Su Ethernet possiamo dire le seguenti caratteristiche

- **Connectionless**.
  Non c'è handshake o particolari forme di connessione, si invia il pacchetto senza chiedere.
- **Unreliable**.
  Abbiamo il Cycle Redundancy Check per la error detection, ma non si inviano ACK o NAK. In caso di errori si butta via il payload ricevuto.
- **Protocollo adottato**.
  Il protocollo di accesso adottato è CSMA/CD.

5.7.5.5 Protocollo CSMA/CD: variante per Ethernet

Il protocollo CSMA/CD si basa sull'ascolto del mezzo, per capire se questo è libero o meno.

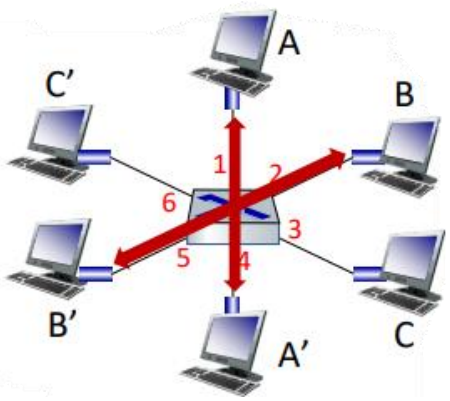
- **Cosa si intende per libero? Quanto tempo si deve attendere per poter dire che il mezzo è libero?**
  Lo standard originale prevede che il mezzo debba essere libero per un tempo necessario a trasmettere 96 bit. Calcoliamo il tempo.
  $$\tau = \frac{96 \text{ bit}}{10 \cdot 10^6 \text{ bit/sec}} = 9.6 \text{ μ sec}$$
  Se il mezzo è libero si può iniziare a trasmettere il frame, e si trasmette l'intero frame. Altrimenti si differisca la trasmissione.
- **Gestione delle collisioni**.
  Durante la trasmissione si continua ad ascoltare nel mezzo, in particolare si misura la potenza ricevuta e la si compara con la potenza del segnale trasmesso. Se la potenza ricevuta è significativamente maggiore della potenza trasmessa allora:
  - non si trasmette più il segnale che si voleva trasmette;
  - al suo posto si trasmette un segnale JAM di 48bit (trasmesso con potenza superiore, come già anticipato per rafforzare la collisione).
- **Invio di pacchetti a seguito di collisione**.
  Problema: se tutti i nodi che hanno colliso aspettassero la trasmissione di 96 bit allora si verificherebbe nuovamente una collisione. Quello che si fa "è sparpagliare nel tempo i tentativi di trasmissione": chi ha generato la collisione sceglie randomicamente un tempo di attesa, noto come *tempo di backoff*. Più precisamente:
  - si considera la collisione i-esima;
  - l'interfaccia di rete del nodo, che ha rilevato la collisione, sceglie un valore K randomico appartenente all'insieme $\{0,1,2,\dots,2^m-1\}$, dove $m = min(i,10)$;
  - si calcola il temo di backoff moltiplicando K per 512 bit, ottengo che il tempo da aspettare è il tempo necessario per trasmettere $K \cdot 512$ byte;
  - dopo 17 tentativi il frame viene buttato via.

  Cerchiamo di dare meglio l'idea. L'insieme dei valori K possibili può essere immaginato come una finestra, che viene ampliata nel caso in cui si continuino a verificare collisioni.
  - Al primo passo $m = min(1,10) = 1$, quindi l'insieme di valori K possibili è $\{0,1\}$
  - Al secondo passo $m = min(2,10) = 2$, quindi l'insieme di valori K possibili è $0,1,2,3$
  - Dopo 10 collisioni m non aumenta più, rimane fisso a 10
  - Dopo 17 collisioni si butta via il frame.
- **Standard di Ethernet diversi**.
  Si distinguono standard diversi dalle seguenti caratteristiche: topologia (bus o hub); uso di cavo coassiale o doppino telefonico. Per quanto riguarda il cavo coassiale si distinguono le seguenti tecnologie:
  - cavo coassiale spesso (segmenti lunghi fino a 500 m, con possibilità di inserire al più quattro ripetitori, quindi al più cinque segmenti collegati con una lunghezza complessiva di 2500 m)
  - cavo coassiale sottile (come prima, ma segmenti lunghi fino a 200 m).

6 RETI A COMMUTAZIONE DI PACCHETTO (PACKET SWITCHED NETWORKS)

6.1 Introduzione: passaggio da reti con hub a reti con switch

In questo capitolo approfondiamo il passaggio da reti con topologia a stella basate su Hub a reti con topologia a stella basate su switch. Si opera a livello datalink e si ragiona nell'ottica dei frame.



- È un **dispositivo decisamente più intelligente** rispetto all'Hub, che gestisce frame Ethernet con approccio store and forward.
- È un **dispositivo trasparente**: gli host non sanno che esiste lo switch, pensano di essere collegati direttamente.
- È un **dispositivo che permette trasmissioni simultanee**, a differenza dell'Hub che aveva un unico dominio di collisione (lo switch garantisce un throughput superiore). Contribuisce a quello soprattutto la bufferizzazione: la presenza di un buffer permette di memorizzare frame ricevuti in simultanea dallo switch, inviandoli successivamente in sequenza. Sono possibili fino ad n trasmissioni simultanee, dove n è il numero di porte possedute dallo switch.
- È un **dispositivo *plug and play***: questo perché gli switch sono *self-learning* (capiremo a breve cosa significa) e quindi non necessitano di essere configurati.

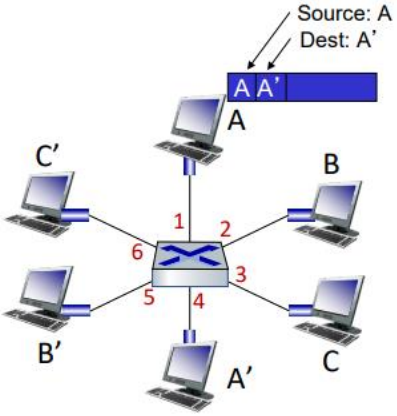
Il protocollo utilizzato con lo switch è CSMA/CD. L'ultimo punto è quello che ci porta a riflettere maggiormente: come può un dispositivo sapere dove reindirizzare un frame, se non lo si configura a priori?

6.2 Tabella di forwarding: concetto e riempimento

Ogni switch mantiene una tabella di commutazione, che per ogni indirizzo di destinazione indica l'uscita verso la quale si deve instradare il pacchetto. È utile perché:

1. lo switch riceve il frame e lo memorizza;
2. guarda l'indirizzo MAC di destinazione;
3. individua l'uscita verso cui redirigere il frame visitando la tabella di forwarding;
4. reindirizza il frame.

Domanda sorge spontanea: chi riempie la tabella? Si tenga a mente che se è trasparente e plug and play non si può pensare a una configurazione a priori da parte dell'amministratore. Supponiamo che un host A voglia trasmettere all'host A'



1. lo switch riceve il frame e lo memorizza;
2. **[Aggiornamento della tabella]** aggiorno la tabella di commutazione registrando l'indirizzo di A (colui che ha trasmesso il frame) e il link attraverso cui lo switch ha ricevuto il pacchetto;
3. guarda l'indirizzo di destinazione (l'indirizzo di A');
4. verifico se nella tabella di commutazione è presente il link di uscita verso cui redirigere il frame;
    a. se è presente si redirige il frame verso quel link di uscita (***invio selettivo***);
    b. se non è presente si redirige il frame verso tutti i link di uscita, si trasmette a tutti (***flooding***).

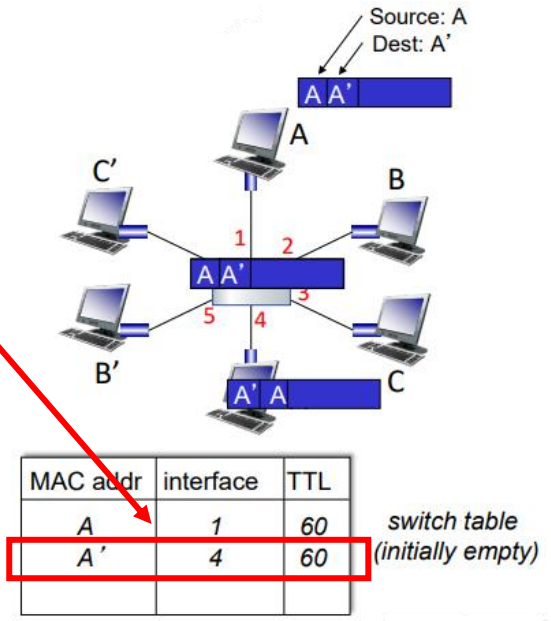
MAC addr	interface	TTL
A	*1*	*60*

Switch table (initially empty)

L'aggiornamento della tabella risulta superfluo nella trasmissione al momento gestita, ma riveste utilità nelle successive trasmissioni.

Supponiamo che l'host A' voglia rispondere all'host A, trasmettendo un nuovo frame. Lo switch si comporta nel seguente modo:

1. lo switch riceve il frame e lo memorizza;
2. aggiorna la tabella di commutazione ponendo le informazioni descritte prima, ma relative ad A';
3. guarda l'indirizzo di destinazione;
4. visita la tabella di commutazione e trova il link di uscita del destinatario (colui che ha inviato prima ha aggiornato la tabella ponendo questa informazione);
5. si fa invio selettivo, lo switch trasmette il frame solo verso il link di uscita indicato dalla tabella di commutazione.



MAC addr	interface	TTL
A	1	60
A'	4	60

switch table (initially empty)

A questo punto siamo contenti: se i due host continuano a interloquire lo faranno solo con invii selettivi, perché adesso la tabella di commutazione presenta i link di uscita di entrambi gli host.

- Se A vuole trasmettere un frame ad A' troverà il link di uscita relativo ad A'
- Se A' vuole trasmettere un frame ad A troverà il link di uscita relativo ad A

Rimane solo una cosa da chiarire della tabella: il campo TTL. Con TTL intendiamo time to leave, e consiste nel tempo superato il quale l'informazione non è più valida: chi visualizza la riga della tabella con TTL superato la ignora e trasmette a tutti. Necessario farlo perché dopo un certo tempo le informazioni in tabella potrebbero non essere più aggiornate.

C'è la possibilità che un host non venga individuato? Solo se questo non trasmette pacchetti (ipotesi rara).

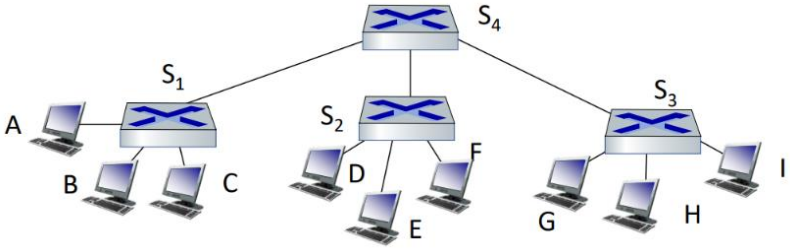
Segue un esempio di pseudocodice sul come viene gestito il frame non appena arrivato nello switch (aspetto in più, si scarta il pacchetto se per la tabella di forwarding l'invio deve avvenire attraverso lo stesso link con cui il pacchetto è arrivato nello switch)

```
when  frame received at switch:
  1. record incoming link, MAC address of sending host
  2. index switch table using MAC destination address
  3. if entry found for destination
     then {
     if destination on segment from which frame arrived
        then drop frame
         else forward frame on interface indicated by entry
      }
     else flood  /* forward on all interfaces except arriving interface */
```

6.3 Switched Ethernet

6.3.1 Gerarchie di switch

Gli switch self-learning possono essere messi insieme. Si costruiscono gerarchie per mezzo di questi collegamenti



- Gli switch che interconnettono direttamente gli host sono detti switch di primo livello.
- Lo switch che interconnette gli switch di primo livello è detto switch di secondo livello.

Si può introdurre un numero maggiore di livelli, ma senza esagerare (occhio alla latenza).

6.3.2 Proprietà dello Switched Ethernet

I vantaggi

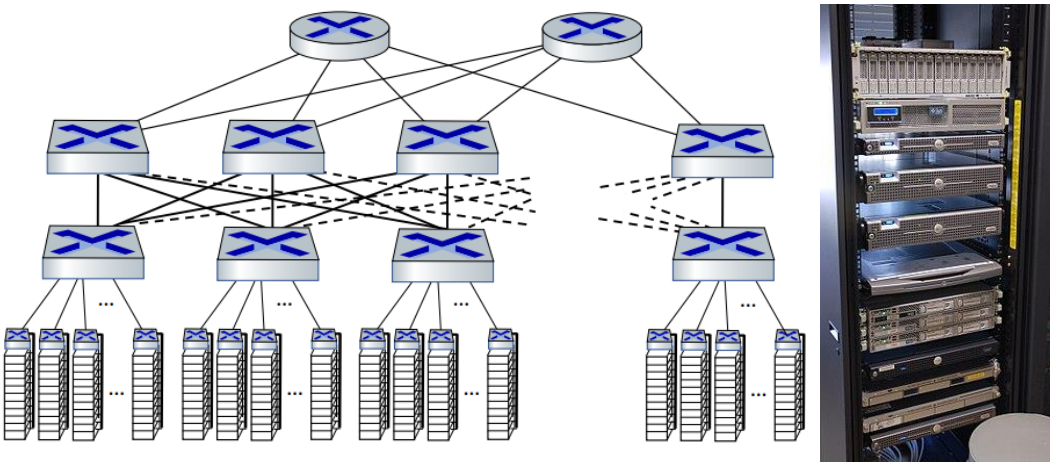
- **Eliminazione delle collisioni**
  Non si ha un unico dominio di collisione.
- **Supporto di link eterogenei**.
  Lo switch è in grado di gestire schede di rete diverse, anche quelle molto vecchie aventi velocità decisamente minore rispetto allo standard attuale.
- **Facilità di gestione**.
  In caso di porte non funzionanti basta isolare la porta e il sistema continua a funzionare correttamente.
- **Migliore sicurezza**.
  La trasmissione non è più broadcast, sniffare diventa più difficile.

6.3.3 Datacenter networks

I datacenter sono un perfetto esempio di Switched Ethernet: abbiamo decine, se non centinaia di hosts ravvicinati tra loro. Sono utilizzati da servizi di e-business (Amazon), content-servers (Youtube, Apple, …), search engines (Google, …).

Le sfide da affrontare sono le seguenti:

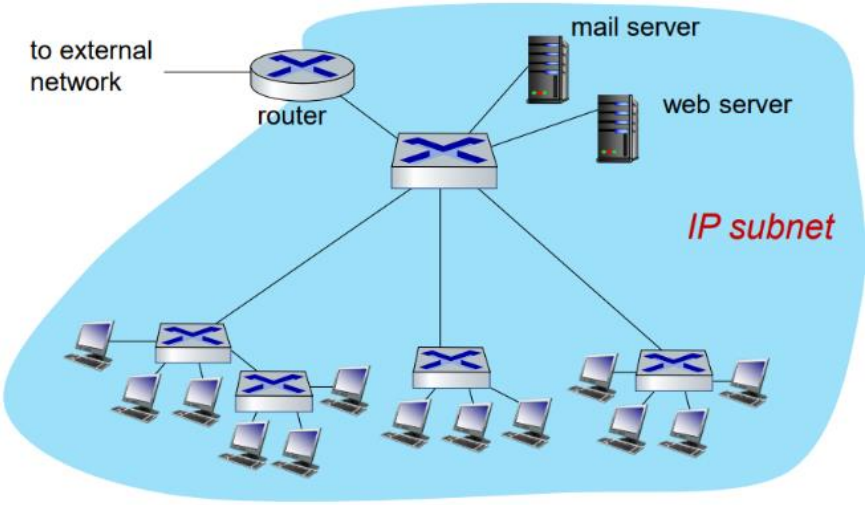
- si offre una molteplicità di servizi, ciascuno richiesto da un numero molto elevato di utenti;
- affidabilità;
- gestione del carico tra i vari hosts presenti nel datacenter;
- evitare effetto collo di bottiglia.



A livello più basso presenti dei racks (esempio nella figura a lato, armadietti contenenti una serie di dispositivi): 20-40 host, collegati a uno switch (top of rack switch), a loro volta collegati a switch di livello 2 e così via.

6.3.4 Esempio: una piccola rete istituzionale

Si introduce come esempio la rete istituzionale a lato, che presenta una gerarchia di switch, ma anche un mail server e un web server.



Si osservi che i due server, che rivolgono servizi anche verso l'esterno, sono posti a frontiera:

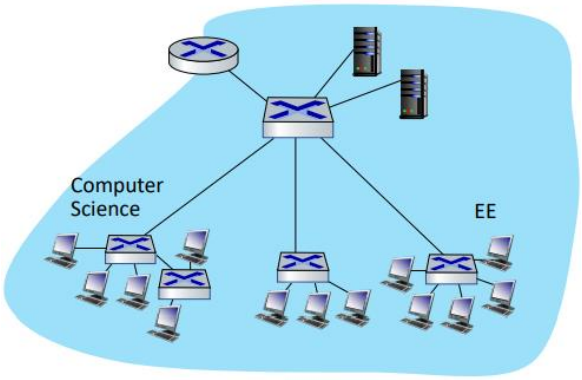
- per ragioni di tempo (occhio alla latenza);
- per isolarle dal resto della rete, per ragioni di sicurezza.

Il numero di switch sarà maggiore rispetto alla semplificazione della figura, ma con un occhio sull'aumento della latenza (ritardo) e la riduzione della probabilità di vedere il pacchetto trasmesso correttamente a destinazione. Problema di reliability, affidabilità: se un server smette di andare non posso ignorare.

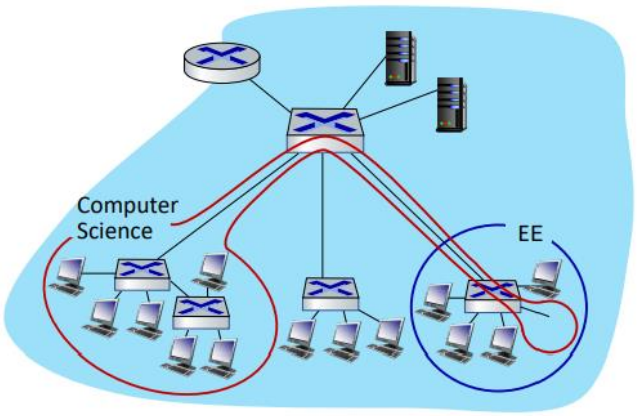
6.4 Reti virtuali (VLAN)

6.4.1 Motivi per cui parliamo di VLAN

Immaginiamoci un contesto dove ogni gruppo di utenti (per esempio gruppi di lavoro nell'ambito della ricerca universitaria) ha un proprio switch e quindi una propria LAN. L'organizzazione descritta presenta i seguenti problemi.



- **Unico dominio di broadcast**.
  In un'organizzazione basata su switch non abbiamo più un singolo dominio di collisione, ma abbiamo un singolo dominio di broadcast! Se un nodo invia un frame in broadcast il traffico fluisce attraverso l'intera rete. In una rete istituzionale dove gli utenti hanno ruoli diversi (si pensi all'Università, ci sono gli studenti ma ci sono anche i docenti e il personale Tecnico-Amministrativo) è estremamente pericoloso non attuare una qualche forma di isolamento del traffico.
- **Uso inefficiente delle risorse**.
  Senza ulteriori modifiche facciamo un uso inefficiente delle risorse. In generale gli switch di livello più basso hanno poche porte effettivamente utilizzate.
- **Utenti che si spostano nell'edificio**.
  Ogni switch è relativo a un gruppo di utenti, e non è detto che un utente si sposti da un punto all'altro dell'edificio solo quando cambia gruppo di utenti.

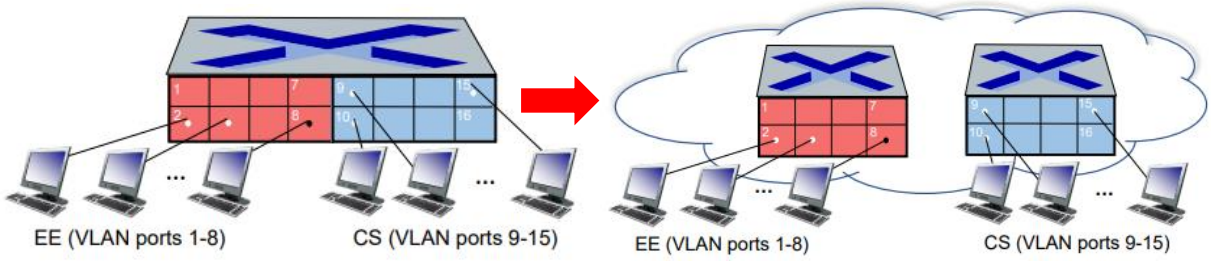


6.4.2 Spiegazione

Risolviamo i problemi descritti introducendo le VLAN, cioè le LAN virtuali. Si utilizza un singolo switch (quindi in realtà abbiamo un'unica LAN), ma su questo si definiscono delle LAN virtuali che operano in modo indipendente tra loro.

<u>Una LAN virtuale è definita assegnato ad essa un certo numero di porte.</u>

Il tutto viene visto dagli host come se ci fossero più switch…

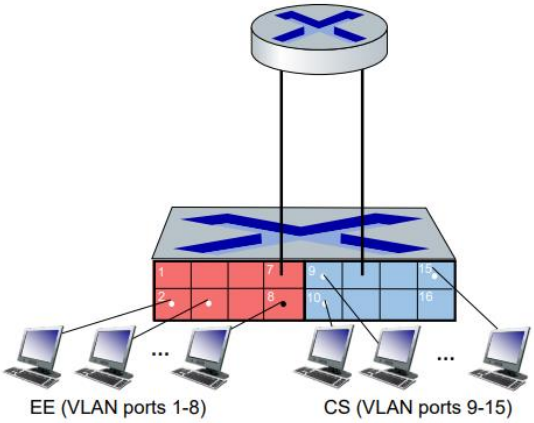


… ma in realtà lo switch è uno solo.

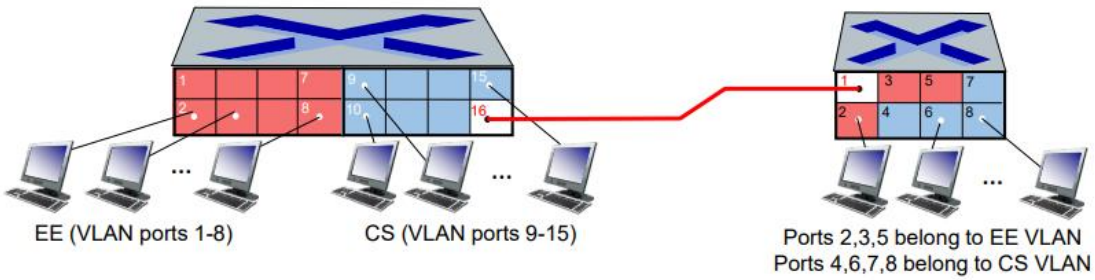
- Si associa a ciascuna VLAN un certo numero di porte. Possiamo immaginare che la VLAN rappresenti una particolare organizzazione, collocata in una particolare area dell'edificio.
- In caso di spostamento di un utente da un punto a un altro dell'edificio basta alterare la configurazione, aggiungendo una nuova porta tra quelle relative alla VLAN dell'organizzazione dell'utente.
- In caso di aumento del numero di utenti afferenti all'organizzazione si altera la configurazione aumentando il numero di porte assegnate alla VLAN (gestione più efficiente delle porte).

Domanda che sorge spontanea: cosa succede se l'host di una VLAN vuole trasmettere un pacchetto all'host di un'altra VLAN? Si introduce sostanzialmente una sorta di "router virtuale"

- Il "router virtuale" è implementato all'interno dello switch dove abbiamo definito le VLAN.



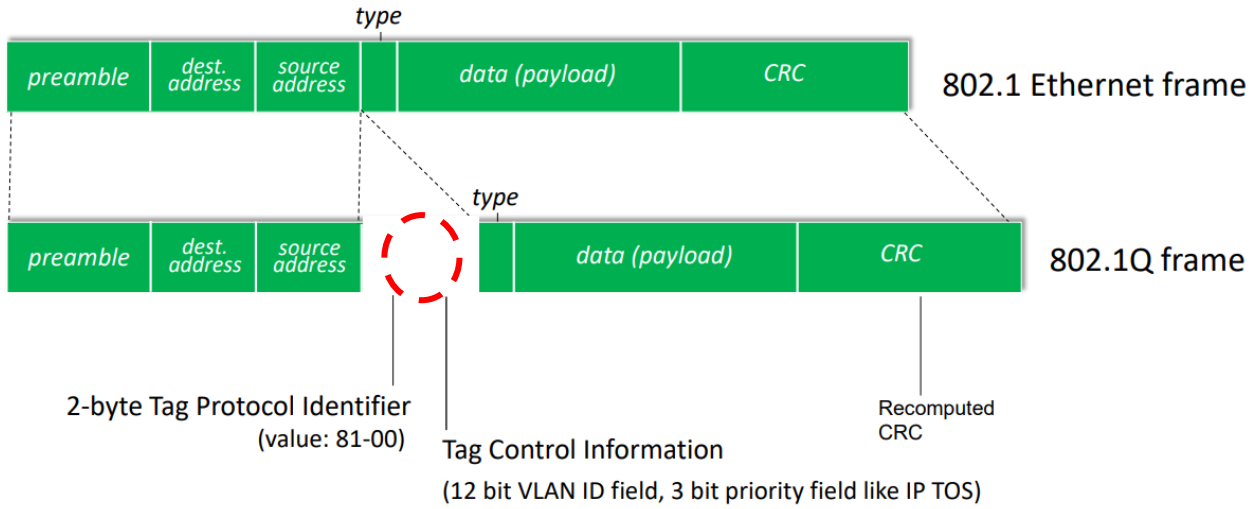
- È possibile che le VLAN si estendano su più switch fisici (cosa non rara, soprattutto in contesti con un numero elevato di utenti). Si consideri la seguente figura, dove le porte di una VLAN sono divise tra due switch.



La soluzione consiste nel cosiddetto *VLAN trunking*: si introducono delle porte speciali non assegnate ad alcun host, che fungono da aggancio tra i due switch.

6.4.3 Modifiche alla struttura del frame Ethernet

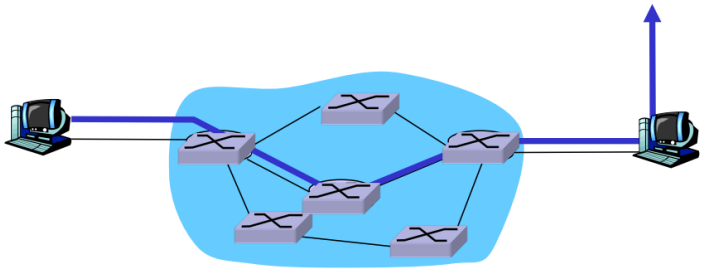
Problema: se la VLAN spazia su più switch è necessario introdurre degli identificatori di VLAN. Necessario modificare la struttura del frame, visto che Ethernet è stato definito prima delle VLAN. Quando si fa la comunicazione tra nodi appartenenti a VLAN diversi aggiungo due campi, in particolare il Tag Control Information che identifica la VLAN.



Lo switch, ricevuto il frame, provvede a rimuovere i due campi prima di trasmettere il frame al destinatario: segue una completa trasparenza dell'operazione.

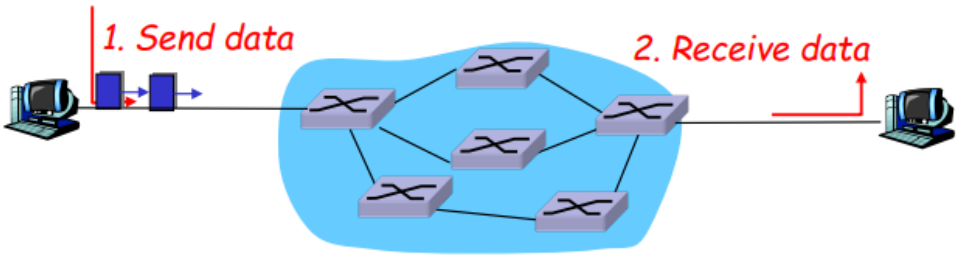
6.5 Wide-Area Packet Switched Networks

Gli approcci descritti possono essere utilizzati per collegare host anche in reti più ampie (reti con copertura non locale). Nulla ci vieta di adottare una struttura diversa da quella gerarchica, per esempio una topologia a maglia (mesh): l'organizzazione non più ad albero rende possibile la creazione di più percorsi per raggiungere un determinato host.



Superata la mera rete locale passiamo dall'identificazione delle schede di rete con indirizzi MAC all'identificazione degli host per mezzo di indirizzi IP[10]. A questo punto si distinguono due tipologie di servizio.

- **Servizio *connectionless* (o *datagram service*)**.
  Ogni pacchetto è gestito per conto proprio senza introdurre a priori una connessione. Utile quando si vuole trasmettere poche informazioni (ad esempio un telegramma). È l'approccio tenuto fino ad ora nelle reti locali.



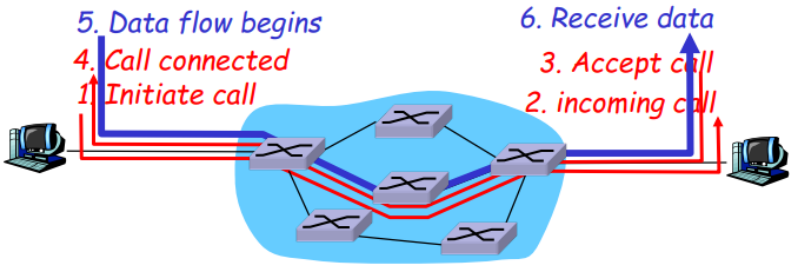
[10] **Osservazione**. Le tabella di forwarding non stabiliscono corrispondenze rispetto a specifici indirizzi IP (solitamente), ma per ogni riga si definisce un range di indirizzi IP per cui vale indicare una particolare porta di uscita. Se l'indirizzo IP destinatario di un pacchetto appartiene a quel range allora si redirige quel pacchetto verso la porta di uscita indicata.

Abbiamo le tabelle di forwarding già viste nelle sezioni precedenti, dove associamo ad un indirizzo di destinazione un particolare link di uscita. Si osservi che:

- non esiste il concetto di connessione (niente convenevoli e assenza di tabelle che mantengono informazioni sulle connessioni stabilite);
- pacchetti inviati in sequenza potrebbero arrivare a destinazione passando attraverso percorsi diversi.

- **Servizio *connection***.

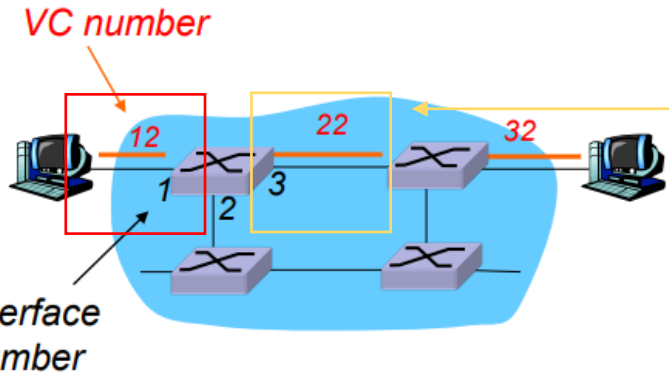
Contrariamente a prima si stabilisce un circuito nella rete che sarà dedicato esclusivamente ai due host. Solo dopo aver stabilito il circuito i due host inizieranno a scambiarsi pacchetti.



Chiaramente la cosa è più costosa. Affinchè si possa gestire il circuito virtuale è necessario introdurre un'ulteriore tipologia di tabella di forwarding (diversa da quella definita precedentemente).

- Il percorso va da un mittente a un destinatario.
- Il percorso è caratterizzato da router e link di collegamento tra host e router.
- Ogni link di collegamento è identificato da un VC number.
- Ogni tabella di forwarding non mantiene l'intero percorso: stabilisce le correlazioni tra porte di ingresso e porta di uscita relativamente a un particolare circuito virtuale. Per fare ciò si utilizzano i VC number.
- I pacchetti presenteranno come destinatario il VC number (non l'indirizzo dell'host).

Per avere le idee chiare si consideri il seguente esempio



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

7 INTERCONNESSIONE DI RETI (INTERNETWORKING): PROTOCOLLO IP

7.1 Introduzione

7.1.1 Astrazione dell'Internetwork

Pila protocollare
Appplication *messages*
Transport *segments*
Network *datagrams*
Link / Datalink *frames*
Physical

Con *internetwork* (*interrete*, detto in italiano) intendiamo una rete di reti, cioè un sistema costituito da un insieme di reti collegate tra loro. Queste reti possono essere basate su tecnologie diverse, ergo:

- trasmettono segnali fisici diversi;
- utilizzano formato dei frame diversi;
- hanno schema di indirizzamento diverso;
- ecc….

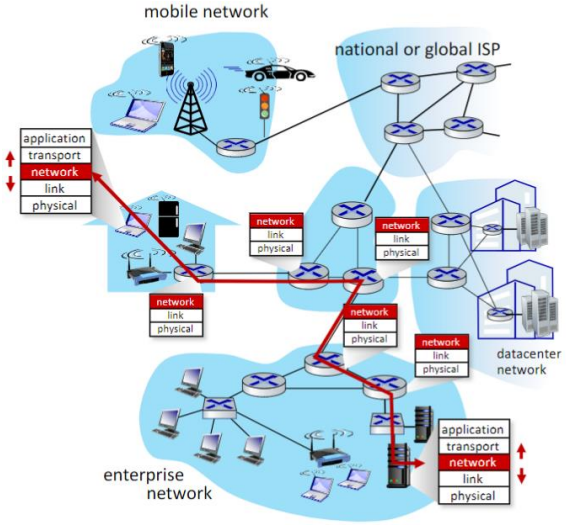
Per permettere la comunicazione tra reti eterogenee è necessaria la presenza di dispositivi che assumono il ruolo di interpreti: questi dispositivi sono i router, dispositivi dotati di porte di ingresso e di porte d'uscita, che ricevono pacchetti e li inoltrano in uscita. Ma è uno switch? Sostanzialmente sì, ma si opera a livello network e non più a livello datalink. Si vuole realizzare un sistema trasparente dove l'utente vede un unico sistema di comunicazione (un'altra astrazione) e non si accorge delle differenze tra reti.

7.1.2 Obiettivo e livello della pila

Nel livello Network abbiamo l'implementazione del cosiddetto *Internet Protocol* (IP).

Lo scopo è creare l'astrazione dell'Internetwork, e permettere il trasferimento dei pacchetti (a questo livello noti come datagram) da un host sorgente a un host destinatario, passando attraverso le varie reti e i vari router (che connettono le reti dell'interrete).

In alcuni casi switch e router sono incapsulati all'interno della stessa macchina, ma per comodità supporremo che i due dispositivi rimangano distinti.



7.1.3 Differenza e legame tra forwarding e routing

Prima di proseguire risulta necessario sottolineare una volta per tutte la differenza tra forwarding e routing:

- Il ***forwarding*** consiste nell'insieme di quelle azioni attraverso cui il router, ricevuto il datagram, decide opportunamente quale uscita instradare il datagram.
- Il ***routing*** è un processo a monte della trasmissione del pacchetto, si individua il percorso più breve per il passaggio del pacchetto dall'host sorgente all'host destinatario.

Possiamo fare un'analogia con un viaggio in automobile:

- Col *forwarding* si decide di viaggiare senza una programmazione a priori, si prendono le strade indicate dai cartelli e non ci interessa se prendendo una certa strada arriveremo a destinazione impiegando un tempo maggiore.
- Col *routing* si decide tutto il percorso prima di iniziare il viaggio. Si sceglie il percorso ideale in modo da minimizzare costi e tempo.

Abbiamo già visto la tabella di forwarding nei capitoli precedenti: si analizza l'indirizzo di destinazione del pacchetto, si consulta la tabella di forwarding e in presenza di una relativa riga si instrada verso una particolare uscita. Il routing consiste nell'aggiornamento della tabella di forwarding: si pongono le uscite associate agli indirizzi in modo tale che il pacchetto attraversi il miglior percorso possibile. Si attua ciò con *algoritmi di routing*.

7.1.4 Recap: tipologie di servizio

Ricapitoliamo quali sono le due tipologie di servizio possibile, su cui vi è stata forte discussione negli anni 90. La prima tecnologia nasce dalla telefonia classica, mentre la seconda è stata pensata per Internet in funzione delle caratteristiche del traffico (irregolare e discontinuo).

- ***Connection (Virtual Circuit)***
  Si stabilisce preliminarmente un circuito virtuale e tutti i pacchetti seguono lo stesso percorso.
  - Pensata per traffico voce e video.
  - Flusso di informazioni continuo.
  - Dispositivi terminali sono i telefoni (quelli di una volta), dispositivi dumb (stupidi).
  - Complessità mantenuta nel core della rete.
- ***Connectionless (datagram service)***
  Ogni pacchetto è gestito individualmente sulla base dell'indirizzo destinatario. Per ogni datagram il nodo: memorizza nel buffer; controlla l'indirizzo di destinazione; consulta le tabelle di forwarding (riempite col routing); decide verso quale uscita instradare il datagram.
  - Niente connection-setup o connection-teardown.
  - Non è necessario tenere in memoria informazioni sulle connessioni.
  - Non è detto che tutti i pacchetti seguano lo stesso percorso (quello ottimale, si consideri che il percorso ottimale potrebbe congestionarsi e non essere più ottimale, oppure potrebbe non essere più percorribile).
  - Approccio adottato e pensato per Internet!
    - Servizio elastico: ho momenti in cui il traffico viene generato, e altri no.
    - Dispositivi terminali, gli hosts, sono dispositivi intelligente.
    - Complessità mantenuta agli estremi della rete (sugli hosts, che sono intelligenti). Questo perché la semplicità del core rende tutto più veloce.

Negli anni 90 i telecomunicazionisti ritenevano che l'approccio migliore fosse quello Virtual Circuit, soprattutto per quanto riguarda l'erogazione di servizi come lo streaming. Alla fine ebbe la meglio l'approccio connectionless, che adesso è sostanzialmente sinonimo di Internet. Un servizio di tipo streaming può essere erogato fornendo una banda sufficiente.

- L'approccio adottato è *best-effort*, semplicità che ha favorito una larga diffusione di Internet.
- Si è privilegiata la velocità e l'elasticità del traffico (mantenendo la complessità agli estremi), a discapito di garanzie esplicite:
  - *Reliable Delivery,*
  - *In-order delivery,*
  - *Guaranteed Minimal Bandwidth,*
  - *Guaranteed Bounded Delay,*
  - *Guaranteed Maximum Jitter* (variazione del ritardo);
  - *Security Services (Data confidentiality, Data Integrity, Source Authentication)*.

Se abbiamo necessità di una connessione affidabile ricorriamo al protocollo TCP (che vedremo più avanti).

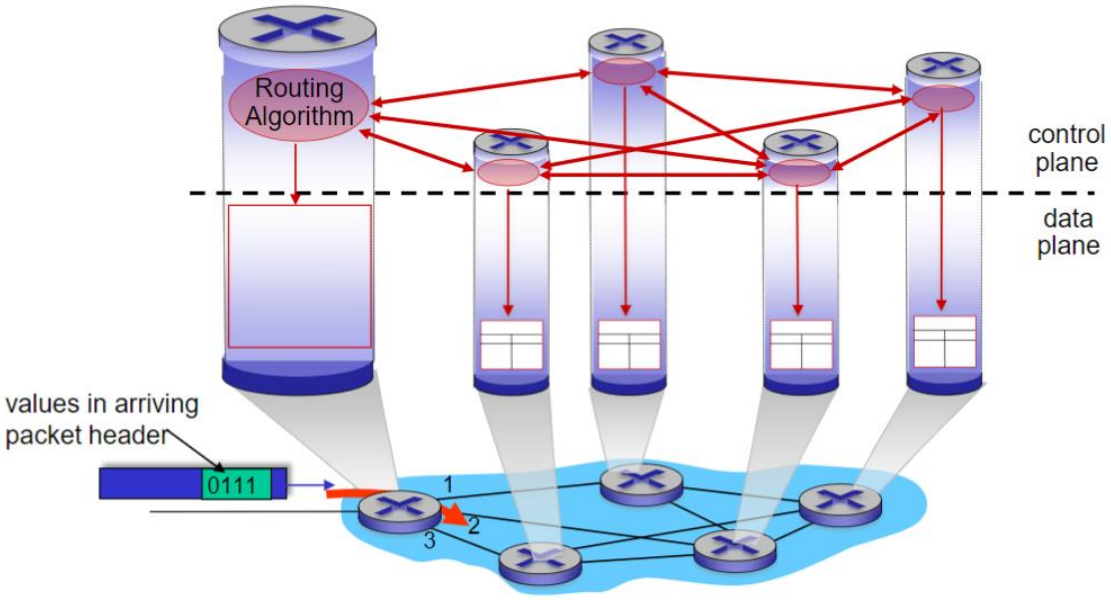
Network Architecture	Service Model	Quality of Service (QoS) Guarantees ?			
		Bandwidth	Loss	Order	Timing
Internet	best effort	none	no	no	no
ATM	Constant Bit Rate	Constant rate	yes	yes	yes
ATM	Available Bit Rate	Guaranteed min	no	yes	no
Internet	Intserv Guaranteed (RFC 1633)	yes	yes	yes	yes
Internet	Diffserv (RFC 2475)	possible	possibly	possibly	no

7.2 Cosa abbiamo dentro un router?

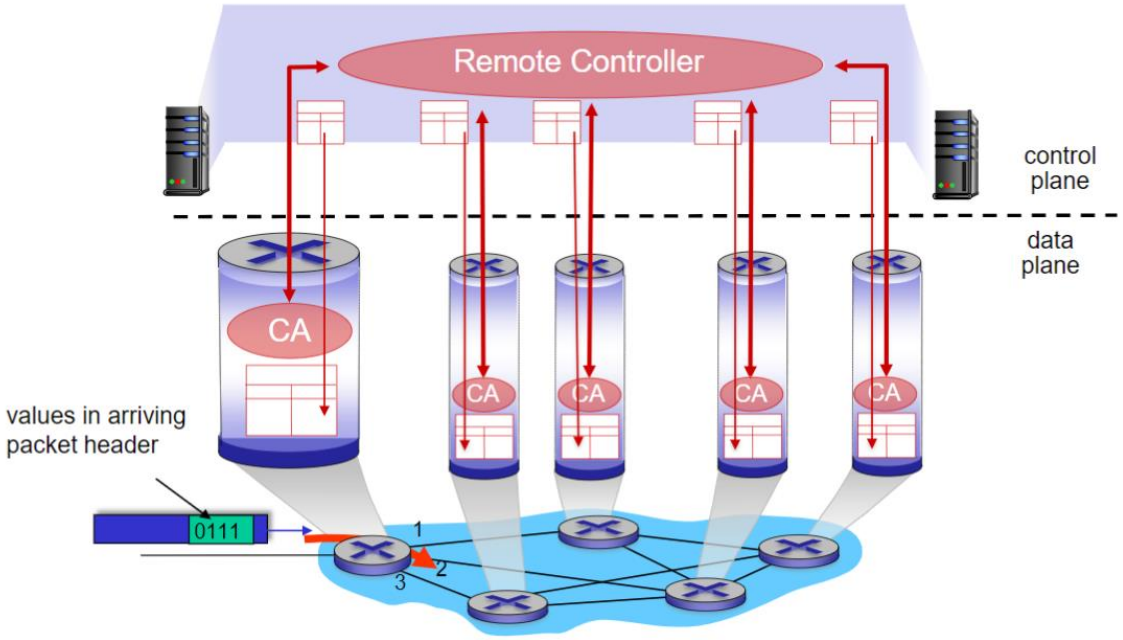
7.2.1 Piani di lavoro nel router

Introdotto forwarding e routing possiamo definire i due “piani di lavoro” con cui descrivere un router.

- **Piano di dati**.
  Locale al router, si determina come porre il datagram in output.
- **Piano di controllo**.
  Si determina il tragitto dei datagram per mezzo di algoritmi di instradamento. Approcci possibili:
  - **Traditional routing algorithms**
    Gli algoritmi di instradamento sono implementati internamente in ogni router, e questi comunicano tra di loro per determinare i valori da inserire nelle tabelle (approccio cooperativo)



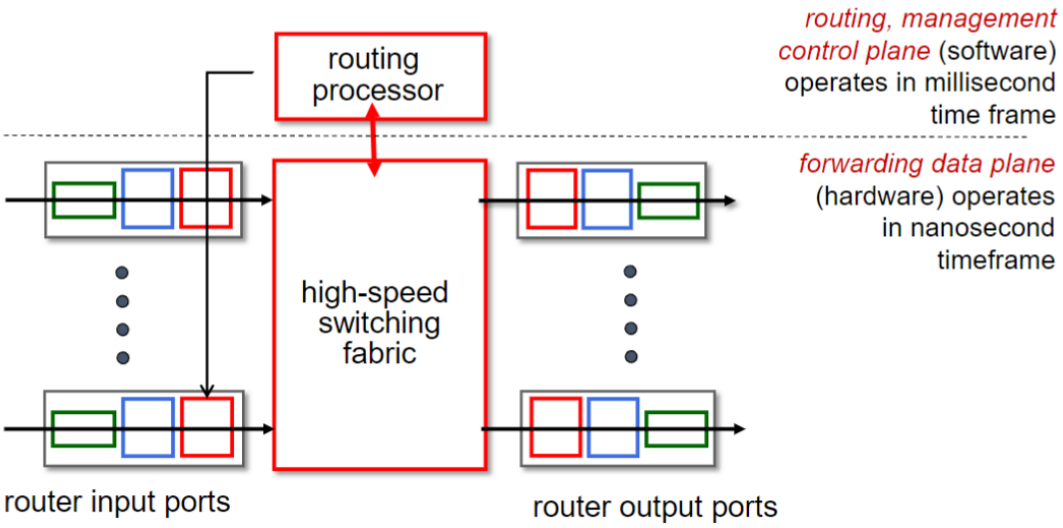
  - **Software-defined networking (SDN)**
    Il routing è delegato ad una unità centralizzata, che vigila sui percorsi dei datagram.



7.2.2 Architettura del router

7.2.2.1 Ruolo del router e componenti dell'architettura

Il router svolge due funzioni fondamentali:

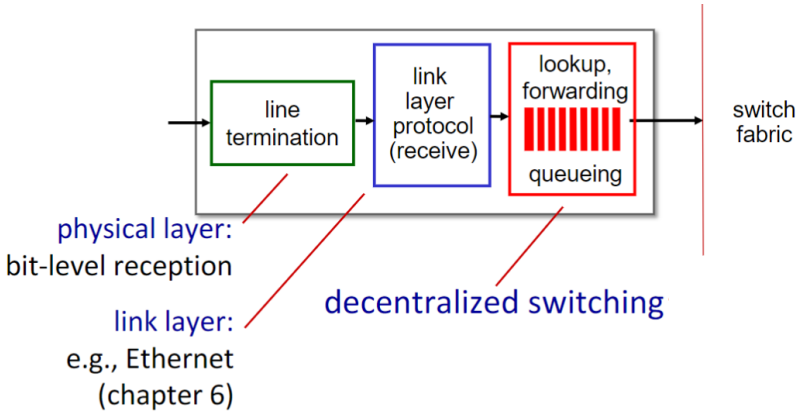


- esegue il forwarding dei datagram nel ***high-speed switching fabric*** (instradare un datagram verso un'uscita);
- esegue protocolli di routing nel ***routing processor*** (che permette di stabilire il percorso ottimale atto a raggiungere la destinazione desiderata).

Il secondo processo è funzionale al primo (non c'è forwarding senza routing).

7.2.2.2 Porte di ingresso

La porta di ingresso implementa sostanzialmente tutti e tre i livelli realizzati sul router:

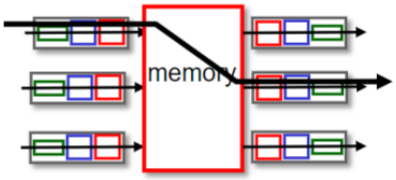


- *livello fisico*, perché la porta è collegata con un link fisico;
- *livello datalink*, si fa error detection (pacchetto scartato se ci sono errori), decapsulation (estrazione dei dati nel payload);
- *livello Network*, il datagram viene ricevuto e (se non è il primo) viene posto in buffer, successivamente si prende l'indirizzo di destinazione, si visita la tabella di forwarding e si instrada verso l'uscita adeguata

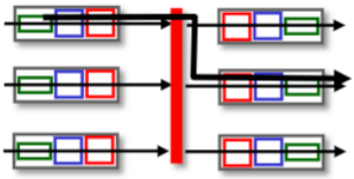
7.2.2.3 Logica di commutazione

Nell'*high-speed switching fabric* si implementa la **logica di commutazione**, il sistema che permette al datagram di spostarsi dal buffer della porta di ingresso al buffer della porta di uscita.

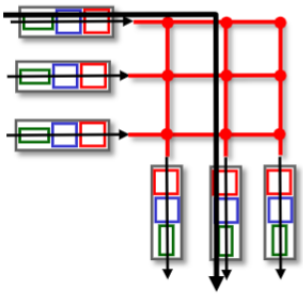
- **Switching con memoria condivisa**.
  In un certo senso possiamo immaginarci un processore per ogni porta di ingresso e ogni porta di uscita. Idealmente abbiamo una memoria condivisa tra tutti i processori (ricordarsi quanto visto a Sistemi Operativi). Non è molto performante, dato che sono necessari accessi in memoria.



- **Switching con bus**.
  Possiamo immaginarci un bus che collega tutte le porte di ingresso e tutte le porte di uscita. Accesso più veloce perché non c'è la memoria condivisa, permette l'invio di più pacchetti al secondo, ma ritorna il problema delle reti locali sull'uso esclusivo (accessi multipli non consentiti, conseguente riduzione del throughput).

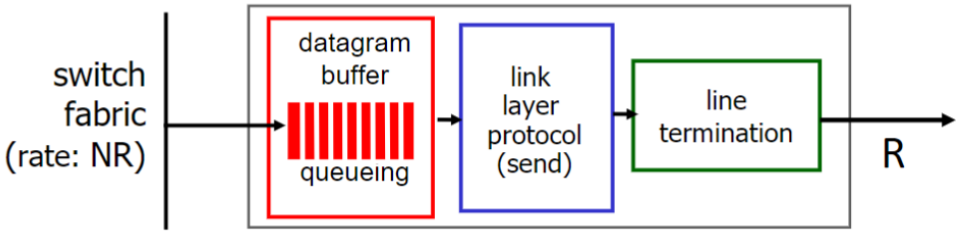


- **Switching con crossbar**.
  Struttura più complessa che permette più instradamenti in contemporanea (Il throughput aumenta). Diverse linee di ingresso e diverse linee di uscita, chiaramente il costo per la realizzazione è maggiore.



7.2.2.4 Porte di uscita

La porta di uscita presenta una struttura simile a quella della porta di ingresso, ma rovesciata.



- A seguito della logica di commutazione il pacchetto si trova sulla coda in uscita. Anche in questo caso si pone il datagram in un buffer.
- La prima idea che abbiamo è che la coda potrebbe essere gestita secondo logica FCFS. In realtà potrebbe essere utile implementare meccanismi di scheduling basati su priorità, per esempio dando precedenza a traffico con particolari esigenze di servizio.

7.3 Protocollo IP (Internet Protocol)

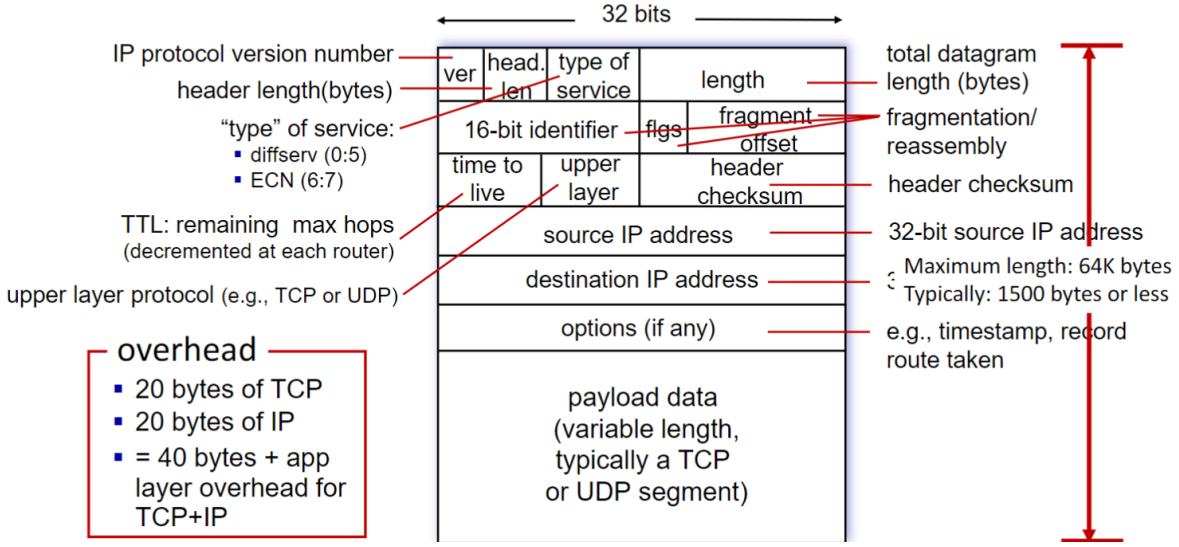
7.3.1 Scopo del protocollo

Il protocollo IP implementa tre funzionalità essenziali:

- definisce formato dei datagram;
- definisce schema di indirizzamento su Internet;
- gestisce il forwarding dei pacchetti.

7.3.2 IP Datagram format

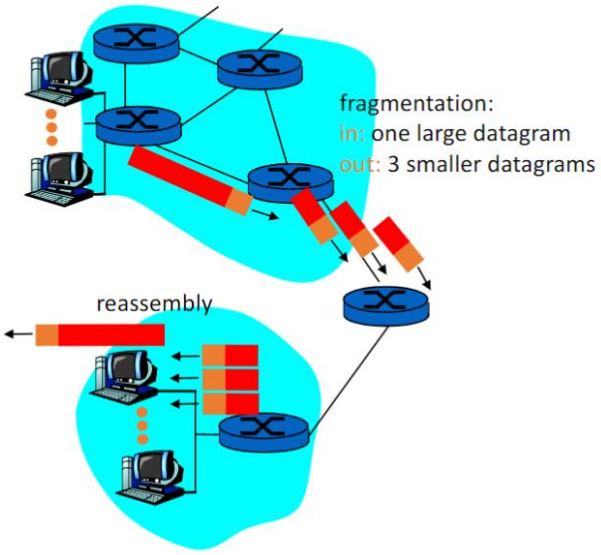
Perché è necessario introdurre un nuovo formato di pacchetto? Ogni rete ha il suo formato di frame, necessario adottare un formato al di sopra di tutte le tipologie di reti.



- Intestazione con informazioni di controllo (dimensione variabile per la presenza di campi opzionali, si suppone per comodità che non ci siano – nel caso 5 parole da 32 bit)
  - `ver` (4 bit): versione del protocollo IP adottata (la versione 4 è la più utilizzata al momento)
  - `head len` (4 bit): lunghezza dell'intestazione in parole da 32 bit.
  - `type of service` (8 bit): tipo di dato trattato
  - `length` (16 bit): dimensione totale del datagram (intestazione + dati, 16 bit)
  - `16-bit identifier`: numero identificativo del datagram
  - `flags` (3 bit), `fragment offset` (13 bit): che chiariremo più avanti.
  - `time to live` (8 bit): bit uguali tutti ad uno, si decrementa ad ogni passaggio di router. Tempo di vita del router, lo si introduce per evitare casi in cui il datagram entra in un loop, e gira di continuo nella rete senza arrivare a destinazione
  - `upper layer` (8 bit): protocollo adottato nel livello superiore, a cui si deve consegnare il contenuto del Payload (TCP o UDP)
  - `header checksum` (16 bit): error detection
    - Si osservi che il *checksum* è calcolato solo sull'header.
      - Si vogliono minimizzare i tempi.
      - I campi dell'intestazione sono più importanti (se si deve scegliere cosa controllare e cosa no): se questi vengono alterati è possibile che il datagram non arrivi a destinazione, quindi è inutile continuarlo ad inviare.
      - Inoltre: non avevamo detto che Internet non pone nessun controllo di affidabilità?
      - Il checksum cambia ad ogni instradamento di router, dato che il time to live viene decrementato ogni volta.
  - indirizzi IP sorgente e destinatario (entrambi 32 bit).
  - campi opzionali, come timestamp e source routing (indirizzo dei router attraverso cui il pacchetto deve essere instradato – mai usato).

7.3.3 Frammentazione dei datagram

Facciamo alcune riflessioni sulla dimensione del datagram, così come sulla dimensione del frame Ethernet. Sappiamo da quanto detto sulla pila protocollare che il datagram verrà "incapsulato" al livello inferiore (livello link) all'interno di un frame (sarà il payload del frame Ethernet). Il frame avrà la forma vista nei capitoli precedenti.



- **Qual è la dim. massima del datagram?** Il campo length è lungo 16 bit, quindi $2^{16} = 64$ kbit
- **Qual è la dim. massima del payload nel frame Ethernet?** 1500 bit.

Il datagram elaborato a livello Network può avere dimensione superiore rispetto alla dimensione massima del campo payload in Ethernet ($64\text{kbit} \gg 1500$ bit).

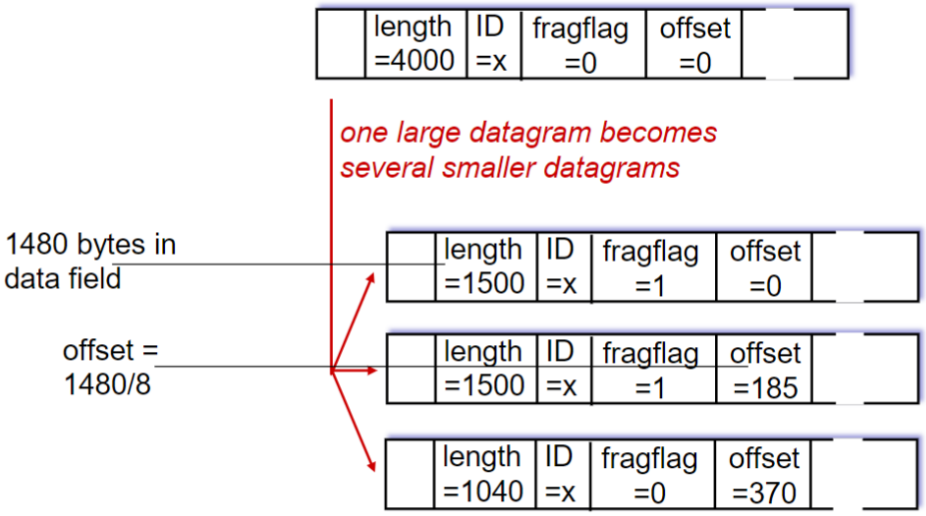
- La soluzione consiste nel dividere il datagram originale in frammenti, ciascuno di dimensione tale da non superare quella massima del payload del frame Ethernet.
- Ogni frammento presenterà la struttura tipica dei datagram, si tenga conto che:
  - l'identificatore è lo stesso per tutti i frammenti del datagram;
  - si segnala per mezzo del campo `fragment offset` il punto di partenza del frammento all'interno del datagram originale;
  - si segnala per mezzo del `fragment bit` se stiamo lavorando con l'ultimo frammento del datagram: 0 in caso affermativo, 1 altrimenti.

  Rimane un problema: la dimensione del fragment offset non ci permette di indicare valori con offset superiore a 2^{13}. Risolviamo dividendo il valore per 8.

Chi fa cosa?

- **Frammentazione del datagram**
  La divisione è attuata dal router. La decisione è correlata al fatto che un pacchetto, nel suo tragitto, potrebbe attraversare aree della rete con protocolli a livello link diversi, e questo potrebbe significare avere MTU diverso (se in un tratto di rete abbiamo MTU ancora più basso si attua un'ulteriore frammentazione).
- **Ricongiungimento dei frammenti**
  Il ricongiungimento è effettuato dall'host destinatario (i frammenti vengono bufferizzati tenendo a mente gli offset, ottenuto l'ultimo frammento si passa il datagram a livello superiore).

Definiamo la dimensione massima del frame Ethernet (**intestazione e Payload messi insieme**, ricordarselo) *Maximum Transmission Unit* (MTU). Supponiamo di avere un datagram di 4000 byte, con MTU = 1500 byte



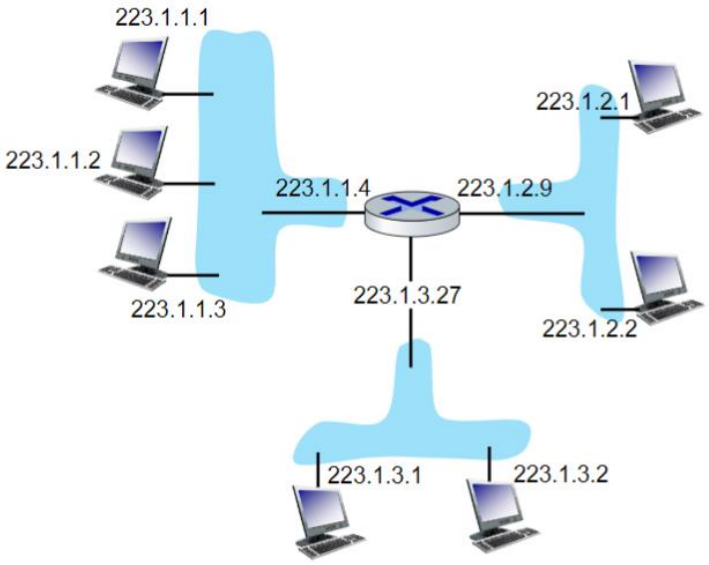
E se non arrivano tutti i frammenti?

- Ipotesi drastica: buttare via tutti i frammenti ricevuti.
- Ipotesi alternativa: retransmission dei frammenti mancanti.
- Si capiscono quali frammenti mancano osservando fragment flag e fragment offset.

7.3.4 IP Addressing

7.3.4.1 Introduzione: cosa identificano, come si leggono e perché sono indirizzi strutturati

In IPv4 l'indirizzo IP è una sequenza di 32 bit atta a individuare host nella rete. Sono rappresentati per mezzo di una notazione decimale puntata.



Un host è connesso alla rete per mezzo di un'apposita interfaccia:

- Se l'host presenta un'unica interfaccia l'indirizzo IP identifica l'host
- Se l'host presenta più interfacce allora l'indirizzo IP identifica l'interfaccia di rete (avremo un indirizzo IP per ogni interfaccia di rete).

Immaginiamoci per comodità che ogni host abbia solo un'interfaccia di rete (quindi un unico indirizzo IP), ma teniamo in considerazione che un router è caratterizzato da molteplici interfacce (una per ingresso/uscita). Ogni indirizzo IP è costituito da due parti:

- parte più significativa (*subnet part*) che identifica la subnet (sottorete) a cui il nodo è collegato;
- parte meno significativa (*host part*), che identifica l'host all'interno della subnet.

Una sottorete (*subnet*) consiste in un insieme di nodi che presentano la stessa *subnet part* dell'indirizzo IP. Possono comunicare tra di loro senza tirare in ballo il router

Si parla di indirizzo strutturato: non solo identifico un nodo in modo univoco, ma ho anche un'idea sulla localizzazione fisica dello stesso (a quale sottorete appartiene). Per avere un'idea si prendano i seguenti documenti di riconoscimento:

- il codice fiscale non è strutturato, in quanto identifico l'individuo ma non ho idea della sua collocazione geografica;
- il numero di telefono è strutturato, identifico univocamente il soggetto e in più capisco dal prefisso telefonico il luogo di chiamata (+39 è chiamata dall'Italia, 050 è prefisso telefonico dell'area del Comune di Pisa)

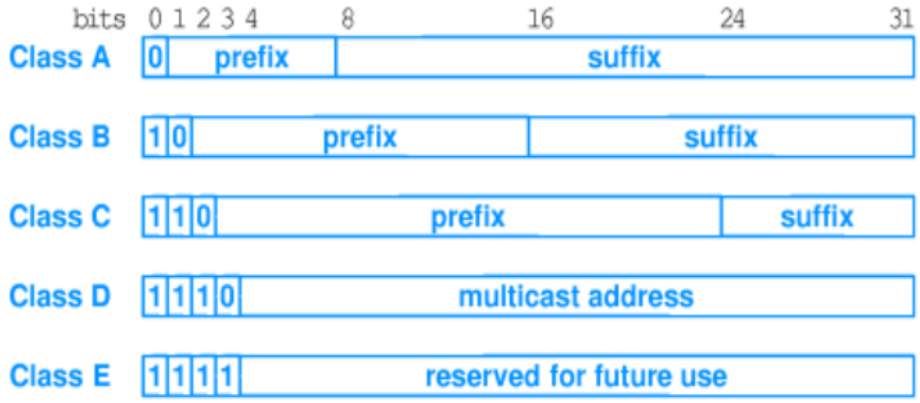
7.3.4.2 Differenze tra indirizzi fisici MAC e indirizzi IP

A questo punto una domanda sorge spontanea: quali differenze si hanno tra indirizzi fisici MAC e indirizzi IP, dato che entrambi sono associati alle schede di rete?

- Gli indirizzi fisici MAC sono sequenze di 48 bit poste in notazione esadecimale (coppie di due numeri) atte all'individuazione della singola scheda di rete in senso globale, l'indirizzo è assegnato permanentemente a una particolare scheda e le 24 cifre più significative permettono di identificare il produttore della scheda stessa. Non sono indirizzi strutturati perché non danno idea della collocazione della rete.
- Gi indirizzi IP sono sequenze di 32 bit poste in notazione decimale puntata (terne di numeri) atte all'individuazione di una scheda di rete (in generale dell'host, solitamente un host ha al più due schede di rete – wireless e wired Ethernet) nel contesto della subnet di appartenenza. L'indirizzo non è permanente (cambia frequentemente, vedremo dopo come). È un indirizzo strutturato (si deduce la subnet di appartenenza dalle cifre più significative).

7.3.4.3 Classificazione degli indirizzi IP in classi (approccio deprecato)

È possibile classificare gli indirizzi IP considerando la dimensione delle due parti (prefix e suffix). La categoria a cui appartiene un indirizzo IP è resa chiara dai bit più significativi:



- In generale si adotta la classe C. Classi B e A sono indirizzi assegnati ad organizzazioni di ingenti dimensioni.

Address Class	Bits In Prefix	Maximum Number of Networks	Bits In Suffix	Maximum Number Of Hosts Per Network
A	7	128	24	16777216
B	14	16384	16	65536
C	21	2097152	8	256

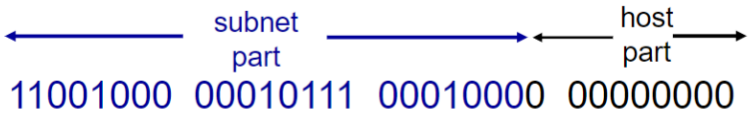
- Supponiamo di avere un'organizzazione con 100 host: quale classe di IP richiediamo?
    - Richiediamo 4/5 indirizzi di classe C?
    - Richiediamo un unico indirizzo di classe B (con tanto margine)?

  Con la seconda proposta abbiamo un'utilizzazione del set di indirizzi IP bassissima: in un contesto con risorse finite non possiamo non assegnare vasti range di indirizzi. Altro risvolto della medaglia: un numero maggiore di sottoreti (quindi prima proposta) richiede un maggior numero di uscite del router, riducendone le prestazioni.

Il problema principale è la rigida divisione in classi, che è stata superata in favore di un approccio classless.

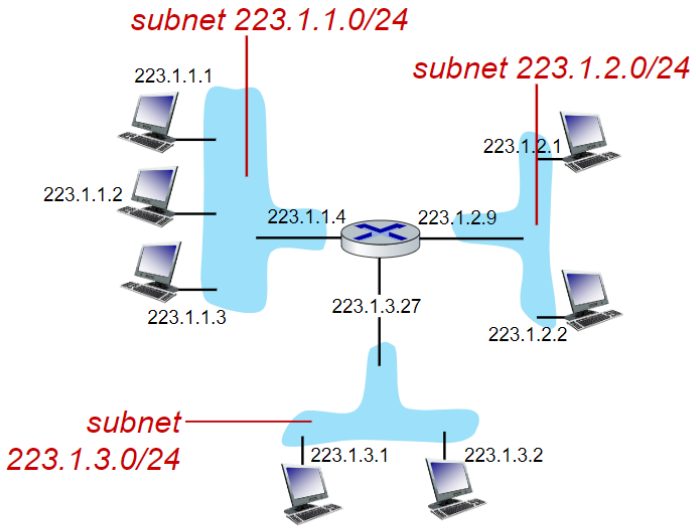
7.3.4.4 Introduzione al Classless InterDomain Routing (CIDR, approccio attuale)

Nell'approccio *Classless InterDomain Routing* (CIDR) gli indirizzi IP continuano ad essere stringhe da 32 bit.



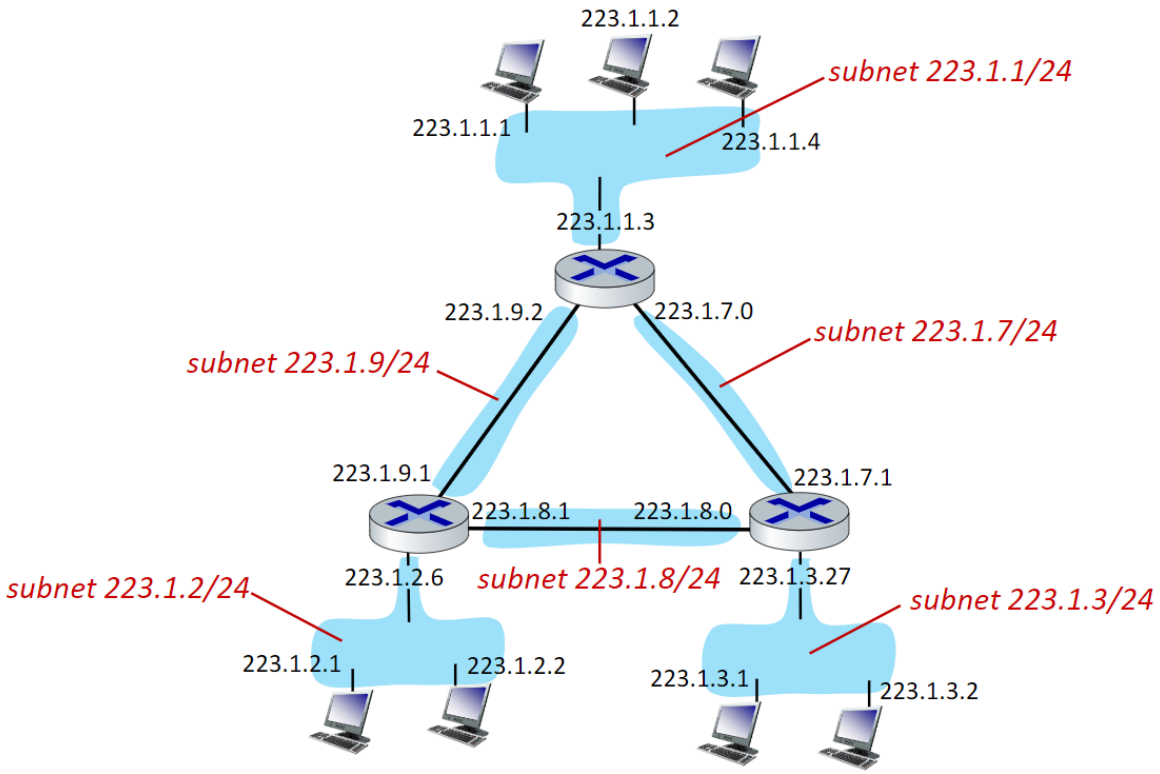
200.23.16.0/23

Si ha la divisione in subnet part ed host part citata all'inizio, ma la dimensione delle due parti non è definita a priori. Abbiamo un approccio elastico dove indichiamo il numero di bit della subnet part. Riprendiamo la foto della pagina precedenti e definiamo le sottoreti indicate in figura:

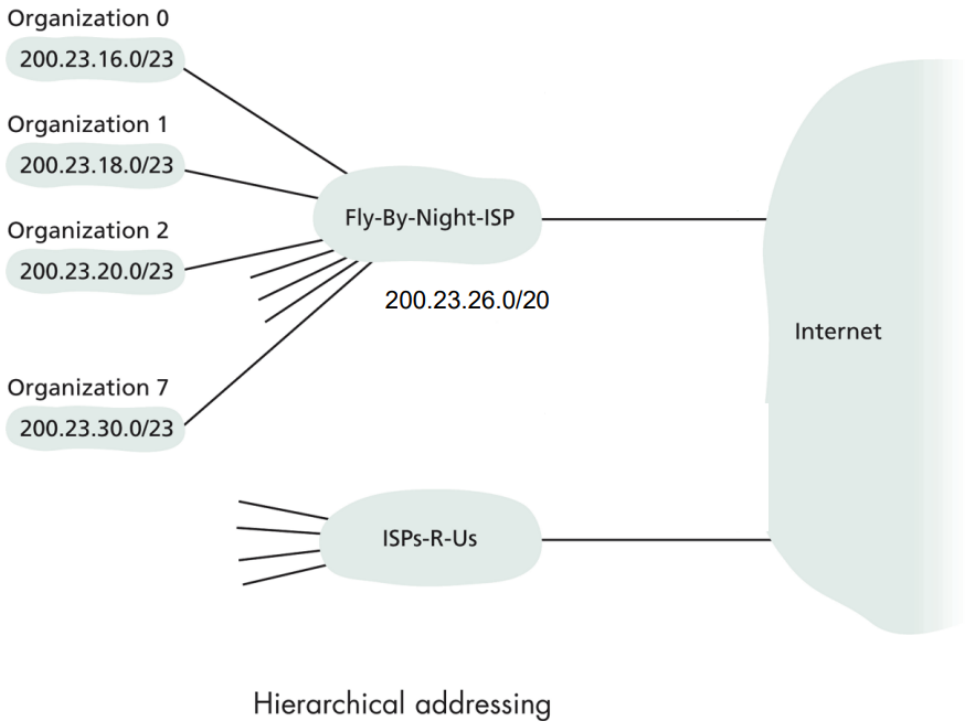


subnet mask: /24
(high-order 24 bits: subnet part of IP address)

Consideriamo anche questo esempio, dove osserviamo che una subnet può essere anche un semplice collegamento tra router: all'interno di queste subnet avremo come unici dispositivi connessi le schede di rete dei due router.



Come si pone l'approccio classless rispetto al forwarding? Supponiamo di trovarci nel seguente contesto e di essere la Fly-By-Night-ISP, che fornisce il servizio di connettività ad organizzazioni che lo chiedono. L'ISP acquisisce a sua volta il servizio di connettività da altri ISP più grandi (che non ci interessano).



Hierarchical addressing

Lo spazio acquisito dall'ISP è costituito da 2^{12} indirizzi (si osservi /20). Idealmente abbiamo supposto che ogni organizzazione abbia richiesto un'ottavo degli indirizzi dell'ISP (sono tutte uguali). Ci chiediamo: i router dell'ISP quante uscite devono avere? Come deve essere compilata la tabella di forwarding?

- Unica voce nella tabella di forwarding, relativa alla sottorete 200.23.26.0/20
- Sufficiente una sola uscita verso la sottorete, non devo avere tante uscite quante le organizzazioni.

Se ci pensate è ciò che succede nella cartellonistica stradale: supponiamo di voler andare in Francia, finchè siamo in Italia abbiamo solo l'indicazione verso la Francia, entrati in Francia avremo le indicazioni verso le principali città francesi dell'area.

7.3.4.5 Indirizzi IP riservati

Esistono una serie di indirizzi IP che non possono essere assegnati, in quanto riservati ad usi particolari.

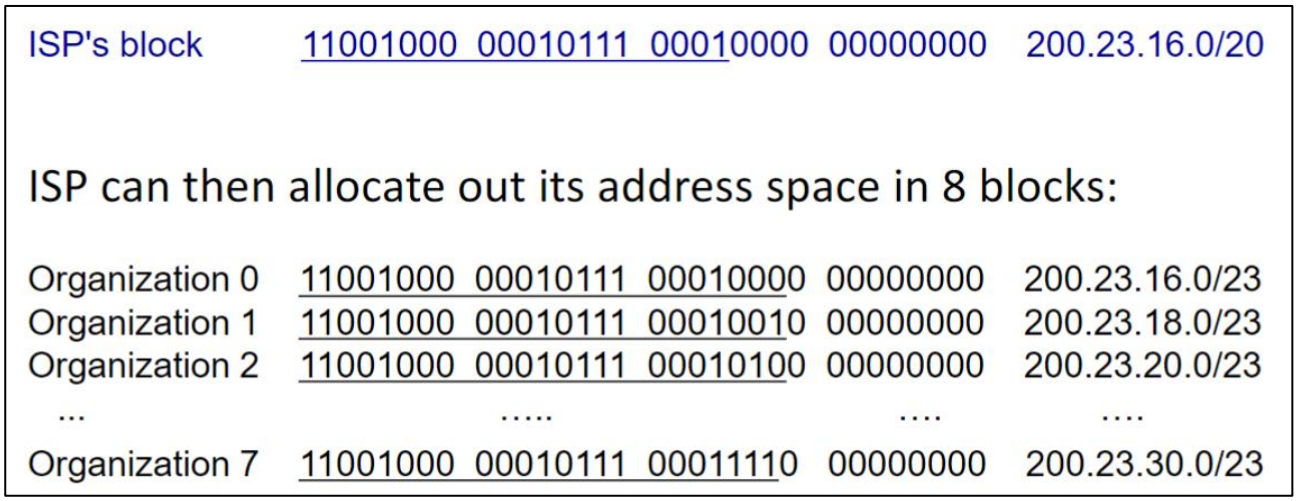
- **Sequenza di tutti zero**.
  Un indirizzo IP di soli zeri è utilizzato in fase di configurazione dagli host che non hanno ancora un indirizzo (si pensi al DHCP).
- **Indirizzi con host number zero**.
  Questi indirizzi identificano una subnet (ricordiamoci che i nodi di una subnet hanno in comune la subnet part).
- **Indirizzi con host number avente bit tutti uguali ad 1**.
  Indirizzo di broadcast di una sottorete. Si usa per recapitare pacchetti a tutti gli host della sottorete.
- **Sequenza di tutti uno.**
  Indirizzo di broadcast ristretto, relativo a tutti i nodi dell'intera rete locale. Un nodo che entra, privo di configurazione, non conosce l'indirizzo della rete.
- **Indirizzi che iniziano per 127 (parte più significativa).**
  Indirizzo di loopback (rammentato dall'ing. Pistolesi). Indirizzo utilizzato durante lo sviluppo di applicazione (è scomodo oltre che poco sicuro essere costretti a sviluppare un'applicazione utilizzando due nodi in rete): il pacchetto viene inviato da un host e restituito allo stesso.

7.3.4.6 Assegnazione degli indirizzi IP

Gli indirizzi IP sono assegnati dagli Internet Service Provider. All'inizio del corso abbiamo visto che vi sono Internet Service Provider di vario livello: questa gerarchizzazione si ripercuote nell'assegnazione degli indirizzi IP.

- ISP di livello alto assegnano certi indirizzi
- ISP di livello inferiore hanno ricevuto indirizzi da ISP di livello maggiore, possono a loro volta assegnare indirizzi IP.

Ovviamente la subnet part si espande negli ISP di livello inferiore: maggiore è la dimensione della subnet part minore è il sottoinsieme di host che stiamo considerando. Condizione fondamentale, in ogni operazione di assegnamento, è il mantenimento dell'univocità degli indirizzi stessi (chi li assegna deve tenere conto di quale set può assegnare).



A livello più alto abbiamo la ICANN (Internet Corporation for Assigned Names and Numbers), che:

- alloca indirizzi IP
- gestisce i DNS;
- delega sulla gestione dei TLD (si pensi all'Italia, registro.it ha ricevuto la delega molti anni fa proprio dall'ICANN).

È vero che la rete è decentralizzata, ma serve un minimo di governance. Si distinguano inoltre due tipologie di assegnamento di indirizzo:

- **indirizzi IP permanenti**, solitamente stabiliti all'interno di file di configurazione (una volta erano gli indirizzi maggioritari, quando la maggior parte dei dispositivi connessi rimanevano sempre accesi e sempre connessi alla rete)
- **indirizzi IP temporanei**, stabiliti in modo dinamico attraverso il DHCP (Dynamic Host Configuration Protocol, indirizzi maggioritari – si pensi che oggi la stragrande maggioranza dei dispositivi non è sempre connessa alla rete, ad esempio un PC portatile spostato frequentemente).

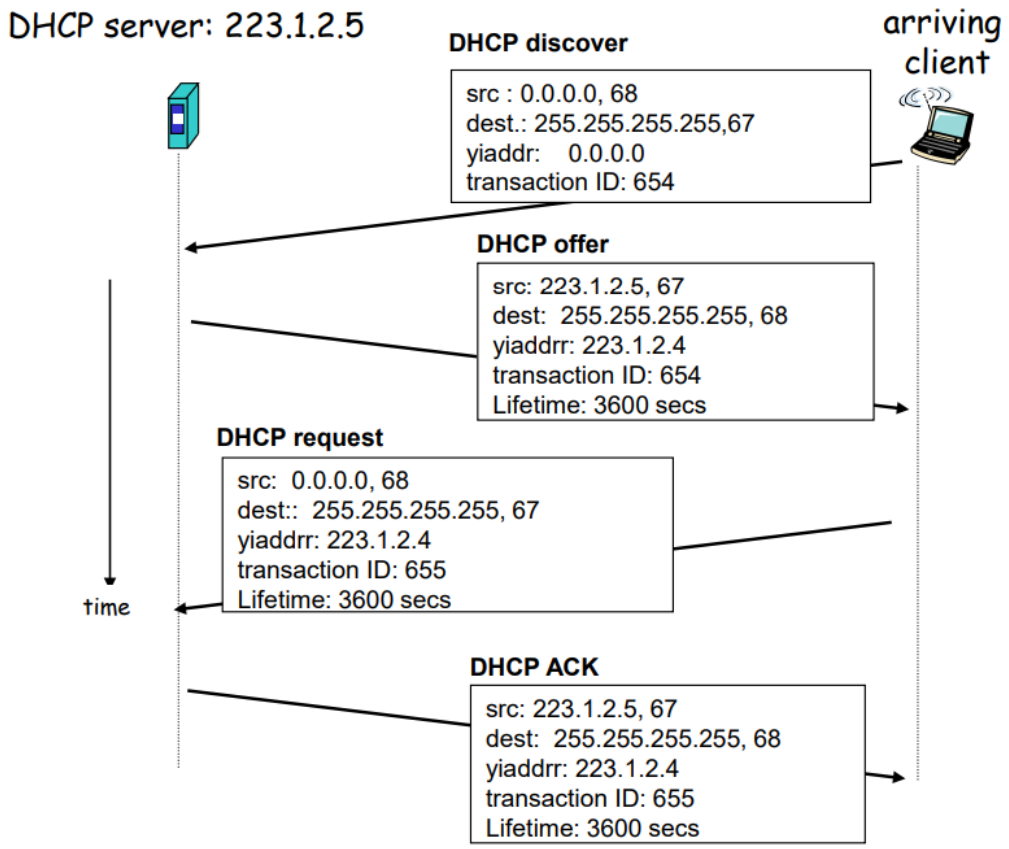
7.3.4.7 Dynamic Host Configuration Protocol (DHCP)

Il DHCP consiste in un'applicazione client-server atta all'assegnazione di indirizzi IP temporanei:

- abbiamo un server che assegna indirizzi, e
- il client che si rivolge al primo per ricevere una proposta di configurazione.

L'applicazione è di tipo *plug-and-play*: è trasparente, il dispositivo collegato non si accorge della sua presenza e non deve fare niente (auto-configurazione).

Per avere una spiegazione completa si rimanda alle lezioni di laboratorio dell'ing. Pistolesi (molto chiare), limitandoci ad alcune riflessioni sull'interlocuzione tra client e server.



Come mai il client comunica ponendo un indirizzo sorgente tutto nullo?
Il client inizialmente non ha un indirizzo proprio, ricorre a uno degli indirizzi riservati. Trasmette così finchè non avrà assegnato un proprio indirizzo (dopo l'ACK).

E l'indirizzo destinatario?
È l'indirizzo broadcast: il client comunica con tutti i dispositivi in rete, inclusi i server DHCP (il dispositivo appena connesso alla rete non può conoscere gli indirizzi dei server DHCP, abbiamo detto che si ha plug and play – autoconfigurazione).

Come fanno il server a sapere che un particolare pacchetto è relativo a una richiesta, in assenza di indirizzo IP src? Come fa il client a sapere che un pacchetto ricevuto è risposta alla sua richiesta?
Dal *transaction ID*.

Come mai il client trasmette il DHCP request in broadcast, pur conoscendo l'indirizzo IP del server DHCP (posto nell'intestazione del DHCP offer)?
Perché si deve tenere a mente che una rete potrebbe presentare al suo interno più DHCP server: si ha un contesto competitivo dove ogni server DHCP presenta una sua proposta. Il client deve fare due cose:

- comunicare al server DHCP incriminato che accetta la sua proposta di configurazione;
- comunicare ai rimanenti server DHCP che non ha accettato la loro proposta di configurazione.

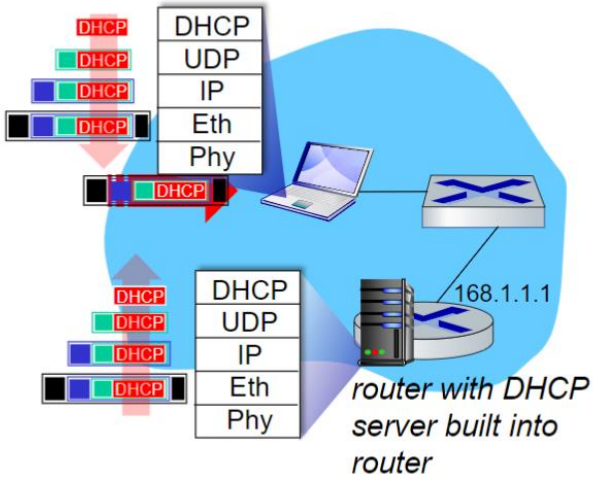
È sempre necessaria la trasmissione dei pacchetti DHCP discover e DHCP offer?
No, il client può inviare diretto un DHCP request se vuole confermare una configurazione precedente.

Cosa offre il DHCP server?
Offre l'indirizzo IP, ma anche un'altra serie di parametri di configurazione.

7.3.4.8 Attenzione al router DHCP: eccezione rispetto alla regola

Il server DHCP può essere implementato in router, che definiremo router DHCP. Facciamo questa breve osservazione in quanto un Router DHCP risulta implementare ulteriori livelli della pila protocollare:

- Il classico router, come già visto, implementa Livello fisico, Livello Datalink e Livello Network
- Il router DHCP implementa anche Livello trasporto (UDP) e Livello applicazione (DHCP).

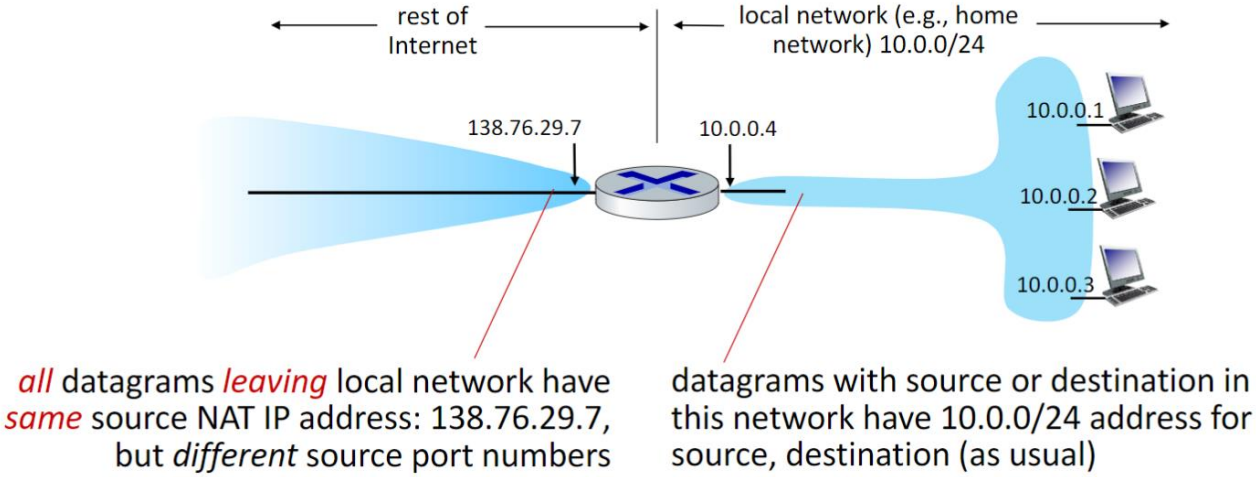


7.3.4.9 Network Address Translation (NAT)

Ci siamo già soffermati sul fatto che gli indirizzi IP non sono infiniti, e che è necessario assegnarli in modo efficiente. Il *Network Address Translation* (nato per IPv4, prima dell'avvento di IPv6) permette un uso efficienti degli indirizzi IP.

- Un solo indirizzo permette di identificare più host, in violazione del principio di univocità.
- **Si introduce la distinzione tra indirizzi IP pubblici e indirizzi IP privati**. Un indirizzo privato è un indirizzo che ha significato solo all'interno di una particolare rete.
- L'univocità degli indirizzi privati viene gestita dal relativo ISP della rete locale, che può muoversi senza notificare il mondo esterno. L'univocità viene garantita rispetto alla rete locale, non rispetto alla rete intera (come nel caso degli indirizzi pubblici).
- Un host che ha indirizzo privato non può connettersi alla rete pubblica: se si prova a fare ciò il router si rifiuta di inoltrare i pacchetti verso la rete pubblica.

Occhio all'ultimo punto: è vero che gli indirizzi privati non possono essere utilizzati, ma è anche vero che questi dispositivi hanno necessità di comunicare con Internet tutto. Si risolve introducendo un router NAT, che si pone come intermediario lungo il percorso tra un host con indirizzo privato e un host su rete globale.

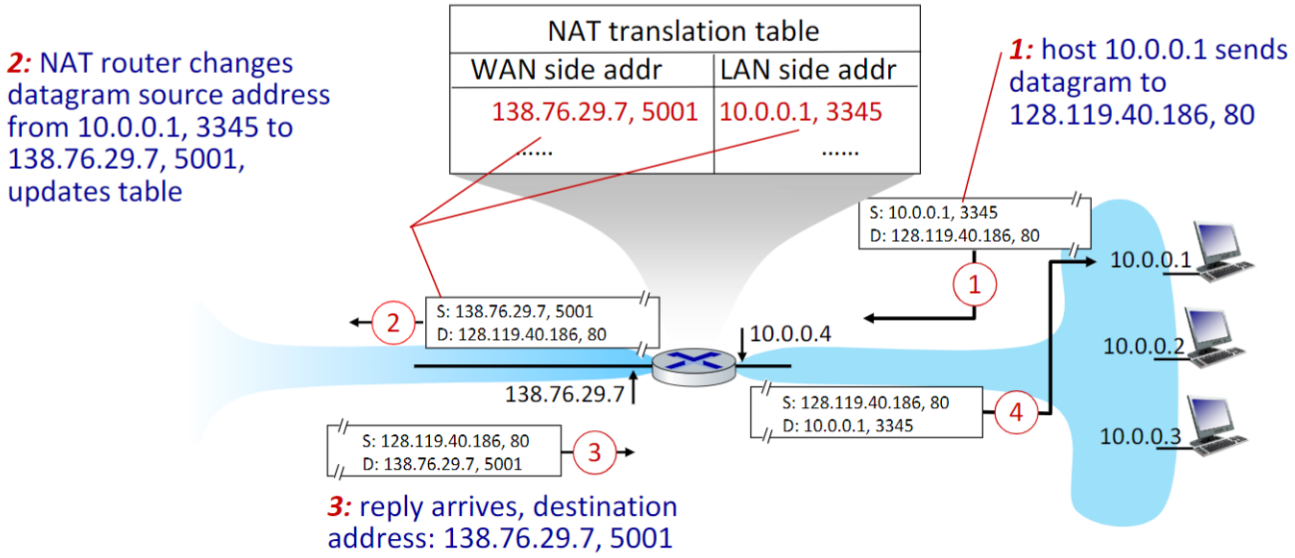


- Abbiamo tre host nella rete locale, identificati rispettivamente con gli indirizzi IP 10.0.0.1, 10.0.0.2, 10.0.0.3. Questi sono indirizzi privati, che non sono interpretabili nella rete Internet globale, e che hanno significato diverso in altre reti locali.

- Il router NAT presenta due indirizzi IP:
    - Un indirizzo IP pubblico, 138.76.29.7
    - Un indirizzo IP privato, 10.0.0.4.

  L'indirizzo privato è utilizzato per identificare il router all'interno della rete locale, l'indirizzo pubblico è utilizzato da host esterni per comunicare con gli host presenti nella rete locale. Qualunque host esterno che desidera trasmettere un pacchetto a un host interno indica come indirizzo IP destinatario quello del router NAT.

Problema risolto? Non proprio. Se ci limitiamo a questo è come spedire una lettera all'indirizzo di un condominio senza indicare quale soggetto all'interno del condominio è il destinatario. Si introduce una *NAT Translation Table* dove si considerano anche i numeri di porta, sia degli host locali che degli host esterni alla rete locale.



- Il numero di porta è un campo di 16 bit: questo significa che lato WAN il router è in grado di supportare 60.000 connessioni simultanee!
- Si osservino i pacchetti in figura. Il router interviene sulle informazioni del pacchetto:
    - nel caso di pacchetti uscenti dalla rete locale altera il sorgente (privato) ponendo l'indirizzo IP del router, più la porta indicata nella NAT Translation Table;
    - nel caso di pacchetti entrati altera il destinatario (indirizzo IP e porta del router) ponendo Indirizzo IP privato e porta ascoltata dal destinatario.

  L'operazione è trasparente: i due interloquiscono senza avere minima idea del passaggio compiuto dal router.

NAT è risultato controverso per una serie di motivi:

- si violano i principi alla base della pila protocollare, in particolare vengono alterate informazioni relative ad altri livelli (indirizzo IP e porta, relative a livelli superiori) e si usano singoli indirizzi IP per rappresentare più host (cambiando solo il numero di porta)
- si viola il principio della connessione end-to-end, nel caso di una connessione TCP gli host sono convinti della presenza di una pipe, cosa non più vera a causa del router intermedio (che altera il contenuto del pacchetto).

8 TRASPORTO: PROTOCOLLI TCP E UDP

8.1 Oggetto del capitolo

Pila protocollare
Appplication *messages*
Transport *segments*
Network *datagrams*
Link / Datalink *frames*
Physical

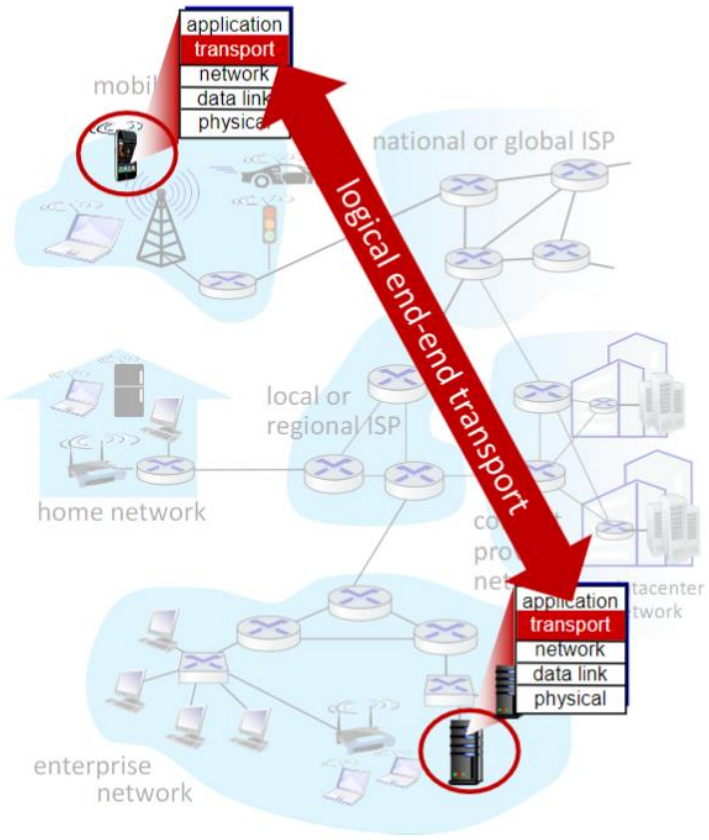
Abbiamo visto nei capitoli precedenti il protocollo IP che trasporta i datagram a livello Network: permette un passaggio logico da livello Network del mittente a livello Network del destinatario (collegamento host-host). La cosa è in un certo senso equivalente al corriere che porta le lettere dalla portineria di un edificio alla portineria di un altro edificio.

Ciò non è sufficiente: noi non facciamo semplicemente un trasferimento da host a host, ma trasferimento dal processo di un host al processo di un altro host. Nelle spiegazioni fatte fino ad ora manca il passaggio finale, quello che permette il trasporto del messaggio dalla portineria dell'edificio all'ufficio del destinatario (la cassetta delle lettere dell'ufficio del destinatario è il socket).

Vedremo le proprietà di questo livello della pila protocollare e spiegheremo nel dettaglio i protocolli di trasporto introdotti all'inizio del corso: TCP e UDP!

8.2 Caratteristiche del livello di trasporto

Nel livello di trasporto si ha un collegamento logico tra processi, in contrasto rispetto al livello Network dove si ha un collegamento logico tra hosts (si tenga a mente l'ovvio, cioè che su un host girano più processi).



Il livello di trasporto offre le seguenti caratteristiche:

- multiplexing/demultiplexing dei pacchetti;
- Reliable Data Transfer;
- controllo sul flusso (flow control);
- controllo sulla congestione del traffico (Congestion control).

8.3 Multiplexing e demultiplexing

La funzionalità di *multiplexing e demultiplexing* è la caratteristica principale del livello di trasporto. Concentriamoci sul demultiplexing.

Siamo nel destinatario e si ricevono datagram dal livello inferiore (il livello Network). Quanto arriva a livello trasporto costituisce un unico flusso che dovra essere "demultiplexato" verso i vari processi destinatari. Quali valori utilizziamo per determinare verso quale processo indirizzare il pacchetto?

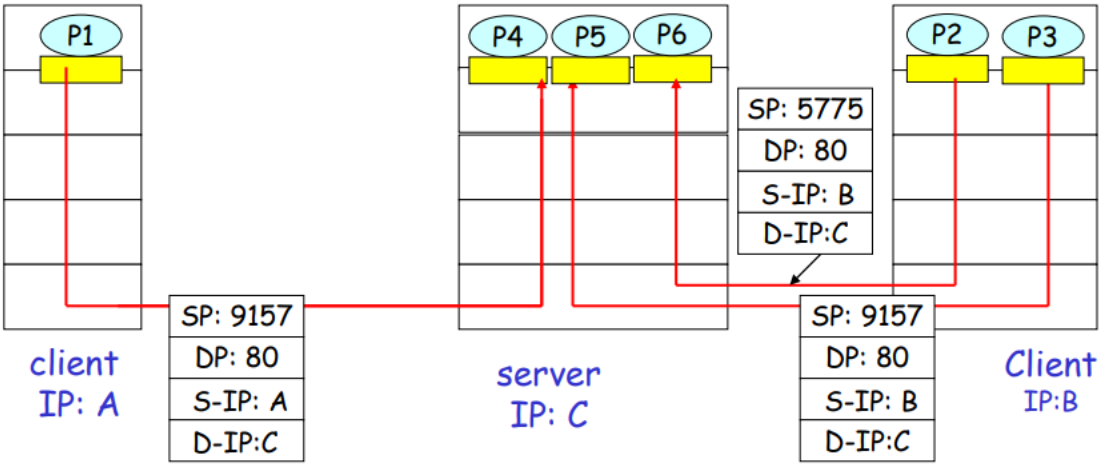
- **Servizio di tipo datagram** (UDP).
  Il socket è identificato dalla coppia `<Indirizzo IP; porta>` relativi all'host dove si trova il socket. Non si ha connessione stabilite a priori con altri socket.
- **Servizio di tipo stream** (TCP).
  Contrariamente a prima il socket non è più identificato dalla coppia detta, ma da quattro valori che tengono conto del socket dell'host mittente e il socket dell'host destinatario.
  `<Indirizzo IP sorgente, porta sorgente, Indirizzo IP dest; porta dest>`

Come mai si fa questa distinzione? Consideriamo un esempio dove abbiamo:

- Un host con indirizzo IP "A" e un host con indirizzo IP "B" che svolgono il ruolo di client
- Un host con indirizzo IP "C" che svolge il ruolo di server.

In ciascun host sono presenti dei processi (P1,P2,P3,P4,P5,P6) che interagiscono tra di loro trasmettendo pacchetti in rete. Consideriamo i valori detti prima:

- Porta del mittente (SP)
- Indirizzo IP del mittente (S-IP)
- Porta del destinatario (DP)
- Indirizzo IP del destinatario (D-IP).



Si osservi che tutti i processi nei client si rivolgono al server indicando come indirizzo IP "C" e come porta il numero 80: tutti!!! Eppure, vediamo dalla fotografia che:

- P1 comunica con P4
- P2 comunica con P6
- P3 comunica con P5

Collegamenti tra processi (e quindi tra socket) sono stabiliti in applicazione del protocollo TCP (che ha una call setup), mentre non esistono nel protocollo UDP. Segue che nel protocollo TCP, per evitare ambiguità, è necessario controllare anche indirizzo IP e porta del mittente:

- leggendo indirizzo IP e porta di P1 capisco che il pacchetto deve essere recapitato al processo P4;
- leggendo indirizzo IP e porta di P2 capisco che il pacchetto deve essere recapitato al processo P6;
- leggendo indirizzo IP e porta di P3 capisco che il pacchetto deve essere recapitato al processo P5.

8.4 User Datagram Protocol (UDP)

8.4.1 Caratteristiche

C'è poco da dire sul protocollo UDP, dato che l'unica caratteristica sostanziale offerta è il multiplexing/demultiplexing.

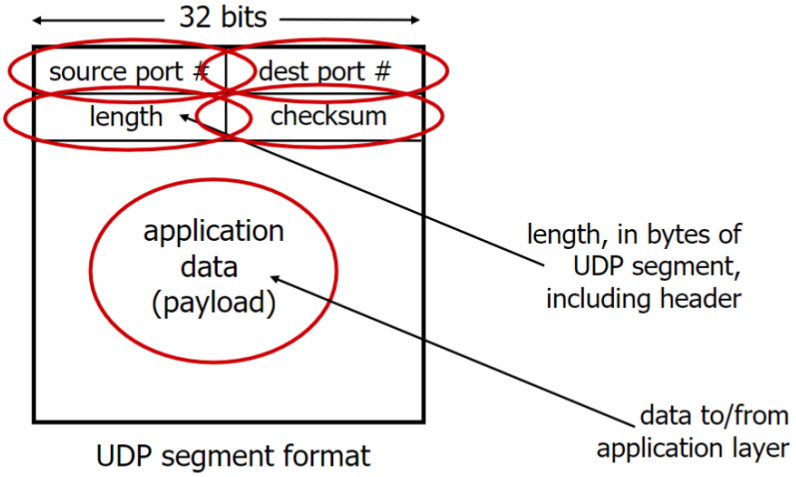
- Ribadiamo che è protocollo semplice (no frills), basato su approccio best effort: questo significa che i pacchetti possono non arrivare a destinazione, oppure arrivare troppo tardi.
- È utile perché l'assenza della call setup e di altri meccanismi di controllo (tipici del TCP) riduce fortemente i tempi (ogni cosa in più introduce un minimo di ritardo).
- La dimensione del *segment header* è piccola.

È chiaro che UDP è il protocollo più adatto quando abbiamo l'esigenza impellente di trasmettere pacchetti a rate elevato.

8.4.2 UDP Segment format

L'UDP segment format indica per l'header:

- numero di porta sorgente;
- numero di porta destinatario;
- lunghezza del segmento (incluso l'header);
- *checksum* (per controllo su errori nel pacchetto – si divide il pacchetto in sequenze da 16 bit, che si sommano facendo complemento a 1 – come già spiegato in passato).



Si osservi che tra i campi non è presente l'indirizzo IP del destinatario:

- il campo con l'indirizzo IP destinatario è presente nell'intestazione del datagram a livello Network;
- quando il pacchetto si trova a livello Transport (dopo aver attraversato dal basso livello fisico, livello Link e livello Network) questo è già arrivato all'host destinatario, ergo l'indirizzo IP è un valore non più necessario (dobbiamo solo consegnare il pacchetto al relativo processo destinatario).

8.5 Transfer Control Protocol (TCP)

8.5.1 Caratteristiche

Sul protocollo TCP è necessario spendere qualche parola in più.

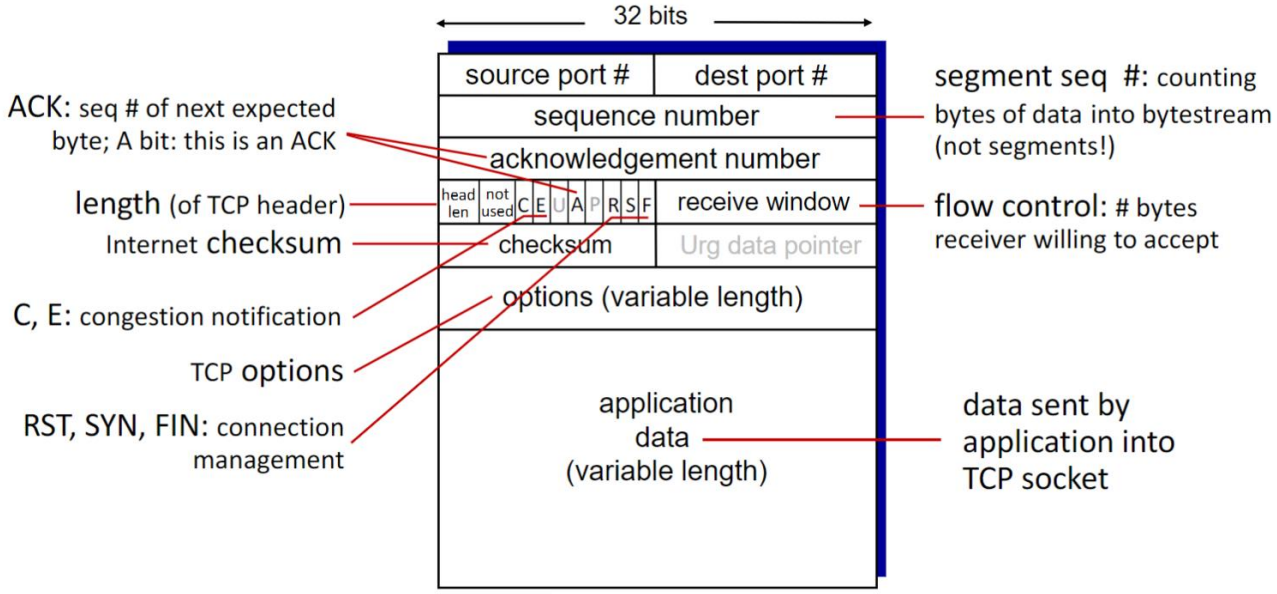
- ***connection-oriented***
  Il servizio offerto è connection-oriented: necessario stabilire una connessione per mezzo di una call setup (si parla di handshaking), stessa cosa quando si vuole chiudere la connessione.
- ***point-to-point e pipe***
  Non si ha un circuito virtuale propriamente parlando: i dispositivi intermedi (network core) non mantengono informazioni sullo stato del circuito. Nei fatti si stabilisce una connessione point-to-point con mittente e destinatario, una pipe che collega i due host (flusso di dati, ecco perché si parla di stream).

- ***Flusso full duplex e bufferizzazione***
  Il flusso è bidirezionale: il mittente scrivere al destinatario ma successivamente il destinatario diventa mittente. Presente un buffer di invio e un buffer di ricezione in tutti gli host.
- **Controllo di flusso**
  Previsti controlli sul flusso, in modo tale che il mittente non invii un numero di pacchetti troppo elevati al destinatario (controlli con cui si interviene sui diversi rate di trasmissione dei nodi, si deve considerare i casi in cui gli host hanno velocità diverse). Se non si gestisce la cosa il buffer di ricezione viene riempito e si ha perdita di pacchetti.
- **Affidabilità**
  Col protocollo TCP abbiamo garanzia sull'arrivo a destinazione del pacchetto.
- **ACK Cumulativi.**
  Il protocollo TCP ricorre ad ACK cumulativi.

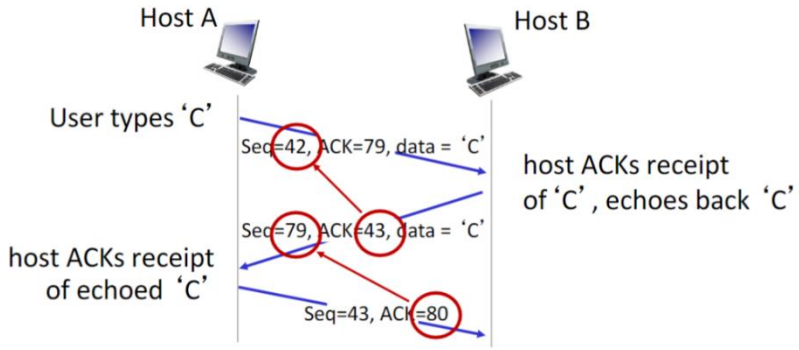
8.5.2 TCP segment structure

Consideriamo, come prima, l'header del segmento. Abbiamo:

- porta del mittente e porta del destinatario (nulla di nuovo);
- *checksum* (nulla di nuovo)
- *sequence number* (parola da 32 bit utilizzata per individuare pacchetti fuori ordine, il dispositivo tiene a mente un numero di sequenza atteso e se non coincide col numero allora si bufferizza). Il numero non è orientato al segmento, ma ai byte!
- *acknowledgement number*, con cui si indica quali sequenze sottoporre ad acknowledge (parola utilizzata solo se si segnala l'ACK col relativo flag).
- una serie di flag nella quarta riga:
  - flag A che segnala se il pacchetto rappresenta un acknowledge oppure no;
  - flag RST, SYN e FIN sono bit utilizzati durante l'apertura e la chiusura della connessione;
  - flag U indica se i dati sono urgenti (quasi mai usato).
  - Flag P non ci interessa, mai usato.
- *receive window* segnala lo spazio disponibile nel buffer di ricezione del mittente.
- *urg data pointer* è puntatore ai dati urgenti presenti nel payload (si usa col flag relativo, quindi anche questo campo è sostanzialmente inutilizzato).
- *options* è area di memoria per i campi opzionali.



Sull'*acknowledgement number* è necessario fare una riflessione: il valore memorizzato non consiste nel numero di sequenza relativo all'ultimo byte inviato, ma nel numero di sequenza che l'host si aspetta per il prossimo pacchetto inviato.



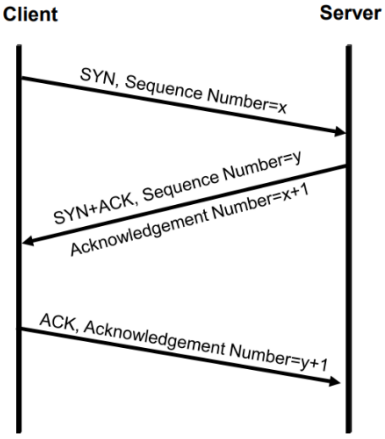
8.5.3 TCP Connection Management

8.5.3.1 Scopo

La fase di apertura della connessione permette l'allocazione delle strutture dati (sia nel sender che nel receiver) che saranno utilizzate per gestire la connessione (variabili di stato come contatori di sequenza, inizializzazione dei buffer, …).

- Nella fase di inizializzazione si avrà la creazione di un socket da parte del client.
  `res = connect(sd, …)`
- Si avrà una richiesta di connessione rivolta a un socket creato in precedenza dal server.
  `conn_sd = accept(sd, …)`

8.5.3.2 Triplice handshake

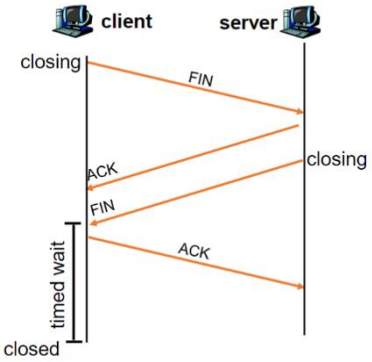


La connessione viene stabilita per mezzo di un triplo handshake.

- Il primo passo è svolto dal client, dove viene lanciata la funzione connect. Si trasmette un segmento con flag SYN, accompagnato da un numero di sequenza casuale x.
- Il server risponde trasmettendo un segmento con flag SYN e ACK, accompagnato da un numero di sequenza casuale y e un acknowledgement number uguale a $x + 1$ (per la regola precedentemente spiegata)
- Il client, ricevuta la risposta, trasmette un segmento con flag ACK e Acnowledgment number uguale a $y + 1$. Questo segmento potrebbe contenere dati nel payload (ad esempio il contenuto della richiesta nel caso del protocollo HTTP).

Perché sequenze casuali? In caso di connessioni TCP consecutive potrebbe succedere che pacchetti relativi alla vecchia connessione potrebbero essere confusi per pacchetti della nuova connessione. Si pongono sequenze casuali in modo tale che i numeri di sequenza successivi siano drasticamente diversi rispetto a quelli di una eventuale connessione precedente.

8.5.3.3 Chiusura della connessione



Come viene gestita la chiusura della connessione?

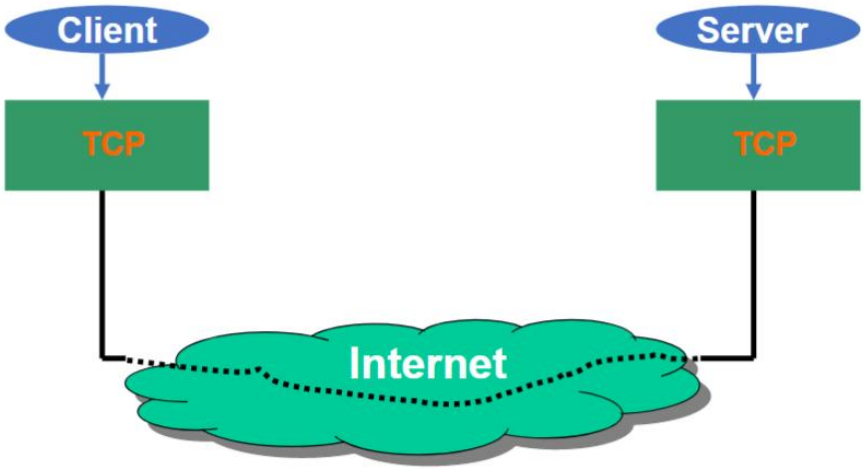
- Il client, colui che ha avviato la connessione, invia un segmento con flag FIN per segnalare al server la volontà di chiudere la connessione.
- Il server risponde inviando un ACK sul segmento precedente. Dopo un po' di tempo, cioè quando sa di poter chiudere la connessione, invia un segmento con flag FIN.
- Il client, ricevuto il segmento FIN, invia un ACK. Si ha l'inizio di un periodo di chiusura (*timed wait*) durante il quale il client gestisce le eventuali ultime trasmissioni di pacchetti prima della chiusura (incluso nuovi FIN nel caso in cui qualcosa vada storto). Superato questo periodo la connessione TCP è effettivamente chiusa.

Cosa succede se il FIN inviato dal server non viene ricevuto dal client? Cosa succede se l'ACK finale inviato dal server non arriva al server? Viene inviato nuovamente il FIN, chiaramente qualcosa è andato storto.

8.5.4 Reliable Data Transfer

8.5.4.1 retransmission: necessità di stimare RTT per il timeout

Abbiamo detto che il protocollo TCP è un protocollo affidabile: questo significa che tutti i dati arrivano a destinazione integri e nello stesso ordine di invio.



Il protocollo adottato è di tipo *Window-based ARQ* (pipeline), cioè basato su acknowledgement e retransmission. Se introduciamo la retransmission la prima cosa che ci poniamo è la seguente: come impostiamo il timeout?

- La prima questione è che non possiamo calcolare il timeout pensando a un collegamento point-to-point, cioè calcolare l'RTT. Il collegamento punto-punto è solo virtuale, ergo si introduce il ritardo di accodamento che non è calcolabile a priori. Determineremo il timeout basandoci su stime del RTT.
- Un'idea iniziale per la stima del RTT potrebbe essere utilizzare il RTT relativo all'ultimo pacchetto spedito: non va bene perché RTT può oscillare pesantemente (come si vede nel grafico più avanti).
- L'idea definitiva è quella di calcolare RTT basandoci su una media pesata di tutte le stime fatte con l'invio di precedenti pacchetti. Pesiamo i valori in funzione della loro collocazione temporale: più la trasmissione è lontana nel tempo, minore è il peso del termine nella media.

Si propongono le seguenti formule, dove il pesco dei pacchetti più vecchi decresce esponenzialmente e in funzione di valore $\alpha < 1$

$$ERTT_1 = RTT_0$$
$$ERTT_2 = \alpha \cdot RTT_1 + (1-\alpha) \cdot RTT_0$$
$$ERTT_3 = \alpha \cdot RTT_2 + \alpha(1-\alpha) \cdot RTT_1 + (1-\alpha)^2 \cdot RTT_0$$
$$\dots$$
$$ERTT_{n+1} = \alpha \cdot RTT_n + \alpha(1-\alpha) \cdot RTT_{n-1} + \alpha(1-\alpha)^2 \cdot RTT_{n-2} + \dots + (1-\alpha)^n \cdot RTT_0$$

Dove RTT è un Round Trip Time misurato (quindi reale), mentre ERTT è un Round Trip Time stimato (basato sugli RTT precedentemente misurati).

Si osservi che:

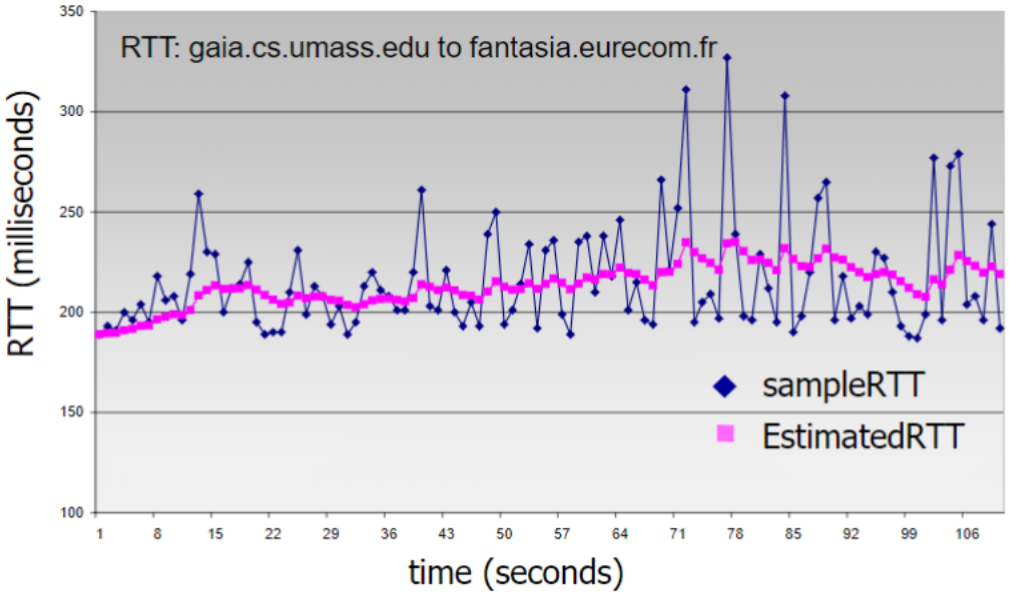
- $\alpha \to 0$: si privilegia la storia passata a discapito di quella recente;
- $\alpha \to 1$: si privilegia la storia recente a discapito di quella passata.

Il valore tipico adottato è $\alpha = 0.125$, valore molto vicino a zero (quindi storia passata abbastanza pesata). Poniamo la formula precedente, avente forma iterativa, in una forma ricorsiva. Si raccolga $1-\alpha$ dal secondo termine in poi:

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

La formula individuata è equivalente alla precedente: è molto più compatta, ma soprattutto ci piace di più perché non abbiamo bisogno di memorizzare tutte le misurazioni precedenti del RTT: ci basta solo l'ultimo ERTT.

Poniamo a confronto l'andamento di ERTT con l'andamento di RTT.



Si osservi come la formula introdotta introduca quello che in inglese è noto come *smoothing* della curva: molto più morbida, cambia molto più lentamente (le fluttuazioni sono assorbite).

8.5.4.2 retrasmission: usare ERTT per determinare un timeout

Capito come stimare RTT ci manca il passaggio finale con cui determinare il timeout a partire da RTT stimato. Nella scelta di un adeguato timeout è importante ricordare che:

- un tempo di attesa troppo piccolo comporta retransmission di pacchetti inutili;
- un tempo di attesa troppo grande aumenta notevolmente i tempi di attesa.

La scelta del timeout deve considerare questo, nell'ottica che un timeout non è in modo assoluto un tempo di attesa troppo grande o troppo piccolo (in alcune circostanze è troppo grande, in altre troppo piccolo, in altre ancora adeguato).

- **Algoritmo di Karn-Partridge**
  L'algoritmo prevede come *timeout* l'ERTT moltiplicato per due.
  $$TimeoutInterval = 2 \cdot EstimatedRTT$$
  Algoritmo un po' euristico (cit, perché 2? È l'abitudine di noi Ingegneri a scegliere valori arbitrari finchè il risultato non ci risulta soddisfacente). L'algoritmo tenta anche di risolvere un'ambiguità dovuta alla retransmission: l'ACK fa riferimento alla prima o a una successiva retransmission? Si risolve escludendo dalla stima di RTT i pacchetti ritrasmessi.

- **Algoritmo di Van Jacobson – Karel**
  L'algoritmo è un'espansione del precedente, che presenta una problematica: giusto stimare RTT, ma in presenza di oscillazioni rilevanti il timeout può non essere adeguato.
  - Valore troppo piccolo moltiplicato due volte è valore ancora troppo piccolo.
  - Valore troppo grande moltiplicato due volte è valore ancora più grande.

  Si risolve introducendo una stima della deviazione del valore stimato rispetto al valore misurato.
  Con lo stesso approccio adottato per calcolare ERTT (EstimatedRTT) poniamo
  $$DevRTT = (1-\beta) \cdot DevRTT + \beta \cdot |SampleRTT - EstimatedRTT|$$
  $$TimeoutInterval = EstimatedRTT + 4 \cdot DevRTT$$
  Dove il valore tipico di β è $\beta = 0.25$. Stimo l'oscillazione e la sommo moltiplicata per quattro al RTT stimato: il risultato è il timeout.

8.5.4.3 TCP sender

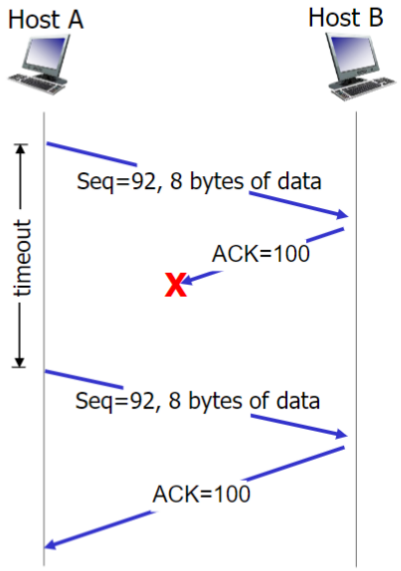
Il TCP usa un singolo timer di retransmission. Si ha retransmission ogni volta che scatta il timeout, oppure in presenza di ACK duplicati (si tenga a mente il go-back-n, non si possono trasmettere ACK su pacchetti successivi fino a quando il pacchetto incriminato non riceverà ACK – quindi si continua a trasmettere quello). Iniziamo a spiegare il comportamento del sender introducendo una versione semplificata dove:

- si ignorano ACK duplicati;
- non si svolge flow control e congestion control.

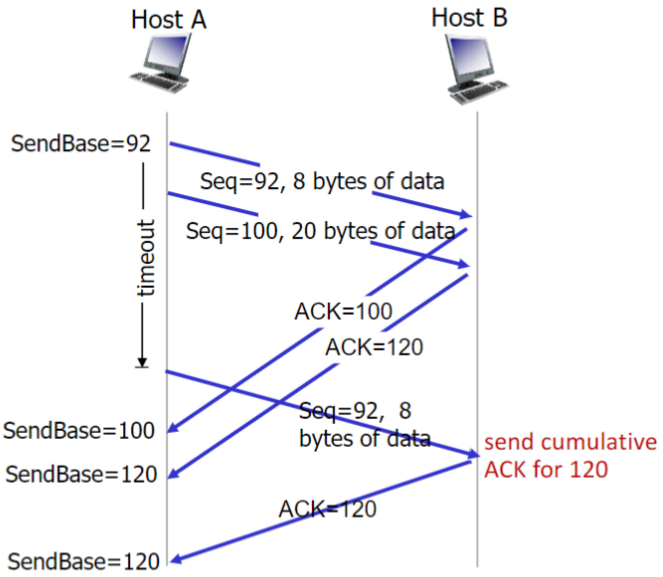
```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

loop (forever) {
  switch(event)

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

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

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

} /* end of loop forever */
```

- Per quanto riguarda l'interpretazione della variabile *SendBase* si invita a ricordare quanto già detto sull'acknowledge number. In questo contesto SendBase il primo pacchetto tra quelli che non hanno ancora ricevuto un ACK cumulativo.
- Supponiamo che il *sender* riceva un messaggio dal livello applicativo superiore.
    - Il protocollo TCP prepara un segmento: nel relativo payload sarà posto il messaggio trasmesso dal livello superiore. Assegna al nuovo segmento un numero di sequenza opportuno (ricordarsi che è stato inizializzato un contatore di numeri di sequenza NextSeqNum durante l'apertura della connessione TCP).
    - Si attiva il timer se questo non è ancora attivo (se stiamo trasmettendo il primo segmento allora il timer non è attivo). Il timeout viene determinato nei modi spiegati precedentemente.
    - Il segmento viene passato al livello IP inferiore.
    - Si conclude aumentando NextSeqNum in modo adeguato (in funzione della dimensione del segmento trasmesso, ricordarsi cosa abbiamo detto sugli identificativi delle sequenze)
- Supponiamo che vi sia *timeout*.
    - Si ritrasmette il segmento che ha causato il timeout.
    - Si riavvia, ovviamente, il timer (senza alterare il segmento di riferimento)
- Supponiamo che il server riceva un ACK.
    - Si verifica se vi sono segmenti che precedentemente non sono stati sottoposti ad ACK.
    - Nel caso si aumenta SendBase. Se ci sono altri segmenti che non hanno ancora ricevuto ACK si fa ripartire il timer (alterando il segmento di riferimento, quello più vecchio tra i presenti).

Consideriamo gli esempi nella pagina

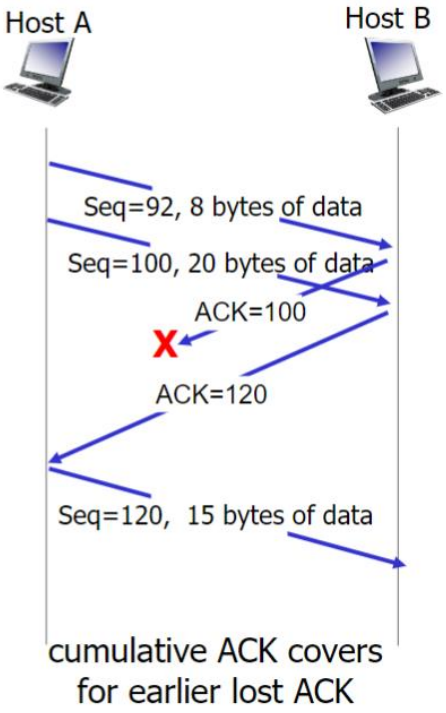


lost ACK scenario



premature timeout

- **Perdita di ACK**.
  - Abbiamo Host A che trasmette una sequenza di 8 byte, con identificativo 92, all'Host B.
  - L'Host B riceve la sequenza e trasmette un ACK con numero di sequenza 100 (92 + 8): questo ACK, purtroppo, non arriverà all'Host A.
  - Superato il timeout l'Host A trasmette nuovamente la sequenza. L'Host B, ricevuta nuovamente la stessa sequenza, trasmette nuovamente l'ACK.



cumulative ACK covers for earlier lost ACK

- **Timeout prematuro**.
  - L'Host A trasmette due sequenze in successione: una da 8 byte (con identificativo 92) e una da 20 byte (con identificativo 100).
  - L'Host B trasmette gli ACK per entrambi, tuttavia l'ACK relativo alla prima sequenza (quella con identificativo 92) arriva a destinazione dopo il timeout.
  - L'Host A, nel frattempo, ha inviato nuovamente la sequenza con identificativo 92.
  - L'Host B, che riceve un duplicato, se nne accorge e invia un unico ACK. Abbiamo risparmiato nell'invio di ACK.
- **ACK cumulativi che coprono perdita di ACK.**
  - L'Host A trasmette due sequenze esattamente come nell'esempio precedente.
  - L'Host B trasmette gli ACK cumulativi per entrambi, ma l'ACK relativo alla prima sequenza non arriva a destinazione. Arriva a destinazione l'ACK relativo alla sequenza più recente: l'Host A comprende che l'Host B ha ricevuto correttamente anche la sequenza più vecchia, nonostante il relativo ACK non sia arrivato a destinazione.

Ultima osservazione che facciamo per il sender è cosa succede nel caso in cui una "sequenza di sequenze" risulti interrotta a causa del non arrivo di una di queste al receiver.

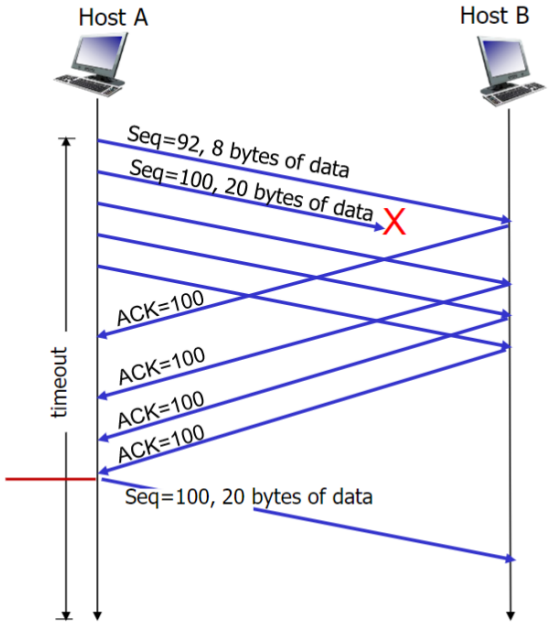
- L'ACK viene sempre inviato per la sequenza più vecchia non riconosciuta. Questo significa che con più pacchetti arrivati a destinazione (a seguito del problema) il receiver risponderà con altrettanti ACK aventi tutti lo stesso numero di sequenza (quello relativo alla sequenza non trasmessa con successo).
- Il protocollo prevede che il sender, dopo aver ricevuto 3 ACK aggiuntivi con lo stesso identificativo, lo invierà nuovamente. **Non si attende il timeout**!

TCP fast retransmit

if sender receives 3 additional ACKs for same data ("triple duplicate ACKs"), resend unACKed segment with smallest seq #

- likely that unACKed segment lost, so don't wait for timeout

Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – lost segment is likely. So retransmit!



8.5.4.4 TCP Receiver

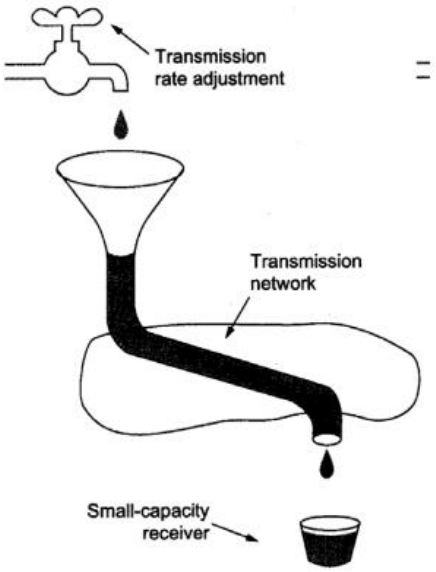
Per quanto riguarda il TCP Receiver limitiamoci ad osservare i seguenti eventi:

Event at Receiver	TCP Receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single cumulative ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expect seq. # . Gap detected	Immediately send *duplicate ACK*, indicating seq. # of next expected byte
Arrival of segment that partially or completely fills gap	Immediate send ACK, provided that segment starts at lower end of gap

8.5.5 TCP Flow Control

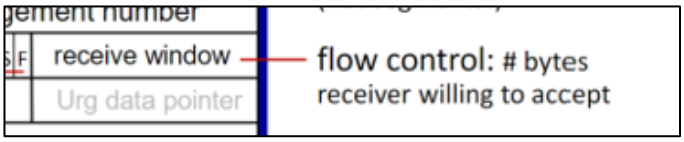
8.5.5.1 Perché parliamo di flow control

Il controllo di flusso serve ad evitare che il TCP sender invii più dati o invii dati a un rate superiore al rate con cui l'applicazione li preleva dal buffer di ricezione. Se il primo rate è superiore il buffer di ricezione tende a crescere fino a quando non ci sarà un trabocco (con conseguente perdita dei dati).



Se entra tanta acqua nel rubinetto ma il bicchiere prende poco (perchè ha poco spazio) chiaramente qualcosa va gestito.

Il ricevitore segnala periodicamente quanto spazio libero ha ancora nel buffer: lo fa per mezzo del campo `Receive Window` introdotto nell'intestazione del TCP.

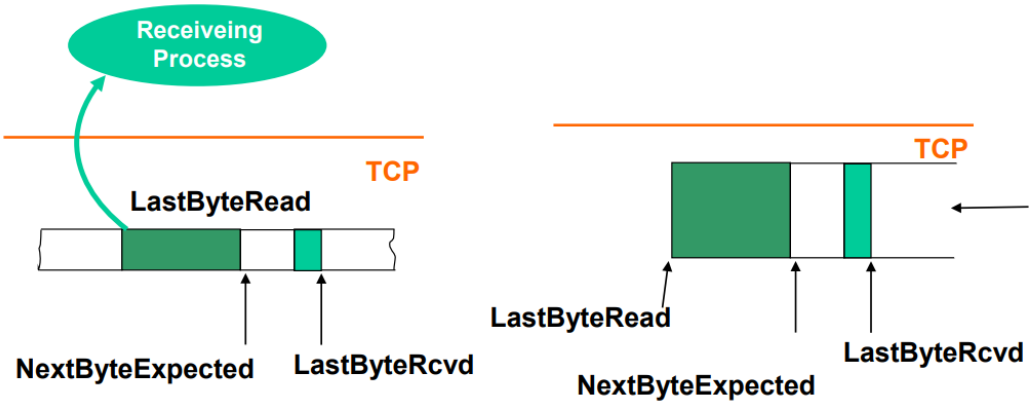


Il sender si adegua di conseguenza! Dobbiamo capire:

- come fa il ricevitore a calcolare lo spazio ancora libero nel buffer;
- come fa il trasmettitore, dopo aver ricevuto l'informazione precedente, a capire quanti byte effettivamente possono essere mandati senza far traboccare il buffer del ricevitore.

8.5.5.2 Lato receiver

Lo spazio libero è lo spazio non occupato: se si conosce dimensione del buffer e spazio occupato nel buffer si ottiene lo spazio libero facendo la differenza. La dim. del buffer è posta nella variabile `RcvBuffer`. Come determiniamo lo spazio occupato? Consideriamo le variabili di stato memorizzate nei dispositivi.



Partiamo dal receiver: `LastByteRcvd` consiste nel numero di sequenza dell'ultimo byte arrivato (non necessariamente in ordine); `NextByteExpected` consiste nel prossimo byte che il dispositivo si aspetta; `LastByteRead` è l'ultimo byte letto dall'applicazione (che non è più di interesse per il ricevitore).

- Da `LastByteRead` + 1 a `NextByteExpected` – 1 abbiamo una bufferizzazione in ordine.
- Possono esserci pacchetti fuori ordine, che lasciano dei gap nel buffer (spazio bianco in figura).

Chiaramente dobbiamo considerare nella dimensione dello spazio occupato i pacchetti fuori ordine, dato che sono stati memorizzati. Domanda: consideriamo anche lo spazio vuoto nella dimensione? Sì, perché quello spazio è relativo a pacchetti che non sono ancora arrivati, ma che si suppone essere stati già spediti.

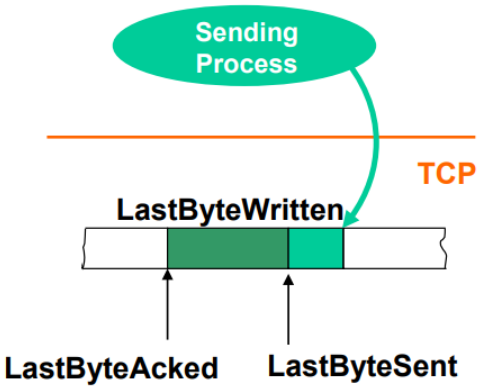
In conclusione

$$\text{SpazioOccupato} = \text{LastByteRcvd} - \text{LastByteRead} \leq \text{RcvBuffer}$$
$$\text{ReceiveWindow} = \text{RcvBuffer} - \text{SpazioOccupato}$$

Il valore calcolato, `ReceiveWindow`, viene posto nell'omonimo campo del segmento TCP.

8.5.5.3 Lato sender

Cosa fa il sender con il valore `ReceiveWindow`? Può inviare un numero di byte pari a `ReceiveWindow`? Se ciò non è possibile quanti byte in meno deve inviare rispetto a `ReceiveWindow`?



Nel lato sender abbiamo memorizzate le seguenti variabili: `LastByteAcked`, che indica il numero di sequenza dell'ultimo byte inviato su cui si è avuto ACK (ricordarsi che si usano ACK cumulativi); `LastByteSent`, che indica il numero di sequenza dell'ultimo byte inviato. Nella figura abbiamo:

- i byte in verde scuro, che sono byte inviati su cui non si è ancora ricevuto un ACK;
- i byte in verde chiaro, che non sono stati ancora inviati.

Il sender, ricevuto il pacchetto, verifica se il flag A è alzato: se lo è si ha un ACK e conseguentemente si va ad aggiornare la variabile `LastByteAcked`.

Riflessione: il valore di `ReceiveWindow` trasmesso dal receiver per mezzo del pacchetto rappresenta una fotografia. Il sender riceve questo valore "dopo un po' di tempo", ergo la fotografia potrebbe non essere più allineata alla realtà. I byte in verde scuro sono stati ricevuti (molto probabilmente) dal receiver dopo averci inviato il pacchetto!

A questo punto possiamo rispondere alla domanda iniziale.

- Non possiamo inviare un numero di byte pari a `ReceiveWindow` per la riflessione fatta prima.
- Si sottrae da `ReceiveWindow` i byte in verde scuro (che chiamiamo `SpazioOccupatoPostInvio`).

In conclusione:

$$\text{SpazioOccupatoPostInvio} = \text{LastByteSent} - \text{LastByteAcked} \leq \text{ReceiveWindow}$$
$$\text{UpdatedReceiveWindow} = \mathit{ReceiveWindow} - \text{SpazioOccupatoPostInvio}$$

Attenzione all'esame: non si inverta minuendo e sottraendo nel calcolo di `SpazioOccupatoPostInvio`: ha poco senso affermare di aver inviato un numero negativo byte.

8.5.5.4 Problema: deadlock con spazio libero nullo nel receiver

Poniamoci la seguente domanda: cosa succede se lo spazio disponibile nel buffer del receiver è pari a zero?

- Otterremo $\text{ReceiveWindow} = 0$, quindi $\text{UpdatedReceiveWindow} = 0$
- Se il receiver pone il primo valore e il sender ottiene il secondo allora il sender non invierà nulla.
- Ma se il sender non può inviare pacchetti allora il receiver non invierà pacchetti di ACK.

Siamo in una situazione di stallo. La soluzione proposta è l'invio periodico di un segmento di un byte dal sender al receiver per stimolare una reazione.

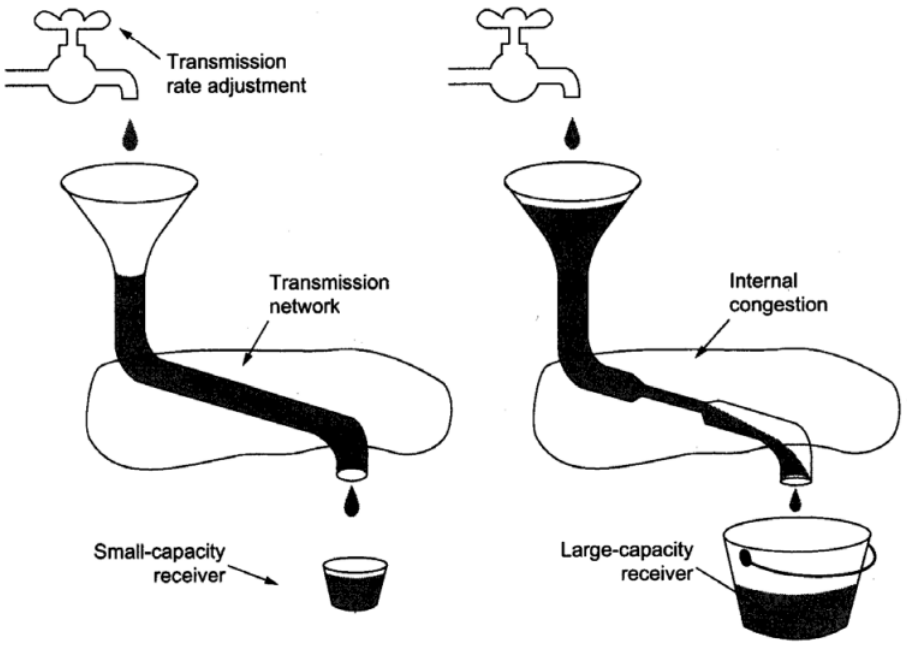
8.5.6 TCP Congestion Control

8.5.6.1 Introduzione

Il Controllo di congestione è implementato nel protocollo TCP, ma si ha anche a livello Network.

- **Cosa si intende con congestione?**
  Si pensi al traffico stradale: un incrocio è congestionato quando arrivano più macchine di quelle che escono. La situazione è simile nei router (analogia incrocio stradale – router nelle reti Internet): considerati i link di entrata e i link di uscita il traffico in ingresso è maggiore del traffico in uscita.
- **Dove si verifica la congestione?** Nei router intermedi, all'interno del core di Internet.
- **Quali sono le conseguenze della congestione?**
  Aumenta il ritardo.
  - Si generano code di pacchetti
  - Si perdono pacchetti perché col perdurare della congestione lo spazio libero sul buffer sarà sempre più piccolo, fino ad annullarsi.
  - Se si perdono pacchetti questi dovranno essere nuovamente trasmessi.
- **Perché si crea congestione presso un router**?
  Tante sorgenti inviano pacchetti presso un particolare router.
- **Confronto tra Flow Control e Congestion Control**.
  In entrambi i casi la conseguenza è il riempimento del buffer con rischio di packet loss. Nel flow control il buffer è quello del ricevitore, nel congestion control si parla dei buffer in entrata dei router.

Con la seguente figura si fa un ulteriore confronto tra Flow Control e Congestion Control



Nel flow control si vuole evitare che un receiver con un piccolo buffer riceva un numero spropositato di pacchetti (o pacchetti di grosse dimensioni). Nel congestion control si ha una sorta di "strozzatura" lungo il tubo: si vuole evitare il caso in cui un receiver con buffer di grandi dimensioni ricevano poco a causa della congestione.

Dove sta la maggiore complessità?

- Nel flow control la questione riguarda solo due host (un sender e un receiver), se la gestiscono autonomamente.
- Nel congestion control si risolvono problematiche che non riguardano specifici host, ma la rete tutta. Necessità di risolvere il problema (*decongestionare*) in modo cooperativo

Gli approcci possibili sono i seguenti (come si verifica la presenza della congestione?):

- ***network-assisted***
  I dispositivi del core segnalano che sono congestionati o sono vicini ad esserlo (ad esempio settano un bit particolare all'interno di un pacchetto, o addirittura segnalano un rate di invio tale da scongiurare la congestione).
- ***end-to-end***
  Si hanno solo informazioni da sender e receiver, non sappiamo nulla dei router. Si verifica se sono presenti i "sintomi" della congestione (timeout frequenti, triplici ACK duplicati[11], …) e si interviene di conseguenza. **Approccio implementato dal protocollo TCP**!

Soffermiamoci sul secondo approccio, e vediamo come viene attuato nel protocollo TCP.

L'obiettivo è trasmettere il più veloce possibile evitando fenomeni di congestione.

Poniamoci le seguenti domande sul sender:

- come si varia il rate di trasmissione?
- come fa a percepire che si è verificata congestione?
- come riduce il rate dopo aver individuato la congestione?

8.5.6.2 Come si varia il rate di trasmissione

Si prenda a riferimento una finestra, un certo intervallo di tempo: il RTT. Il rate di trasmissione si altera trasmettendo un numero maggiore o inferiore di byte all'interno di quell'intervallo di tempo. Quindi:

- l'applicazione decide quanti byte inviare nell'intervallo di tempo
- riduce il numero di byte da inviare se vuole ridurre il rate (in caso di congestione);
- aumenta il numero di byte se vuole aumentare il rate (se non percepisce congestione).

La quantità di byte che si vuole trasmettere nella finestra è detta `cwnd`, cioè *congestion window*. Per mezzo del seguente rapporto otteniamo il rate di trasmissione (dove `cwnd` cambia nel tempo)

$$\text{rate} = \frac{cwnd}{RTT}$$

8.5.6.3 Stabilire che si è verificata congestione

Come si capisce che da qualche parte c'è congestione?

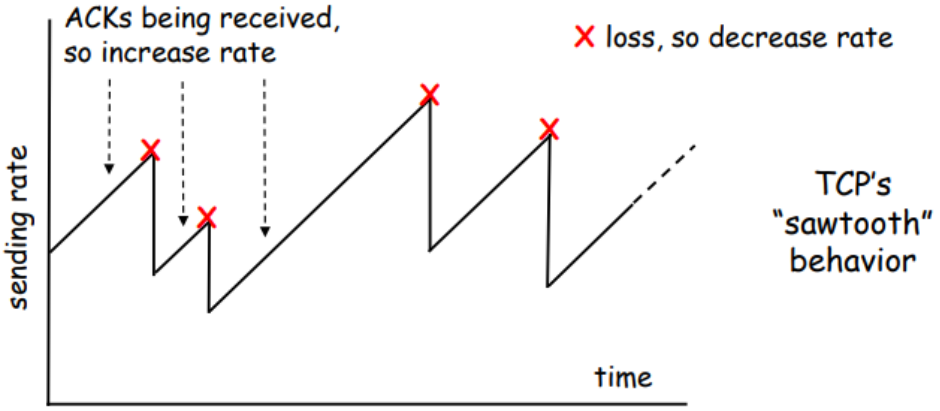
- **Eventi negativi**.
  Abbiamo anticipato che TCP percepisce la perdita di pacchetti come il manifestarsi di congestione in qualche router dal sorgente alla destinazione. È un'assunzione fatta da TCP, che non è proprio verissima. Il protocollo assume che si verifica congestione con i seguenti eventi:
  - triplice ACK;
  - timeout.

  Al manifestarsi di entrambi gli eventi si riduce il rate di trasmissione.
- **Evento positivo**.
  Il protocollo considera un evento positivo la ricezione degli ACK: se viaggiano ACK la rete non è congestionata, e quindi si può aumentare il rate di trasmissione.

[11] E' automatico avere congestione in presenza di triplice ACK duplicato? No (si pensi fortemente inaffidabili), ma il TCP fa questa assunzione.

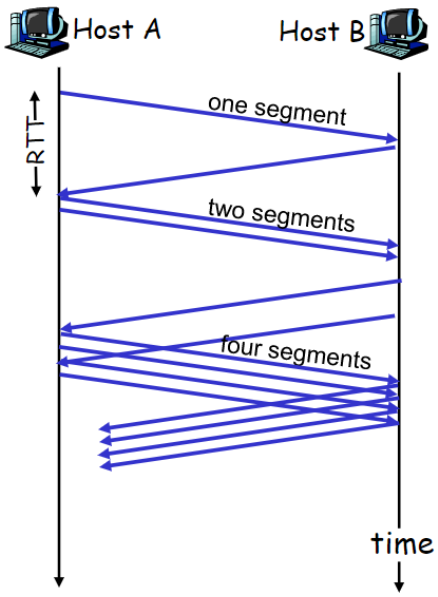
8.5.6.4 Alterazione del rate di trasmissione

Il rate di trasmissione, citando il professore, si altera "a sentimento": si aumenta linearmente la finestra all'arrivo degli ACK, la si riduce drasticamente in caso di perdite (triplice ACK duplicato e/o timeout). Si ottiene il cosiddetto "andamento a dente di sega":



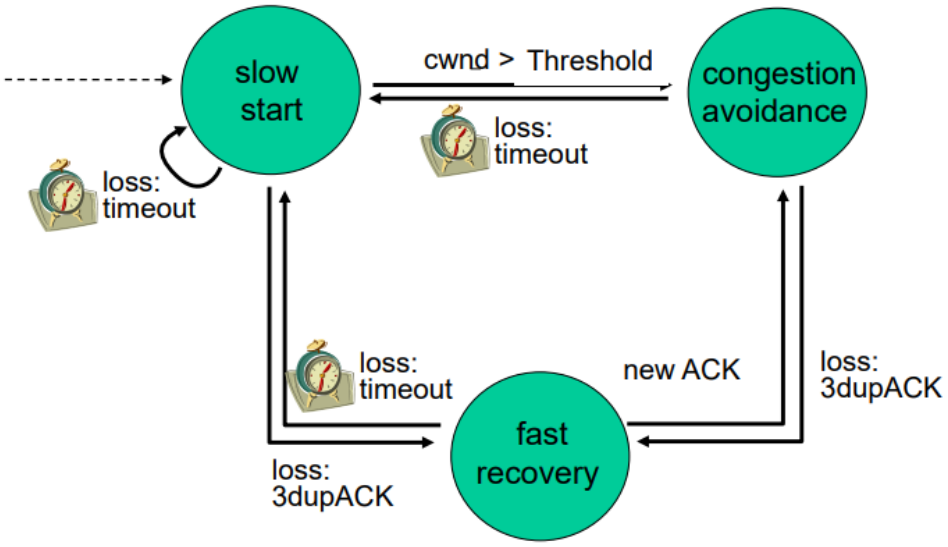
In realtà l'andamento appena descritto si ha dopo un po' che il protocollo TCP ha lavorato. A tal proposito andiamo a distinguere le seguenti fasi:

- ***slow start***
  La fase iniziale è detta slow start. Si parte con una finestra molto piccola pari a `cwnd = 1 MSS`, dove MSS sta per *Maximum Segment Size* (la dimensione massima del payload di un segmento).
  - La decisione è dovuta all'incertezza, non abbiamo elementi per dire se possiamo partire veloci o lenti.
  - Nella fase iniziale non si fa un aumento lineare, ma un aumento esponenziale (figura a lato)! Questo per evitare un'espansione iniziale troppo lenta.
  - L'espansione mantiene andamento esponenziale finchè `cwnd` non raggiungerà una soglia `threshold`.
  - A quel punto si passa nella fase successiva, detta di *congestion avoidance*.



- ***congestion avoidance***
  La fase a regime è detta congestion avoidance, che è quella tipica. Si ha l'andamento a dente di sega descritto precedentemente. Cosa succede se si ha perdita di pacchetti? Abbiamo già citato gli eventi che sono considerati sintomo di perdita di pacchetti, ma non abbiamo detto che il timeout è considerato un evento più grave rispetto al triplice ACK duplicato: seguono comportamenti diversi.
  - ***Azioni nel caso in cui si abbia triplice ACK duplicato.***
    Nel caso di triplice ACK duplicato si dimezza il threshold e si altera cwind (attenzione, cwind non viene dimezzato, se facessi solo questo potrei avere una finestra nulla)
    `threshold = cwind /2`
    `cwind = cwind/2+3 MSS`
    Si ha il passaggio in uno stato detto *fast recovery*, da cui si esce non appena si riceve un ACK (a quel punto si torna in congestion avoidance, ergo si ha crescita lineare di cwind). Si osservi che in cwnd viene posto un valore ≈ `threshold`
  - ***Azioni nel caso in cui si abbia timeout***.
    Il timeout è percepito come un evento di maggiore gravità rispetto al triplice ACK duplicato. Il threshold viene dimezzato come prima, ma il cwind viene riportato al valore iniziale!
    `threshold=cwind/2`
    `cwind=1 MSS`
    Si ha il ritorno nello stato di slow start, dove cwind cresce esponenzialmente.

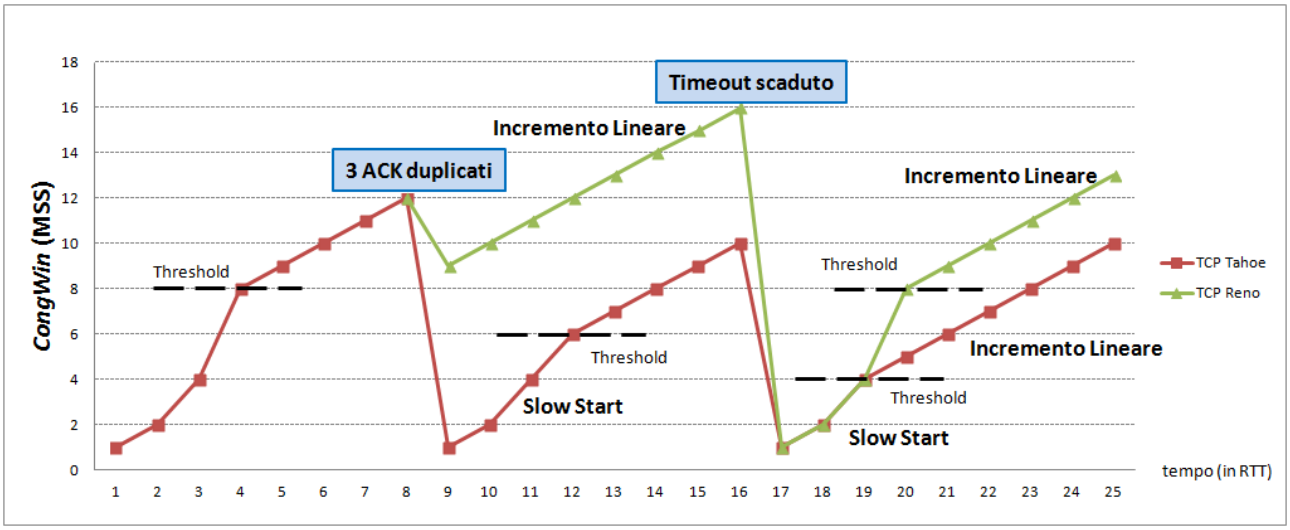
Formalizziamo quanto spiegato col formalismo dell'automa a stati finiti



E concludiamo mostrando l'andamento completo della finestra di congestione. Si fa confronto tra due versioni del congestion control:

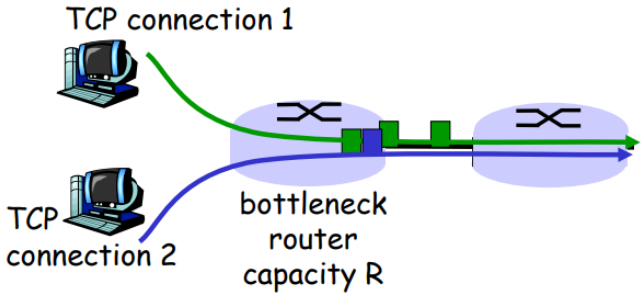
- TCP Tahoe, versione iniziale dove il triplice ACK duplicato risultava essere trattato come il timeout;
- TCP Reno, versione attuale e spiegata precedentemente.

E' chiaro che con TCP Reno il congestion control permette una trasmissione di byte maggiore!



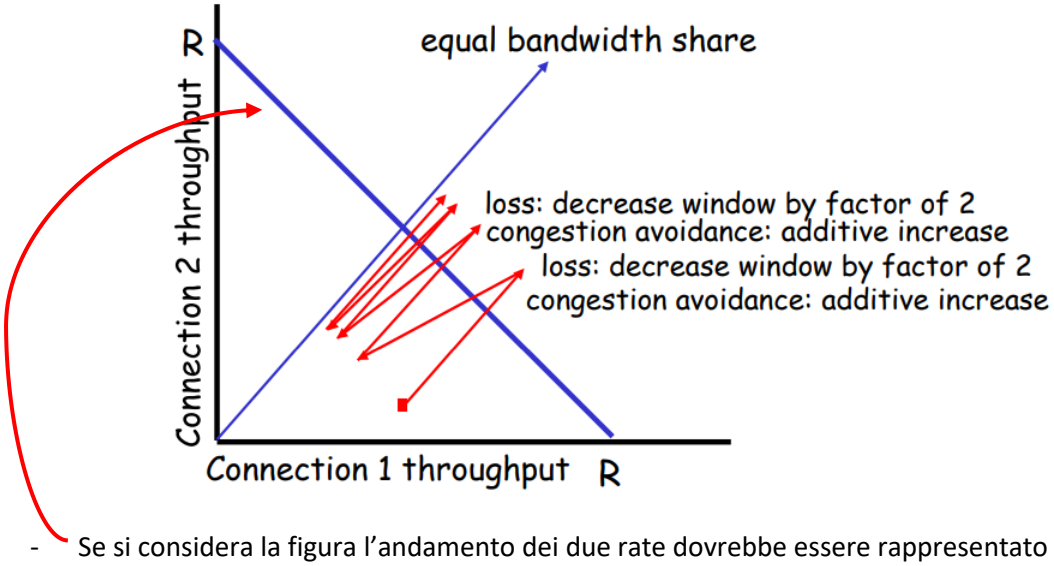
8.5.6.5 Fairness della congestione

Il protocollo adottato può essere definito fair?



- Tutti i dispositivi percepiscono una congestione.
- Tutti i dispositivi abbassano il rate nei modi spiegati.
- Possiamo dire che tutti i K dispositivi avranno le stesse possibilità, cioè che ciascun nodo potrà usufruire di un rate R/K? Oppure alcune connessioni saranno favorite a discapito di altre.

La somma del throughput delle due connessioni deve essere uguale al rate R.



- Se si considera la figura l'andamento dei due rate dovrebbe essere rappresentato con la retta che va dal secondo al quarto quadrante (in cima ho un rate uguale ad R e l'altro nullo, in basso a destra i due rate sono invertiti).
- L'idea è che con i due throughput non si dovrebbe andare fuori dall'area del triangolo limitato dalla retta detta prima, dall'ascissa e dall'ordinata.
- L'andamento effettivo è quello in rosso, fortemente irregolare ma "sostanzialmente fair": in alcuni tratti si esce dal triangolo, ma con le azioni intraprese a seguito di triplici ACK duplicati e timeout si rientra all'interno del triangolo!

In conclusione: il protocollo è fair!

Coperta corta perché molte applicazioni sono limitate da questi controlli di congestione, in particolare quelle che richiedono un throughput minimo. Molto spesso gli sviluppatori adottano delle "scappatoie" per aggirare il congestion control:

- utilizzare il protocollo UDP,
- aprire più connessioni TCP in parallelo.

9 SICUREZZA

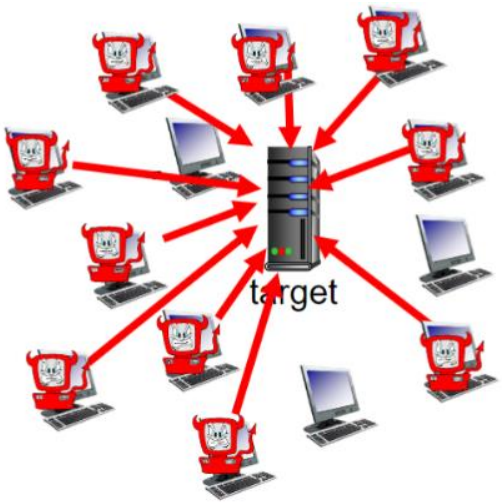
9.1 Introduzione

Ci limitiamo ad introdurre la questione, in quanto questi argomenti verranno affrontati alla magistrale: modulo Cybersecurity della LM Computer Engineering o la LM interdipartimentale in Cybersecurity.

- **La sicurezza agli albori della rete**.
  Inizialmente la rete viene costruita secondo una visione ottimistica, dove i soggetti che operano in rete sono benevoli e sicuramente non agiranno per danneggiare la rete stessa e gli utenti connessi. Segue che la questione non è stata affrontata all'inizio (per dare un'idea, all'inizio le password erano registrate in chiaro).
- **La sicurezza oggi**.
  L'espansione della rete ha portato ad un aumento considerevole del numero di utenti. Non possiamo dare per scontato che gli utenti non faranno nulla di malevolo, se sono tanti. Per principio una rete non potrà mai essere sicura al 100%, ma si può agire in modo tale da ottenere una percentuale accettabile di sicurezza. La questione si articola in:
  - progettare applicazioni valide dal punto di vista della sicurezza;
  - innalzare la consapevolezza dell'utente (*awareness*) verso i pericoli della rete (il primo nemico della rete è l'utente stesso).

  Abiamo i malware (software malevoli), che si classificano in:
  - *virus*, programmi dolosi che si autoreplicano (scaricare un programma dalla rete)
  - *worm*, programmi dolosi che si autoreplicano ricevendo oggetti dalla rete (si può ricevere un worm anche semplicemente connettendosi a una rete Internet).

  Un host potrebbe essere infettato e arruolato come botnet (il nostro dispositivo compie azioni per conto di terzi insieme ad altri dispositivi, ad esempio una botnet potrebbe essere utilizzata per compiere un attacco *Denial of Service*).



Quello che faremo in questo capitolo, in sostanza, è capire:

- i principi alla base della sicurezza,
- cosa intendiamo con sicurezza (diversità di significati),
- quali sono le minacce alla sicurezza, e
- quali contromisure possiamo adottare.

Vedremo come questi principi vengono attuati all'interno della pila protocollare. Introdurremo anche firewall e sistemi di individuazione di intrusione.

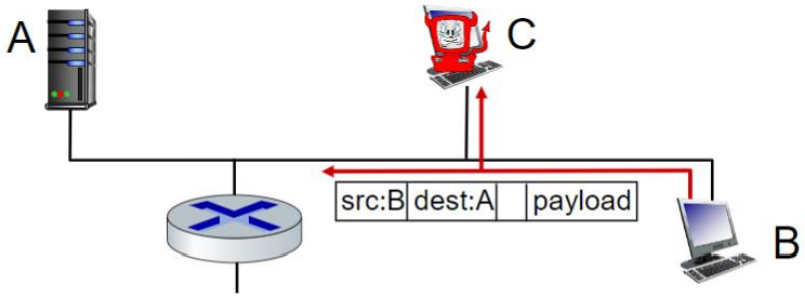
9.2 Possibili esempi di azioni di utenti malevoli

9.2.1 Packet sniffing

Collegare un dispositivo su Internet ha delle conseguenze dal punto di vista della sicurezza. *La prima cosa che si può fare è sniffare* (cit.):

- abbiamo un server A e un client B che dialogano tra loro;
- tra i due server si trova un client C.

Il client C intercetta i pacchetti con lo scopo di leggerne il contenuto

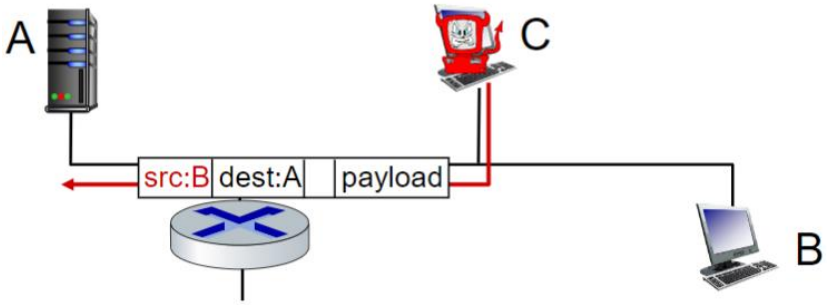


La cosa non è per forza malevole (in alcuni casi si attua packet sniffing con lo scopo di studiare le caratteristiche del traffico della rete), ma è chiaro che un'operazione del genere è problematica in un contesto dove gli utenti scambiano informazioni sensibili (Esempio: passwords). È più facile *origliare* nel caso di canali broadcast (informazione inviate a tutti i dispositivi presenti).

Esempio di programma: *Wireshark* è un packet sniffer gratuito.

9.2.2 Fake identity / IP Spoofing

L'utente malevolo può andare oltre il semplice sniffare. Non si limita a leggere i pacchetti transitanti sul link, ma introduce nuovi pacchetti interloquendo sia col server A che col client B.



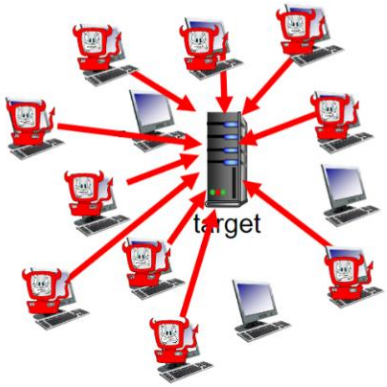
L'obiettivo dell'utente malevolo è:

- scrivere al server A dando l'impressione che l'interlocutore sia il client B;
- scrivere al client B dando l'impressione che l'interlocutore sia il server A.

La cosa è ben peggiore del semplice sniffare: il client B potrebbe essere indotto a fornire informazioni sensibili all'utente malevolo. La cosa è scoraggiata: i router, ad esempio, sono pensati in modo tale da non inoltrare datagram con indirizzo IP sorgente diverso da quello della rete da cui provengono.

9.2.3 Attacco *Denial of service* (DoS)

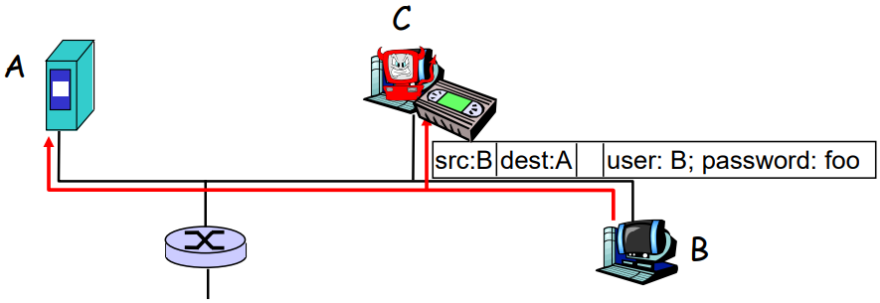
L'attacco *Denial of Service* ha lo scopo di impedire l'accesso ai servizi di un particolare server. I soggetti che attaccano inviano un numero tale di richieste da provocare il collasso del server, impedendone il funzionamento.



Chiaramente l'attacco ha maggiore successo se distribuito, in modo tale che non risulti riconoscibile l'autore dell'attacco: l'insieme dei dispositivi che contribuiscono all'attacco costituisce una ***botnet***.

9.2.4 *Record and playback*

Un soggetto malevolo potrebbe attuare *packet sniffing*, memorizzare il pacchetto trasmesso dal sender al receiver, e infine inviarlo nuovamente allo stesso receiver dopo un po' di tempo.



L'idea è far credere al receiver che il messaggio sia stato trasmesso veramente dal server: la cosa potrebbe essere usata per richiedere nuovamente operazioni sensibili (ad esempio bonifici bancari).

9.2.5 Introduzione di software malevoli

Soggetti malevoli potrebbero iniettare software malevoli (malwares) all'interno di un host.

- Potrebbero essere introdotti software spia che registrano i tasti utilizzati, i siti web visitati, in generale qualunque informazione sensibile.
- L'host potrebbe essere trasformato in una botnet, quindi sfruttati per l'esecuzione di attacchi distribuiti (come già anticipato)
- Software malevoli possono essere in grado di auto-replicarsi, cioè si muovono in modo tale da diffondersi in nuovi dispositivi.

Gioca un ruolo fondamentale l'ignoranza dell'utente, che tante volte introduce software malevoli in modo inconsapevole nel proprio dispositivo (*trojan horse*).

9.3 Definizione di network security

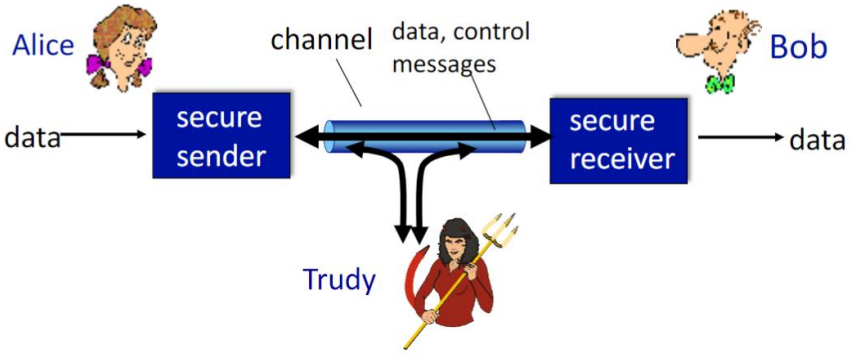
La network security è un concetto molto vasto. Varia in funzione dell'applicazione che si sta considerando: in alcuni casi la network security include tutte le caratteristiche che diremo, in altri casi solo alcune.

- **Confidenzialità**
  Il contenuto deve essere accessibile soltanto al sender e al receiver.
- **Autenticazione**
  Il sender vuole avere certezza sull'identità del receiver, e viceversa.
- **Integrità del messaggio**
  sender e receiver vogliono essere certi che il messaggio ricevuto non sia stato alterato.
- **Accesso e disponibilità**
  I servizi o le risorse devono essere accessibili solo agli utenti che effettivamente ne hanno diritto.

9.4 Implementazione della confidenzialità

9.4.1 Introduzione alla Crittografia

Introduciamo nelle nostre riflessioni Bob, Alice e Trudy.

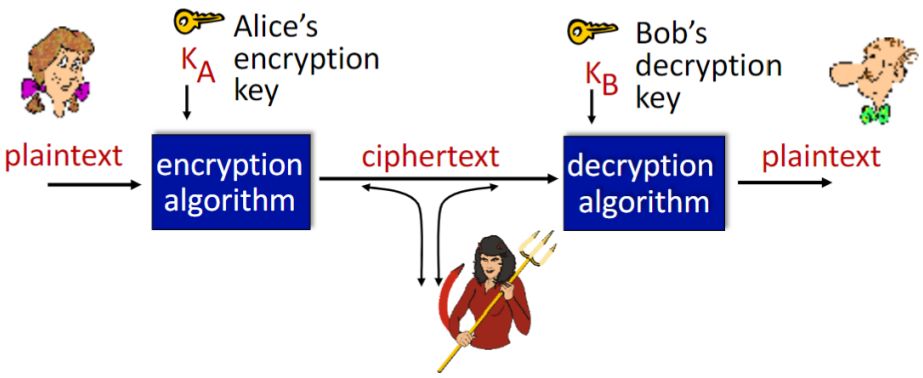


- Bob e Alice vogliono scambiarsi messaggi in modo "sicuro"

- Si ha un terzo soggetto, Trudy, che si pone tra Bob e Alice: intercetta i messaggi per leggerli, in alcuni casi potrebbe addirittura modificarli o introdurne di nuovi.

Con "sicuro" alludiamo senza ombra di dubbio alla confidenzialità, ma non è l'unica cosa che ci interessa: Alice e Bob hanno interesse anche all'integrità dei messaggi, così come all'autenticazione della controparte.

Cercheremo di affrontare queste problematiche per mezzo della crittografia.



- L'idea è la trasformazione del messaggio in chiaro (*plaintext*) in un messaggio cifrato (*ciphertext*): questo ultimo è il messaggio che il mittente trasmetterà sul canale di comunicazione.
- Alice esegue un algoritmo di cifratura ponendo in ingresso:
  - messaggio in chiaro *plaintext*
  - chiave di cifratura K_A

  L'algoritmo restituisce il messaggio cifrato *ciphertext* $K_A(m)$.
- Bob esegue un algoritmo di decifrazione ponendo in ingresso:
  - messaggio cifrato *ciphertext*
  - chiave di decifrazione K_B

  L'algoritmo restituisce il messaggio in chiaro *plaintext* m.
- Gli algoritmi per uso civile sono pubblici e conosciuti da tutti (anche Trudy). Trudy può intercettare solo *ciphertext* e non conosce la chiave di decifrazione K_B: questo significa che non può decifrare il messaggio.
- Osservazione: $m = K_B\big(K_A(m)\big)$, dove m consiste nel messaggio in chiaro (*plaintext*).
- **Prima classificazione**.
  - Si parla di crittografia a chiave simmetrica se $K_A = K_B$
  - Si parla di crittografia a chiave pubblica se $K_A \neq K_B$
  - In alcuni casi si utilizzano funzioni hash per garantire servizi di sicurezza. La cosa non è proprio crittografia in senso stretto.
- **Rompere uno schema di cifratura**.

  Si ricordi da Crittografia (per chi l'ha seguita) le seguenti tipologie di attacco (riportate brevemente in una diapositiva di questo corso).
  - ***cipher-text only attack***

    Trudy possiede solo del ciphertext, che può analizzare.
  - ***known-plaintext attack***

    Trudy possiede delle coppie plaintext-ciphertext
  - ***chosen-plaintext attack***

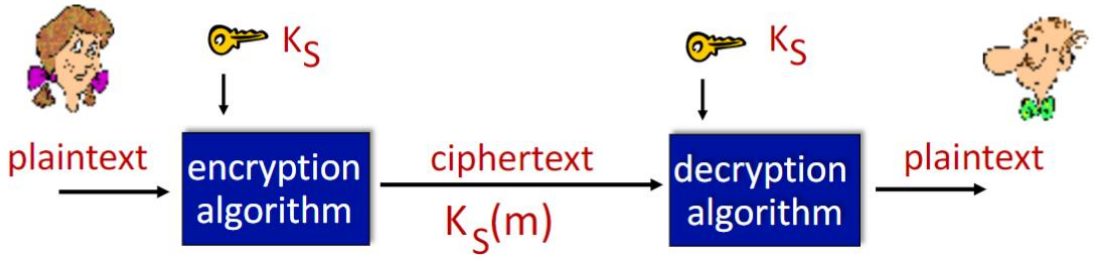
    Trudy possiede il ciphertext di un particolare plaintext che ha potuto scegliere!

  I due approcci principali sono l'attacco forza bruta (*brute force*) e l'analisi statistica (*statistical analysis*)

9.4.2 Crittografia a chiave simmetrica

9.4.2.1 Idea di base

La crittografia a chiave simmetrica, come già anticipato, prevede una completa uguaglianza tra chiave di cifratura e chiave di decifrazione.



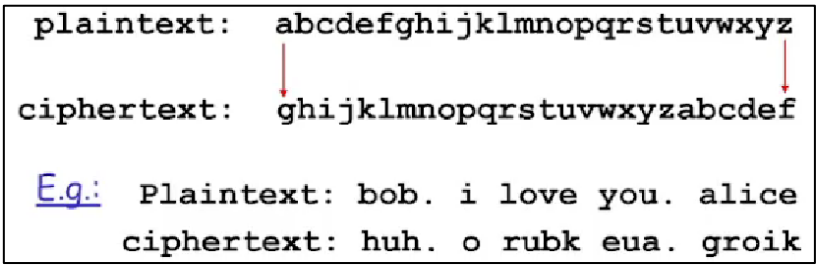
I due interlocutori, Alice e Bob, devono concordare la chiave prima dello scambio di messaggi (chiaramente non possono farlo attraverso il link di comunicazione, Trudy potrebbe leggere la chiave e utilizzarla).

9.4.2.2 Esempi

9.4.2.2.1 Cifrario di Cesare

L'esempio più semplice di cifrario è il cosiddetto Cifrario di Cesare: data una stringa di caratteri sostituiamo shiftando ogni singolo carattere di tot posizioni (ci si muove all'interno dell'alfabeto, trattando il tutto come un array circolare – si osservi come vengono tradotte le ultime lettere dell'alfabeto).

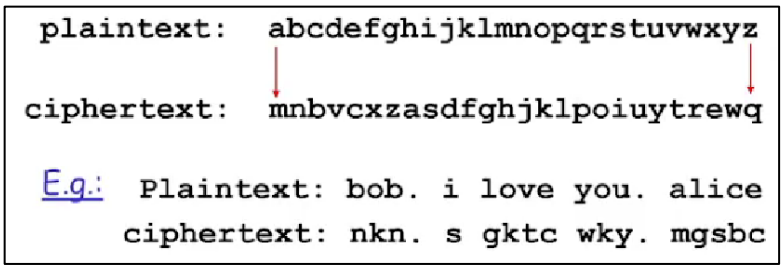
La chiave consiste nella dimensione dello shift. Nell'esempio ogni carattere è shiftato di sei posizioni:



Difetto. Solo 26 possibili chiavi (tante quanti i caratteri dell'alfabeto): questo significa che una persona che scopre il cifrario adottato può decifrare il messaggio con un attacco forza bruta.

9.4.2.3 Cifrario monoalfabetico

L'esempio precedente è il più banale dei cifrari monoalfabetici. Un cifrario monoalfabetico, in generale, è un cifrario dove una ogni lettere è sostituita da un'altra (sempre e solo quella, in ogni caso). Si può rappresentare le regole di trasformazione per mezzo di una tabella di corrispondenza



Lettera nel plaintext	Lettera nel ciphertext
A	M
B	N
C	B
D	V
…	

In questo caso le chiavi aumentano: si passa dalle 26 chiavi possibili del cifrario di Cesare in particolare a 26! Chiavi possibili di un cifrario monoalfabetico in generale.

9.4.2.3.1 Cifrario polialfabetico

Facciamo un passo ulteriore e introduciamo i cosiddetti cifrari polialfabetici. In questo caso la corrispondenza tra lettera del messaggio in chiaro e lettera del messaggio cifrato dipende anche dalla posizione nella stringa della prima. In un certo senso è come se io andassi a definire una tabella di corrispondenza per ogni posizione

- Voglio tradurre il carattere "a"

- Se questo si trova in posizione 1 sarà tradotto in “d”
- Se questo si trova in posizione 3 sarà tradotto in “o”
- Se questo si trova in posizione 4 sarà tradotto in “g”

In conclusione, prendendo a riferimento il cifrario di Cesare e una stringa di n caratteri:

- Un cifrario monoalfabetico consiste nell’applicare n volte il cifrario di Cesare
- Un cifrario polialfabetico consiste nell’applicare n volte un cifrario monoalfabetico.

9.4.2.4 Tipologie di cifrari simmetrici

Si distinguono due tipologie di cifrari

- **Cifrario con approccio di tipo stream**.
  Si lavora bit per bit.
- **Cifrario a blocchi**.
  Si lavora su blocchi di bit.

9.4.2.5 Esempio: Data Encryption Standard (DES)

Il Data Encryption Standard (DES) è un algoritmo a chiave simmetrico, ormai soppiantato.

- È un cifrario a blocchi, dove ciascun blocco è una sequenza di 64 bit.
- La chiave è una sequenza di 56 bit.
- Si adotta cipher block chaining per incrementare la sicurezza (si lega la cifrazione di un blocco al ciphertext del blocco precedente).
- **Quanto è sicuro?** Purtroppo poco: è stato decifrato con approccio forza bruta in meno di un giorno. Questo significa che in un tempo abbastanza breve (se si confronta con altri algoritmi) è possibile decifrare informazioni sensibili.
- 3DES. Prima dell’abbandono definitivo del DES è stato elaborata una variante 3DES: in essa ci cifra tre volte utilizzando tre chiavi diverse.

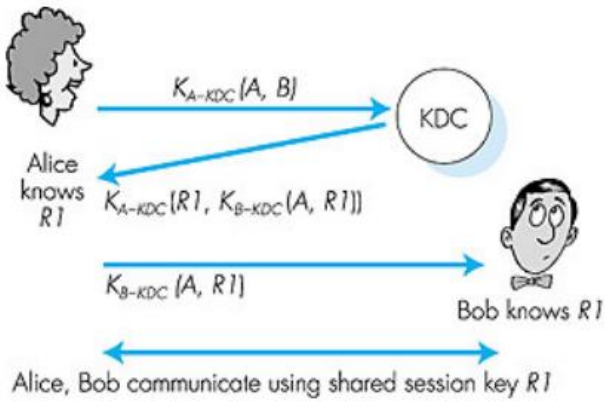
9.4.2.6 Esempio: Advanced Encryption Standard (AES)

L’Advanced Encryption Standard è un algoritmo a chiave simmetrico, ancora utilizzato.

- Anche questo è un cifrario a blocchi, ciascuno avente dimensione di 128 bit.
- La chiave può essere di 128, 192 o 256 bit (dimensione variabile, scelta in mano allo sviluppatore)
- Quanto è sicuro? Decisamente di più del DES. Per avere un’idea possiamo affermare che se un attacco forza bruta col DES richiede un secondo allora nel caso di AES richiederà 149 trilioni di anni!

9.4.2.7 Key Distribution Center per la condivisione di chiavi

Il Key Distribution Center (KDC) consiste in un’entità che agisce come intermediario tra due interlocutori.

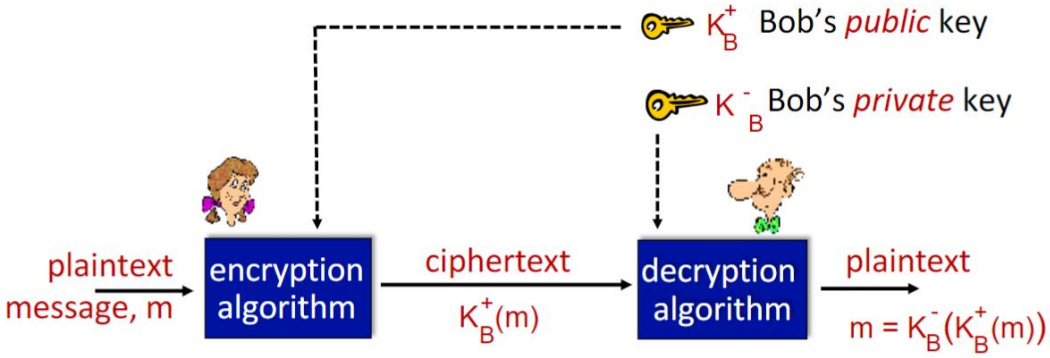


- Alice e Bob desiderano comunicare in maniera riservata e utilizzando la crittografia a chiave simmetrica. Non si possono incontrare perché abitano troppo lontano, e non possono usare la crittografia a chiave pubblica.
- Alice si rivolge al KDC e ottiene la chiave simmetrica K_{A-KDC}, che utilizzerà per dialogare col KDC.

- Bob si rivolge al KDC e ottiene la chiave simmetrica K_{B-KDC}, che utilizzerà per dialogare con KDC.
- Supponiamo che Alice voglia parlare con Bob. Lo comunica al KDC per mezzo di un messaggio cifrato con la chiave simmetrica K_{A-KDC}
- Il KDC risponde ad Alice trasmettendo un messaggio cifrato con K_{A-KDC}, contenente:
  - la chiave di sessione $R1$,
  - un messaggio cifrato da KDC con la chiave simmetrica di Bob K_{B-KDC}, contenente $R1$ e dati identificativi di Alice.
- Alice decifra il messaggio ricevuto dal KDC con la chiave K_{A-KDC}: prende atto della chiave di sessione $R1$ e trasmette a Bob il messaggio cifrato con K_{B-KDC}.
- Bob decifra il messaggio cifrato ricevuto da Alice con la sua chiave simmetrica K_{B-KDC}: prende atto della chiave di sessione $R1$ e si rende conto che è un messaggio trasmesso dal KDC ad Alice (solo il KDC, oltre a Bob, conosce la chiave simmetrica K_{B-KDC}).

9.4.3 Crittografia a chiave pubblica

La crittografia a chiave pubblica prevede, come già anticipato, due chiavi distinte.



Supponiamo di avere funzione di cifratura f e funzione di decifrazione g

- Si rivoluziona la crittografia, proponendo un approccio alternativo a quello tradizionale con una sola chiave (che deve rimanere privata e deve essere condivisa in modo sicuro tra i due interlocutori).
- Ogni interlocutore ha a disposizione due chiavi. Consideriamo Bob, che ha quanto segue…
  - Una chiave pubblica K_B^+, che l'utente diffonderà in broadcast a tutti i soggetti terzi che potrebbero essere interessati a interloquire con lui. La si usa per cifrare i messaggi.
    $$K_B^+(m) = ciphertext$$
  - Una chiave privata K_B^-, che l'utente manterrà segreta. La si usa per decifrare i messaggi!!
    $$K_B^-(ciphertext) = K_B^-\left(K_B^+(m)\right) = m$$
- **Proprietà tipiche della crittografia a chiave pubblica**.
  Possiamo aspettarci le seguenti proprietà:
  - $K_B^-\left(K_B^+(m)\right) = m$
  - non deve essere possibile poter calcolare la chiave privata a partire dalla chiave pubblica
  - Ulteriore proprietà che useremo più avanti (cifrare con chiave pubblica e decifrare con chiave privata è equivalente a cifrare con chiave privata e a decifrare con chiave pubblica)
    $$K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$$
- **Esempio di cifrario**: RSA (Rivest, Shamir, Adelson)

- **Perché la crittografia a chiave simmetrica non è stata soppiantata?**
  Per il semplice fatto che la crittografia a chiave pubblica ha un costo computazionale maggiore. Si consideri che DES è 100 volte più veloce di RSA. La soluzione prevalente è la seguente:
  - uso di cifrari simmetrici per lo scambio di informazioni, ma
  - preceduto da ricorso alla crittografia a chiave pubblica per permettere ai due interlocutori di scambiarsi la chiave del cifrario simmetrico sul canale di comunicazione (la chiave del cifrario passa cifrata sul canale, e Trudy non può leggerla!!)

9.5 Implementazione dell'integrità del messaggio

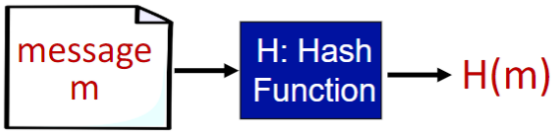
9.5.1 Cosa intendiamo con integrità del messaggio

L'integrità del messaggio è uno dei concetti relativi alla network security. Vogliamo essere sicuri che il messaggio sia "autentico", cioè:

- l'autore del messaggio sia effettivamente quello indicato;
- il contenuto del messaggio non venga alterato;
- il messaggio non sia replicato nel tempo (prevenzione attacchi di tipo *record and playback*);
- si mantenga la sequenza con cui i messaggi sono stati inviati.

9.5.2 Message digests

Supponiamo di avere un messaggio m molto grande, idealmente di dimensione infinita, e una funzione hash dove poniamo in ingresso il messaggio.



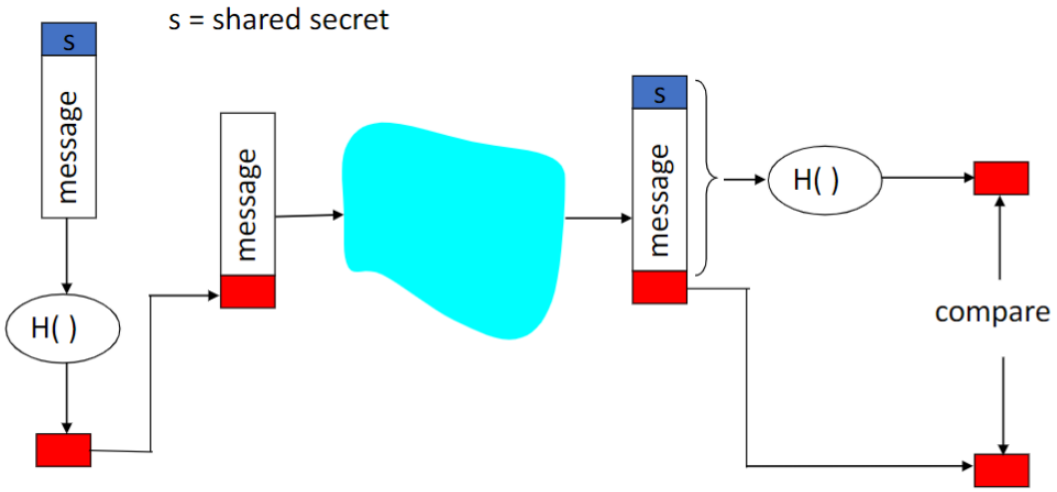
Il risultato della funzione Hash è una stringa di dimensione fissa che potremo definire, in un certo senso, il riassunto del messaggio. Adottiamo funzioni hash che presentano le seguenti proprietà:

- deve essere semplice da calcolare;
- deve essere irreversibile (complessità accettabile per calcolare il digest partendo dal messaggio, complessità esponenziale per trovare il messaggio a partire dal digest)
- deve essere resistente alle collisioni (deve essere minima la probabilità di avere due messaggi m ed m' diversi tra loro tali per cui $H(m) = H(m')$)
- il valore restituito deve apparire randomico.

Esempi di funzioni hash: MD5 e SHA1. Sequenze rispettivamente di 128 bit e 160 bit in modo tale da minimizzare le possibilità di collisione (si pensi alla checksum, è una funzione hash molto debole perché con stringhe di soli 16 bit è estremamente facile avere collisioni).

9.5.3 *Message Authentication Code* (MAC)

Supponiamo di avere due interlocutori che vogliono essere certi dell'integrità del loro messaggio (vogliono essere certi che il pacchetto non sia stato manipolato durante il suo passaggio nel canale di comunicazione).



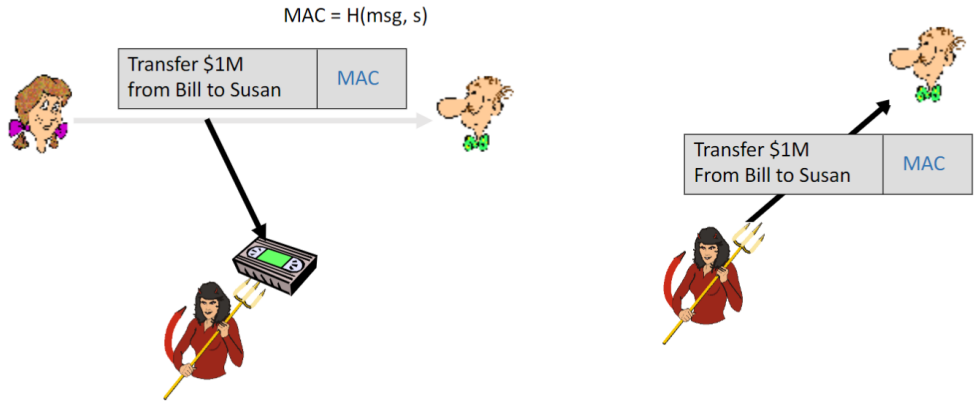
- I due interlocutori hanno un "segreto condiviso", che conoscono solo loro.
  Ogni volta che si vuole trasmettere un messaggio il mittente procede ponendo in ingresso nella funzione hash il messaggio concatenato al segreto condiviso. L'hash $H([m|s])$ ottenuto verrà concatenato al messaggio da trasmettere e posto sul link di comunicazione.

- Il destinatario riceve una stringa che consiste nel messaggio concatenato all'hash. Conosce il segreto condiviso, quindi pone anche lui $H([m|s])$. Dopo aver eseguito la funzione mette a confronto l'hash da lui generato con l'hash ricevuto dal mittente.
    - Se coincidono si può affermare che il messaggio non è stato manipolato.
    - Se non coincidono allora abbiamo due ipotesi:
        - il mittente ha cambiato il segreto condiviso e il destinatario non è stato aggiornato;
        - il messaggio è stato manipolato durante la trasmissione.
- Si garantisce anche l'autenticazione del mittente, dato che il segreto condiviso è conosciuto solo dai due interlocutori: l'hash corretto può essere trasmesso solo da chi conosce il segreto condiviso!
- Non garantisce sulla replicazione dei messaggi: il destinatario potrebbe ricevere lo stesso messaggio più avanti, ma in realtà il vero mittente è Trudy (che ha copiato il messaggio in passato e lo ha inviato nuovamente dopo un po' di tempo).
- **Variante HMAC**.
  Si prevede una variante nota come HMAC dove si fanno le stesse cose spiegate prima, ma si va a fare un ulteriore hash

$$H([\, H([s|m]) \mid m \,])$$

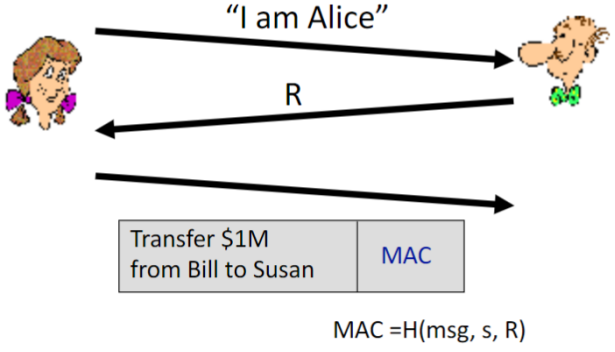
9.5.4 MAC contro Record and playback

Message Authentication Code può essere usato per contrastare il *Record and Playback* spiegato qualche pagina indietro. Abbiamo detto che il soggetto malevolo (Trudy) potrebbe copiare un messaggio e trasmetterlo nuovamente dopo un po' di tempo. Pensiamo a un bonifico bancario!



Senza ulteriori accorgimenti questo nuovo messaggio risulta regolare agli occhi del destinatario, nonostante sia stato inviato dal soggetto malevolo: se uno fa la verifica dell'integrità risulta tutto ok, salvo alterazione del segreto condiviso da parte dei due interlocutori (Alice e Bob).

- **Prima soluzione**.
  La prima soluzione prevede il coinvolgimento del TIMESTAMP nel lancio della funzione hash, in modo tale da permettere una verifica della data di invio del messaggio.
- **Seconda soluzione**.
  Proposta alternativa è prevedere una *call setup* dove il mittente (Alice) si presenta al destinatario (Bob) e quest'ultimo restituisce una stringa R che sarà utilizzata nella funzione Hash.



Il problema è risolto in quanto la stringa R risulta diversa in ogni sessione: segue che un nuovo invio del messaggio da parte di soggetti malevoli (Trudy) sarà evidente al destinatario (Bob).

9.5.5 Firma digitale: caratteristiche e ricorso alla crittografia a chiave pubblica

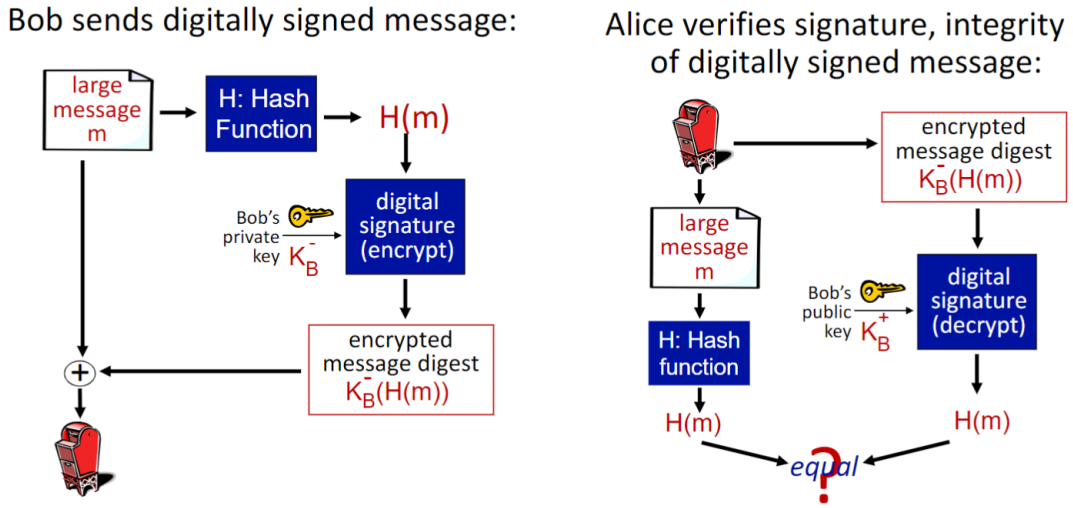
La firma digitale si basa su tecniche di crittografia ed è realizzata in modo tale da avere gli stessi effetti giuridici della firma cartacea. Supponiamo che Bob firmi un documento digitale e che questo venga trasmesso ad Alice. Possiamo affermare che la firma digitale verifica le seguenti proprietà:

- **Verificabile (*Verifiable*)**
  Il destinatario (Alice) può verificare e dimostrare che il documento è stato firmato dal mittente (Bob), e nessun altro.
- **Non falsificabile (*Non-forgeable*)**
  Il mittente (Bob) può dimostrare di non essere stato lui a firmare un particolare messaggio. Detta in altro modo: può dimostrare che qualcun altro ha firmato un particolare documento.
- **Non ripudiabilità (*Non repudiation*)**
  La firma apposta a un documento non può essere ripudiata. Il destinatario (Alice) può dimostrare che il mittente (Bob) ha firmato il documento x e non il documento x'.
- **Integrità del messaggio (*Message integrity*)**
  Si garantisce l'integrità del messaggio, descritta precedentemente. Il mittente (Bob) può dimostrare che ha firmato il documento x e non il documento x'.

A questo punto chiediamoci: queste proprietà sono rispettate dal Message Authentication Code?

- L'integrità del messaggio è garantita, ma da sola non basta.
- Non si ha la verificabilità: il segreto è condiviso tra due interlocutori, questo significa che il messaggio potrebbe essere siglato da uno dei due interlocutori.
- *Non-forgeable* e *Non repudiation* non sono più rispettate di conseguenza.

Segue che MAC non è sufficiente. Risolviamo ricorrendo alla crittografia a chiave pubblica.



- Il mittente (Bob) genera la firma in due step:
  (1) calcolo della funzione hash ponendo in ingresso il messaggio che si vuole firmare;
  (2) cifrazione della stringa ottenuta al punto precedente per mezzo della chiave privata del mittente.

  Il risultato della cifrazione al punto (2) è la firma, che si trasmette insieme al messaggio.
- Il destinatario (Alice) riceve il messaggio e la firma. Vuole verificare l'integrità del messaggio per mezzo della firma. Procede nel seguente modo:
  (1) calcola la funzione hash ponendo in ingresso il messaggio ricevuto dal mittente;
  (2) decifra la firma utilizzando la chiave pubblica del mittente (il risultato della decifrazione è l'hash generato dal mittente)
  (3) mette a confronto l'hash calcolato al punto (1) con l'hash ottenuto dalla decifrazione al punto (2).

  Se i due hash coincidono abbiamo l'integrità del messaggio!

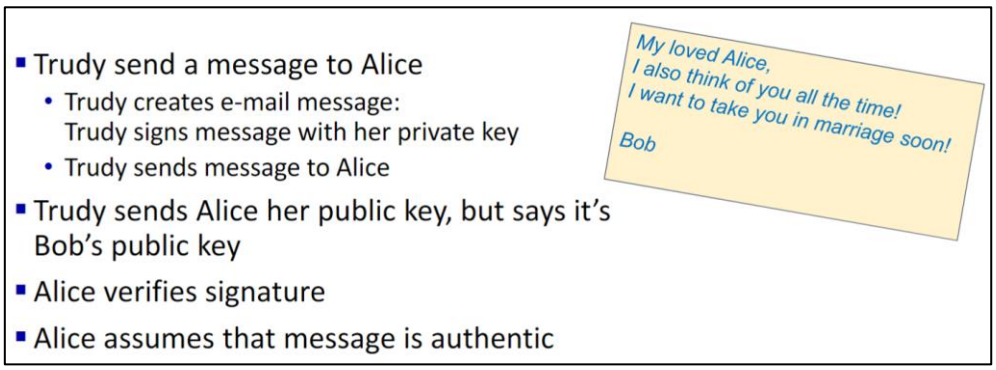
Sono rispettate le proprietà della firma digitale?

- Sicuramente abbiamo la verificabilità. Viene meno l'ambiguità del mittente (non abbiamo più il segreto condiviso conosciuto solo da due interlocutori, i possibili mittenti).
- Sicuramente abbiamo l'integrità del messaggio.
- Verificata anche la proprietà non-forgeable, non è possibile la falsificazione.
- Verificata anche la non repudiation: la firma può essere utilizzata da soggetti terzi per dimostrare che il mittente ha firmato un particolare documento (la chiave pubblica del mittente è conosciuta da tutti).

9.5.6 Public Key Certification

9.5.7 Problema

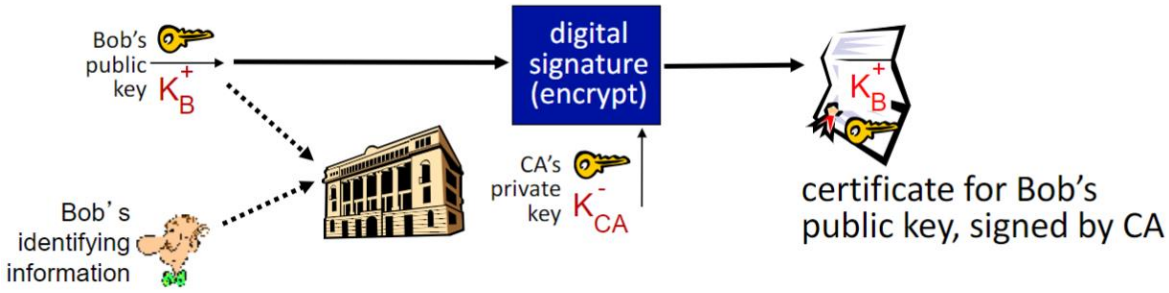
Quanto detto prima è molto bello, ma abbiamo ragionato in un contesto ideale dove supponiamo che la distribuzione delle chiavi pubbliche tra tutti gli utenti avvenga senza problemi. In realtà dobbiamo ipotizzare che utenti malevoli (Trudy) potrebbero trasmettere la loro chiave pubblica a terzi affermando che è di altri (in questo caso di Bob).



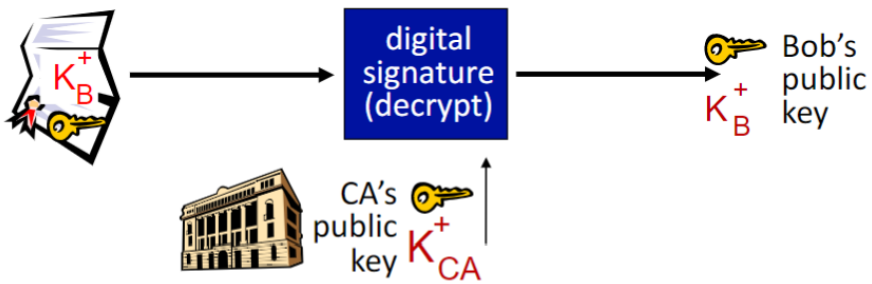
9.5.8 Certification Authority

Risolviamo il problema introducendo un'autorità atta a certificare le chiavi pubbliche.

- Ogni soggetto che ha interesse a firmare digitalmente i propri documenti si registra presso una delle possibili autorità (ce ne sono tante, si pensi allo SPID). L'authority, a partire dalle informazioni del soggetto, elabora un'identità digitale a cui viene associata una chiave pubblica.



- Il compito fondamentale dell'Authority è rilasciare un certificato atto a validare la chiave pubblica.
- Possiamo associare a un'Authority in particolare una chiave pubblica e una chiave privata, che per semplificare la spiegazione supporremo inviolabili (il tutto si basa su standard internazionali e su una P*ublic-Key Infrastructure* costituita dall'unione di tutte le certification authorities).
    - L'Authority emette il certificato utilizzando la sua chiave privata.
    - Qualunque soggetto può verificare la validità del certificato utilizzando la chiave pubblica dell'Authority.



9.5.9 Differenza tra MAC e firma digitale

Concludiamo osservando alcune differenze tra MAC e firma digitale.

- **Proprietà che garantiscono entrambi**.
  Entrambi garantiscono l'integrità del messaggio.
- **Proprietà non garantite da MAC (e garantite dalla firma digitale)**.
  Il MAC non garantisce la verificabilità, la non falsificabilità e la non ripudiabilità.
- **Firma digitale basata sulla crittografia**.
  Il MAC non è basato sulla crittografia, ma solo sulle funzioni hash. Nella firma digitale si mantengono le funzioni hash nell'ottica di ottimizzazione (se ne potrebbe fare a meno).

9.6 Implementazione dell'autenticazione

Abbiamo, come sempre, Bob e Alice che interloquiscono. Alice vuole dimostrare a Bob la sua identità: come può fare, tenendo conto della presenza ingombrante di Trudy? Elaboriamo un protocollo di autenticazione (Authentication Protocol).

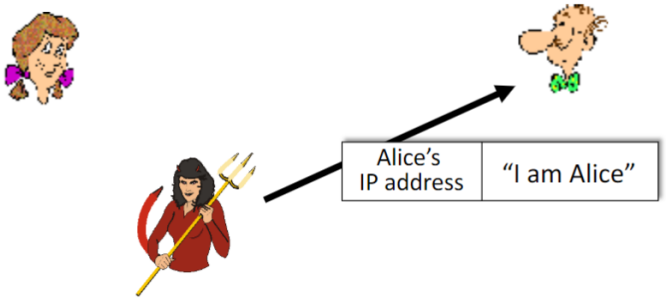
9.6.1 Prima proposta di protocollo (sbagliata)

Alice si presenta a Bob con un semplice messaggio. Ovviamente questo non va bene perché Trudy potrebbe inviare un messaggio dello stesso tipo affermando di essere Alice.

in a network, Bob can not "see" Alice, so Trudy simply declares herself to be Alice

9.6.2 Seconda proposta di protocollo (sbagliata)

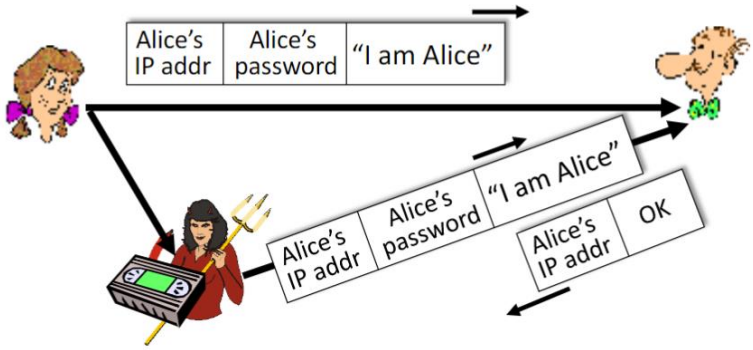
Alice si presenta a Bob con un semplice messaggio, ma ponendo in aggiunta il suo indirizzo IP. Il problema non è ancora risolto, dato che Trudy potrebbe fare spoofing e copiare l'indirizzo IP (per usarlo nei suoi messaggi).



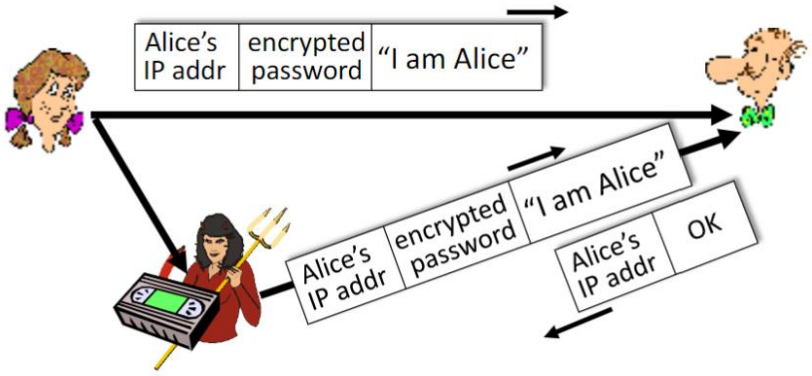
Trudy can create a packet "spoofing" Alice's address

9.6.3 Terza proposta di protocollo (sbagliata)

Alice si presenta a Bob indicando il suo indirizzo IP e la sua password segreta. Problema non risolto per due aspetti: la non segretezza della password, ma soprattutto la vulnerabilità del protocollo ad attacchi di tipo playback. Trudy può memorizzare il pacchetto e inviarlo più avanti

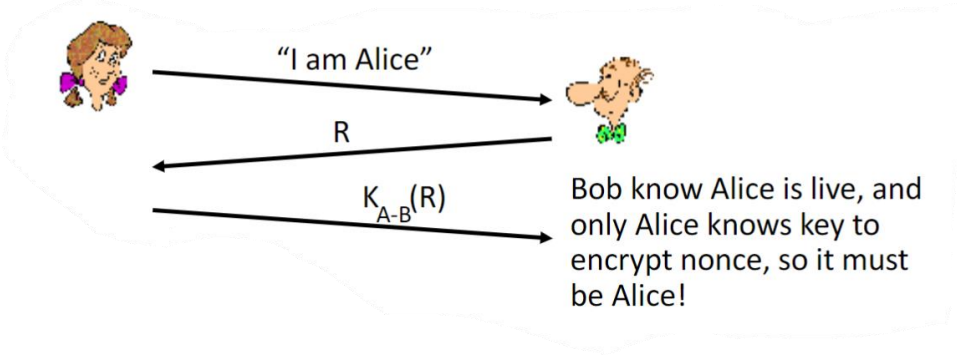


La prima cosa che si potrebbe fare è cifrare la password: il primo dei due problemi è risolto, ma permane la vulnerabilità ad attacchi di tipo playback.



9.6.4 Quarta proposta (pesante e incompleta)

Per evitare attacchi di tipo playback si può introdurre il nounce, come già visto precedentemente per risolvere gli attacchi di tipo playback in MAC.

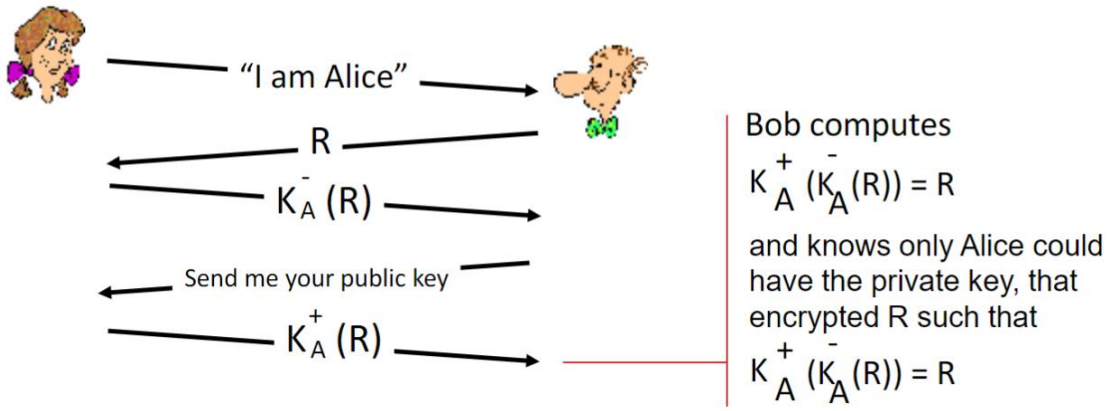


- Alice si presenta a Bob.
- Bob presenta il nounce.
- Alice cifra il nounce con la chiave simmetrica, chiave conosciuta solo da lei e Bob.

La soluzione inizia ad essere interessante, ma presenta il problema della chiave simmetrica: come possono condividerla i due interlocutori?

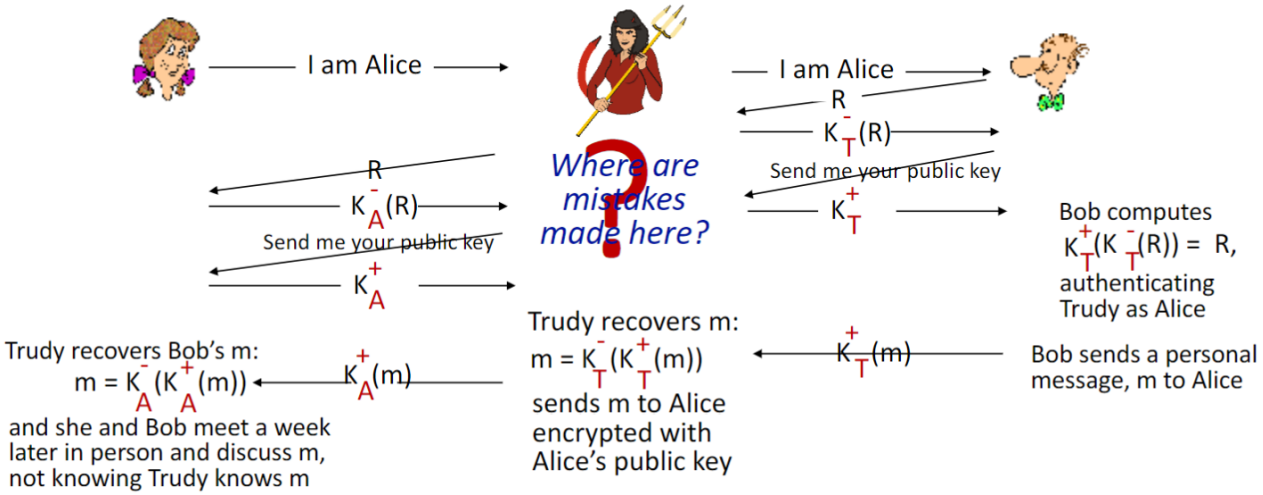
9.6.5 Quinta proposta (definitiva con osservazione su attacchi man in the middle)

Manteniamo lo stesso approccio precedente, ma lavoriamo per mezzo della crittografia a chiave pubblica.



- Alice si presenta a Bob
- Bob presenta il nounce.
- Alice cifra il nounce con la sua chiave privata e trasmette la sequenza a Bob
- Bob richiede la chiave pubblica di Alice
- Alice trasmette la sua chiave pubblica.

Bob decifra con la chiave pubblica e ottiene nuovamente il nounce, che può essere stato cifrato solo da Alice. In realtà rimane un problema: gli attacchi man in the middle (o woman in the middle – cit. Anastasi)



- Trudy intercetta sia i messaggi di Bob che i messaggi di Alice.
- Bob non riceverà il nounce cifrato da Alice, ma il nounce cifrato da Trudy.
- Bob non riceverà la chiave pubblica di Alice, ma la chiave pubblica di Trudy.
- Bob crede di aver ricevuto nounce cifrato da Alice e chiave pubblica di Alice, ma in realtà ha ricevuto nounce cifrato da Trudy e chiave pubblica di Trudy. Alice crede che i valori da lei trasmessi siano arrivati a destinazione con successo, quando in realtà sono stati intercettati da Trudy.
- Trudy possiede tutti gli elementi per poter leggere la corrispondenza tra Alice e Bob (la sua chiave privata e la sua chiave pubblica).

Come viene risolto il problema? Introducendo la Certification Authority di cui abbiamo già parlato.

9.7 Esempi di implementazione della sicurezza

9.7.1 Scelta del livello di implementazione

Nell'implementare la sicurezza dobbiamo scegliere il livello in cui operare. Lo sviluppatore può decidere in quale livello della pila protocollare implementare la sicurezza: ogni scelta ha i suoi pro e i suoi contro.

Prendiamo ad esempio il livello applicazione, possiamo dire che:

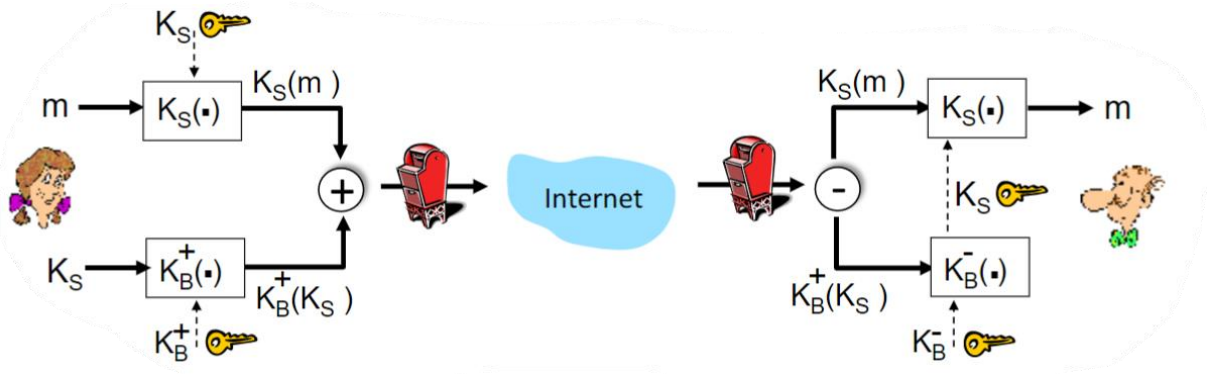
- si ha massima flessibilità nell'implementazione (si ricordi che la definizione di "sicurezza" varia da applicazione ad applicazione), ma
- la complessità è spostata a livello applicazione e riguarderà un'applicazione nello specifico (si deve replicare l'implementazione della sicurezza in ogni possibile applicazione se rimaniamo nel livello applicazione).

Spostarsi nei livelli inferiori della pila protocollare riduce la flessibilità, ma garantisce un'implementazione unica e valida per tutte le implementazioni a livello superiore (ad esempio nel livello applicazione).

9.7.2 Implementazione a livello applicazione: posta elettronica

9.7.2.1 Implementazione della confidenzialità

Supponiamo che per sicurezza intendiamo esclusivamente la confidenzialità, cioè è interesse dello sviluppatore fare in modo che il contenuto dei messaggi non possa essere letto da terzi.



Si adotta il seguente approccio:

- il messaggio è cifrato ricorrendo alla crittografia a chiave simmetrica;
- la chiave simmetrica è scambiata tra i due interlocutori per mezzo della crittografia a chiave pubblica (ricordarsi che la crittografia a chiave pubblica ha un costo computazionale superiore alla crittografia a chiave simmetrica, quindi la utilizziamo per scambiare la chiave simmetrica)

Nell'esempio in figura…

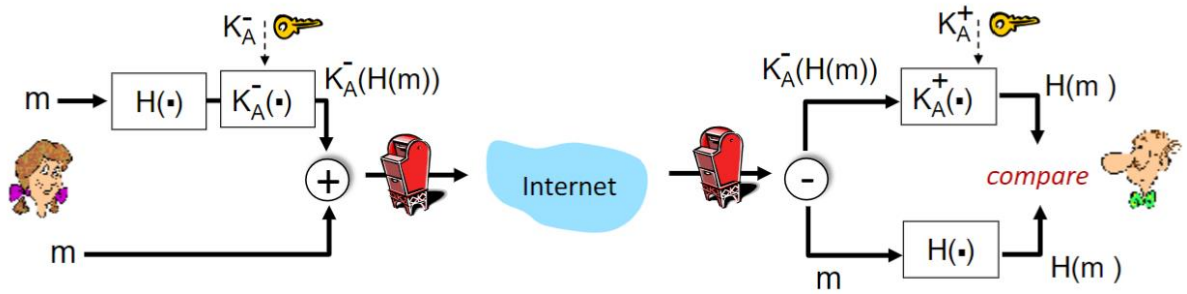
- Alice cifra il messaggio con la chiave simmetrica, e cifra la chiave simmetrica utilizzando la chiave pubblica di Bob. Concatena la chiave cifrata al messaggio cifrato e trasmette in Internet.
- Bob riceve il pacchetto, decifra la chiave simmetrica utilizzando la sua chiave privata e utilizza la chiave simmetrica appena ottenuta per decifrare il messaggio.

9.7.2.2 Implementazione dell'integrità e dell'autenticazione

Supponiamo che per sicurezza intendiamo integrità del messaggio e autenticazione, cioè è interesse dello sviluppatore fare in modo che

- Il contenuto del messaggio non venga alterato;
- sia possibile identificare il mittente del messaggio.

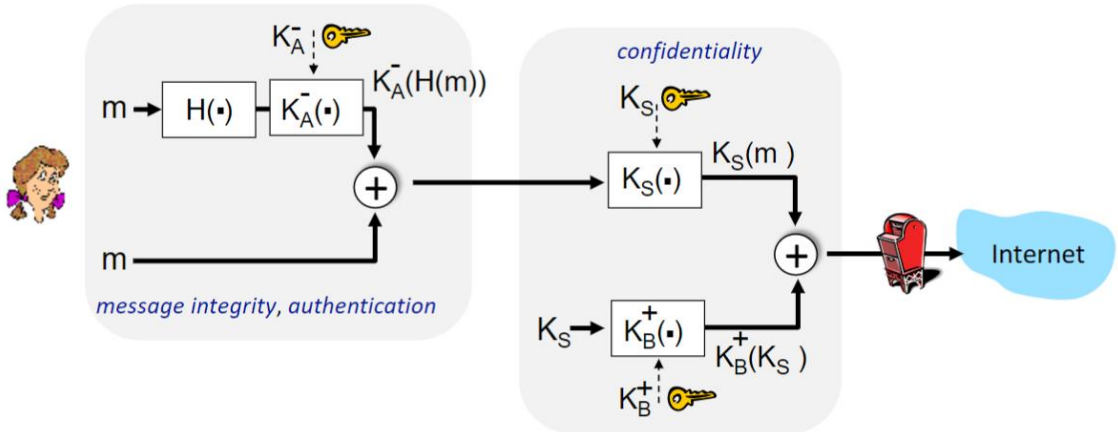
Ricorriamo a funzioni hash (per il digest) e crittografia a chiave pubblica (nei modi già spiegati precedentemente).



- Alice genera il digest del messaggio usando una funzione hash. Cifra il digest ottenuto con la sua chiave privata. Trasmette il messaggio concatenato al digest cifrato.
- Bob riceve il pacchetto. Decifra il digest del messaggio usando la chiave pubblica di Alice. Genera il digest utilizzando il messaggio ricevuto e mette a confronto i due digest: se coincidono il messaggio è integro, ma soprattutto l'identità del mittente è valida (la chiave privata è conosciuta solo da Alice, quindi solo Alice può aver trasmesso il digest cifrato).

9.7.2.3 Unione delle implementazioni precedenti

Nel caso in cui la sicurezza comprenda tutto quanto (riservatezza, integrità e autenticazione) possiamo unire le due implementazioni appena viste.



- Per prima cosa si gestisce integrità e autenticazione, nello stesso modo visto precedentemente.
- Successivamente si gestisce la confidenzialità, ponendo come messaggio cifrato la concatenazione tra messaggio m e digest cifrato.

9.7.3 Implementazione a livello Applicazione: Transport Layer Security (TLS)

Assente

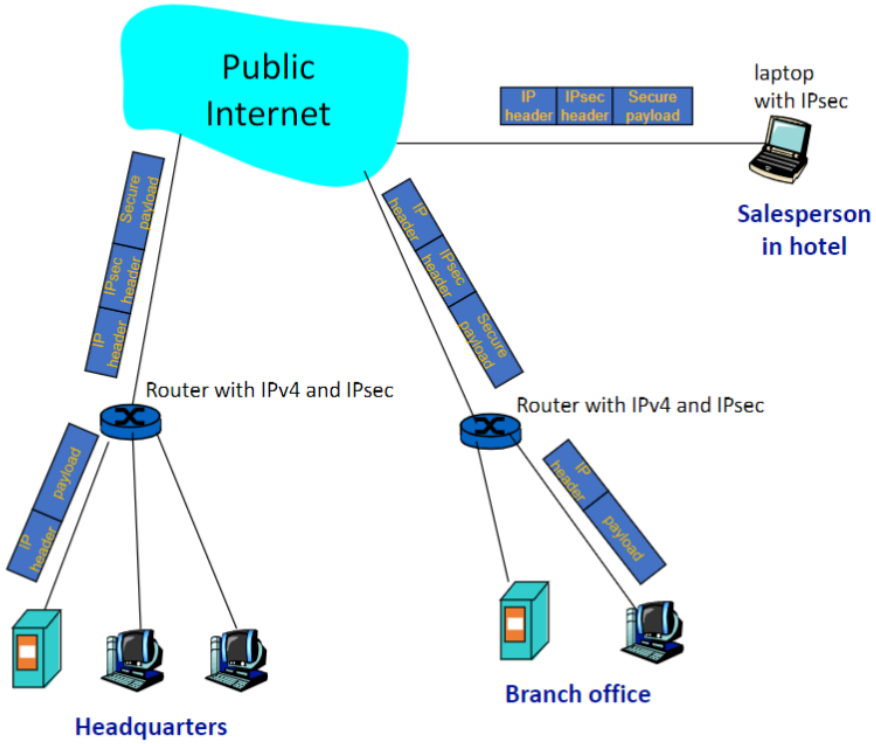
9.7.4 Implementazione a livello Network: protezione dalla rete esterna con IPsec (IP sicuro)

9.7.4.1 Spiegazione

Ogni istituzione ha interesse a comunicare la propria rete a Internet, in modo tale da poter usufruire dei servizi offerti. Questo è anche un rischio (attacchi e incertezza che il contenuto arrivi integro e/o non sniffato a destinazione)!

L'ideale sarebbe avere una Internet privata tutta nostra (*con blackjack e squillo di lusso*, cit. non detta da Anastasi) con i vantaggi della rete internet pubblica. Problemi: non abbiamo il controllo della rete Internet pubblica, e all'interno di essa circolano messaggi non relativi all'organizzazione.

Introduciamo la *Virtual Private Network* (VPN), realizzata a partire dalla rete Internet pubblica. Finchè la comunicazione avviene all'interno della rete dell'organizzazione non si hanno problemi. Può succedere che per la struttura dell'organizzazione sia necessario attraversare per un tratto la rete pubblica, nel passaggio da un nodo dell'organizzazione a un nodo di un'altra organizzazione.

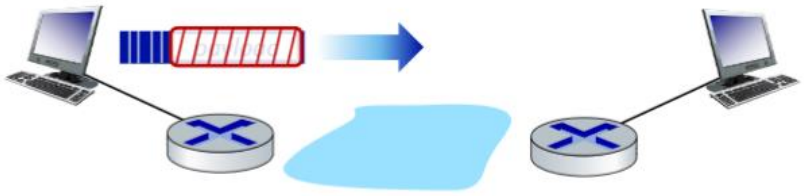


Nella VPN è prevista la cifratura dei pacchetti mentre questi passano dalla rete pubblica. Due possibili protocollari per attuare l'IPsec (Protocollo IP sicuro):

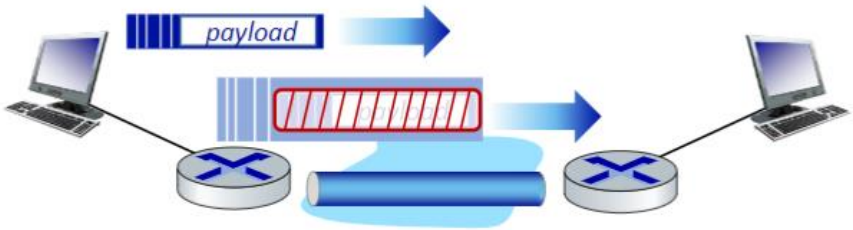
- *Authentication header* (AH, fornisce *data integrity, origin authentication*)
- *Encapsulation Security Protocol* (ESP, fornisce i due già citati e la *confidentiality*).

Ci interessa ESP, il più completo.

- Protocollo connection-oriented (in contrasto con IP che è *connectionless*): si stabilisce una security association tra due end point (connessione simplex, in una sola direzione). Due modalità:
    - ***Transport mode***
      Pacchetti posti fin da subito (fin dal mittente) in formato IPsec, e vengono ricevuti in formato IPsec dal destinatario. La security association è stabilita tra host mittente e host destinatario.



- ***Tunnel mode***
  I pacchetti vengono posti nel formato IPsec solo quando raggiungono il border router, e vengono riportati al formato originario dopo aver attraversato la rete pubblica. La security association è stabilita tra i due border router.

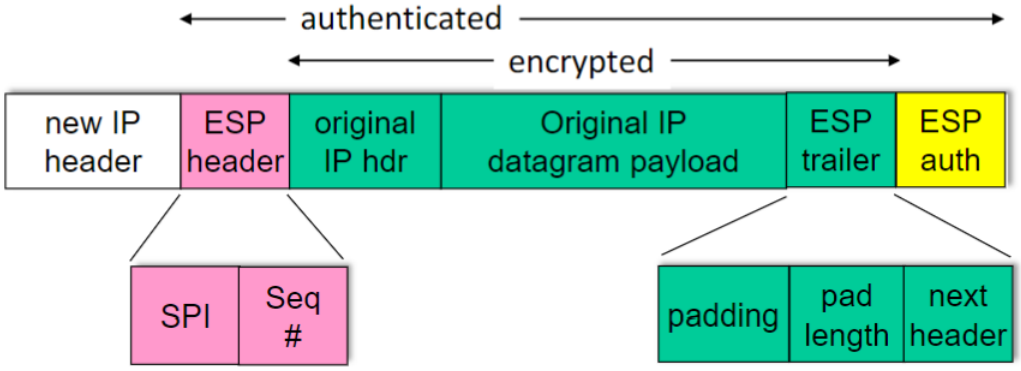


- Si inizializzano strutture dati adeguate nei due end point. Tra le cose che memorizziamo abbiamo:
  - identificatore da 32 bit della security association, denominato Security Parameter Index (SPI);
  - indirizzo di origine della security association, così come l'indirizzo destinatario (quali siano gli indirizzi dipende dalla modalità adottata nel protocollo – tunnel o transport mode);
  - tipo di cifratura e tipologia di controllo dell'integrità (algoritmo utilizzato per il MAC);
  - chiave di cifratura e chiave di autenticazione.

  Le informazioni sono mantenute all'interno di ***Security Association Database*** (SAD).
- Si ha anche un ***Security Policy Database***, che segnala per ogni datagram se è necessario utilizzare IPsec o se può essere trattato come un datagram tradizionale. Se si deve usare IPsec è segnalata la relativa *security association*.

9.7.4.2 Formato IPsec

Consideriamo il formato del pacchetto IPsec, ponendoci in tunneling mode.



Abbiamo già anticipato che il datagram inizialmente è posto nel formato IP tradizionale, e successivamente incapsulato all'interno del formato IPsec (quando il datagram raggiunge il border router).

- Original IP datagram e Original IP header consistono insieme nel datagram originale.
- ESP trailer permette di gestire una cifratura orientata al blocco: la dimensione complessiva deve essere multiplo della dimensione del blocco, ergo è solitamente necessario aggiungere dei bit di padding. Con pad length si segnala la dimensione del padding (necessario, dato che rimuoveremo il padding quando ripristineremo il formato IP tradizionale).
- I valori detti prima costituiscono la parte cifrata del datagram IPsec, cifrata con chiave e algoritmi stabiliti nelle appropriate strutture dati (citate precedentemente).
- Si aggiunge un ESP header per indicare:
  - Il Security Parameter Index, identificativo della Security Association;

    - Un numero di sequenza per prevenire attacchi *record and playback*
- Si gestisce l'autenticazione con MAC, generando l'ESP auth.

Si pone, infine, un new IP header che presenta la stessa struttura dell'header del formato IP tradizionale. Questo perché si vuole impedire a chi sta nella rete pubblica di distinguere un datagram in formato IP da un datagram in formato IPsec.

ESP tunnel mode: actions

at R1:

- appends ESP trailer to original datagram (which includes original header fields!)
- encrypts result using algorithm & key specified by SA
- appends ESP header to front of this encrypted quantity
- creates authentication MAC using algorithm and key specified in SA
- appends MAC forming *payload*
- creates new IP header, new IP header fields, addresses to tunnel endpoint

IPsec sequence numbers

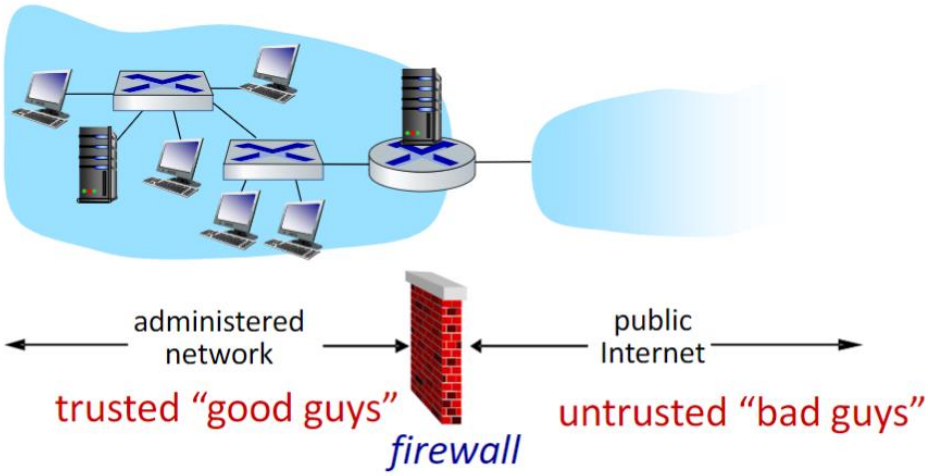
- for new SA, sender initializes seq. # to 0
- each time datagram is sent on SA:
  - sender increments seq # counter
  - places value in seq # field
- goal:
  - prevent attacker from sniffing and replaying a packet
  - receipt of duplicate, authenticated IP packets may disrupt service
- method:
  - destination checks for duplicates
  - doesn't keep track of *all* received packets; instead uses a window

9.7.5 Implementazione a livello Network: protezione della rete interna

La nostra organizzazione è caratterizzata da host e reti. La IPsec vista precedentemente permette l'invio di datagram verso l'esterno, proteggendo i datagram nel passaggio dalla rete pubblica (che è per definizione insicura). Fino ad ora abbiamo parlato della pericolosità della rete pubblica, ma non abbiamo detto nulla sulla rete interna, che è tutt'altro che sicura (potremo avere malintenzionati, provenienti dalla rete, pubblica, che cercano di "intrufolarsi" nella nostra rete).

9.7.5.1 Firewall

Introduciamo il firewall, che permette "l'isolamento" della rete interna da Internet tutto, stabilendo quali pacchetti possono transitare e quali devono essere bloccati. La cosa può essere immaginata alle mure difensive di una città, e alle guardie che stanno ad ogni porta controllando i passanti.



Il firewall ci piace perché:

- si prevengono attacchi di tipo Denial of Service;
- si prevengono accessi illegali consentendo l'accesso alla rete interna soltanto a chi ne ha diritto.

La porta di accesso alla rete è un router di frontiera, dove si ha l'implementazione del firewall (si vedano le lezioni di Pistolesi per avere un'idea chiara, non ho ribadito alcune cose dette da Pistolesi). I firewall che a noi interessano sono i cosiddetti *firewall a filtraggio di pacchetto*, che possono essere implementati secondo i seguenti approcci: *stateless, statefull e application gateway*.

- ***Stateless***.
  Si analizza pacchetto per pacchetto a livello network, ponendo criteri e target per mezzo di una tabella nota come *Access Control Lists* (CL).
  - Il target consiste nell'azione che viene eseguita rispetto a quel pacchetto: consentire il passaggio o bloccarlo.
  - I criteri sono le caratteristiche che deve avere un particolare pacchetto affinchè si applichi il target. Si parla di valori come indirizzo IP sorgente, IP destinatario, porta sorgente, porta del destinatario, presenza di TCP SYN e ACK bits, ICMP message type.

  Si scorrono le regole presenti nella tabella dall'alto verso il basso, e si applica la prima dove i criteri corrispondono. Esempi di politiche attuabili:
  - **Esempio 1**.
    Bloccare tutto il traffico in ingresso e in uscita per i pacchetti che hanno IP protocol (protocollo del livello superiore) è uguale a 17, con la porta sorgente o destinatario uguale a 23. Questo significa bloccare il traffico UDP e le connessioni telnet.
  - **Esempio 2**.
    Bloccare i segmenti TCP in ingresso col campo ACK uguale a zero. Si impedisce ad utenti esterni di aprire connessioni TCP, ma si permette ad utenti interni di aprire connessioni TCP verso l'esterno.

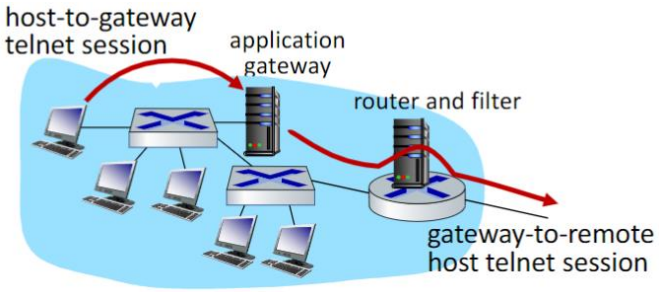
  Segue un esempio estremamente semplificato di ACL

action	source address	dest address	protocol	source port	dest port	flag bit
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----
deny	all	all	all	all	all	all

  Si osservi che la default rule decisa (cioè la regola che si applica quando non si ha corrispondenza con nessuno dei criteri presenti nella ACL) prevede il rifiuto del pacchetto.
- ***Statefull***.
  Si estende l'approccio stateless considerando le connessioni attive (ad esempio se è stata stabilita una connessione TCP). Questo significa escludere, ad esempio, pacchetti che non hanno senso dal punto di vista del protocollo (Es: SYN ACK che non può essere ricevuto a connessione TCP aperta). Segue estensione dell'esempio di ACL visto prima: si stabilisce se è necessario controllare lo stato della connessione prima di ammettere un pacchetto.

action	source address	dest address	proto	source port	dest port	flag bit	check connection
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any	
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK	X
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---	
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----	X
deny	all	all	all	all	all	all	

- ***Application gateway***.
  Non si opera più a livello Network, ma a livello applicazione (eccezione rispetto alle cose citate in questa sezione della dispensa).



  La richiesta di connessione viene deviata all'application gateway (operazione trasparente, l'utente non se ne accorge), che verifica se ci sono le condizioni per far passare il pacchetto.
  Si ha una visione maggiore del contesto: nei metodi precedenti si prendono decisioni a livello di pacchetto, in questo caso si possono assumere decisioni rispetto all'utente (ad esempio decidere che un utente può trasmettere pacchetti e un altro no).

Il firewall è uno strumento potente, ma non sufficiente: il problema dell'IP spoofing, ad esempio, non è risolto (non si accorge da solo se i pacchetti sono stati manipolati). **RIVEDERE DIAPOSITIVA 88**

9.7.5.2 Intrusion Detection System (IDS)

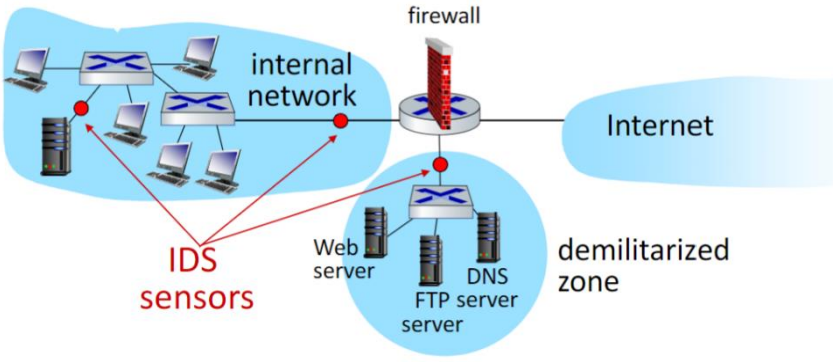
Ulteriore strumento a nostra disposizione è l'Intrusion Detection System. Esso consiste in un sistema che analizza i pacchetti con maggiore profondità, controllando:

- contenuto del pacchetto (se, ad esempio, sono presenti caratteri particolari all'interno del messaggio)
- sequenza dei pacchetti e correlazione tra questi.

L'analisi di queste cose permette l'individuazione di: port scanning, network mapping e attacchi DoS.

9.7.5.3 Uso congiunto dei due strumenti

Firewall e IDS sono utilizzati insieme. Si guardi la figura:



- abbiamo il firewall nel router di accesso;
- la rete interna è divisa in zona demilitarizzata (dove non si attuano fin troppe restrizioni, in alcuni casi l'accesso non solo è utile ma anche consigliato – si pensi a chi offre servizi di web server) e rete interna propriamente detta;
- si pongono più IDS all'interno della rete perché non è sufficiente un solo IDS per lo studio dei pacchetti (servono costantemente nuove informazioni).

10 RETI WIRELESS E MOBILI

10.1 Introduzione: le reti wireless e la diffusione maggiore di dispositivi

Per larga parte del corso abbiamo considerato in modo implicito che i collegamenti tra le reti fossero di tipo wired, ma sappiamo che esistono reti di tipo Wireless. Negli ultimi anni queste reti si sono sviluppate enormemente favorendone la diffusione, portando quindi a un uso maggiore di dispositivi portatili e/o indossabili. Si prevede che questo trend aumenterà sempre di più, tenendo conto di quanto detto a inizio corso sull'Internet of Things.

Mark Weiser prevedeva negli anni 90 l'*ubiquitous computing*, cioè i dispositivi saranno pervasivi al punto da non essere più notati.

> Specialized elements of hardware and software, connected by wires, radio waves and infrared, will so ubiquitous that no one will notice their presence
>
> The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.
>
> Consider writing, perhaps the first information technology. […] Today this technology is ubiquitous in industrialized countries. Not only do books, magazines and newspapers convey written information, but so do street signs, billboards, shop signs and even graffiti.
>
> The constant background presence of these products of "literacy technology" does not require active attention, but the information to be conveyed is ready for use at a glance. It is difficult to imagine modern life otherwise.

10.2 Caratteristiche delle reti Wireless

10.2.1 Differenze tra reti wireless e reti wired

I collegamenti wireless sono un po' diversi da quelli wired.

Pila protocollare
Appplication *messages*
Transport *segments*
Network *datagrams*
Link / Datalink *frames*
Physical

- **Degradazione del segnale**.
  La potenza del segnale emesso si degrada all'aumentare della distanza (*path loss*): fenomeno già visto nelle reti wired, ma presente con maggiore rilevanza nel contesto delle reti wireless (con degradazione avente andamento d^2, o addirittura potenza maggiore).
- **Maggiori interferenze**.
  La comunicazione wireless è maggiormente soggetta ad interferenze.
- **Propagazione multidirezionale**.
  Abbiamo già detto a inizio corso che le reti wireless non si basano su link guidati: questo significa che un segnale wireless può propagarsi in più direzioni. Questi segnali possono essere alterati in direzione e velocità (ad esempio a causa di un ostacolo, un muro). Il destinatario del segnale non riceve un solo segnale, ma più segnali provenienti dalla stessa fonte: l'esito può essere un aumento della potenza del segnale, ma anche una diminuzione a causa dello sfasamento dei segnali.

Dati certi bit quale sarà la percentuale di bit alterati? Il *bit error rate* dipende:

- da quanto il canale è disturbato;
- dalla codifica adottata.

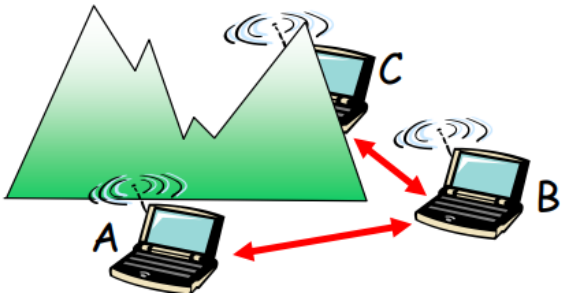
Esistono particolari codifiche del segnale che garantiscono un bit error rate minore, ma allo stesso tempo un bit rate inferiore. Altre codifiche hanno un bit rate più elevato, ma un error bit rate più elevato. Cosa ci conviene? L'utente vuole velocità elevate, ma è chiaro che con bit error rate elevato si avranno retransmission più frequenti. La soluzione è una politica adattiva: in base alle caratteristiche del canale si sceglie una particolare codifica (privilegiando ove possibile una codifica con bit rate maggiore).

10.2.2 Problema del nodo nascosto

Normalmente nella comunicazione wired in presenza di un mezzo broadcast tutti i dispositivi sentono la stessa cosa. In Ethernet:

- tutti ricevono i pacchetti;
- tutti si accorgono, prima o poi, di eventuali collisioni.

Le cose dette non valgono nelle reti Wireless: introduciamo quello che è noto come **problema del nodo nascosto**.



- Supponiamo di avere un nodo A che vuole trasmettere a un nodo B.
- Supponiamo di avere un ulteriore nodo C che vuole trasmettere al nodo B.

In presenza di ostacoli di dimensioni rilevanti il nodo A e il nodo C sono l'uno nascosto rispetto all'altro: A trasmette a B, ma non si accorge che C sta scrivendo a B (e viceversa).

Quali sono le conseguenze? Possibilità di collisioni di cui però i nodi non si accorgono.

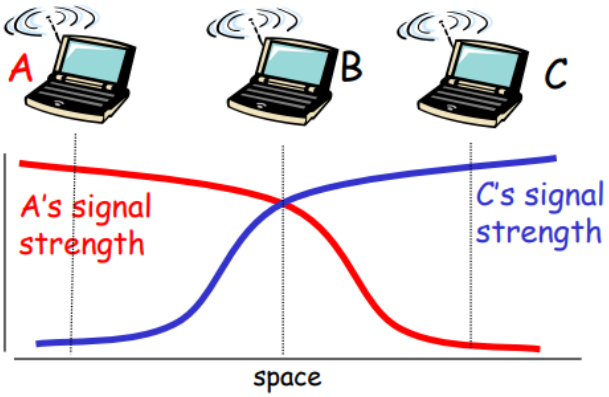
- A vuole trasmettere a B, C vuole trasmettere a B
- Il nodo A ascolta il mezzo e per il problema descritto crede che non stia trasmettendo nessun'altro.
- Il nodo C ascolta il mezzo e per il problema descritto crede che non stia trasmettendo nessun'altro.
- Segue che entrambi trasmettono il segnale e c'è sovrapposizione senza individuazione di collisione.

10.2.3 path loss: un'altra manifestazione del problema del nodo nascosto

Abbiamo già parlato brevemente di *path loss*. Riprendiamo l'esempio di prima.

- Supponiamo di avere un nodo A che vuole trasmettere a un nodo B.
- Supponiamo di avere un ulteriore nodo C che vuole trasmettere al nodo B.

Non abbiamo ostacoli tra il nodo A e il nodo C, ma questi sono posti a una certa distanza (immaginiamo che C sia più lontano). Supponiamo che A e C trasmettano un segnale a B: il nodo B riceve bene il segnale del nodo A perché è poco distante, ma il segnale del nodo C viene ricevuto a potenza molto bassa.

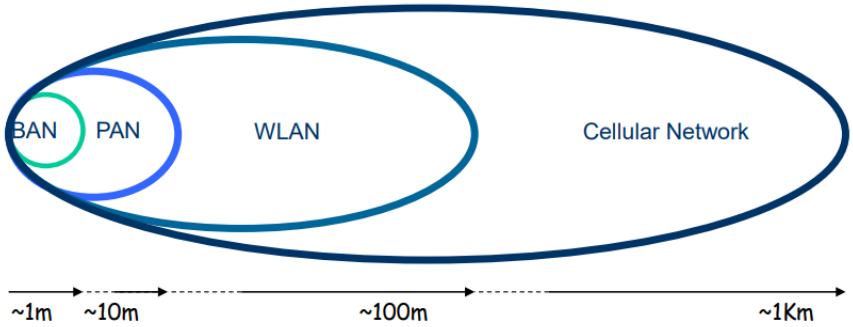


La potenza del segnale ricevuto degrada velocemente all'aumentare delle distanze, fino ad assumere un livello tale da impedire distinzione tra un rumore di fondo e un segnale effettivamente trasmesso da un dispositivo. Le stesse considerazioni valgono per un segnale trasmesso da C. Questo porta nuovamente al problema del nodo nascosto perché la degradazione del segnale impedisce l'individuazione di una collisione tra segnali.

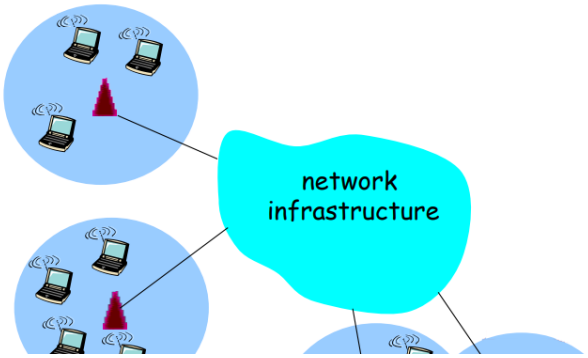
10.2.4 Classificazione delle reti wireless: copertura, infrastruttura, hop

La reti Wireless possono essere classificate tenendo conto dei seguenti aspetti:

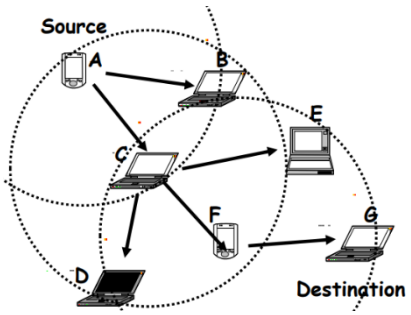
- **Area di copertura**
  - Reti cellulare (raggio di copertura di qualche chilometro)
  - WLAN (Wireless LAN, reti locali)
  - WPAN (Wireless Personal Area Network, interconnessione di dispositivi ad uso personale – ad esempio i dispositivi disposti su una scrivania, nell'ottica di ridurre i cavi presenti)
  - BAN (Body Area Network, interconnessione di dispositivi indossabili, raggio di copertura non superiore al metro)



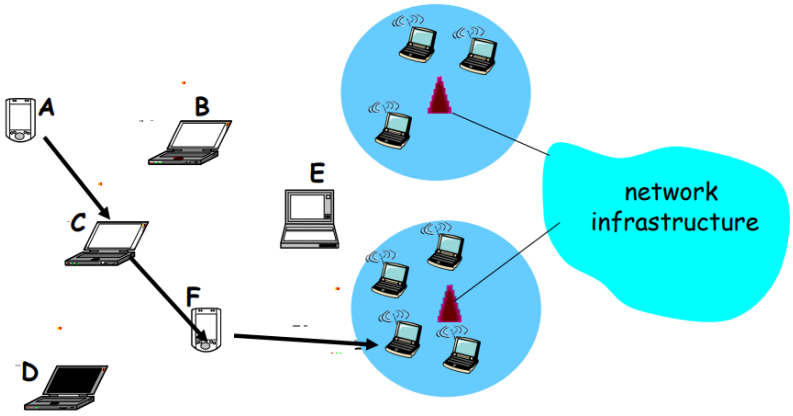
- **Presenza di infrastrutture tipiche della rete fissa**
  - reti infrastructure-based
    - Reti cellulare, basate sulla presenza di Base station che offrono il servizio di connettività ai dispositivi mobili
    - Reti Wifi, basate su infrastrutture di linea fissa che terminano con degli Access Point che offrono il servizio di connettività Wireless ai dispositivi nelle aree coperte.



  - reti infrastructure-less
    - I nodi comunicano tra di loro solo perché dotati di interfaccia Wireless.
    - Un esempio è la comunicazione Bluetooth tra dispositivi. Un altro esempio sono le reti utilizzate dalla Protezione Civile (chi soccorre non può pensare di usare la rete cellulare o una rete Wifi, in generale la Protezione Civile interviene a seguito di calamità naturali che potrebbero aver compromesso questi servizi)
    - Si distinguono
      - reti single-hop
      - reti multi-hop, equivalente in un certo senso al query flooding (segnale trasmesso ai vicini, che a loro volta trasmetteranno ai loro vicini) o in generale all'approccio di Internet (passaggio dei pacchetti dai router fino a quando il pacchetto non è arrivato a destinazione).



- forme ibride, una parte senza infrastruttura e una parte con infrastruttura



Con le cose dette fino ad ora possiamo introdurre una tassonomia con le principali reti.

	single hop	multiple hops
infrastructure (e.g., APs)	host connects to base station (WiFi, WiMAX, cellular) which connects to larger Internet	host may have to relay through several wireless nodes to connect to larger Internet: *mesh nets, sensor nets*
no infrastructure	no base station, no connection to larger Internet (Bluetooth, ad hoc nets)	no base station, no connection to larger Internet. May have to relay to reach other a given wireless node MANET, VANET

Nelle seguenti pagine ci concentreremo sulle reti con infrastruttura single-hop, mentre reti senza infrastruttura (in particolare quelle multiple-hop) saranno oggetto del corso di Internet of Things alla magistrale.

10.3 Reti WiFi (o reti WLAN, Wireless Lan)

10.3.1 Caratteristiche

WiFi è acronimo di Wireless Fidelity. WiFi è anche un consorzio di case costruttrici che ha lo scopo di promuovere la diffusione di questa tecnologia. Si ha uno standard IEEE alla base di questa tecnologia.

- Presente infrastruttura di rete fissa (tipicamente Ethernet) a cui sono collegati degli Access Point.
- Ogni Access Point è un bridge, un dispositivo con due interfacce: da una parte l'interfaccia Ethernet (che permette al dispositivo di comunicare con l'infrastruttura di rete fissa), dall'altra l'interfaccia Wifi (con cui offre il servizio di connettività ai dispositivi all'interno dell'area di copertura)
- Si definisce l'area di copertura di un Access Point con la dicitura *Basic Service Set* (BSS): ogni Access Point ha il suo BSS.
- In alcuni casi il dispositivo è in grado di ricevere il segnale di più Access Point:
  - si ha una procedura di selezione dove il nodo scansiona i canali disponibili, scorre le frequenze disponibili e verifica se arrivano segnali;
  - l'Access Point trasmette periodicamente il cosiddetto segnale di Beacon, pacchetto speciale contenente il SSID (identificatore dell'Access Point);
  - se non riceve Beacon scarta il canale;
  - se il nodo riceve il pacchetto Beacon su più canali si sceglie il canale che ha trasmesso il segnale con potenza maggiore.
- Access Point vicini devono operare su frequenze differenti, in modo da non interferire.

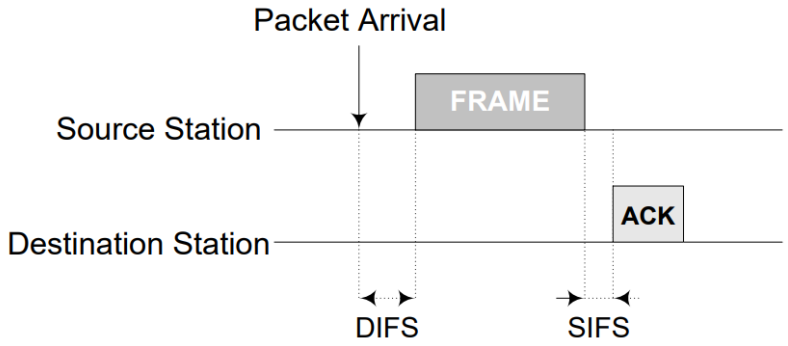
10.3.2 Protocollo CSMA/CA

La comunicazione è di tipo broadcast: quando un nodo trasmette tutti i nodi presenti sentono. Avendo un mezzo condiviso non è possibile avere una trasmissione simultanea: serve un protocollo di accesso. L'idea iniziale sarebbe utilizzare CSMA/CD visto per le reti Ethernet, ma non è possibile!

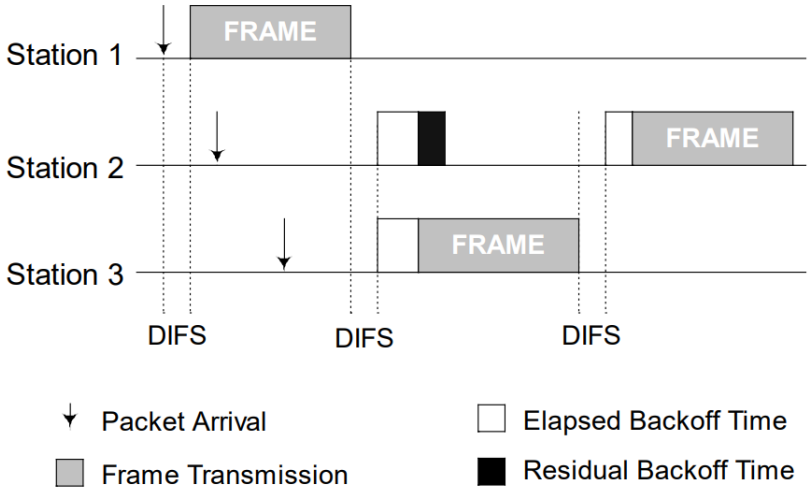
- **Problemino (superabile)**.
  Il nodo con interfaccia Wifi è dotato di una sola antenna (se l'antenna trasmette non può ricevere allo stesso tempo). Si risolve mettendo due antenne invece di una.
- **Problema del nodo nascosto**.
  Le collisioni non possono essere rilevate. Possibile verifica solo alla fine della trasmissione, con invio dell'ACK (protocollo ALOHA?): aumenta il tempo, non ci piace questa soluzione.

Si vuole prevenire le collisioni. Il protocollo proposto è CSMA/CA, dove CA sta per *Collision Avoidance* (in contrasto con *Collision Detection* di CSMA/CD).

- Nonostante il nome del protocollo le collisioni si verificano. Dobbiamo essere in grado di individuarle al termine della trasmissione.
- Il nodo destinatario invia un ACK, come in ALOHA. Approccio conservativo: se non si sa nulla si suppone che ci sia stata collisione
- Tempo diviso in slot, ma non nel senso di Slotted ALOHA (in realtà si potrebbe dire che il tempo è continuo). Gli slot sono di piccola dimensione e hanno come unico scopo quello di stabilire l'inizio delle operazioni di trasmissione. Si consideri la seguente figura



  - Il nodo sorgente riceve da livello superiore (*Packet arrival*) un pacchetto da trasmettere. Ascolta il mezzo, che deve essere libero per un intervallo detto DIFS. Passato un tempo DIFS si trasmette il frame.
  - Finita la trasmissione del frame il nodo destinatario switcha da modalità ricezione a modalità trasmissione (aumento dei tempi). Attende un intervallo detto SIFS, che deve essere strettamente inferiore a DIFS, per dare priorità al nodo destinatario che trasmette l'ACK (se fossero uguali si potrebbe avere collisione tra trasmissione di nodi terzi e l'ACK)
- L'esempio precedente assume che un nodo, quando riceve il pacchetto da livello superiore, ascolta il mezzo e lo trova libero: non è assolutamente detto. Consideriamo la seguente figura



- I tre nodi presenti in figura ricevono in istanti diversi il segnale da trasmettere. Il nodo 1 è il primo a riceverlo, e inizia la trasmissione nei modi detti. Gli altri due nodi ricevono il segnale mentre il canale è occupato (per la trasmissione del frame da parte del nodo 1).
- Per comodità la figura non include gli ACK, che vengono inviati nei modi spiegati prima.
- Ogni nodo che trova il canale occupato si pone in attesa e genera in modo casuale un tempo di backoff (espresso in numero di slot).
- Terminata la trasmissione del frame precedente i nodi che sono rimasti fino ad ora tentano l'invio del loro segnale superato l'*intervallo di backoff* (misurano la cosa con un timer). "Vince" il primo che trasmette, cioè il nodo che ha randomicamente generato l'intervallo di backoff più piccolo.
- Ogni volta che il canale viene nuovamente occupato tutti i nodi in attesa mettono in pausa il loro timer, e lo riavviano quando si accorgono che il canale è stato liberato.

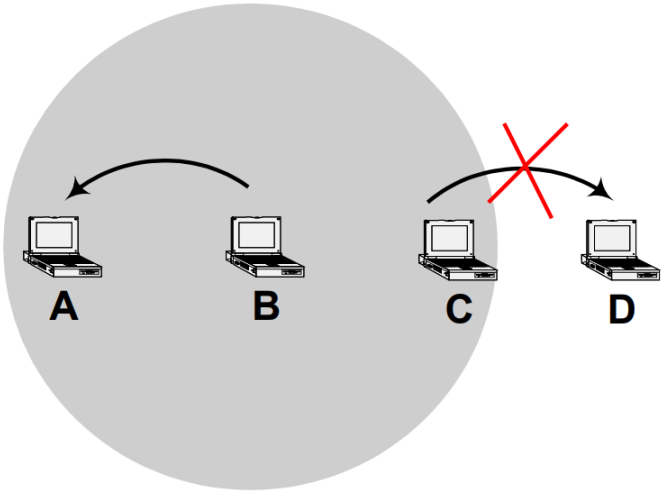
- **Come viene scelto l'intervallo di backoff?**

L'approccio è simile a quello di Ethernet, ma con differenze (occhio a non confondere all'esame).

- Si sceglie un valore iniziale appartenente all'intervallo $[0; \mathrm{CW} - 1]$, dove CW sta per Contention Window.
- Inizialmente $CW = CWmin$
- Nel caso in cui ci siano collisioni (missed ACK) si pone $\mathrm{CW} = 2 \cdot \mathrm{CW}$
- Si fa così fino a quando $\mathrm{CW} = \mathrm{CWmax}$
- CWmin e CWmax sono valori dipendenti dal livello fisico.

Possibili collisioni?

- Se due nodi generano lo stesso valore dell'intervallo $[0; \mathrm{CW} - 1]$
- **Problema del nodo nascosto**.
  Può verificarsi il problema del nodo nascosto, di cui abbiamo già parlato.
- **Problema del nodo esposto**.
  Può verificarsi il problema del nodo esposto! Immaginiamoci la seguente situazione dove la circonferenza rappresenta il raggio di trasmissione del nodo B



Se rispettiamo il protocollo precedentemente spiegato C non trasmette subito a D, in quanto il vanale è occupato dal nodo B che sta trasmettendo un segnale al nodo A. Si consideri che: A è sicuramente fuori dal raggio di trasmissione di C, B sta trasmettendo (il problema ci sarebbe se ricevesse). Morale della favola: non c'è interferenza se trasmette, ma non lo fa per il protocollo CSMA/CA.

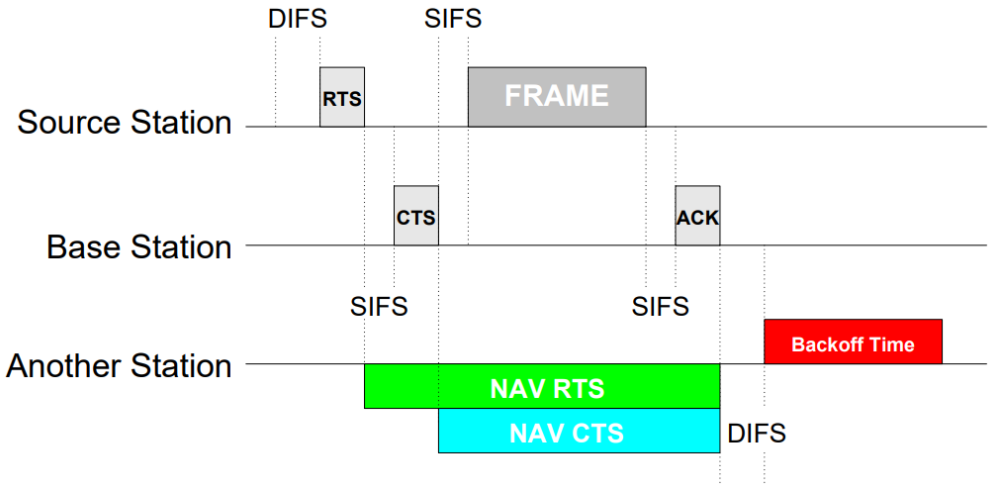
Tra i due problemi è sicuramente più critico il primo, perché ci si accorge della collisione solo alla fine.

10.3.3 *Virtual Carrier Sensing* per alleviare il problema del nodo nascosto

Si distinguono due Carrier Sensing:

- il Physical Carrier Sensing, che è quanto spiegato fino ad ora (analisi del canale prendendo a riferimento la potenza dei segnali per individuare se un canale è occupato o meno);
- il Virtual Carrier Sensing, che è quanto diremo a breve.

Il Physical Carrier Sensing è soggetto al problema del nodo nascosto, che noi vogliamo mitigare. Il Virtual Carrier Sensing si basa sull'idea di segnalare ai nodi vicini che il canale sarà occupato per una certa durata: si fa ciò introducendo ulteriori slot. Supponiamo che *Source Station* abbia un frame da trasmettere a *Base Station*



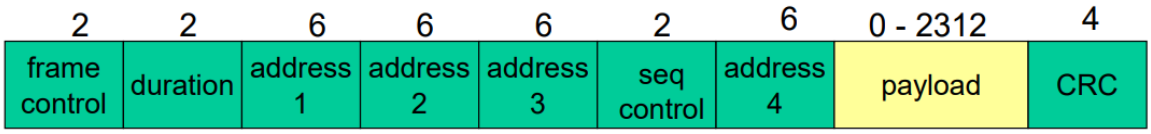
- Anche qua si hanno le semplificazioni grafiche già incontrate (non mostra la trasmissione ACK).
- ***Source Station***, dopo aver atteso un tempo DIFS, non invia subito il frame, ma trasmette un pacchetto RTS (*Request To Send*). Questo pacchetto contiene al suo interno un campo *Duration*, con cui si segnala la durata della trasmissione corrente (tempo che va dalla fine della trasmissione di RTS alla fine della trasmissione dell'ACK). La durata è facile da calcolare per Source Station: conosce le dimensioni di RTS, degli intervalli DIFS e SIFS e del frame che vuole trasmettere.
    - RTS sarà trasmesso al destinatario del frame, Base Station
    - RTS sarà trasmesso anche a tutti i vicini di Source Station, dato che la comunicazione Wifi è una comunicazione broadcast. Tutti i vicini ricevono, di conseguenza, il campo Duration.
- ***Base Station***, ricevuto RTS, invia un pacchetto CTS (*Clear To Send*). Questo pacchetto contiene un campo Duration in linea con RTS (ridotto dato che è passato un po' di tempo) e ha una valenza duplice:
    - viene inviato al mittente Source Station come ACK
    - viene visto anche dai nodi vicini a Base Station (la cosa ci piace perché i vicini di Base Station potrebbero essere quelli che non vedono Source Station)
- I nodi che ricevono RTS e CTS attivano un timer NAV RTS (o NAV CTS, dove NAV sta per ***Network Allocation Vector***). A questo punto abbiamo ciò che volevamo: invece di verificare se il canale è libero si controlla se è attivo il NAV, se lo è non trasmettiamo (il canale potrebbe risultare libero, ma si andrebbe incontro a una collisione per il problema dei nodi nascosti).
- Tutti i nodi che si pongono in attesa calcolano l'intervallo di backoff e si comportano in modo non diverso da quanto già spiegato.

Le collisioni non sono evitate del tutto: se due nodi trasmettono RTS contemporaneamente si ha collisione! La buona notizia è che RTS ha una piccola dimensione, quindi la collisione sarebbe di durata decisamente inferiore rispetto a una collisione tra frame veri e propri.

Qual è il costo? Riduzione del throughput dato che si trasmettono RTS e CTS.

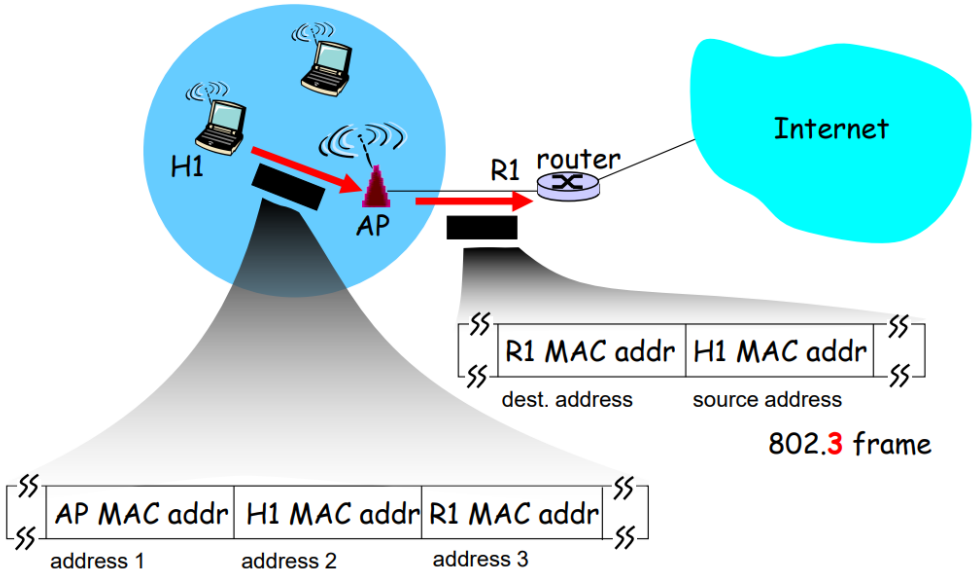
10.3.4 Frame Wifi

Il frame in Wifi risulta essere un po' più complesso rispetto a quello visto per Ethernet.



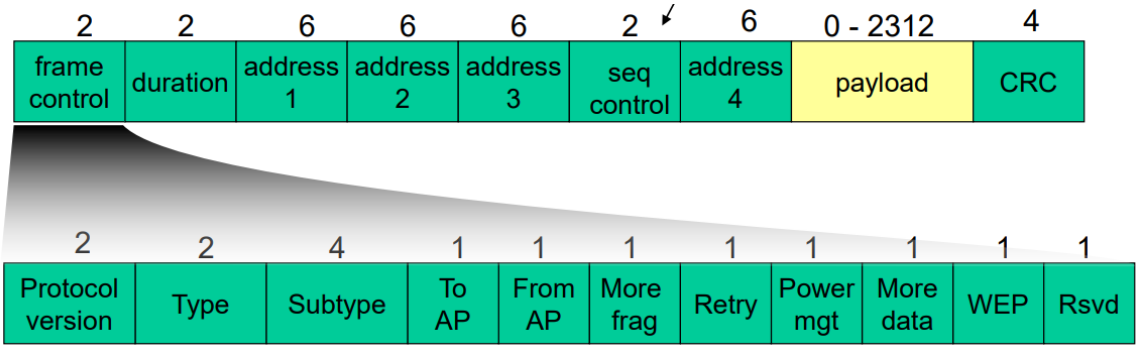
- Il campo *Frame control* contiene informazioni di controllo
- Abbiamo già parlato del campo *Duration*
- Abbiamo quattro campi relativi agli indirizzi:
  - *Address 1* è l'indirizzo MAC del destinatario
  - *Address 2* è l'indirizzo MAC del mittente
  - *Address 3* è l'indirizzo MAC dell'interfaccia router a cui l'Access Point è collegato (ricordarsi che l'Access Point è collegato a un'infrastruttura di rete fissa, l'indirizzo MAC è relativo all'ultimo HOP del sistema)
  - *Address 4* è un indirizzo che ci interessa poco (utile solo in modalità ad hoc – quando due dispositivi dialogano tra di loro direttamente, evitando la base station).

  Si consideri la seguente figura per avere le idee più chiare sugli indirizzi 1, 2 e 3. Aspetto rilevante è la trasformazione del frame Wifi in frame Ethernet dopo il passaggio dall'Access Point: l'indirizzo mittente rimane lo stesso (quello del dispositivo che si è connesso con la Wifi all'Access Point), ma cambia l'indirizzo destinatario (che diventa quello del router)



- *seq control* indica un numero di sequenza (per cose che possiamo già immaginarci)

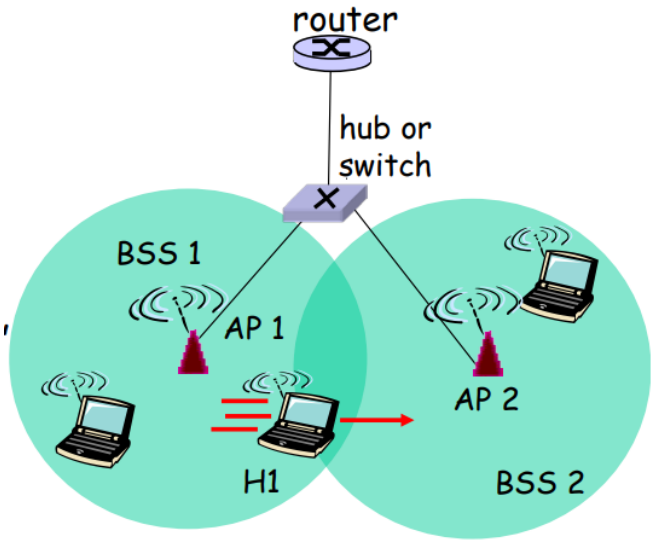
A proposito del campo Frame control abbiamo i seguenti valori:



- *Protocol version* è la versione del protocollo;
- *Type* è il tipo di frame (RTS, CTS, ACK o Data)
- *To AP* segnala se il frame è diretto verso un Access Point o no.

10.3.5 Mobilità dei dispositivi connessi alla Wifi

La Wifi permette la mobilità dei dispositivi connessi alla stessa, a patto che si rimanga all'interno dei raggi di trasmissione dei vari Access Point.



- **Il dispositivo H1 si sposta, ma rimane nel raggio di trasmissione di AP1**.
  Non è un problema perché il dispositivo comunica con AP1 finchè rimane nel raggio di trasmissione. Dall'altro lato gli switch presenteranno nelle tabelle di forwarding informazioni che condurranno i pacchetti diretti ad H1 verso AP1.
- **Il dispositivo H1 si sposta nel raggio di trasmissione di AP2**.
  Nel caso in cui ci si sposti verso AP2 interviene la procedura di associazione agli Access Point già affrontata: invio dei segnali di Beacon e scelta dell'Access Point in grado di trasmettere il precedente segnale con potenza maggiore. Lo switch (che è self-learning) non recepirà immediatamente questa cosa, ma rimuoverà il relativo record dalla tabella di forwarding superato il *time to leave* (ttl).

10.3.6 Power Management

Gli host stazionari sono collegati alla rete elettrica, mentre in presenza di host mobili e collegamenti alla Wifi si pone un problema energetico: la batteria ha un budget limitato (in funzione anche della sua dimensione, è chiaro che le batterie enorme di un'automobile ha una capacità maggiore rispetto alla batteria piccola di un telefono).

Lo standard 802.11 incorpora un meccanismo di *Power management* dove i nodi possono "andare a dormire" e risparmiare energia. Questione:

- se il nodo dormiente vuole trasmettere un pacchetto non ci sono problemi, si sveglia e trasmette il pacchetto;
- se il nodo dormiente deve ricevere pacchetti è un problema, chi dorme non piglia pesci!

L'ideale sarebbe dormire e pigliare pesci (cit.). Come si risolve? I nodi si affidano agli Access Point:

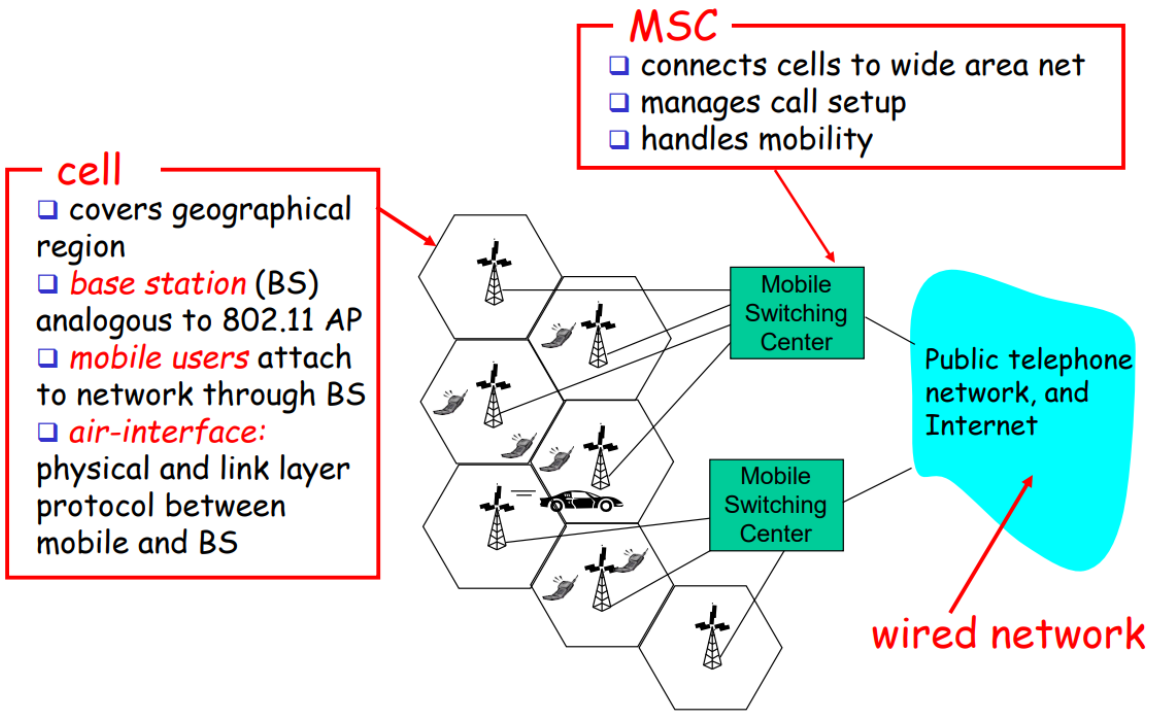
- Sappiamo che l'Access Point emette un Beacon ogni 100 ms contenente informazioni sul clock, per permettere la sincronizzazione.
- Gli Access Point sono sempre accessi (sono a tutti gli effetti nodi della linea fissa)
- I nodi che vanno a dormire lo segnalano all'Access Point.
- L'Access Point mette da parte i frame rivolti ai nodi dormienti e li fornisce successivamente. Segnala per mezzo del Beacon quali sono i nodi per cui ci sono pacchetti in attesa.
- I nodi che si accorgono di avere pacchetti in attesa li richiedono esplicitamente all'Access Point.

10.4 Reti cellulari: pillole

La rete cellulare è una rete Wireless con infrastruttura:

- è costituita da centraline (Mobile Switching Centers);
- ciascuna centralina gestisce un certo numero di Base Station.

Ogni Base Station fornisce il segnale per un raggio di trasmissione di qualche chilometro. I dispositivi si associano alle Base station in modi non diversi da quelli descritti nelle pagine precedenti, ricorrendo a protocolli Multiple Access (quelli già visti, mossa necessaria dato che il mezzo è l'etere).



Introduciamo brevemente le generazioni di Reti cellulare che si sono viste negli anni:

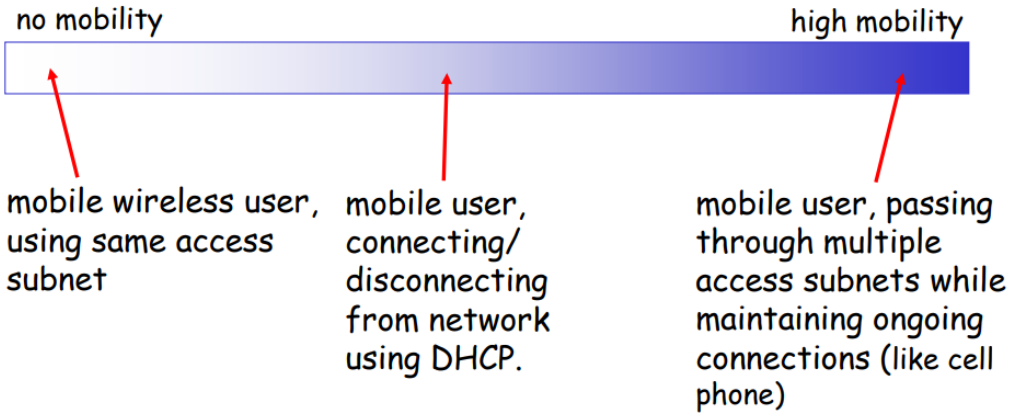
- **2G (o GSM, *Global System for Mobile Communications*)**
  Tecnologia analogica, pensato solo per la voce. Combinazione di TDMA ed FDMA. 13kb al sec
- **2.5G (o GPRS, *General Packet Radio Service*).**
  Tecnologia precedente con possibilità di avere traffico dati. Approccio non diverso da quello visto con la tecnologia DSL.
- **3G (o UMTS, *Universal Mobile Telecomunications Service*).**
  Integrazione di voce, video e dati in un'unica rete.
- **4G**
  Rete IP (non c'è più differenza tra parte telefonica e parte dati) basata su pacchetti. Packet-switching ha vinto definitivamente!
- **5G**
  Pensata per applicazioni real-time mobili (automobili in movimento, tactile Internet).
- **6G?**
  Rete intelligente, basata sui concetti di intelligenza artificiale (roba estremamente astratta al momento).

10.5 Mobilità

10.5.1 Definizione di mobilità

Internet è stata designata nell'ottica che i nodi siano sostanzialmente stazionari (protocollo IP in testa). Cosa succede se gli host non sono più stazionari? Si pensi a smartphone, smartwatch o qualunque altro dispositivo portatile…

Cosa si intende con mobilità?



- Mobilità dal punto di vista di IP, da non confondere con la mobilità dell'utente.
- Con mobilità si intende che l'host cambia frequentemente il punto di accesso alla rete.
    - Cosa si intende: il cambio di access point? Per il protocollo IP no, posso avere access point diverso ma mantenere lo stesso indirizzo IP (in quel caso l'host è stazionario, nonostante l'utente si sia mosso).
        - Esempio: mi sposto da un access point a un altro access point dello stesso edificio.
    - **Si intende sottorete di accesso, spostarsi da una sottorete di accesso con un certo indirizzo IP ad un'altra sottorete di accesso con un altro indirizzo IP**.
        - Esempio: mi sposto in un altro edificio. Il dispositivo acquisisce un nuovo indirizzo IP perché ci si è spostati in una nuova sottorete.
        - Questa mobilità può essere molto elevata, si pensi a quando si viaggia in automobile.

La prima idea è di gestire la mobilità con DHCP: l'utente acquisisce un nuovo indirizzo IP in maniera trasparente. In realtà non è vero: DHCP è trasparente rispetto all'utente, ma non rispetto all'applicazione.

- In generale applicazioni client-server non presentano problematiche se si adotta DHCP, ma…
- … si pensi a un'applicazione di streaming basata su protocollo TCP: cosa succede quando ci si sposta da una sottorete a un'altra? Cambia l'indirizzo IP e quindi cade la connessione TCP (ricordarsi che il socket è identificato dalla quaterna *Indirizzo IP sorgente – Porta sorgente – Indirizzo IP destinatario – Porta destinatario*): il servizio streaming non è garantito con continuità dato che si deve stabilire una nuova connessione TCP.

10.5.2 Mobile IP: comunicazione in un contesto di mobilità

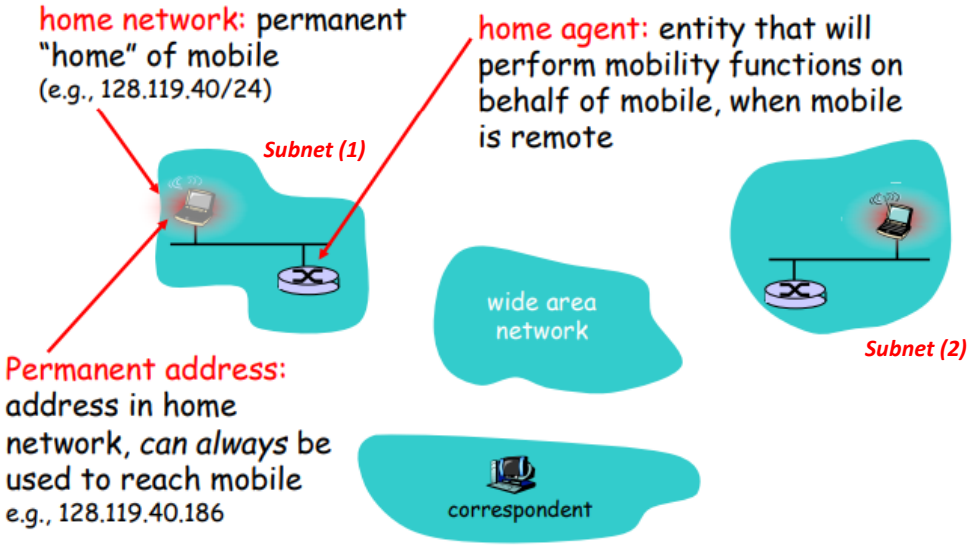
10.5.2.1 Introduzione per Mobile IP e i protocolli in generale

In generale sono problematici tutti quei contesti che richiedono il mantenimento dello stesso indirizzo IP nel tempo! Come si risolve?

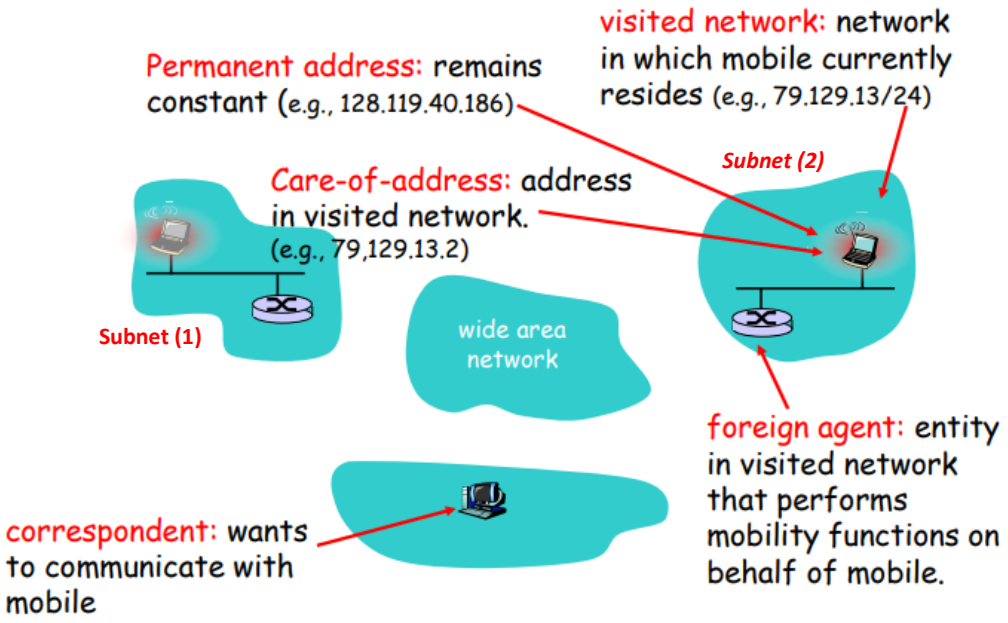
- Dire all'utente di non muoversi (sicuramente non risolveremo così)
- Fare in modo che l'utente continui a utilizzare il vecchio indirizzo IP nonostante si sia spostato in una nuova rete dove utilizza un nuovo indirizzo IP. La cosa pare in contraddizione e certamente questa cosa non può essere fatta col protocollo IP introdotto.

Si introduce mobile IP, un estensione del protocollo IP. Si introducono i seguenti soggetti:

- ***host mobile***
  L'host che consideriamo è un dispositivo mobile che si muove dalla sottorete 1 alla sottorete 2.
- ***home network e permanent address***
  L'host mobile risiede da qualche parte (come le persone, risiedono da qualche parte). Definiremo la rete in questione come home network e assegneremo all'host mobile un indirizzo permanente che gli altri dispositivi potranno utilizzare per contattare l'host mobile, anche quando questo non risiede nell'home network.
- ***home agent***
  Si introduce nell'home network un dispositivo detto home agent, che gestisce la comunicazione coi dispositivi che vogliono comunicare con l'host mobile quando questo non si trova presso l'home network.



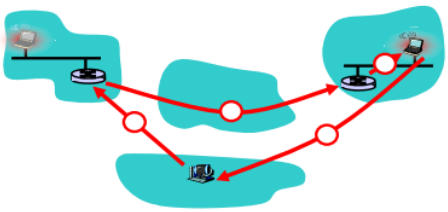
Supponiamo che l'host mobile si sposti in un'altra rete: come permettiamo la mobilità dell'host garantendo la continuità dei servizi?



- ***visited network***
  Nuova sottorete visitata dall'host mobile. L'host mobile si vede assegnare un nuovo indirizzo quando entra in questa nuova rete.
- ***foreign agent***
  Entità presente nella visited network che gestisce la mobilità dell'host mobile.
- ***correspondent***
  Soggetto che vuole comunicare con l'host mobile mentre quest ultimo è in movimento.

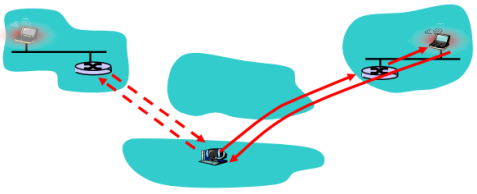
Come si gestisce la comunicazione tra host mobile e correspondent? La prima idea che salta alla mente è usare le tabelle di routing: non è possibile, la tabella di routing contiene record relativi a sottoreti e non relativi a nodi individuali. Creare record per ogni nodo individuale è una soluzione non scalabile (si pensi a una rete con migliaia di dispositivi mobili). Introduciamo la soluzione adottata:

- **Registrazione**.
  La prima fase è quella della registrazione: l'host mobile, entrato nella visited network, si registra presso il foreign agent. Fornisce informazioni personali: il suo indirizzo IP, l'indirizzo IP della rete di provenienza, l'indirizzo IP dell'home agent.
- **Segnalazione all'*home agent***.
  Il foreign agent contatta l'home agent per segnalare che l'host mobile si trova presso la rete da lui rappresentata. A questo punto:
  - il foreign agent è informato della presenza dell'host mobile;
  - l'home agent sa dove si trova l'host mobile, e chi è il relativo foreign agent.
- **Interlocuzione tra *correspondent* ed *host mobile***.
  Come fa il correspondent a comunicare con l'host mobile quando questo non si trova più presso l'home network? Due approcci possibili
  - **Approccio indiretto (<u>adottato da Mobile IP</u>)**.

    - Il correspondent scrive all'indirizzo permanente dell'host mobile.
    - L'home agent intercetta il pacchetto e si accorge dall'indirizzo di destinazione che qualcuno vuole scrivere all'host mobile.
    - L'home agent invia personalmente il pacchetto all'host mobile, che in questo momento si trova nella visited network.
    - L'host mobile risponde direttamente al correspondent.

    L'approccio è trasparente per il correspondent: non si accorge dello spostamento dell'host.
    - Si segnala un effetto collaterale di questa "triangolazione": si allungano i tempi per un dispositivo che vuole scrivere all'host mobile quando entrambi appartengono alla stessa sottorete (cioè la rete del dispositivo coincide col visited network).
    - Osservazione. Cosa succede se l'host mobile cambia continuamente visited network? Quello che abbiamo detto prima: l'host mobile si registra presso il foreign agent e il foreign agent comunica all'home agent la nuova rete. L'operazione non è istantanea, ergo non sarà percepita immediatamente la variazione di rete: pacchetti trasmessi all'host mobile in questo frangente di tempo saranno perduti.
  - **Approccio diretto**.

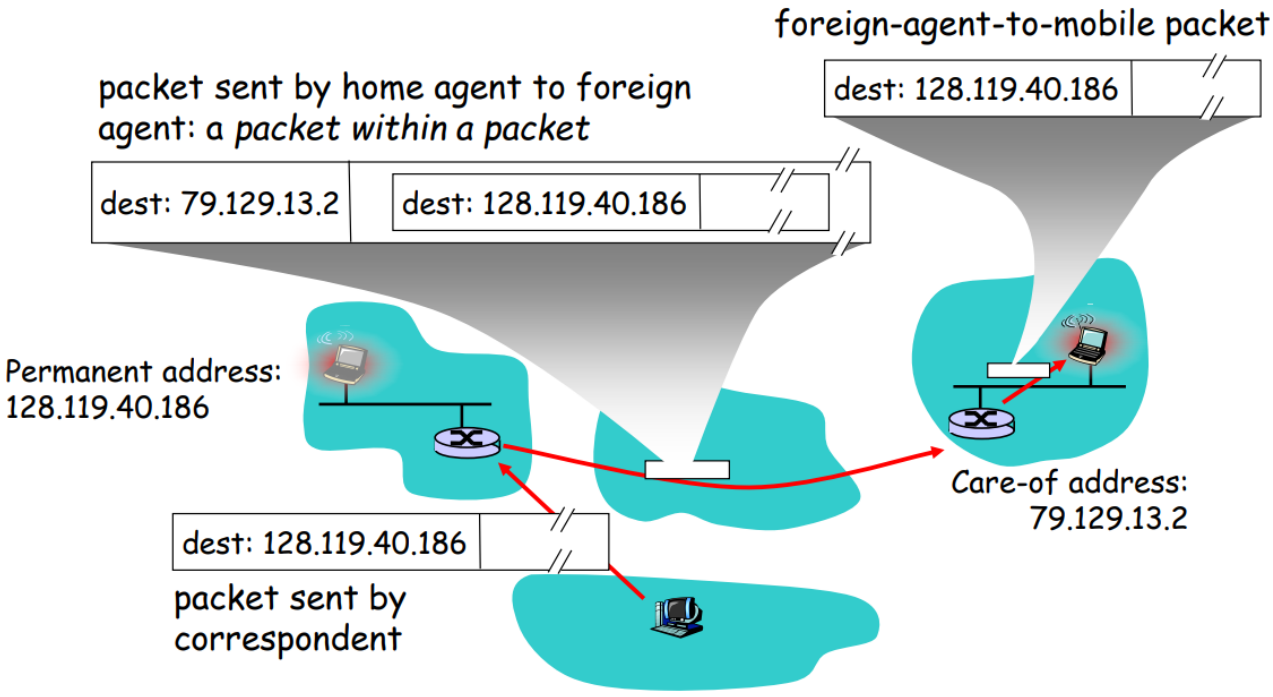
    

    L'approccio diretto mira al risolvere il problema della triangolazione del routing.
    - Il correspondent scrive all'indirizzo permanente dell'host mobile.
    - L'home agent intercetta il pacchetto e si accorge dall'indirizzo di destinazione che qualcuno vuole scrivere all'host mobile.
    - L'home agent segnala al correspondent che l'host mobile non è presente nell'home network e segnala l'indirizzo dove l'host mobile può essere raggiunto.
    - Il correspondent apre una connessione con l'host mobile, sfruttando le informazioni ottenute dall'home agent.

    L'approccio non è trasparente: il correspondent viene a sapere che l'host mobile è in movimento, ha necessità di inviare ancora i pacchetti al nuovo indirizzo. Il problema della triangolazione è superato, ma si ha discontinuità nel servizio se l'host mobile si muove frequentemente (non sono istantanee le cose spiegate). Come si risolve l'ultima questione? Il nuovo foreign agent scrive al vecchio foreign agent, e non solo all'home agent, che l'host mobile si è spostato (e inoltra pacchetti all'host mobile se necessario).

10.5.2.2 Forwarding dei pacchetti

Il forwarding dei pacchetti, in un contesto dove si adotta l'approccio in diretto, prevede l'invio di pacchetti per mezzo di incapsulamento.

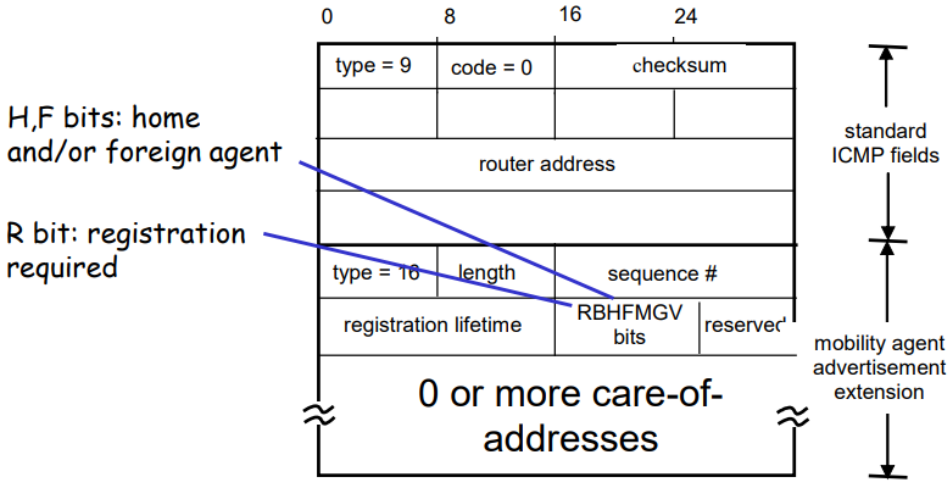


- Per prima cosa il correspondent invia un pacchetto all'indirizzo permanente.
- L'home agent incapsula il datagram all'interno di un nuovo datagram e invia il tutto al foreign agent (utilizzando l'indirizzo Care-of-address).
- Il foreign agent scompone il datagram recuperando il payload e trasmette quanto ottenuto all'host mobile: il risultato è la trasmissione del pacchetto inviato dal correspondent. Si osservi che il livello superiore a cui si fornisce il datagram è nuovamente il protocollo IP stesso.

10.5.2.3 Agent discovery

Come fa l'host mobile a scoprire il foreign agent quando si trova nella nuova rete? Si adotta un meccanismo simile al DHCP, ma si utilizzano messaggi di tipo ICMP. Questi sono inviati periodicamente dai vari nodi per pubblicizzare i propri servizi. All'interno del pacchetto ICMP vi sono i seguenti bit:

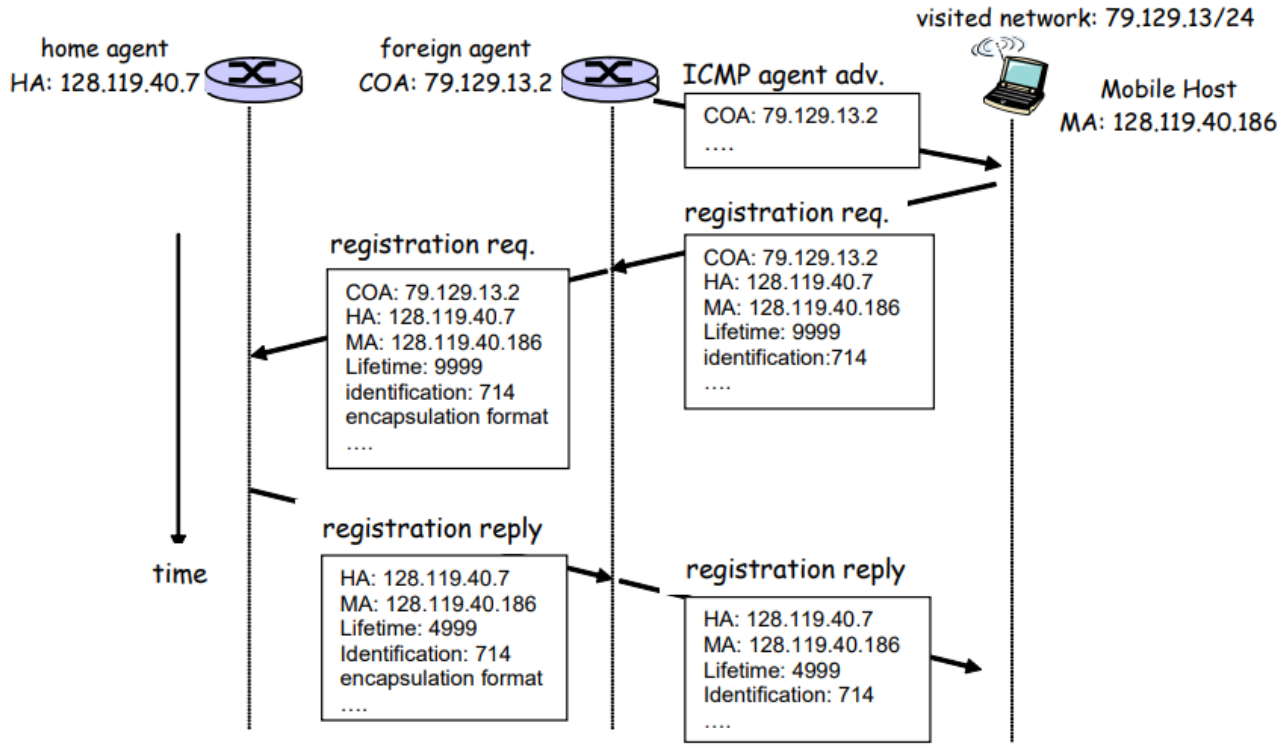
- Se il dispositivo è *home agent*
- Se il dispositivo è *foreign agent*
- Se è richiesta la registrazione quando l'host mobile accede alla rete (tipicamente sì)



I foreign agent inviano periodicamente questi pacchetti, quindi l'host mobile attende la ricezione di uno di questi pacchetti per poter contattare il foreign agent.

10.5.2.4 Registration

Supponiamo che l'host mobile sia arrivato presso la visited network.



- L'host mobile attende la ricezione di un ICMP contenente il Care of Address. L'host mobile prende atto del contenuto e se lo assegna.
- Inizia la procedura di registrazione trasmettendo il COA precedentemente ricevuto. Riferisce anche il suo indirizzo permanente, l'indirizzo dell'home agent e un lifetime (si indica un presunto tempo di permanenza nella connessione). Si pone anche un identificatore, in linea col DHCP.
- Il foreign agent riceve le informazioni e ne prende atto. Invia all'home agent un pacchetto per informarlo della nuova posizione dell'host mobile. Si segnala, tra le cose, il formato di incapsulamento adottato.
- L'home agent invia una risposta al foreign agent, che a sua volta riferirà all'host mobile. Il lifetime è ridotto visto il passare del tempo.

10.5.3 Effetto della mobilità sui livelli superiori

Abbiamo già detto che i protocolli sono stati pensati con l'ipotesi implicita che i mezzi fossero wired e i nodi statici. Quale effetto avremo sulla mobilità?

- Si hanno dei periodi in cui si perdono pacchetti a causa della mobilità. Non si interviene: approccio best-effort.
- Il protocollo TCP assume che perdita di pacchetti equivale a congestione, ergo non distingue la congestione dalla perdita dovuta a mobilità dei dispositivi. Segue che TCP abbassa il rate e il throughput risulta ridotto, con conseguente riduzione della performance.

Si potrebbe pensare a una modifica del protocollo TCP (solitamente non lo si fa). Altra idea (anche questa non proprio popolare) potrebbe essere dividere la connessione TCP in due "tronconi", uno dedicato alla connessione wired e uno alla connessione wireless: il primo applica il TCP come lo abbiamo conosciuto, il secondo applica un protocollo TCP modificato.

La soluzione adottata è una risoluzione a livello locale con retransmission di pacchetti.

10.6 Reti infrastructure-less

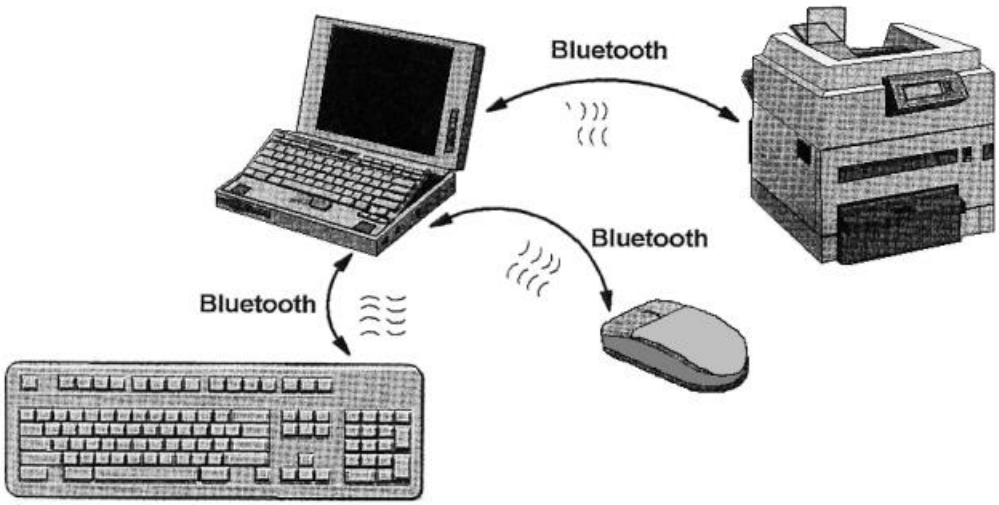
Fino ad ora abbiamo parlato di Reti Wireless basate su infrastruttura di reti Wireless, con edge che forniscono il servizio di connessione Wireless.

Le reti che andiamo a introdurre sono prive di infrastruttura (*infrastructureless*, o *ad hoc* in quanto la rete può essere creata quando server). La rete è costituita dai soli nodi, ciascuno caratterizzato da un'interfaccia Wireless.

10.6.1 Bluetooth

L'esempio principale di rete infrastructure less è la tecnologia Bluetooth (nome commerciale della WPAN).

- Si hanno varie tipologie di WPAN: reti ad elevata potenza, reti a bassa potenza (IOT, reti di sensori).
- Utilizzi
    - Inizialmente pensato come *cable replacement*: dispositivi statici, ma si minimizza la presenza dei fili eliminando i collegamenti tra i vari dispositivi.

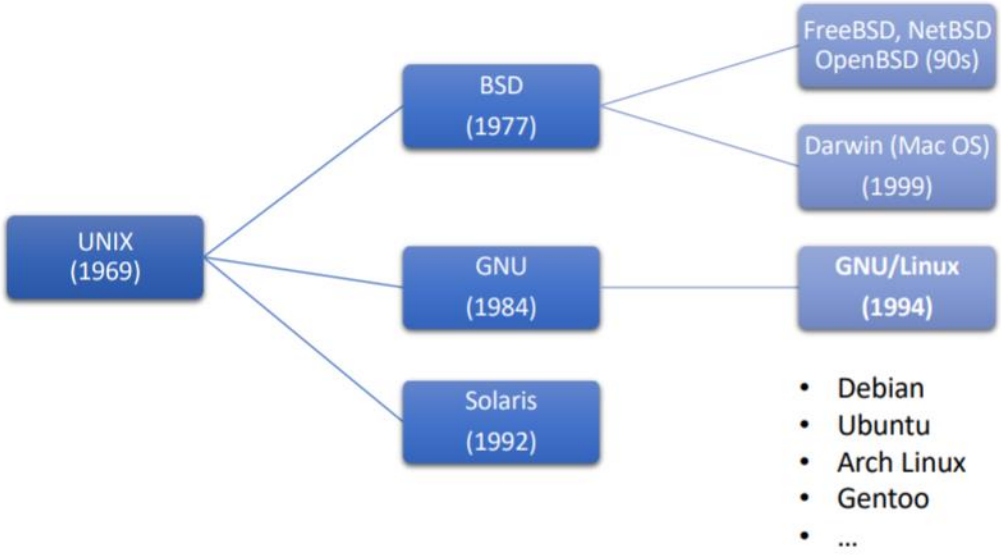


      Conseguenza: eliminazione dei fili necessità introduzione di una batteria.
    - Comunicazione telefonica in ambito telefonico. Tre telefoni in uno:
        - telefono cellulare, che si usa in auto con le cuffie bluetooth;
        - telefono cordless;
        - comunicazione con telefono vicino via Bluetooth (tariffa all'epoca in cui è stata pensata questa cosa non era flat, in sostanza si è pensato a una sorta di walkie-talkie).
    - Comunicazioni per scambiarsi informazioni (ad esempio file).
- **Rete Piconet**.
  Una rete Bluetooth è detta Piconet, al suo interno si distingue un dispositivo detto master e i rimanenti detti slave. Il master è colui che ha avuto l'iniziativa.

- **Protocollo di accesso alla rete Bluetooth**.
  In Bluetooth si hanno 78 frequenze operative (rispetto alle 11 di Wifi). La cosa interessante è che possono essere utilizzate tutte per mezzo di un meccanismo noto come *frequence hopping* (divido il tempo in slot e mi occupo di una certa frequenza in un certo slot). L'obiettivo è rendere più robusto il Bluetooth (che ha potenze basse) in un contesto dove le frequenze sono già ampiamente utilizzate da tante tecnologie (tra cui la Wifi).
    - Al momento della creazione della Piconet si crea una sequenza pseudorandomica di frequence hopping, che tutti i dispositivi seguiranno rimanendo sincronizzati.
    - La durata dello slot è 625 microsecondi, ogni nodo può trasmettere o ricevere un pacchetto durante lo slot.
    - Il protocollo di accesso è detto Time Division Duplexing, che è un adattamento del TDA visto in passato. La fairness viene meno pesantemente: gli slot pari sono assegnati al master, gli slot dispari sono assegnati a tutti gli altri slave messi insieme.
    - Come fa lo slave a sapere che deve trasmettere?
      - Trasmette in slot dispari
      - Tiene conto della sequenza pseudorandom e quindi le frequenze differenti.
      - Il master decide chi deve trasmettere trasmettendo un pacchetto di polling in uno slot pari. Allo slot successivo lo slave è abitato a trasmettere un pacchetto di dati, se non ha niente trasmette un pacchetto null.
    - Può succedere che si trasmettano pacchetti più lunghi, ma la lunghezza deve essere dispari per non alterare la sequenza. La frequenza rimane la stessa in tutto lo slot espanso.

11 LABORATORIO: INTRODUZIONE AI SISTEMI LINUX/UNIX

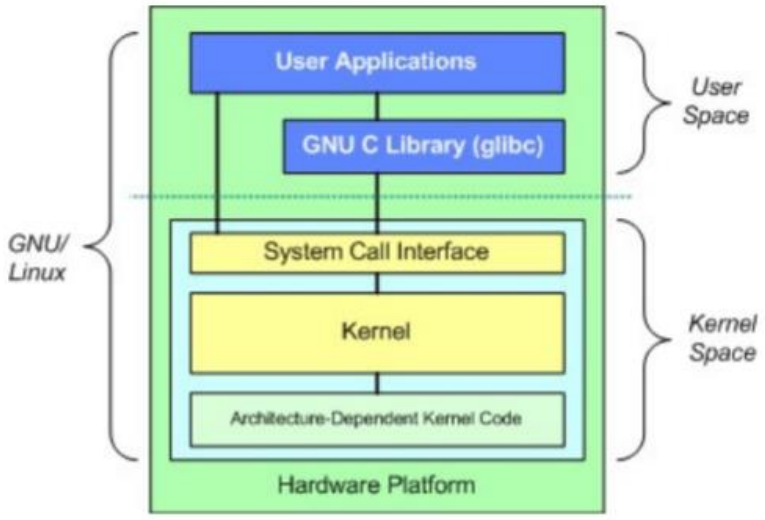
11.1 Nascita di Unix



Unix nasce nel 1969 come sistema operativo in ambito accademico, si cerca di produrre un sistema operativo stabile e funzionante. Nel tempo è emersa l'esigenza di rendere questo sistema operativo libero, si è evoluto in GNU nel 1984. Da GNU si sono sviluppati altri sistemi operativi, in particolare Linux (noi useremo Debian).

11.1.1 Struttura di UNIX

La struttura è quella tipica dei sistemi operativi.



- **Kernel**
  In basso abbiamo il kernel, che in sostanza è l'anima del nostro calcolatore.

  Come comunica con noi il kernel? Attraverso le chiamate di sistema[12]: chiamiamo dei servizi offerti dal sistema operativo. Ci sono delle cose che non posso fare scrivendo una funzione, ma che posso fare chiamando una primitiva (e quindi facendo una chiamata di sistema).
  Nell'esecuzione si passa al livello inferiore e quanto fatto è questione del sistema, il programmatore non può influire in alcun modo sulle cose fatte e può solo usufruire dei risultati della primitiva.
  - **Esempio**: primitive semaforiche viste a Calcolatori elettronici.

[12] tappo del tombino, non si può aprire – se no si muore asfissiati (cit.)

    - Noi dobbiamo permettere a un'applicazione, che gira su un host, di inviare e ricevere dati da un'altra applicazione, che gira su un altro host. Nel fare questo dobbiamo tener conto dei protocolli (un *modus operandi*)
- Sopra il tombino abbiamo le **librerie** e le **applicazioni scritte** dall'utente. Noi staremo in questa parte, e interloquiremo col kernel con le system calls.

11.1.2 Caratteristiche di UNIX

- **Multitasking**
  Possibilità di eseguire più processi "contemporaneamente" (un'illusione nei sistemi multiprogrammati, come visto a Calcolatori elettronici).
- **Multiutente**
  Più utenti possono interagire, con livelli di privilegio diversi.
- **Portabilità**
  Impiegabile su hardware diversi senza dover subire modifiche.
- **Modularità**
  Programmi semplici, componibili e riusabili.

11.1.2.1 Nozioni introduttive agli utenti

Il sistema è multiutente. Possiamo distinguere due tipologie di utenti:

- L'utente **root**, amministratore del sistema che può compiere qualsiasi tipo di operazione.
- Gli altri utenti, non *root* e soggetti a restrizioni.

Questi utenti possono usare il sistema, ma non possono agire (ad esempio) sulla configurazione di rete, non possono configurare un firewall, non possono configurare un server web. Queste cose, onerose, possono essere fatte solo innalzandosi a livello root.

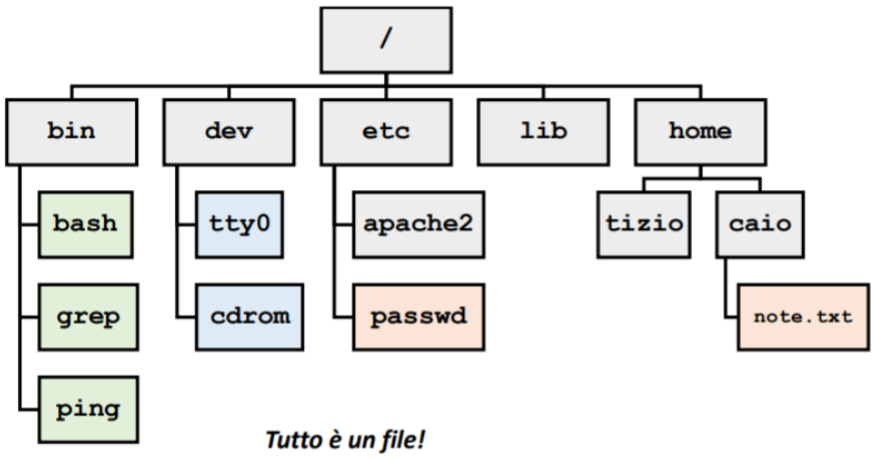
Nota bene. Solitamente si crea un utente normale per l'uso abituale e si ricorre a root SOLO se necessario. La distribuzione ha già gli utenti configurati.

- **Utente non root**: username *studenti* e password *studenti*
- **Utente root**: username *root* e password *studenti*.

11.1.2.2 File system, percorsi

L'idea di base su UNIX è che tutto è un file. Il file system è il modulo che si occupa di gestire i file: documenti, cartelle, files di configurazione… Alcune osservazioni…

- Il **root**[13] **del file system** è la radice, il punto di partenza di tutti i percorsi che conducono a files
- **Cartelle importanti**



Tutto è un file!
Ogni documento, cartella, dispositivo I/O, interfaccia di rete, stream di byte, ecc., è accessibile dall'unico file system.

    - / è la directory principale

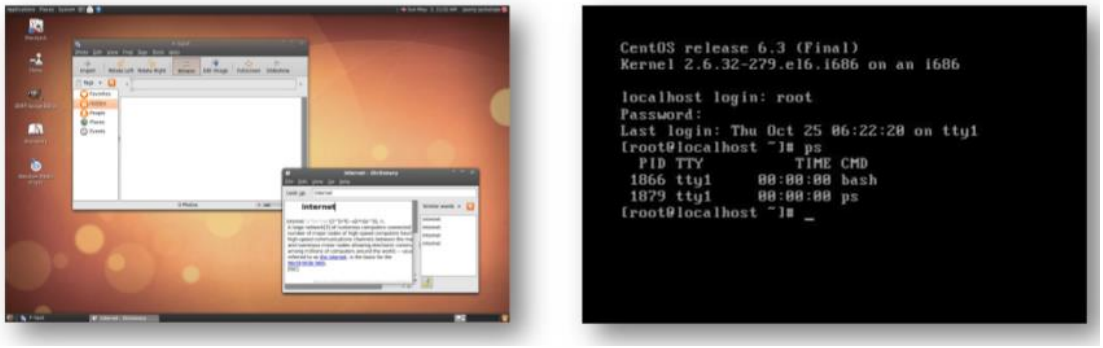
[13] L'utente root non c'entra nulla.

    - /home contiene le varie directory degli utenti (se sulla macchina ho utente A e B ciascuno avrà una propria cartella)
    - /sbin (*system binaries*) contiene i programmi di sistema
    - /etc contiene i file di configurazione (abbiamo all'interno la cartella *apache2*, che contiene tutta la configurazione di Apache).
    - /media rende accessibile il contenuto dei supporti rimovibili.
- **Metodologie per specificare i path (percorsi)**. I percorsi possono essere specificati in due modi:
    - Percorso assoluto (si esprime a partire dalla root)
      `/home/tizio/Documents/appunti.txt`
    - Percorso relativo (si esprime a partire dalla cartella in cui ci si trova)
      `Documents/appunti.txt`
- **Simboli speciali**
    - Con la tilde ~ si indica la nostra home directory
    - Con un punto . si indica la directory corrente
    - Con due punti .. si indica la directory padre.

11.1.2.3 Shell

La shell è l'interprete dei comandi che il sistema mette a nostra disposizione (possiamo chiamarlo terminale). Ne abbiamo due:

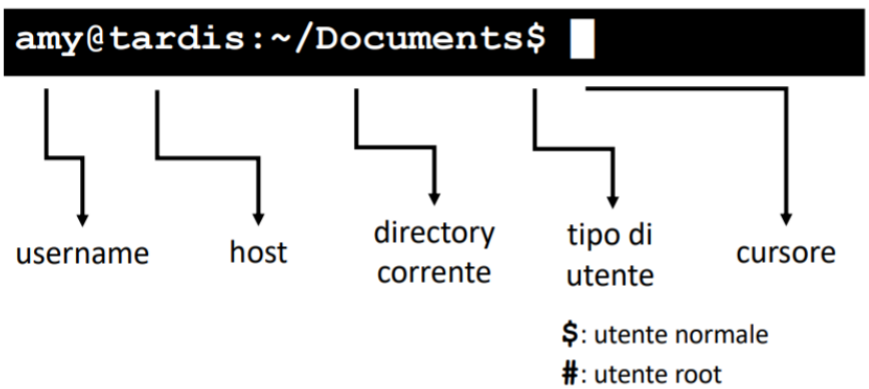
- Shell grafica (GUI, Graphical User Interface)
    - Più facile da utilizzare
- Shell testuale (CLI, Command Line Interface)
    - Senza fronzoli, veloce e potente, ma brutta.
    - *Noi useremo questo perché a Ingegneria si usano cose brutte* (cit.).



La shell:

- esegue il comando;
- segnala un errore nel caso in cui non sia in grado di completarlo;
- stampa l'eventuale output del comando.

Esempio di prompt.



11.1.2.4 Metacaratteri

I metacaratteri possono essere usati per indicare insiemi di file o cartelle

- L'asterisco * sostituisce zero o più caratteri
- Il punto interrogativo ? sostituisce un singolo carattere
- [a,b,c] oppure [a-z] sostituisce un carattere nell'insieme specificato (vale anche con cifre).

- Si considerino i seguenti esempi:

  `$ ls` ←Lista completa
  aa.c abc.c a.c a.h axc.c
  `$ ls *.c` ← solo i files con estensione .c
  aa.c abc.c a.c axc.c

  `$ ls a*.c` ← solo i files con estensione .c e che iniziano col carattere a
  aa.c abc.c a.c axc.c

  `$ ls ?.?` ← solo i files che hanno un'estensione identificata da un solo carattere e per nome un solo carattere
  a.c a.h

  `$ ls a??.c` ← solo i files con estensione .c, che iniziano col carattere a seguito da altri due caratteri
  abc.c axc.c

  `$ ls a[b-t]c.c` ← solo i files con estensione .c, che iniziano col carattere a e che hanno un secondo carattere compreso tra b e t.
  abc.c

  `$ ls a[4,f,x]c.c` ← solo i files con estensione .c, che iniziano col carattere a, che hanno come secondo carattere uno di quelli elencati, e che hanno come terzo carattere c.
  axc.c

11.1.3 Comandi utili

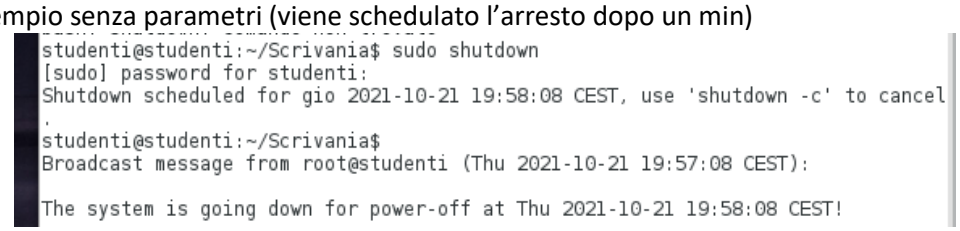
11.1.3.1 Comandi per chiuder/pulire il terminale (`logout`, `clear`) o arrestare/riavviare il sistema (`shutdown`)

- `logout`
  In alternativa si può usare la scorciatoia CTRL+D
- `shutdown`
  - Arresto o riavvio del sistema.
  - Invocabile solo dall'utente root.

  `# shutdown -h now` ⟶ Arresto

  `# shutdown -r now` ⟶ Riavvio

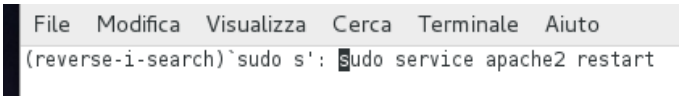
  - Esempio senza parametri (viene schedulato l'arresto dopo un min)



- `clear`
  Comando per svuotare il terminale.

11.1.3.2 Autocompletamento, history dei comandi, ricerca all'interno della storia

- Tasto TAB
  - Autocompletamento.
  - Inizio a scrivere e premo TAB: il terminale mi restituisce il nome completo (utile quando la directory contiene caratteri ostici)
- Tasti freccia su e freccia giù
  - History dei comandi recenti.
- Combinazione tasti CTRL + R
  - Ricerca all'interno della storia

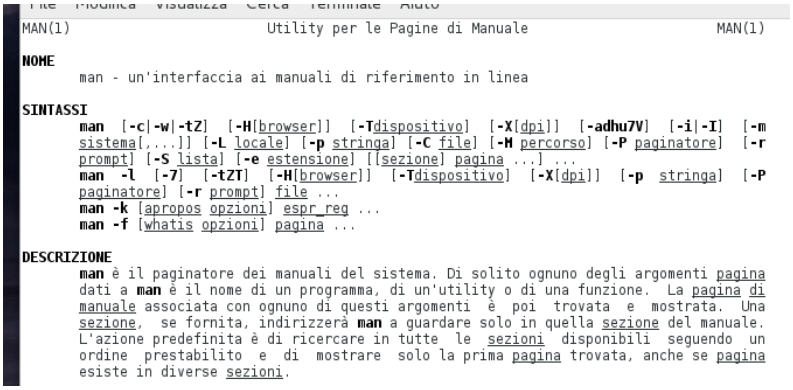


11.1.3.3 Informazioni sui comandi (`man`, `whatis`, `apropos`)

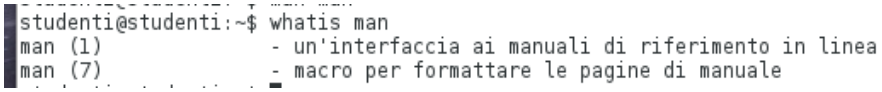
- `man`
  - Comando per accedere al manuale contenente una descrizione esaustiva, con sintassi, opzioni e possibili messaggi di errore.
  - È diviso in sezioni. A noi interessano le seguenti sezioni…
    - Sezione 1: comandi del terminale
    - Sezione 2: funzioni delle librerie C
    - Sezione 5: file di configurazione.
  - Normalmente non è necessario indicare la sezione, ma in alcuni casi potrebbe essere necessario per evitare ambiguità (si va in sezione 1 di default, o nella sola sezione contenente informazioni)

    ```
    $ man printf <- comando su terminale
    $ man 3 printf <- funzione C
    ```

  - Prima parte di schermata con `$ man man`



- `whatis`
  - Comando che restituisce una descrizione breve di una pagina del manuale.
  - Se la pagina è presente in più sezioni vengono stampate le descrizioni brevi relative a tutte le sezioni.
  - Utile per sapere al volo cosa fa un comando.
  - Esempio col comando man:



- `apropos`
  - Comando per ricercare una parola in nomi e descrizioni.
  - Utile quando vogliamo fare qualcosa e non ci ricordiamo il comando preciso.
  - Comando whatis a confronto con apropos

```
studenti@studenti:~$ whatis unzip
unzip (1)            - list, test and extract compressed files in a ZIP archive
studenti@studenti:~$ apropos unzip
bunzip2 (1)          - a block-sorting file compressor, v1.0.6
funzip (1)           - filter for extracting from a ZIP archive in a pipe
gunzip (1)           - compress or expand files
preunzip (1)         - prefix delta compressor for Aspell
unzip (1)            - list, test and extract compressed files in a ZIP archive
unzipsfx (1)         - self-extracting stub for prepending to ZIP archives
```

11.1.3.4 *Comandi per navigare nelle directory (`cd`, `pwd`, `ls`)*

- `cd`
  - *Change directory*, permette di modificare la directory corrente.

```
studenti@studenti:~$ cd Scrivania
studenti@studenti:~/Scrivania$
```

  - I percorsi possono essere espressi come parametro nei modi detti prima: percorso assoluto o percorso relativo.
    - Necessario l'accesso in esecuzione (si veda più avanti i permessi) per poter indicare una certa directory.
- `pwd`
  - *Print working directory*
  - Stampa il percorso assoluto della directory corrente

```
studenti@studenti:~$ pwd
/home/studenti
```

- `ls`
  - Stampa il contenuto della directory specificata.
  - Se non viene specificata la directory si considera quella corrente
  - Nell'indicare la directory desiderata posso usare il percorso assoluto o quello relativo.
  - Opzioni:
    - *-l* per stampare una lista con maggiori dettagli
    - *-a* per stampare anche file e cartelle nascoste

    Osservazione: le opzioni sono cumulabili (-a -l o -al)
  - Esempio senza opzioni

```
studenti@studenti:~$ ls
Documenti  Modelli  Pubblici     Scaricati  Video
Immagini   Musica   public_html  Scrivania
```

  - Esempio con opzioni cumulate

```
studenti@studenti:~$ ls -la
totale 168
drwxr-xr-x 22 studenti studenti  4096 ott 21 20:01 .
drwxr-xr-x  3 root     root      4096 ott 28  2016 ..
-rw-------  1 studenti studenti  8323 ott 21 19:58 .bash_history
-rw-r--r--  1 studenti studenti   220 ott 28  2016 .bash_logout
-rw-r--r--  1 studenti studenti  3515 ott 28  2016 .bashrc
drwx------  9 studenti studenti  4096 ott  1 12:09 .cache
drwx------ 15 studenti studenti  4096 ott  1 12:09 .config
drwx------  3 studenti studenti  4096 ott 28  2016 .dbus
drwxr-xr-x  2 studenti studenti  4096 set 27  2017 Documenti
drwx------  3 studenti studenti  4096 apr 11  2017 .gconf
-rw-r-----  1 studenti studenti     0 ott 28  2016 .gksu.lock
```

11.1.3.5 *Comandi per la creazione/rimozione delle directory (`mkdir` ed `rmdir`)*

- `mkdir`

  Crea una directory con la sintassi

  `mkdir nome_directory`

- `rmdir`
  - Rimuovi una directory con la sintassi

    `rmdir nome_directory`
  - Attenzione: il comando funziona solo se la directory è vuota

- `rm`
  - Comando per rimuove file o directory.

    `rm file1 file2 …`
  - Per rimuovere una cartella con tutto il contenuto è necessario usare l'opzione -r. Senza l'opzione le cartelle non vengono rimosse.

11.1.3.6 Comandi per copiare/spostare directory (`cp` ed `mv`)

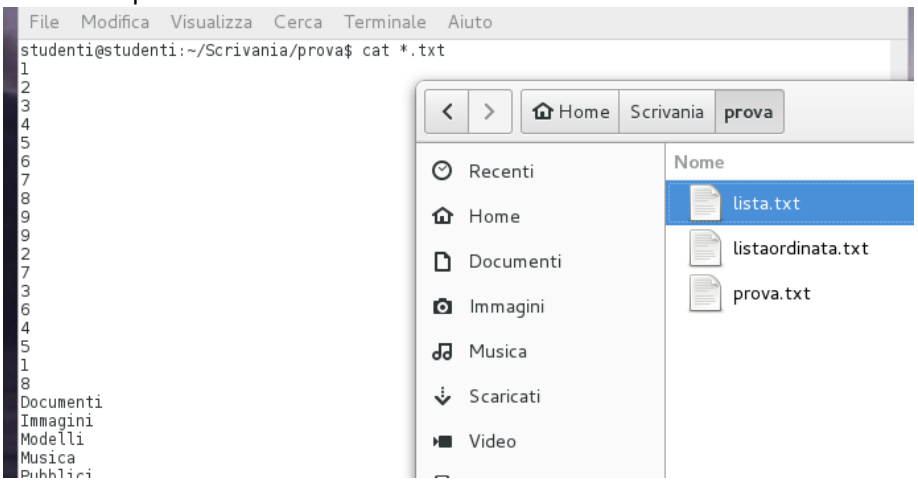
- `cp`
  - Comando per copiare files o directories in una destinazione.
  - Possibile:
    - Copia di un file o di una directory in un'unica directory destinataria
    - Copia di più file o directory in un'unica directory destinataria
  - Sintassi: `cp src1 src2 … dst_dir`
- `mv`
  - Equivalente del comando precedente, ma per lo spostamento in una directory destinataria.

11.1.3.7 Comando `touch`

- `touch[ nomefile]`
  - Indicando il nomefile aggiorna il relativo timestamp di accesso e modifica.
  - Se il file non esiste viene creato.

11.1.3.8 Lettura e concatenazione di file (comandi `cat`, `less` e `head/tail`)

- `cat`
  - Comando per concatenare uno o più file, stampandoli nello standard output (sul terminale).
  - Può essere utile per visualizzare velocemente file brevi nel terminale.



- `less`
  Visualizzare il file un po' alla volta e interattivamente. Versione migliorata del comando more
- `head/tail`
  Visualizzare prima o ultima parte di un file. Posso specificare il numero di byte da mostrare con -c o il numero di righe con -n. Se non si indica si mostra di default 10 righe.

11.1.3.9 Comandi `su` e `sudo`

- `su` (*switch user*)
  - Comando per accedere al terminale di un altro utente.
  - Se l'utente non viene specificato si accede al terminale di root.
  - Viene richiesta la password dell'utente con cui si vuole accedere.
- `sudo nome_comando` (*switch user and do*)
  - Serve per lanciare un comando come un altro utente.
  - Se non specificato l'utente è quello root.
  - Viene chiesta la password dell'utente corrente (**E NON DELL'UTENTE INDICATO**).

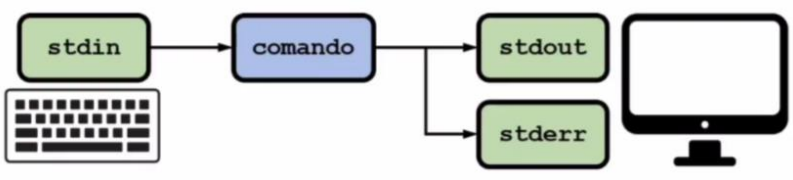
```
studenti@studenti:~/Scrivania/pipeline$ sudo sudo
[sudo] password for studenti:
```

  - L'utente deve fare parte nel gruppo *sudoers*.

11.1.4 Redirezione I/O

Gli input e output da riga di comando sono rappresentati come tre “file”:

- `stdin`
  input da tastiera
- `stdout`
  output su schermo
- `stderr`
  messaggi di errore su schermo

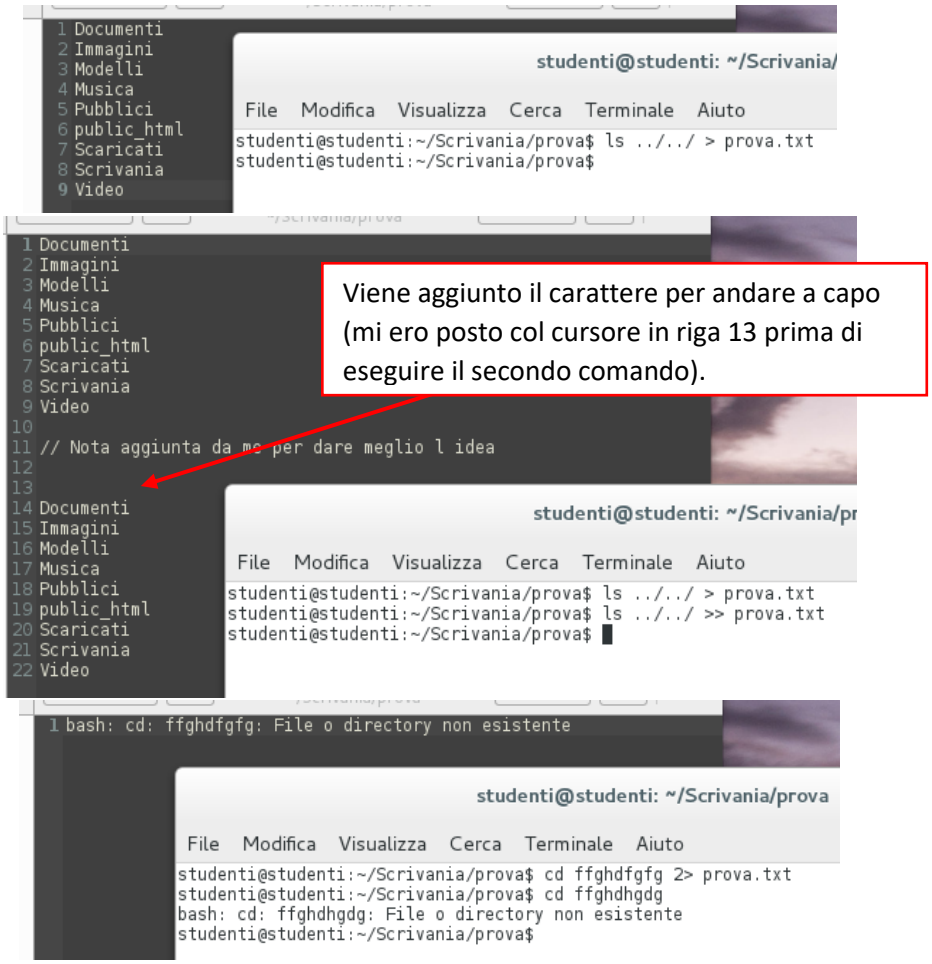


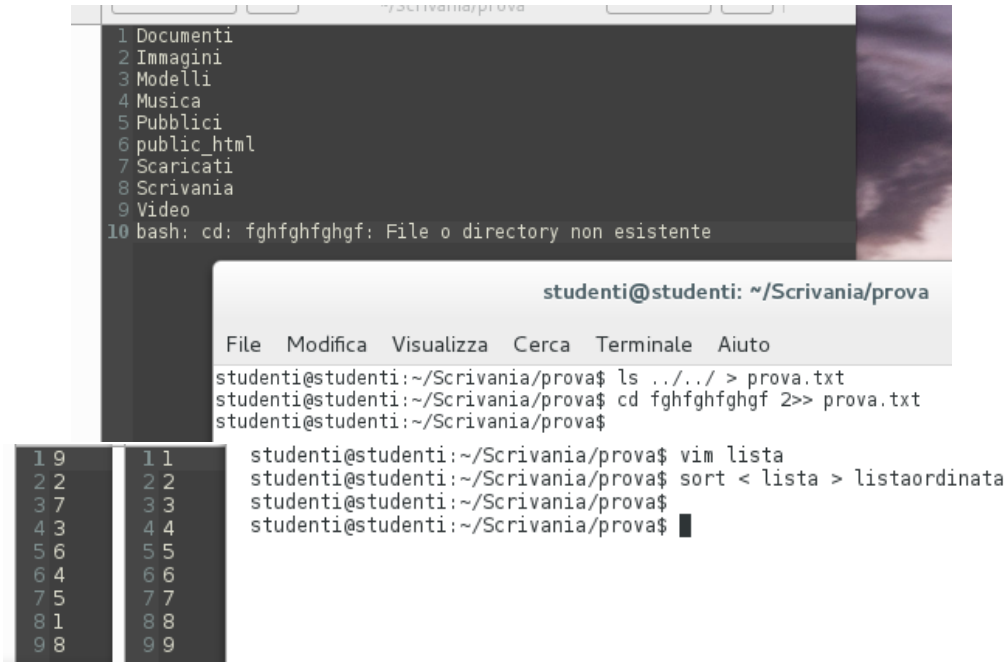
Possiamo deviare l’output di un comando verso un file oppure acquisire l’input a partire da un file.

- Invio dello stdout a un file
  `$ ls -1 > filelist.txt`
  Se il file non esiste viene creato, se il file esiste viene sovrascritto
- Invio dello stderr a un file
  `$ ls -1 2> filelist.txt`
- Invio sia del primo che del secondo a un file
  `$ ls -1 &> filelist.txt`
- Aggiunta dell’output a un file
  `$ ls -1 >> filelist.txt`

  `$ ls -1 2>> filelist.txt`
  `$ ls -1 &>> filelist.txt`
- Possibile fare la redirezione dei due output su due file diversi
  `$ comando > out.txt 2> errors.txt`
  Possiamo mettere come comando qualsiasi comando che preveda un’uscita
- Possibile combinare le redirezioni, ad esempio, nell’uso del comando sort
  `$ sort < list.txt`
  `$ sort < list.txt > sortedlist.txt`

Esempi di esecuzione di comandi.





11.1.5 Pipeline

La pipeline permette di porre l'output di un comando come input del successivo comando. Si consideri il seguente esempio

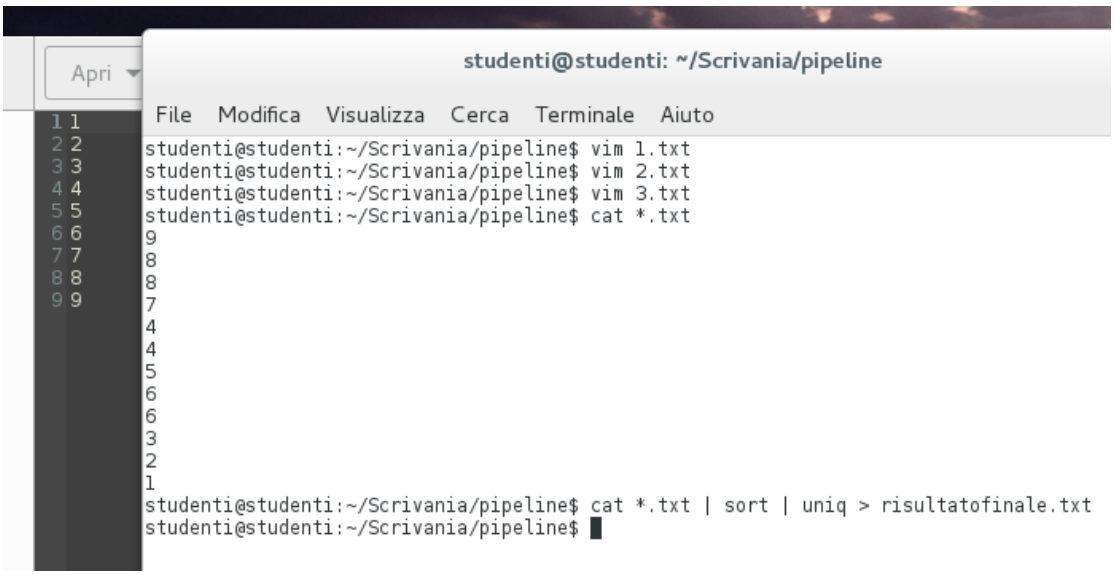
```
$ ls -l mydir | less
```

Per prima cosa andiamo a generare la lista dettagliata dei file/directory contenuta in `mydir`, questa lista la stampiamo all'utente con l'editor less.

È possibile usare la pipeline più volte e in combinazione con le redirezioni.

```
$ cat *.txt | sort | uniq > risultatofinale.txt
```

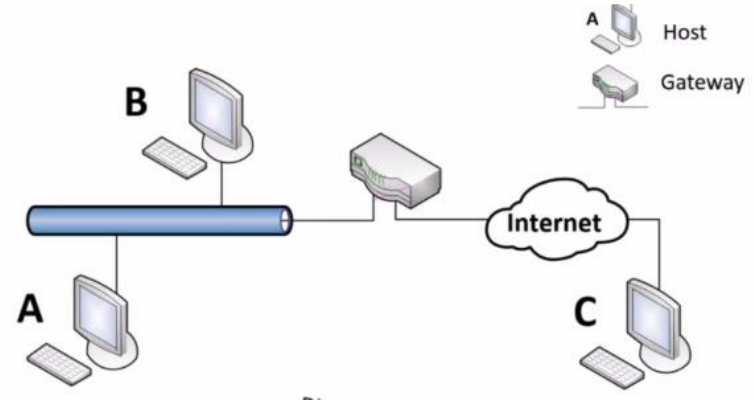
- Per prima cosa concateniamo il contenuto di tutti i file aventi estensione txt
- La concatenazione non viene stampata, ma posta come input in sort, dove avviene l'ordinamento degli elementi
- L'output di sort non viene stampato, ma posto come input di uniq, che eliminerà le righe "doppioni"
- L'output di uniq, grazie alla redirezione dello standard output, viene posto nel file `risultatofinale.txt`



12 LABORATORIO: CONFIGURAZIONE DELLA RETE

Per ora la nostra macchina non è connessa alla rete. Siamo all'interno di Virtualbox, una scatola all'interno della quale stiamo emulando un host finto con sistema operativo Debian. Vogliamo configurare la macchina virtuale affinchè possa inviare dati in rete.

12.1 Gli hosts: esempio introduttivo



Prendiamo un semplice scenario di rete, dove abbiamo tre host:

- Host A,
- Host B, e
- Host C.

L'host è un qualunque dispositivo in grado di connettersi alla rete. Gli host A e B sono collegati alla stessa sottorete, non si attraversano router di frontiera per andare da A a B e viceversa. Da qualche altra parte si ha C, che si collega ad Internet per poter comunicare con A e B. Per permettere la comunicazioni tra gli host è necessario avere i seguenti elementi:

- Indirizzo IP
- Maschera di rete
- Indirizzo del gateway
- Indirizzo del server DNS

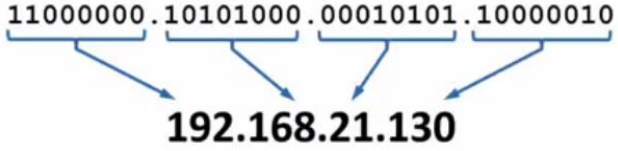
Procediamo introducendo subito i primi due elementi.

12.2 Indirizzi IP

12.2.1 Nozioni di base

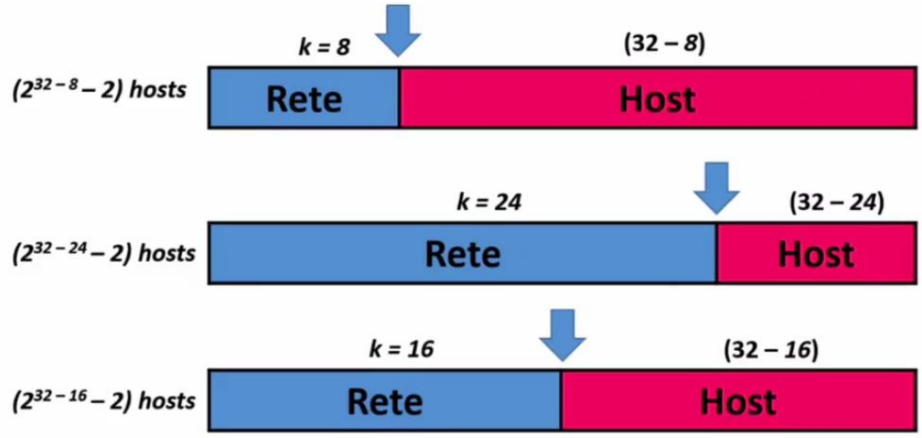
L'indirizzo IP è l'identificativo dell'host all'interno della rete. Per essere più precisi, non è esattamente così: l'indirizzo IP non è univoco, un dispositivo connesso in rete ha un numero di indirizzi IP pari al numero di schede di rete del dispositivo (se siamo collegati con Ethernet abbiamo un indirizzo, se ci colleghiamo col Wifi avremo un altro indirizzo).

- **Sequenza di bit**. Parliamo di una sequenza di 32 bit, che rappresentiamo in modo compatto per mezzo di una *notazione decimale puntata*. Si divide la sequenza in quattro gruppi di 8 bit, separati da un punto e scritti in base 10.



- **Identificazione della rete e dell'host**. L'indirizzo IP è logicamente diviso in due parti:
  - I primi k bit a sinistra identificano la rete;
  - I rimanenti $32 - k$ bit identificano l'host

Logicamente affermiamo che minori sono i $32 - k$ bit minori sono gli indirizzi IP assegnabili agli host connessi alla rete.
Contrariamente a prima non è necessario fare coincidere la fine delle due parti con un punto della notazione: il *Classless Inter-Domain Routing* permette di alterare il numero di k bit senza dover dipendere dalla divisione in ottetti. Chiaramente maggiore è k maggiori sono le reti assegnabili, ma diminuiscono gli host assegnabili ad ogni singola rete.



Osservazione: sottraggo due perchè devo sempre togliere indirizzo di rete e indirizzo di broadcast, che non possono essere assegnati a nessuna rete (logicamente).

12.2.2 Maschera di rete (netmask)

La netmask è una sequenza di 32 bit che presenta i seguenti bit:

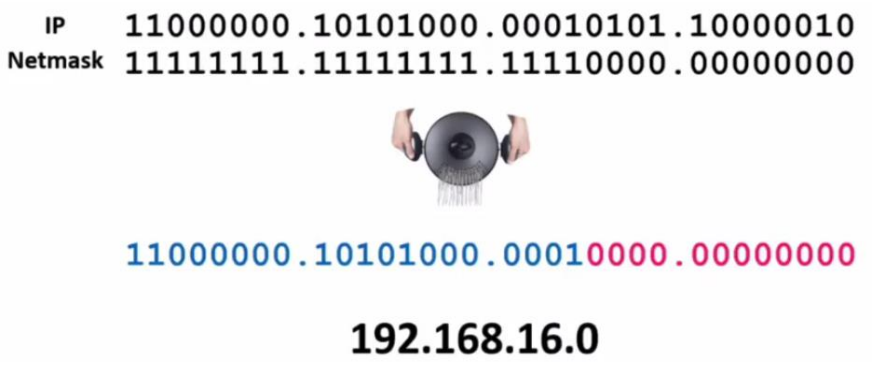
- $32 - k$ bit meno significativi uguali a zero;
- i k bit più significativi uguali a uno.

Supponiamo di voler utilizzare 20 bit per rappresentare la rete e 12 bit per rappresentare l'host: otteniamo la seguente maschera di rete

Per indicare il numero di bit rappresentanti la rete si pone slash e numero dei bit subito dopo l'indirizzo IP.

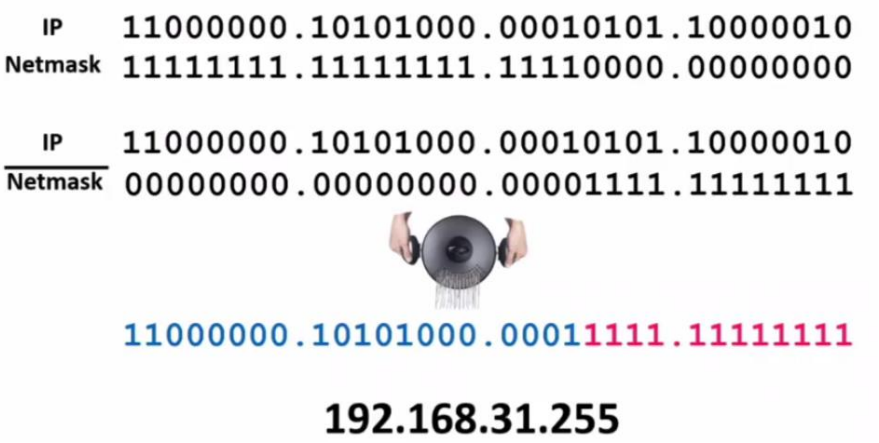
12.2.2.1 Individuazione dell'indirizzo di rete utilizzando la netmask

L'indirizzo di rete si ricava per mezzo di un AND bit a bit tra indirizzo IP e maschera di rete.



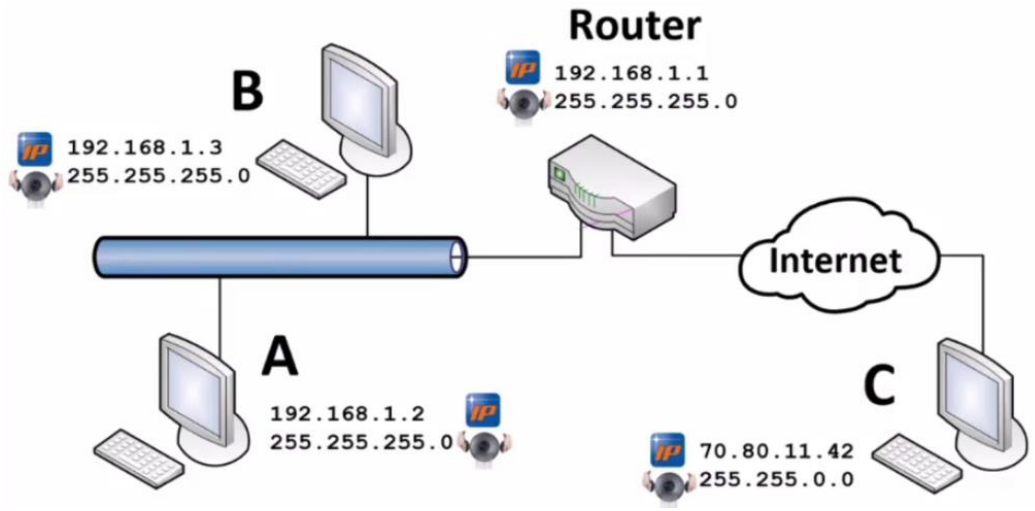
12.2.2.2 Individuazione dell'indirizzo di broadcast di rete utilizzando la netmask

L'indirizzo di broadcast (cioè l'indirizzo per inviare messaggi a tutti coloro che si trovano in una certa rete) si ricava per mezzo di un OR bit a bit tra indirizzo IP e maschera di rete negata.



12.3 Indirizzi IP e maschere di rete nell'esempio iniziale

Ok, a questo punto abbiamo indirizzo IP e maschere di rete per tutti gli host nella rete.



Gli indirizzi relativi alla rete domestica iniziano tutti per 192.168.1, in particolare abbiamo il router con indirizzo 192.168.1.1. Il router perde l'indirizzo fornito dal provider (TIM, Vodafone) quando si spegne. Il router espone un indirizzo pubblico, quello fornito dal provider, per permettere a chi sta fuori di raggiungermi.

12.4 Comando *ip*

12.4.1 Informazioni generali

Il comando ip permette di visualizzare e manipolare le impostazioni di rete.

- Sintassi del comando
  `$ ip`

```
Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }
WHERE  OBJECT := { link | address | route | … }
…
```

- Stampa di tutte le interfacce
  `$ ip addr show`
- Stampa delle interfacce accese
  `$ ip addr show up`

Andiamo ad analizzare l'output del comando

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00 :00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:28:fd:4c brd ff:ff:ff:ff:ff:ff
    inet 192.168.50.2/24 brd 192.168.50.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe28:fd4c/64 scope link
       valid_lft forever preferred_lft forever
```

Nella nostra macchina sono presenti due interfacce di rete:

- **lo**
  - Interfaccia di *loopback*.
  - Un'interfaccia "finta", se io spedisco pacchetti verso questa interfaccia ritorneranno a me. L'interfaccia è utile per il test di alcune funzionalità. Non la descrivo (cit.), quella che ci interessa davvero è la seguente.
- **eth0**
  - Interfaccia basata sul protocollo Ethernet.
  - `BROADCAST` e `MULTICAST` non importa che ve li descriva (cit.)[14]
  - `UP` segnala che la scheda è abilitata
  - `LOWER_UP` segnala che il cavo è stato collegato.
  - `mtu` è acronimo di *Maximum Transmission Unit*, dimensione massima (in byte) del pacchetto IP
  - `qdisc` sta per *Queuing discipline*, politica con cui si stabilisce il prossimo pacchetto da inoltrare (solitamente politica FIFO). Tre code a priorità:
    - quella di livello 0 per i processi interrativi (quelli che hanno bisogno di maggiore velocità di spedizione);
    - i livelli inferiori, con priorità inferiore, che possono muoversi a velocità inferiori.
  - `state` indica se la scheda è abilitata (UP) o disabilitata (DOWN)
  - `qlen` sta per *Transmission Queue Length*, consiste nella lunghezza della coda di trasmissione.
  - Con la seguente riga è indicato l'indirizzo MAC (indirizzo fisico) della scheda di rete. Con brd si intende *broadcast* (dominio di broadcast impostato)

    ```
    link/ether 00:0c:29:28:fd:4c brd ff:ff:ff:ff:ff:ff
    ```

  - Con la seguente riga si indica l'utilizzo del protocollo internet IPv4 a livello network, l'indirizzo IP associato alla scheda di rete, il numero di bit utilizzati per rappresentare la rete (maschera). Segue l'indirizzo di broadcast, e la visibilità della scheda (nell'esempio globale, cioè visibile nella rete)

    ```
    inet 192.168.50.2/24 brd 192.168.50.255 scope global eth0
    ```

12.4.2 Attivazione e disattivazione dell'interfaccia di rete col comando

Il comando *ip* può essere utilizzato per attivare o disattivare un'interfaccia di rete. Supponiamo di voler abilitare l'interfaccia eth0: poniamo quanto segue

```
# ip link set eth0 up
```

se si vuole disabilitare la stessa interfaccia si scrive lo stesso comando, ponendo down invece di up:

```
# ip link set eth0 down
```

Si osservi che entrambe le righe di comando devono essere inviate con privilegi innalzati (utente root).

[14] Distinzione:
- Il broadcast è la comunicazione da un host a tutti gli altri host collegati alla rete.
- Il multicast è la comunicazione da un host a più host collegati alla rete.

12.4.3 Configurazione manuale degli indirizzi IP di un'interfaccia col comando (temporanea)

Gli indirizzi IP di una macchina possono essere configurati staticamente (lo fa l'utente manualmente) e dinamicamente (assegnazione automatica per mezzo del DHCP, non appena la macchina viene connessa alla rete).

- **Impostare l'indirizzo IP di una particolare interfaccia (eth0 nel nostro caso)**
  Vogliamo associare all'interfaccia eth0 l'indirizzo IP 192.168.1.42/24. Impostiamo l'indirizzo di broadcast 192.168.1.255
  ```
  # ip addr 192.168.1.42/24 broadcast 192.168.1.255 dev eth0
  ```

- **Rimozione dell'indirizzo IP associato a una particolare interfaccia (sempre eth0)**
  Due possibili strade:
  ```
  # ip addr del 192.168.1.42/24 dev eth0
  ```
  oppure
  ```
  # ip addr flush dev eth0
  ```

Brutta notizia: la configurazione svolta per mezzo del comando ip è annullata dal riavvio della macchina. Segue metodo alternativo per poter svolgere una configurazione manuale delle interfacce di rete.

12.5 Configurazione manuale (permanente) degli indirizzi IP di un'interfaccia di rete

12.5.1 File *interfaces*

La configurazione permanente può essere fatta alterando il file di configurazione

`/etc/network/interfaces`

Consideriamo un possibile esempio di configurazione

```
auto lo

iface lo inet loopback

iface eth0 inet static
     address 192.168.1.2
     netmask 255.255.255.0
     broadcast 192.168.1.255
```

- Con la prima riga indichiamo che l'interfaccia di loopback deve essere abilitata all'avvio della macchina.
- Con le righe successive configuriamo gli indirizzi associati alle interfacce di rete.
  - Per quanto riguarda `lo` segnaliamo che è adottato il protocollo IPv4 con `inet`, e che è l'interfaccia di loopback con l'omonima keyword
  - Per quanto riguarda `eth0` segnaliamo la stessa cosa per quanto riguarda il protocollo, inoltre indichiamo che gli indirizzi IP sono statici. Segue l'assegnazione degli indirizzi IP.

E se volessi abilitare l'interfaccia di rete eth0 all'avvio della macchina? Altero la prima riga.

12.5.2 Comandi ifup e ifdown

I comandi ifup e ifdown possono essere utilizzati per attivare e disattivare le interfacce di rete, sfruttando i dati presenti nell'interfaccia di configurazione.

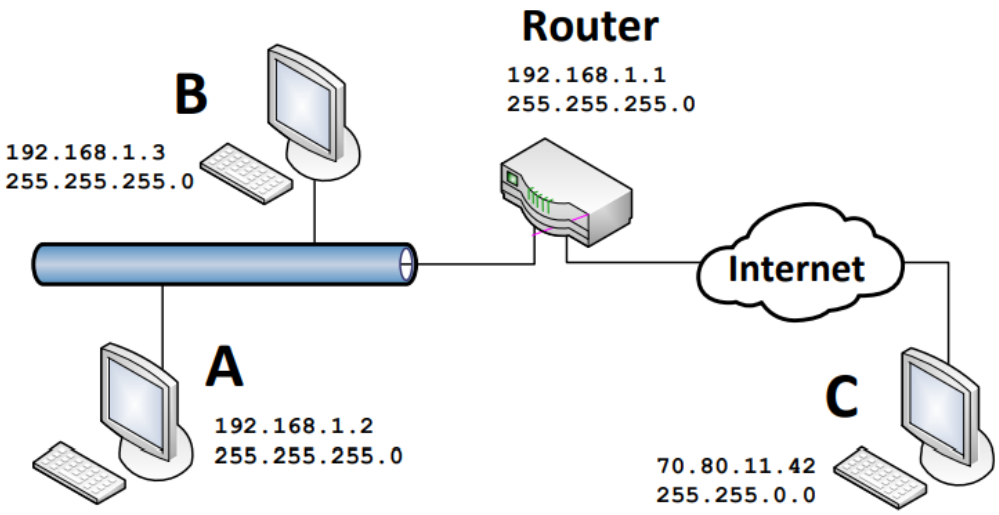
- **Abilitazione di una particolare interfaccia (ad esempio eth0) con la configurazione nel file**
  ```
  # ifup eth0
  ```
- **Disabilitazione di una particolare interfaccia**
  ```
  # ifdown eth0
  ```
- **Attivazione di tutte le interfacce di rete indicate nella sezione auto del file di configurazione.**
  ```
  # ifup –a
  ```

Sull'ultimo comando si tenga a mente che:

- è eseguito all'avvio della macchina;
- abilita le interfacce nell'ordine indicato.

12.6 Esempio iniziale con comportamento nell'invio dei pacchetti

Riprendiamo l'esempio iniziale: osserviamo come ogni dispositivo presente nella figura abbia un proprio indirizzo e una certa maschera di rete.



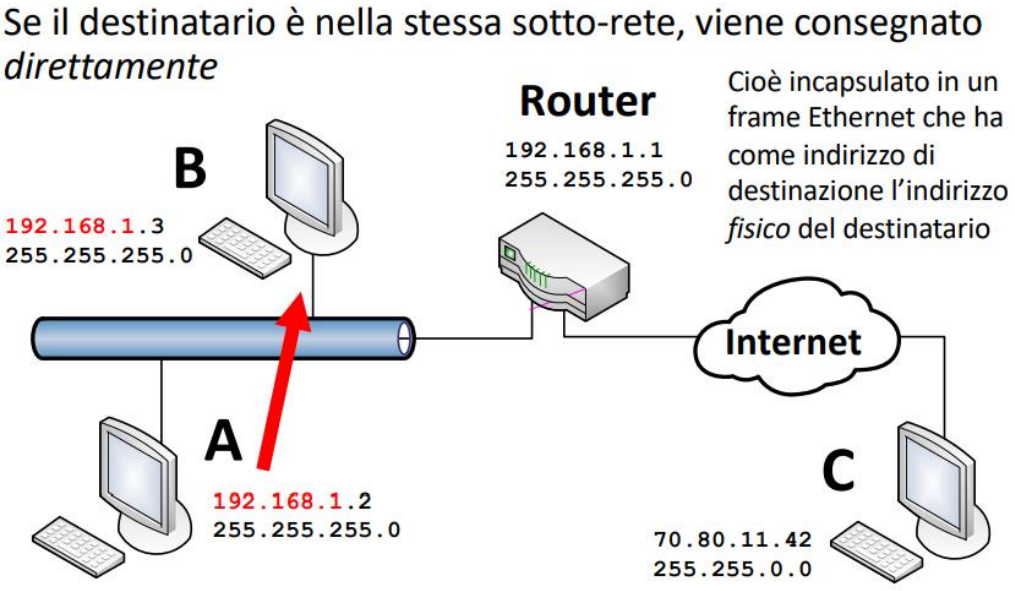
La maschera di rete, come possiamo aspettarci, è la stessa per tutti gli host appartenenti a una particolare rete:

- A,B e Router, che si trovano nella stessa rete, hanno una maschera di rete in comune;
- C ha una maschera diversa, dato che si trova in una rete diversa.

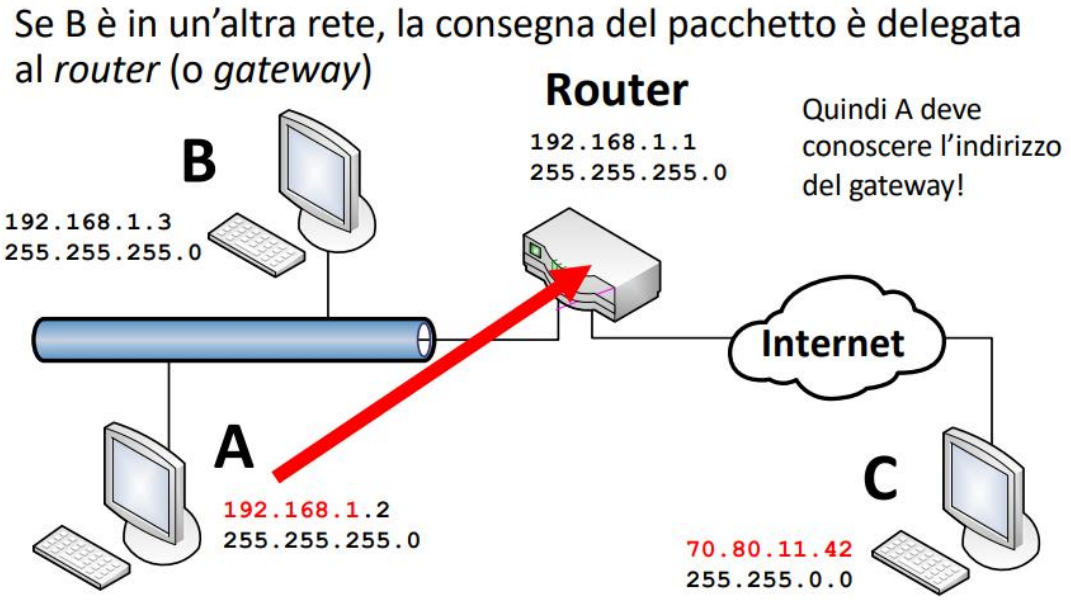
La nuvola di Internet consiste nel network core, dove i percorsi svolti dal pacchetto (path) avvengono passando da router: è necessario introdurre meccanismi in grado di ottimizzare il path (si pensi all'algoritmo di Dijkstra, che permette l'individuazione del cosiddetto *shortest path*). La questione fondamentale da capire è il comportamento dell'host a seconda del destinatario di un pacchetto.

```
dest_subnet .= my_netmask & dest_addr;

if (dest_subnet == my_subnet) then
      deliver to dest_addr;
else
      forward to default_gateway;
end if
```

- Se il pacchetto deve essere trasmesso a un host della stessa rete allora la cosa è semplice.



- Se il pacchetto deve essere trasmesso a un host di una rete diversa allora lo si inoltra al *default gateway*, che introduciamo di seguito.



12.7 default gateway: configurazione

Il default gateway consiste nel router della rete (della rete dove sono presenti gli host A e B, se si guarda l'esempio). Lo si può definire staticamente aggiungendo una riga nel file `/etc/network/interfaces`, che abbiamo già visto

```
auto lo

iface lo inet loopback

iface eth0 inet static
     address 192.168.1.2
     netmask 255.255.255.0
     broadcast 192.168.1.255
     gateway 192.168.1.1
```

- Per visualizzare la tabella di *routing*:

```
$ ip route show
default via 192.168.1.1 dev eth0
192.168.1.0/24 dev eth0 … src 192.168.1.2
```

- Per aggiungere rotte:
  - Invio diretto nella rete locale

```
# ip route add 192.168.1.0/24 dev eth0
```

  - Default gateway

```
# ip route add default via 192.168.1.1
```

- Per scoprire la rotta usata:

```
# ip route get 70.143.3.67
```

12.8 Risoluzione dei nomi

12.8.1 Motivazioni

Per semplificare l'uso della rete è cosa conveniente associare dei nomi agli indirizzi IP. Si consideri il seguente esempio:

```
127.0.0.1 = localhost
192.168.1.3 = hostB
131.114.73.85 = www.unipi.it
```

Chiaramente è molto più semplice ricordarsi unipi.it rispetto a una sequenza di 32 bit, anche se posta in notazione decimale compatta. Vogliamo introdurre i meccanismi che permettono di associare indirizzi IP a particolari nomi:

- definizione statica per mezzo di un apposito file;
- sistema dei nomi di dominio (DNS, Domain Name System).

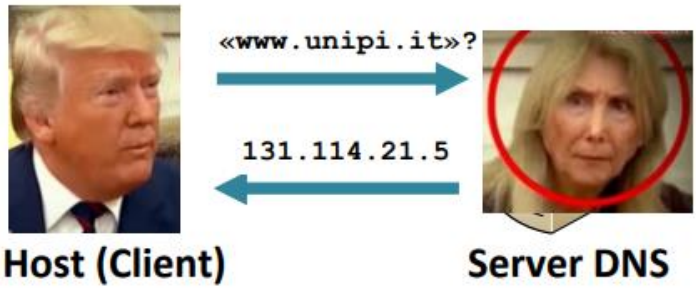
12.8.2 Definizione statica dei nomi

Per mezzo del file `/etc/hosts` andiamo a definire un elenco di associazioni indirizzo IP – nome

```
127.0.0.1           localhost
127.0.1.1           localhost
151.101.37.140      www.reddit.com
131.114.73.85       www.unipi.it
```

12.8.3 Domain Name System (DNS)

Il DNS è un database distribuito e gerarchico su più server (cosiddetti server DNS), a cui l'host invia delle richieste per conoscere, dato un nome, il corrispondente indirizzo IP.



Per mezzo del file /etc/resolv.conf indichiamo gli IP dei server DNS che l'host può contattare: se un server DNS non individua corrispondenze a seguito di richiesta allora l'host può rivolgersi ad altri server DNS.

```
nameserver 8.8.8.8
nameserver 8.8.4.4
```

Domanda all'esame: il DNS è configurato correttamente? Posso verificare il contenuto del file, ma soprattutto potrebbe essere utile utilizzare il seguente comando:

```
$ nslookup nome_dominio
```

Lanciare una richiesta da browser può mostrare che qualcosa non funziona, ma non è automatico che un browser non funzionante implichi DNS non configurato correttamente.

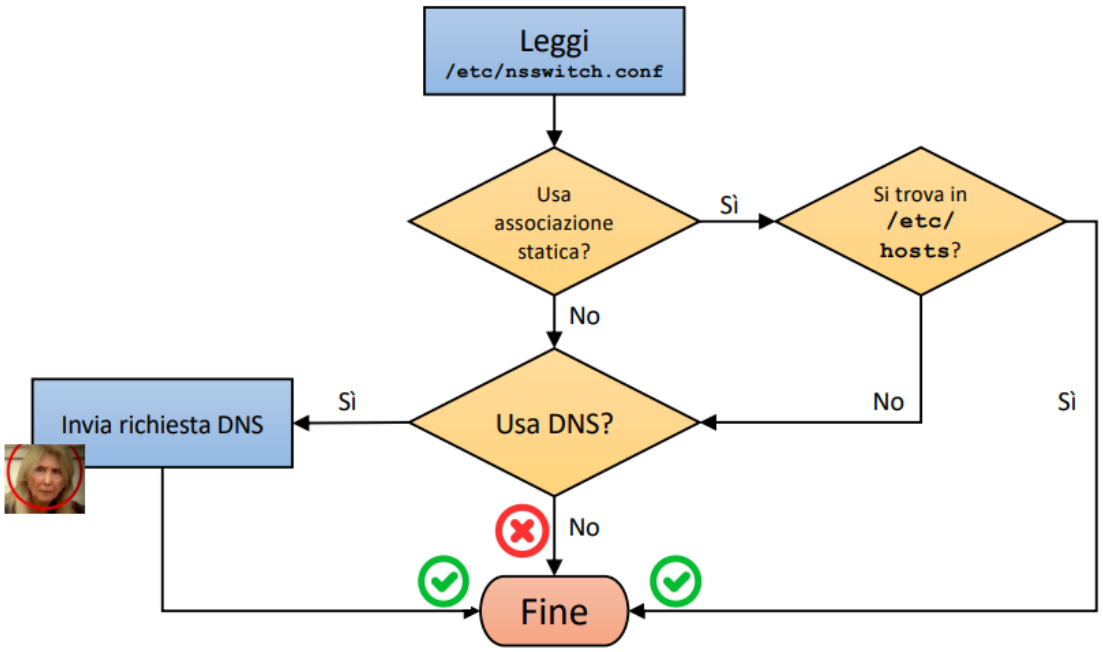
12.8.4 Name Service Switch

Il Name Service Switch (NSS) è il meccanismo utilizzato dai sistemi Unix per la risoluzione dei nomi. Centrale è il file `/etc/nsswitch.conf`, utilizzato per definire le fonti da utilizzare per la risoluzione dei nomi di dominio, e l'ordine con cui utilizzare tali fonti.

```
…
hosts: files dns
…
```

12.8.5 Recap

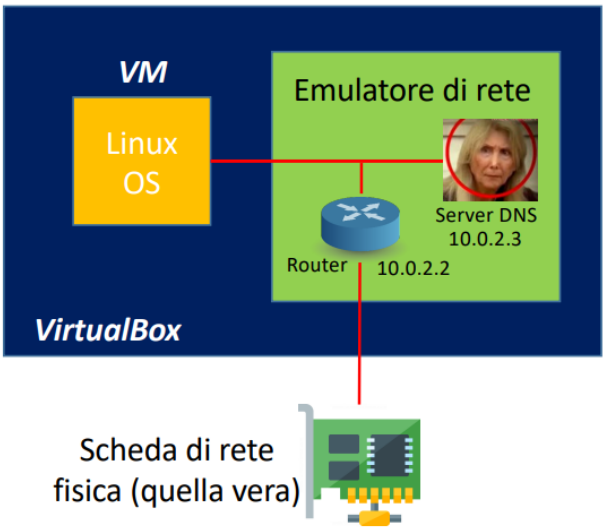
Concludiamo ripercorrendo i vari step con cui si arriva alla risoluzione di un nome di dominio



- Lettura di `nsswith.conf` per individuare le fonti da utilizzare e l'ordine di utilizzo.
- Se l'ordine è quello spiegato nelle ultime pagine allora procediamo nel seguente modo:
    - si controlla se è utilizzata l'associazione statica, nel caso si verifica nell'apposito file se è stata definita associazione tra indirizzo IP e nome, in caso affermativo abbiamo finito;
    - se lo step precedente non porta a individuazione di corrispondenze allora verifichiamo se è utilizzato il DNS, nel caso si lancia una richiesta.

12.9 Infrastruttura di rete in VirtualBox

All'interno della macchina virtuale gira una distribuzione Linux, grazie all'emulazione dell'hardware del calcolatore. Nella macchina è presente anche un emulatore di rete, che presenta server DNS e default gateway.



Il mapping esce da VirtualBox e collega la scheda di rete fisica (quella vera), che avrà un indirizzo IP vero associato a un provider.

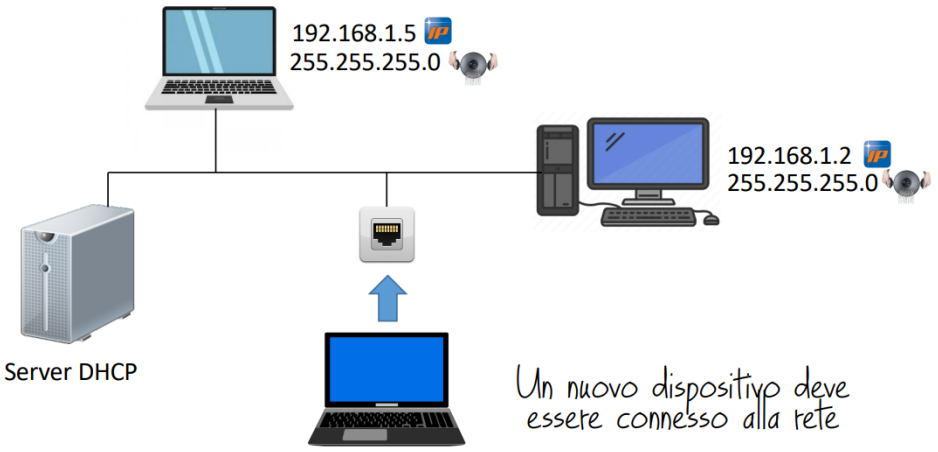
12.10 Dynamic Host Configuration Protocol (DHCP)

12.10.1 Scopo di DHCP

Il protocollo DHCP consente la **configurazione automatica e dinamica** dei parametri TCP/IP degli host:

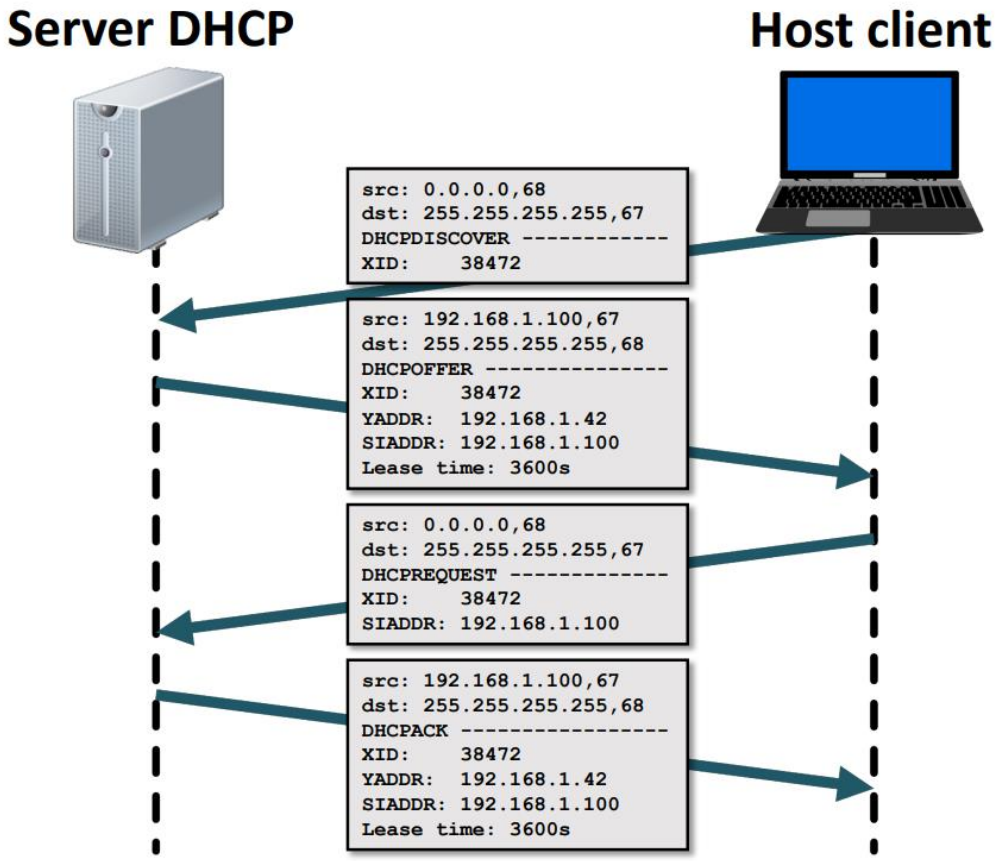
- Indirizzo IP
- Maschera di rete
- Indirizzo del gateway
- Indirizzo del server DNS

All'interno della rete è presente un server DHCP (possono essercene di più, ma per comodità supponiamo che ce ne sia uno solo): è un server costantemente in ascolto, in attesa di richieste da client che vogliono collegarsi alla rete ma che non possono farlo in quanto privi dei necessari parametri di configurazione.



12.10.2 Funzionamento in breve

La spiegazione completa del protocollo è stata fatta dal prof. Anastasi. L'ing. Pistolesi si è limitato ad anticipare le nozioni base. Supponiamo di avere un host che a un certo punto viene connesso alla rete (quello indicato nella figura precedente).



- Osservazioni:
    - L'indirizzo 0.0.0.0 è un indirizzo finto (in assenza di configurazione il client non ha ancora un indirizzo IP).
    - L'indirizzo 255.255.255.255 è l'indirizzo di broadcast (invio di un messaggio a tutti
    - Le porte 68 e 67 sono riservate alla trasmissione di processi DHCP.
    - Nei vari messaggi si indica uno XID, in modo tale da distinguere le varie richieste DHCP (dato che, ribadiamo, il client non ha ancora un proprio indirizzo IP).
- Il client, alla ricerca di una configurazione, trasmette un messaggio broadcast di tipo `DHCPDISCOVER`, attraverso cui chiede l'invio di proposte di configurazione ai DHCP server presenti (per comodità immaginiamo che ce ne sia uno, ma in presenza di più server si ha una competizione dove ogni server invia la sua proposta e il client sceglie tra le proposte ricevute).
- Tutti gli hosts ricevono il messaggio `DHCPDISCOVER`. I DHCP server presenti rispondono con un messaggio broadcast `DHCPOFFER`, contenente una proposta di configurazione. Si osservi il *lease time* con cui si indica una scadenza temporale della proposta (deve esserci risposta da parte del client entro il tempo indicato).
- Tutti gli hosts ricevono il messaggio `DHCPOFFER`. Il client trasmette un messaggio broadcast `DHCPREQUEST` con cui segnala la volontà di accettare la configurazione proposta.
- Tutti gli hosts ricevono il messaggio `DHCPREQUEST`. Il server DHCP conclude inviando un messaggio `DHCPACK` (un acknowledge) direttamente al client (primo messaggio non broadcast trasmesso nella sequenza). Si segnala nel messaggio la durata della configurazione (sempre il solito lease *time*)

12.10.3 Installazione e configurazione

Il server DHCP si install per mezzo del seguente comando

```
apt-get installi sc-dhcp-server
```

A seguito dell'installazione avremo il file di configurazione `/etc/default/isc-dhcp-server`, all'interno del quale si indica (tra le cose) l'interfaccia di rete attraverso cui il server gestirà l'interlocuzione spiegata nella pagina precedente

```
INTERFACES = 'eth0'
```

Un altro file è `/etc/dhcp/dhcp.conf`

```
option domain-name-servers 192.168.0.2, 8.8.8.8;
option routers 192.168.0.1;
default-lease-time 3600;
subnet 192.168.0.0 netmask 255.255.255.0 {
    range 192.168.0.10 192.168.0.100;
}
```

man dhcpd.conf

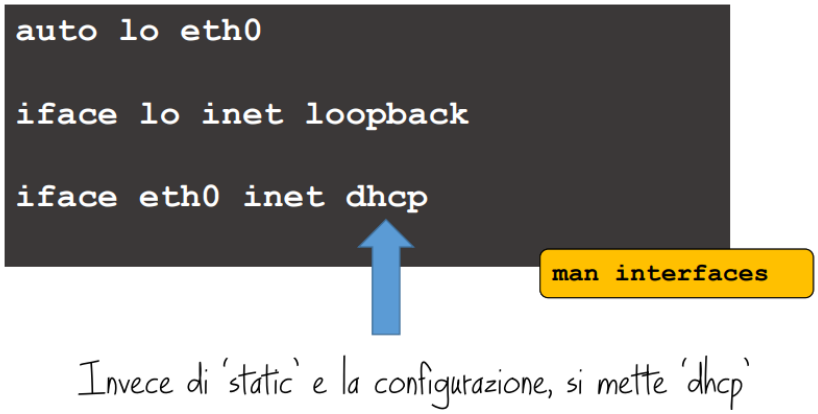
- Si indica l'indirizzo o gli indirizzi per i DNS
- Quali sono i router
- Il lease time (che appare per la prima volta in DHCPOFFER)
- Il "panierino", cioè il range di indirizzi IP che il DHCP server che stiamo configurando può assegnare alle macchine che si collegano alla rete.

Osservazione. A seguito delle modifiche è necessario riavviare il processo relativo all'applicazione che gestisce il DHCP server.

```
systemctl restarti sc-dhcp-server.service
```

12.10.4 Differenze nel file interfaces

Nel file `/etc/network/interfaces` emerge una semplificazione a proposito della configurazione dell'interfaccia di rete eth0.

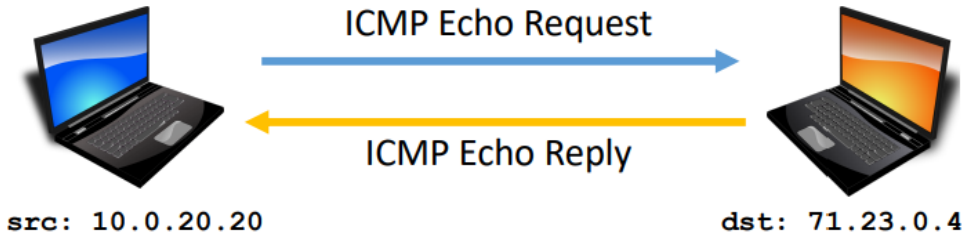


12.11 Test di connettività: comando ping

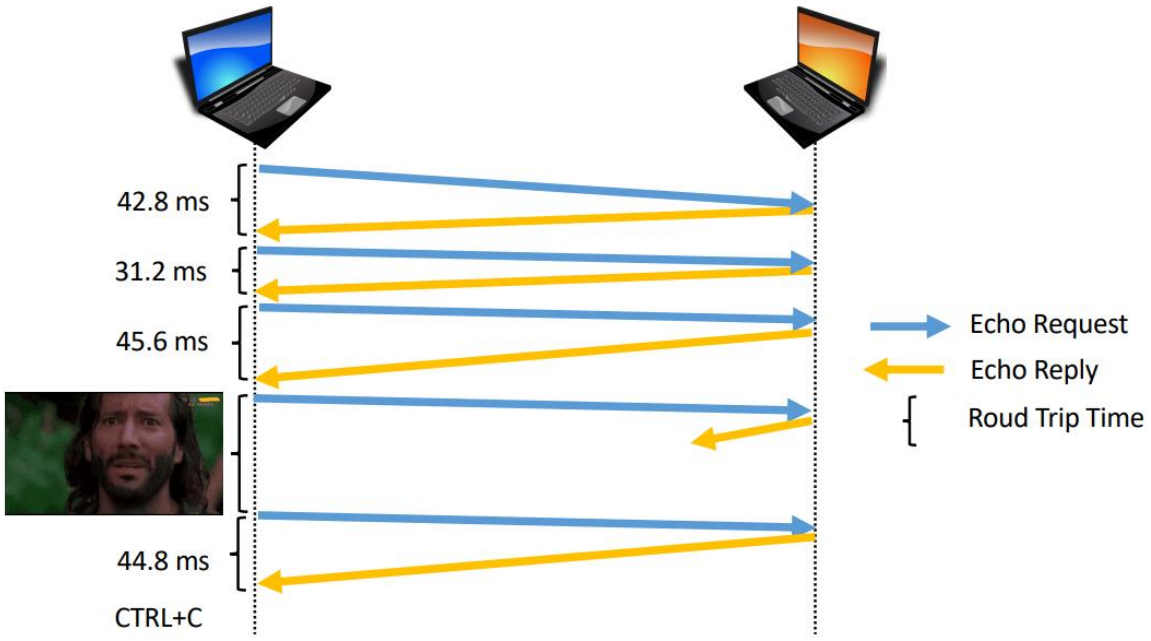
Il comando ping permette di testare la connettività tra l'host che lo esegue e un host remoto. Esempio:

```
$ ping www.unipi.it
$ ping 192.168.2.34
```

Il comando invia un messaggio *ICMP Echo Request* e attende dall'host remoto un messaggio *ICMP Echo Reply* (in sostanza fare ping significa inviare un pacchetto a un destinatario e vedere se questo risponde). Misura il tempo impiegato a raggiungere l'host destinatario e a ritornare indietro (il *Round Trip Time*, lo misura in millisecondi).



ICMP è acronimo di *Internet Control Message Protocol*, appunto un protocollo di servizio (incapsulato nel protocollo IP) che permette di rilevare malfunzionamenti. In realtà il comando non invia un solo messaggio, ma una sequenza di richieste, che saranno utilizzate per generare le statistiche restituite in output.



Gli invii continuano finché non si interrompe il comando con la combinazione CTRL + C (a meno che non si indichi con apposito parametro il numero di richieste da inviare, si veda più avanti). Se entro un certo tempo (un timeout) non arriva il messaggio di risposta lo si segnala e si passa al pacchetto successivo.

Due ipotesi:

- *ICMP Echo Request* non è arrivato al destinatario;
- *ICMP Echo Request* è arrivato al destinatario e questo ha trasmesso *ICMP Echo Reply*, ma ICMP Echo Reply non è arrivato a chi ha lanciato il comando ping (oppure arriva fortemente in ritardo).

Quello che si deve tenere a mente è che un errore non implica automaticamente l'assenza dell'host. Rientrano tra i possibili errori i seguenti:

- ***Network unreachable***
  L'host locale non ha route valide per raggiungere l'host remoto.
- ***100% packet loss***
  L'host locale non ha ricevuto alcun pacchetto di risposta.
- ***Unknown host***
  Non è stato possibile risolvere il nome dell'host specificato.

```
> ping www.google.com

PING www.google.com (172.217.21.68): 56 data bytes
64 bytes from 172.217.21.68: icmp_seq=0 ttl=114 time=25.812 ms
64 bytes from 172.217.21.68: icmp_seq=1 ttl=114 time=25.195 ms
64 bytes from 172.217.21.68: icmp_seq=2 ttl=114 time=25.446 ms
64 bytes from 172.217.21.68: icmp_seq=3 ttl=114 time=18.834 ms
64 bytes from 172.217.21.68: icmp_seq=4 ttl=114 time=26.525 ms
64 bytes from 172.217.21.68: icmp_seq=5 ttl=114 time=23.773 ms
64 bytes from 172.217.21.68: icmp_seq=6 ttl=114 time=27.082 ms
64 bytes from 172.217.21.68: icmp_seq=7 ttl=114 time=25.157 ms
64 bytes from 172.217.21.68: icmp_seq=8 ttl=114 time=26.405 ms
64 bytes from 172.217.21.68: icmp_seq=9 ttl=114 time=23.274 ms
^C
--- www.google.com ping statistics ---
10 packets transmitted, 10 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 18.834/24.750/27.082/2.268 ms
```

Concludiamo segnalando alcuni parametri opzionali per il comando *ping*:

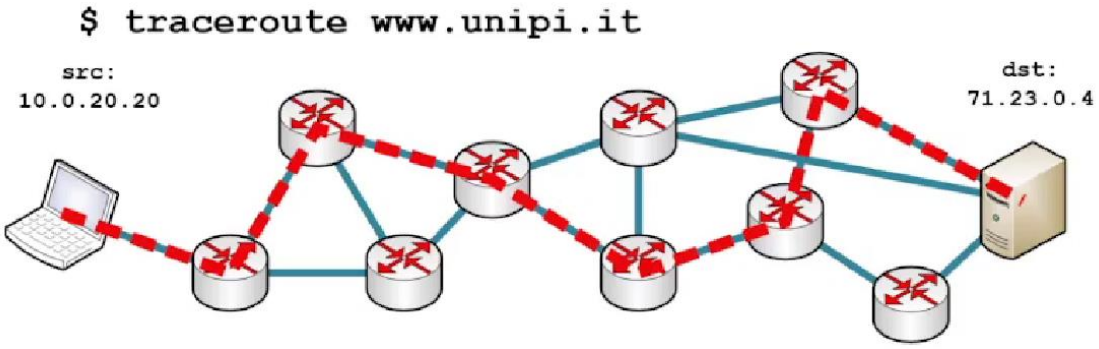
- Opzione **`-c`** (*count*)
  - Specifica il numero di richieste da inviare
- Opzione **`-i`** (*interval*)
  - Specifica l'intervallo tra le richieste
- Opzione **`-q`** (*quiet*)
  - Visualizza solamente le statistiche finali
- Opzione **`-s`** (*size*)
  - Dimensione in byte del pacchetto, *al netto* degli header ICMP di 8 byte

12.12 Percorso del pacchetto: comando traceroute

12.12.1 Premessa: il shortest path

Il comando traceroute restituisce il percorso che un pacchetto IP effettua per raggiungere un host destinatario. Il termine chiave è *hop*: salti, il numero di passaggi compiuti dai nostri pacchetti tra i router per raggiungere il destinatario (fino al default gateway abbiamo certezza sul percorso dei pacchetti, ma passato quello quali router attraverseranno i nostri pacchetti?).

Osservazione: il percorso cambia ogni volta, secondo criteri di ottimizzazione. Esiste un algoritmo di ottimizzazione che prendendo la distanza individua il *shortest path*: Dijkstra!



Con che criteri definiamo la distanza?

1. Il numero di hop, cioè il numero di passaggi tra router necessari per arrivare a destinazione.
2. La distanza geografica.
3. Il livello di congestione sui collegamenti (occupazione della banda).

La realtà è chiaramente più complessa rispetto ai problemi di Ricerca Operativa. Si parla di *multi-object optimization* (si ottimizzano più oggetti, roba che vedremo alla magistrale) e le priorità dei criteri dipende dal contesto: prevale, ad esempio, il criterio (3) in momenti della giornata in cui la rete è utilizzata maggiormente. Riprenderemo questo argomento più avanti!

12.12.2 Campo TTL nel datagram IP

Nel datagram IP è presente il campo TTL che specifica il numero di router che il pacchetto può attraversare:

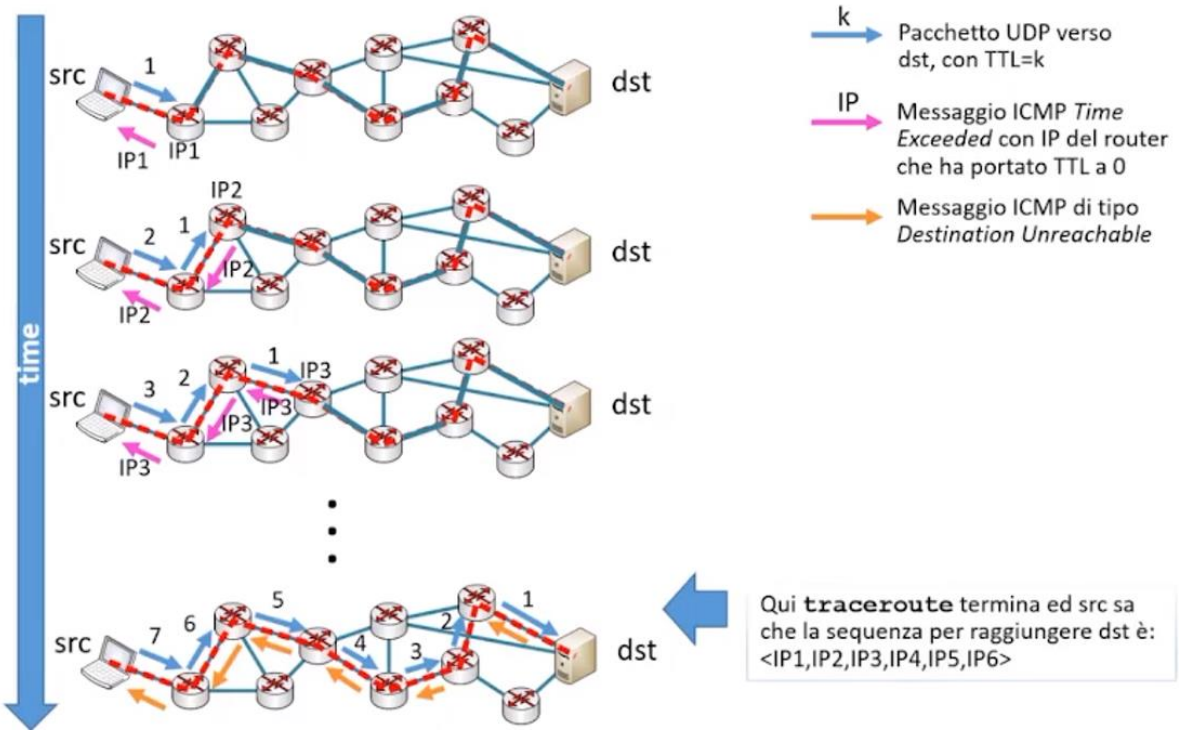
- il parametro è decrementato ad ogni passaggio da un router;
- quando avremo TTL nullo il router coinvolto scarta il pacchetto e invia al relativo mittente un messaggio di errore ICMP di tipo *Time Exceeded* (indicando anche il suo indirizzo).

12.12.3 Idea alla base di traceroute

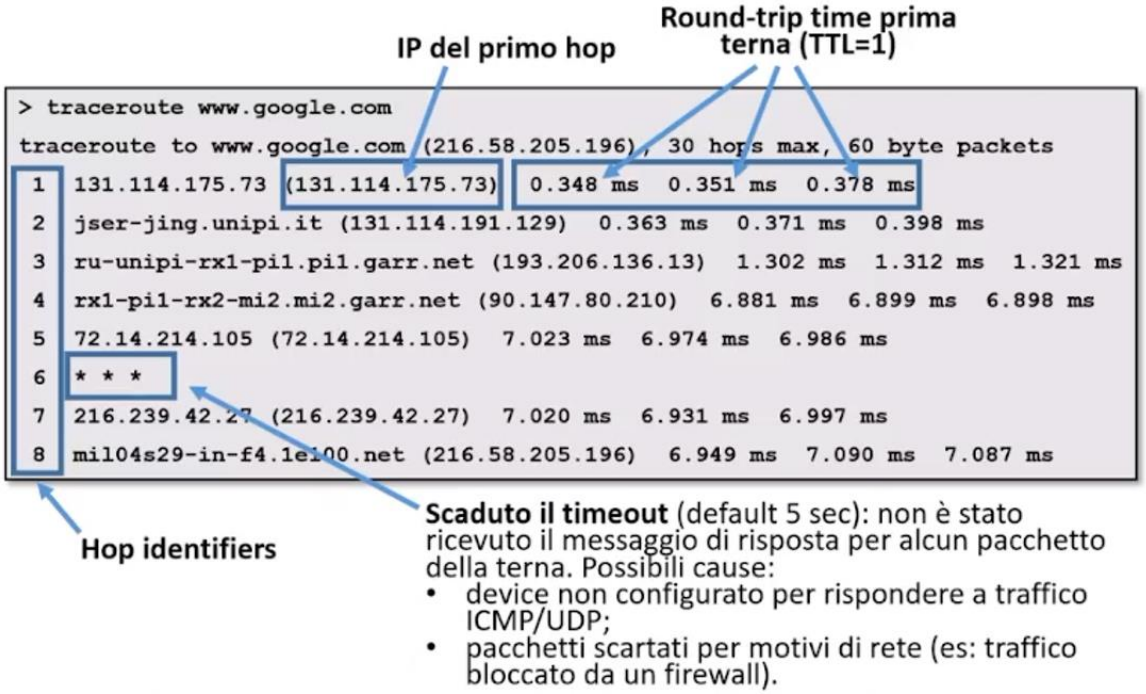
Il comando *traceroute* sfrutta il campo TTL e l'errore Time Exceeded per costruire il percorso effettuato dai pacchetti.

- Si invitano terne terne di pacchetti.
- La prima terna ha TTL uguale ad 1, la successiva uguale a 2, e così via.
- Per ogni pacchetto della terna si invia *ICMP Time Exceeded* quando TTL assume valore nullo. Con l'invio di Time Excedeed si invia anche l'indirizzo IP del router. Il mittente (cioè il dispositivo che ha lanciato traceroute) tiene da parte l'indirizzo IP.
- Si prosegue con l'invio di terne di pacchetti fino a quando non si arriverà a destinazione. Quando un pacchetto arriva a destinazione viene inviato un *ICMP Destination Unreachable*. Si consideri che:
  - l'invio dei pacchetti avviene via UDP
  - UDP richiede di indicare a porta
  - è improbabile beccarne una in ascolto indicando numeri di porta alla cieca.

  Se si dice che la porta è unreachable significa che siamo arrivati a destinazione.



Al termine dell'esecuzione avremo costruito un percorso di router grazie agli indirizzi IP inviati con gli errori *ICMP Time Exceeded*.



Nell'hop 6 è scaduto il timeout per ogni pacchetto della terna. Il router associato all'hop 6 riceve ICMP Echo Request, ma per qualche motivazione (le principali tra le possibili sono quelle elencate in figura) il mittente delle terne non riceve *ICMP Time Exceeded*.

12.12.4 Difetti

Il comando è sicuramente utile per costruire delle statistiche, in particolare può essere interessante per avere un'idea su quali aree della rete siano congestionate (nell'esempio precedente occhio all'hop 4).

Si tenga a mente un difetto rilevante: il percorso determinato dagli algoritmi di ottimizzazione può cambiare durante l'esecuzione del comando (cosa non strana se la rete è congestionata). Questo significa che potrei avere indirizzi IP relativi ad un percorso e indirizzi IP relativi a percorsi diversi! Si inviano terne e non singoli pacchetti anche per rimediare a questo difetto: la media dei RTT permette di ottenere un dato decisamente più realistico.

13 LABORATORIO: PROGRAMMAZIONE

13.1 Introduzione al linguaggio C e differenze rispetto al C++

13.1.1 Definizione di variabili

Quando utilizziamo il C dobbiamo tenere a mente le seguenti limitazioni:

- si richiede di dichiarare le variabili tutte in cima a un blocco
- si richiede l'uso della keyword *struct* quando si istanzia una variabile di tipo struttura

```
struct Complex {
      double Re;
      double Im;
}

int main() {
      int a = 5, i;
      int b = f(a);
      struct Complex c;
      a = func(1000);
      // …
      for(i = 0; a < 100; i++) {
            int c = 0;
            b = f(a);
      }
}
```

13.1.2 Memoria dinamica (*heap*, funzioni `malloc` e `free`)

Il C non offre le keyword per l'allocazione di memoria dinamica viste in C++ (non c'è la new, o la delete). Facciamo le stesse cose utilizzando le seguenti funzioni:

- `void *malloc(size_t size)`
  Data la dimensione del blocco di memoria, espressa in bytes, restituisce l'indirizzo dell'area di memoria allocata. Restituisce NULL in caso l'allocazione non abbia successo.
- `void free(void *ptr)`
  Viene deallocato il blocco di memoria a cui punta il puntatore posto in ingresso. Se si pone NULL in ingresso non succede niente. Non restituisce niente.

Per poter eseguire queste funzioni è necessario includere la libreria `stdlib`. Due possibili situazioni:

- **Allocazione di un'area di byte generica**
  Metodo standard in cui non teniamo conto di particolari tipi (otteniamo un mero puntatore ad un'area di byte).

```
#include <stdlib.h>

int main() {
  int dimensione = 5;
  void* punt;
  punt = malloc(dimensione);
  if (punt == NULL) {
    // Gestione errore
  }
  // …
  free(punt);
}
```

alloca un'area di memoria di 5 byte che sarà puntata da *punt*

rilascia l'area di memoria di 5 byte

- **Allocazione di un array di elementi di un certo tipo (allocazione con coercizione)**
  Supponiamo adesso di voler allocare un array di particolari elementi, per esempio un array di double. Prendiamo il codice visto del metodo standard e alteriamo:
  - tipo del puntatore (si mette il double)
  - riga della `malloc` (convertiamo da void* a double* e indichiamo la dimensione dell'area di memoria in funzione di due parametri: dimensione in byte del singolo double e numero di elementi dell'array)

```
#include <stdlib.h>

int main() {
  int dimensione = 10;
  double* punt;

  punt = (double*)malloc(sizeof(double)*dimensione)

  if (punt == NULL) {
    // Gestione errore
  }
  // ...
  free(punt);
}
```

l'area di memoria è trattata come array di double

numero di byte da allocare (8*10=80)

13.1.3 Input/Output (`printf` e `scanf`)

In C non possiamo utilizzare lo *stream* come in C++. Ricorriamo alle funzioni della libreria `stdio.h`

- `int printf(char* formato, lista_argomenti)`
  Dato il formato e un eventuale elenco di argomenti restituisce il numero di caratteri stampati.

Stampa **`lista_argomenti`** sullo standard-output (video) nel **`formato`** ricevuto come primo parametro.

Restituisce il numero di caratteri stampati

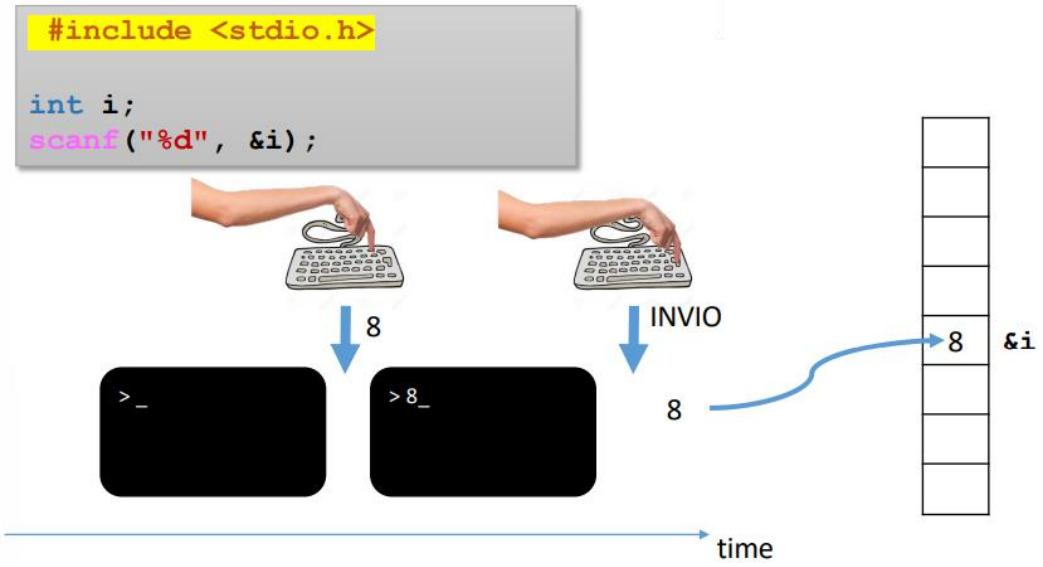
NOTA: Se si usa, per esempio, "%3.2f" si stampa un valore in virgola mobile con 3 cifre e 2 cifre di precisione (dopo la virgola)

Formato	Cosa viene stampato
%d, %i	Intero decimale
%f	Valore in virgola mobile
%c	Un carattere
%s	Una stringa di caratteri
%o	Numero ottale
%x, %X	Numero esadecimale
%u	Intero senza segno
%f	Numero reale (float o double)
%e, %E	Formato scientifico
%%	Carattere '%'

```
#include <stdio.h>

char* str = "Com'è bello il C!\n";
// Es. 1
printf(str);
// Es. 2
printf("Messaggio:\n %s", str);
// Es. 3
printf("Da oggi pane e C!");

int i = 5;
// Es. 4
printf("i = %d\n", i);
```

```
// Es. 1
> Com'è bello il C!
_

// Es. 3
> Da oggi pane e C!_

// Es. 2
> Messaggio:
  Com'è bello il C!
_

// Es. 4
> i = 5
_
```

- `int scanf(char* formato, lista_indirizzi)`
  Legge dalla tastiera una sequenza di caratteri (cifre o lettere) e li memorizza agli indirizzi ricevuti in `lista_indirizzi` nel `formato` ricevuto come primo argomento.



13.1.4 Stringhe (`strlen`, `strcmp`, `strcpy`, `strcat`)

Per prima cosa ricordiamo che una stringa è immaginabile come un array di caratteri char, quindi introduciamo un puntatore char*

```
char* str1 = "Gatto \n";
```

Si ricordi che al termine di ogni stringa viene sempre aggiunto il carattere di terminazione \0 (quindi se memorizziamo una stringa di n caratteri avremo un array di $n + 1$ elementi char).

Introduciamo le funzioni principali per la manipolazione delle stringe (dalla libreria `string.h`)

- `size_t strlen(char *str)`
  Dato il puntatore alla stringa otteniamo la lunghezza della stringa. Ricordarsi che la lunghezza della stringa non è il numero effettivo di caratteri memorizzati (il carattere di terminazione non è considerato nel conteggio).

```
#include <string.h>

// Lunghezza
char* str1 = "Gatto \n";
int len;
len = strlen(str1);
```

G	a	t	t	o		\n	\0

8 byte allocati, ma `strlen()` restituisce **7**!

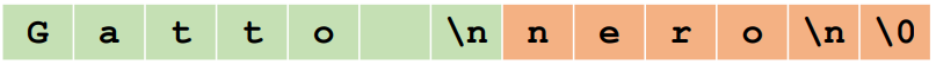
- `int strcmp(char *str1, char *str2)`
  Funzione che permette la comparazione di due stringhe (`str1` ed `str2`). L'esito del confronto è espresso per mezzo dell'intero restituito:
  - 0 se le stringhe sono identiche;
  - intero < 0 se le due stringhe sono diverse e il primo carattere di `str1` diverso dal corrispondente carattere di `str2` ha codifica ASCII minore
  - intero > 0 se le due stringhe sono diverse e il primo carattere di `str1` diverso dal corrispondente carattere di `str2` ha codifica ASCII minore

```
#include <string.h>

// Confronto
char* str2 = "Gattone \n";
int ret;
ret = strcmp(str1, str2); // -1
```

- `char *strcpy(char *dest, char *src, int size)`
  Funzione che permette di copiare una stringa da un'area di memoria a un'altra. Attenzione a `size`: si deve considerare anche il carattere di terminazione nel conteggio, altrimenti copieremo la stringa senza il carattere di terminazione.

```
#include <string.h>

// Copia
char str1[20];
n = sizeof(str1);
strcpy(str1, "Gatto \n", n);
```

- `char * strcat (char* str1, char* str2, int str2_size);`
  Funzione che permette di concatenare la stringa `str2` al termine della stringa `str1`. Il carattere di terminazione di str1 verrà sovrascritto dal primo carattere di `str2`. Anche per quanto riguarda str2_size si deve considerare il carattere di terminazione.




```
#include <string.h>

// Copia
char str1[20];
n = sizeof(str1);
strncpy(str1, "Gatto \n", n);

// Concatenazione
char* str2 = "nero\n";
strncat(str1, str2, 6);
```

13.1.5 Gestione dei file (`fopen`, `fscan`, `fprint`)

Per l'apertura di un file ricorriamo a una funzione della libreria `stdio.h`

`FILE *fopen(char* file_name, char* mode_of_operation);`

Funzione per l'apertura di un file. Si pone come primo parametro il percorso del file (con nome ed estensione del file), come secondo parametro la modalità di apertura del file.

Tra le possibili modalità abbiamo le seguenti:

- r: sola lettura
- w: sola scrittura (il file viene creato se non esiste)
- r+: lettura e scrittura
- a: append (il file viene creato se non esiste)
- a+: lettura e append

La funzione restituisce NULL se qualcosa va storto.

Nel seguente esempio abbiamo il lanciato delle funzioni `fscanf` ed `fprintf`, equivalenti di scanf e printf: l'unica differenza sta in ulteriore parametro dove indichiamo il file coinvolto nell'operazione, precedentemente aperto (si interviene sul file e non su stdin o stdout).

```c
#include <stdio.h>

// Lettura da file
int ret, n;
FILE* fd1;
fd1 = fopen("/tmp/gatti.txt", "r");
ret = fscanf(fd1, "%d", &n);

// Scrittura su file
char* str = "Gatto!\n";
FILE* fd2;
fd2 = fopen("/tmp/gatti2.txt", "w");
ret = fprintf(fd2, "%s", str);
```

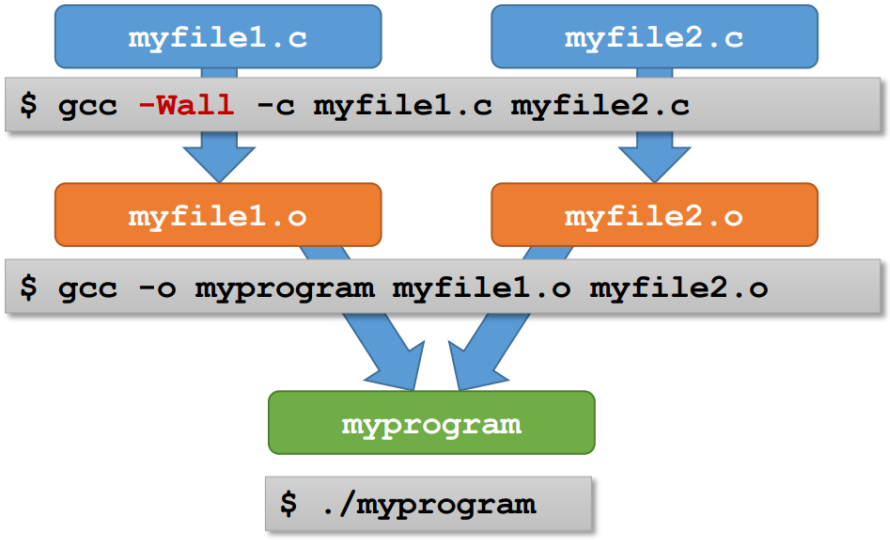
13.2 Compilazione con GNU

Per la compilazione utilizzeremo la GNU Compiler Collection (GCC).

Con la **compilazione**, si creano i *file oggetto* a partire dai *file sorgente*

Con il **linking**, si crea un *file eseguibile* a partire dai *file oggetto*

Esecuzione



13.3 Nozioni iniziali di programmazione distribuita

13.3.1 Modalità di cooperazione e socket

Ricordiamo le modalità di cooperazione tra processi:

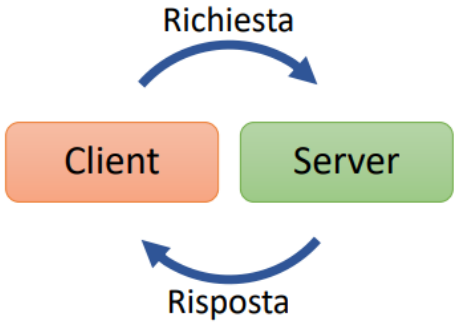
- **Sincronizzazione**
  - Semafori
- **Comunicazione**
  - Memoria condivisa
  - Chiamata a procedura remota
  - Scambio di messaggi

La novità rispetto a quanto già visto in altri corsi è l'interazione tra processi appartenenti a dispositivi diversi (si parla di sistemi distribuiti). Introduciamo a tal proposito il socket, ossia un'astrazione implementata dal sistema operativo e atta a permettere la comunicazione tra processi in un contesto distribuito.

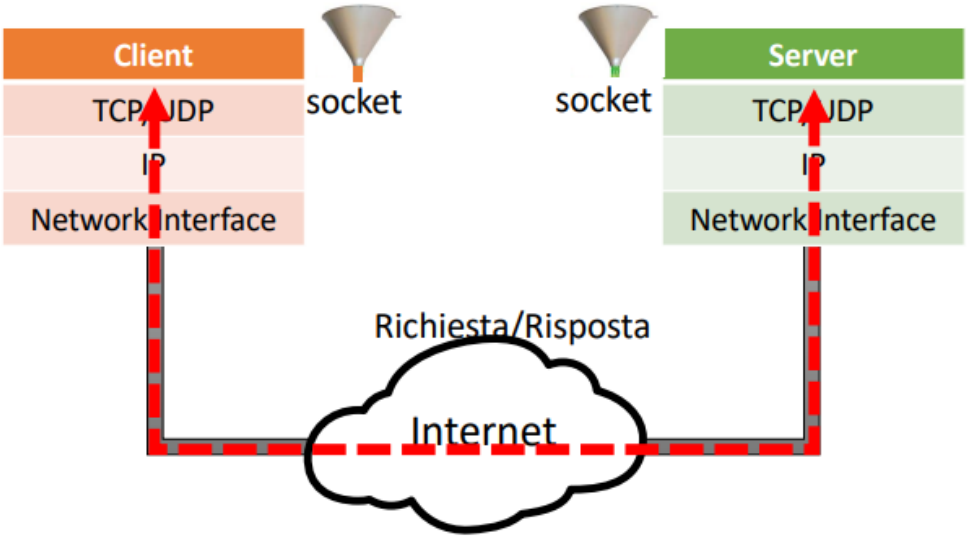
13.3.2 Modello client-server

L'astrazione del socket si basa sul paradigma client-server, che prevede una comunicazione bidirezionale tra un client e un server.

- Il client trasmette una richiesta al server
- Il server trasmette una risposta al client



I socket sono lanciati da livello applicazione: il messaggio attraversa i livelli inferiori della pila protocollare fino a raggiungere il livello fisico (attraversa Internet), arrivato a destinazione percorre a ritroso la pila protocollare raggiungendo il livello applicazione. Sia il mittente che il server lanciano dei socket, ergo possiamo immaginarli come le estremità di un canale di comunicazione.



13.3.3 Primitive per la gestione dei socket

Il sistema operativo fornisce le primitive per:

- creazione del socket;
- assegnazione di un indirizzo e di una porta al socket;
- connessione ad un altro socket;
- accettazione di una connessione;
- invio e ricezione di dati per mezzo del socket.

Andiamo nel dettaglio.

13.3.3.1 Primitiva `socket`

La prima primitiva che andiamo a vedere è quella atta alla creazione del socket

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol)
```

- Pongo in ingresso:
  - *domain*, la famiglia di protocolli utilizzati
    - AF_LOCAL, Comunicazione locale (comunicazione sulla stessa macchina)
    - AF_INET, Protocolli IPv4, TCP e UDP
  - *type*, tipologia di connessione
    - SOCK_STREAM, connessione stream basata sul protocollo TCP
    - SOCK_DGRAM, connessione datagram basata sul protocollo UDP
  - protocol, porre uguale a 0 senza stare porsi il perché (semplifichiamoci la vita)
- Il valore restituito è il descrittore di file (che rappresenta il socket), che va messo da parte per poter manipolare il socket attraverso altre primitive. Viene restituito il numero -1 nel caso in cui ci sia stato un errore.
- Si osservi che dopo l'esecuzione di questa primitiva abbiamo creato il socket, ma non è ancora associato a un indirizzo IP o a una porta.

13.3.3.2 Strutture per definire gli indirizzi (`sockaddr_in` e `in_addr`)

Nella configurazione dei socket è necessario conoscere le seguenti strutture, che presentano al loro interno parte della configurazione del socket esterno. Abbiamo:

- La famiglia di protocolli (di nuovo, indicheremo sempre AF_INET)
- L'indirizzo IP e la porta (attenzione, **espressi in network order – occhio all'endianess**)[15]

man 7 ip

```
#include <sys/socket.h>
#include <netinet/in.h>

/* Struttura per memorizzare l'indirizzo di un socket */
struct sockaddr_in {
  sa_family_t    sin_family;  /* Address Family: AF_INET */
  in_port_t      sin_port;    /* Port (in network order) */
  struct in_addr sin_addr;    /* IP Address (istanza di struct in_addr)
*/
};

/* Struttura per memorizzare un IP address*/
struct in_addr {
  uint32_t s_addr;           /* IP Address (in network order)*/
};
```

Esiste anche la struttura **`sockaddr`** (usata da alcune funzioni descritte nel seguito), ma non la useremo direttamente.

[15] Indirizzo IP e porta di cosa? Si veda più avanti quando parleremo meglio di client e server.

13.3.3.3 endianess: concetti di base e funzioni di conversione ("funzioni ninja")

Sappiamo da Calcolatori elettronici che nel memorizzare una sequenza di byte dobbiamo decidere se:

- porre all'indirizzo meno significativo il byte meno significativo (memorizzazione little-endian)
- porre all'indirizzo meno significativo il byte più significativo (memorizzazione big-endian)

Memorizzazione *big-endian* - Per primo il byte più significativo (MSB)

Indirizzo A	Indirizzo A+1	Indirizzo A+2	Indirizzo A+3
0000 0000	0000 0110	0111 0100	0100 1110

Memorizzazione *little-endian* - Per primo il byte meno significativo (LSB)

Indirizzo A	Indirizzo A+1	Indirizzo A+2	Indirizzo A+3
0100 1110	0111 0100	0000 0110	0000 0000

Sappiamo che:

- in rete è adottata la memorizzazione big-endian (**regola fissa**);
- nell'host dipende (Intel usa little-endian, mentre IBM usa big-endian)

Questo significa che se l'host adotta memorizzazione little-endian dobbiamo effettuare un'operazione di conversione prima di trasmettere la sequenza di byte in rete, stessa cosa l'host riceve una sequenza di byte dalla rete. Ricorriamo alle seguenti funzioni di conversione (denominate "funzioni ninja" da quel simpaticone di Pistolesi):

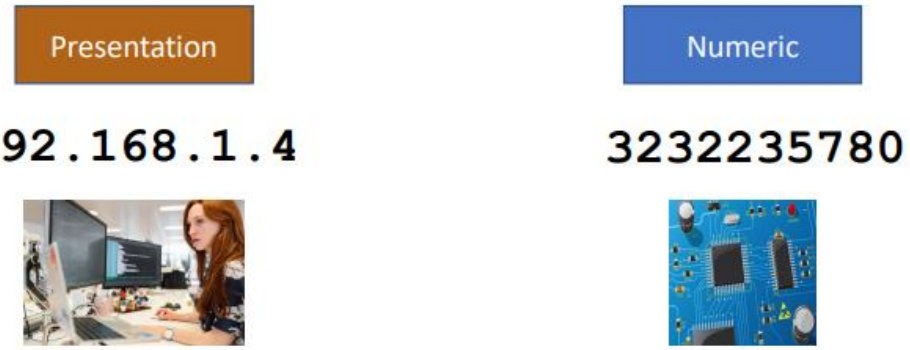
```
#include <arpa/inet.h> // funzioni ninja
#include <stdint.h> // tipi uint16_t e uint32_t
```

- ***host to network long***
  `uint32_t htonl(uint32_t hostlong);`
- ***host to network short***
  `uint16_t htons(uint16_t hostshort);`
- ***network to host long***
  `uint32_t ntohl(uint32_t netlong);`
- ***network to host short***
  `uint16_t ntohs(uint16_t netshort);`

dove con `uint` intendiamo unsigned int (intero senza segno). I due tipi diversi permettono di controllare la dimensione delle variabili, in base alle nostre esigenze.

13.3.3.4 Formato degli indirizzi IP e passaggio da un formato a un altro (`inet_pton` e `inet_ntop`)

Gli indirizzi IP possono essere rappresentanti in due forme:



- **Formato numerico (usato dal calcolatore)**
  Rappresentiamo l'indirizzo come un intero a 32 bit, una semplice sequenza di bit
- **Formato presentazione (usato dagli esseri umani)**
  Rappresentiamo l'indirizzo per mezzo di notazione decimale puntata.

Per mezzo delle seguenti funzioni gestiamo il passaggio da un formato a un altro:

- **Passaggio da formato presentazione a formato numerico**
  `int inet_pton(int af, const char* src, void* dst);`
  - Pongo in ingresso:
    - `af`: famiglia dei protocolli (again, AF_INET)
    - `src`: stringa contenente l'indirizzo IP posto in formato presentazione
    - `dst`: puntatore a un'istanza di struttura `in_addr` (si veda l'esempio di inizializzazione di socket, più avanti)
  - Non ci interessa il valore restituito.

- **Passaggio da formato numerico a formato presentazione**
  `const char* inet_ntop(int af, const void* src, char* dst, socklen_t size);`
  - Pongo in ingresso:
    - `af`: famiglia dei protocolli (again, AF_INET)
    - `src`: puntatore a un'istanza di struttura `in_addr`
    - `dst`: puntatore a un buffer di caratteri avente lunghezza size
    - `size`: dimensione del buffer dovrà verrà memorizzato l'indirizzo IP in formato presentazione (può variare in base all'architettura con 16, 32 o 64 bit, per comodità porremo come valore `INET_ADDRSTRLEN` – 16 bit).
  - Non ci interessa il valore restituito.

13.3.4 Snippet di codice con creazione e inizializzazione di un socket

Introdotte le primitive iniziali possiamo commentare un esempio di codice

```
#include <arpa/inet.h> // funzioni ninja
#include <sys/types.h> // costanti utilizzate in alcune primitive
#include <sys/socket.h> // socket()
#include <netinet/in.h> // inet_pton() e strutture sockaddr_in e in_addr

int main () {

      // Ricordarsi che tutte le variabili devono essere definite in cima

      // Creazione del socket con l'apposita primitiva
      int sd = socket(AF_INET, SOCK_STREAM, 0);

      // Creazione della variabile struttura contenente indirizzi e porta
      struct sockaddr_in indirizzo;

      // E' buona regola fare pulizia dell'area di memoria utilizzata
      memset(&indirizzo, 0, sizeof(indirizzo));

      // Famiglia dei protocolli (IPv4)
      indirizzo.sin_family = AF_INET;

      // Port inizializzata a 4242, in network order (funzione ninja!!!)
      indirizzo.sin_port = htons(4242);

      // IP address "192.168.4.5", convertito in formato numerico
      inet_pton(AF_INET, "192.168.4.5", &indirizzo.sin_addr);

      // ...
}
```

Arrivati a questo punto abbiamo creato il socket, definito indirizzo IP e porta. Osservazioni:

- non abbiamo ancora detto indirizzo IP e porta di cosa, e non li abbiamo associati al socket;
- non abbiamo introdotto le primitive necessarie per gestire l'interlocuzione vera e propria.

13.4 Programmazione distribuita lato server

13.4.1 Processo server

Quando parliamo di applicazione distribuita alludiamo a quanto eseguito sia nel client sia nel server. Parlare di programmazione distribuita lato server significa porre il focus su quanto eseguito nel server.

- Con *processo server* alludiamo al programma in esecuzione sulla macchina server, precisamente nello strato applicazione.
- Il processo server offre uno o più servizi e fa uso di system call per utilizzare i socket.

Le system call utilizzano primitive del sistema operativo.

13.4.2 Primitiva `bind` per associare indirizzo IP e porta al socket

La primitiva bind permette di associare il socket a un indirizzo IP e a una porta. **Precisamente si vanno a stabilire indirizzo IP e porta dove il server riceve richieste di connessione**.

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

- Pongo in ingresso:
  - `sockfd`: descrittore del socket (*socket file descriptor*)
  - `addr`: puntatore a struttura di tipo `sockaddr` dove indicheremo indirizzo IP e porta dove il server riceve le richieste di connessione
  - `addrlen`: dimensione di addr
- Viene restituito 0 se l'associazione ha successo, -1 in caso di errore.

Porremo come riga di codice quanto segue:

```
ret = bind(sd, (struct sockaddr*)&my_addr, sizeof(my_addr));
```

Si osservi che nel secondo parametro occorre cast del puntatore (lavoriamo solo con `sockaddr_in`).

13.4.3 Primitiva `listen` per definire un socket passivo (in ascolto)

La primitiva listen permette di dichiarare un socket appena creato come socket passivo, cioè un socket che si pone in ascolto e attende richieste da parte di altri host.

```
int listen(int sockfd, int backlog);
```

- Pongo in ingresso:
  - `sockfd`: descrittore del socket (*socket file descriptor*)
  - backlog: dimensione della coda, cioè il numero massimo di richieste che possono essere messe in attesa di gestione[16].
- Viene restituito 0 se la dichiarazione ha successo, -1 in caso di errore.

Si osservi che questa primitiva **NON E' BLOCCANTE**: è solo la dichiarazione del socket come passivo.
Porremo come riga di codice quanto segue

```
ret = listen(sd, 10);
```

13.4.4 Primitiva `accept` per l'accettazione di richieste e definizione di socket di comunicazione

La primitiva accept permette di gestire la coda degli host che hanno lanciato richiesta.

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

- Pongo in ingresso:
  - `sockfd`: descrittore del socket (*socket file descriptor*)
  - `addr`: puntatore a una struttura (vuota) di tipo `sockaddr`, dove la primitiva andrà a memorizzare l'indirizzo del client di cui è stata accettata la richiesta

[16] Non si confonda la cosa col numero di richieste gestite in simultanea dall'host. Per comodità lavoreremo con un *server iterativo*, cioè un server che gestisce una richiesta alla volta.

- `addrlen`: puntatore a una variabile dove la primitiva andrà a memorizzare la dimensione della struttura passata in addr.

- Il valore restituito è il descrittore di un nuovo socket detto ***socket di comunicazione***, che useremo per lo scambio di dati. In caso di errore viene restituito -1.
- **PRIMITIVA BLOCCANTE**: il socket è di default bloccante.
  - Il programma si ferma in attesa di una richiesta di connessione, se la coda è vuota.
  - Se invece sono presenti richieste il programma non si ferma, e viene gestita subito la prima richiesta in coda secondo logica FIFO.

13.4.5 Snippet di codice con uso delle primitive lato server

Riprendiamo lo snippet di codice già visto nella fase introduttiva, e utilizziamo le nuove primitive.

```
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main () {

    int ret, sd, new_sd, len;
    struct sockaddr_in my_addr, client_addr; // Strutture per gli IP
    /* Creazione socket */
    sd = socket(AF_INET, SOCK_STREAM, 0);

    /* Creazione indirizzo socket*/
    memset(&my_addr, 0, sizeof(my_addr); // Pulizia
    my_addr.sin_family = AF_INET ;
    my_addr.sin_port = htons(4242);
    inet_pton(AF_INET, "192.168.4.5", &my_addr.sin_addr);
    // In alternativa, per mettere in ascolto il server su tutte le
    interfacce: my_addr.sin_addr.s_addr = INADDR_ANY;

    ret = bind(sd, (struct sockaddr*)&my_addr, sizeof(my_addr));
    ret = listen(sd, 10);
    len = sizeof(client_addr);
    new_sd = accept(sd, (struct sockaddr*)&client_addr, &len);

    //...
 }
```

13.5 Programmazione distribuita lato client

13.5.1 Primitiva `connect` per l'invio di una richiesta di connessione

La primitiva connect viene lanciata nel processo client per inviare una richiesta di connessione tra un socket locale (quello presente nel client) e un socket remoto (quello presente nel server). Ribadiamo con questa introduzione quanto segue:

- il socket rappresenta un estremo del canale di comunicazione;
- server e client creano ciascuno un loro socket prima di andare a stabilire la connessione.

La connessione viene richiesta dal client con la primitiva che stiamo introducendo. Il server accetta la connessione con la primitiva accept, introdotta nelle pagine precedenti.

```
int connect(int sockfd, const struct sockaddr* addr, socklen_t addrlen);
```

- Parametri di ingresso:
  - `sockfd`: descrittore del socket (*socket file descriptor*)
  - `addr`: puntatore alla struttura contenente l'indirizzo del socket remoto (anche qua cast sul puntatore, dato che noi lavoriamo solo con `sockaddr_in`)
  - `addrlen`: dimensione di addr
- La primitiva restituisce 0 se la connessione viene stabilita con successo, -1 in caso di errore.
- **Primitiva bloccante**! Il programma si ferma in attesa che la richiesta di connessione sia accettata.

```
ret = connect(sd, (struct sockaddr*)&sv_addr, sizeof(sv_addr));
```

13.5.2 Snippet di codice con uso della precedente primitiva

Si osservi che la prima parte di codice eseguita nel processo client è letteralmente la stessa eseguita nel processo server (la differenza sta nell'indirizzo IP e nella porta indicati). L'unica differenza si ha nell'invocazione della connect al posto delle primitive tipiche del server.

```
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main () {

      int ret, sd; struct sockaddr_in server_addr; // per il server

      /* Creazione socket */
      sd = socket(AF_INET, SOCK_STREAM, 0);

      /* Creazione indirizzo del server */
      memset(&server_addr, 0, sizeof(server_addr); // Pulizia
      server_addr.sin_family = AF_INET ;
      server_addr.sin_port = htons(4242);
      inet_pton(AF_INET, "192.168.4.5", &server_addr.sin_addr);
      ret = connect(sd, (struct sockaddr*)&server_addr,
      sizeof(server_addr));

      // ...

}
```

13.6 Scambio di dati

13.6.1 Primitiva `send` per l'invio di un messaggio

La primitiva `send` permette l'invio di un messaggio con un socket connesso (non quello di ascolto, occhio).

```
ssize_t send(int sockfd, const void* buf, size_t len, int flags);
```

- **Parametri di ingresso**:
  - `sockfd`: descrittore del socket
  - `buf`: puntatore al buffer contenente il messaggio da inviare (occhio, non il buffer del kernel)
  - `len`: dimensione del messaggio in byte
  - `flags`: per ora teniamolo a zero.
- La primitiva restituisce il numero di byte inviati, -1 in caso di errore. Si osservi che un invio parziale non è errore (dobbiamo confrontare il numero di byte inviati col numero di byte del messaggio).
- **La primitiva è bloccante!!** Il programma si ferma finchè non ha scritto tutto il messaggio (si suppone idealmente che non avremo il problema di trasmettere messaggi di dimensione tale da saturare il buffer, in generale il programma può fermarsi anche se il buffer risulta saturato).

Si consideri il seguente esempio di snippet di codice

```
int ret, sd, len;
char buffer[1024];

// …

strcpy(buffer, "Hello Server!");

// invio
ret = send(sd, (void*)buffer, len, 0);

if (ret < len) {

     // Gestione errore

}
```

13.6.2 Primitiva `recv` per la ricezione di un messaggio

La primitiva receive permette la ricezione di un messaggio da un socket connesso: copia dal buffer del kernel al buffer del programma (quello che indicheremo nei parametri).

```
ssize_t recv(int sockfd, const void* buf, size_t len, int flags);
```

- **Parametri di ingresso**:
  - `sockfd`: descrittore del socket
  - `buf`: puntatore al buffer in cui salvare il messaggio
  - `len`: dimensione del messaggio in byte
  - `flags`: per ora teniamolo a zero.
- La primitiva restituisce il numero di byte ricevuti, -1 in caso di errore, 0 se il socket remoto si è chiuso (questione che sarà chiarita più avanti).
- **La funzione è bloccante!!** Il programma si ferma finchè non ha letto qualcosa. Cosa si intende con qualcosa? 1 byte (tenerlo a mente all'esame – qualcosa cosa? cit.).

```
int ret, sd, bytes_needed;
char buffer[1024];

// …

bytes_needed = 20;

// ricezione
ret = recv(sd, (void*)buffer, bytes_needed, 0);
// Adesso 0 < ret <= bytes_needed
if (ret < bytes_needed) {

     // Gestione errore

}

ret = recv(sd, (void*)buffer, bytes_needed, MSG_WAITALL);
// Adesso ret == bytes_needed
```

13.6.3 Primitiva `close` per la chiusura del socket

La primitiva close permette la chiusura di un socket: con "chiusura" intendiamo che il socket non potrà più essere utilizzato per inviare o ricevere dati.

```
#include <unistd.h>
int close(int fd);
```

- L'unico parametro di ingresso è il descrittore del socket fd
- La funzione restituisce:
  - 0 se ha successo;
  - -1 in caso di errore.
- La recv lanciata sull'host remoto riceverà 0.

13.7 Pillole sulla gestione degli errori

13.7.1 Variabile `errno`

Abbiamo capito che in caso di errore una qualunque primitiva restituisce come valore -1. In caso di errore la primitiva andrà a settare la variabile *errno*, che può essere utilizzata per capire quale errore si è effettivamente manifestato. La cosa ci permette di stabilire nel codice comportamenti diversi in base all'errore.

```
#include <errno.h>
// …
     ret = bind(sd, (struct sockadrr*)&my_addr, sizeof(my_addr));
     if(ret == -1) {
          if(errno == EADDRINUSE) {  /* Gestione errore */ }
          if(errno == EINVAL) { /* Gestione errore */ }
     }
```

13.7.2 Primitiva `perror` per la stampa dell'errore

La primitiva perror, leggendo la variabile errno, stampa su schermo l'errore in formato leggibile.

```
#include <errno.h>
// …
     ret = bind(sd, (struct sockadrr*)&my_addr, sizeof(my_addr));
     if(ret == -1) {
          perror("Error: ");
          exit(1);
     }
```

13.8 Server concorrenti: gestione simultanea di più richieste

13.8.1 Server iterativo VS Server concorrente

Fino ad ora abbiamo ragionato esclusivamente nell'ottica di server iterativi, cioè server che gestiscono una richiesta alla volta. Distinguiamo server iterativo da server concorrente!

- **Server iterativo**.
  - Gestisce una richiesta alla volta
  - Per ogni richiesta accettata per mezzo della primitiva accept, il processo server la elabora e accoda le richieste che nel frattempo sopraggiungono.
- **Server concorrente**.
  - Gestisce più richieste alla volta
  - Per ogni richiesta accettata il processo server crea un processo che la elabora, che chiameremo processo figlio (figlio del processo server, che chiameremo processo padre).

13.8.2 Primitiva `fork` per la creazione di un processo clone (processo figlio)

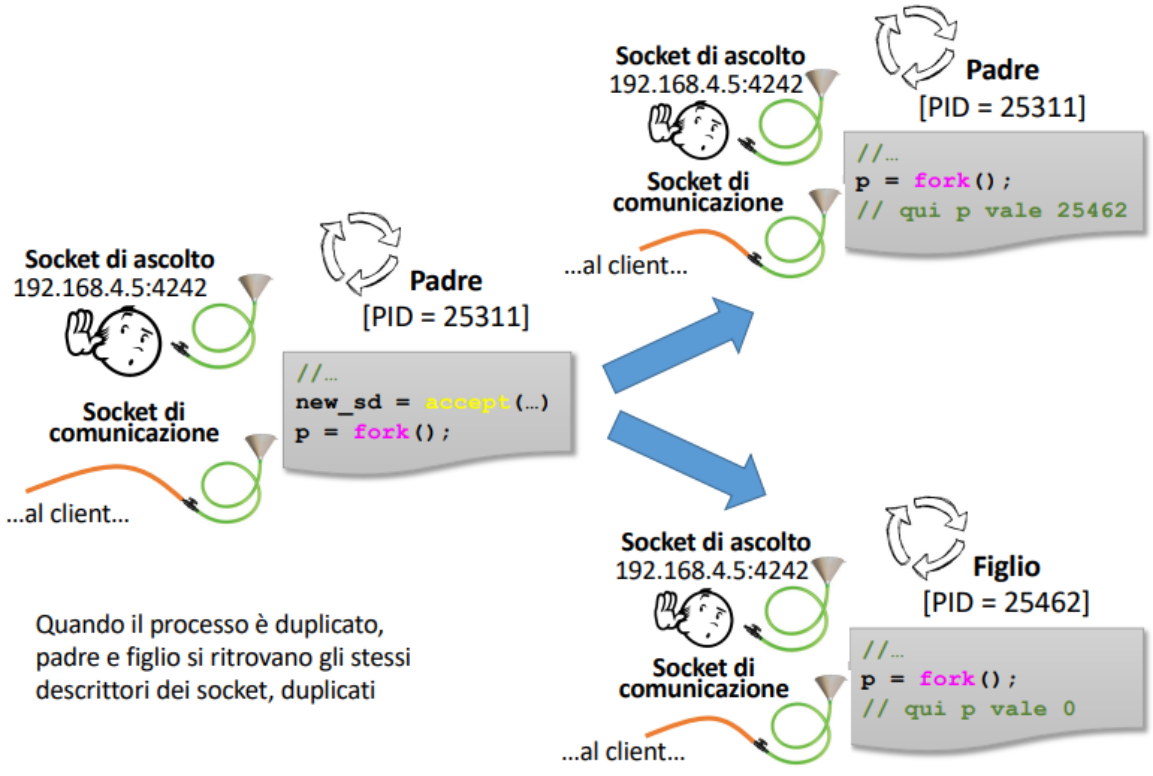
La primitiva fork permette la clonazione di un processo. Abbiamo:

- un processo che invoca la primitiva, detto processo padre;
- il processo nato dall'esecuzione della primitiva, detto processo figlio, che esegue lo stesso identico codice del processo padre.

```
#include <unistd.h>
pid_t fork(void);
```

- Non si hanno parametri di ingresso.
- Viene restituito un intero con segno:
  - -1 in caso di errori;
  - 0 nel processo figlio;
  - il process identifier (PID) nel processo padre.

Occhio ai socket! Il processo figlio è uguale in tutto e per tutto al processo padre, anche nei socket (vedremo più avanti come differenziare i socket tra i due processi).



Si consideri anche un altro aspetto: entrambi i processi presentano lo stesso socket di ascolto e lo stesso socket di comunicazione.

- **A cosa server il socket di comunicazione al processo padre?**
  Il processo padre lancia la fork perché prevede che la comunicazione verrà gestita dal processo figlio! Non ne ha bisogno.
- **A cosa server il socket di ascolto al figlio?**
  Stessa cosa, se si occupa solo della comunicazione non avrà bisogno del socket di ascolto.

Segue che nel codice faremo in modo che:

- il processo padre chiuda il *socket di comunicazione*;
- il processo padre chiuda il *socket di ascolto*.

13.8.3 Snippet di codice con uso della `fork`

```
#include …

int main () {

      int ret, sd, new_sd, len;
      struct sockaddr_in my_addr, cl_addr;

      //...

      pid_t pid;
      sd = socket(AF_INET, SOCK_STREAM, 0);
      /* Creazione indirizzo */
      ret = bind(sd, (struct sockaddr*)&my_addr, sizeof(my_addr));
      ret = listen(sd, 10);
      len = sizeof(cl_addr);
      while(1) {

            new_sd = accept(sd, (struct sockaddr*)&cl_addr, &len);
            pid = fork();
            if (pid == -1) { /* Gestione errore */ }
            if (pid == 0) {

                  /* Sono nel processo figlio */
                  close(sd);
                  /* Gestione richiesta (send, recv, …) */
                  close(new_sd);
                  exit(0);

            }
            // Sono nel processo padre
            close(new_sd);

}
```

13.9 Modello di I/O: socket bloccanti e socket non bloccanti

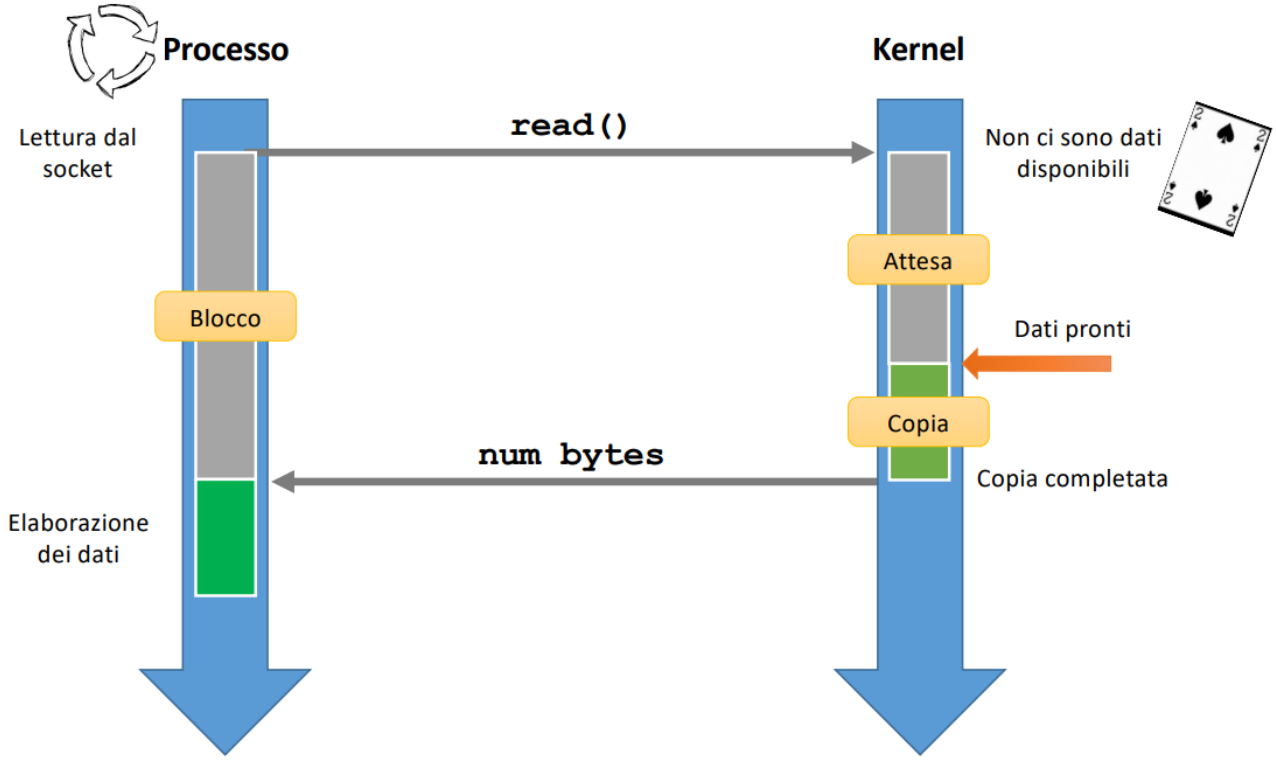
13.9.1 Socket bloccanti: recap

Fino ad ora abbiamo ragionato esclusivamente con socket bloccanti (che è anche il modello di default). Riprendiamo le primitive bloccanti viste fino ad ora:

- `connect()`
  Il processo rimane bloccato fino a quando il socket non viene connesso (o qualora si manifestino errori fatali).
- `accept()`
  Il processo rimane bloccato fino a quando non arriva una richiesta di connessione.

- `send()`
  Il processo rimane bloccato finchè tutto il messaggio “non è stato inviato” (si intende finchè il messaggio non è stato posto nel buffer di sistema). Possibile che non vengano gestiti tutti i byte del messaggio (buffer saturo)
- `recv()`
  Il processo rimane bloccato finchè non ci sono dati disponibili (cosa che succede di default) o finchè tutto il messaggio richiesto non è disponibile (se si pone il flag `MSG_WAITALL`).

Si tenga a mente il comportamento assunto da una generica primitiva `read` bloccante (il comportamento è quello della primitiva `recv`):



Vedremo che il comportamento cambia se decidiamo di definire dei socket non bloccanti.

13.9.2 Definizione di un socket non bloccante

Se siamo interessati a definire un socket non bloccante quello che dobbiamo fare è introdurre il flag SOCK_NONBLOCK nel secondo parametro di ingresso della primitiva socket:

```
socket(AF_INET, SOCK_STREAM|SOCK_NONBLOCK, 0);
```

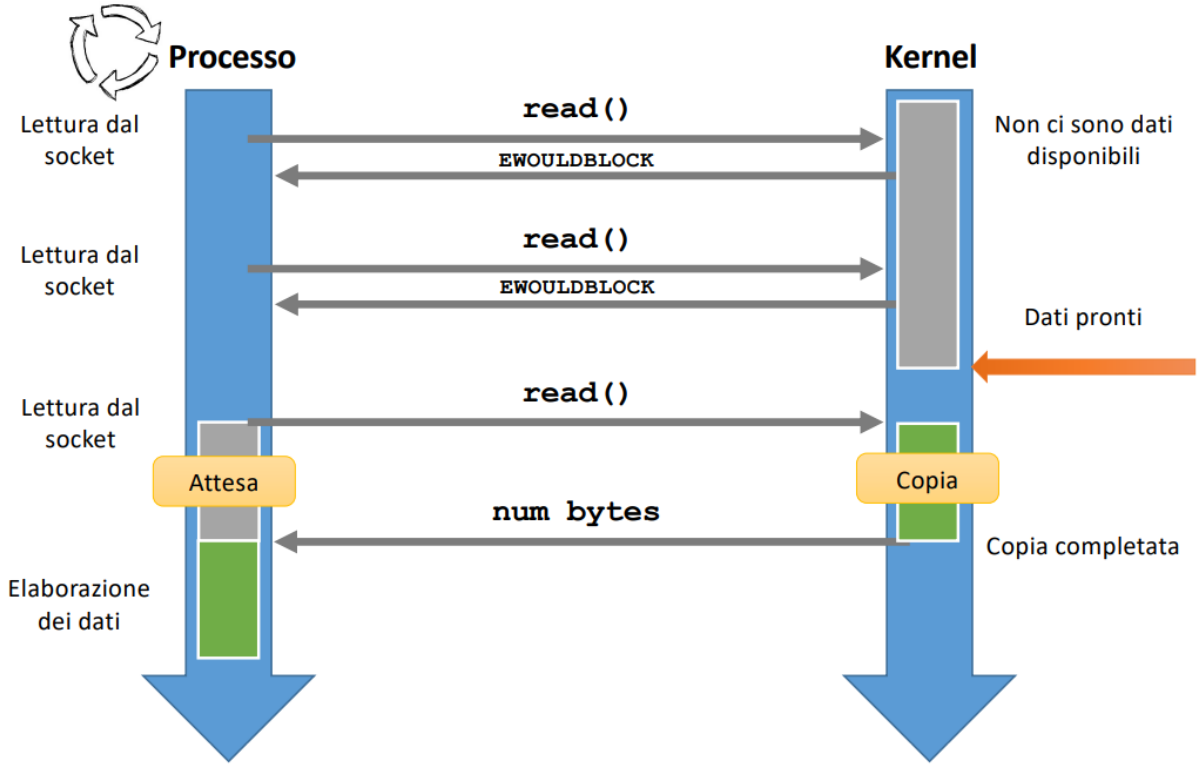
13.9.3 Socket non bloccanti e variabile di errore

Abbiamo capito che qualunque primitiva, in generale, restituisce -1 nel caso in cui si manifestino errori. Rientrano tra i possibili errori i seguenti:

- `connect()`
  La primitiva non bloccante, se non è possibile connettersi, restituisce -1 e imposta errno uguale a `EIMPROGRESS`.
- `accept()`
  La primitiva non bloccante, in assenza di richieste, restituisce -1 e imposta errno uguale a `EWOULDBLOCK`.
- `send()`
  La primitiva non bloccante, se non può inviare tutto il messaggio (buffer pieno), restituisce -1 e imposta errno uguale a `EWOULDBLOCK`
- `recv()`
  La primitiva non bloccante, se non ci sono messaggi, restituisce -1 e imposta errno uguale a `EWOULDBLOCK`.

Per quanto riguarda il comportamento della solita primitiva `read` osserviamo nello schema successivo le differenze rispetto all'approccio bloccante: il processo deve lanciare la primitiva più volte, consapevole che in assenza di messaggi da leggere verrà segnalato errore con la variabile `errno`.

In sostanza: il processo può fare altro prima di lanciare nuovamente la primitiva `read`.



13.10 I/O multiplexing

13.10.1 Problema di base: gestire più socket in contemporanea

Rimaniamo nel contesto del server iterativo.

- Abbiamo un socket di ascolto da cui otteniamo socket di comunicazione dopo aver invocato una o più volte la primitiva accept.
- In un contesto molto realistico è altamente improbabile che io chiuda subito una connessione dopo aver gestito una richiesta con alcune righe di codice: potrei avere l'esigenza di mantenere la connessione e recuperarla più avanti.
- Con gli strumenti visti fino ad ora è abbastanza difficile gestire tanti socket di comunicazione: abbiamo bisogno di un meccanismo di multiplexing.

Le possibili soluzioni ai problemi detti sono:

- controllo di più descrittori/socket allo stesso tempo (poco elegante);
- multiplexing con la primitiva select (si esaminano più socket, il primo che è pronto viene usato).

Si parla quindi di ***socket pronto*** a svolgere una certa operazione.

13.10.2 socket pronto: in quali circostanze?

Cosa significa socket pronto?

- **Operazioni di lettura**.
  Il socket è pronto se:
  - c'è almeno un byte da leggere;
  - il socket è stato chiuso;
  - è un socket in ascolto e ci sono connessioni effettuate (qualcuno ha chiamato la connect);
  - c'è un errore (primitiva `read` restituirà -1).

- **Operazioni di scrittura**.
  Il socket è pronto se:
  - c'è spazio nel buffer per scrivere;
  - c'è un errore (primitiva `write` restituirà -1.

13.10.3 Set di descrittori: tipo e macro per la manipolazione

Introduciamo i set di descrittori, dove:

- Un descrittore consiste in un intero appartenente all'intervallo $[0;\ FD\_SETSIZE]$ (di solito `FD_SETSIZE` è 1024)
- Un insieme di descrittori (variabile di tipo `fd_set`) può essere immaginato come un barattolo all'interno del quale mettiamo tante palline quanti sono i socket da monitorare.

Il set di descrittori può essere alterato per mezzo delle seguenti macro

```
// Aggiunta di un descrittore "fd" all'insieme di descrittori "set"
void FD_SET(int fd, fd_set* set);

// Controllare se un descrittore "fd" è nell'insieme di descrittori "set"
int FD_ISSET(int fd, fd_set* set);

// Rimozione di un descrittore "fd" dall'insieme di descrittori "set"
void FD_CLR(int fd, fd_set* set);

// Svuotamento dell'insieme di descrittori "set"
void FD_ZERO(fd_set* set);
```

13.10.4 Primitiva `select`

La primitiva select permette il controllo di più socket, rilevando quelli pronti secondo quanto detto poco indietro.

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int select(int nfds, fd_set* readfds, fd_set* writefds,
           fd_set* exceptfds, struct timeval* timeout);
```

- **Parametri di ingresso**:
  - `nfds`: numero del descrittore più alto tra quelli da controllare, incrementato +1.
  - set di descrittori `readfds` e `writefds`: due set distinti, uno per i descrittori da controllare per la lettura e uno per i descrittori da controllare per la scrittura.
  - exceptfds: eccezioni, non ci interessa
  - timeout: intervallo di timeout.
- **È una primitiva bloccante!!!** Si blocca fino a quando uno dei descrittori controllati non è pronto, oppure finchè non scade il timeout.
- In caso di errore restituisce -1, altrimenti restituisce il numero di descrittori pronti. Nel caso in cui scada il timeout restituisce 0 (e non errore!!!).
- `timeout` consiste in una particolare struttura dove possiamo definire il timeout indicando secondi e/o microsecondi.

```
#include <sys/socket.h>
#include <netinet/in.h>

struct timeval {
    long tv_sec; /* seconds */
```

```
    long tv_usec; /* microseconds */
};
```

Esempi:

```
timeout = NULL // attesa indefinita, fino a quando un
descrittore è pronto
timeout = { 10; 5; } // attesa massima di 10 secondi e 5
microsecondi
timeout = { 0; 0; } // attesa nulla, controlla i descrittori ed
esce immediatamente (polling)
```

13.11 Primitive per i socket UDP

13.11.1 Introduzione

Fino ad ora abbiamo ragionato di socket dando per scontato l'adozione del protocollo TCP. Facciamo uno step ulteriore introducendo l'uso dei socket col protocollo UDP. Ricordiamo la differenza fondamentale:

- UDP è connectionless, cioè non si eseguono operazioni preliminari per instaurare una connessione
- UDP è più veloce di TCP in quanto non prevede recupero dei pacchetti, riordino dei pacchetti fuori sequenza, o controllo di flusso.
- UDP non è affidabile come TCP: la maggiore velocità va a detrimento dell'affidabilità, in quanto i pacchetti possono andare persi e/o corrompersi.

Si pensi alle logiche differenze di codice tra UDP e TCP

- non utilizzeremo le primitive introdotte per svolgere le operazioni preliminari dei socket in TCP;
- viene meno la distinzione tra socket di ascolto e socket di comunicazione.

13.11.2 Primitiva `sendto` per l'invio di un messaggio

La primitiva sendto permette l'invio di un messaggio attraverso un socket all'indirizzo specificato

```
ssize_t sendto(int sockfd, const void* buf, size_t len, int flags,
               const struct sockaddr* dest_addr, socklen_t addrlen);
```

- Parametri di ingresso:
  - `sockfd`: descrittore del socket
  - `buf`: puntatore al buffer contenente il messaggio da inviare;
  - `len`: dimensione in byte del messaggio;
  - `flags`: non ci interessa, poniamo 0;
  - `dest_addr`: puntatore alla struttura (già vista) contenente l'indirizzo del destinatario;
  - `addrlen`: lunghezza di dest_addr
- Viene restituito il numero di byte inviati, o -1 in caso di errore. Si mantiene l'ipotesi semplificativa che un'unica send riesce a trasmettere tutto il messaggio.
  - E se non riesco a trasmettere tutto? Invio parti del messaggio distintamente, ciclando sulla send.
  - Cosa vuol dire inviare qualcosa su un socket?
    - Con protocollo TCP un messaggio è inviato quando riversato nel buffer del kernel. Se si verifica un errore? Chissene importa.
- **Primitiva bloccante!!** Il programma si ferma finchè non ha scritto tutto il messaggio.

13.11.3 Primitiva `recvfrom` per la ricezione di un messaggio

La primitiva recvfrom permette la ricezione di un messaggio attraverso un socketò.

```
ssize_t recvfrom(int sockfd, const void* buf, size_t len, int flags,
                 struct sockaddr* src_addr, socklen_t addrlen);
```

- Parametri di ingresso:
  - `sockfd`: descrittore del socket
  - `buf`: puntatore al buffer dove verrà memorizzato il messaggio ricevuto;
  - `len`: dimensione in byte del messaggio;
  - `flags`: non ci interessa, poniamo 0;
  - `dest_addr`: puntatore alla struttura (già vista) contenente l'indirizzo del mittente);
  - `addrlen`: lunghezza di src_addr
- Viene restituito il numero di byte inviati, o -1 in caso di errore. Si mantiene l'ipotesi semplificativa che un'unica send riesce a trasmettere tutto il messaggio.
  - E se non riesco a trasmettere tutto? Invio parti del messaggio distintamente, ciclando sulla send.
- **Primitiva bloccante!!** Il programma si ferma finchè non ha letto qualcosa (un byte!).

13.11.4 Snippet di codice del server

Consideriamo il seguente snippet di codice relativo a un generico server.

```
int main () {

     int ret, sd, len;
     char buf[BUFLEN];
     struct sockaddr_in my_addr, cl_addr;
     int addrlen = sizeof(cl_addr);

     /* Creazione socket UDP */
     sd = socket(AF_INET, SOCK_DGRAM, 0);
     /* Creazione indirizzo */
     memset(&my_addr, 0, sizeof(my_addr); // Pulizia
     my_addr.sin_family = AF_INET ;
     my_addr.sin_port = htons(4242);
     my_addr.sin_addr.s_addr = INADDR_ANY;
     ret = bind(sd, (struct sockaddr*)&my_addr, sizeof(my_addr));

     while(1) {
          len = recvfrom(sd, buf, BUFLEN, 0, (struct sockaddr*)&cl_addr,
          &addrlen);
          //fai cose ...
     }
     // …

}
```

Osserviamo che:

- Nell'inizializzazione ci limitiamo solo alla creazione del socket con l'apposita primitiva, e alla definizione della struttura contenente l'indirizzo del server. Si osservi che abbiamo creato il socket ponendo come flag `SOCK_DGRAM` invece di `SOCK_STREAM`.
- Si cicla con la `recvfrom`, e ogni volta che si riceve qualcosa si fa qualcosa.

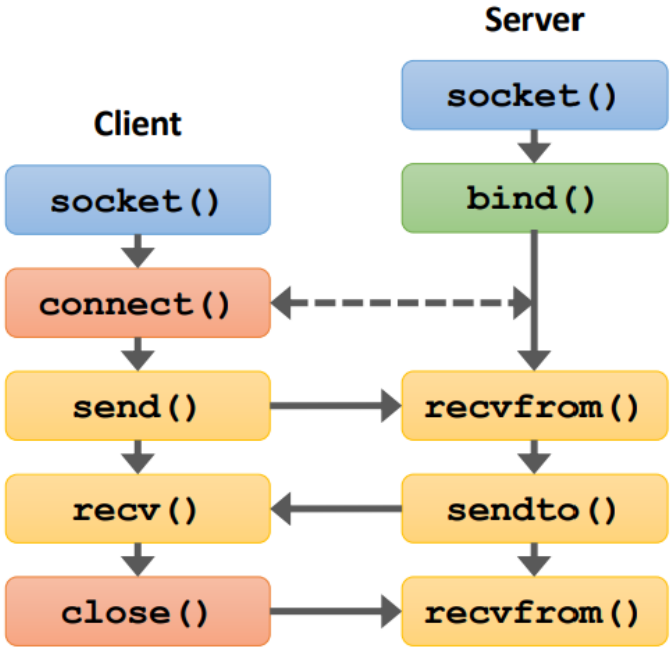
13.11.5 Snippet di codice del client

Abbiamo struttura abbastanza simile anche per il client (unica differenza è l'esecuzione della `sendto` invece della `recvfrom`).

```
int main () {

      int ret, sd, len;
      char buf[BUFLEN];
      struct sockaddr_in sv_addr; // Struttura per il server

      /* Creazione socket */
      sd = socket(AF_INET, SOCK_DGRAM, 0);

      /* Creazione indirizzo del server */
      memset(&sv_addr, 0, sizeof(sv_addr); // Pulizia
      sv_addr.sin_family = AF_INET ;
      sv_addr.sin_port = htons(4242);
      inet_pton(AF_INET, "192.168.4.5", &sv_addr.sin_addr);

      while(1) {

            len = sendto(sd, buf, BUFLEN, 0, (struct sockaddr*)&sv_addr,
            sizeof(sv_addr));
            // fai cose...

      }

      // …

}
```

13.11.6 Socket UDP "connesso"

Il client, per velocizzare le chiamate di invio e ricezione, potrebbe utilizzare la primitiva `connect` qualora l'indirizzo del server a cui il client si rivolge sia sempre lo stesso.



L'associazione tra un indirizzo remoto e il socket non implica assolutamente l'introduzione di una qualche forma di connessione:

- l'esecuzione della `connect` avviene nel client **e il server non ne è a conoscenza**;
- non si introduce alcun tipo di controllo;
- il protocollo di trasporto eseguito è sempre il protocollo UDP!

Dopo aver eseguito la `connect` le strade sono due:

- eseguire la `sendto` (o la `recvfrom`) ponendo NULL il parametro `dst_addr` (o `src_addr`);
- eseguire la `send` (o la `recv`)

13.12 Protocolli *text and binary*

13.12.1 Differenze: vantaggi e svantaggi

Distinguiamo due tipologie di protocollo:

- ***Text protocols***.
  Inviano messaggi in formato testo, solitamente con codifica ASCII. Esempio: protocollo HTTP.
  - **Difetti**.
    - Overhead di codifica e decodifica.
    - Pacchetti "sniffabilissimi": stiamo passando un messaggio in chiaro.
  - **Vantaggi**.
    - Leggibilità e semplicità.
- ***Binary protocols***.
  Inviano strutture dati!
  - **Difetti**.
    - Necessaria conoscenza della struttura, dei dati che si sta manipolando.
    - Attuare la serializzazione (si prende una struttura con una certa altezza e larghezza, la si fa diventare una sequenza di byte).
  - **Vantaggi**.
    - Minore occupazione di memoria rispetto a una trasmissione in ASCII.
      Si consideri il seguente esempio:

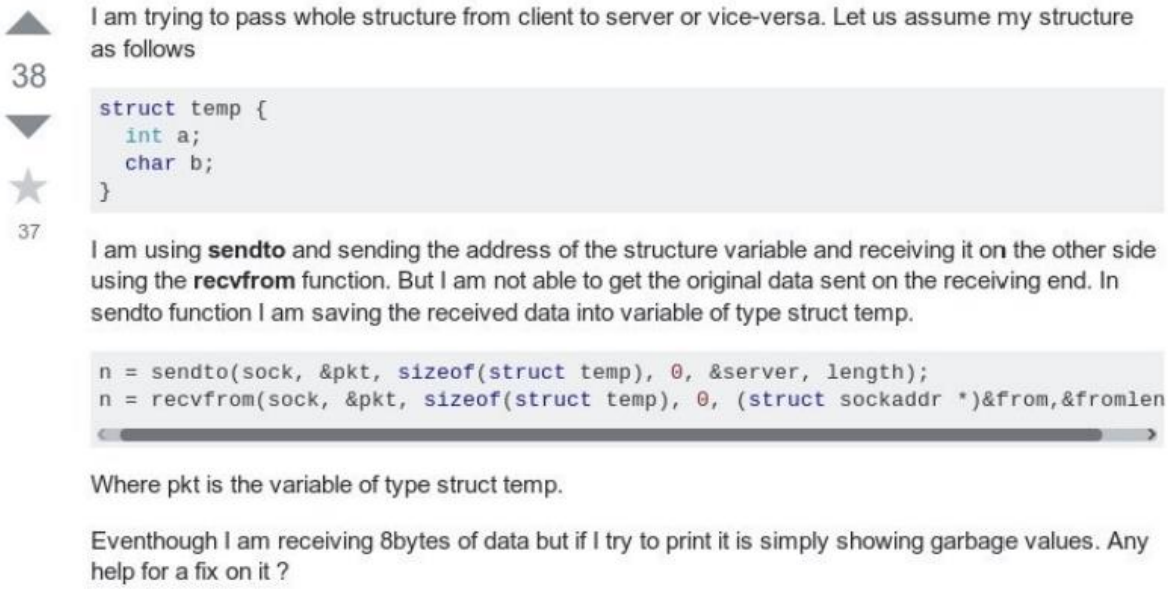
binary 1234	byte 0	byte 1	byte 2	byte 3
	00000000	00000000	00000100	11010010

text "1234"	byte 0	byte 1	byte 2	byte 3
	00110001	00110010	00110011	00110100

    - Molto più difficile un attacco su dati di tipo binary.

13.12.2 Occhio ai binary protocols: serializzazione delle strutture dati

La cosa che non deve passare per la testa di nessuno è l'idea che si possa inviare una struttura dati per mezzo della `send` indicando direttamente l'indirizzo della struttura dati e la dimensione della stessa: prima dell'invio è necessario attuare una serializzazione.



38

37

I am trying to pass whole structure from client to server or vice-versa. Let us assume my structure as follows

```
struct temp {
  int a;
  char b;
}
```

I am using **sendto** and sending the address of the structure variable and receiving it on the other side using the **recvfrom** function. But I am not able to get the original data sent on the receiving end. In sendto function I am saving the received data into variable of type struct temp.

```
n = sendto(sock, &pkt, sizeof(struct temp), 0, &server, length);
n = recvfrom(sock, &pkt, sizeof(struct temp), 0, (struct sockaddr *)&from,&fromlen
```

Where pkt is the variable of type struct temp.

Eventhough I am receiving 8bytes of data but if I try to print it is simply showing garbage values. Any help for a fix on it ?

Don't ever write a whole struct in a binary way, not to a file, not to a socket.

Always write each field separately, and read them the same way.

You need to have functions like

```
unsigned char * serialize_int(unsigned char *buffer, int value)
{
  /* Write big-endian int value into buffer; assumes 32-bit int and 8-bit char. */
  buffer[0] = value >> 24;
  buffer[1] = value >> 16;
  buffer[2] = value >> 8;
  buffer[3] = value;
  return buffer + 4;
}

unsigned char * serialize_char(unsigned char *buffer, char value)
{
  buffer[0] = value;
  return buffer + 1;
}

unsigned char * serialize_temp(unsigned char *buffer, struct temp *value)
{
  buffer = serialize_int(buffer, value->a);
  buffer = serialize_char(buffer, value->b);
  return buffer;
}
```

Altrimenti usare la htonl()

13.12.3 *text protocols*: gestire passaggio da struttura a stringa e viceversa

Un text protocol prevede la rappresentazione di informazioni per mezzo di stringhe in codifica ASCII. Si gestisce il passaggio da struttura dati a stringa e viceversa per mezzo delle seguenti righe di codice, abbastanza auto-esplicative.

Passaggio da struttura a stringa	Passaggio da stringa a struttura

`char buffer[1024];`
`// Dichiarazione`
`struct temp{`
`  int a;`
`  char b;`
`};`
`// Istanziazione`
`struct temp t;`
`// Conversione a stringa`
`sprintf(buffer, "%d %c", t.a, t.b);`
	`...`
	`// Istanziazione`
	`struct temp t;`
	
	`// Ricezione ...`
	
	`// Parsing e memorizzazione nei campi`
	`sscanf(buffer, "%d %c",&t.a, &t.b);`

13.12.4 binary protocols

Un binary protocol prevede la trasmissione delle strutture dati come sequenze di bit.

- I messaggi hanno una struttura fissata, con campi che rappresentano le informazioni da scambiare.
- Ciascun campo ha una lunghezza e un tipo che possa essere trasferito.
- Se un campo prevede lunghezza variabile si definisce comunque una lunghezza massima dei messaggi.

Nell'invio si gestisce ogni campo singolarmente!

Col seguente codice si gestisce l'invio:

```
…
struct temp{
     uint32_t a;
     uint8_t b;
};
struct temp t;
…
// Convertire in network order prima dell'invio
t.a = htonl(t.a);
// Spedire i campi sul socket 'new_sd'
ret = send(new_sd, (void*)&t.a, sizeof(uint32_t), 0);
…
ret = send(new_sd, (void*)&t.b, sizeof(uint8_t), 0);
…
```

Col seguente, invece, si gestisce la ricezione:

```
…
struct temp t;
…
ret = recv(new_sd, (void *)&t.a, sizeof(uint32_t), 0);
if (ret < sizeof(uint32_t)) {
     // Gestione errore
}
// Convertire in host order il campo 'a'
t.a = ntohl(t.a);
ret = recv(new_sd, (void *)&t.b, sizeof(uint8_t), 0);
if (ret < sizeof(uint8_t)) {
     // Gestione errore
}
…
```

14 LABORATORIO: FIREWALL

14.1 Introduzione

Il firewall è un meccanismo di protezione che un host della rete può utilizzare. Si hanno varie tipologie di firewall, atte a vietare particolari accessi o permetterne altri: quella che noi vedremo è il cosiddetto firewall a *filtraggio di pacchetto*. Tra le cose possiamo pensare a:

- consentire che la mia macchina possa raggiungere tutti i siti, tranne quelli ospitati su web server di una particolare sotto rete.
- vietare il traffico HTTP in uscita da una certa sottorete.
- consentire il traffico in uscita solo in UDP;
- …

Potete pensarlo *come un buttafuori, il gorilla che applica le regole* (cit.): si verifica che ogni pacchetto rispetti le regole indicate, lo scartiamo in caso contrario. Oggi c'è necessità di proteggere reti e computer da accessi indesiderati e malware: il firewall rientra tra gli strumenti adottati a tutela della sicurezza.

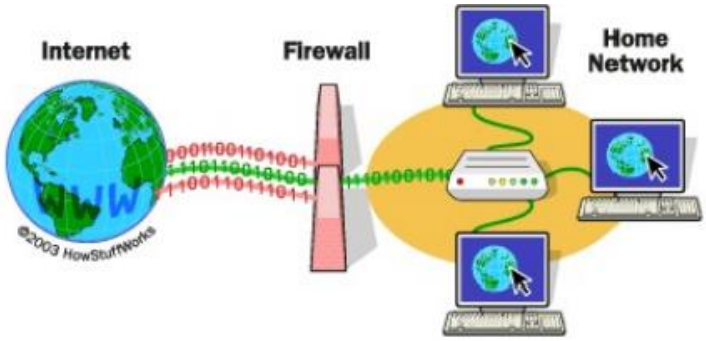
Quindi, in modo più formale:

> *Sistema hardware o software che controlla le connessioni in ingresso e uscita, e applica regole.*

14.2 Tipologie di firewall

I firewall sono classificati in base a dove operano:

- possono operare a **livello rete** (network firewall) o nella **macchina** (host-based firewall)



- per quanto riguarda quelli che operano nella macchina abbiamo
  - **Network layer (Firewall a filtraggio di pacchetto, *packet filter*)**
    Opera a livello trasporto, analizzando gli header IP, TCP e UDP.
  - **Application layer**
    Operano a livello applicazione, facendo deep packet inspection. E' un firewall decisamente più potente ed efficace (ovviamente con maggiore risorse computazionali): più si sale nella pila, più siamo consapevoli del contesto (e quindi abbiamo più criteri su cui basare l'accettazione o l'esclusione di pacchetti).

Level	Shallow Packet Inspection	Medium Packet Inspection	Deep Packet Inspection	ISO/OSI
7				Application
6				Presentation
5				Session
4				Tranport
3				Network
2				Data Link
1				Physical

14.3 Firewall a filtraggio di pacchetto (packet filter)

14.3.1 Tipologie

I firewall a filtraggio di pacchetto si dividono in:

- ***stateless***
  Analisi del pacchetto in base a campi statici come indirizzo di sorgente o destinazione.
- ***statefull***
  Tiene traccia delle connessioni TCP e degli scambi UDP in corso, e discrimina le connessioni legittime da quelle sospette (visibilità più ampia, non si guarda solo pacchetto per pacchetto ma ci si basa su un contesto più ampio). È più efficace, ma ovviamente più complesso e pesante rispetto al filtraggio stateless.

14.3.2 Tabella di regole

14.3.2.1 Caratteristiche

Un firewall è caratterizzato da una *tabella di regole*. Ogni regola contiene:

- **Caratteristiche del pacchetto** (*criteria*)
  - IP sorgente
  - Porta sorgente
  - IP destinatario
  - Porta destinatario
- **Azione da intraprendere** (*target*)
  - Accettazione del pacchetto (ACCEPT)
  - Pacchetto scartato (DROP)

Prendiamo la seguente tabella, con due regole:

Indice	IP sorgente	Porta sorgente	IP destinatario	Porta dest.	Azione
1	131.114.0.0/16		131.114.54.4	80	SCARTA
2	0.0.0.0	23	112.143.2.2		ACCETTA

- **Prima regola**. Scartare tutti i pacchetti provenienti dalla sottorete 131.114.0.0/16 destinati alla porta 80 del destinatario 131.114.54.4
- **Seconda regola**. Accetta tutti i pacchetti diretti al destinatario 112.143.2.2, provenienti dalla porta 23 di qualsiasi sorgente.

Ogni regola è identificata da un'indice. Quando si considera un pacchetto si scorrono tutte le regole dall'indice più piccolo a quello più grande: non appena si individua una regola applicabile la si applica e si esce dal ciclo (ignorando regole successive).

14.3.2.2 default rule

Nel caso in cui non si applichi nessuna regola presente nella tabella il sistema va ad applicare la default rule. Se la default rule prevede lo scarto del pacchetto si parla di *firewall inclusivo*, altrimenti di *firewall esclusivo*.

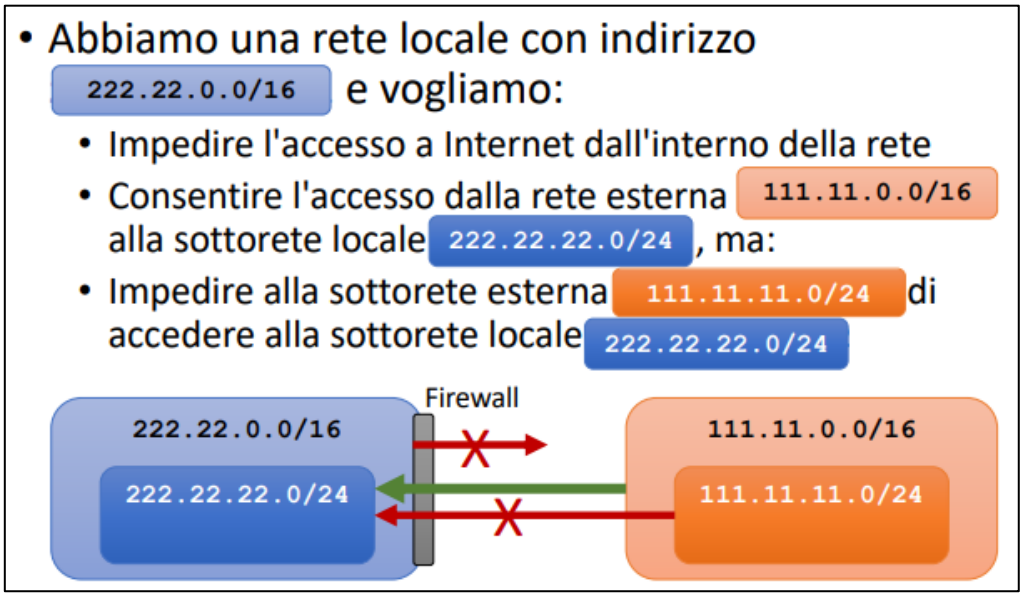
***Applicare la porta NOT**: inclusivo ed esclusivo sono termini che suonano strani da un punto di vista logico.*

- Il firewall inclusivo è sicuro, ma scomodo. Non si può accedere a nulla se non si definiscono regole. *Definisco cosa è giusto fare*.
- Il firewall esclusivo è comodo, ma insicuro. Devo prevedere e inserire manualmente tutte le regole che ritengo utili. *Definisco cosa è giusto bloccare*.

La regola di default è settata col comando *iptables* (opzione -P).

14.3.2.3 Importanza dell'ordine delle regole

Consideriamo il seguente esempio, che renderà chiaro il fatto che le regole non possono essere disposte casualmente.



In che ordine disponiamo le regole?

Indice	IP sorgente	Porta sorgente	IP destinatario	Porta dest.	Azione
1	111.11.0.0/16		222.22.22.0/24		ACCETTA
2	111.11.11.0/24		222.22.0.0/24		BLOCCA
3	0.0.0.0		0.0.0.0		BLOCCA

La seguente tabella presenta un ordine sbagliato. Se arriva un pacchetto diretto alla sottorete 222.22.22.0/24, proveniente dalla sottorete 111.11.11.0/24 (che è interna a 111.11.0.0/16), il pacchetto viene lasciato passare nonostante le regole da noi definite precedentemente (i criteria del pacchetto sono già rispettati nella prima regola che prevede accettazione, quindi si esegue il target e si ignorano le regole successive).

Risolviamo il problema invertendo prima e seconda regola

Indice	IP sorgente	Porta sorgente	IP destinatario	Porta dest.	Azione
1	111.11.11.0/24		222.22.0.0/24		BLOCCA
2	111.11.0.0/16		222.22.22.0/24		ACCETTA
3	0.0.0.0		0.0.0.0		BLOCCA

Adesso la prima regola applicata è quella che prevede blocco per pacchetti provenienti da sottoreti 111.11.11.0/24 e destinati a sottoreti 222.22.0.0/24.

14.3.3 netfilter e iptables

14.3.3.1 Introduzione

netfilter è il componente del kernel di Linux che offre le funzionalità di:

- stateless/statefull packet filtering;
- NA[P]T (che vedremo più avanti).

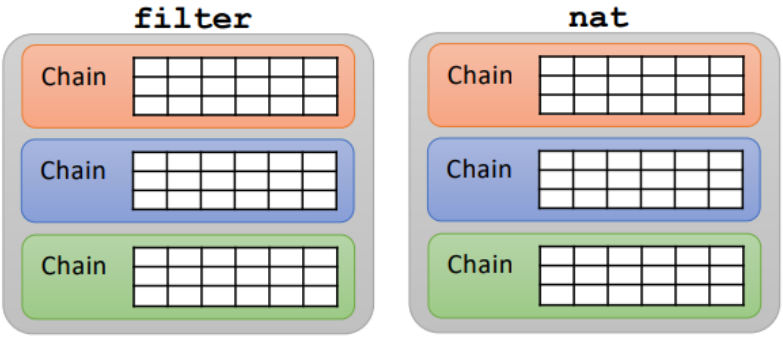
iptables è un comando che permette la manipolazione della tabella di regole. In realtà il comando permette la manipolazione di due tipologie di tabelle:

- *filter*, la tabella di regole descritta precedentemente;
- *nat*, tabella che permette il *Network Address Translation* (che affronteremo più avanti).

14.3.3.2 Organizzazione delle tabelle

Ogni tabella è organizzata in catene (chain): ogni catena raggruppa regole dipendentemente da cosa il pacchetto sta facendo (quindi regole che si applicano a una categoria di pacchetti). Domanda che sorge spontanea: quali azioni può compiere un pacchetto?

- **INPUT**
  Pacchetti in ingresso destinati ai processi locali
- **OUTPUT**
  Pacchetti in uscita dai processi locali
- **FORWARD**
  Pacchetti in transito, cioè da inoltrare ad altri host.



14.3.3.3 Comando iptables: visualizzazione e manipolazione della tabella filter

Il comando iptables, da eseguire con privilegi di root, permette la manipolazione delle tabelle filter e nat. Limitiamoci per il momento a introdurre le azioni possibili per la tabella di regole.

- **Visualizzare la tabella di regole**.
  Poniamo quanto segue
  ```
  iptables [-t table] -L [chain]
  ```
  - Se la tabella non è specificata si intende la tabella filter (la tabella di regola)
  - Se la catena non è specificata vengono elencate tutte le catene

- **Aggiunta di regole**.
  Poniamo quanto segue
  ```
  // Aggiunta in fondo alla catena
  iptables [-t table] -A chain rule-specification
  // Aggiunta nella posizione indicata (1 se omesso)
  iptables [-t table] -I chain [num] rule-specification
  ```

  Cosa intendiamo con `rule-specification`? Una stringa con cui viene indicata la regola, secondo una certa sintassi. Ci interessano i seguenti parametri:

`rule-specification` è una stringa, dove specificare:

• `-p <protocollo>`	protocollo (TCP, UDP, ICMP, …)
• `-s <address>`	indirizzo IP sorgente
• `-d <address>`	indirizzo IP destinazione
• `--sport <port>`	porta sorgente
• `--dport <port>`	porta destinazione
• `-i <interface>`	interfaccia di ingresso
• `-o <interface>`	interfaccia di uscita
• `-j <target>`	azione (DROP/ACCEPT)

- **Rimozione di regole**.
  Poniamo quanto segue

```
// Rimozione di una particolare regola (si indica la regola o la posizione)
iptables [-t table] -D chain rule-specification
iptables [-t table] -D chain num

// Rimozione di tutte le regole da una o più catene (se non si indica si ha
eliminazione completa)
iptables [-t table] -F [chain]

// Alterazione della regola di default
iptables [-t table] -P target
```

14.3.3.4 Esempi di righe di comando con interpretazione

```
# iptables -A OUTPUT -p tcp -d 10.0.5.4 --dport 80 -j DROP
```

Aggiungi in fondo alla catena OUTPUT della tabella filter una regola che scarti tutti i pacchetti TCP destinati alla porta 80 (HTTP) dell'host 10.0.5.4

```
# iptables -A INPUT -p udp -s 121.0.0.0/16 -j ACCEPT
```

Aggiungi in fondo alla catena INPUT della tabella filter una regola che lasci passare Tutti I pacchetti UDP provenienti dalla sottorete 121.0.0.0/16

```
# iptables -A INPUT -p icmp -i eth0 -j DROP
```

Aggiungi in fondo alla catena INPUT della tabella filter una regola che scarti tutti i pacchetti ICMP provenienti dall'interfaccia d'ingresso eth0

Appunto personale. Se si deve interpretare un comando già scritto ci si muove nel seguente modo:

1. lettura della tabella e della catena;
2. lettura del target (DROP o ACCEPT?);
3. lettura della parte rimanente dove si indicano le caratteristiche della regola.

"Il comando stabilisce l'aggiunta di una regola nella catena X della tabella Y che prevede l'accettazione (lo scarto) di un pacchetto che…"

14.3.3.5 Salvataggio e caricamento delle regole

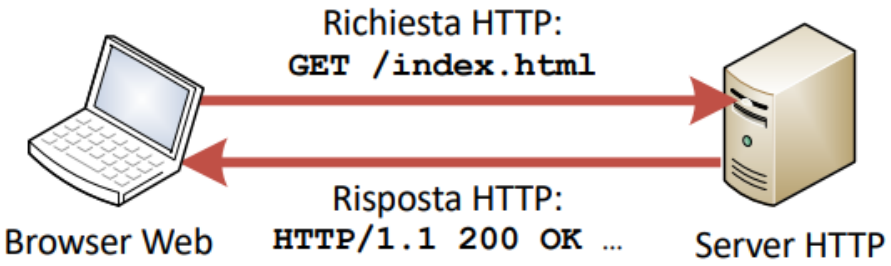
Le regole non sono memorizzate permanentemente, ergo è necessario reimpostarle all'avvio. Salviamo e carichiamo una particolare configurazione di regole utilizzando i seguenti comandi (anche qua sono necessari privilegi di root)

```
iptables-save > file
iptables-restore < file
```

15 LABORATORIO: SERVER WEB APACHE

15.1 Richiamo veloce al protocollo HTTP

Abbiamo già introdotto col prof. Anastasi il protocollo HTTP (Hypertext Transfer Protocol). Esso consiste in un protocollo di livello applicazione che permette di richiedere oggetti sul Web: si parla di file HTML, ma anche di file multimediali (si pensi a immagini, video, …)



Si tratta di un protocollo stateless: il server non tiene traccia delle precedenti connessioni. Ogni scambio richiesta/risposta è indipendente dai precedenti.

15.2 Apache HTTP Server

15.2.1 Informazioni iniziali

Apache permette di trasformare il nostro dispositivo in un server web.

- Su Debian 8 (la distribuzione utilizzata nel corso) abbiamo la versione 2.4.
- È implementato tramite il programma `apache2`
  (in altre distribuzioni il programma potrebbe essere `httpd`)
- Quando è in esecuzione si può aprire un browser e accedere al sito http://localhost/
- La pagina mostrata al sito detto prima consiste nel file HTML
  `/var/www/html/index.html`.
- Apache può accettare e servire più richieste contemporaneamente.
- Si mantiene come ipotesi semplificativa che il server web ospita un solo sito web.
- La directory principale è `/etc/apache2`.

15.2.2 Comandi per invocare Apache

Esistono due comandi per interagire col web server (per entrambi sono necessari i privilegi di root):

`apache2ctl <comando>`
Dove *comando* consiste in uno dei seguenti valori:

- start
- stop
- restart
- status
- configtest

`service apache2 <comando>`
Dove *comando* consiste in uno dei seguenti valori:

- start
- stop
- restart
- reload

15.2.3 File di configurazione

15.2.3.1 Direttive e direttive contenitore

Abbiamo già detto che la directory principale consiste in `/etc/apache2/`. Una qualunque configurazione si fa per mezzo di ***direttive***, eventualmente raggruppate in ***direttive contenitore***.

```
# Esempio di commento
Direttiva1 valore
Direttiva2 valore

# Inizio contenitore
<Contenitore valore>
        Direttiva3 valore
        Direttiva4 valore
</Contenitore>
# Fine contenitore
```

15.2.3.2 Parti di configurazione

Il file di configurazione principale consiste in `/etc/apache2/apache2.conf`

- Su Debian abbiamo Apache che utilizza un sistema modulare. Il file di configurazione recupera le varie parti di configurazione da altri file, utilizzando la direttiva *include*.
- I file inclusi con la direttiva appena citata sono convenzionalmente posti nella cartella `/etc/apache2/conf-available`.
- I file devono essere abilitati utilizzando il comando [*apache 2 enable configuration*]
  `# a2enconf <nome_file>`
  La conseguenza è la creazione di un soft link nella directory
  `/etc/apache2/conf-enabled`
  Tutte le configurazioni abilitate sono incluse all'avvio.
- Un file si disabilita con
  `# a2disconf <nome_file>`

Attenzione: ogni volta che si "razzola" nel file di configurazione di Apache è necessario riavviare il server per rendere effettive le modifiche effettuate.

15.2.3.3 Moduli

Le parti di configurazione che forniscono funzionalità complesse sono i moduli, e si trovano in /etc/apache2/mods-available

- Si abilitano con
  `a2enmod <nome_file>`
- Si disabilitano con
  `a2dismod <nome_file>`

Per ricaricare la configurazione è necessario riavviare il server.

15.2.4 Direttive globali

Si parla di direttive globali in quanto hanno effetto su tutto il server.

15.2.4.1 Direttiva Listen

Con la direttiva Listen si stabiliscono le porte che il processo di Apache deve ascoltare. Possibile indicare più parti, lanciando più volte la direttiva.

```
Listen 80
Listen 8080
```

Attenzione: direttiva obbligatoria, il server non parte se la direttiva non è presente. Nel caso di Debian la direttiva è presente in `/etch/apach2/ports.conf`, che è incluso automaticamente da `apache2.conf`

15.2.4.2 Direttiva ServerRoot

La direttiva ServerRoot specifica la directory principale dei file di configurazione di Apache. I path relativi specificati nelle altre direttive sono risolti partendo da questa directory

```
ServerRoot /etc/apache2
```

All'avvio del servizio dal comando apache2ctl la direttiva ServerRoot è configurata automaticamente (aprire apache2.conf per modificare il path – possibile domanda d'esame).

15.2.4.3 Direttiva KeepAlive e KeepAliveTimeout

La direttiva KeepAlive specifica se offrire o meno le connessioni persistenti tipiche di HTTP 1.1. (Troveremo on quasi sempre).

```
KeepAlive on
```

La direttiva KeepAliveTimeout, invece, specifica quanti secondi attendere la successiva richiesta dal client su una stessa connessione, prima di chiuderla.

```
KeepAliveTimeout 5
```

Il valore va scelto adeguatamente (si giustifica, ad esempio sperimentando):

- se si sceglie un valore piccolo si rischia di chiudere e aprire connessioni inutilmente, contribuendo a un aumento dell'overhead;
- se si sceglie un valore grande si rischia di lasciare il server in attesa inutilmente.

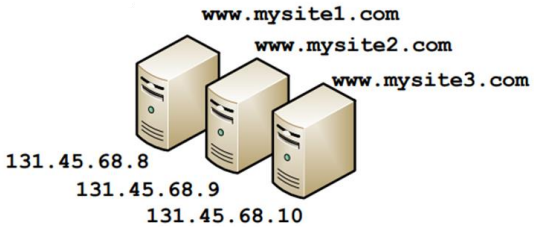
15.2.4.4 Direttiva ErrorLog

La direttiva ErrorLog permette di indicare un file dove saranno memorizzati i log, precisamente i messaggi di errori che si manifestano nel tempo.

```
ErrorLog /var/log/apach2/error.log
```

15.2.5 Virtual Host

Apache nella sua configurazione di base gestisce un sito alla volta. Non è una soluzione adatta: non scala! Sarebbe assurdo pensare di avere una macchina per ogni sito web: ciò richiederebbe ingenti risorse (si pensi anche all'occupazione fisica, tanti computer quanti i siti web esistenti).



Apache 2.0 ha introdotto la possibilità di configurare più siti sullo stesso server Web, sulla stessa macchina che ha un solo indirizzo IP: ogni sito web, in questo contesto, è denominato *virtual host*.

I virtual host (siti) sono posti in una directory dedicata: `/etc/apache2/sites-available`

- **Comandi per abilitare/disabilitare i siti disponibili**.
  I siti posti nella cartella sono abilitati o disabilitati ricorrendo ai seguenti comandi (anche questi comandi richiedono privilegi root)
  `a2ensite <nome_file>`
  `a2dissite <nome_file>`
  Per ogni sito abilitato si crea un soft link nella cartella `sites-enabled`. Come prima è necessario riavviare il server per ricaricare la configurazione.
- Nel caso più semplice Apache ha un default Virtual Host abilitato in `/etc(apache2/sites-available/000-default.conf`
- All'interno dei files della cartella è presenta la direttiva VirtualHost, che è un esempio di direttiva contenitore.

15.3 Multi-Processing Module

15.3.1 Introduzione

Moduli che permettono al server di gestire le sue risorse minimizzando i tempi di attesa (uso efficiente delle risorse computazionali disponibili). Ricordiamoci quanto già detto: Apache accetta e serve più richieste contemporaneamente.

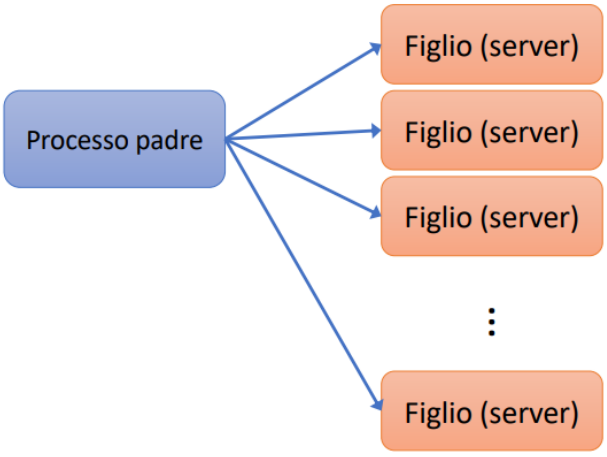
- Gestione dei socket.
- Gestione del binding delle porte.
- Processazione delle richieste usando processi figli e thread (che riducono i tempi).

In UNIX abbiamo tre MPM: prefork, worker ed event.

15.3.2 MPM prefork

Il modulo prefork implementa un server multi-processo senza thread, si basa solo sui processi figli.

- All'avvio abbiamo un processo padre che lancia un certo numero di processi figli (cosiddetto preforking). Questi processi che saranno già disponibili per fare le richieste, ergo non sarà necessario lanciare una fork all'arrivo di una richiesta.
- Dopo aver servito una connessione il figlio torna disponibile per accettare una nuova connessione. Ogni figlio viene riciclato per un numero massimo di connessioni: superato tale numero il processo figlio viene terminato.



- Possibile un'espansione runtime del "pool di figli", qualora sia necessario. L'aspetto fondamentale è mantenere sempre alcuni processi figli disponibili.

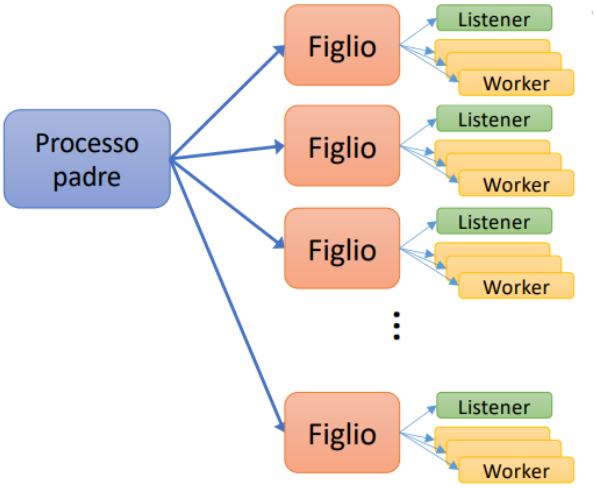
Consideriamo le seguenti variabili:

- Il numero di figli generati all'avvio è `StartServers`.
- Il numero massimo di figli è indicato con la variabile `MaxRequestWorkers`.
- È possibile stabilire un numero minimo di figli inattivi e un numero massimo di figli inattivi per mezzo delle variabili `MinSpareServers` e `MaxSpareServers`.
- Il numero massimo di connessioni gestite per processo figlio è indicato con la variabile `MaxConnectionsPerChild`.

Vantaggi	Svantaggi
Compatibilità. Alcuni moduli Apache o librerie potrebbero non supportare il multithreading **Stabilità**. Un processo che crasha interrompe una sola connessione (dato che ogni processo figlio si occupa di una connessione).	**Occupazione di memoria**. La memoria occupata di un processo è maggiore rispetto a quella di un thread (se si chiamano processi leggeri c'è un motivo). **Complessità del tuning**. Troppi processi inattivi occupano inutilmente memoria, troppo pochi causano più overhead da fork in caso di picchi di richieste.

15.3.3 MPM worker

Il modulo worker implementa multi-processo e multi-thread. Soluzione ibrida che prende il buono di tutti gli approcci possibili.



- Il processo padre genera un certo numero di processi figli (anche qua il preforking)
- Ogni processo figlio genera:
  - un thread listener che accetta/smista le connessioni.
  - un pool di thread worker che servono le richieste.
- Da una parte riduciamo l'overhead grazie al preforking (come prima), ma grazie ai thread si ha anche un risparmio di memoria!

Consideriamo le seguenti variabili:

- Il numero di processi figli generati all'avvio è `StartServers`.
- Il numero massimo di thread totali è indicato con la variabile `MaxRequestWorkers`.
- È possibile stabilire un numero minimo di thread inattivi e un numero massimo di thread inattivi per mezzo delle variabili `MinSpareThreads` e `MaxSpareThreads`.
- Il numero di thread worker per processo figlio è `ThreadsPerChild`.
- Il numero massimo di connessioni gestite per processo figlio è indicato con la variabile `MaxConnectionsPerChild`.
- Il numero massimo di figli è necessariamente `MaxRequestWorkers/ThreadsPerChild`

15.3.4 MPM event

Il modulo event è il modulo di default in Apache. Si risolve un difetto dei MPM precedenti: il non gestire connessioni temporaneamente inattive. Gestire queste connessioni in modo adeguato permette di liberare ulteriori risorse all'interno della macchina.

- **Primo esempio**.
  Un worker è connesso a un client che tarda a inviare una richiesta. Il thread, invece di aspettare, restituisce il controllo del socket al listener e passa a servire un altro client. Quando il primo client invierà una richiesta il listener lo assegnerà ad un altro thread.
- **Secondo esempio**.
  Un worker sta servendo un client con una connessione lenta e il buffer di invio del socket si riempie. Invece di attendere, restituisce il controllo del socket al listener che lo assegnerà ad un altro worker non appena sarà di nuovo scrivibile (quando il buffer si "spiena" – cit.)

Aumenta il numero di connessioni servibili in contemporanea a parità di numero di thread, eliminando i "tempi morti".

15.3.5 Valori globali e valori default dei parametri

Concludiamo segnalando ulteriori parametri globali (cioè relativi a tutti gli MPM)

- `ThreadLimit`: limite massimo configurabile per il numero di thread attivi per processo.
- `ServerLimit`: limite massimo configurabile per il numero di processi attivi.

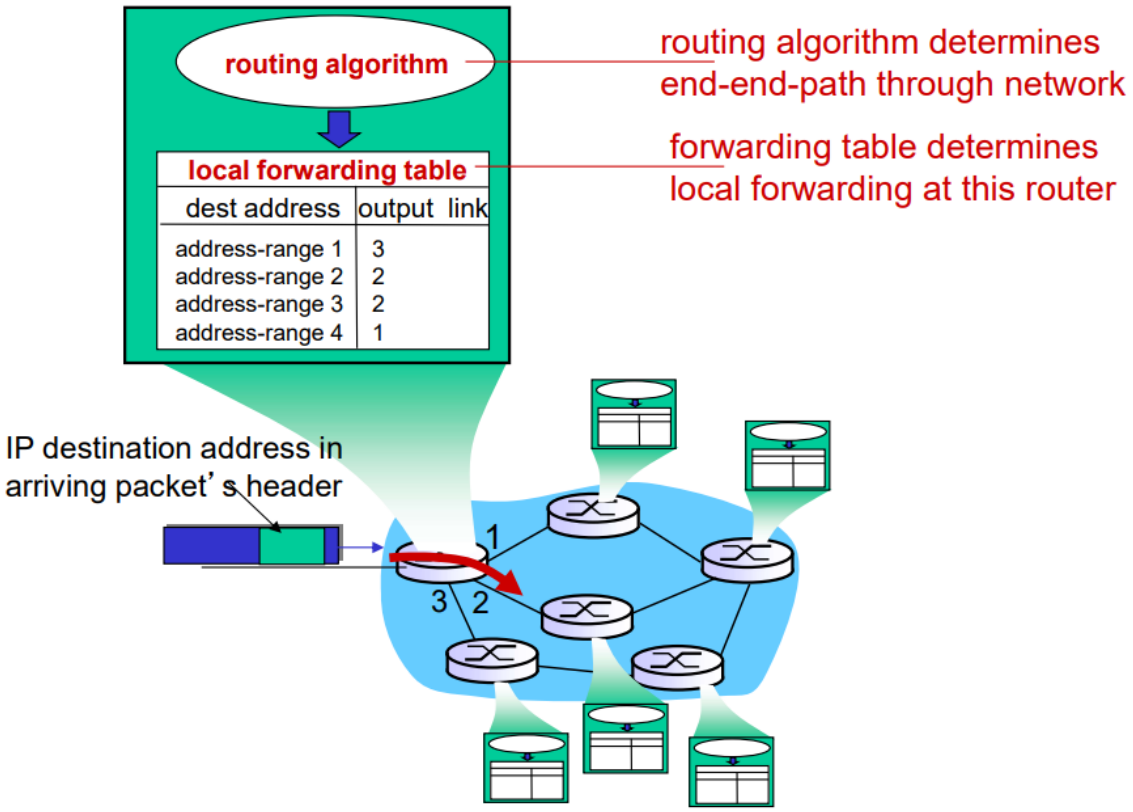
Valgono le seguenti condizioni:

- **Prefork**: $MaxRequestWorkers \leq \text{ServerLimit}$
- **Worker ed Event**:
  - $ThreadsPerChild \leq ThreadLimit$
  - $\text{MaxRequestWorkers} \leq \text{ServerLimit} * \text{ThreadsPerChild}$

16 LABORATORIO: ALGORITMI DI INSTRADAMENTO

16.1 Cosa sappiamo di già e obiettivi del capitolo

Abbiamo già visto in capitoli precedenti alcuni aspetti relativi all'instradamento dei pacchetti. Abbiamo visto, in particolare, l'esistenza di una tabella di forwarding (*local forwarding table*) presente in ogni router: dato l'indirizzo del destinatario del pacchetto verifichiamo se l'indirizzo fa match con una particolare riga della tabella, individuando così l'uscita verso cui redirigere il pacchetto.

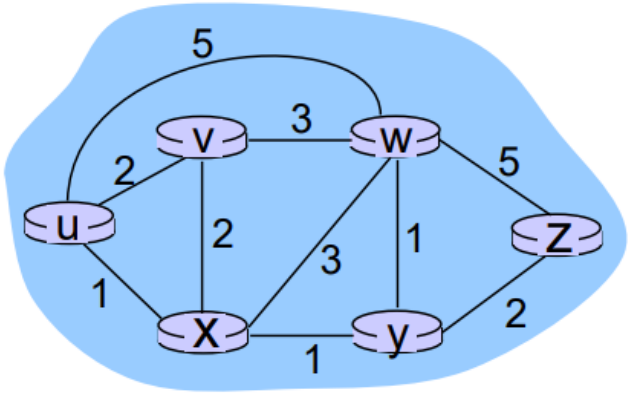


In questo capitolo vogliamo approfondire due aspetti:

- quali sono gli algoritmi adottati per l'individuazione dello *shortest path*, cioè il cammino "migliore" per passare da un nodo a un altro nodo (si veda poco più avanti la definizione);
- come si coordinano i router della rete nell'impostazione delle tabelle di forwarding (i router conoscono la topologia della rete, oppure ne conoscono solo una parte?).

16.2 Astrazione del grafo

Riprendiamo il concetto di grafo.



Sappiamo che un grafo è definito da una coppia di insiemi: l'insieme dei nodi e l'insieme degli archi. Si prenda l'esempio in figura (dove gli archi sono bidirezionali, dato che non ci sono frecce):

$$G = \{N, E\}$$
$$N = \{u, v, w, x, y, z\}$$
$$E = \{(u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z)\}$$

Negli algoritmi di instradamento parleremo di ottimizzazione del costo. Cosa si intende con costo?

- Nella figura abbiamo associato ad ogni arco un valore numerico, che consiste nel costo necessario per attraversare quel particolare pacchetto.
- Se un particolare arco non appartiene al grafo porremo come costo $+\infty$
- Il costo totale consiste nella somma dei costi degli archi percorsi.
- Individuare lo *shortest path* significa individuare un percorso tale da minimizzare il costo totale, il *cammino di costo minimo*!

Ricordarsi che i criteri adottati nell'ottimizzazione sono diversi, e possono portare a un'interpretazione diversa del costo:

- **Minimizzazione del numero di hop**
  Il nostro interesse è minimizzare il numero di router attraversati. In questo caso i costi degli archi sono tutti uguali.
- **Minimizzazione della distanza geografica**
  Vogliamo che il nostro pacchetto attraversi i router in modo tale da minimizzare il percorso geografico (se devo trasmettere un pacchetto dall'Italia alla Norvegia non ha fare un giro lungo passando dalla Russia). In questo caso il costo di un arco che collega due router rappresenta la distanza geografica tra i due router.
- **Riduzione della congestione**
  Vogliamo fare in modo che il pacchetto eviti di passare da collegamenti che presentano un elevato utilizzo di banda. In questo caso avremo una proporzionalità diretta tra congestione del collegamento e costo (un arco che rappresenta un collegamento fortemente congestionato deve presentare un costo maggiore rispetto ad archi che rappresentano collegamenti non congestionati).

Ricordarsi che non abbiamo l'ottimizzazione di un unico obiettivo: gli algoritmi adottati sono complessi, e i criteri da ottimizzati pesati in base al contesto (in un contesto di rete fortemente congestionata è chiaro quale criterio prevale).

16.3 Classificazione degli algoritmi di instradamento

Classifichiamo gli algoritmi di instradamento

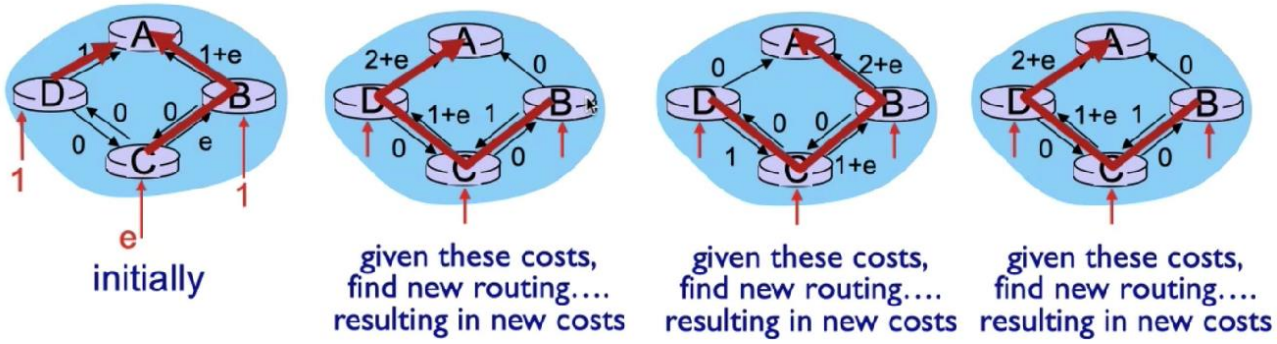
Global or decentralized?	***Static or dynamic?***
Algoritmi global (o algoritmo centralizzato) L'algoritmo calcola il percorso a costo minimo tra una sorgente e una destinazione avendo conoscenza globale e completa della rete. Si ha un'unica esecuzione in un controller centralizzato oppure un'esecuzione replicata in ogni nodo della rete. Sono detti ***algoritmi link-state*** dato che richiedono conoscenza dello stato di ciascun collegamento, e quindi del relativo costo.	**Algoritmi static** I percorsi cambiano lentamente nel tempo, spesso come risultato di un intervento umano (modifica manuale delle tabelle di forwarding).
Algoritmi decentralized Lo *shortest path* viene calcolato in modo distribuito e iterativo. Ogni nodo fa la sua parte, ma conosce solo parte della topologia della rete (quella dei router "vicini"). I router vicini si scambiano informazioni (in primis sul costo dei link). Sono detti ***algoritmi distance-vector*** dato che si basano su un vettore contenente la stima dei costi (le distanze) verso tutti gli altri nodi della rete.	**Algoritmi dynamic** I percorsi cambiano con estrema frequenza, al variare della congestione o delle caratteristiche della topologia della rete.

16.4 Algoritmi link-state

Abbiamo capito che per l'esecuzione di algoritmi link-state si pone in input l'intera topologia della rete. Si esegue l'algoritmo e per mezzo di comunicazione di tipo broadcast si fa in modo che tutti i nodi della rete presentino le stesse informazioni.

L'esempio per eccellenza è l'algoritmo di Dijkstra, che abbiamo già conosciuto a Ricerca Operativa.

- Ci piace perché ha complessità quadratica $O(n^2)$, dato che abbiamo $\frac{n(n+1)}{2}$ confronti.
- Possibile avere implementazioni più efficienti con complessità semi-logaritmica $O(n \log n)$
- **Drawback: il problema delle oscillazioni**.
  In presenza di elevata dinamicità Dijkstra non è l'algoritmo più adatto, in particolare quando il costo degli archi dipende dal livello di congestione. Esistono implementazioni di Dijkstra che alleviano il problema, ma non è possibile eliminarlo del tutto. La figura seguente evidenzia il cosiddetto problema delle oscillazioni: costi che portano a nuove esecuzioni di Dijkstra i cui cambiamenti portano a nuove esecuzioni di Dijkstra…



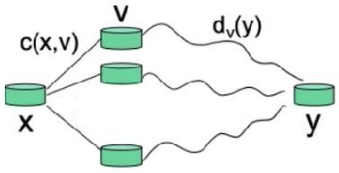
16.5 Algoritmi distance-vector

16.5.1 Premesse ed equazione di Bellman-Ford

Abbiamo capito che l'esecuzione di algoritmi distance-vector si basa su una conoscenza locale, cioè solo sulle informazioni dei router "vicini" (in particolare quelli a cui il router è direttamente connesso).
L'approccio adottato è quello della programmazione dinamica: divisione del problema in sotto-problemi e individuazione di una sequenza di questi sotto-problemi tale da condurci all'ottimo.

Introduciamo come premessa l'***equazione di Bellman-Ford.*** Sia $d_x(y)$ il costo dello shortest-path da un nodo x a un nodo y: allora diremo che

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



min cost from v to destination y

cost to neighbor v

min taken over all neighbors v of x

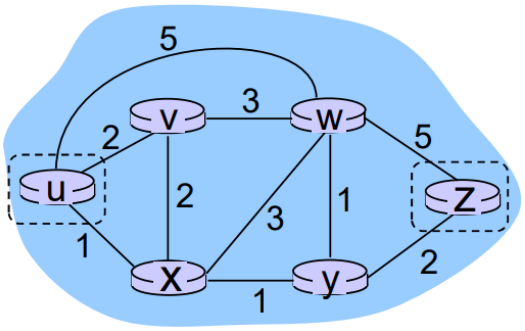
In sostanza affermiamo che il costo dello *shortest-path* da un nodo x a un nodo y lo si trova individuando il nodo v vicino tale per cui la somma posta nella formula risulti minimizzata.

Osservazione.

- Negli algoritmi link-state si ottiene lo shortest path.
- Negli algoritmi distance-vector non si ottiene lo shortest path, ma soltanto il next hop!

Ogni nodo esegue l'algoritmo per conto suo e non ha conoscenza dell'intera topologia (quindi conosce solo la situazione dei suoi "vicini").

Prendiamo il seguente esempio: supponiamo che il nostro pacchetto si trovi nel nodo u e che debba essere trasmesso al nodo z. Deve essere individuato il *next hop* che mi garantisce un percorso a costo minimo.



Il nodo u conosce

- il costo dei collegamenti diretti (archi $c(u,v) = 2, c(u,x) = 1, c(u,w) = 5$)
- il costo minimo degli shortest path dai nodi v, x, w (su cui diremo a breve)

$$d_v(z) = 5 \qquad d_x(z) = 3 \qquad d_w(z) = 3$$

Dall'equazione di Bellman-Ford otteniamo che il costo dello shortest path dal nodo u è uguale a 4

$$d_u(z) = min\{c(u,v) + d_v(z); c(u,x) + d_x(z); c(u,w) + d_w(z)\}$$
$$d_u(z) = min\{2 + 5; 1 + 3; 5 + 3\} = 4$$

shortest path che possiamo percorrere passando dal nodo x

$$n^* = \arg \min_v \{c(x,v) + d_v(y)\} = x$$

16.5.2 Idea alla base degli algoritmi distance-vector

Siamo nel nodo x, sia $D_x(y)$ una stima del costo dello shortest path dal nodo x al nodo y. Sia D_x un vettore avente per componenti le stime dei costi minimi per andare da x verso tutte le altre destinazioni.

$$D_x = [D_x(y) : y \in N]$$

dove N, ricordiamo, è l'insieme dei nodi introdotto all'inizio del capitolo (quando abbiamo riesumato l'astrazione del grafo). Ogni nodo x:

- conosce i costi verso i nodi v vicini ($c(x,v)$)
- conosce i vettori D_v relativi ai vicini v

Affinchè ogni nodo x conosca i vettori D_v è necessario che ogni nodo v trasmetta di volta in volta il suo vettore delle distanze D_v ai vicini!

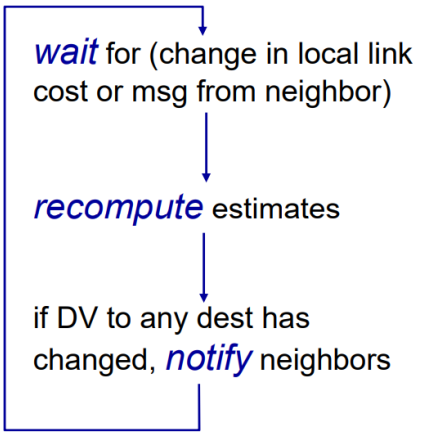
Ogni volta che il nodo x riceve un vettore D_v dal nodo v procede ad aggiornare il suo vettore D_x utilizzando l'equazione di Bellman-Ford, sfruttando il contenuto del vettore ricevuto

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

L'algoritmo converge all'effettivo costo minimo

$$\boxed{D_x(y) \longrightarrow d_x(y)}$$

- È iterativo e asincrono. Iterazioni locali causate da variazioni dei costi verso i nodi vicini, o dalla ricezione di vettori delle distanze D_v
- È distribuito, perché ogni nodo notifica le variazioni del proprio vettore delle distanze D_v solo ai nodi vicini. I vicini notificheranno a loro volta i loro vicini se necessario.



16.6 Algoritmi link-state e distance-vector a confronto

Due osservazioni

- Negli algoritmi link-state (LS) si hanno $n \cdot E$ messaggi spediti, dove n è il numero dei nodi ed E il numero degli archi (numero dei link)
- Negli algoritmi distance-vector (DV) si hanno tempi di convergenza variabili

16.7 Routing gerarchico

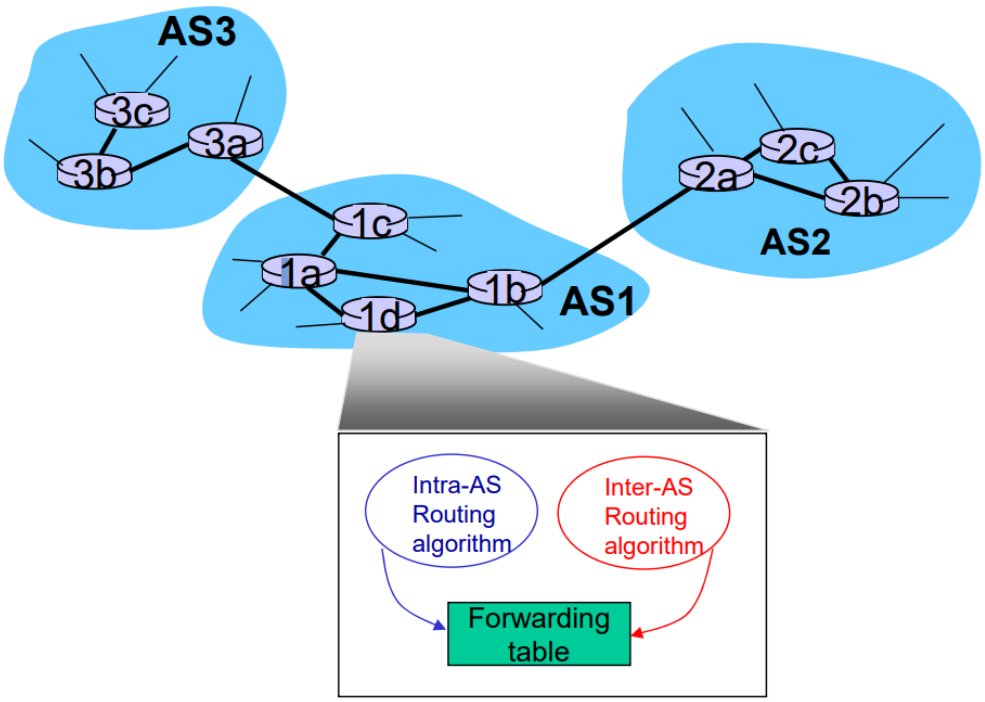
16.7.1 Autonomous systems

Fino ad ora abbiamo ragionato considerando reti con un numero ridicolo di router. Se la topologia di rete si avvicina a quella reale è necessario introdurre granularità per i nostri algoritmi di instradamento. Sappiamo che la rete Internet consiste in una rete di reti, dove ogni rete è caratterizzata da tipologie di router diversi oltre che politiche diverse: non posso dire con oltre 600 milioni di visualizzazione che l'intera Internet è basata su Dijkstra o su Bellman-Ford.

- Rete di reti: si raggruppano insiemi di router, che vanno a costituire *autonomous systems*.
- In ciascun *autonomous system* l'amministratore ha libertà di scelta si protocolli di instradamento.
- A confine di ogni autonomous system si trovano *gateway routers*, che permettono il collegamento di un autonomous system agli altri autonomous systems.

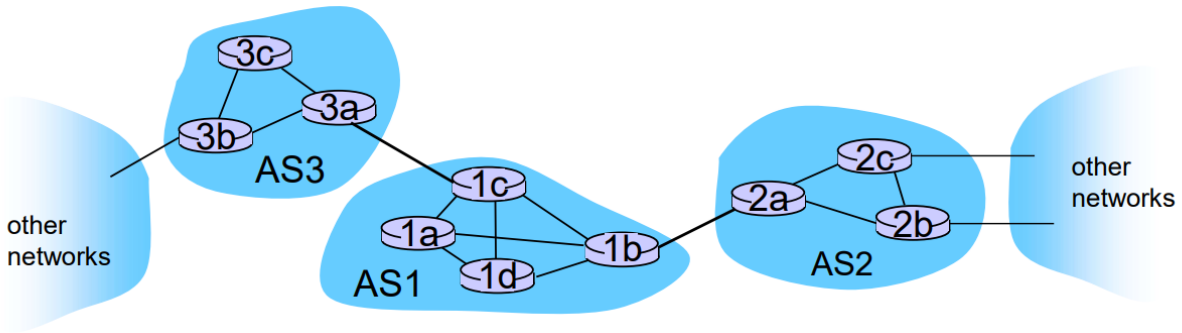
16.7.2 Classificazione degli algoritmi: intra-AS e inter-AS

Prendiamo la seguente figura



- Abbiamo tre autonomous system: AS1, AS2 e AS3.
- Consideriamo un particolare router: 1d. La tabella di forwarding è gestita per mezzo di due tipologie di algoritmi:
  - *algoritmi Intra-AS* (individuare il next hop per destinazioni interne ad AS1 – potrei decidere di adottare Dijkstra, ad esempio)
  - *algoritmi Inter-AS* (individuare il next hop quando il pacchetto deve uscire dall'autonomous system – next hop per uscire col minor costo da AS1, per raggiungere quindi il default gateway "più vicino")
- **Algoritmi Inter-AS**.
  Supponiamo che un pacchetto debba passare da un nodo in AS1 a un nodo di altri AS. AS1 deve

conoscere quali destinazioni possono essere raggiunge per mezzo di AS2 e quali per mezzo di AS3: tutto ciò deve essere propagato ad ogni nodo presente in AS1.



- **Algoritmi Intra-AS**.
  Sono detti anche Interior Gateway Protocols (IGP). Due tipologie comuni di algoritmi:
  - **RIP**: Routing Information Protocol
  - **OSPF**: Open Shortest Path First

16.7.3 Protocolli Intra-AS

Individuazione di percorsi all'interno di un autonomous system

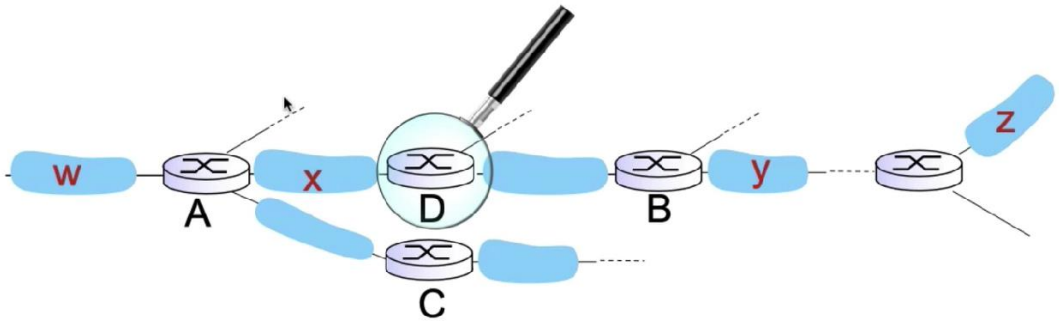
16.7.3.1 Routing Information Protocol (RIP)

Il RIP è un algoritmo *distance-vector*.

- La metrica di distanza è il numero di hop, quindi ogni arco ha costo 1.
- Il massimo numero di hop all'interno della rete è 15.
- Scambio di informazioni (messaggi che chiamiamo ***advertisements***) sui distance-vector ogni 30 secondi.
- Ogni *advertisement* è una lista di al più 25 possibili destinazioni.

- **Che senso ha introdurre l'advertisement se gli archi hanno tutti costo 1?**
  Cambiano i collegamenti: nuovi collegamenti, eliminazione di collegamenti già esistenti…

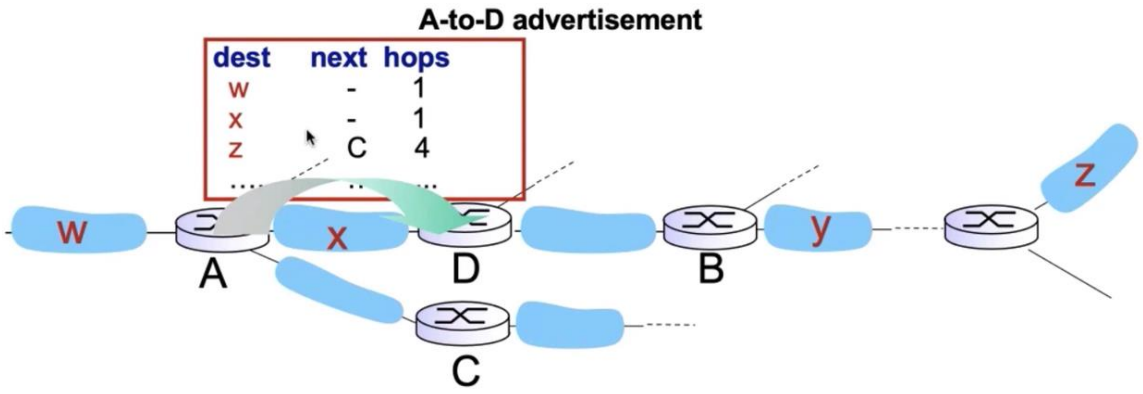
Facciamo un esempio: consideriamo la tabella di forwarding che presenta il router D.



Destination subnet	*Next router*	*# hops to dest*
w	A	2
y	B	2
z	B	7
x	--	1
…	…	…

Si noti che in questa tabella si afferma che la destinazione z può essere raggiunta con 7 hops.

A un certo punto il router D riceve un advertisement dal router A: questo segnala che è possibile raggiungere la destinazione z con quattro hop (a partire dal router A, quindi 5 hop se si parte dal router D).



Destination subnet	*Next router*	*# hops to dest*
w	A	2
y	B	2
z	**B -> A**	**7 -> 5**
x	--	1
…	…	…

L'advertisement ha segnalato al router D un percorso migliore: per tale motivo aggiorna la tabella di forwarding (per raggiungere z il next router non è più B, ma A, inoltre il numero di hop per arrivare a destinazione si riduce da 7 a 5).

16.7.3.2 Open Shortest Path First (OSPF)

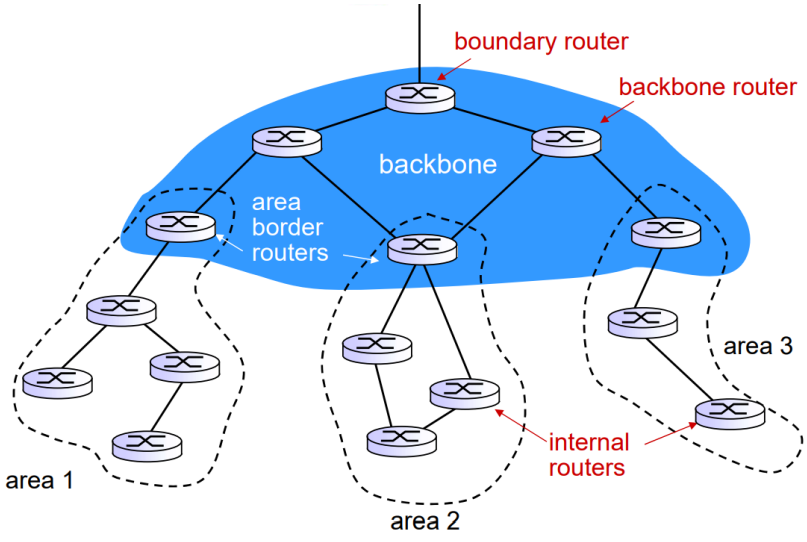
L'OSPF è un algoritmo di tipo link-state, ergo ha bisogno dell'intera topologia di rete e non si basa sullo scambio di informazioni tra vicini.

- Disseminazione di pacchetti LS (flooding) per far conoscere la topologia di rete a tutti i router di un autonomous system.
- La computazione è effettuata utilizzando l'algoritmo di Dijkstra.
- Gli advertisement presentano un'entrata per vicino.
- Si introduce un OSPF gerarchico per gestire autonomous system di dimensioni discrete.

I router appartenenti a un AS sono classificabili in

- router di frontiera (*boundary router*)
- router dorsali (*backbone router*)
- router appartenenti ad aree OSPF.

Complessivamente si ottiene una gerarchia a due livelli, come quella in figura:



- Grazie alla suddivisione in aree il flooding è più leggero: gli advertisement di un'area contengono la topologia solo di quell'area e sono disseminati solo tra i router di quell'area.
- I backbone router eseguono OSPF limitato alla backbone, i router di frontiera d'area hanno database con la topologia della rete.
- Se un router interno di un'area vuole trasmettere un pacchetto a un router di un'altra area allora dovrà rivolgersi ai router di frontiera della propria area, che si rivolgeranno a loro volta ai backbone router.

E se la destinazione non è parte dell'autonomous system? Entrano in scena gli algoritmi inter-AS.

16.7.4 Protocolli inter-AS

Individuazione di percorsi che attraversano più autonomous systems

16.7.4.1 Border Gateway Protocol (BGP)

16.7.4.1.1 Caratteristiche base: duplice anima e struttura dell'advertisement

Il BGP è il principale protocollo inter-AS.

- Duplice anima (cit.)
  - ***eBGP (External BGP)***: recupera dagli autonomous system vicini le informazioni di raggiungibilità, capire quali autonomous system vicini portano quali prefissi di rete.
  - ***iBGP (Internal BGP)***: propagazione delle informazioni di raggiungibilità a tutti i router di un autonomous system.
- Permette a un autonomous system di segnalare la propria esistenza al resto della rete.
- Il protocollo determina le rotte "migliori" sulla base delle informazioni di raggiungibilità **e delle politiche adottate**.
  - In che senso politiche adottate? Non si era parlato fino ad ora di shortest path?
  - **Esempio**: un autonomous system aziendale potrebbe non essere disposto a trasportare pacchetti generati da un AS estraneo e diretti a un altro AS estraneo (anche se si trova sullo shortest path). Potrebbe essere disposto a farlo per autonomous system specifici che hanno pagato per il servizio (*servizio di transito*).
  - **Esempio 2**: non si vuole traffico commerciale su reti di ricerca.
  - **Esempio 3**: il traffico da/verso Apple non deve passare attraverso Google.

  Ciascun autonomous system può definire politiche di routing liberamente, e ciò va combinato con l'esigenza di individuare lo shortest path.

Anche questo protocollo si basa su ***advertisements***, cioè messaggi dove si pubblicizzano una o più rotte (si propongono *path*): per questo motivo si parla di *path-vector*. Gli advertisement presentano in particolare le seguenti informazioni (`Prefisso + Attributi = Percorso`):

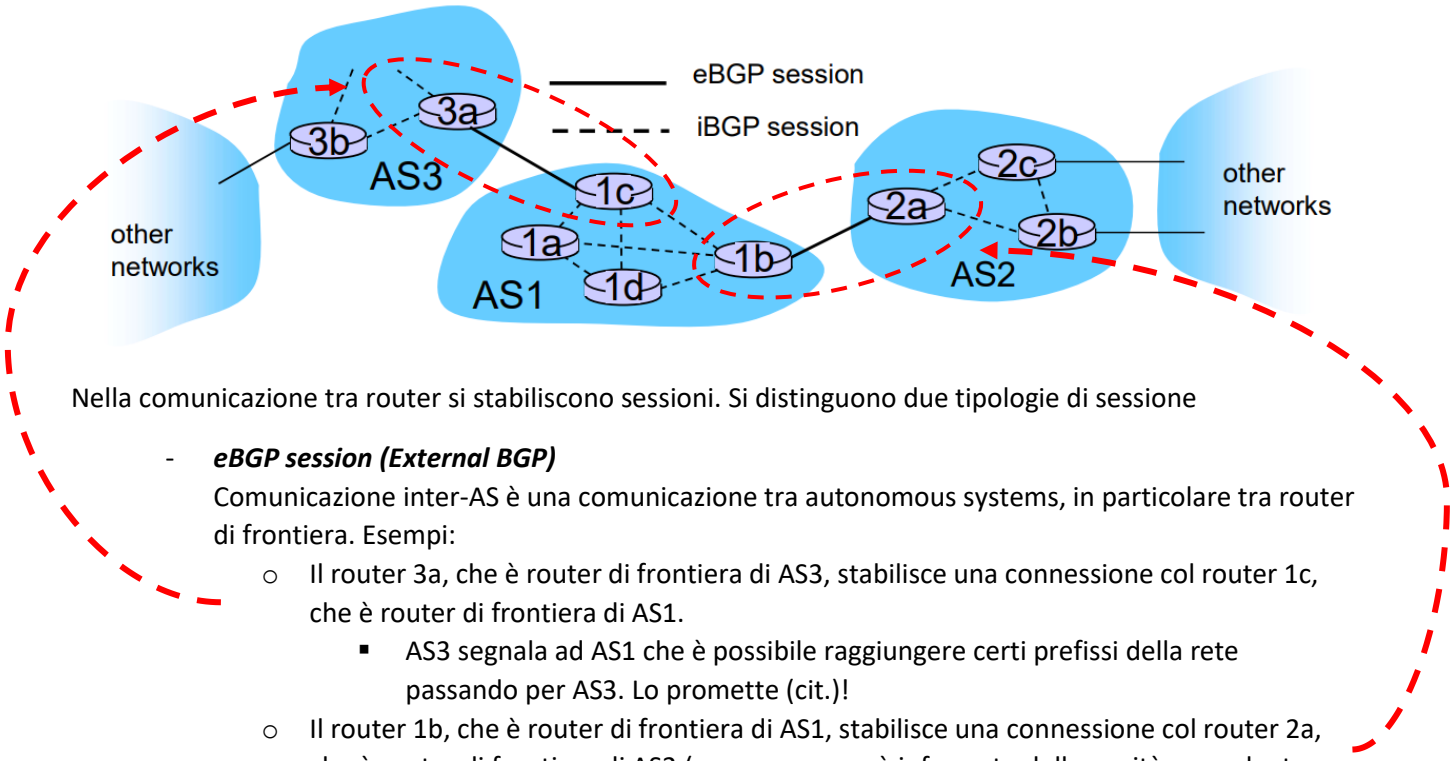
- **Prefisso di rete**, che rappresenta la "sezione" di rete[17] raggiungibile;
- **Attributi**
  - ***AS-PATH***, contiene tutti gli autonomous systems tramite i quali l'advertisement, che pubblicizza un particolare prefisso di rete, è passato (il percorso all'indietro è il *path*);
  - ***NEXT-HOP***, indica il router interno all'autonomous system al quale bisogna inoltrare il pacchetto affinchè segua il path indicato in AS-PATH.

Per avere meglio un'idea si consideri il seguente esempio di advertisement, dove si propone una rotta:

ADVERTISEMENT	Traduzione in parole povere
Prefix: 138.16.62/22	*"Se vuoi raggiungere una rete con prefisso di rete 138.16.64/22 io ti ci porto attraverso il path AS3 AS131. Per farlo inoltra il messaggio al nodo 201.44.13.125".*
AS-PATH: AS3 AS131	
NEXT-HOP: 201.44.13.125	

[17] Salvo eccezione difficilmente si pone in tabella di forwarding una riga relativa a uno e un solo network. Con una riga è possibile gestire più istanze verso una particolare area della rete.

16.7.4.1.2 BGP sessions esterne e interne



Nella comunicazione tra router si stabiliscono sessioni. Si distinguono due tipologie di sessione

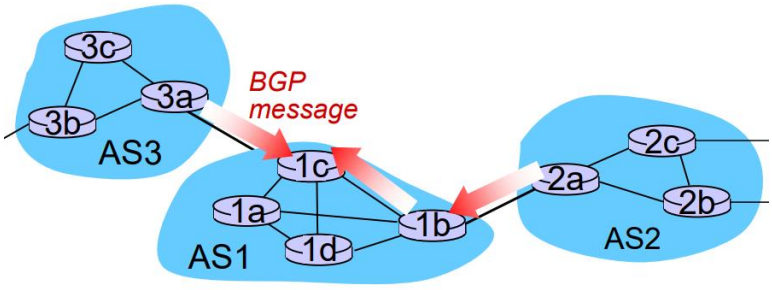
- ***eBGP session (External BGP)***
  Comunicazione inter-AS è una comunicazione tra autonomous systems, in particolare tra router di frontiera. Esempi:
  - Il router 3a, che è router di frontiera di AS3, stabilisce una connessione col router 1c, che è router di frontiera di AS1.
    - AS3 segnala ad AS1 che è possibile raggiungere certi prefissi della rete passando per AS3. Lo promette (cit.)!
  - Il router 1b, che è router di frontiera di AS1, stabilisce una connessione col router 2a, che è router di frontiera di AS2 (non appena sarà informato della novità precedente trasmessa da 3a al router 1c).
    - Viene segnalato ad AS2 che è possibile raggiungere i prefissi di rete segnalati precedentemente dal router 3a per mezzo di AS1.
- ***iBGP session (Internal BGP)***
  Riferisce le informazioni ottenute nella eBGP session ai router del suo AS, in particolare il router di frontiera deve segnalare ai router della rete che è "l'uscita privilegiata dalla rete" per raggiungere certi prefissi di rete.

16.7.4.1.3 Selezione delle rotte

Il gateway router che riceve advertisement tiene a mente le politiche adottate, e rifiuta l'advertisement nel caso in cui la proposta contrasti con le sue politiche (Esempio banale: legge *AS-PATH* e si accorge che il path passa da un particolare autonomous system).

I router possono ricevere più rotte verso un particolare prefisso. Nel confronto si dovrà scegliere quale rotta sia quella migliore per l'AS: si tiene conto…

- delle politiche adottate,
- di quale sia lo shortest AS-PATH,
- quale sia il next-hop più vicino…



Si consideri il seguente esempio e si scelga la rotta in base allo shortest AS-PATH

ADVERTISEMENT PROPOSTI	
AS2 AS17	to 138.16.64/22
AS3 AS131 AS201	to 138.16.64/22

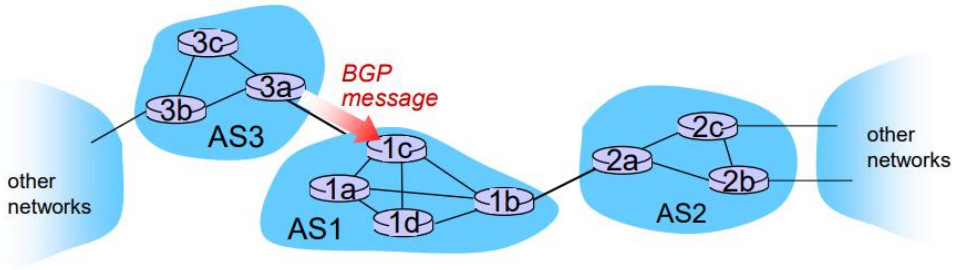
Il router sceglierà la prima rotta, in quanto il numero di autonomous systems da percorrere è minore!

16.7.4.1.4 Aggiornamento della tabella di forwarding

Cosa fanno i router quando ricevono l'advertisement? Ragioniamo da una ***High-level overview*** e mettiamoci nei panni del router 1c, appartenente ad AS1

1) ***Router becomes aware of prefix***

   Il router diventa consapevole del prefisso dopo aver ricevuto la coppia $<$ Prefisso,Attributi $>$

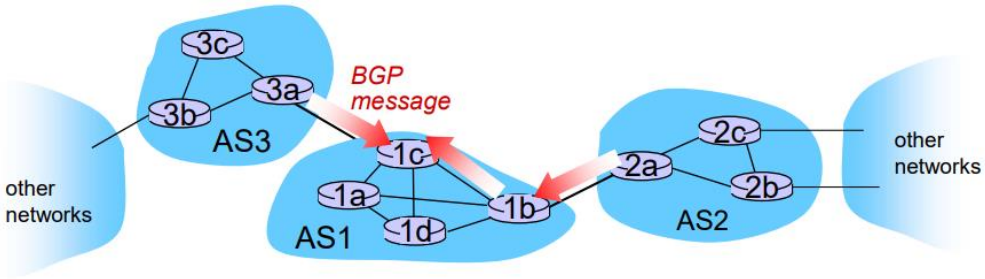


   Riesumiamo l'esempio di advertisement visto qualche pagina indietro

ADVERTISEMENT	Traduzione in parole povere
Prefix: 138.16.62/22	*"Se vuoi raggiungere una rete con prefisso di rete 138.16.64/22 io ti ci porto attraverso il path AS3 AS131. Per farlo inoltra il messaggio al nodo 201.44.13.125".*
AS-PATH: AS3 AS131	
NEXT-HOP: 201.44.13.125	

2) ***Router determines output port for prefix***

   Il router determina la porta di uscita per quel prefisso (dopo aver scelto il percorso migliore, sulla base di quanto detto prima)



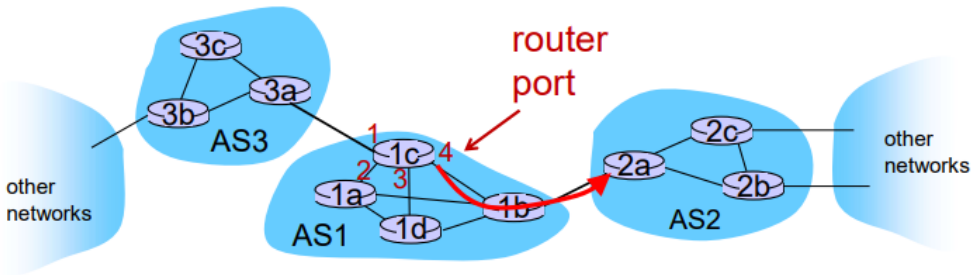
   Anche qua riesumiamo l'esempio di shortest AS-PATH visto prima

ADVERTISEMENT PROPOSTI	
AS2 AS17	**to 138.16.64/22**
AS3 AS131 AS201	to 138.16.64/22

   Scelto lo shortest AS-PATH prendiamo l'attributo NEXT-HOP. Abbiamo già detto che NEXT-HOP consiste nell'indirizzo IP del nodo a cui inviare il pacchetto da trasmettere in un particolare prefisso di rete: è l'inizio dell'AS-PATH!

```
AS-PATH: AS2 AS17
NEXT-HOP: 111.99.86.55
```

   Si applica l'algoritmo OSP per trovare il percorso più breve da 1c al NEXT-HOP 111.99.86.55: segue che la output port è la numero 4, come in figura:



   Abbiamo individuato il criterio che adottiamo per scegliere il percorso nel caso in cui vi sia ex aequo tra due AS-PATH: la scelta dell'AS-PATH con il router di gateway (frontiera dell'AS) più vicino al nodo di partenza.

3) ***Router enters prefix-port in forwarding table***
   Il router, individuata l'output port, aggiorna la tabella di forwarding

16.7.5 Riepilogo

Riepiloghiamo schematicamente su quanto detto fino ad ora.

- **Router becomes aware of prefix**
  Il router diventa consapevole di un prefisso grazie al meccanismo degli advertisement.
  - Con l'algoritmo BGP gli advertisement arrivano da altr
- **Determine router output port for prefix**
  Il router determina la porta di uscita.
  - Usa l'algoritmo BGP per trovare la migliore rotta inter-AS
  - Successivamente usa l'algoritmo OSPF per individuare la migliore rotta intra-AS.
  - La somma dei due algoritmi permette di individuare l'output port e quindi l'inizio della migliore rotta inter-AS. Si tenga a mente che il messaggio dovrà percorrere:
    - un pezzo di AS a cui appartiene il router di partenza (bisogna raggiungere il gateway, si utilizza OSPF per trovare il percorso migliore all'interno dell'AS);
    - tutti gli altri AS (si utilizza BGP per determinare gli AS da percorrere).
- **Router enters prefix-port in forwarding table**
  Il router, individuata l'output port, aggiorna la tabella di forwarding

<u>Perché abbiamo bisogno di protocolli Intra e Inter?</u>

- Perché esistono politiche diverse, potremo dire "esigenze diverse", in una rete e in un'altra
- Lo hierarchical routing permette di ottenere tabelle di dimensione minore così come permette di minimizzare il traffico.

Per quanto riguarda la performance si osservi che gli algoritmi intra-AS pongono il focus sulla performance, mentre gli algoritmi inter-AS pensano alle prestazioni, ma la scelta del path potrebbe essere dominata dalle politiche adottate.

17 RIFERIMENTI AL CORSINI

Sezione speciale con le reference al prof. Paolo Corsini, fondatore del corso di laurea in Ingegneria Informatica e ispiratore del motto *Ingegneria deve essere difficile*.

Meme del canale "Longo & co. Shitposting"

1) **Sulla presentazione di progetti agli esami**.
   Come direbbe il Corsini, il progetto serve se non ce l'hai e non serve se ce l'hai.
2) **18 a tutti**
   Se bocci uno studente ti tocca rifargli l'esame. Lui non bocciava nessuno e dava a tutti 18, solo che lui lo dava sistematicamente a tutti.
3) **Dimostrazioni efficaci**
   Quando ero studente il prof. Corsini spiegò questo protocollo. Uno studente alzò la mano: secondo me così non può funzionare. Corsini chiese chi avesse fatto la domanda, e rispose: "È una vita che esiste e funziona".
4) **Corsini e le forbici**
   Abbiamo mantenuto la rete basata su cavo sottile fino a inizio anni 2000. Quando abbiamo inaugurato la nuova sede di dipartimento molti colleghi si sono spostati nella sede nuova. Si era deciso di passare dalla versione su cavo sottile a quella basata su switch. Si programmava sempre e poi non si faceva. Corsini si è armato di forbici e appena scopriva un cavo sottile ZAAACK, lo tagliava: questo significava impedire alle persone di lavorare. Tagliando tagliando arrivò a tagliare anche il cavo del suo telefono. Lo abbiamo preso in giro per anni su questa cosa ("Corsini c'hai le forbici in tasca?")
5) **Corsini diceva: le cose sono o non sono!**
   Fare la tesi con Paolo Corsini era molto divertente perché si doveva stare attenti a cosa si diceva. Una mattina chiese a un tesista: *funziona lo switch?* Il collega rispose: *insomma*. Segue partaccia: *le cose funzionano o non funzionano*!
6) **La nascita di "Ingegneria deve essere difficile"**
   Il secondo anno diventa difficile, con Reti logiche al primo semestre e Calcolatori elettronici al secondo. D'altronde avete avuto la strada facile, dovete abituarvi! Quando ero studente ci fu una delle solite riunioni [...] che gli studenti fanno per aumentare il numero di esami, che è già record mondiale adesso (ndr. discorso del 2005-2006). Quando ero studente volevano l'appello mensile, o qualcosa del genere: i docenti erano ovviamente contrari, ci sono tre appelli e bastano [...]. Gli studenti dicevano no: per alcuni esami la probabilità di accesso è molto bassa, e quindi aumentando il numero di prove aumentano le possibilità di accesso. Si alzò un professore che voi conoscete bene, ma di cui non vi dico il nome (ndr. Paolo Corsini), che dice: "è inutile stare a discutere e a perder tempo, INGEGNERIA NON PUO' NON ESSERE DIFFICILE". Dopo qualche giorno alcuni ragazzi si fecero chiudere dentro l'edificio del triennio (ndr. il polo A) e murarono la porta d'ingresso del nostro dipartimento, usando il [...] di quelli di scienze delle costruzioni, che solitamente usano per fare le prove di stress. Impastarono la calce [...], portarono su i provini e con questi fecero un muro chiudendo la porta del dipartimento: sopra la porta del dipartimento scrissero "INGEGNERIA DEVE ESSERE DIFFICILE". Da allora questa frase `e il motto di Ingegneria, anche se la frase originale era stata "Ingegneria non può non essere difficile". [...] Ingegneria deve essere difficile, non può non essere difficile altrimenti non sarebbe più Ingegneria! Dovete abituarvi a soffrire!!

Foto di Piero Bellini